Bozó Balázs - Prievara Zsolt

Az Amiga programozása C és Assembly nyelven

1996.

Aurum Könyvek sorozat #8
Sorozatszerkesztő:
Arany Sándor

Az Amiga programozása Assembly és C nyelven

Kiadja az AURUM DTP Stúdió Tiszafoldvár,
Felelős kiadó. Arany Sándor
Tervezés és nyomdai előkészítés: Sóti Gábor
Nyomda: CIFI Nyomda Szolnok

Köszönetnyilvánítás

Ezúton szeretnénk köszönetet mondaiii ***Arany Sándornak*** *a sok fáradó zásáért, s a lehetőségért, hogy elkészíthettük ezt a könyvet. Továbbá* ***Misku Istvánnak*** *áldozatos munkájáért amit ajavüás során végzett*

Rengeteg hálával tartozunk szúleinknek, hogy lehetőséget kaptunk a számítástechnika világának megismeréséhez, s hogy elnézik nekünk az ezzel járó életformát.

Szeretnénk köszönetet mondani az alábbi barátoknaJc az évek alatt nyítjtott sok segítségért:

Barna János, Frankó Tamás, Kopácsi Szabolcs, Pintér Attila, Varga Gábor

Utoljára, de nem utolsósorban a következő uraknak:

Jay Miner, *M. Sinz, P. Cherna, D.Creenwald, R. Jesup, S. Shanson, C. Green, B. Whitebook, A. Havemose, E. Cotton, M. Taillejer, D. Junod, P. Pawlik, K. Kuwata, B. Jackson, G. Miller, K. Dyke*

... az Amigáéit

A szerzők

TARTALOM

BEVEZETŐ

Manapság amikor a csapból is a számítástechnika folyik, lehet hogy hibának tűnik még egy könyvet kiadni ebben a témában, pláne, ha figyelembe vesszük azt, hogy sokan az Amigát már leírták mint számítógépet.

Ennek ellenére íródik a könyv, lévén én, és még sokan mások az Amiga megszállottjai. A könyv azoknak készült akiknek szintén van. vagy lesz Amigájuk és nem merül ki a gép használata a játékban, és a grafikai alkalmazásokban, hanem esetleg szabadidejükben szeretnének egy-két kisebb programot saját problémáik megoldására, vagy esetleg játékötleteik megvalósítására. Bár a kötet célja nem az, hogy professzionális játékot lehessen írni a segítségével, de kisebb játékokat meg lehet valósítani. Nem célunk az, hogy programozni tanítsunk, vagy a C nyelvet ismertessük, sem a gép assembly nyelvét, mert ezt mások már megtették több-kevesebb sikerrel.

A célunk az, hogy olyan magyar nyelvű könyvet adjunk a felhasználók kezébe, amely segítségével a programjainkat úgy írhatjuk meg, hogy kihasználjuk az Amiga csodálatos lehetőségeit és programjaink a megkívánt küllemet oltsék.

A könyv felépítéséről csak annyit, hogy az Amiga rendszere annyira kerek egész, és minden mindennel összefügg, hogy nehéz lenne kiválasztani honnan is kellene kezdeni ennek bemutatását. így inkább azt az utat választottuk, hogy sok gyakorlati példán keresztül, programozás közben mutatjuk be, és mindent ott magyarázunk. Főleg azért választottuk ezt a módszert, mert mi is így tanultuk.

Reméljük, hogy sok hasznát veszi a kedves olvasó a könyvnek.

A szerzők

Szentes '95

1. Kezdjük el

1.1 Az Amigáról...

1.1.1 Egy kis történelem

Valamikor a '80-as évek elején pár fiatalember úgy gondolta, hogy kellene esinálniuk egy sokkal jobb gépet, mint amit lehetett akkoriban kapni. Persze nemcsak jobbat, hanem olcsóbbat is akarlak csinálni, kihasználva a Motorola 68000-es piocesszorának képességeit, mely az akkori (pl. 8086, Z8000, stb.) processzorokhoz képest lényegesen többet tudott. Benne már megtalálhatók a niultitaszkos oprendszer íiásához szükséges lehetőségek.

Azonban való >zínűleg nem voltak olyan jó managerek, mint hardver és szoftyer szakemberek - csődbe jutott az AMIGA Inc.

Úgy adódott, hogy a Commodore éppen nézelődött a környéken, mivel valaki megjósolta a C64 elavulását. Mivel belátta, hogy ez valószínűleg előbb-utóbb be fog következni, észrevette eme kis cég tönkremenetelét és fantáziát látott a dologban, ezért megvette.

Amikor megvette a nagy C= a kis Amigát, *az* sajnos még nem volt teljesen kész ami néhány helyen még most is fellelhető (pl. BFFR, stb.J.

Mindenesetre 1985-ben a Commodore bemutatta az AlOOO-t, amely szinte sokkoló erejű volt mind teljesítmény, mind ár szempontjából (kb. 5500 DM).

Abban az időben egy átlagos munkahelyen itthon csak álmodoztak számítógépről, és külföldön is inkább az IBM 286 os AT-jal voltak többségben alkalmazva és elteijedve, valamint a nagyobb intézményeknekben UNIX op. rendszerrel működő több állomást kiszolgáló nagyteljesímányű gépek. Az egyetlen hasonlót nyújtó gépek a APPLE Macintosh és az Atarí gépei voltak, amelyeket szintén a Motorola 68000-cs családja hajtott, és hajt a mai napig (kivételek persze most már a PowerMac-ek). Abban az időben a gép bombaként robbant: volt aki szidta, volt aki szerette (inkább ez utóbbiak voltak többen), de az biztos, hogy hidegen senkit sem hagyott. Az AlOOO-es tudása az alábbiakban vázolva a következő:

CPU: MC68000 7.14MHz, 32 bites belső felépítés!
RAM: 256Kb + 192Kb ROM,
kútón kérésre szállították 512KB + ROM-mal is.
Hátértár: 880Kb-os 3.5"-os lemezegység,
melyek száma max. 4db lehet,
kérésre 10-20 Mb-os Winctiester.

Grafika: 320x200
320x400
640x200
640x400
max. 4096 szín a 4096-ból,
vagy 64, 32, 16, 8, 4, 2 szín.
Hang: 2x2 *csatorna (oldalanként 2),*
8 bit felbontásban, max 28 KHz-es mintavételi frekvenciával,
(A CD-é 44.1 KHz, igaz 14 biten vagy most már inkább 16 biten. A mulíibites rendszert nem is ecsetelve, 256 szoros túlmintavétel, de csak egybtíes felbontás).
A hullámforma teljesen szabadon programozható
Koprocesszorok: 2 db - a BLJTTER és a COPPER - a *grafika segítésére.*

A koproresszorok teljesítménye jellemzően olyan nagy, hogy ha a prcesszornak kellene végrehajtania a műveleteket kiváltásképpen, akkor kb. 70MHz-cel kellene söpörnie az adatbuszon, ami abban az időben még a nagygépeknél is szinte álom volt.

Ahhoz, hogy fel tudjuk mérni a gép jelentőségét, állítsuk egy-két kortársa mellé. Kezdjük talán az IBM PC-vel mely irodai alkalmazásra lett kifejlesztve és az IBM eredetileg monokróm videó csatolóval szállította, 360Kb-ra formattálható lemezegységgel és egy olyan processzorral, amely belül ugyan 16 bites, a külvilág felé azonban csak 8.

A turbó változatnál is csak 8MHz-en poroszkált a buszon. Külön rontja a helyzetét az a tény, hogy a processzor regiszterei erősen követik az elód struktúráját, amely első ránézésre nem egy új processzorcsalád első tagjának néz ki, hanem sokkal inkább az Intel 8085 processzorának 16 bites változata.

Bár mire az Amiga megjelent, elterjedt a PC-k AT sorozata, amely az Intel 80286-os processzorét tartalmazta lOMhz-es órajellel, 16 bites külső buszszerkezettel. Későbbi változatai elérik az igen tekintélyes 14Mhz-es órajelet. Igaz már ebben a prociban fellelhető a multitaszkos fejlődési irány, ezt azonban a lefelé való kompatibilitási igény miatt nagyon sokáig nem használták ki. Ezenkívül meg kell említeni azt, .hogy a processzor nem igazán tudja egyben látni a memóriát. Nevezetesen 64Kb-os szeletekre osztja fel, és mivel még a Pentium is bekapcsoláskor 8086-ként jelentkezik be, ez a hiba igen érzékenyen érinti a sebességet. Az Intel sorozatú processzorok másik szépséghibája, hogy követi azt az egy központi regiszter koncepciót, amit az első processzorok alkalmaztak, mely szerint egy központi regiszterben (accumulator) történnek az események, a többi regiszter csak az operandusokat tartalmazza. Azonkívül a PC-nek hangja sem igazán volt abban az időben, a beépített csicsergőn kívül, amely egy, azaz 1 bites felbontásban, igaz majd 8Mhz-el képes volt csicseregni! Kár hogy az ember csak 20 KHz-ig képes hallani, bár kutatások szerint a felette lévő hangokat is felfogjuk, de ez nem tudatosodik. A PC-nek ennek ellenére vannak előnyei, amelyek a következők: a szabad bővíthetőség (végül is mindenféle kártyát dughatunk bele: hang, vi-

deo, stb.), de ezt egyben hátránynak is lehetne nevezni, mert ez azt is jelenti, hogy az írandó program ha nem veszi ezt figyelembe, akkor valószínűleg csak egyfajta hardverkiépítésben fog csillogni-villogni, azaz futni. A másik előnye az, hogy immár sok játékprogram létezik rá, ezáltal szerintem nemsokára játékgépnek kell átminősíteni a PC-t.

Bár a sebesség és a MlPS-ek terén a PC igen elhúzott az átlag Amigától, azért a '060-as igen nagy pofon a Pentiumnak, mivel nem csak előbb jelent meg, hanem a teljesítménye is jóval felülmúlja azt.

Mindenesetre meg kell jegyezni, hogy Assembly szinten a 68000-est sokkal kényelmesebb programozni mint egy 8086-ost, nem beszélve a memóriakezelésről. Viszont magas szintű nyelven a processzor hátrányait annyira nem lehet észrevenni. Bár az Amiga operációs rendszere igen komplex a multitaszkos felépítés miatt, ezért ezt sem olyan egyszerű programozni (azért nem kell megijedni). Ezért viszont óvatosabban kell eljárni, de ezt a tiszta logikus rendszer inkább örömnek érezteti, mintsem hátránynak. A processzor különbségek pedig nagyon összemosódnak egy magas szintű nyelv használatakor (pascal, modula, oberon stb.).

• Tehát mint láttuk, hardveresen az Amiga fölénnyel söpörte le az asztalról a PC-t (az más kérdés, hogy az átlagfelhasználó aki a munkahelyén nagy nehezen elsajátította a PC kezelését, már nem vesz más gépet otthonra, mert fél tőle), az APPLE Macintosh-ait, melyek az áruk miatt nem igazán versenytársai az Amigának, tehát maradt az Atari ST sorozata.

Az Atari ST-k is Motorola 680x0 sorozatú procikra épülnek, de képileg és oprendszerileg súlyos hiányosságok róhatok fel nekik, viszont tény, hogy aki hanggal akar profi, ill. félprofi szinten dolgozni, annak csak ajánlani tudom az ATARI Falcont a maga 16 bites hangjával és az ezt támogató Motorola DSP-vel (digitális jelprocesszor, jellemző a teljesítményére az, hogy a NEJÍT kockában egy ilyen DSP végzi az emberi hang szintetizálásához szükséges számításokat, a 3D-s vektorok számításait - mindezt 24MHz-es órajellel és még van ideje lekezelni a DMA csatornákat is).

Az ATARI gépeket én inkább a PC-re kifejlesztett hardverek Motorola környezetben való használatának jó példájának tartom. Tulajdonképpen ez a gép lett volna a C64 igazi utódja, mivel a Commodore-1 otthagyva ide távozott a C64 fejlesztőinek egy része. Ez az a lépcsőfok, amely átvezette volna a felhasználót a kisgépes rendszerből a nagyobbak felé, grafikus felülettel. Sajnes az oprendszere nem multitaszkos, sőt a APPLE gépei sem voltak azok a közelmúltig.

Nos, be kell látnunk, hogy abban az időben ez a hardver egyedülálló volt a maga nemében. Ha az ESCOM - aki a Commodore utódja - kihasználná azt, hogy az Amiga fejlesztői az utolsó pillanatig dolgoztak, tehát fejlesztettek, nos, akkor az Amiga most is felvenné a versenyt. (Bizonyos hírek keringenek a HP által gyártott, de a Commodore mérnökei által fejlesztett Amiga **RISC** processzorról, 24 bites grafikáról.) Az igazság kedvéért meg kell említeni, hogy volt egy komoly vetélytársa az Amigának, nevezetesen az Arcon cég Archimedes nevű gépe, ami kinézetre nagyon hasonlított az Amigára, de belül a teljesítményt egy akkor még igencsak ígéretesnek számító RISC (csökkentett utasításkészletű processzor) processzor adta. Sajnos nem terjedt el. A gépet tudtommal csak amerikában hozták forgalomba.

Az Amigáról...

Nem sokkal az AlOOO-es kiadása után megjelent az A500-as ami kisebb dobozban (az AlOOO-es PC szerű dobozban volt) jelent meg, némi belső változtatás után. Nevezetesen a rustom chip-jeit még jobban integrálták.

Azokat a nagy integráltsága VI,SI technológiával készülő áramköröket nevezik így, amelyekben a társprocesszorok, a DMA vezérlő, a CHIP ram vezérlő, valamint a floppy, a hang és egyéb funkciókat végző áramkörök vannak integrálva.

A dobozba beépítették a billentyűzetet, valamint az oprendszer több részét is, amit az AlOOO-en úgy kellet mindig lemezről betölteni egy erre a célra használt RAM-ba.

Az A500-ba ezenkívül 512Kb RAM-ot **(CHIP)** tettek alapesetben amit tovább bővíthettünk akár 8Mb-ig is **(FÁST).** Természetesen a behelyezett mindenféle bővítést a rendszer ugyanúgy azonnal, installálás nélkül észrevette és kezelte, mint ahogyan most is. Nem sokkal ezután kiadták az A2000-et, amely nagy PC-s dobozban kapott helyet, több bővítési lehetőséggel rendelkezett (3 Zorroll-es busz, 3 PC busz), valamint IMb CHIP RAM-al szállították. Nem sokkal később jelent meg egy A2500-as gép, mely csak abban külöqbözött elődjétől, hogy MC68020-as processzor gondoskodott a nagyobb sebességről. Majd némi időveszteséggel kiadták az A500+ és az A3000-es gépeket, amelyekben bővített grafikuskészlet volt. Ez megkövetelte az alapesetben is IMb-os CHIP RAM-ot.

Bár a színek száma nem változott, a felbontást megnövelték az új chipkészlettel, amit ECS-nek neveznek (Enhanced Chip Set).

Az op. rendszer immár 2.0 verzióra növekedett az eddigi AlOOO-nél 1.1, A500-iiál 1.2, később 1.3-ról. A3000-es ezenkívül még tartalmazott néhány újítást, ami abban mutatkozott meg, hogy a PC-s dobozba belekerült egy SCSI csatoló, valamint **Zorro III-as** bővítő portok, amik mar 32 bitesek voltak, mivel az A3000-ben egy '030-as processzor dolgozik 25Mhz-es órajelen. Az A500+ elődjével lényegileg megegyezett a CHIP RAM-on és az ECS készlet változtatásán kívül. Ezenkívül a Power feliratú LED nemcsak az audio szűrő bekapcsolt állapotát jelezte hanem a gép tápfeszültségének meglétét is. Ezek után kis idővel - a jó munkához idő kell, a lassúhoz pedig még több - megjelentek a szinte teljesen más, nagytudású Amigák, az A4000 és az A1200, valamint a játékra szánt A600-as, amely az A500+ átdobozolt és IDE/AT csatlakozóval felszerelt mása.

Az A1200 és az A4000 az **AGA** (Advanced Graphic Adapter) grafikával immár nem 4096 színű palettából rakják össze a színeket, hanem 16 millióból.

Igaz ebből HAM8 üzemmódban max. 262144 lehet egyszerre a képernyőn. Az op. rendszer pedig áttért a 3.0-ás verziószámra, valamint már lehet tudni, hogy lesz A1300-as amely ugyan lényeges változást nem fog tartalmazni az A1200-eshez képest, csak egy nagyobb procit. Az A1200-csben egy **68EC020-as** van ami 14,7Mhz-en jár, ahol az EC jelölés azt jelenti, hogy a proci 32 bites címbuszából (4Gb-ig lehetne címezni) csak 24 bit van kivezetve, - mint az Intel családnál az SX sorozat - így a proci olcsóbb volt. Szóval az új gépben egy 'EC030 fog izegni, és a gépet egy CD-ROM meghajtóval fog-

ják egybeépíteni. Az A4000-ben egy '040-es gondoskodik a gép fűtéséről, s mindezt 25Mhz-en teszi. Ezenkívül az A4000-es HD-s floppy-val rendelkezik, ami 1.7Mb-ot jelent formázva. Mind a két gépben 2Mb CH1P RAM található.

Pár hónap elteltével azután kihoztak egy '030-assal működő A4000-est, közel fele annyiért. Sajnos ezeknél a gépeknél a Commodore szakított az A3000-ben jól bevált SCSI vezérlővel és helyette egy IDE/AT csatolót tettek, ami persze jóval lassúbb, viszont olcsóbbak a winchester-ek hozzá, ami már lassan nem lesz igaz. Itt térek ki arra a tényre, hogy az Amigának van két mostoha testvére a CDTV amit a Commodore interaktív multimédiás gépnek szánt és nyugodtan nevezhetjük a Phillips CD-I elődjének, mert bár megelőzte azt, nem vált szabvánnyá. Megjelenését tekintve az A500+- előtt jelent meg, de már ECS-t tartalmazó gép volt. Ez kinézetre egy HiFi toronyba illeszthető, és oda illő tudású CD játszó, amelyhez TV-t kapcsolva, az játék konzolként használható. Egyéb kiegészítőkkel CD-ROM-al ellátott normál Amigává lehetett alakítani. Majd az AGA-s gépek megjelenésével egyidőben megjelent a CD32 amely tulajdonképpen egy A1200-es, erősen játék konzol kinézetű rokona (ilyen pl. SEGA Megadrive). Ez magasabb verziószámú op. rendszert tartalmaz, nevezetesen a 3.1-eset, amely kezeli a CD-t. Ez a gep is átalakítható teljes értékű A1200 géppé. A ROM-ja a kiegészítések miatt 512Kb-ról IMb-ra duzzadt. Természetesen az A3000-es és a A4000-es létezik torony kivitelben is. Amiért ezt megemlítjük, az az érdekes tény, hogy az **A3000T**-hez lehetett kérni a **UNIX** op. rendszert is.

1.1.2 A hardver

Elöljáróban le kell szögeznem, hogy ez a rész is feltételez már némi jártasságot hardver környezetben. Tehát kezdjük az elején. Az AlOOO-el ne foglalkozzunk részletesen, mivel elég kevés volt belőle iithon. Az A500-as az amiről érdemes kezdésnek szólni. Nos, A500-ban található egy mikroprocesszor, melynek a külső megjelenését a Motorola gyár adta meg. A lípusjelzése M68000, ami abból adódik, hogy állítólag pont ennyi tranzisztor található a szilícium lapkán. Na most ennek még nem számoltam utána, de akár igaz is lehet, tekintve, hogy a processzor egyébként abban az időben igen nagy teljesítményűnek számított, állítólag az, amerikai Persing rakéták agyát képezte, és emiatt COCOM listás is volt. Tudtommal az oroszok nem is tudták lemásolni, legalábbis nem tudok orosz változatáról, ellenben az Intel processzoroknak létezik. A processzor belső felépítése egyébként a következő: található benne 8 adatregiszter (D0-D7-ig), valamint ugyanennyi címregiszter (A0-A7, de az A7 a rendszer által veremmutatónak használt regiszter és ebből kettő van. Külön user- és külön supervisor, amely a processzor állapotától függően változik, így érhető el a szeparált veremkezelés) van egy programszámláló regisztere (PC), valamint egy állapotjelző regiszter (SR). Minden regiszter 32 bit szélességű, kivétel az SR amely csak(?) 16 bites, de hát minek is több.

Azért szólni kell arról, hogy ezekből sajnos a külvilág csak 16 bites adatbuszt és 24 bites címbuszt lát. így egy regisztert két buszciklusból lehet csak

megtölteni, valamint a tár nem 4Gb-ig címezhető, hanem csak 16Mb-ig. Nagy erőssége a processzornak, hogy minden regiszter mindenre használható, nincs akkora kötöttség mint az Intel családnál. A processzor másik nagy előnye hardver szempontból, hogy a busz ciklusainak nem kell feltétlenül szinkronizáltnak lenniük, minden perifériához a saját órajelének megfelelően szólhat a processzor. így egy kellően gyors egységgel teljes órajellel beszélhet, míg egy másikkal csak lassabban tudja magát megértetni. A processzornak egyébként 54 alaputasítása van, de a 14 féle címzésmód miatt az utasítások száma meghaladja az ezret. Később részletesebben szólni fogunk még a processzorról az Assembly miatt.

A processzor amúgy 7.14Mhz-en söpör az adatbuszon, ami főleg akkor porzik igazán, ha a proci a CHIP RAM-hoz nyúl valamilyen okból. Rögtön látni fogjuk, hogy mit is jelent ez pontosan, de előtte szólnánk néhány szót a többi, alaplapon tévelygő áramkörről is. Tehát a gépben található még egy **FAT AGNUS IC** is, ami a nevét a kinézete miatt kapta (PLCC 68-as tokozású).

Itt érdemes lesz egy kicsit elidőzni, mivel igen érdekes dolgokat művel a kövér hölgyemény. Vessünk egy pillantást a 2. képre amely az áramkör belsejét szemlélteti. Első ránézésre is kiválóan látszik, hogy nem unatkoztak akik tervezték. Ha figyelmesebben megnézzük észrevesszük, hogy ez az áramkör tartalmazza a Cl IIP RAM frissítéséhez szükséges áramköröket, valamint az Amiga két nagyágyúját a **BLITTER-t** és a **COPPER-t.**

Ezen társprocesszorok nélkül az Amiga csak egy 68000-es alapú IBM PC lehetne ("hardveresen). Ezek a társprocesszorok nem a processzor társprocesszor koncepcióját valósítják meg!

A COPPER képes az AMIGA ún. custom regiszterei közül bármelyiket megváltoztatni, valamint folyamatosan nyilvántartja a monitoron látható képet létrehozó elektronsugár X és Y koordinátáit. Mindössze három utasítása van (igazi RISC), amik a következők: MOVE, WAIT, SKIP. A MOVE segítségével adatokat mozgathatunk, a WAIT a megadott képernyőpozíció eléréséig vár, a SKIP pedig egy feltétel teljesülésétől függően átugorja a következő utasítást. Feledatát egy speciális program, a copper listával lehet megadni, amely tartalmazhatja a bitplane-ek elhelyeszkedését a memóriában, a sprite adatok helyét, az aktuális üzemmódot, stb.

A **BLITTER** segítségével rendkívül nagy sebességgel lehet a Cl IIP RAM tartalmát mozgatni. A mozgatás során a BLITTER az összes logikai művelet elvégzésére is alkalmas, így lehetségesek a szuper nagyságú alakzatok, a **BOB**-ok (Ellitter OBject) - mivel a sprite-ok csak 16 bit szélesek lehetnek - képernyőn való mozgatása a háttérkép érintetlenül hagyásával. A vonalak, körök és a különféle területek feltöltése" (fillezése) is a BLITTER segítségével érhető el. Mindezeket a feladatokat igen nagy sebességgel tudja elvégezni, amire a proci nem lenne képes (most azért már egy jól megtermett '040 valószínűleg kenterbe vágná, na de hol volt az még akkor?). A BLITTER működése nem befolyásolja a CPU által végzett feladatokat (ha azok a Fastramban találhatóak), tehát a proci sebessége nem csökken, amíg a BLITTER dolgozik.

A FAT AGNUS-ban található továbbá a DMA áramkör is. Sajnos csak abban a memóriában képes ezeket a csodálatos dolgokat elvégezni, amit **CHIP** RAM-nak nevezünk. Emiatt minden megjelenítendő képnek és hangnak itt, ebben a memóriában kell elhelyezkednie.

Ha már itt tartunk, meg kell jegyeznem, hogy az Amiga nem csak **CHIP** RAM-ot használhat, hanem a processzor saját elérésű, ún. **FÁST** RAM-ját is, ami azért kapta a nevét, mert a Custom chip-ek nem szólnak bele annak elérésébe, így az adatok a processzornak bármikor a rendelkezésére állhatnak. Szerencsés esetben ha a program ebben a típusú memóriában helyezkedik el, akkor amíg a BLITTER, a COPPER, ül. ha a DMA dolgozik a CHIP RAM-ban, akkor a processzor nem áll és vár a memóriára, hanem nyugodtan molyol a **FÁST** RAM-ban található programmal. Itt jegyezném meg, ho'gy az A500+ és az A3000-hez egy új AGNUST fejlesztettek ki, amely BIG FAT AGNUS néven terjedt el és vált népszerűvé. Nevét nem annak köszönhette, hogy most még nagyobb kocka lett az alaplapra forrasztva, hanem hogy IMb-ot volt képes **CHIP** RAM-nak kezelni, amire persze szükség is volt.

A következő IC az alaplapon amit megvizsgálunk, a **PAULA** nevet viseli, a hangok és néhány periféria kezeléséért felelős. A 4 hangcsatorna - a kor elvárásainak megfelelően - sztereóban szól: két hangcsatorna a bal, valamint két hangcsatorna a jobb oldalon. Némi manipulációval rá lehet bírni, hogy akár 8 csatornán is szóljon. A csatornák hangereje 64 lépésben szabályozható, amit néhány szemfüles programozó kihasználva készített a gépre olyan programot, amely az egyébként 8 bites felbontást 12 bitessé alakítja. A lrekvenciatartomány 9 oktáv, ami a zenék megszólaltatását tekintve bőségesen elegendő (egy zongorán is ennyi van, valamint ez 20Hz-20Khz-ig terjedő frekvenciatartományt jelent). A megszólaltatott hangok hullámformáját teljesen szabadon lehet definiálni. A hangok modulálására lehetőség van akár amplitúdóban (AM), mind frekvencia moduláltan (FM) is. Ezenkívül lehetséges még egy aluláteresztő szűrőt is iktatni a hang útjába, aminek 5KI íz tájékán mutatkozik meg a hatása. Célja a kvantálási hiba kiszűrése, amelyet olyan jól tesz, hogy a magas hangok nem nagyon szólnak utána, de a feladatát tökéletesen ellátja, mivel a kvantálási hiba sem hallatszódik. Egyébként kikapcsolt állapotáról a Power feliratú LED ad tájékoztatást, újabb gépeken csak félfényerővel világít ha ki van kapcsolva. A PAULA-t mostanában sok támadás éri, nem is ok nélkül.

Már régóta ígérgetik a 16 bites hangot az Amigába, de ne feledjük, hogy ha elmegyünk egy számítógépes találkozóra, az IBM PC-sek a jó öreg Amigáról átmásolt zenéket hallgatják, pedig nekik van 16 bitük, és sok csatornájuk! A PAULA ezenkívül még kiveszi részét a soros illesztő munkájából is olyan módon, hogy szabadon programozható az átviteli sebessége, így elérhető olyan nem szabványos átviteli sebesség is mint a MIDI-é, aminek hardveres megvalósítása lecsökken néhány hétköznapi optocsatolóra, valamint három csatlakozóra. A PAULA ezenkívül még részt vesz a floppy vezérlésben is mint adat iró lll. olvasó, valamint, ellátja a processzor megszakítás-kezelését.

A **DENIS** a grafikáért felelős. Mint tudjuk az Amiga igen érdekesen kezeli a képernyőt. A képernyőt az Amiga ún. bitplane-ken keresztül látja, ami annyit jelent, hogy a memcsiben több képernyő is van. Erre azért van szük-

ség, mert a képernyő pixelei nemcsak egymás mellett helyezkednek el, hanem térben egymás mögött. A megértéséhez nézzük meg az 5. ábrát. Mint latjuk, ha az egyes számú pixel színére vagyunk kíváncsiak egy négy bitplan-es képernyőn (ami ossz. 16 szírit jelent), akkor minden bitplane első pixelét meg kell vizsgálnunk, és az így kapott szám mutatja meg a DENIS-ben található színt tartalmazó regiszter számát. Ha ebben a regiszterben mondjuk fehér szín van, akkor a pixel fehér. Ezt a fajta képernyőkezelési módot nevezik **planar-nak.** A PC-n ez máshogyan néz ki. Ott a pontok egymás mellett helyezkednek el és egy pont mérete bájt nagyságú. Ez a bájt tartalmazza a PC videokezelő áramkörének regiszterszárnát. A PC-s képtárolást nevezik **chunky-nak.** Mind a kettőnek megvannak a maga előnyei: egy olyan játékot mint a Doom, vagy a WolfEinstein3D, sokkal könnyebb megírni PC-n, mint Amigán! Természetesen a magas szintű programozási nyelvek ezeket a különbségeket részben elfedik előlünk.

Mégis a 3.0-ás rendszer már tartalmaz egy függvényt, aminek használatával elvégezhetünk egy **Chunky-Planar** konverziót, .és így az Amigánkat is rhunky módszerrel programozhatjuk. A CD32~ben ezt egy hardver végzi, ami lényegesen gyorsabb mint az A1200-es szoftveres megoldása. Az újabb Amigákban ígérik, hogy ez hardveresen benne lesz. De térjünk vissza a DENIS-hez.

A graiikus felbontás - normál a 320x256-tól a 640x512 képpontig terjed, de lehetőség van a keret nélküli megjelenítésre **(overscan)** is, ekkor a vízszintes és függőleges felbontás megnő (maximálisan 720x560 pixelig). A különböző képek megjelenítéséhez a következő grafikus üzemmódok közül válogathatunk:

- STANDARD: 2,4,8,16,32 szín 320x256, vagy - interlace módban - 320x512 képpontos lelbontas mellett.
- 1II-RES: 2,4,8,16 szín 640x256, vagy - interlace módban - 640x512 pixel felbontás mellett.
- EXTRA IIALFBR1GHT (extra félfényerő): 64 szín 320x256, - interlace mód ban 320x512 pixel felbontás mellett.
- HAM (Hold And Modily): 4096 szín 320x256, vagy - interlace módban - 320x512 pixel felbontás mellett.
- DINAM1C HAM: ez 4096 szín mellett akár 640x512 pixel felbontást is tud, a copper lehetóségeitkihasználva (nem standard üzemmód).

Az ECS gépek ezenfelül tudnak még ún. Productvity módot is, amely 1280x512 és - interlace nélkül - 640x480.

A **DENIS** segédprocesszor felelős a sprite-ok megjelenítéséért is, mint ez nyilvánvaló, ha megnézzük a 3. ábrát, amely a DENIS belső világát szemlélteti. Egyszerre 8 darab sprite használható, amelyek vezérlésére 16 sprite-regiszter áll rendelkezésre. A sprite vízszintes mérete nem változtatható, de lehetőség van a sprite speciállis felhasználására is, azaz több sprite összekapcsolható (egyébként megjegyzendő, hogy a látványos játékokban mozgatott nagy objektumok nem sprite-ok, hanem ún. BOB-ok).

A **GARY** áramkör tulajdonképpen csak olyan dolgokat tartalmaz, amelyek a rendszer éleiéhez de nem látványosak. A legfontosabb, hogy ő az egyetlen férfi a csapatban. Valószínűleg azt a filozotiat Kovet-

ték az Amiga fejlesztői, amit egy rövid történettel világítok meg: miszerint a 60 éves házassági évfordulón megkérdezik Józsi bácsit, hogy ugyan Józsi bácsi, osztán hogy tetszett bírni ennyi ideig a Mari nénivel? Hát liarn úgy, hogy a kisebb döntéseket orá hagytam, és én csak az igazán nagy döntéseket hoztam meg. És Józsi bácsi, életében hányszor kényszerült nagy döntésre? Ahogy így ... belegondolok fiam, az életben nincs is semmi olyan igazán nagy dolog. De visszatérve a GARY re: olyan ícladatokat lát el, mint a RAM/ROM irányítása, valamint ő generálja a CPU számára a _DTACK jelet, valamint részt vesz a floppy vezérlésében is.

A **CIA-k.** Ezek a chipek kezelik a tulajdonképpeni perifériákat, mint pl. a soros port handshaking jelei, párhuzamos port, ők olvassák a billentyűzet mikrokontrollerét, valamint segítenek a floppy-nak, de ott vannak a joystick, valamint az egér kezelesében is. Általuk lehet ki/be kapcsolgatni az akilátereszto' szűrőt, amit a PAULA-nál mar tárgyaltunk. Egyébként igen lassú áramkörök, mert még 8 bites adatbusz koncepcióhoz készültek. Ezért is helyezkednek ugy el, hogy az egyik az adatbusz alsó felét, a másik a felsőt foglalja.

A rendszer egyébként elég nyílt, bővíthető úgynevezett auloconfig-os kártyákkal, amiknek csak a processzor memóriacímző képessége szab határt (pl.: lKb helyfoglalása kártyákból akár százat is rá lehetne rakni). Itt jegyezném meg, hogy a Windows '95 pont emiatt is késett a legtöbbet (plug and play). A PC-kbe ezt most akarják bevezetni, amit az AMIGA-mar '85-ben az A1000-ben is használt, és kiválóan működik még most is, igaz az adat- és címvonalat 32 bitessé bővítették az újabb változatban **(Zorro III).**

Az **AGA** chipkészlet nagy változásokat hozott az Amiga belvilágában, nézzük meg azokat:

ALICB a korábbi FAT AGNUST váltotta fel némi bővítgetéssel, nevezetesen már 2Mb CHIP RAM-ot kezel, valamint ezt mar 32 biten teszi.

, Sajnos a BLITTER-t és a COPPER-t nem változtatták meg, így azok 16 bitesek, és gyorsaságuk sem növekedett meg.

LISA, ő a **DENIS-t** váltotta fel úgy, hogy a sprite-ok már akár 64 pixel szélesek is lehetnek. Valamint változtak a felbontások a következő méitékben: a színek száma minden felbontásban 256, valamint **IIAM8, HAM** stb.

PAL:	**Hires** 640x256,
	Hires lace 640x512,
	Lores 320x256,
	Loreslace 320x512
	Super Hires 280x256
	Super Hires lace 1280x512
	az ossz. frekv.50Hz,15.60KHz
NTSC:	**Hires** 640x200
	Hires lace 640x400
,,	**Lores** 320x200
	Lores lace 320x400
	Super Hires 1280x200
	Super Hireslace 1280x400
	az ossz. frekv,60Hz,15.72KHz

Euro36:	**Hires** 640x200
	Hires lace 640x400
	Lores 320x200
	Lores lace 320x400
	Super Hires 1280x200
	Super Hireslace 1280x400
	az ossz. frekv.73Hz,15.69KHz
Euro72:	**Productivity** 640x400
	Productivity lace 640x800
	az ossz. frekv.70Hz,31.43KHz
Super72:	**Hires** 400x300
	Hires lace 400x600
	Super Hires 800x300
	Super Hires lace 800x600
	az ossz. frekv.72Hz,24.62KHz
Multiscan:	**Productivity** 640x400
	Productivity lace 640x800
	az ossz. frekv.60Hz,31.44KHz
DBLPal:	**Hires** 640x256
	Hires lace 640x1024
	Hires no flicker 640x512
	Lores 320x256
	Lores lace 320x1024
	Lores no flicker 320x512
	az ossz. frekv.50Hz,29.45KHz
DBLNtsc:	**Hires** 640x200
	Hires lace 640x800
	Hires no flicker 640x400
	Lores 320x200
	Lores lace 320x800
	Lores no flicker 320x400
	az ossz. frekv.59Hz,29.20KHz
A2024:	lOHz 1024x1024 4színű!
	15Hz 1024x1024 4színű!
	a frekveciái 50Hz,15.72KHz.

Mint látható azért ez nem olyan rossz, főleg ha figyelembe vesszük, hogy minden szín 16,8 milliós palettáról kerül elő, az, igaz viszont, hogy a HAM8 (ahol a képernyőn egyszerre 262144 szín) nem olyan mint az igazi 24 bites képek (ahol mind a 16,8 millió szín kint lehet). Fűzzük hozzá még azt a hibáját az AGA chipkészletnek, hogy ha egy képet akarunk scrollozni a képernyőn, nem PAL vagy NTSC kép esetében (pl. DBLPal képernyőn) a hardver igen csúf kinézetű hibát produkál Ennek ellenére a képernyőnk azonban virtuálisan 16384x16384 pixel méretű lehet, amit a rendszer automatikusan lekezel, és az éppen aktuális képernyőhelyre scrolloz, ha ezt megkívánjuk tőle. A másik dolog amit csak megjegyzésképpen fűzök a dolgokhoz, hogy ha nyitunk egy más típusú képernyőt, pl. eddig DBLPalban dolgoztunk majd elindítunk egy olyan programot, ami mondjuk PAL rendszerben kezeli a kép-

ernyőt, akkor az Amiga nagyon egyszerűen megoldja a feladatot: ha a Screent lehúzzuk, a felbontást odaigazítja, amit azért én még egyetlen grafikus kártyán sem láttam (PC-n SVGA stb.), s ezt a kis Amiga helyből megcsinálja!

Van egy kis bibi a rendszeben, amiről azért illik szólni. Bár ez a bibi nem volt olyan feltűnő a régebbi Amigákon, csak ezeknél az újabbaknál lett feletteb zavaró. Nevezetesen az, hogy mint már említettük a processzor addig unatkozik és nem csinál semmit, ameddig a costum chip-ek dolgoznak a CHIP RAM-ban. Tehát ha képet jelenítenek meg, zenét játszanak (ha van a gépünkben FÁST RAM, ez a helyzet javul, mert a processzor ugyan dolgozhat a FÁST RAM-ban, de ebből a képernyőn nem sokat fogunk tapasztalni) a processzor nem fér hozzá, csak például akkor amikor a képet kirajzoló elektronsugár éppen visszafut, hogy kezdje a másik kép kirajzolását. Vagy a kép rajzolása éppen az overscan területen tart. Ilyen kedvszegő dolgokat akkor vagyunk kénytelenek elviselni, ha a képernyőn a nagy felbontás mellett igen sok a szín is. Nézzük meg, hogy milyen körülmények között is fordul ez elő:

A CHIP-RAM buszterhelés a különböző sávszélességek esetén

Mód:	**Bitplane-ek száma**	**Színek száma**	1x	2x	4x
Lores					
320x200/256	6	64	75%	38%	19%
320x400/512	7	128	88%	44%	22%
160x480	8	256	100%	50%	25%
160x960 lace	8HAM	262144+	100%	50%	25%
Hires					
640x200/256	5	32	-	63%	32%
640x400/512 lace	6EHB	64 (régi EHB)	-	75%	38%
320x480	6 HAM	4096 (régi HAM)	-	75%	38%
320x960 lace	6	64	-	75%	38%
	7	128	-	88%	44%
	8	256	-	100%	50%
	8 HAM	262144+	-	100%	50%
SuperHires					
1280x200/256	1	2 (a 16 mill.-ből)	50%	25%	13%
1280x400/512 lace	2	4 (a 16 mill.-ból)	100%	50%	25%
640x480	3	8	-	75%	38%
640x960 lace	4	16	-	100%	50%
	5	32	-	-	63%
	6EHB	64 (régTEHB)	-	-	75%
	6HAM	4096 (régi HAM)	-	-	75%
	6	64	-	-	75%
	7	128	-	-	88%
	8	256	-	-	100%
	8 HAM	262144+	-	-	100%

Ahogy kitűnik a táblázatból, az ïij felbontások **111.** színek használatakor eléggé le lesz terhelve a Chip-RAM busz, és ez okozza a lassulást. A táblázatban szereplő 100% azt jelenti, hogy a processzor és a Blitter gyakorlatilag nem fér a CH1P RAM-hoz.

Szóljunk egy pár szót ismét a processzorról, csak azért mert az AGA-s gépekben egy kicsit nagyobbak vannak. Pédául az A1200-esben gubbaszt egy MC68EC020-as ami pont a jobb Amiga gomb alatt található (ha valaki bíztatni szeretné azt az imént említett gomb brutális használatával könnyedén megteheti). Egyébként ha valaki megnézi akkor látni fogja, hogy a legkisebb százlábú a többi között. Az hogy EC-s annyit jelent, hogy a címbuszból nincs kivezetve csak az alsó 24 bit, és nincs benne **MMU** (Memory Management Unit, amivel pl. virtuális memóriát lehetne kezelni). Viszont ígyjóval olcsóbb. Persze a bővített utasítások benne vannak! Valamint a A3000 gépből ismert '030-as, valamint az A4000-kben már a '040-es processzor ami egy tokon belül egyesíti a **CISC** és a **RISC** processzorok előnyét, ami meg is látszik a teljesítményén amellett, hogy teljesen megérti a 68000-s utasításait is. Természetesen mindegyik imént említett processzorban található némi cache memória, igaz nem olyan nagy mértékben mint a PC-kből esetleg ismerős 256Kb, viszont az a kicsi (pl. a '060-asban 32Kb) mind azon a szilícium lapkán helyezkedik el amin maga a processzor is. Ez annyit jelent, hogy rettentő gyors az elérése, amit külsőleg csatlakoztatható társai fizikai okok miatt nem érhetnek el.

De szóljunk még arról a tényről is, hogy a PC-nek nagy előnye az a virtuális tár lehetősége, ami az Amigákon csak a drágább gépek része. Vegyük figyelembe azonban azt, hogy a PC meghalna ha ezt nem tudna helyből, mivel a processzora 64Kbyte-os szeletekben látja az egész memóriát. Tehát, ha a PC-ben nagyobb memória van mint az alap 640Kbyte, akkor az ezen felül eső részt a rendszer 64Kbyte-os részletekben látja. Még a Pentium processzor is bekapcsolás után ilyen állapotban jelentkezik be! Nem véletlen az, hogy a PC-ről ennyit beszélek, mert az Amigát mindig ahhoz akarják hasonlítani. Ez nem igazán szerencsés, mert a két gépet egészen más okból fejlesztették ki. Másrészt sajnos az itthoni kereskedelemben az Amigához igen kevés dolgot lehet kapni, a PC-hez viszont jóval több kiegészítőt, így szegény Amigást rákényszerítik, hogy a PC-hez is értsen (sokszor jobban mint a PC-sek maguk!), és ismerje fel. hogy saját gépénél mit, s hogyan tud hasznosítani.

1.1.3 Az operációs rendszer

Az Amiga alapvető kezelése az úgynevezett „ikonvezérelt operációs rendszer" elméletén alapul, amelyet a Xerox cég fejlesztett ki, bár nagy karriert az Apple gépein futott be. Ennek a rendszernek elsődleges jellemzője és inputforrása az egér használata: a programok kijelölése, másolása, törlése stb. az egér segítségével, az ún. ikonokon keresztül történik. Ez a módszer rendkívül egyszerű, olyannyira, hogy egy-két perc gyakorlás után bárki elsajátíthatja. A WorkBench az Amiga operációs rendszerére, az AmigaDOS-ra támaszkodva teszi könnyen kezelhetővé a gépet.

Az **AmigaDOS** - mint a **UNIX** egyik leszármazottja - az Amiga multitasking szervezésű operációs rendszere. A multitasking azt jelenti, hogy egy számítógépen egyidejűleg több feladat **(task)** futhat. Minden tasknak saját prioritása, elsőbbségi szintje van. Az egyes taskok külön ablakokat használhatnak a képernyőre való kivitelhez, és felosztják egymás között a memóriát. A multitasking miatt az Amigát alacsonyszintű nyelven - C-ben vagy Assemblyben - programozni nem olyan nehéz feladat, mint ahogy első ránézésre tűnik. Szabályok sorát kell ugyan betartani ahhoz, hogy a különböző programok ne zavarják egymást. Jó segítséget nyújt ehhez a **Kickstart-ROM** könyvtárainak használata.

Mielőtt továbbmennénk ejtsünk néhány szót a Kickstart ROM-ról. A **Kickstart ROM** tulajdonképpen nem más mint PC-n a **ROM-BIOS,** annyi különbséggel, hogy nemcsak a gép bekapcsolás utáni dolgokat intézi el, és nem csak megmondja, hogy melyik eszközt hogyan kell kezelni a gépnek, hanem nekünk programozóknak is tartalmaz egy-két dolgot. Tartalmaz egy sor olyan rutingyűjteményt, ami rettenetesen hasonlít ahhoz mintha szabvány C rutin könyvtárak lennének. Például alapvető függvény a **ReadfJ, OpenO. CloseO, WritePixelfJ** stb. Ezek ugyanúgy mintha megvettük volna pl. DorlandC-t (ami PC-n kedvelt C fordító) rendben sorakoznak könyvtárszerkezetben és a rendszer részei. Igaz, így a Kickstart ROM már eléri az IMb terjedelmet (lásd CD32). viszont biztosítja, hogy a gép működése szempontjából optimális programokat lehessen írni anélkül, hogy a programok egymást zavarnák, vagy a programozónak bele kellene merülni a hardver részletes ismeretébe. Mégis, viszonylag egyszerűen, gyorsan, igen látványos programokat lehet írni, azzal a nem titkolt előnnyel, hogy utána egy másik Amigán is pont úgy fog futni, sőt, nagyobb verziójú Kickstarton is.

A programozásáról csak annyit, hogy inkább idézek egy, az AMIGA megjelenésekor íródott újság cikkéből, ahol az újszülöttről kérdeztek szakmabeli embereket, s Bauman Gábor az Andrornéda Szoftverház fejlesztője a következő kijelentést tette: „Mint programozó szeretem is a gépet és haragszom is rá. Ez a gép már egy nagyon bonyolult rendszer. Nem lehet vele megcsinálni, ami más gépekkel a kedvelt módszerünk, hogy teljesen kipucoljuk és utána elölről az egész rendszeri bitenként felépítjük. Ez az Amigánál nem megy, meg kell tartanunk az alapszoftvert, és ennek csak a megismerése is több hónap lesz. De szerintem egy tiszta felépítésű rendszer, amit meg lehet kedvelni." (Ötlet 86 1986.05.22. 42.old)

A számítógépben állandóan külön task fut az RS232-es port, a billentyűzet, külön-külön minden lemezegységnek é& a **CLI** (Command Line Interface. az **AmigaDos** parancsértelmezője) kiszolgálására. Az én gépemen pl. Bootolás után 23 task fut! Az AMIGA rendszerszoftvere modulokból épül fel. Ezek egy része a Kickstart ROM-ban helyezkedik el, a többi pedig szükség szerint lemezről tölthető be. A 4. ábra az egyes modulok kapcsolatát szemlélteti. A rendszerszoftver legmagasabb szintje a **Workbench** és a **Command Line Interface (CLI).** amelyek a felhasználó által is „láthatók".

A **Workbench** az **Intuition** nevű modult a képernyőkezeléshez, az **AmigaDos-t** pedig a fájlkezeléshez használja. Az Intuition viszont az **InputDevice**-al a bemeneteket, a **Graphics** és **Layers** könyvtárrakkal pedig a kimeneteket éri el. Az AmigaDos tartja kézben a filekezelést. önmaga pedig az Exee-

re épül. amely a task-okért, a megszakításokért, a rendszerüzenetek továbbításáért, a B/K műveletekért és sok más funkcióért felelős. A **Trackdlsk Device** a legalacsonyabb szintű lemezkezelő interfész, ami az olvasófej mozgását és az írási/olvasási műveletet látja el. A Bili. és a **Gameport Device** kezeli a billentyűzetet, az egeret és a botkormányt, sorrendbe állítva a bemeneteken lezajlott eseményeket az **Input Device** nevű modul számára. Az **Audio Device** feladata a hanggenerálás, a **Serial** és **Paralell Device** feladata a portok közvetlen kezelése. Minden operációs rendszer egyik fő feladata, hogy a meglehetősen különböző B/K tevékenységet egy magasabb szinten egyesítse, megkönnyítve a programozók munkáját. Természetesen elkerülhetetlen, hogy bizonyos speciális funkciók el ne vesszenek. Az Amigán ezt a feladatot a **dos.library** tölti be. A dos.library több szempontból is speciális. Az egyik fontos jellemzője, hogy nagyon sok könyvtárral kapcsolatba lép, illetve léphet. Részben ez az oka annak, hogy a C programokban nem kell a DOS-rutinok használata előtt a könyvtárat megnyitni, mivel ezt a startup kód elvégzi. A DOS-t csak speciális taskokból lehet meghívni, melyeket a **folyamat (process)** elnevezéssel illetnek. Erre akkor kell figyelni, amikor több párhuzamosan futó részfeladatra bontjuk a programot - nagyobb programoknál elég gyakori megoldás ez. Egy **process** az **Exec** [ez az operációs rendszernek az a modulja amelyik a multitaszkot is vezérli) szempontjából semmiben sem különbözik egy normál taszktól. A DOS viszont számos egyéb információt kapcsol a task struktúrához. A process-struktúra olyan információkat tartalmaz, mint például az aktuális directory. ami egy közönséges task-nál fölösleges. Ha a **CLI** vagy a **Workbench** alatt egy programot behívunk, a program élvezheti a process-ek számára biztosított minden „kényelmet". Különbség mégis van: a **CLI** nem kreál új folyamatot, egyszerűen saját processzében a vezérlést adja „kölcsön" a programnak. így önmaga leáll és nem értelmez parancsokat. A teljesség kedvéért megemlítem, hogy a CLI a vermet sem adja át, hanem készít egyet a STACK felhasználói parancsban beállított méretűt a programunknak. A **CLI** ablakába mégis gépelhetünk, ennek az az oka, hogy a **Console Device** saját taskban fut.

CLI-ben új process-t a **RUN.** a **NEWCLI** és a **NEWSHELL** parancsokkal indíthatunk. A Workbench alatt ezzel szemben minden behívott program önálló process-ként fut. A Dos minden Device-hoz is rendel egy process-t. amely a standard B/K parancsokat speciális B/K parancsokká alakltja. Technikailag így van megoldva az egységes elérés. Ennek a belső komunikációs rendszernek a leírása külön könyvet is megtöltene. de a programozók túlnyomó többségének erre semmi szüksége. így csak a legfelső kapcsolódási felülettel foglakozunk. Ha az olvasónak az előbbiek bonyolultnak tűnnek, ne keseredjen el - nyugodtan induljon ki abból, hogy a multitasking-gal remekül elszórakozik maga az operációs rendszer. A későbbiekben előfordulhat, hogy érdekesnek fogunk találni egy-két adattípust amiket a rendszer dokumentációkban találhatunk, amelyek a **BCPL** nyelvből származnak, mely a C elődje volt. Az történt ugyanis, hogy a Commodore félkészen vásárolta meg a Dos-t. amelyet még nem C-ben írtak. Például a BPTR egy BCPL .^..i^t,-, imeivpt eny normál C mutatóból 4-el való osztással kapunk.

Mielőtt áttérnénk a fordítók ismertetésére, racgcminc™nh *><*.&„„ b é - kességet amik általában kimaradnak az olyan művekből ahol csak az oprendszer használatát ismertetik.

1.1.4 A rendszer debugger

Sokan nem is tudják, hogy már az első Amiga is tartalmazott egy debug rendszert, melyet az **Exec.Library Debug** funkciója indított el. Az 1.3-as Kickstart-nál ezt meghívva a soros portra kötött másik számítógépen keresztül lehetett a debugger-rel kommunikálni (null-modefn kábel segítségével).

A másik számítógépen csak egy egyszerű terminál programot kellett elindítani 9600 baud sebességgel (akár egy PC a Norton Terminál funkciójával) és már készen is volt a rendszer. Ez a 3.0-ás rendszeren kicsit átalakult, a másik gépnek is egy Amigának kell lennie. A debug rendszert a Workbench Debug menüjéből a **ROMWack** opciót választva lehet elindítani (a loadwb parancsot -DEBUG opcióval kell elindítani). Ekkor a gép látszólag leiagy, de valójában a soros porton megpróbál kapcsolatba lépni a másik Amigával. A másik gépen egy ROMWack nevű programnak kell futnia, és ettől kezdve már lehet is az Amiga memóriájában turkálni.

1.1.5 A SetPatch

A **SetPatch** program az, amivel a Kickstarf-ban levő, a fejlesztők által ejtett „bug-okat" tudjuk kiküszöbölni. A könyv írásakor a legújabb SetPatch 40.16-os verziószámmal van ellátva, ez a CD32 hibáit is javítja (mert hogy van mit javítani). Persze ez a javítás csak a következő reset-ig él. utána újra el kell indítanunk, hogy minden jól működjön.

1.1.6 Kickstart-ba ágyazott üzenetek, avagy a mókás fejlesztők

Az Amiga fejlesztői bizonyos üzenetekel hagytak az utókor számára, amikor a Kickstart-ot fejlesztették. A AlOOO-esen beérték annyival, hogy a gép dobozát, az összes fejlesztő a kézjegyével látta el. Azonban a többi gépnél ez máshogyan alakult. Az 1.2-es és 1.3-as Kickstart-al rendelkező gépeken ezt az üzenetet a Workbench képernyőn az alábbi módon lehet elővarázsolni úgy. hogy ezeket egyszerre csináljuk: - Ctrl+I.Shift+LAlt+RShift+RAH gombok együttes lenyomása - egy funkcióbillentyű lenyomásával - a lemezegységbe egy lemez berakása/kivétele

Ehhez a szép kis procedúrához sajnos két kéz nem elég. a készítők gondoltak arra is. hogy azért olyan egyszerűen ne lehessen ezeket előhívni.

A 3.0-ás Kickstart-al rendelkező gépek esetén egy picit módosítottak az előbbi eljáráson, valahogy így néz ki: - Ctrl+LShift+LAlt+RShift+RAlt gombok együttes lenyomása - eközben 12 db About requester megnyitása a Workbench-en A 12. ablak felé haladva egy picit lassulni fog már a gép, de ne adjuk fel! Ha nem jelenne meg, akkor nyissunk még néhányat, lehet hogy elszámoltuk...

1.2 SAS/C

1.2.1 A C Editor

Az Amigán elég sok programozási nyelv közül lehet választani, pl.: **pascal, modula, oberon, C, assembly, Basic,** valamint ezeknek a nyelveknek a keverékei, ún. hibrid nyelvek pl.: **Amos** (Basic alapú), E (C és pascal).

Annak, hogy mégis a C-t választottuk két oka van. Az egyik az, hogy az Amiga operációs rendszerét is e nyelvben alkották meg, valamint assemblerben (ezt is bemutatjuk). A másik ok. hogy a nyelvek közül a C tűnik a leghatékonyabb nyelvnek, persze nem csak Amigán. Rendelkezik a legtöbb algoritmikus nyelveknél megszokott döntő tulajdonságokkal (strukturáltság, könnyen kezelhető adatszerkezetek stb.), és kifinomult fordítóval rendkívül megközelíti az assembly sebességét. A hibrid programozással pedig a kritikus pontoknál betoldhatunk assembly részleteket. Magával a C nyelvvel itt nem foglalkozunk, ezt helyettünk már megtették mások, valószínűleg nagyobb sikerrel (magyarul megjelent C könyvek közül néhány, amiből el lehet sajátítani a C nyelv sajátosságait az alábbiak: Ü.W. Kerningham - D.M. Richie: A C programozási nyelv; Benkő Tiborné - Benkő László - Tóth Bertalan: Programozzunk C nyelven! stb.).

Ebben a könyvben nem kívánunk foglalkozni ill. részletesen tárgyalni sem az editort sem mást mivel ezt a feladatot a programmal adott dokumentációk hivatottak ellátni.

Az Amigán több C fordító is létezik, de talán a legáltalánosabban használható a SAS Instilute, Inc. Cary, NC (1992-1995) cég implementációja.

Régebben volt egy másik fordító is ami igen népszerű volt az Amigások körében, az **Aztec** C fordítója, amely Assemblyre fordított első menetben, de erről a fordítóról mostanában nem hallani, s tudtommal 3.0-ás rendszer alá nem is készült el.

A SAS/C fordítója képes a **C++** programok fordítására is (ebben az esetben a forrásnak .cxx-re. vagy .cc-re, ill. .cpp-ra kell végződnie).

A SAS/C tulajdonképpen utódjának tekinthető a **Lattice C** compiler-nek, amit a Commodore cég is támogatott (az op. rendszer részeinél ezt használták).

Amigán a C programozás nem bonyolultabb feladat mint bármilyen más gépen, a hírekkel ellentétben, főleg ha nem használjuk ki az Amiga lehetőségeit.

Tehát ha szabványos könyvtár rutinokat használunk észre sem fogjuk venni, hogy a programunk multitaszkos környezetben fut. Az igaz, hogy nem is fog úgy kinézni, mint ahogy egy Amigás programtól joggal elvárnánk.

Amigán az igazi gond azonban nem az, hogy hogyan csináljuk, hanem az. hogy mit csináljunk - aranyszabály az Amiga programozásában. Szinte nincs olyan probléma, amelyre ne jutna eszébe az embernek két-három kivitelezhető, praktikus megoldás a lehetőségek dzsungelében. Ez a jellemző a program szerkezetben is érvényesül; ha egy tipikus Amiga program forráslistáját megnézzük - az első ijedtség után - megállapíthatjuk, hogy a program túlnyomó része adatstruktúrákat inicializál, include fájlokat szerkeszt be. Az

B/K tevékenység, amelynek elfogadható kivitelezése más rendszerekben sok munkával jár. általában nagyrészt azt jelenti, hogy a program átpasszolja a struktúrákat, helyesebben azok rímeit az operációs rendszernek.

Általában amikor elkezdünk egy gépel programozni, az első nehézségek még akkor jelentkeznek, miközben megpróbálunk futtatható programot csinálni, ezek után általában kiderül, hogy nem működik a program és kezdhetjük elölről.

Ezt nagyban megkönnyíti a SAS/C editorába integrált fordítási lehetőség és más egyéb lehetőségek is. de mielőtt ezt megtárgyalnánk nézzük meg. hogyan is terpeszkedik el wincsinken maga a SAS/C. Természetesen lehetőség van lemezre installálni is. de ez nem változtat a most következőkön. Lemezről azonban iszonyú lassú szerintem. A fordító minimális hardver követelménye 1 Mb memória és legalább 2 Floppy meghajtó.

Persze nem árt egy winchester, és 2 Mb szabad memória. Az alap A1200-esen egy komolyabb program fordítása már nehézkes a memória miatt. De kisebb programoknál ez a veszély nem áll fenn. Ezen segíthetünk kisebb trükkökkel (pl. A Wb-t 2 színűre állítjuk, kilépünk az se-ből. és CLI-bőI fordítunk). A SAS/C installálás után majd 10 Mbyte-ot foglal el a szabad területeinkből. Az installálás után lépjünk be az se: nevű könyvtárba.

Itt a következőket fogjuk látni:

```
c (dír)
icons (dir)
enamples (dir)
source (dir)
hełp (dir)
extrás (dir)
starter_project (dir)
REHD.ME
Read.Me.6.55
```

Miután elolvastuk a két read me fájlt kattintsunk rá a c fiókra. Ez tartalmazza a tulajdonképpeni **compiler-t.** a **debugger-t.** és sok egyéb hasznos segédprogramot. A **cpr** a SAS/C debugger-e. az **scsetup** tartalmazza a beállításokat, az **se** a SAS editor, az **SMFind** egy keresőprogram amely egy string-et keres a megadott útvonalon lévő fájlokban. Most vegyük szemügyre azokat a programokat amiknek nincs ikonjuk. Többnyire a nevük elárulja, hogy mire is használhatóak, ezért külön nem tárgyaljuk mindet, csak néhányat. Az **se** maga a fordító, amellyel CLI-ből indítva mi is fordíthatunk (néha szükségessé válhat ilyesmi ha kevés a memóriánk). A fordító automatikusan meghívja a szükséges programokat, a C vagy assembly fordítót és a **linkért.** Mivel a SAS/C elődje a **Latice C** volt. így kellet gondolniuk a lefelé való kompatibilitásra is. Ezért található egy **sc5.** amely a parancsokat és az opciókat átalakítja a Latice C 5.xx-ről az új fordítónak, és meghívja azt a fordítás elvégzésére. Az **letose** viszont csak kiírja az új opciókat. Mind a kettő figyelembe veszi az **scopts-ot,** ha az létezik. Az **scopts** meghívásával variálhatunk a C fordítás menetén, később részletesen meg fogjuk nézni. Az **síink** a **linker** amely szintén CLI-ből indítható és felparaméterezhelő A ^st és a

hypergsl-vel a RAM rezidens **GST** szimbólumok megtekinthetőek. A **GST** (Global Symbol Table) egy nagyon has/nos dolog. A régi fordító header fájljait van hivatva felváltani. Ezekkel a szimbólum táblákkal gyorsabbá tehetők a fordítások, mert ha a szabad memória mérete lehetővé teszi, akkor két fordítás között is a RAM-ban maradnak.

Az icons könyvtárban azokat az ikonokat találjuk meg amelyeket az editor ad, lll. adhat az állományoknak.

Az examples könyvtárban olyan példaprogramokat láthatunk amelyek segítenek, például hogyan írunk device-t, library-t.

A sources könyvtárban is igen hasznos dolgokat találhatunk, ha van türelmünk őket átböngészni akkor mindenképp tegyük meg. Az itt található forrásokat a fordító a programunkba fogja beágyazni, ezek gondoskodnak a programunk megfelelő indításáról. A beállításoknál a Linker Options részben erre bővebben kitérünk.

A help könyvtárban igen hasznos leírások találhatók, mind az editorról (se.guíde) mind a szabványos és csak a SAS/C-re jellemző könyvtármüveletekről **(scjib.guide),** valamint a hibaüzenetek részletes leírása **(scmsg.guide)** is itt van. A fordítóról olvashatunk az **sc.guide-ban.** ha valami problémánk van a fordítással nyugodtan nézzük meg a **sc_prob.guide-ban.** Az sc.util.guide-ban részletes leírást kapunk arról, hogy mik is találhatók a c könyvtárban és hogyan kell azokat használni (pl. forrás szintű debugger amely igen hasznos dolog, de majd később mi is foglalkozunk vele).

Az extras-ban szintén hasznos dolgokat találhatunk ha a hibakeresés mélyére akarunk ásni, bár szerintem aki idáig jut, inkább át kéne gondolnia, hogy nem lenne-e jobb máshogyan hozzáfogni a programhoz. Ha már azt kell vizsgálnia, hogy a lefoglalt területről kiszaladt-e a program, többnyire igen nagy gondok vannak.

Végül a starter_project. amiben csak ikonok vannak abból a célból, hogy ezeket az ikonokat másoljuk át saját project-jeinkhez. hogy ne kelljen ide visszatérni ezek használatáért. Egyébként is igen előnyös, hogy legalább az SCoptions-t másoljuk át, mert ha megváltoztatunk egy' opciói az egyik programnak nem biztos, hogy azt egy másik program is használni fogja, vagy esetleg miatta nem is fordít. Nálam például a C forrásoknak külön könyvtára van és ebben is minden leendő, vagy már létező programnak külön könyvtára, ahol ezek az ikonok megtalálhatók a programok forrási és egyéb hozzájuk tartozó dolgok mellet. Egyébként a könyvhöz kapott lemez is hasonló szervezésű.

A nem látható könyvtárak között van egy igen érdekes, az include nevű. amelyben azok a szabvány és Amigás inciude-ok találhatók amiket a program elején be fogunk szerkeszteni. Ha a fordító hibát jelez, olyan hibákért mint az. ha egy struktúra olyan elemére hivatkoztunk amely nem létezne, valószínűleg elírás következett be. Ha itt vagy a preferences-ben megnézzük gyorsan kijavíthatjuk a hibát. Azonkívül érdemes itt is bogarászni, elég sok dologra rá lehet jönni a rendszerről pusztán az include-ok átnézésével. Természetesen nem pótolja a ROM kernel manual-t. és egyéb az Amigával foglalkozó részletes leírást, amelyek ezeket részletesen tárgyalják.

Nos. akkor lássuk magát az editort aminek SE a neve. Az editor nem a legkényelmesebbek egyike, de egy pár száz soros program megírása után mar, megS^ohjuK, llf» nem mindjnrt. egy másik editorral kpydtünk neki. Eh-

hez a megoldáshoz azonban **Arexx** kell ami legalább 100 Kb memóriái jeleni, viszont jolob editorhoz jutunk. De maradjunk egyenlőre ennél az editornál. Kinézetre rögtön szembetűnik az, hogy az ablak alján húzódik egy csík. amelyen a következő feliratok olvashatók:

LINE: ami a kurzor helyének sorát mutatja
• COL: ami a kurzor helyének oszlopát mutatja
FILE: a szerkesztett állomány nevét
{} között; azt mutatja, hogy a szerkesztőben lévő állomány hányas számú a szerkesztett állományok közül.

Az alatta található sorokban az editor itt közli ill. itt várja az információinkat. Például ha keresni akarunk a szövegben, ide kell beírni a keresendő stringet (szót).

A menük:

Project menüben először a **Save&Close** menüvel találkozunk amely annyit jelent, hogy a forrás kimentése után kilép az editorból. Mielőtt tovább mennénk meg kell említenem, hogy csak az éppen szerkesztett ablakból lép ki. Ha összesen egy volt csak szerkesztve, akkor lép ki az editorból. Elmentés nélkül bezárja a szerkesztő ablakot a **CloseWindow** menü választásakor.

A **Save&Contlnue-ra** kimenti a forrást és lehet folytatni a szerkesztést.

A **Save&ReOpen-re** kimenti a forrást, majd berak egy requester-t amelyben az imént kimentett forrás neve áll. amit az OK megnyomására újra visszatölt.

Az **Open New Flle-ra** hajlandó betölteni az editorba egy új fájlt szerkesztésre.

A **Rename** menüre átnevezhetjük az éppen aktuális forrásunk nevét.

Az **InsertFile** menüre a kurzor pozíciójától beszúrja a kiválasztott fájlt.

A **Display Names** esetén megmutatja az editorba betöltött fájlok neveit, és azt, hogy melyik lapon található a 9 közül, majd egy billentyű lenyomásra vár, hogy visszatérjen az editorba.

Az utána következő menükkel ki ill. vissza lehet menteni ill. tölteni a makró fájt. **(Savé Macro, Load Macro).**

Az Undo Last Change menüponttal vissza lehet állítani az utolsó változtatás előtti állapotot.

A **Help** azt hiszem, hogy egyértelmű (a help könyvtárból az se.guide-of tölti be).

Az **Exit** SE-re kilép az editorból.

A második menü a **Block** menü, aminek az almenü jelentései a következők. Blokkot az egérrel tudunk kijelölni, úgy hogy a blokk kezdetének szánt részen az egér bal szemét kinyomva, 's azt nyomva tartva húzzuk a végéig, majd elengedjük. A kijelölt blokk inverzbe vált.

Tehát a menük:

Copy ami a kurzor pozíciójához másolja a blokkot.
Delete letörli a blokkot,
Move a kurzorhoz mozgatja a blokkot. Az előző helyen hagy egy üres sort.
Print kinyomtatja.

Read betölt egy blokkot, amit a kurzor pozíciójához illeszt.
Write kiírja a blokkot lemezre.
Beginning a blokk elejére ugrik.
End a blokk végére ugrik.
Input From Clip betölt egy blokkot a **Clipboard-ról.**
Output To Clip kimenti a blokkot a **Clipboard-ra.**
Például így lehet blokkot másolni a CygnusED-röl közvetlenül a ÖAS editorába. Mert a CED mikor blokkot vág (CUT) a **Clipboard-ra** másolja azt.

A **Windows** menü a szerkesztő ablakokra vonatkozik.
Open Window egy új szerkesztő ablak nyitása, és abba egy fájl betöltése.
Toggle Display Size az új ablak alapesetben csak a másik feléig ér. és ezzel a menüvel lehet egész ablakossá tenni a szerkesztését, ill. visszaváltani.
Switch Windows-al a szerkesztendő ablakok között lehet lépkedni. Ezt a funkciót egyébként ellátja a numerikus kínpadon lévő 5 azaz ötös billentyű.
Create New CLI nyit egy új CU ablakot.
Interlace Toggle az editort interlace-ba kapcsolja, vagy onnan vissza.

Ezek után a **Search** menü következik - a keresést mindig a kurzortól kezdi a fájl végéig!
Line Number azt hiszem egyértelmű, hogy a keresendő sor számát várja.
String Search String keresés a szövegben.
Search and Replace megkeresi a stringet majd visszakérdez, hogy cserélje-e vagy sem. Mielőtt kicserélné megkérdezi, hogy ki akarod-e cserélni vagy meggondoltad magad, és ha mégis ki akkor az összes előfordulásnál is megtegye-e a cserét.
Search Again ami tovább keresi a szövegben a keresendőt.

Végül az utolsó menü az **Options.**
Configuration Screen az editor összes beállítása itt állítható. Ha rákattintunk, akkor egy új ablakot nyit. amelyben látható az egész billentyűzet.
A billentyűkhöz rendelt funkciók egy egér kattintással itt változtathatóak. A billentyűzet alatti részen három szöveg található: a **key name** amely a lenyomott bili. neve. mellette a billentyű **Raw** Code-ja látható, valamint a **Quelifier** kódja. Mindezek alatt helyezkedik el egy nagyméretű, szöveg beírására szolgáló ún. string gadget. amelybe a kiválasztott billentyűkhöz rendelést beállíthatjuk, ill. láthatjuk a már beállítottál. A string gadget alatt két sort találunk, amelyből a felsőn a tabulátor pozíciókat lehet beállítani. A C-ben használatos "{}" kapcsos zárójelek helyzetét lehet állítani. Az alattuk található bizgentyúkön (nevezzük az egyszerűség kedvéért gadget-eknek) a következőket lehet kicsikarni a programból:

SAVÉ elmenti a beállított értékeket és visszatér a szerkesztőbe.
USE használni fogja azokat és visszatér a szerkesztőbe.
CANCEL magyarul szénszál, mindent változatlanul hagy és visszatér.
CLBAR TABS Lordi ci me\r bcallllolt. tabulóu-r *i-t*.w,-it a
REPEAT TABS visszaállítja azokat.

A fennmaradókkal azt lehet kipipálni, hogy a jobb és a bal ALT. illetve SHIFP külön számítson-e avagy sem. De aki igazán s/ét akarja kavarni annak a menüben is tanácsos szétnéznie. Tehát következzéka menük leírása a teljesség igénye nélkül:

Project-ben a szokásos opciók, úgy mint **Open,** betöltése egy eoníig-nak. **Savé as** kimentése más néven (tehát nem ez lesz az alap beállítás!). **Quit** visszalép az editorba.

Options 1

Input Processing bemeneti művelet
No Process nem csinál semmit
Tab expansion method a tabulátort hogyan használja.
Use TAB character tabulátor karaktert használjon.
Expand Tabs to spaces szóköz karakterrel feltölti a tab helyeket.
Backup File Mode n biztonsági másolatról
No Backup File ne legyen biztonsági mentés ill. fájl.
Place Backup in Backup dir ha már van. legyen a backup könyvtárban.
Rename Backup with Backup Ext az aktuális könyvtárba teszi, és Backup kierjesztést ad neki.
Search Method keresési metódus
Use Regular Expression csökentett kifejezés szerint.
String Matching egy az egyben, úgy ahogyan beírtuk neki.
Collomun Display az editor mutassa-e, hogy melyik oszlopban vagyunk
Prompt before Undo visszakérdezzen-e mielőtt visszaállítunk valamit
Case Insensitive Search kereséskor különbözzenek-e a nagy betűk a kicsiktől avagy nem.
Colorize C **files** színesben mutassa-e meg a C forrásokat az editor
Font selector ... itt beállíthatjuk, hogy milyen betűvel akarunk írni az editorunkban.
Colorized prefs ... itt állíthatjuk be a C forrásunk színeií
Keyl ... **Key3** azokat a rövidítéseket tartalmazzák és rakják a string gadget-be, amelyeket a billentyűkhöz rendelhetünk.

De visszatérve az editorhoz, és úgy használva mint azt elvárnánk, rá fogunk jönni, hogy elég jól elboldogulunk már vele. Főképpen akkor vesszük ezt észre, ha olyan hatalmas programot írunk mint a mi első programunk.

Miután megírtuk a programunkat, felmerülhet a kérdés, hogy miként is lehet ezt lefordítani? Nos. a válasz igen egyszerű, meg kell nyomni az F4-es billentyűt. Ha minden jól volt beállítva akkor Workbench-ből, a forrás könyvtárában leitűnik egy futtatható fájlt jelentő ikon (pl. a francia kulcsos ábra). Erre rákattintva (click-elve) ha szerencsénk van rögtón fut a program. Ám. ha nincsen minden jól beállítva akkor többnyire mar futtatható fájlt sem kapunk. De mindezeket be tudjuk állítani ha a Ctrl-F4-et használjuk.

Workbench alól az sc:starter_project-ben találunk egy SCOPTIONS nevű ikont, ami pont ugyanezt fogja eredményezni. Ezért célszerű ezt az ikont lemásolni a fejlesztendő programunk könyvtárába.

Nos nézzük mit is lehet itt állítani. Az ablakban először a **COMPILER OPTIONS...** gadget-tel fogunk találkozni, amit aktivizálva egy újabb ablakban beállíthatjuk a fordító paramétereit, éspedig az alábbiak szerint: (ha elállítunk valamit, vagy ami a default-tól eltérő azt a szín változásával észrevehetővé teszi számunkra).

NoDebug a debug információk beszerkesztése, hogy a programot lehessen debuggolni a SAS/C debug-jával. A kívánt mélység állítható.

NoShortlntegers az **Integer** típus alapesetben 32 bites, vagy 16 bites legyen.

NoUnsignedChar a Char változó előjeles avagy előjel nélküli legyen-e? (-128.. 127. vagy 0-255) Természetesen ezek csak a default-ot állítják, a forrásunkban ezeket máshogyan is definiálhatjuk (pl.: short int biztosan 16 bites, unsigned char biztosan előjel nélkülinek fogja venni stb).

Multiplelnclud.es az include-okat a forrás tartalmazza, vagy a jobb oldalon található ún. listwiev (listanéző) gadget-ben megadható legyen.

NoGSTImmediate használjon-e include-ok helyett GlobalSymbolTable-t avagy nem.

Ha használjon, akkor a GST string gadget-ben kell ennek nevét megadni. Ez lehet az inciude:all.gst fájl. Ezt a SAS/C installáláskor készíti el nekünk, ha kérjük. Használatával felgyorsul a fordítás, mivel a RAM-ban tárolja az eddig header fájlban megadott szimbólumokat, persze ha van elég RAM. és a programjainkból eltűnhetnek az **#include< proto/lntuition.h** > sorok, így maga a forrásunk is rövidebb lesz ezáltal.

Icons használjon-e ikonokat a fordítás során (pl. a kész programnak).

NoPreProcessOnly ha átállítjuk nem fog fordítani csak az előíeldolgozót engedi rá a forrásunkra.

NoCXXOnly ha átállítjuk, akkor csak C++ forrást fog várni a fordító. .

MemSize=Large választhatunk a memória modellek közül **(Hugh = extranagy, Large = nagy, Tiny = pici, Small = kicsi, Médium = közepes).**

WarnVoidReturn szóljon ha egy függvény nem ad vissza semmit, ill. void-ot ad vissza.

A **GST** string gadget-be lehet megadni a **GST** fájl nevét. Alatta azt a nevet lehet megadni, hogy milyen néven hozzon létre GST fájlt. A név megadásakor a forrást átalakítja GST állománnyá. így ne legyen benne más mint #include típusú sorok vagy egyéb a GST-ben megszokott egyebek. A Define nevű listview gadget-ben "#define" szerűen lehet definiálni szimbólumokat.

Az opciók közül a következő a **MESSAGE OPTIONS...,** ahol a hibajelzéseket szabályozhatjuk az alábbiak szerint: **NoAnsi** ha átállítjuk, akkor csak azt nem szólja meg ami a szabványos **ANSI** C-ben megállja a helyét (nagyon hasznos ha olyan kódot akarunk írni ami hordozható, tehát másik gépen is csak fordítani kell és fut, pl. PC-n).

NoErrorRexx a hibáról értesíthet más külső programot **ARexx-en** keresztül.

ErrorConsole a hibáknak nyit egy konzol ablakot.

KrrorListing a hibákat felsorolja, nem csak a tényét kozh.

ErrorSource lehetőséget ad. hogy az észleli hibát megmutassa a forrásban is. Ha rákattintunk a hibára a hibakonzolon akkor a szerkesztőbe oda ugrik a kurzor a hibás sorra.

OnError=Stop Ha hibát észlel megáll és nem linkeli a hibás kódot.

Az ablakban található gadget-ekről ejtsünk néhány szót. Van kettő ami igen hasznos, mégpedig a **MaxError** string gadget és a **MaxWarn.** Mind a kettőben a maximális hibák számát **1ll** figyelmeztetések számát állíthatjuk be. így nem fordulhat az elő. hogy egy lehagyott pontosvessző miatt kilométeres hiba lajstromot írjon ki, holott csak egyetlen hiba volt, amit persze göngyölített.

A Következő a **CODE OPTIONS...,** amellyel a létrehozandó kód mikéntjeit lehet állítani. Itt állíthatjuk be. hogyan is nézzen ki a kész kód.

NoMath ahol a használni kívánt matematikai rendszert állíthatjuk be.

Math=STANDARD ha a szabványos C-beli kódot akarunk. Ha az IEEE szabvány szerinti eljárást akarunk, akkor a **Math=IEEE-t** válasszuk, ez a Commodore IEEE.lib-jén keresztül történik; valamint ha az FFP.lib-en keresztül akarjuk, akkor válasszuk a **Math=FFP-t.** Ha közvetlenül matematikai koprocesszorra akarjuk optimalizálni akkor a Math=68881-et válasszuk.

Figyelem! Ez utóbbi kód csak olyan gépeken fog futni amiben van FPU! **CPU=ANY** ez azt jelenti, hogy programunk 68000 kódban készül, tehát minden Amigán futni fog. De ahhoz, hogy kihasználjuk nagyobb teljesítményű processzorok teljesítményét nyugodtan használjuk a nagyobb processzorokat.

Például ha olyan programot írunk aminek A1200-ason kell futnia, használhatjuk a 68020 opciót stb. De vegyük figyelembe azt is. hogy akár A5OO-ason is lehet 3.0-ás rendszer! Tehát ha hordozható programot akarunk írni azt **CPU=ANY** állásban tegyük. Jó megoldás, ha kész programunk két vagy több processzorra is optimalizálva van. így nem kényszerítjük rá szegény user-t. hogy egy 68040 turbó kártyán (vagy A4000-ese mellet) egy 68000 alapú kóddal szuttyogjon.

Parms=STACK azt jelenti, hogy a függvényeknek a paramétereit miben adja át meghívásukkor. A verembe (ez a szabványos, előfordul hogy ha szabványos könyvtárakkal dolgozván átállítjuk, egy GURU-val fogja honorálni), regiszterbe (akkor célszerű használni ha a C forrásunkhoz Assembly-ben megírt rutinokat fűzünk), vagy mindkettőt használhatjuk.

StackCheck alapesetben a veremről mindig készít ellenőrző kódot, hogy az esetleges hibákra ez rámutasson. Ha kikapcsoljuk sok esetben gyorsabban futó kódot kapunk. Használta akkor célszerű, ha már belőttük a kódunkat.

A **LIST/XREF OPTIONS...**-ban beállított értékek csak akkor hatásosak, ha az ablakban beállítottuk az alapesetben **NoList** vagy **NoXref** gadget-et arra, hogy valamelyiket használja. A List egy olyan nyomtatóra illesztett fájlt takar, amiben a forrás mellé ki van listázva a kapcsos zárójelek használata (block néven). Az Xref szabványos keresztreferencia a szimbólumokról. A megnyíló ablakban ezeket lehet finomítani, hogy a Lista ill. **Xref** mire is vonatkozzon és mit tartalmazzon.

Az **OPTIMIZER OPTIONS...** azt a célt szolgálja, hogy a kódot valamilyen irányvonal mentén lehet optimalizálni, például futási időre, hogy gyorsan fusson az adott gépen. Ez nem azt jelenti, hogy nagyobb processzorra fordít, hanem esetleg egy néhány változót pl. regiszterekben tárol, és nem egy memóriacímen stb. Vagy például méretre optimalizálhatunk, tehát hogy a fordított kód kisebb helyet foglaljon stb.

NoOptimize legyen-e oplimalizáció. rsak akkor lesznek figyelembe véve a fordítás során ha ez „optiniize" állásban van! OpUrnizeGlobal A globális változók optimalizálása.

OptimizePeep optimalizálás úgy, hogy a processzor feldolgozási képessége nőjön csak az utasítások sorrendjét változtatva. *

OptimizeSchedule optimalizálás listába rendezés útján, jó eredmény érhető el 68040, valamint 68882-es használatánál.

OptlnLine optimalizálás egy soron belül az olyan függvények esetében amelyek a **inline** kulcsszavat tartalmazzák.

NoOptlnLocal a lokális változók optimalizációja.

OptLoop olyan ciklusok optimalizációja amelyeknél lehetőség van a ciklus minden tagjának processzoron belül tartására, s így gyakorlatilag a processzor 0 buszciklus alatt végzi el a ciklust.

NoOptSize a kód nagyságára való optimalizálás.

NoOptTime a futási időre való optimalizálás, ha ez aktív akkor nem lehetséges a kód méretére is optimalizálni, és viszont.

OptDepth az optimalizáció mélysége állítható be, a default a 0

A PROTOTYPE OPTIONS... azt mondja meg, hogy a fordítás során generáljon-e prototípust a függvényeinknek, változóinknak stb. Ezt az alábbi formában tehetjük meg.

NoGenProto készüljön vagy nem prototípus deklaráció.

GenProtoExtern a készülő prototípusok extern előtagot kapnak.

NoGenProtoStatic a Static osztályú változókról ne készüljón.

NoGenParam a paramétereket nem veszi bele.

GenProtoTypedef a típus definíciókról is készüljön.

GenProtoDataltem a külső változókról is készüljön prototípus deklaráció.

GenProtoFile ebben a stnng gadget-ben lehetne megadni a készülendő prototípus fájl nevét. Mondom lehetne, mert nekem nem veszi figyelembe. Minden esetben a forrás fájlnevét veszi alapul, csak a kiterjesztése lesz .h. Egyébként a fordítás során ezt készíti el először, és így lehetőség van arra a megoldásra, hogy az így készített .h-t rögtón be is szerkesztessük a fordítóval. Csak egy **#include "forrás fájl neue.h"** sorra van szükség.

A LINKER OPTIONS... gadget-et aktiválva bejutunk a linker opciók közé, ahol igen sok mindent tehetünk. Ha *ÍXÍ* imént említett gadget alatt nem állítjuk át a gadget-et **Link-re,** nem fogunk futtatható kódot kapni, mivel az object fajiunkból nem lesz aki futtathatót linkel, hacsak nem csináljuk meg kézzel! A kézi linkelésről majd később beszélünk, mivel szükség lehet rá ha nincs sok RAM-unk. Tehát lássuk; **NoSmollCode** a kód szegmens kicsi legyen-e? **NoSmollData** az adat szegmens kicsi legyen-e, avagy nagy? NoAddSym Ne adjon hozzá egyéb szimbólumokat.

NoStripDebug hozzá szerkessze-e a/ összes Debugger szimbólumot a kódhoz.

ChkAbort a programunkból ki lehessen lépni Ctrl+C-vel. vagy ne. **NoBatch** ha linkelés közben nem definiált szimbólumot talál a linker megajánlja-e annak kézi definícióját. **Startup=c** a linker honnan vegye az alap objecteket a szerkesztéskor.

oonh. rr«si-s5'---=-aw "" * h e l «* sMíriuk fijtvelembe kell vennünk néhány apróságot. Nevezetesen, hogy nekünk kell megírni a programunk azt a reszel amely pl. a Workbench. vagy CLl környezetből való futáshoz szükséges. A de-

fault beállításban ezeket az előre megírt dolgokat az sc:source könyvtárban találjuk. A linker is itt keresi. Tehát elég halott ötlet ezeket letörölni, hacsak mi már nem írtunk jobbat. Itt megadható több opció is, attól függően, hogy milyen kódot akarunk. Ilyen pl. a **Startup=cres,** amikor is a kódunk elindítása után rezidensé lesz. vagy a **Startup=cback,** amikor kódunkat úgy inicializálja a fordító, hogy háttérben való futásra teszi alkalmassá. Vagy a **Startup=libinitr** esetén, ahol a kódból library-t csinál. Egyébként az itt beállítottak a lib: könyvtárban megtalálhatóaknak kell lenniük, mivel azok beszerkesztését jelenti. A default beállítása a startup=c pl. a c.o fájlra utal.

A linkelés, és úgy általában a programozás során szükségünk lehet arra. hogy egy-két rutint amit már megírtunk, hozzá szeretnénk linkelni a kódunkhoz. Ezek lehetnek természetesen saját magunk, vagy mások által megírt rutin könyvtárak is. A használata igen egyszerű: a forrásban meg kell adnunk a használandó függvény prototípus deklarációja mellé az extern opciót, valamint a linker-nek vagy a Lib nevét, vagy az object nevét. Ezt a **Libraries/Objects** listview gadget-ben kell megadnunk. Az alatta lévő string gadget-ben a linker-nek szánt egyéb kínzó módokat adhatjuk meg. pl. az overlay technika gyönyöreit részletezhetjük (lásd. 1.2.2.2-es pontja a könyvnek).

Azokat az opciókat amiket nem írtunk le. azt azért tettük mert nem tartottuk ahhoz fontosnak, hogy kezdésként részletezzük. De aki kíváncsi, annak ajánlom áttanulmányozni a help-et. valamint a SAS/C kézikönyvét. Ezt részletesen fogjuk tárgyalni a következő kötetben.

Azon az egyszerű fordítási és linkelési meneten, ami ezek használatával adódik, könnyen bonyolít egy kis RAM probléma, nevezetesen annak csekély volta. Bár ennek hátránya máshol is jelentkezik. Azonban ha már megpróbáltunk mindent, lehúztuk a Workbench-et két színűre, és Startup-Sequence nélkül boot-oltunk. de még mindig nem linkel, akkor vágjunk bele a kézi linkelésbe. Rájövünk, hogy nem is olyan félelmetes. Szóval nyissunk egy Shell ablakot (ez lehet akár KingCon is, vagy CLI). A linker fájlneve **Síink.** Az Síink meghívásának formáját megnézhetjük, ha beírjuk azt. hogy **Síink ?.** Az argumentumoknál, ahol több fájl is megadható, ez egyes fájlneveket plusz jellel, vesszővel vagy szóközzel válasszuk el.

A paraméterek jelentése (a teljesség igénye nélkül):

FROM a felhasználandó tárgykódok. A felhasznált tárgykódok melleit meg kell adni a lib:c.o fájlt is.

TO A készítendő fájl. Ha ezt az argumentumot nem adjuk meg nem készül futtatható fájl.

WITH A paraméterfájl. amelyet a parancssorban előforduló argumentumok megadására használhatunk. A paraméterfájlban minden sornak az itt felsorolt kulcsszavakkal kell kezdődnie (FROM, TO stb). Ezek után állhat a kívánt argumentum, és végül pontosvesszővel elválasztva egy megjegyzés. Ha egy argumentum mind a parancssorban, mind a paraméterfájlban szerepel, a parancssorban szereplő az érvényes. A paraméterfájl alkalmazásával megmenekülhetünk a parancssor ismételt begépelésétől.

VER Meghatározza, hogy hová küldje a linker üzeneteit: ha ezt nem definiáljuk, az üzenetek a standard kimenetre kerülnek, amely általában az aktuális ablak.

LIB vagy **LIBRARY** meghatározza a felhasználandó scanned könyvtárakat (scanned libraries). A scanned (letepogatott) szó arra utal. hogy az ilyen típusú könyvtáraknak a használat módjában semmi közük sincs az operációs rendszer rutinjait tartalmazó könyvtárakhoz. Az sc.lib tartalmazza a szabványos C függvényeket (printf(). strcpyl) stb). Az Amiga.lib pedig áthidalja azt a problémát, hogy a C a vermen keresztül adja át a függvényparamétereket, az operációs rendszer pedig a regisztereket használja. A fenti kél könyvtárat mindig célszerű megadni a LIBRARY argumentumnál. A scanned könyvtárak a lib: directory-ban helyezkednek el:

Az Amiga.lib-bel kapcsolatban fontos szólni a bázismutatókról. A könyvtárak eléréséhez használt bázismutatókat mindig a megadott névvel **(IntuitionBase, GfxBase** stb.). függvényen kívül kell definiálni, mivel ezekre hivatkozások vannak az Amiga.lib-ben is. **Figyelem! A kis- és nagybetűket a C-ben nem lehet szabadon helyettesíteni egymással!**

MAP Meghatározza a map fájl nevét, amely tájékoztatja a programozót az egyes programrészekben definiált szimbólumokról. A map normál ASCII szövegfájl. Ha nem adjuk meg a map paramétert, nem készül ilyen fájl.

XREF Hasonló a map fájlhoz, azzal a különbséggel, hogy minden egyes programrészhez felsorolja azokat a kulsö szimbólumokat, amelyekre hivatkozunk.

WIDTH A sorhosszúság megadására szolgai', amelyet a map és xref fájl elkészítéséhez használ a linker.

Az összeszerkeszlés során különböző jelzéseket kaphatunk. A linker leggyakrabban azért ad hibajelzést, mert egy szimbólumot vagy kétszer, vagy egyszer sem definiáltunk, illetve nem pontosan adtuk meg a parancssort.

A sok paraméter megadása helyett, sokkal jobban járunk, ha a linkelést rábízzuk az se fordítójára. Az se elvégzi helyettünk a paraméterezést, az előre definiált SCOFPINOS segítségével. Vigyázzunk, mert ha nem adtuk meg a linkelési opciót, akkor nem fog linkelni, bár jóllehet lenne már elég memóriánk hozzá. Így csak annyi memóriát nyerünk, amennyit az editor foglalt el. Vagy használhatjuk a Build-ot. amelyre rákattintva elvégzi helyettünk az összes fordítási problémát. Ügyeljünk arra. hogy a nem szabvány könyvtárak, ill. saját c forrásaink, object-jeink, valamint assembler forrásaink, amiket használni szeretnénk egy könyvtárban legyenek

Legyünk körültekintőek, mert ha a Build úgy találja, hogy van futtatható, ill. léteznek az object-ek. akkor nem fog csinálni semmit ha újra akarjuk fordíttatni. Tehát újra fordítás előtt töröljük le azokat. De ha mégis sajátke-/Cüleg akarunk linkelni ezt például így tegyük:

A shell ablakban jelöljük ki aktuális útvonalnak a forrásunk helyét. Végezzük el az editorból a fordítást. Ha memória hiányában jutottunk erre a sorsra, úgy is csak addig jutunk, hogy az object-ekig sikerül lefordítani. Nézzünk egy konkrét példát az első programunkon:
írjuk be a shell ablakba dfO:Első_programunk
utána: Síink FROM libx.o "Elso_programunk" TO "Elso_programunk" LIB lib:Amiga.lib lib:sc.lib

Ezek után elkezd molyolni. majd kiböki ezeket:
Warning 626: Libcode used on modulé lib:sc.lib

SLINK Complete - Maximum code size = 5356 ($000014ec) bytes

Final output file size = 5372 ($000014fc) bytes
ha ezt elolvastuk akár el is indíthatjuk a programot, ami kiírja az eredményt, nevezetesen: Hello world!

Beszélnünk kellene még a debugger-ről is. de az az igazság, hogy sok dologra eddig még nem használtuk. Mert egyrészt úgy gondoljuk, hogy a programunknak enélkül is kell működnie. Ha valami baj van. valószínűleg így sem fogjuk megtalálni. Azért az alapokról csak szólunk egy keveset. Először is mielőtt használni szeretnénk feltétlenül szerkesztessük be a fordítóval a debug információkat (lásd. COMPILER OPHONS / NoDebug. Debug=Line stb). A debugger-t mind parancssorból, mind Workbench-ből indíthatjuk. A Workbench-ből való indításnál a paramétert a Workbench-ben szokásos módon oldhatjuk meg. A debugger feltételezi a forrás meglétén túl a futtatható állomány meglétét is! A debugger-nek paraméterként meg kell adni a debug-golandó fájl nevét, és a debugger nyit magának egy új képernyőt, ahol először a menüből kiválasztjuk a main-t. mire megjelenik a forrás. A forrásban más színnel megjelenik az éppen végrehajtott utasítás. A képernyőn alapesetben két ablak látható az egyikben a forrásunk van, a másik úgynevezett dialógus ablak, ahol a debugger közöl velünk ezt-azt Ezeken kívül nyithatunk még egy ablakot amelyben ellenőrizhetjük az összes regiszterek tartalmát, ha van akár az FPU-ét is (F4-es bili.). A forrásunkat elindítva **(Jobb Amiga+S** ill. lásd **Run** menü) a forráson soronként ellenőrizhetjük a futást.Fűszerezhetjük a dolgot némi Breakpoint-okkal úgy. mint az assembly debugger-eknél. Természetesen beállítható, hogy akár assembly szinten is követhessük a történeteket, vagy hogy követhessük a függvények mélyére is. A debugger-rel részletesebben a következő könyvünkben fogunk foglalkozni, ahol a Device-k. Task-ok mélyére fogunk hatolni. A debugger előnyei ott tornyosulnak ki igazán.

1.2.2 Amitől Amigás C a SAS/C

A SAS/C tulajdonképpen egy olyan jól megírt C implementáció, amely mind az **ANSI** C, mind egy **UNIX** alatti C-t egyesít magában, plusz még az Amiga, illetve az AmigaDos sajátosságait is figyelembe veszi. A SAS/C-nek csak olyan Amigás sajátosságait szeretnénk itt bemutatni, amit jól tudunk hasznosítani programjainkban is. A felsorolás azonban nélkülözni fogja a teljességet, mivel nem célunk teljes kézikönyvet írni a SAS/C-hez hiszen ezt megtették a program írói. Nagy segítséget nyújtanak ezen felül a help könyvtárban található guide formátumú egyéb dokumentációk, amik részletesen tárgyalják az egyes függvényeket. Mi itt csak kiemelünk olyan direktívákat, amik szélesebb körben érdeklődést válthatnak ki. Ezen felül a könyv hátralévő részében a tényleges programozás során is ismertetünk ilyen dolgokat, szintén bemutatási célból.

1.2.2.1 A SAS/C specifikus fordítói, direktívák, függvények.

Már az előbbiekben is megemlítettük, hogy az Amiga programozásában igen nagy szerepet játszanak a rendszer rutin könyvtárai. Már fentebb írtunk róla, hogy mely könyvtárak miért is felelnek, de még nem tárgyaltuk meg ezek'használatát. A legfontosabb, az hogy ha használni akarunk függvényeket, vagy akár csak egyet is. az őt tartalmazó könyvtárat meg kell nyitnunk, hogy használhassuk. Valamint nem árt ha beszerkesztünk egy pár olyan fájlt, amiben deklarálták, hogy mik is vannak abban a könyvtárban, valamint emberibbé tették bizonyos definíciókkal azok használatát. Ha C-ben programozunk, maga a SAS/C gondoskodik arról, hogy programunk számára megnyissa az Exec könyvtárat, valamint a Dos könyvtárat. Tehát ezeket nekünk nem kell. Azonban ha hivatkozni akarunk ezeknek a könyvtáraknak a bázismutatójukra akkor bizony ezeket is meg kell nyitnunk saját magunknak.

Arra. hogy ezeket a könyvtárakat maga a SAS/C megnyitja azért van szükség, mert maga az a függvény amely megnyitja a könyvtárakat, az is egy könyvtár függvénye. Mégpedig az Exec könyvtáré. Nos. ha már megnyitottuk a könyvtárat illik be is zárni, mikor befejezte a programunk a futását. Csak megjegyzésképpen a Dos könyvtárat azért nyitja meg a SAS/C, mert a standard be Ül. kimeneti rutinjai ezen keresztül érintkeznek a rendszerrel, és ezt használják. Tehát ahhoz, hogy használni tudjuk az exec rutinjait, nem kell megnyitnunk magát az Exec-et, mert ezt már megnyitották nekünk. Nézzük meg akkor hogyan is néz ki egy könyvtár megnyitása. Először is szerkesszük be azokat a szükséges definíciókat amik majd a használatához kellenek. Nézzük meg mondjuk az Intuition megnyitását.

#include <eKec/types.h> (Ez tartalmazza az alapvető Amigán használt típusok definícióját)

#ínclude <intuition/intuition.h> (Ebben benne vannak az alap Intuition definíciók)

Majd definiálnunk kell egy változót, ami az Intuition báziscímét tartalmazza.

struct IntuitionBase "IntuitionBase;

Mint látható ez nem egy sima mutató, hanem egy struktúra mutató. Maga a struktúra is definiálva van az inluition/intuition.h-ban. A struktúra sok mindent tartalmaz, pl. az egér pointerének koordinátáit is. Vigyázzunk rá, hogy a struktúrát mi magunk ne írjuk, és ne változtassuk meg, mert esetleg ez a rendszer összeomlásához vezethet. Egyszerűbb esetben egy sima guruval megúszhatjuk, rázósabb esetben kárt tehetünk állományainkban is.

letlclL UcKlnuutunK cgj li.tuiu=„D>——,i„.kt.wí™ mutatń rntuitionRnse változót. Annak, hogy ez legyen a neve bizonyos okok miatt van szükség (nevezetesen olyan állományokban amik később kerülnek a forráshoz fordítás

során). Emiatt minden könyvtár bázismutatónak megvan a neve amit úgy és csakis úgy használjunk. Például a graphirs könyvtárnak **GfxBase.** Itt még egyszer felhívnánk a figyelmet arra a tényre, hogy a C különbséget tesz a kis- és nagybetűk között! A második fejezetben ahol a könyvtárakat részletesebben tárgyaljuk, kitérünk a tárgyalt könyvtárak könyvtár bázis neveire Hl. típusára. Akkor nyissuk végre meg az Intuition-t. ami az alábbiak szerint néz ki:

```
main (uoid)
{
        IntuitionBase= (struct IntuitionBase *)
                        OpenLibraryC'inuition.library",    0);

        if(IntuitionBase==NULL)
            exit(); (ha nem sikerült megnyitni, ami pl.: kevés
                     memória miatt előfordulhat, kilép a programból.)

        ... (A mi programunk ahol használhatjuk most már az Intuition
függvényeit is...

mielőtt kilépnénk zárjuk be az Intuition-t!
        CloseLibraryí IntuitionBase);
}
```

Az OpenLibrary függvény prototípusa és paramétere a következőképpen alakul:

struct Library *OpenLibrary(STRPTR LibNév, ULONG verzió)

Mint látjuk az OpenLibrary függvény egy Library mutatóval tér vissza. Ezért kellet az Intuition megnyitásakor egy típuskonverziót végrehajtatni a fordítóval. Paraméterként egy string pointert vár (rhar *). ami a megnyitandó könyvtár nevét tartalmazza. **Figyelem!** A könyvtár neveket mindig kisbetűvel írjuk és a **.library** -ra végződnek, szintén kisbetűvel! Valamint a megnyitni kívánt könyvtár verzió számát kell megadnunk. Ha ez 0 akkor bármilyen verziószámot talált is a könyvtárban, megnyitja azt. Vigyázzunk, mert ha 2.0-ás rendszer alá írunk programot és a könyvtárat O-ás verziószámmal nyitottuk, kihasználtuk azt, hogy a 2.O ás rendszernek több függvénye van mint mondjuk az 1.3-asnak. Ha a programunkat ilyen feltételek mellett egy 1.3-as rendszer alatt indítjuk valószínűleg nem fog futni, elgurul stb. Viszont ha a 2.0-ásnak megfelelő 37-es verziószámmal nyitjuk simán kilép, esetleg jelzi, hogy azért lépett ki. mert hiányolta a 2.0-ás rendszert. Lehetőleg úgy írjuk programjainkat, hogy az user-eknek ne az ujjúkból kelljen kiszopniuk azt, hogy miért nem fut a program. Erre számtalan eszköz áll a rendelkezésünkre (lásd. példa programok a lemezen).

Ahhoz azonban, hogy ne keljen beírni minden könyvtár nyitáskor a könyvtárak verziószámát, a SAS/C kis segítséget nyújt (lásd. lejjebb a__os-**libversion-nál).**

Természetesen az OpenLibrary függvény akkor is megnyitja a könyvtárat ha az nem kickstart könyvtár, hanem a SYS:libs-ében található könyvtár.

Gondoskodik a betöltéséről a memóriába, stb. Tehát ha már nem használjuk illik bezárni, hogy ne foglalja a memóriát feleslegesen más alkalmazások el dl. Erre használjuk a **CloseLibraryO** függvényt. A CloseLibrary nem ad vissza értéket. Paraméterként a becsukandó könyvtár báziscímére mutató pointert vár. Sajnos nem ellenőrzi, hogy nem kétszer zártuk-e már be azt a könyvtárat. A rendszer viszont egy guruval szokta díjazni ha kétszer zárjuk be a könyvtárunkat!

A SAS/C kicsit különbözik más szabvány C implementációktól, mert egy pár érdekes dolgot tartalmaz, ami az Amiga rendszere kapcsán került bele.

Nézzünk néhány ilyen, az előfeldolgozónak megadható opciót.

__chip vagy **chip** segítségével rábírhatjuk a fordítót arra. hogy a deklarált változót a chip ram-ba helyezze el a kód inicializálásakor. Erre azért van szükség, mert ha esetleg egy ábrát, vagy egy sprite-ot, esetleg BOB-ot. vagy hangot szeretnénk a programunkba megjeleníteni annak a chip ram-ban kell lenni ahhoz, hogy meg tudja az Amiga jeleníteni. Persze mi is elvégezhetnénk ezt egy exec függvénnyel, sőt a SAS/C is azt használja, csak sokkal kényelmesebb, és nem lesz meg kétszer ugyanaz a memcsiben. Használata a következő:

```
USHORT__chip kep[]={Oxffff,Qxffff,Qxffff,OxDOQQ
                    0x0000,0x0000,0x0000, Oxffff};
```

Ekkor a kép[] tömb a chip ram-ba kerül, amiről akár meg is győződhetünk, ha kiíratjuk a &kép[] értékéi. A kép vagy egyéb object megjelenítésével később foglakozunk.

__stack, ennek segítségével a programunknak adhatunk elinduláskor akkora stack méretet amire szüksége van. Nem kell amiatt leállnia. ha ezt a meghívásakor az user nem biztosította számára. Használata a következőképpen néz ki:

```
#include <dos.h> (itt van deklarálva az alapértelmezés, elhagyható)

long__Stack = 10000L; (lOOOOO üres bájtot bocsátottunk a
                       rendelkezésre a programunknak)
main(uoid)
{
(ide kerülhet a programunk)
}
```

__STKNEED-el megadhatjuk a programunk számára szükséges minimális stack méretét. Alapértelmezése 400 byte míg a__stack alapértelmezése 8192 byte.

Használatakor a rendszer megszakításkor használja a vermet de mindig üresen tart belőle 400 byte-ot. Használata megegyezik a__stack-éval.

__priority, a segítségével a programunk az itt megadott prioritással indul el. így nem Kell e/,i Kívüliül incgvó.lu>r-.tc\tni, «&gy Állítani Használata ese-

tén adjuk meg a linker-nek a cback.o-t is. mert nélküle nem értelmezett! Azonban azt sem felejtsük el. hogy a prioritás csak -128. 127-ig vehet fel értéket. Használata így néz ki:

```
#include <dos.h>  (itt van deklarálva az alapértelmezés, elhagyható)

long __príority = 50; (a programunk prioritását 50-re változtattuk)

uoíd maín(uoid)
{
(ide kerülhet a programunk)
}
```

__procname segítségével háttérprocedúra nevet definiálhatunk. Bármilyen nevet használhatunk, de a névnek jelen kell lennie a programban. A használata esetén szintén meg kell adni a linker-nek a cback.o-t is.

Ennek a használata hatástalan, ha a programot Workbench alól indítottuk.

Használata:

```
#include <dos.h> (itt van deklarálva az alapértelmezés, elhagyható)

char *___procname="SpeciállisEffektus" (beállítottuk a
                                        "SpeciálisEffektus"  névre)

uoid main(uoid)
{
(ide kerülhet a programunk)
}
```

_SLASH használatával lehetőségünk lesz a 'V karaktert is használni az Amigán szokásos '/' helyett. Jelentősége például az **strmünO** (make a filename from components), azaz készíts egy fájl nevet a komponensekből.

Vagy az **strmfpfj** azaz készíts egy fájlnevet az útvonal vagy node-ból.

A használata rendkívül egyszerű.

```
eKtern char _SLRSH;
```

Ha a programunk Workbench alól fut, akkor a standard bemeneti ill. kimeneti függvények a Konzolról (Con:) veszik, ill. adják az eredményüket, ill. várják az user megmozdulásait (pl.: **printffj, getchfj, putchO** stb.).

Tehát, ha a programunkban használunk olyan függvényt amely bemenetének ill. kimenetének tekinti a konzolt, akkor **___stdiowin** segítségével definiálhatjuk a megnyitásra kerülő konzol ablak méreteit. Alapesetben a dos.h-ban ez úgy van definiálva, hogy a 10,10-es képernyő' pozíciójában nyit egy 320 pixel széles és 80 pixel magas ablakot. •

Használata az alábbiak szerint nézhet ki, például;

```
char___stdÍOLüinl]="C0N:10                (az ablak x koordinátája)
                   /10                    (az ablak y koordinátája)
                   /600                   (az ablak szélessége)
                   /108 .                 (az ablak magassága)
                   /flz_én_abalakom";     (az ablak fejléce)
```

Azonban, ha ki szeretnénk használni a 2.0-ás Workbench-töl élő lehetőségeket akkor használhatjuk az **___stdiov37 opciót** is. amely használat szempontjából a következőképpen néz ki:

```
char_stdiou37[]="/flUT0/CL0SE/lüflIT";
```

Megjegyzésképpen ezt a sort használja default-ban a SAS/C-is (lásd. dos.h).

A 2.0-ás rendszer megengedi, hogy az alábbi paramétereket használjuk:

AUTÓ: addig nem nyitja meg az ablakot amíg valami nem hivatkozik rá

CLOSE: az ablakra kiteszi a close gadget-et

WATT: megvárja míg becsukjuk az ablakot, nem kapja el. ezt tehetjük a close gadget-tel, valamint ha egy fájl vég jelet írunk be (Ctrl+\).

WINDOW Ox' cím +: megadhatjuk, hogy használjon mutatót az ablakra. A címet természetesen hexában várja a *"h+'* jelek között.

SCREEN név: megadhatjuk a pubscreen nevét amelyre az ablak nyílni fog. Ez pl.: a Workbench-en kívül lehet mondjuk a DirOpus-é is.

Ezeken felül még használhatjuk valamennyi kulcsszót, amit az Intuition is használ a **"window flag"-ek** címén (lásd. az Ablakok és Képernyők c. fejezetet) pl. néhány: **NODRAG, NOSIZE, NOBORDER, BACKDROP, SIMPLE és SMART.**

Programban való használatuk az alábbi:

```
#include <dos.h>

char___stdiowin[]="C0N:30/30/32Q/5B/flz én programom";
char____stdiou37[]="flUT0/CL0SE/LUHIT/N0SIZE";

uoid maín(uoid)
{
printfí" Hz én programom! \n");
!
```

Az___oslibversion opciót arra használhatjuk, hogy a programunk elején deklarálva, a könyvtárak megnyitásakor az itt megadott verziószámot veszi alapul. Használata a következő pl. 2.0-ás rendszer alatti programokhoz:

```
long___oslíbucrsion — 37 ;
```

Az egyes operációs rendszer verzió számai a következő könyvtár számot takarják:

Oprendszer verzió	Library verzió
1.2	33
1 3	34
2.0	36 (az első verzió, kis tételben létezik)
2.04	37 (ez az általános)
2.1	38
3.0	39
3.1	40

A példa programja az alábbi:

```
#include <dos.h>

long__oslibuersion = 37;

uoid main(uoid)
{
(a programunk helye)
}
```

A másik library-vel foglalkozó opció a **__autoopenfail** függvény, melynek használata olyankor segít ha deklaráltuk, de nem definiáltuk a rendszer könyvtár bázisát - ekkor ő azt automatikusan inicializálja. Ha az autoinitialized könyvtárakat nem tudtuk megnyitni, akkor meghívódik az **__autoopenfailfj** függvény. A függvény így néz ki: **uoid__autoopenfail(char *líb).** Részletesebb információ az sc:source/autoopenfail.c-ben található.

__ctype egy karakter osztály tábla, amelyben a normál ASCII karakterekhez tarozó attribútumok találhatóak. A tábla 257 bájt hosszú, bájtonként egy ASCII karakter plusz az EOF ami -1. Az egyéni bitek jelentése a táblázat bájtjain:

Bit	Érték	Jelentés
_U	**1**	nagybetű (A-Z)
_L	**2**	kisbetű (a-z)
_N	**4**	decimális szám (0-9)
_S	**8**	szóköz karakter
_P	**16**	írásjelek
_C	**32**	kontrol karakter
_B	**64**	blank (üres) karakter
_X	128	Hexadecimális szám (0-9, a-f, A-F)

A táblán eszközölt változtatások hatással lesznek az alábbi függvényekre: **isalnumfj. isalphafj, isasciifj, iscrtnlO, isdigitfj, isgraphfj. islowerfj, isprintO, ispunctfj, isspacefj, isupperfj, isxdigitfj.**

Használatakor be kell szerkesztetni a **<ctype.h>** fájlt.

_CXBRKO üzenet kiírás és kilépés a CLr]+C billentyűk lenyomására. A függvény szinopszisa a következő: **UOid _CHBRK(uoíd).** Ez a függvény meghívódik a__chkabort által, amikor a___chkabort aktivizálja. A függvény alapbeállítása szerint meghívásakor a "**Break and exit" üzenet jelenik meg az ụser-nek a szabvány kimeneten (általában a konzolon). Ha saját üzenetet akarunk azt egyszerűen csak a függvény kapcsos zárójelei közé kell megadnunk az alábbiak szerint:

```
uoid___regargs___CKBRK(uoid);

uoid___regargs___CXBRK(uoid)
{
(Az üzent rutinunk helye)
}
```

Alihoz, hogy kikapcsoljuk a programunk számára a Ctrl+C esetén fellépő program megszakítást egyszerűen csak újra kell deklarálnunk a__chkabort() függvényt az alábbiak szerint:

```
uoid___regargs___chkabort(uoid);

uoftl___regargs___chkabort(uoid)
{
}
```

Ezek után a programunkat nem lehet megszakítani Ctrl+C segítségével.

__buffsize második szintű B/K puffer nagyság definíció. A szinopszisa: **eKtern int___buffsize.** További tájokoztatást találunk az **fopenQ** ill. a **setbuffj** függvényeknél. Vigyázzunk arra. hogy már nyitott fájl esetén ne használjuk!

_finask segítségével beállíthatjuk a SAS/C fájl B/K műveleteinél alkalmazott fájl védelmi bitjeinek értékét. Definiálva a fcntl.h fájlban van. Az állítás értéke hatással lesz természetesen az ANSI C állapotra is.

__emitfj függvénnyel közvetlenül végrehajtathatunk egy 680x0 utasítás szót. Használatával körültekintően járjunk el. mivel nem ellenőrzi a kód valódiságát! Használata:

```
#include <dos.h>

int Í=0H4180;
```

uoid___emit(i) (A *CHK dO.dO Assembly parancs végrehajtása)*

A paraméternek megadott szám csak 16 bites lehet. (A 68000 miatt!)

getregO Egy 680x0 specifikus regiszter bevétele. A függvény beolvas egy processzor regisztert és az értékével tér vissza. Argumentumként egy int számot vár amely az olvasni kívánt regiszter számát tartalmazza. A függvény szinopszisa a következő: **long getregtint regiszter_néu).** Szerencsére a regiszterek szimbólumoknak is definiálva vannak . Nézzük át ezeket:

érték	regiszter	érték	regiszter	érték	regiszter
0	REG_DO	**8**	REG_AO	**16**	REG_FTO
1	REG_D1	**9**	REG_A1	**17**	REG_FP1
2	REG_D2	**10**	REG_A2	**18**	REG_FP2
3	REG_D3	**11**	REG_A3	**19**	REG_FP3
4	REG_D4	**12**	REG_A4	**20**	REG_FP4
5	REG_D5	**13**	REG_A5	**21**	REG_FP5
6	REG_D6	**14**	REG_A6	**22**	REG_FP6
7	REG_D7	**15**	REG_A7	**23**	REG_FP7

putregO A getregO ellentétje, mert ez ír a processzor regisztereibe. A szinopszisa: **uoid putregíint reg,long *u),*** ahol a regiszter információk megegyeznek a **getregO-nél** tárgyaltakkal. Használatánál legyünk figyelemmel arra a tényre, hogy nem igazán szerencsés multitaszkos gépen közvetlenül írni a processzor regisztereit. A lebegőpontos regiszterek természetesen csak akkor élnének ha a gépben van **FPU,** és a fordító opciói között szerepel az. hogy math=68881.

__inline Ez a kulcsszó a global optimalizáló része, amelynek segítségével az adott rutin minden helyre befordításra kerül (mintha egy makró lenne).

1.2.2.2 Programok összefűzése SAS/C környezetben

A C-beli munkáink során gyakran szükségünk lehet arra. hogy bizonyos előre megírt rutinjainkat használhassuk az új programjainkban is. Most ezt tárgyaljuk meg, hogy ez miként is működik. Természetesen mint mindent, ezt is példák útján a legjobb magyarázni. A példákra azonban majd abban a fejezetben térünk ki. amelyben magát a programot tárgyaljuk. Inkább tekintsük ezt amolyan előlegnek.

A legegyszerűbb megoldásnak az ígérkezik, ha egy korábban megírt C programunkból vesszük ki a nekünk aktuálisan szükséges részeket. Ezt megtehetjük úgy. hogy a felesleges dolgokat kitöröljük, majd esetleg kiegészítjük az új funkciójának megfelelő dolgokkal. Kipróbálása után egyszerűen kitöröljük a programból a main()-t és a fordítási direktíváknál nem adjuk meg a linkelést. Az új programunkba pedig beszúrunk egy " **#include <elòzó_prográmunk.c>** " sort. Ha viszont nem akarjuk, hogy fordítás során mindig újrafordítsaaz előző programunkat is a fordító, egyszerűbb, ha az előző program már object-ként kerül az új programunkhoz. Ekkor viszont azt le kell fordítani, majd az új programunkhoz linkeltetni.

Ezeket úgy tehetjük meg, hogy az előző programnál a main{) nélküli verziót, tehát a végleges verziót az editorból lefordítjuk NoLink opcióval. így ha a program hibátlan volt. létrejön egy elözö_program.o nevű fájl. Ezt a nevet be kell írni az új programunk fordítói direktívák közé. ott is a linker részhez, hogy (COPMRILER OPTIONS Index LINKER OPTIONS Libraries/Objects listview gadget-ébe az add gadget-re kattintással egyszerűen bepötyögjük a mellette található helyre az object-ünk nevét) előző_programunk.o. Ezzel még nem végeztünk mert még a fordító tudomására kell hoznunk, hogy az előző_program-ból használni kívánt függvények később kerülnek a kódunkhoz. Ezt a C-ből ismerős módon adhatjuk meg, nevezetesen a prototípus deklarációnk elé tesszük az **extern** kulcsszót. Ez ilyen formában nézhet ki:

```
extern uoid elözö_programból_függuény1(int argumentum!,
int argumentum2);

extern int előzó_programból_függuény2(uoid);
```

Erre példát láthatunk a könyv 1.6 fejezet példaprogramjaiban (Gadget_12.c).

Ha azonban assembly rutint szeretnénk hozzálinkelni, annyiban változik a dolog, hogy assernblyben előszeretettel használunk regisztert a paraméter átadásához, és a visszatérési értéket is regiszterben szoktuk visszaadni.

Mivel az ember nem szereti a szokásait megváltoztatni, nézzük meg hogyan lehet erre a fordítót rábírni. Ha már az assembly rész megvan, és object-ként a rendelkezésünkre is áll, nagyon egyszerű dolgunk van (természetesen ugyanúgy, ahogyan azt az előbb tettük itt is be kell írni a linker-nek az object elérését, lásd fent). Egyszerűen a függvény prototípus deklarációjakor megadjuk, hogy mit, hol várunk. Az alábbi példa szerint:

```
extern__asm uoid assemby_rutinunk(regíster__D8 long, register
__aü char *);
```

Ahol az extern utal arra. hogy a rutin külsőleg kerül a kódhoz. Az **__asm** utalás arra, hogy a függvény assembly. A void azt. hogy a függvénynek nincs visszatérési kódja. Az assembly_ruUnunk a függvényünk neve. amire a korlátozások olyanok mint a C-beli függvényekre általában. A **"register"** azt jelzi, hogy a paramétert a fordítónak regiszteresen kell a függvény rendelkezésére bocsátani. A__dO azi jelentí. hogy a paramétert a függvény a dO-ás processzor regiszterbe várja, és a paraméter típusa long, mivel a regiszter 32 bites. Az__aO-nál ez annyiban módosul, hogy a char változó címét adjuk át a függvénynek és azt az aO-ás processzor címregiszteren keresztül. Ez a C felől mint láttuk nem is okoz problémát, viszont van egykét dolog amit az assembly részénél, a függvény írásakor be kell "tartanunk.

Az assembly programban definiálni kell, hogy a többi program mely belépési pontokat lássa (címkéket) az **Hdef** címke fordítási direktívával. Ha az assembly programunk használna külső rutinokat akkor az **xref-el** azokat is definiálnunk kell. Ezután legalább egy szekciót is definiálni kell a **section** direktívával, pl.: **section random_generator,code.**

Ekkor a lényeg a code. mely most publir memóriaként szerepel. A/ assembly részt le kell zárnunk egy **End** direktívával.

Most álljon itt egy nagyon egyszerű példa, mely meghívásakor egy ULONG értéket ad vissza a C programnak.

A C forrás:

(Próbálkozgatunk...)

```
#include <stdio.h>
#include <ewec/types.h>

uoid maín(uoid); e«tern___asm ULONG probaíregister__DB ULONG);

uoid maín(uoid)
{
    ULONG a;

    a=proba(6);
    printf("Eredmény:%d\n",a); }
```

Ezután lássuk az assembly programot:

```
; Próba a SflS/C 6.51-hez

            Kdef_proba

            section program,code

_proba:   asr.l #1,dO
            rts

            end
```

Miután a SAS/C-t így kitárgyaltuk, lássuk az első programunkat, hogyan is néz ki. Mondjuk a C-ben megírt részhez nem sok hozzáfűzni való van (lásd 1.5 rész).

1.3 Az Asm-One assembly fordító (verzió 1.26)

1.3.1 A kezdet

A számítógépekben a processzor közvetlen programozását gépi kódban lehet elérni. Ez a nyelv teljesen a számok világa, csak a proceászor érti. emberek számára igencsak érthetetlen. E probléma áthidalására találták ki az assembly-t. Ez egy olyan nyelv, amely rövidítésekből **(mnemonik)** áll. s ezek utalnak az utasítások funkcióira. Az assembler ezt a szöveget (forráskódot vagy source-t) lefordítja számokra, amit a processzor már megért és végre tud hajtani. Az Amiga számítógépre az első ilyen elterjedt fordító a SEKA assembler volt. Saját beépített szövegszerkesztőjével, sok kényelmi funkciójával igen hamar belopta magát a programozók szívébe. Ma a legelterjedtebb Asm-One fordítónak volt az öse. Az Asm-One legelső verziója 1990-ben jelent meg, ma az 1.26-os verziószámmal ellátott fordítónál tartunk. Ez már 680x0 jellel van ellátva, ami azt jeleni, hogy 68000-től a 68040-ig minden utasítást ismer, kezeli a 68881/82-es koprocesszort és a 68851-es MMU-t. Az évek során sok ember vett részt a fejlesztésében, remélve, hogy egy jobb és használhatóbb assembler-t sikerül készíteniük.

A programot elindítva egy képernyő jelenik meg. melyen a verziószám és a készítők neve szerepel. A program kéri a munkaterület típusát: **fást, chip, public** vagy **absolute.** A munkaterület az a rész ahová a forráskód, és ha másképp nem akarjuk, akkor a lefordított gépi kódú program kerül. A fást a fást ram-ot jeleni, a chip a chip ram-ot. a public azt, hogy ha van fást ram. akkor oda rakja, ha nincs akkor a chip. ram-ba. az absolute pedig konkrét tárcímet vár. A abszolút címet csak oda engedi rakni a program, ahol más program kódja vagy lefoglalt adatterület nincs. A megfelelő kezdőbetű beírása után vagy egyszerű enter-rel (public) kell megadni a munkaterület nagyságát (minimum 100 Kbyte ajánlott).

Ha mindez megvan akkor az assembler készen áll a munkára, egy parancssort kaptunk. Innen lehet a forráskódunkat lefordítani, futtatni, kilépni, stb.

Próbaképp gépeljük be azt. hogy "x" és utána Enter. Ha minden jól ment akkor egy kisebbfajta táblázatot kaptunk eredményül, ami a processzor regisztereit foglalja össze. A táblázat jelentésére nemsokára kitérünk a parancsok részletes leírásánál. Ezután, ha megnyomjuk az ESC gombot akkor a szövegszerkesztőbe jutunk, ezzel lehet a parancssor és a beépített editor között váltogatni. A Control + ESC egy félképernyős editorba vált.

Lépjünk be az editorba és gépeljük be az alábbi rövid programot:

```
STHRT: MOUE.L #2,D0
RTS
```

Készen van első kisebb assembly programunk. Az assembly utasítások részletes leírásától eltekintünk, ezt 'A 68000-es mikroprocesszor' című könyvben megtalálhatjuk. A START egy úgynevezett címke, ennek van egy címe, hivatkozásra' használható. A **MOUEL #2,D0** már egy utasítás, a DO arlatregiszterbe berakja a decimális 2 értéket. A MOVE az angol mozgat szó. a .L pedig a használt méretet jeleni, ami lehet **Byte, Word** és Long. Mi most a Long-ot használtuk, ez a legnagyobb. 32 bit. Az **RTS** utasítás pedig (szubrutinból való visszatérés) lezárja a programot amire mindig szükség van.

Ezután térjünk vissza a parancssorhoz és írjuk be. hogy "a" és Enter. Ha a következő három sor jelenik meg akkor jól dolgoztunk:

```
Pass 1..
Pass 2..
No Errors
```

Az első programunkat a fordító sikeresen lefordította a memóriába. A futtatáshoz már csak be kell gépelni, hogy "j" utána Enter és már fut is a programunk. Újra megjelent az "x"-hez hasonló táblázat néhány inverz értékkel. Mit is jelent ez? Azt. hogy ezek az értékek megváltoztak.

Sikerült a DO adatregiszter előző értékét megváltoztatni. Fordítsuk le újra és futtassuk le a programunkat megint. Az eredmény az. hogy csak egy érték lett inverz az **SSP** (Supervisor Stack Pointer), de azzal nekünk nem kell foglalkozni (és nem is szabad!) Mivel a DO értéke 2 volt é& ismét 2-t raktunk bele így az értéke nem változott, ezért nem lett inverz.

A Asm-One egyszerre tíz lorrás szerkesztését teszi lehetővé, melyek között az editor részben az F1-F1O billentyűkkel lehet váltani vagy a megfelelő menüponttal, parancs módban pedig az "as" activate source parancs és a megfelelő forrás szám beírásával lehet. Pl: "as5" Enter. Az egyes parancsok csak az aktuális forrásra vonatkoznak.

A program használja a reqtools.library-t. requester-ek megjelenítésére a kényelmesebb használat érdekében. Ha ez nem áll rendelkezésre, akkor sincs semmi baj. a program így is működőképes, de akkora parancsokat paraméterezni kell.

1.4.2 Az Asm-One parancsai

Most pedig következzék a parancssor utasításainak leírása, részben a menük szerint haladva:

ZS - **Zap Source** - Forráslista törlése.
Az aktuális szöveget törli. Ha az utolsó mentés óta volt változtatás akkor rá is kérdez.

O - **Old** - Törölt forráslista visszahozása.
Ha töröltük a forráslistát és még nem csináltunk semmit, akkor visszahozza. Amennyiben ESC-vel beléptünk az editorba és vissza akkor már nem tudja visszahozni a forrásunkat. Megjegyezzük, hogy más módsze-

rekkel még a forrás nagy részéi ezután is vissza lehet hozni. Ha abszolút munkaterületet kértünk akkor azon a címen megtalálhatjuk a forráslistánkat, csak az első néhány byte séfült meg. Ezután azt a memória részt ki lehet menteni egy másik paranccsal.

R - Read Source - Forráslisla betöltése.
Az aktuális editor részbe betölt egy forráslistát. Ha az nem üres és még nem lett elmentve akkor a program rákérdez. Paraméterezve is használható mint a legtöbb parancs, pl.: "R CUT.S". Ha nem adunk meg paramétert akkor egy requester jelenik meg. Sikeres betöltés után kiírja a file méretét,

RB - Read Binary - Bináris flle betöltése.
Egy file-t tölt be a megadott helyre, ami lehet címke vagy memórialerület. Ezt a BEG1N: után kell beírni. Az END: kérdésre lehet Enter-t ütni, ekkor a teljes file betöltődik. Sikeres betöltés után kiírja a betöltött blokk méretét. Ha a betöltendő lile kisebb mint a megadott méret akkor a filet a szükséges méretre vágja.

RO - Read Object - Futtatható file betöltése.
Egy futtatható file-t tölt be szabad memóriaterületre. Sikeres betöltés után megadja a program belépési pontját, innen lehet futtatni a "j" paranccsal. ['l.: "j $101ea08".

W - Write Source - Forráslista kimentése
Az aktuális forráslisla kimentése. Sikeres kimentés után kiírja a mentett forráslisla méretét. Létező file esetén rákérdez, hogy felülírja-e?

WB - Write Binary - Bináris adat kimentése
Bináris adatokat lehet kimenteni a memória bizonyos részéből. Megadható címtartomány vagy címkék. Sikeres mentés esetén a kimentett blokk méretét kiírja. Többek közölt ezzel lehet az Old parancsnál leírt módszerrel az elveszett forrást kimenteni.

WO - Write Object - Futtatható file kimentése
Ez a/ egyik legfontosabb parancs. Ezzel lehet kimenteni a hibátlanul lefordított programot, ellenkező esetben No Object hibával leáll.
Előfordulhat olyan eset. hogy hibás fordítás után mégis hajlandó kiírni egy 0 byte-os file-t. de a No Object hibaüzenet ekkor is megjelenik.

WL - Write Link - Linkelhető file kimentése
Egy linkelésre alkalmas file-t lehet létrehozni vele. Lefordított kód van benne, de futtatásra így még nem alkalmas. Használni kell ilyenkor a **XDEF** és a **SECTION** parancsokat, amikre később kitérünk,

I - Insert - Forráslista beszúrása
Az aktuális forráslistába a kurzor helyétől forráslistát lehet beszúrni.

U - Update - Forráslista frissítése.
Ez egy nagyon hasznos funkció, a már létező forráslistát kimenti ugyanazon a néven, vagyis felülírja minden rákérdezés nélkül. A program futtatása előtt érdemes használni, ha a gép lefagyna akkor is meg legyen az utolsó verzió.

ZF - Zap File - File törlése lemezről.
A megadott file-t törli. nem kérdez rá.

ZI - Zap IncMem - Include-k törlése a memóriából.
A gyorsabb fordítás érdekében az include-kat csak egyszer tölti be a memóriába a fordító. Ha a fordító újabb include-k betöltése közben hibát jelezne, akkor érdemes használni ezt a funkciót. Ez azért van. mert olyan hivatkozást talál, amely egyszer már definiálva volt.

=M - Add WorkMem - Munkaterület növelése.
Az eredetileg lefoglalt munkaterületet lehet vele megnövelni, ha Workspace Memory full hibaüzenettel leállna a fordítás. Csak bizonyos határig lehet növelni a memóriafregmentáoió miatt. Ha nem sikerül akkor érdemes a programot újraindítani és nagyobb memóriát foglalni.

- **About** - Információk a programról.
A programról és a szerzőkről információk.

! - **Quit** - Kilépés a programból
Kilépés a programból vagy a program újraindítása. Ha valamelyik forráslista nincs elmentve akkor rákérdez. A Restart funkcióval a program újraindítható kilépés nélkül (a nagyobb munkaterület lefoglalására jó).

!! - **Quick guit** - Gyors kilépés
A program semmit sem kérdez azonnal kilép.

A - Assemble - Fordítás
Az aktuális forráslistát lefordítja. Két menetben fordít, ha hibát talál akkor a fordítás abbamarad. Hiba esetén ESC-vel belépve a szerkesztőbe a kurzor a hibás sorra áll.

AO - Assemble with Optimize - Fordítás optimalizálással.
Ugyanaz mint az assemble, csak optimalizálja a kódot, és a forrást is ennek megfelelően átírja. Pl.: **"STHRT: BRH STflRT" -> "STHRT: BRH.B STRRT"** ezzel a kód gyorsabb is lesz és rövidebb is.

T - Jump Top - Forráslista elejére.
Az aktuális forráslista elejére állítja a kurzort es kiírja a sor tartalmát.

B - Jump **Bottom** - Forráslista végére.
Az aktuális forráslista végére állítja a kurzort és kiírja a sor tartalmát.

L - Search - Keresés.
A forráslistában a megadott szöveget megkeresi a kurzor sorától kezdve, paraméter nélkül pedig folytatja a keresést Pl.: "lbeta". ekkor a "béta" szót keresi. Vigyázzunk, hogy az "l" után ne rakjunk szóközt, mert azt is beleveszi a keresendő szövegbe.

ZL - Zap Line(s) - Sor(ok) törlése.
A kurzortól kezdve megadott számú sort töröl. Paraméter nélkül egy sort töröl. Mindig rákérdez, hogy biztos-e?

P - Print Line(s) - Sor(ok) kiírása.
A kurzor pozíciójától kezdve megadott számú sort kiír a képernyőre.

EL - **Extend Labels** - Címkék kibővítése.
Ez egy nagyon hasznos funkció lenne, de sajnos bug-os és nem működik. A kurzor sorától kezdve minden címkéhez hozzá lehetne fűzni az elejére vagy a végére egy szöveget.

M - Edit - Memória szerkesztése.
A megadott címtől kezdve a memória tartalmát byte-onként lehet módosítani. Csak az Enter lenyomására marad a régi érték és a következőre lép. ESC-re pedig visszatér a parancssorhoz. Az egyes szerkesztendő elemek mérete is megadható, pl.: **"m.l $100000".**

D - Disassemble - Visszafordítás.
A megadott címtől kezdve a memória tartalmát mnemonikokra visszafordítja. Példaként első programunk lefordítása után írjuk be, hogy "d start" és megjelenik a lefordított programunk, utána pedig sok "értelmetlen" sor. Ebben a részben soronként is átírhatjuk a programunkat.

H - HexDump - Hexadecimális Duinp.
A megadott címtől kezdve a memória tartalmát hexadecimális byte. word és long szervezésben kiírja, jobb oldalon pedig ASCII formátumban. Ebben a részben is át lehet írni a memória tartalmát, mind hexa mind az ASCII formátumban. Megadható méret is. "h.l $1000". "h.w" vagy "h.b $180000".
Fordítás után ha beírjuk, hogy "h start" akkor megláthatjuk első programunkat lefordított állapotban: 20 3c 00 00 00 02 4e 75. Ez az amit, a processzor már végre tud hajtani.

N - ASCII - ASCII Dump
A megadott címtől kezdve a memória tartalmát ASCII formátumban jeleníti meg. A memória tartalmat át lehet írni.

@D - Disassemble Lines - Sorok visszafordítása
A megadott címtől kezdve megadott számú sort visszafordított sort kiír a képernyőre.

@A - Assemble - Soronkénti fordítás.
A megadott címtől kezdve soronként lehet assembly sorokat beírni, a/onnal fordítja. Hiba esetén újra megismétli a sor számát, újra be kell írni a sort. Kilépés az ESC billentyűvel.

@H - Hexadecimal Lines - Hexadecimális Dump.
A megadott címtől kezdve 8 hexadecimális sort ír ki a képernyőre.

@N - ASCII Lines - ASCII formátumú Dump.
A megadott címről kezdve 8 sornyi ASCII memóriatartalmat ír ki a képernyőre.

@B - Binary Lines - Bináris formátumú Dump.
A megadott címtől kezdve 8 egymás utáni byte bináris értékét írja ki.

S - Search Memory - Keresés a memóriában.
A BEG> kérdésre fordítás után címke vagy egy cím adható meg mint kezdet, az END> kérdésre szintén címke vagy cím. A DATA> lehet egy szöveg, pl.: "AMIGA", itt ki kell tenni az idézőjeleket, vagy adatok, pl.: $01,$ff,$fe Az összes előfordulási hely címét kiírja.

F - Fill Memory - Memória feltöltése.
Megadott értékkel megadott memóriát feltölt. Lehet így is megadni az adatot: "amiga". Vigyázzunk vele. nehogy olyan területet torsünk fel ami az operációs rendszeré vagy esetleg egy másik futó task.

C - Copy Memory - Memóriablokk másolása.
Megadott blokkot a célterületre másolja. Ezzel is körültekintően kell bánni! Címke is megadható.

Q - Compare Memory - Memóriarészek összehasonlítása.
Megadott memória részeket hasonlít össze. Ha a két terület megegyezik "Equal Areas". egyébként "Not Equal Areas" üzenetet kapunk.

CS - Create Sinus - Sinus generálása memóriába.
Az alábbiak szerint sinus hullámot lehet a memória egy részébe generálni:
DEST> memóriacím vagy címke, ide generálja az adatokat
BEG> fokban a sinus kezdete
END> fokban a sinus vége >
AMOUNT> számolt értékek száma
AMPLITUDE> a maximális amplitúdó
YOFFSET> az y tengelyhez képet eltolás, lehet negatív is
SIZE (B/W/L)> a számolt értékek mérete. 8/16/32 bit
MULTIPLIER> szorzó, ezzel minden számolt értéket megszoroz
HALF CORRECTION> fél lépéses korrekció
ROUND CORRECTION> kerekítés
Sikeres generálás után "Sinus Created" üzenettel leáll.

ID - Insert Disassembly - Visszafordított kód beszúrása a forráslistába.
A kurzor helyétől kezdve visszafordított listát szúr be a forráslistába.
A "Remove unused labels (Y/N)" kérdés arra vonatkozik, hogy az olyan címkéket amelyekre nincs hivatkozás kiszedje-e?

```
PL: BEG>$í800d2
    END>$f80100
    Remove unused labels (Y/N)?y
```

IH - Insert HexDump - Hexadecimális értékek beszúrása a forráslistába.
A kurzor helyétől kezdve megadott memóriaterület hexadecimális (lump értékeit szúrja be a forráslistába. Megadható a méret is, pl.: "ih.b", "ih.w" és "ih.l".

IN - Insert ASCII Durap - ASCII byte-ok beszúrása a forráslistába.
A kurzor helyétől kezdve ASCII karaktereket szúr be a forráslistába, ha ez nem lehetséges akkor Hexadecimális értékre cseréli.

```
Pl.: BEG>$f80000
     END>$lB0100
```

IB - Insert Binary Dump - Bináris adatok beszúrása a forráslistába.
A kurzor helyétől kezdve bináris értékeket szúr be a forráslistába.

IS - Insert Sinus - Sinus értékek beszúrása a forráslistába.
A kurzor helyétől kezdve sinus táblázatot generál a forráslistába, a CS - Create Sinus működéséhez hasonlóan.

AD - Debugger - Debug avagy "bogártalanítás".
Az Asm-One egyik leghasznosabb része. Itt tudjuk a programunk futását sorról-sorra nyomon követni. Ez tartalmaz egy normál fordítást is, hiba esetén nem lép be a nyomkövetésbe. Később részletesen tárgyaljuk.

=S - Symbols - Szimbólumlista.
Fordítás után a használt címkékről ad információkat, a helyükről, fordítástól függően, (lsd. org)

PS - Paraméter Set - Program paraméterek beállítása.
Futtatáskor milyen paramétereket adjon át a futó programunknak, mint például "dir" dos parancsnak a "work:asm-one" paraméter.

J - Jump - Program indításéi JSR utasítással.
A lefordított programot általában ezzel szoktuk indítani. Ha nem adunk meg paramétert akkor az első lefordított soron kezdi a futtatást.
Paraméternek megadható címke vagy memóriacím. A programot RTS utasítással kell lezárni.

G - Go - Program indítása JMP utasítással és töréspont megadásával
Általában nyomkövetésre használjuk, óvatosan kell vele bánni. Hasonlóan működik mint a "j" parancs. A BREAKPOINT> kérdésre címkét vagy címet adhatunk meg, ha nincs több akkor Enter. Ha a program eléri az adott címet, akkor kilép.

K - Step - Lépésenkénti végrehajtás
Az aktuális programszámlálótól kezdve végrehajt egy utasítást. Fordítás után a programszámlálót a programunk elejére állítja.

X - Status - A processzor állapota
A processzor regisztereit összefoglaló táblázatot kapunk. A legfelső sor az un. adatregisztereket tartalmazza, jelölésük pl.: DO. A következő sor a címregisztereket tartalmazza, pl A6. Ezekbe közvetlenül lehet niax. 32 bites értéket írni, olvasni és persze ezekkel mindenféle műveleteket végezni. A következő sor olyan regisztereket tartalmaz amiket programunk működése során nem nagyon tudunk megváltoztatni közvetlenül és jogunk sincs hozzá, mert azokat csak az operációs rendszernek szabadna átírni (ez persze nem jelenti azt. hogy nem is lehet, mi is át tudjuk írni egy kis trükkel). Nézzük meg mik ezek a regiszterek:

SSP - Supervisor Stack Pointer - Rendszergazda veremmutatője
USP - User Stack Pointer - Felhasználó veremmutatője
SR - Status Register - Állapotregiszter
PC - Program Counter - Programszámláló
VBR - Vector Base Register - Vektor bázis regiszter 68010+

Az SR és a PC között levő jelek a SR részletesebb kibontása. Például a TI jelzés akkor jelenik meg **(Trace)** amikor az SR 15. bitje 1. Ez azt jelenti, hogy a nyomkövetés aktív (pl.: ha használjuk a "k" parancsot).
Az SR regiszter:

0. bit - Carry / Átvitel
1. bit - Overflow / Túlcsordulás
2. bit - Zero / Zérókapcsoló
3. bit - Negative / Negatív kapcsoló
4. bit - Extension / Bővítőkapcsoló
5-7. bit - Nem használt
8-10. bit - Interrupt Mást / Megszakítás maszk
11-12. bit - Nem használt
13. bit - Supervisor / Rendszergazda állapot
14. bit - Nem használt
15. bit - Trace / Nyomkövetés

Ha valakinek van 68881 /82-es matematikai koprocesszora akkor a program még néhány sornyi információt kiír:
FPCR - Mode Control Register / Mód vezérlő regiszter
FPSR - Status Register / Állapotregiszter
FPlAR - Instruction Address Register / Utasítás cím regiszter

A löbbi "furcsa" adat hasonló a normál SR regiszter részletezéséhez. Itt is megtalálható a zéró bit (Z). a negatív (N). sőt van olyan is, hogy "végtelen" bit (1 - Inflnily) vagy "nem szám" (NAN - Not A Number).
A könyv következő részében részletesen szólunk majd a 68881/82-es koprocesszorok lelkivilágáról.

Az "x" parancs használható paraméterezve is: "xd2". ekkor a D2 regiszter tartalmát tudjuk megváltoztatni. Az "x" és a "d2" közé nem kell space!

ZB - Zap BreakPoints - Töréspontok törlése.
Az elhelyezett töréspontokat törli.

RS - Read Sector - Szektor beolvasása a lemezegységről.
Paraméterként megadott lemezegységről (pl.: RS 0 - ekkor a DFO-ról) egy memória helyre (RAM ITR) beolvas szektorokat, ami lehet címke vagy cím. Meg kell adni. hogy melyik szektortól (DlSK FFR) mennyit olvasson be (LENGTH). Ha nincs lemez a meghajtóban "No disk in drive" hibaüzenettel leáll. Egy szektor mérete 512 byte. érdekességképp meg kell. hogy említsük a nullás szektor tartalmazza az un. "boot block"-ot. A gép boot-oláskor innen olvassa be a boot programot, ha az első pár byte után tudta azt azonosítani.

RT - Read Track - Track beolvasása lemezegységről.
Hasonlóan a RS parancshoz track-eket olvas be a memóriába, csak itt egy track 11 szektor.

WS - Write Sector / Szektor kiírása lemezegységre.
Hasonlóan a RS parancshoz memóriából a lemezegységre ír ki megadott számú szektort. Ha a lemez írásvédett "Write protected" hibaüzenettel leáll.

WT - Write Track - Track kiírása lemezegységre
Teljesen ugyanúgy működik, mint a WS parancs, csak track-et ír ki. vagyis 11 szektornyi adatot.

CC - Calculate Checksum - Ellenőrző összeg kiszámítása
A bootblock-nak van egy longword hosszúságú helye ahol egy úgynevezett ellenőrző összeg van. Ennek a kiszámítását oldja meg ez a parancs. az összeget felírja a lemezre. Paraméterként megadható a lemezegység száma, pl.: "CC 0". ekkor a DFO-ra vonatkozik.

E - Extern - Külső adatok betöltése
Az "extern" assembly direktívához kapcsolódik ez a parancs. Hatására a megjelölt helyekre (fordítás után) betölti a megfelelő file-okat.

> - Output - Kimenet beállítása
Az összes képernyőre kiírt szöveg átirányítható egy file-ba. Lehet PRT: is. ekkor a nyomtatóra küldi ki az adatokat.

? - **Calculate** -' Számológép.
Matematikai műveleteket végezhetünk egész számokkal. Az eredményt hexadecimálisán, decimálisán, négy ASCII karakterként és végül binárisan jeleníti meg. Használhatjuk egyszerű konverzióra is: ? "AI3CD".
A különböző számrendszerek jelölése a következő:
$ - **hexadecimális** (16-os) számrendszer, pl.: $C000
% - **bináris** (2-es) számrendszer, pl.: %11011
@ - **oktális** (8-as) számrendszer, pl.: @7
A decimális (10-es) számrendszernek itt nincs külön jelölése (a fordítás közben lesz!).

"text" - ASCII szöveg jelölése.

[- **Calculate Float** - Lebegőpontos számítás.
Hasonlóan működik mint a "?" parancs, de nem egész számokkal is számolhatunk, de ASCII szöveget természetesen itt nem használhatunk.

=R - **Custom Registers** / A Custom regiszterek listája.
Az s: könyvtárban lennie kell egy regsdata file-nak. ez tartalmazza a custom regiszterek leírását. Lehet paraméterezni is. pl.: "=r colorOO". ekkor a colorOO custom regiszter leírását írja ki.

Amiga + '=' -Aga Guide / Aga leírás betöltése.
Az s: könyvtárban AGA.GUIDE néven szerepelnie kell a leírásnak.

1.4.3 Ami a menükből kimaradt

AS - Activate Source - Forráslista aktiválása.
Az aktív forráslistát állítja be. nem kell belépnünk az editorba.

=C - **Colors** - Színek beállítása.
A képernyő színeit állíthatjuk be a saját ízlésünknek megfelelően.

=A - **Asm-One Location** - Az Asm-One helye a memóriában.
Az Asm-One program memóriában elfoglalt helyéről ad felvilágosítást.

CD - **Create Direcrory** - Alkönyvtár létrehozása.
A megadott nevű könyvtárat létrehozza az aktuális könyvtárban.

V - **Directory** - Aktuális könyvtár tartalma.
Paraméter nélkül kilistázza az aktuális útvonalon elérhető file-okat, paraméterrel megváltoztatja az aktuális útvonalat. A parent funkció a következő módon érhető el: "v /".

P - **Print** - Nyomtatás.
A megadott címkétől kiírja a képernyőre a forráslistát, de a kimenetet át lehet irányítani egy file-ba is, ami később kinyomtatható.

Y - **Execute** - Külső parancs végrehajtása
Saját ablakban egy megadott file-t elindít, a végén az Enter lenyomására vár. PL: "y dir".

BS - **BootBlock Simulator** - BootBlock szimulátor
Ha saját BoolBlock programunkat akarjuk tesztelni nem kell mindig reset-elni a gépet és boot-olni. E/, a funkció elintézi azt. hogy a programunk úgy lássa, mintha most boot-olna a gép.

WP - **Write Prefs** - Beállítások kimentése
A beállításokat elmenti az ENVARCiAsm-One.preffile-ba.

1.4.4 *Az editor*

Az editor az Asm-One egyik legjobban sikerült része. Nagyon gyors képernyőkezelése van. és nagyon sok kényelmi funkciója, kezdve az egyszerű keresésektől egészen a bonyolult definiálható makro funkciókig.

Ha belépünk az editorba a legalsó sorban feltűnik egy inverz sor tele számokkal és betűkkel. Ezek jelentése sorban a következő:

Line - melyik sorban van épp a kurzor
Col - melyik oszlopban
Bytes - ennyi byte-ból áll a forráslistánk
Free - szabad chip+fast memória / szabad chip memória kbyle-ban.

Ezulán öt darab kis mínuszjel van, melyek ilyen állapotban nem aktívak. Ezek jelentése balról jobbra haladva:

1. - ha nagy "A" betűt látunk ott akkor lefordított program van a memóriában (ha eltűnik akkor is lehet meg futtatni), ha kis "a" betűt látunk ott akkor már futtattuk a programot
2. - az utolsó mentés óta megváltozott a forráslista, az "u" (update) parancs ezt figyeli, ha ott a csillag akkor nem írja felül a file-t
3. - makró készítő művelet folyamatban
4. - blokk kijelölő művelet folyamatban
5. - passz, esetleg tudja valaki, hogy mi ez?

Time - az aktuális idő. csak akkor frissíti a program, ha megnyomunk egy billentyűt

Most pedig lássuk az editor menüit:

Block Mark - Blokk kezdete.
A kijelölendő blokk kezdete, ekkor jelenik meg lent a "B" betű.

Block Copy - Blokk másolása.
A kijelölt blokkot elteszi a memóriába úgy. hogy az eredetit meghagyja.

Block Cut - Blokk kivágása.
Mint az előző, de a blokkot ki is szedi a forráslistából. Vigyázat! Ha ESC-vel kilépünk az editorból a blokkunk elveszik!

Block Insert - Blokk beillesztése.
A kurzortól kezdve mind a Copy-val mind a Cut-tal kivágott blokkot beilleszti. Ez többször is megismételhető egymásután.

Block Fill - Ugyanaz mint a Block fnsert.

Block UnMark - Blokk kijelölés megszűntetése.
Az eddigi kijelölést megszünteti, ha esetleg volt már egy blokk a memóriában azt nem törli.

Block LowerCase - Blokk kisbetűre alakítása.
A kijelölt blokkon belül levő betűket kicseréli kisbetűkre

Block UpperCase - Blokk nagybetűre alakítása.
A kijelölt blokkon belül levő betűket kicseréli nagybetűkre

Block Rotate - Blokk forgatása.
A kijelölt blokkon belül a sorokat visszafele rendezi át. tehát ami legfelül volt az legalulra kerül

Block Register - Regiszterek kikeresése.
Megadja, hogy a blokkon belül mely regisztereket használtuk.

Block Write - A kijelölt blokk elmentése.
A kijelölt blokkot el lehet menteni egy file-ba.

Block Verticai Fill - Blokk beillesztése. *
Hasonló az Insert-hez. de a kurzor nem a beillesztett blokk végére kerül, hanem egy sorral lentebb, a következő blokk sorába (ha ez lehetséges).

Block Comment - A kijelölt blokk átalakítása megjegyzésre.
A blokkon belül minden sor elejére egy pontosvesszőt tesz. így ezek a sorok megjegyzésként kerülnek majd értelmezésre a fordítás során.

Block UnComment - A kijelölt blokk elejéről a pontosvesszők leszedése.
A blokkon belül minden sor elejéről - ha van ott pontosvessző - leszed egyet.

Search - Keresés a forráslistában.
Megadott szöveget keres, a kurzor sorától kezdve.

Search Forward - Következő előfordulás keresése.
Az előzőleg megadott szöveg következő előfordulását keresi.

Replace - Szó kicserélése másikra.
A megadott szövegei megkeresi és lecseréli egy másik szövegre. Ez is a kurzor sorától kezdi. Ha megtalálta akkcjtr a következő esetek közül választhatunk:

Yes - lecseréli és keresi a következőt
No - nem cseréli le és keresi a következőt
Last - lecseréli és nem folytatja tovább a keresést, kilép
Global - minden előfordulást lecserél
Abort - kilép

Replace Forward - Következő előfordulás keresése.
Ha megállítottuk a lecserélést, de tovább akarjuk folytatni akkor ezzel a funkcióval megtehetjük.

Delete Line - Sor törlése.
A kurzor sorában levő sort kitörli, de bekerül a memóriába. Olyan mintha blokk művelettel kijelöltük volna és Cut-tal elraktuk volna a memóriába. Block Instert művelettel újra berakható.

Set Marks - Megjelölt helyek beállítása.
Tíz olyan helyet tudunk beállítani, melyekre azután gyorsan oda tudunk ugrani. Nagyon kényelmes funkció.

Jump Marks - Megjelölt helyre ugrás.
Valamelyik megjelölt helyre tudunk ugrani.

Jump ;; - Ugrás a következő ;;-ra.
A következő olyan sorra ugrik amelyben két pontosvessző található.

Jump Line - Ugrás megadott sorra.
Megadott sorszámú sorrá ugrik.

Move Begin of Line - Sor elejére ugrás.

Move End of Line - Sor végére ugrás.

Move Page Up - Egy oldalnyit fel.

Move Page Down - Egy oldalnyit le.

Move Up 100 - 100 sornyit ugrik fel.

Move Down 100 - 100 sornyit ugrik le.

Move Top - A forráslista elejére ugrik.

Move Bottom — A forráslista végére ugrik.

Move Left Word - A bal oldali szó elejére ugrik.

Move Right Word - A jobb oldali szó elejére ugrik.

Make Macro - Makro készítés.
Ez egy nagyon kényelmes funkciója az editornak. Bármilyen billentyűzet kombinációkat eltárol és később azt vissza tudja játszani. Például, ha minden sort egy tabulátorral beljebb akarunk tolni akkor csak egyszer kell megtennünk, a többit ezzel a funkcióval pillanatok alatt megtehetjük. Ugyanez a funkció jelzi a makro készítés végét. A makro készítés alatt alul az M betű jelenik meg.

Do Macro - Makro elindítása
A már elkészített makrónkat tudjuk vele elindítani.
Az editor egyik kényelmi szolgáltatása a tabulátorok szabad beállítása. amit a legelső sorban kell elhelyezni, valahogy így:

Ekkor a tabulátor pozíciók a "T" betűknél lesznek.
Másik kényelmi szolgáltatás az egérrel való blokk kijelölés. Kétszer kell klikkelni az egér bal gombjával és már lehet is kijelölni. Újabb két klikk eltünteti a kijelölést.
Ha az assembler preferences részben be van kapcsolva az **Ali Errors** akkor a CTRL+E vagy Jobb Amiga+E billentyűzetkombinációval tudunk ugrani a következő hibára.

1.3.5 A debugger

A beépített debugger a programunk sorról sorra való nyomon követésére való.
Előbb-utóbb mindenkinek szüksége lesz erre a funkcióra. így hát alaposan ki fogjuk vesézni.
A debugger-be az "ad" paranccsal tudunk belépni, ha a fordítás sikeres volt.
A képernyő jobb oldalán megjelennek a regiszterek, ezek tartalma a futtatás során ha megváltozik, akkor az inverzbe vált. A legalsó sorban a soron következő utasítás jelenik meg. ennek a hatása fog megjelenni a regisztereken. Most pedig nézzük meg az egyes funkciók hatását:

Step One - Egy lépés.
A lefelé nyíl segítségével egy sort végrehajt a programból, azt amelyik inverz volt (ez a sor van legalul is). Szubrutinba ez nem lép be. csak végrehajtja azt!

Enter - Belépés szubrutinba.
A jobbra nyíl segítségével ha egy szubrutin hívás következik, akkor be is lép abba egyébként csak végrehajtja azt a sort, mint a lefele nyíl funkció.

Run - Elindítja a programot.
Az aktuális sortól kezdve elindítja a programot, csak a végén áll meg vagy ha BreakPoint-ot talál.

Step N - N darab sor végrehajtása.
Az aktuális sortól kezdve N darab sort hajt végre.

Skip Instruction - Utasítás átlépése.
A soron következő utasítást átlépi, tehát nem lesz hatása.

Run until Here - Futtatás míg megint ide nem ér.
Addig fut a program míg a program számláló megint ide nem ér, például ciklusok tesztelésére jó.

Animate - Állandó futtatás.
Futtatja a programot úgy, hogy a regisztereket nyomon tudjuk követni.

Edit Regs - Regiszterek átírása.
A megadott regiszter értékét át tudjuk írni.

Add Watch - új Watch létrehozása.
Olyan sorokat tudunk létrehozni (max. 8-at) amelyek a memória egy részét mutatják, annak minden változását. Ezek lehetnek Ascíi. String, Hex. Decimai, Binary vagy Pointer típusúak. A Pointer típusnál meg kell adni a mutató méretét és azt. hogy milyen típusú adatra mutat.

Del Watch - Watch törlése.
Valamelyik Watch sort törölhetjük.

Zap Watch - Minden Watch sor törlése.
Az összes eddigi Watch sort törölni tudjuk.

Jump Address - Ugrás címre.
Ugrást tudunk végrehajtani egy megadott címkére, de címet is megadhatunk.
Ha címet adunk meg akkor azt körültekintően tegyük!

Jump **Mark** - Ugrás kijelölése.
Ki tudjuk jelölni, hogy a programunk honnan folytatódjon tovább. Ezt érdemesebb használni, mint a Jump Address-t.

B.P. **Address** - TörésPont (BreakPoint) megadása.
Töréspontokat tudunk elhelyezni programunkban, ahol a futtatás megszakad. A töréspontot címkével vagy címmel adhatjuk meg. A Run paranccsal együtt használjuk.

B.P. Mark - Töréspont kijelölése.
Töréspontokat tudunk kijelölni úgy, hogy ráállunk a megfelelő sorra.

Zap Ali B.P. - Töréspontok törlése.
Az összes eddig kijelölt és megadott töréspontot törli.

Change Dx/FPx - Regiszterek cseréje.
Az adatregiszterek kijelzését megcseréli a koprocesszor regisztereire.

1.3.6 A monitor

A monitor a hexadecimális. ASCII, bináris és disassemble funkciókat foglalja magába. Ezekkel a memória tartalmát tudjuk megnézni és átírni.
A parancsok ismertetésénél már leírtuk, hogy hogyan lehet belépni ezekbe, most a monitor funkció menü parancsaival ismerkedűnk meg.

Disassemble - A mnemonikok visszafordítására vált át.
Hex Dump - Hexadecimális megjelenítésre vált át. -
ASCII Dump - ASCII megjelenítésre vált át.
Binary Dump - Bináris megjelenítésre vált át.
Jump **Address** - A megadott helyre ugrik
Last Address - Az ugrás előtti címre ugrik vissza.
Quick Jump - Hasonlóan a Jump Address-hez más címre ugorhatunk.
Set Mark - Három ugrási címet állíthatunk be
Jump Mark - A három ugrási cím valamelyikére ugorhatunk.
Set Start - Egy blokk elejéi állíthatjuk be
Set End - A blokk végét állíthatjuk be.
Savé Bin - A kijelölt blokkot elmenthetjük.

1.3.7 *Preferences*

A Preferences az a része a programnak, ahol a program működésére vonatkozó adatokat tudjuk beállítani. Ennek két része van: **environment** és assembler.

Az Environment beállítások:

ReqTooI Library
A program használja-e a ReqTools Library-t. Ennek használatával a program használata sokkal kényelmesebbé válik, pl.: file requester jelenik meg a file műveletek során. Ennek feltétele, hogy a Libs: -ben ott legyen a ReqTools.library file.

Savé Marks
A kimentett lbrráslisla elejére kimentse-e a beállításokat. Ez igen jó. de ha más programmal nézzük meg a forráslistát. kicsit zavaró lehet, hogy esetleg nem szövegnek ismeri fel a program. A beállítások mindössze 44 byte-ot foglalnak el.

Source. ASM
A file requester pattern sorába beírja-e azt, ami a Source Extension sornál van, pl.: "#?.ASM". Ez azt jelenti, hogy csak a .ASM kiterjesztésű file-ok jelennek meg a listában.

Update Check
Figyelje-e, hogy az utolsó update óta volt-e változtatás a forrás listában.

Printer Dump
Ha van printer akkor minden parancsot és üzenetet kinyomtat.

Resident Registers
Azt állítja, hogy a STegsdata file állandóan a memóriában legyen-e.

Close Workbench
Ha a Workbench képernyőn nem fut program akkor azt be lehet zárni. így némi chip memória felszabadul. Csak a program indításakor csukja be! Kilépéskor a Workbench képernyő újra megnyitásáról nem gondoskodik . a program, ez azt jelenti, hogy ha más program is fut amely külön screen-t használ akkor kilépéskor nem nyitja meg a Workbench screen-t. A rendszer persze gondoskodik arról, hogy ha kilépünk a másik programból azért a Workbench screen-t újra megkapjuk.

One Bitplane
Csak egy bitplane-t használ a program. így is felszabadul némi chip ram.

Parameters
A "PS" parancsot engedélyezi. A debugger rész is megkapja a paramétereket.

Default. Dir
Ez a könyvtár lesz a program indításakor az alapértelmezés szerinti könyvtár. A "v" parancs ezt fogja kilistázni.

BootUp
A program indításakor milyen .pref file-t használjon. Nálunk ez nem működik.

Source Extension
A Source .ASM kapcsoló ezt a sort rakja be pattern-neK a. nie request be

Select New ScreenMode
Kiválaszthatjuk, hogy a program milyen képernyőt használjon. Lehetőleg 640 vízszintes és 256 vagy 512 függőleges felbontású képernyőt válasszunk, mert a többit nem kezeli jól.

ASCII Only
A monitor funkciókban csak az **ASCII** karaktereket jelenítse-e meg. Ha ki van kapcsolva akkor az AMIGA speciális, nem **ASCII** karaktereit is megjeleníti.

Dis assembly
Ha be van kapcsolva a debugger legalsó sorában megjelenik az épp végrehajtásra váró utasítás.

Show Source
A debugger részben a forráslistát avagy a disassembly listát mutassa-e. Ebben a verzióban ez nem igazán működik, ne használjuk, mert teljesen más programot akar debug-golni.

Enable/Permit
Ha programunkban **Forbid** vagy **Disable Exec.library** hívást használunk és ezeket nem zárjuk le. az Asm-One ezután furcsán viselkedhet. Ennek elkerülésére beépítettek egy **Enable/Permit Exec.library** hívást. így már biztos, hogy nem lesz baj.

Libcalls Dec
Az ID (Insert Disassemble) parancsnál a **JSR** -xx(A6) utasítást bekapcsolt állapotban decimális értékben írja ki. A könnyebb áttekinthetőség miatt.

AutóIdent
Enter lenyomása után ha ki van kapcsolva mindig a sor elejére ugrik a kurzor, bekapcsolva az első betű alá. Nagyon kényelmes funkció, érdemes használni!

Extended ReqTools
A szerkesztőben használjon-e bővített Requester-eket vagy ne. Ki lehet próbálni például a Search funkció működésénél.

Line Numbers
A sorszámozást kapcsolja ki és be a szerkesztőben.

CTRL Up/Down
Bekapcsolva a CTRL+Up/Down a forrás legelejére és legvégére ugrik, kikapcsolva nincs hatása.

KeepX
Bekapcsolva a szerkesztőben a kurzor vízszintes pozícióját megtartja, ha lehet, kikapcsolva mindig a rövidebb sor végére ugrik.

Az assembler beállítások:

Rescue
Programunk befejeztével automatikusan visszaállítja a copper listát, ha az nem lenne jó.

Levél 7
Külső 7-es szintű megszakítással programunk leállítható.

NumLock
Ha be van kapcsolva a tízes numerikus billentyűk nem számokat írnak, hanem a rajtuk levő funkciókat látják el.

AutoAlloc
Automatikusan lefoglalja a programunknak a helyet. Az assembly direktíváknál tárgyaljuk részletesebben.

Debug
A debugger számára állít elő adatokat. Ha ez be van kapcsolva, akkor gyorsabban belép a debugger részbe, viszont több memóriát igenyel.

List File
Fordításkor kilistázza a fordítás eredményeit.

Paging
Ha a List File be van kapcsolva akkor be lehet állítani, hogy oldalanként megszámozza-e az oldalakat.

Halt File
Ha a List File be van kapcsolva akkor be lehet állítani, hogy oldalanként megálljon-e. Billentyűlenyomással folytatja.

Ali Errors
Ha nincs bekapcsolva akkor az első hibánál leáll a fordítás. Ha be van kapcsolva akkor van értelme az editorban az CTRL+E funkciónak.

Progress Indicator
Ha be van kapcsolva akkor a Progress by Line szerint fordításkor kiírja, hogy melyik sornál tart vagy hány százaléknál.

Progress by Line
Ha be van kapcsolva akkor sorszámlálót, ha nincs, akkor százalékszámlálót használ.

DS Clear
Ha be van kapcsolva akkor a **BSS** típusú szekciónál a DS-sel lefoglalt területet feltölti nullával.

Label:
Ha be van kapcsolva akkor a címkék után kettőspontot kell rakni.

UCase=LCase
Ha be van kapcsolva akkor a kis- és nagybetűket azonosnak veszi.

; **Comment**

Ha be van kapcsolva akkor a következő sorra hibát jelez:

```
start: moue.l #2,data          +2
```

Ekkor a megjegyzés csak pontosvesszővel kezdődhet.

ProcessorWarn
Figyelmeztet, ha nem alap 680Ö0-es utasítást használunk.
PL: **moue.l #1,(aQ,dQ.I*4)** Ebben a verzióban hibásan működik.

FPU Present
Ha be van kapcsolva akkor úgy veszi a fordító, hogy van a gépben koprocesszor és annak megfelelően fordít és írja ki a regisztereket.

68020++ Odd Data
Páratlan címen is elhelyezhetünk long vagy word adatokat. 68020+ processzorok páratlan címről is tudnak word vagy long adatot olvasni.

68851 Present
Van a gépben **MMU.** Az alsó gadget a processzor típusát váltja, ami azt jelenti, hogy annak a processzornak az utasításkészletét fordítja. Ellenkező esetben csak felhívja a figyelmünket, hogy nagyobb processzor kell a futtatáshoz.

1.3.8 Az assembly direktívák

Az assembler kezel olyan „utasításokat" melyek nem a 68000-es utasításai, de szorosan összefüggenék azzal, esetleg az operációs rendszerrel. Ezeket asselmbly direktíváknak hívják.
Nézzük meg ezek jelentését, néhány fontosat részletezve:

SECTION- [címke] SECTION név,[típus+_C/_F/_P]

Az ezután következő program kód. adat típusát tudjuk beállítani.

```
Pl: section program,code_c  ;kód a chip ram-ba kerül
    section adatok,data_f   ;adatok a fást ram-ba kerülnek
    section adatok2,bss_p   ;adatok, ha uan fást akkor oda, ha
                            ;níncs akkor a chip ram-ba kerülnek
```

Ha _f helyet adunk meg. de nincs fást ram akkor a futtatható program nem fog elindulni, ezért érdemesebb a _p (public) memóriatipust használni. A **BSS** egy különleges adatfajta, csak a méretét kell megadni. Csak futtatáskor kerül tényleges lefoglalásra. Ha 100K területet akarunk lefoglalni és a programunk kódja csak kb. 5K akkor a futtatható állomány is csak kb. 5K lesz. míg. ha dala típusú lenne akkor kb. 1O5K lenne.

RORG - [címke] RORG cím Relatív programkezdet.

ORG - [címke] ORG cím Abszolút programkezdet.
Pl.: ORG $100000

LOAD - [címke] LOAD cím Abszolút betöltési hely. az ORG-al együtt használatos.
Pl.: LOAD $100000

OFFSET - [címke] OFFSET cím Offsel definíció.
PL:

```
Def:    OFFSET  $100000
Datál: dc.l     0
Gata2: dc.l     1
        ENDOFF
```

Ekkor a DataO címe a $100000. a Datál címe pedig $100004.

ENDOFF Ofl'set definíció vége

END Forráslista vége. az Asm-One-ban nem muszáj kiírni, a SasC fordítójának viszont igen.

BASERĖG - [címke] BASEREG cirnke2.Ax Bázisregiszter meghatározása.
Lássunk rá egy példát:

```
Start: moue.l  #DataO,a6
       moue.l  $4(a6),dO
       rts
DataO: dc.l    1
Datál: dc.l    2
```

Helyette **BASEREG** használatával átláthatóbb programot készíthetünk, de a fordító az előző programot fogja generálni:

```
Start:  BRSEREG  DataO,a6
        moue.l   #DataO,a6
        moue.l   data1(a6),dO
        rts
DataO: dc.l      1
Datál: dc.l      2
```

DC - [címke] DC.x adat(ok).
Segítségével code vagy data területen tudunk adatokat létrehozni:

```
Datál: dc.b     1,2,3,4,$10,"a"
Data2: dc.ui  $21Q7,$fffe,$0180,$0Q0Q
Data3: dc.l     $1,20
```

DCB - [címke] DCB.x hossz.adat.
Segítségével "hossz" darab "adat"-ot tudunk lefoglalni:

```
Datál: dcb.b   10,$ff          ;10 darab $ff byte
```

DS - [címke] DS.x hossz.
Segítségével bss területen tudunk helyet foglalni:

```
Datál: ds.b    10000           ;10000 darab byte hely
```

BLK - [címke] BLK.x hossz.adat.
Segítségével "hossz" darab "adat"-ot tudunk lefoglalni:

```
Datál: blk.b   100,0           ;100 darab 0 byte
Data2: blk.l   100,0           ;100 darab 0 long
```

Ugyanaz mint a DCB.

DR - [címke] DR.x cimke.
Relatív adat foglalása:

```
a:      dr.b    c              ;értéke 2 lesz
b:      dr.b    c              ;értéke 1 lesz
c:      dr.b    c              ;értéke 0 lesz
d:      dr.b    c              ;értéke 255 uagyis -1 lesz
```

EQU - azonosító EQU kifejezés.
Értékadás, a könnyebb átláthatóság kedvéért érdemes használni. Pl.:

```
COLORO EQU $DFF180

STRRT:  M0UE.U) #$OFFF,COLORO
```

értékét csak egyszer állíthatjuk be.

SET - azonosító SET kifejezés.
Ugyanúgy működik, mint az EQU, de értékét többször is beállíthatjuk.

EgUR - azonosító EQUR Ax.

Valamelyik címregisztert helyettesíthetjük az alábbi módon:

```
MOUE.L  (H5)+,D0
```

Helyettesítve:

```
DRTR1   EQUR    R5
        MOUE.L  (DRTH1)+,DO
```

REG - azonosító REG regiszterlista.
Regisztereket lehet helyettesíteni:

```
Datas   REG D0-D7
        MOUEM.L Datas,-(SP)
        ..
        ..
        MOUEM.L (SP)+,Datas
        RTS
Vagy:
Regs    REGDO/D1/R1
```

RS - [címke] RS érték.
Általában könyvtárrutinók címeinek kiszámításához használjuk.
PL:

```
                RSRESET
                RSSET       48
InitCode:       RS.B        -6
InitStruct:     RS.B        -6
MakeLibrary:    RS.B        -6
MakeFunctions:  RS.B        -6
```

RSRESET - [címke] RSRESET Az RS számlálóját nullázza. Lásd az előző példát.

RSSET - [címke] RSSET érték.
Az RS számlálóját a megadott értékre állítja. Lásd az előző példát.

MACRO - címke MACRO.
Makro definíció kezdete. Ne tévesszük össze az editor makro funkciójával! A makrót, mint parancsot lehet meghívni paraméterekkel, melyekre \1 \2 stb jelöléssel hivatkozhatunk. PL:

```
CRLLEKEC:       MRCRO
                MOUE.L  $4,R6
                JSR     \1(R6)
                ENDM

STRRT:          MOUE.L  #$QBRDCBDE,D7
                CHLLEHEC -108
                RTS
```

NARG
Ez nem teljesen assembly direktíva, hanem egy változó. Ebből megtudhatjuk, hogy például egy makrónak hány paramétert adtunk meg.
PL:

```
PROBR:  MRCRO
        PRINTLI NfIRG
        EHDm

STRRT:  PROBR DO 1
```

ENDM Makro definíció vége.

MEXIT Kilépés a makróból. Elágazásoknál kilépésre használjuk. Pl.:

```
EHH:    MHCRO
        IFC\1,"
        DC.B$D9
        MEKIT
        ENDC
        FRIL 'Hiba! EKK-nek nincs argumentuma!'
        ENDM
```

A példa egy Z80 fordítóból való. csak akkor fordít $d9-et, ha a makrónak nincs argumentuma.

CMEXIT
Visszatérés rekurzív makro hívásból. Sehol sem láttam még erre példát, csak az eredeti dokumentáció szerint ez a jelentése.

REPT - [címke] REPT szám.
Blokk ismétlésére való, nézzünk is egy példát:

```
REPT 10
nop

ENDR
```

így tíz darab NOP-ot fordít.

ENDR A REPT lezárására használjuk.

CNOP
Megadhatjuk, hogy a fordítás milyen címen folytatódjon tovább. Kél argumentumot kell megadnunk. A~ második jelenti, hogy ilyen számmal osztható címen folytatódjon, az első pedig egy eltolás.
Példák:
CNOP 0.4 - a következő' néggyel osztható cím
CNOP 0.2 - a következő páros cím. ugyanaz mint az EVÉN
CNOP 1.8 - a következő nyolccal osztható cím után egyel

ALIGN Ugyanaz mint a CNOP. csak a kompatibilitás kedvéért.

EVÉN
A fordítást páros címen folytatja. A "Word at Odd Address" hibánál használhatjuk.

ODD A fordítást páratlan címen folytatja.

IFE9 - [címke] IFEQ kifejezés.
EQual / egyenlő - igaz. ha a kifejezés ériéke nulla. Pl.:

```
X:      EQU 5

        IFEQ    (X-5)
        PRINTT  "HZ K ÉRTÉKE 5."
        ENDC
```

Viszont bug-os a fordító:

```
IFEQ    kep\l
IFEQ    gep\l
```

Az ilyen és ezekhez hasonló hárombetűs szavakra igencsak furcsa dolgokat tud művelni.

IFNE - [címke] IFNE kifejezés.
Not Equal / nem egyelő - igaz. ha a kifejezés értéke nem nulla.

IFGT - [címke] IFGT kifejezés.
Grealer Than / nagyobb mint - igaz. ha a kifejezés értéke nagyobb mint nulla.

IFGE - [címke] IFGE kifejezés.
Great or Equal / nagyobb vagy egyenlő - igaz. ha a kifejezés értéke nagyobb vagy egyenlő mint nulla.

IFLT - [címke] IFLT kifejezés.
Less Than / kisebb mint - igaz. ha a kifejezés értéke kisebb mint nulla.

IFLE - [címke] IFLE kifejezés.
Less or Equal / kisebb vagy egyenlő - igaz. ha a kifejezés értéke kisebb vagy egyenlő mint nulla.

IF - [címke] IF kifejezés. Igaz, ha a kifejezés nem nulla.

IFC - [címke] IFC szövegl .szöveg2
Igaz, ha a kél szöveg egyenlő. Pl.:

```
IFC     \1,'DB'
PRINTT  "R PHRRMÉTER R DO REGISZTER"
ENDC
```

IFNC - [címke] IFNC szövegl.szöveg2 Igaz. ha a két szöveg nem egyenlő.

IFD - [címke] IFD azonosító.
Igaz. ha az azonosító már definiálva van. Általában az include file-ok használják. Pl.:

```
DEBUG:  EQU 1
        IFD DEBUG
        ...
        ENDC
```

IFND - [címke] IFND azonosító. Igaz. ha az azonosító ninrs definiálva.
IFB - [címke] IFB kifejezés Igaz. ha a kifejezés üres. PL:

```
PELDH: MRCRO
       IFB \1
       PRINTT "üres."
       ENDC
       ENDM
```

START: PÉLDA

IFNB - [címke] IFNB kifejezés. Igaz. ha a kifejezés nem üres.

IFI - [címke] IF1. Igaz. ha a fordító az első fázisban van.

IF2 - [címke] IF2. Igaz, ha'a fordító a második fázisban van.

ELSE - [címke]. IF feltételnek "egyébként" ágat tudunk biztosítani.

ENDC - [címke]. ENDC IF feltétel blokk vége.

PAGE Oldalanként megálljon és számozza az oldalakat.

NOPAGE Oldalanként ne álljon meg.

LIST Legyen listázás.

NOLIST Ne legyen listázás.

LLEN - LLEN szám. Egy sor hosszát lehet beállítani. 60 és 132 kozott kell lennie.

PLEN - PLEN szám. Egy lap hosszát lehet beállítani. 20 és 100 közölt kell lennie.

SPC - SPC szám. Megadott számú üres sort lehet kiírni.

TTL - TTL szöveg. Megadhatjuk a programunk nevét.

FAIL - FAIL száveg. A fordítást megszakíthatjuk.

MASK2 - Nincs semmi hatása, csak a más fordítókkal való kompatibilitás megőrzése érdekében van benne a programban.

PRINTT - PRINTT szöveg. Az utána írt szöveget fordítás közben kiírja.

PRINTV - PRINTV kifejezés. Az utána álló kifejezés értékét kiírja.

XDEF - XDEF címke.
Címkét definiálhatunk, ha a programunkat összelinkeljük más programmal, akkor az látni fogja ezt a címkét. Később mutatunk példát, hogy a SasC-hez hogyan lehet hozzálinkelni gépi kódú programot.

XREF - XREF címke.
Címkét definiálhatunk, mely másik programban van.

ENTRY - ENTRY címke.
Ugyanaz mint az XDEF. csak a kompatibilitás megőrzése érdekében.

EXTRN - EXTRN címke.
Ugyanaz mint az XREF, csak a kompatibilitás megőrzése érdekében.

GLOBAL - GLOBAL címke.
Ugyanaz mint az XDEF. csak a kompatibilitás megőrzése érdekében.

JUMPPTR - JUMPPTR címke.
Programunk belépési pontját határozhatjuk meg vele. ha nem a legelső soron akarjuk, hogy induljon.

INCBIN- [címke].
INCBIN file, Bináris file-t tudunk betölteni. Pl.: SCREEN1: INCBIN "Work:pictures/hattérl .raw"

IMAGE - [címke] IMAGE file. Ugyanaz, mint az INCBIN. Kompatibilitás.

INCLUDE - 1NCLUDE file.
Include file-ok betöltésére való. Pl.: INCBIN "Exec/Macros.i"

INCDIR - INCDIR útvonal.
Az include file-ok útvonala. PL: INCDIR "Work:Include/"

>EXTERN - >EXTERN "file",címke.
Az előre lefoglalt helyekre betólt flle-okat. az incbin kényelmesebb. PL:

```
>EHTERN "WORKrSOURCES/PICl.RRW'.SCREENQ
..
..
SCREENO: BLK.B 81920,0
```

IDNT szöveg Ugyanaz mint a TTL.

AUTÓ - AUTÓ parancsok.
A fordítás végén automatikusan végrehajt parancsokat. Az Enter-t a \ jelzi. PL:

```
      MUIU    riu riLiunt i .nmmumnw
DRTR: BLK.B   81920,0
```

Ha valakit igazán érdekel a makro készítés minden csínja-bínja az nézze meg a SasC include könyvtárában az Exec/macors.i flle-t. Abban rengeteg makro készítési trükk van, nekünk is sok érdekességgel szolgált.

1.3.9 Hogyan optimalizáljunk?

Végül pedig szeretnénk az olvasónak néhány hasznos tanáccsal szolgálni, hogyan írjunk minél gyorsabb programot, és olyat, amely minden gépen működik:

- CLR.L DO helyett használjuk a MOVEQ #O.DO utasítást, gyorsabb.
- kettő hatványaink (2,4,8,16 std.) mul szorzása helyett használjunk eltolást, pl.: MULU #8,D0 helyett ASL.L #3,D0.
- kettővel való szorzás esetén: ADD.W DO.DO.
- ugyanígy kettő hatványainak osztása helyett jobbra való eltolást használjunk, pl.: DIVS #16,D5 helyett ASR.L #4,D5.
- ha szorzónk nem kettő hatványa akkor használjunk szorzó táblázatot, persze csak akkor, ha a táblázat mérete megengedi. Például a mulu #40.dO helyett használhatjuk a következőket, ha 68000-es processzorunk van és dO csak 0 és 255 között lehet:

```
moue.l #mtable,aO  ; a szorzó tábla helye a memóriában
add.ut  dl,dl  ; a word értékek miatt szorozzuk kettöuel
moue.ui O(aO,d1 .Lu),d1 ; di-be kerül a szorzott érték
```

Ha 68020+ processzorunk van, akkor egyszerűbb a helyzet:

```
moue.l #mtable,aO
moue.w e(aB,d1.w*2),d1
mtable:  dc.w 0000,0040,0080,0120,0160,0200,0240,0Z80 ...
```

- ne használjunk jsr és jmp utasításokat, helyette a bsr.s és bra.s utasításokat használjuk, így a programunk a memóriában bárhova áthelyezhető (és gyorsabb is!).
- ha egy rutint sokszor kell meghívni, de van még szabad regiszter akkor - mint az előző példában - címet csak egyszer olvassunk be és tároljuk el valamelyik nem használt regiszterben.
- NE használjunk önmódosító kódot, ha nagyon muszáj akkor is csak abban az esetben ha az utasítás cache-t töröltük és kikapcsoltuk! Ez 68000-es esetében nem gond (mert nincs is utasítás cache), de 68020-on már nem biztos, hogy működne a programunk!
- sose használjuk a TAS utasítást.
- könyvtár rutinok hívásánál mindig az A6 regiszter tartalmazza a báziscímet, pl.: jsr -108(a6).
- Ne használjuk egy mutató felső 8 bitjét adat tárolásra, mert ez csak 68000-es esetén működött, a 24 bit-es címbusz miatt, 68020+ processzoroknál már 32 bites.

- Ilyet se használjunk, az előző 24 bit-es címbusz miatt: MOVE.W #$0.$ABDFF188. Ez csak 68000-es processzoron működik, címnek a $dffl 88-as címet látja, a többit egyszerűen elhagyja.
- Ne használjuk a CLR utasítást hardware regiszter torlésére (pl CLR.W $DFF188). helyette MOVE.x #O,cím utasítást használjuk
- Ne használjuk a move SR.xx utasítást. Helyette könyvtárrutinnal kérdezzük le az állapotregisztert (exec.library / GetCCO).
- 68020+ processzoroknál a megszakítási vektorok nem biztos, hogy a nulla címtől kezdődnek, ez a VBR regisztertől függ. Ellenőrizzük le!
- A curslom regiszternél ha egy bit nem használt akkor azt nullázzuk ki, hogy újabb fejlesztésű rendszereken is működjön a programunk.
- A custom regisztereket csak működésüknek megfelelően használjuk, a csak olvasható regisztereket ne írjuk és a csak irhátokat ne olvassuk.
- Könyvtár rutin hívása után ne hagyatkozzunk az állapotregiszterek tartalmára, mindig teszteljük le a DO regisztert.

1.4 Első programunk

Most pedig elérkeztünk a könyv ama részéhez, ahol fejest ugrunk a programozás rejtelmeibe. Mint minden jóravaló könyv, mi is az alábbi C nyelvű programmal kezdjük:

```
/* Elsó programunk */

#include<stdio.h>

uoid main(uoid)
{
    printfC'Hello ujorld!\n");
}
```

Ez eddig nem is lenne olyan túl bonyolult, de hogy megnehezítsük a kedves olvasó dolgát, csinálunk egy hasonló programot assembly nyelven is:

```
Start:    mouem.l   dO-d7/aO-a6,-(sp)   ;elmentjük az összes
                                          regisztert
          moue.l    $4,a6               ;az e«ec library bázis-
                                          címe az H6-ba
          moue.l    #DosName,a1         ;mutató a dos.library
                                          szöuegre, uagyis
                                          annak a címe
          moue.l    #Q,dO               ;minimális uerziószám
          jsr       -552(a6)            ;OpenLibrary
          tst.l     dO                  ;DO-ba kapjuk a bázis-
                                          címet, ha 0 akkor
                                          nem sikerült megnyitni
          beq.s     Uege
          moue.l    dO,DosBase          ;sikerült, a báziscímet
                                          elrakjuk

          moue.l    Dos8ase,a6          ;R6-ba a dos.library
                                          báziscime, ez a fiigg-
                                          uény már abból lesz!!
          moue.l    #ConName,d1         ;mutató a ConName-re
          moue.l    #1006,d2            ;MODE_NEUJFILE
          jsr       -30(a6)             ;0pen
          tst.l     dO                  ;D0-ba kapunk egy
                                          mutatót, a file mutatót
          beą.s     CloseDos            ;nem sikerült, kilépünk,
                                          de a dos.library-t
                                          le kell zárni!!
```

```
                moue.l    dO.File          ;siker! Itt már meg-
                                            nyílt egy ablak!

                moue.l    DosBase,a6       ;nem muszáj, mert
                                            nem uáltozott meg,
                                            de azért biztos ami
                                            biztos
                moue.l    File,dl
                moue.l    #Te«t,d2         ;mutató a Tent-re,
                                            uagyis a TeKt címe a
                                            Di-be
                moue.l    #EndTeKt-Text,d3 ;a kiírandó szöueg
                                            hossza
                jsr       -48(a6)          ;Write

;CloseCon:      moue.l    DosBase, a6      ;R6-ba a dos.library
                                            báziscíme
;               moue.l    File,dl          ;D1-be a File
;               jsr       -36(a6)          ;Close

CloseDos:       moue.l    $4,a6            ;enec báziscime
                moue.l    DosBase,ai       ;a dos.library bázis-
                                            címe az fli-be
                jsr       -414(a6)         ;CloseLibrary
Uege:           mouem.l   (sp)+,d0-d7/afl-a6 ;uisszaállítjuk az
                                            összes regisztert
                rts                        ;kilépés

File:           dc.l      0
DosBase:        dc.l      0
DosName:        dc.b      "dos.library",0  ;uégig kisbetű és
                                            B-ual kell lezárni!
ConName:        dc.b      "C0N:0/0/200/1 OO/Első programunk/
                          HUT0/CL0SE",0     ;console
TeKt:           dc.b "Hello world!",13,18,u
EndTeKt:
```

Akkor jöjjön a magyarázat! A program elején gyorsan-frissen meg is nyitjuk a dos.library-t. De mi is az a library?! Ha a legegyszerűbben akarnánk megfogalmazni, akkor rutingyűjteménynek neveznénk. Minden hbrary-nek van egy speciális szakterülete, pl.: a graphics.library a grafikai dolgokkal foglalkozik, a dos a lemezműveletekkel, és így tovább. Vannak olyan library-k melyek a Kickstart ROM-ban vannak elhelyezve (dos. graphics. exec. intuition, gadtools, stb.) és vannak olyanok, melyek a háttértárolón csücsülnek a lihs- könyvtárban. A ROM-ban elhelyezett library-knek megvan az a hátrányuk, hogy azokat csak a ROM kicserélésével tudjuk frissebb verzióra cserélni, míg a lemezen levők helyett újabbat veszünk fel. Ahhoz, hogy ezeket használni tudjuk meg kell nyitnunk őket. Ezt az exec.library-ben levő Open-

Library függvénnyel tudjuk megtenni, ami ha sikerül akkor visszakapjuk a library úgynevezett "báziscímét". De ha az exec is egy library, akkor hogy tudjuk azt is megnyitni?! *Az* Amiga készítői úgy oldották ezt meg. hogy az exec.library mindig nyitva van. már csak a báziscímét kellene tudnunk. Általában az Amigában semmi sincs konkrét helyen de azt biztosra vehetjük, hogy ez mindig ott lesz. nevezetesen a 4. memóriacímen, longword nagyságban. Tehát MOVE.L $4,A6 és már meg is van az exec.library báziscíme. Hogy tovább léphessünk, bepillantunk a könyvtár bázisának lelkivilágába.

C nyelven a Library: (exec/libraries.h file)

```
struct Library {
        struct      Node lib_Node;
        UBVTE       lib_Flags;
        UBVTE       lib_pad;
        UWORD       lib_NegSize;  /* byte-ok száma a library előtt •/
        UWORD       Mb_PosSize;   /* byte-ok száma a library után */
        UWORD       lib_Uersion;  /* major uerziószám •/
        UWORD       líb_Reuision; /• minor uerziószám*/
        HPTR        Nb_IdString;  /* RSCII azonosító */
        ULONG       lib_Sum;      /* ellenőrzőösszeg */
        UWORD       Mb_OpenCnt;   /* jelenlegi megnyitások száma */
Í;
```

Kicsit érthetőbben: (Structure.offsets file)

Library:

$0022	34	sizeof(Library)	
$0000	0	14	lib_Node
$000e	14	1	lib_Flags
$000f	15	1	lib_pad
$0010	16	2	lib_NegSize
$0012	18	2	lib_PosSize
$0014	20	2	lib_Version
$0016	22	2	lib_Revision
$0018	24	4	HbJdString
$001c	28	4	lib_Sum
$0020	32	2	lib_OpenCnt

A 4-es memóriacímen levő érték egy ilyen struktúrára mutat (ez egy mutató), a lib_Node legelső byte-jára. Nézzük meg mi is van a 4-es címen, például így : "h 4", persze Asm-One-nal. Ha A1200-es gépünk ésn/vagy 3.0-ás Kickstart-unk van, akkor a megjelenő első négy byte feltehetőleg ez, vagy hasonló: 00 00 14 C4 . Ez azt jelenti, hogy a $14C4 címen van az exec library bázisa. Most nézzük meg, mi van a $14C4 címen (h $14C4)! Számunkra ami most még kézzelfogható, az a libJVersion és a lib_Revision.

Ez a kettő a library verziószáma, amit az alábbi dos paranccsal meg is tudhatunk: "version exec.library". Eredményül az alábbi sort kapjuk: "exec.library 39.47" (csak 3.0-ás Kickstart!) Most próbáljuk ki ezt: "h $14c4+$14" (az előző táblázatból!). 00 27 és utána 00 2F, ez két word hexadecimális formában, decimálisán 39 és 47. Nocsak, az exec library verziószámai! Nem is olyan bonyolult! Ha beírjuk, hogy "d $14c4" és elindulunk visszafelé akkor rengeteg JMP $F8xxxx utasítást látunk.

Ezek ugranak a tényleges rutinra. Azért JMP, mert ROM és abszolút helyen vannak, de mi ne használjuk azt, mert ezek a JMP értékek Kickstartonként változnak. Ezek a rutinok RTS utasítással vannak lezárva, a JMP-re egy címhez viszonyítva kell ugranunk, ezért ezt csak JSR utasítással tudjuk megtenni, valahogy így: JSR -552(A6). Ezt a -552 értéket az LVO3.0 könyvtárban levő execjib.i'file-ból néztem ki. és az OpenLibrary eltolási értéke. Az AutoDocs leírásból vagy az exec library leírásánál megtalálhatjuk az egyes rutinok paraméterezését, vagyis melyik regiszterbe mi kell.

Az OpenLibrary-nek az Al-be, a library nevére és DO-ba mutató minimális verziószám, amit még megnyithat.

Ennyi kitérő után nézzük tovább a programot! Miután meghívtuk az OpenLinbrary függvényt, meg kell vizsgálnunk, hogy sikerült-e megnyitni. Ha nem akkor kilépünk a programból. Az. hogy nem sikerül megnyitni azért lehet, mert a verziószám nem jó (túl régi) vagy esetleg ilyen library-t nem talál az operációs rendszer. Ha sikerült megnyitnunk, akkor a DO-ban levő értéket eltároljuk, és az A6 regiszterbe is azt rakjuk (az Asm-One fejezetben leírtuk, hogy a library-k báziscíme mindig az A6-ban legyen).

Ezután az a dos library-ben levő Open függvénnyel megnyitunk egy Console ablakot, erre fogjuk kiírni a szöveget. Az Open rutint egyébként inkább flle-ok megnyitására vagy létrehozására használjuk. Meg kell vizsgálnunk, hogy az előző művelet sikerült-e? Ha nem, akkor kilépünk, de a már megnyitott dos library-t le kell zárnunk. Ha sikerült a Console ablakot megnyitni akkor már csak a szövegünket írjuk ki. és kész is vagyunk. A szöveg kiírása a Write függvénnyel történik, amit szintén file műveletekhez használunk, de a Console-t tekinthetjük egy file-nak is, így most abba írunk.

Miután ez készen van. be kellene zárni az ablakot, de akkor az ablak csak egy villanásra jelenne meg. így ezt a rész kihagytuk, pontosabban csak megjegyzésként raktuk a programba, de hogy ez így működjön, a Console paramétereihez be kellett írnunk, hogy AUTO és CLOSE.

A CLOSE azt jelenti, hogy az ablaknak legyen bezáró gadget-je, az AUTÓ pedig azt. hogy az ablak bezárását a rendszer *vég<i7Xt* el (ha az AUTO-t kihagynánk, az ablakot nem tudnánk bezárni, hiába van ott a bezáró gadget).

Ezután a programunk már csak a CloseLibrary rutint hívja meg. az Alben a library báziscímével.

Meg kell jegyeznünk, hogy az Exec. az Intuition és a Graphics library-nak a bázis része nem igazán olyan mint a többi library-é, tartalmaznak még rengeteg extra adatot is. Ebből adódóan, ha C-ben akarjuk megnyitni valamelyiket, akkor típuskonverziót kell alkalmaznunk. PL:

```
Definíciója· struct IntuitionBase *IntuitíonBase;

Megnyitása: IntuitionBase=(struct IntuitíonBase *)OpenLib-
            raryC'intuition.library ",0);

Bezárása: CloseLibrary((struct Librapy *)IntuitionBase);
```

Ez azért kell. mert az OpenLibrary egy Library struktúrára mutató pointert ad vissza, a CloseLibrary pedig Library struktúrára mutató pointert vár.

A lemezmellékleten ehhez a fejezethez találunk még két programot, melyek ugyanazt csinálják, de az egyik C. a másik pedig assembly nyelvű A program csak 2 O-ás Kickstart-tól működik, a dos library CLI függvényét használva lekérdezzük, hogy Cli-ből lett-e indítva a program. Ha igen. akkor a Write függvénynek megkeressük a kimeneti egységét (StandardOutput). és arra írjuk ki a szöveget.

1.5 Ablakok és képernyők

1.5,1. Az ablakok

Ebben a fejezetben megnézzük, hogy hogyan is lehet kirakni egy ablakot az Amiga képernyőjére, valamint hogyan csinálhatunk tetszőleges képernyőt magunknak, amire aztán tetszőleges ablakot nyithatunk.

De vágjunk bele. Mint már említettük, a képernyők és egyéb feladatokért az Intuition könyvtár a felelős. Hogy pontosan mit csinál, azt most megnézzük. Annyit azonban tisztázzunk, hogy ahhoz, hogy egy ablakot nyithassunk, létezni kell egy már nyitott **screen-nek,** mivel ablakot csak már nyitott screenre nyithatunk. Az ablakjainkat kétféle screenre nyithatjuk, az egyik fajta, amikor tetszőleges képernyőre nyitjuk, tehát saját képernyőnkre **CUSTOM SCREEN,** valamint mikor a rendszer feléledésekor létrejövő **WORKBENCH SCREEN-re.** A 2.0-ás gépektől további képernyőkre nyithatunk, az úgynevezett **PUB SCRENN-re.** ami a nevéből is kitűnik, hogy nyitott, már létező ún. nyilvános képernyőt takar, (ilyen pl., a DirectorOpus-é is). A nyitandó abalakunk mindég olyan tulajdonságokkal bír, amelyeket a képernyőtől örökölt. Ezért fontos megválasztanunk a képernyő típusát. Hiszen több színt nem használhatunk, mint amennyit a screen örökül hagyott ránk. Ugyanez a helyzet például a felbontással is, ami nem lehet nagyobb mint amilyen a screené. Ha tehát a Workbenchre nyitjuk az ablakunkat, ami kényelmes, hiszen a fontok, a színek, minden a rendelkezésünkre áll. meg kell elégednünk azzal, amit a Workbenc örökül hagyott ránk. Ha azonban több színt, vagy más fontot (betűtípust) akarunk használni, és ezt esetleg nagyobb felbontásban is szeretnénk tenni, mint mondjuk a Workbench. akkor nyissunk saját képernyőt. Ha azonban egy programhoz akarunk kiegészítést írni. például a DirOpus-hoz. akkor használhatjuk annak a beálításait. ha a screen-jére rátelepszünk, így programunk mindig olyan lesz. mint amilyen az a program amelyikre rátelepedett. Azonban ebben az esetben annak a programnak amire rátelepedik. - vagy is inkább csak a screenét lopja el - már futnia kell. amikor a mi programunk elindul! Különben nem lesz mit ellopnia. Most már nyissunk ablakot, de milyet is ? Az ablakoknak több típusuk van, amik az alábbiak:

A **BACKDROP** ablakok olyan ablakok, amelyek nyitásukkor mindig a többi ablak mögé kerülnek, és ott is maradnak. Még akkor is. ha az user nyomkodja azt a gadget-et. amivel előtérbe lehetne hívni. Ezért aztán nem is reagál más gadget nyomkodásra, mint a "Close" gadget-ére. amely esetleges becsuka&áról van hivatva tájékoztatni (Az user persze rögtön ezt fogja nyomni). Mindensetre ezt az ablakot jól használhatjuk olyan esetben, mikor a háttérben szeretlünk volna rajzolni. Azért előnyösebb, mintha közvetlenül a screenre rajzolnánk, mert a rendszer úgy ke/.eli mint az ablakot, és így élvez-

ll*-t[LAl-v itltli^Lí ^l-^i.y^-t -* ^^.^c**-, ƒ;^ *:i>omKf»n

A BORDRLESS ablakok olyan normál ablakok, amelyeknek nincs keretjük, azaz bordér nélküliek. Borderek azok a keretnek használt vonalak, amik

2.0-án felül már igen tetszetős. 3D-szerű kinézetet kölcsönöznek az ablakoknak. Ide tartoznak még az ablak becsukását. háttérbe rakását, ill.. nagyságát állító gadget-eket. ill. a fejlécet tartalmazó keretek. Ha tehát ilyen ablakot használunk, ablakunk ezektől mentesek lesznek, bár a gadget-ek működnek, csak éppen nem láthatóak. Fejlécet is megadhatunk, de látni azt sem fogjuk. Használatára jellemző, hogy akkorára szokták nyitni, mint maga a screen. így annak keretei sem látszanak. Ilyen ablakot használ a ProTracker (ki is lehet lépni belőle, ha a close gadget helyén megnyomjuk). Valamint a DirOpus is. így lehet annak méretét állítani. - Wb-re kell nyitni.

SUPERBITMAP olyan típusú ablak, amelynek a látóterét hogy használja, mi adjuk meg, nem a rendszer. Emiatt nekünk is kell azt foglalnunk. De azzal az előnnyel kecsegtet, hogy virtuállis képmezőt alakíthatunk ki, amelyet az ablakunkba scrollozhatunk, ha az user úgy kívánja. Ezzel elérhető, hogy nagyobb felbontású képünk legyen, mint amit a rendszer maga adhat. A tulajdonképpeni méretnek csak a meóriánk szab határt. Esetleg a fantáziánk.

Hamar ilyen sokat beszéltünk az ablakban található gadget-ekről, talán nézzük meg őket jobban. Az ablakunkon álltalában 5 gadget hejezkedhet el. amit a rendszer hejezhet el. Természetesen több is lehet, de arról nekünk kell gondoskodni, hogy ott legyenek. Szóval a rendszer gadget-ek a következők. Az ablak bal felső szélén helyezkedhet el az ún. **„Close"** gadget. amely arról adhat tájékoztatást nekünk, hogy az user be akarja csukni az ablakunkat. Emellett található egy olyan gadget. amelynek a mérete változhat az ablak méretével együtt. Ebben található az ablak fejléce is.

Ez a gadget az ún. **„Drag"** gadget, amellyel az abalakunkat mozgathatjuk ide-oda a képernyőn. Mellette található a háttérbe tevő gadget, ún. **„Depth Arragement Gadget-Back",** vagy 2.0-án az ablakot minimális méretre húzó gadget. Mellette természetesen az előtérbe helyező gadget **"Depth Arragement Gadget-Up".** Végül a **"Sizing"** gadget, amellyel az ablak méretét állíthatjuk, (lásd 6. kép)

Ennyi ismeret után nézzük meg magát az ablakot inicializáló struktúrát, hogy miként is működik. Ahhoz, hogy az ablakot megtudjuk nyitni, az **OpenWindowfJ** függvénynek szüksége van egy **NewWindow** struktúrát megcímző mutatóra, amely megmodja neki, hogy hogyan is néz ki az az ablak. A struktúra a következő:

```
struct NeujfUindoui
{
    SHORT LeftEdgeJopEdge;
    SHORT lilidth, Height;
    UBVTE DetailPen, BlockPen;
    ULONG IDCMPFlags;
    ULONG Flags;
    struct Gadget *FirstGadget;
    struct Image •CheckMark;
    UBVTE *Title;
    struct Screen *Screen;
    struct BitMap *BitMap;
```

```
    SHORT MinIüidth, MinHeígth;
    SHORT MaKIDidth, MaxHeigth;
    USHORTType;
};
```

Ahol a struktúra tagok jelentése a következő:

LeftEdge: Az ablak megjelenésének x koordinátája.

TopEdge: Az ablak megjelenésének y koordinátája.

Width: Az ablak szélesége. pixelben. Workbench ablak esetén általában 1 -től 640ig.

Height: Az ablak magassága. Az értéke 1-től a Screen Height értékéig lehet megadni.

DetailPen: Annak a színregiszternek a megadása, amely színnel a fejlécszöveget szeretnénk kiíratni. OS 2.0-ától ez a megadás nem hatásos, mert ott mindig az 1 -es színnel lesz írva. Amigán általában a színek számán nem azt a számot értjük, mint mondjuk a PC esetében, hanem egy szín regiszterszámát. Ez a színregiszter tartalmazza a tulajdonképpeni színt. PC-n ha azt írjuk egy szín megadásánál, hogy például O-ás, az feketét fog jelenteni (az 1-es kéket stb.). Amigán ez csak egy színregisztert jelent. Azonban ha az ablakot Workbench screenhez csatoljuk, akkor az ablakunk színei meg fognak egyezni a Wb színekkel. Ezek általában, az alábbiak: O-ás szín a háttér színe, és 1.2. 1.3-as gépeknél ez a szín kék. A 2.O-ás gépeken ez többnyire szürke (lásd. MagicWb). Az 1-es szín többnyire fekete. A 2-es szín a fehér. A 3-as szín 1.2, 1.3-as gépeken többnyire narancs, míg 2.0-ástól ez a szín kék. Az 1.2, 1.3-as gépeken a Wb-et nem lehet átalakítani többszínűre. 2.0-ás gépeken általánosabb a 8 színű Wb. Ennek színeit megnézhetjük a sys:prefs/palette program segítségével.

BlockPen: Annak a s/ínregiszternek a megadása, amely színt a háttérszínnek választjuk a fejléc alá. OS 2.0-ától a rendszer maga állítja a színeket attól függően, hogy az ablak aktív-e, avagy sem. Ezt megelőzően ezt nem színnel jelölte, hanem az ablak fejlécét alkotó részt úgy rajzolta, hogy csak minden második pixelt tett ki. ha az ablak nem volt aktív. Egyébként rendesen kirajzolta.

IDCMFlags: Ezt a következő fejezetben fogjuk részletesen megnézni!

Flags: Ha az ablakunkra akarunk rendszergadget-eket, ill.. speciális ablakot akarunk használni. (BORDERLESS. BACKDROP stb.) azt itt kell beállítani. A flag-eknek megfelelő értékek, definiálva vannak az **íntuition/intuition.**h-ban. A flag-ek az alábbiak:

Rendszer gadget-ek:

WFLG_CLOSEGADGET
Ez a flag fogja kitenni, az ablak bal felső sarkába, a Close gadgetet. Figyelem, a close gadget-et. nem a rendszer fogja, lekezel-ni. hanem, csak egy üzenetet küld a programunknak, hogy <A^ UO*ÍI la^híWmjs^ t^ ^í^i-au^t í-c^iicni A? riHlíik foec«iukását nekünk kell elvégezni, a CloseWindowfJ függvény meghívásával.

WFLGJJRAGBAR
Ez a flag csinál egy drag gadget-et az ablakra, aminél fogva az ablakot el lehet majd mozdítani.

WFLG_DEPTHGADGET
Ez a flag kirakja az ablakunkra a Depth gadget-et a jobb felső sarokba.

WFLG_SIZEGADGET
Ez a flag az ablak jobb alsó sarkába teszi ki az ablak méretét állító, ún. SlZE gadget-et. Korlátot tudunk szabni, hogy az user metó'l meddig nyissa ki az ablakot. Ebben segítségünkre vannak a **MaKLUidth/MaKHeight/ MinUJidth/ MinHeígth** váltózók. Ha ezt a flag-et megadjuk, további két flag-et használhatunk. **A WFLG_SIZEBBOTTOM** ami az ablak aljára is rajzoltat az intuitionnnal egy bordert, valamint a **WFLG_SEZEBRIGHT,** ami ugyanezt teszi, csak az ablak jobb szélére rajzolja. Természetesen használhatjuk mind a kettőt is.

speciállis ablakok:

WFLG_BACKDROP
Használatával ablakunk backdrop ablak lesz.

WFLG_BORDERLESS
Használatával ablakunk bordér nélkülivé válik.

WFLG.GIMMEZEROZERO
Használatával GimmeZeroZero. ablakhoz jutunk.

WFIXLSUPERBITMAP
Ez a flag teszi ablakunkat SuperBitMap ablakká. Ne feledkezzünk meg azonban arról, hogy ebben az esetben nekünk kell gondoskodnunk a BitMap-ról. (példát a későbbiekben mutatunk).

az ablak frissítését szolgáló flag-ek:

WFLG_SIMPLE_REFRESH
A saját programunknak kell. az ablak frissítését elvégezni.

WFLGJ3MART_REFRESH
Az intuition végzi az ablak frissítését. Ha az ablakunk elég nagy, vagy grafikákat, ill. egyebeket rajzoltunk bele. célszerű saját magunknak frissíteni az ablakot, mert az intuition. ezeket nem frissíti!

más egyéb flag-ek:

WFLG_REPORTMOUSE
Ezt a flag-et akkor állítjuk be, ha az egér pointeréről (a nyíl) mozgásáról tudni akarunk. A vissza adott érték x,y szerűen jelentkezik.

WFLG_NOCAREREFRESH
Ezt a flag-et akkor használjuk, ha nem akarunk semmilyen visszajelzést az ablak frissítéséről.
WFLG_RMBTRAP
Ezt llag-et beállítva elérjük azt, hogy az aktív ablakunkon az user hozzáférjen a menükhöz.
WFLG_ACTIVATE
Ha ezt beállítjuk, az ablakunk rögtön a nyitáskor aktív lesz.

OS 2.0-ától élő flag-ek:

WFLG_NEWLOOKMENUS
Az ablakunkon új kinézetű menük lesznek láthatók, amennyiben használunk menüket.
WFLGJWJEXTENDED
Használatával utalunk arra. hogy a bővített adatok fognak következni. Csak az **extended NeiulLlindoui** struktúránál használható.

FirstGadget: mutató az első gadget-struktúrára az ablakunkon. A gadgel-ek egymásra mutató pointerekkel vannak megadva, és ide a gyökér elem, tehát az első gadget címét várja. (A gadget-eknél részletezzük.)

CheckMark: Mutató egy Image struktúrára. Részletesen a grafikai résznél fogunk erről szólni. Ennél is, és az előbbinél is, ha nem adunk meg semmit, NULL-t kell beírni.

Title: Mutató egy Stringre. amely a fejléc szövegét tartalmazza, egyébként NULL.

Screen: Mutató egy Screen struktúrára, ha az ablakunkat a Workbench Screen-hez akarjuk csatolni, akkor NULL. Azt. hogy ide mit kell írni. befolyásolja az, hogy mit adtunk meg, hogy milyen screenhez akarjuk csatolni az ablakunkat, (lásd. type: CUSTOMSCREEN. WBSCREEN)

BitMap: Ha SuperBitmap ablakot akarunk használni, ide kell beírni az általunk allokált BitMap struktúra címét, egyébként ide is NULL-t kell írni.

Ha használtuk a WFLG_SIZING opciót, akkor megadhatjuk azt, hogy az user milyen határok között állíthatja az ablakunk méretét. Erre való a;

MinWidth: Minimum szélessége az ablakunknak, és a
MinHeight: Minimum magassága az ablakunknak.
MaxWidth: A maximális szélesség, amit az ablakunk felvehet, és a
MaxHeight: a maximális magasság, amire az ablakot ki lehet húzni.

Ha nem kívánjuk állíthatóvá tenni, adjunk meg 0.0-t. Ekkor az ablak csak a fentebb már megadott adatokkal fog megjelenni, és méretét nem lehet megváltoztatni, még ha a Sizing gadget rajta is van.

Type: Ha azt akarjuk, hogy az ablakunk csatlakozzon a Workbench screenhez, akkor írjunk WBENCHSCREEN-t, vagy ha saját magunk által létrehozott screen-hez akarjuk, akkor COSTUMSCREEN-t. Ha saját screent

llclõAlléÜLinK, i* Be reen nel meţ* kf»ll ítrlni ^ ,á mutáló ooinLert.

Ezek után már képesek vagyunk megnyitni egy ablakot, csak használnunk kell az intuition egyik függvényét. Ez a függvény az OpenWindowfj függvény.

Argumentumának meg kell adni egy pointert, az előbb tárgyalt **NewWindow** struktúránkra. Ha sikerült neki megnyitni az ablakot, visszatér egy Window struktúrát megcímző mutatóval. Ha tehát a visszatérési érték NULL-lenne. akkor valami baj van. Ez a baj általában abból szokot eredni, hogy nincs elég memória az ablak megnyitásához. De nézzük meg azt az esetet amikor sikerült neki megnyitni, hogy mit is ad vissza.

```
struct UJindow
{
        struct IJJindoLu *NextUUindow;     /'Mutató a kóuetkező ablak
                                                    struktúrára*/
        SHORT LeftEdge, TopEdge;                  /*flz ablak pozíciója*/
        SHORT Illidth, Heigth;                    /*flz ablak mérete*/
        SHORT MouseV, MouseH;          /*Rz egér pozíciója az ablak bal
                                                felsó sarkához reletíuan.*/
        SHORT MinWídth, MinHeight;     /*Mínimum/MaHímum mérete
                                                az ablaknak.*/
        USHORT MaxUJidth, MaxHeight;
        ULONG Flags;                              /*Rz ablak flagjeí*/
        struct Menü *MenuStrip;  /*Pointer az első Menü struktúrára*/
        UBVTE *Title;                             /*flz ablak fejléce*/
        struct Hequester *FirstRequester;
        struct Requester *DMRequester;
        SHORT ReqCount;
        struct Screen *UIScreen;       /*Mutató a screen struktúrára,
                                        amelyhez az ablak csatlakozik*/
        struct RastPort *RPort;                   /*Rz ablak RastPortja*/

        BVTE BorderLeft, BorderTop, BorderRight, BorderBottom;

        struct RastPort *BorderRPort;     /*Ha az ablak GimeZeroZero
                                        ablak, a külsórész RastPortja.*/
        struct Gadget *FírstGadget;          /*Mutató az első Gadget
                                                    struktúrára*/
        struct lilindoLu *Parent, *Descendant;

        USHORT *Pointer;             /*Hz egér pointer adatára mutató*/
        BVTE PtrHeight;                /*flz egér pointer szélessége*/
        BVTE PtrUJidth;                /*Rz egér pointer magassága*/
        BVTE HOffset, VOffset;           /*Rz egér pointer "Hot Spot"
                                                    pontjának helye*/
        ULONG IDCMPFlags;                       /*flz IDCMP flag-ek*/

        struct MsgPort *UserPort, *UJindoujPort;
        struct IntuiMessage *MessageKey;

        UBVTE DetailPen, BlockPen;
        struct Image *CheckMark;
```

```
    UBVTE *ScreenTitle;          /*Rz ablak milyen fajta Screenhez
                                   csatlakozzon*/

/*Rz alábbiakat csak akkor használjuk, ha az ablak GimmeZeroZe-
ro típusú*/

    SHORT GZZMouseK;             /*Rz egér koordinátája, relatíuan a
    SHORTGZZMouseV;                          belső ablakhoz*/
    SHORT GZZUJidth;          /*flz ablak belső részének nagysága*/
    SHORT GZZHeigth;

    OBVTE *EntData;
    BVTE *UserData;
    struct Layer *IDLayer;
    struct TentFont *IFont; };
```

Tehát megnyitottuk az ablakunkat (lásd. a lemezmelléklet ide vonatkozó első programja "Elsö_ablakom.c"). A programhoz annyit még hozzáfűznénk, hogy a használt DelayQ függvényt azért érdemes várakozásra használni, mert úgy képes várakozni a megadott ideig, hogy közben a gépen futó többi task nem lassul. Ellenben, ha mondjuk for ciklust használnánk, a programunk foglalná feleslegesen a többi task-tól a task időt. Ezért sokkal előnyösebb ezt használni, mint a for ciklust. Azokívül ez a megadott ideig vár, függetlenül a gép processzorétól (a for ciklus egy gyorsabb gépen kevessebb ideig tart. mint egy lassabb gépen).

Ha már idáig eljutottunk, nézzünk egy példát a SuperBitmap típusú ablak használatára. A különbség a normál ablakoktól csupán abban van, hogy a BitMapnak nekünk magunknak kell lefoglalni a helyet, és nem az intuitionnak. Ahhoz, hogy ezt megtehessük a következő lépésekel kell megtennünk:

1. Deklaráljuk és inicializáljuk a NewWinow struktúránkat. Állítsuk be a flagokat, WFLG_SUPERBITMAP, és a BitMap=NULL-al.

2. Deklaráljuk a BitMap struktúrát.

   struct BitMap my_bitmap;

3. Inicializáljuk a BitMap struktúránkat.

   lnitBitMap(&my_bitmap,Depth,UJidth,Heigth);

   &my_bitmap: Pointer a bitmap struktúránkra.
   Depth: A használni kívánt bitplane-ok száma.
   UJidth: A BitMap szélessége / pixelben /
   Heigth: A BitMap magassága

4. Allokáljuk a kép memóriát a BitMapnak.

```
for(loop=0; loop<Depth; loop++)
if (my_bitmap.Planes[loop] = RllocRastedOJidth,
    Height)) == NOLL)
{
    Ha nem sikerült, akkor ide jut, tehát itt kezeljük le a hibát.
    Nincs elég memória!
}
```

5. Miután allokáltuk a helyet, nem árt, ha kitörlünk onnan mindent.

```
for(loop=B; loop<Depth; loop++)
    BltClear(my_bitmap.Planes[loop], RHSSI ZEfLUídth, Height), Q);
```

6. Befűzzük a BitMap struktúránk mutatóját a NewWinodow struktúránkba.
```
my_neiu_windoiu.BitMap = &my_bitmap;
```

7. Végül megnyitjuk az ablakot.
```
OpenWindow(G'my_new_ujindow);
```

8. Majd felszbadítjuk a memóriát, bezárjuk az ablakunkat és lezárunk mindent.
```
CloseUJindou4G'my_neiij_window);

for(loop=0; loop<Depth; loop++j
    íf(my_bitmap.Planes[loop])
      FreeRaster(my_bitmap.Planes[loop], LUidth, Heiyht);
```

Mint látjuk nem olyan bonyolult ez. Nézzük meg a futó programot is a lemezmellékletről, (lásd. SuperBitMap.c) Részletesen fogjuk tárgyalni a felhasznált függvényeket a könyv vége felé elhelyezkedő függelékben. Itt minden könyvtári függvényt megnézünk.

1.5.2. A screen-ek, avagy a képernyőkről

Az ablakoknál beszéltünk arról, hogy ahhoz hogy ablakot nyithassunk, léteznie kell mar egy megnyitott képernyőnek is. Ezt a screent a Workbench srreennek nevezzük. Bár maga a Workbench nem biztos, hogy már elindult, de ahhoz, hogy a rendszer kirakjon egy AmigaDos ablakot, annak is kell egy nyitott képernyő.

Vagy ha sikerül kilépni a Workbenchbó'l (Workbench menü Quit pontja). akkor is egy üres workbench screenhez jutunk. Tehát mint látjuk, egy screen mindenféleképpen nyitva, és rendelkezésünkre áll. így a screen-eket mindjárt két külön csoportra is oszthatjuk. Az egyik a **Workbench Screen.** Erről a screenről tudnunk kell. hogy bár a színeit megváltoztathatjuk, de színeinek számát már nem. és a felbontását sem. Azoknál az Amigáknál amellyek 2.0-ás előtti rendszert használnak, sajnos nem sok lehetőségünk van ennek a screen-nek a változtatására. A 3.0-ás vagy feletti rendszert használóknak azonban lehetőségük van akár HAM-8-as (tehát 262144 színű) Workbench figyelésére. De nyugodtan vehetjük azt alapul, hogy egy 2.0-nál régebbit használó gépen a Workbench 4. azaz négy színű. A 2.0-felett 8 szín mondható álltalánosnak. A felbontás úgy alakul, hogy általános a 640x256 (PAL) felbontás (én még nem láttam olyat, aki az A500-asát 512-es Interl^ace módban használta volna. Hacsak nincs egy FixedFlicker nevű ketyeréje, ami a villódzást kiküszöböli.) A 2.0 felettit használók azonban olyat használnak, amilyet csak lehet. Azért ha általánosnak vesszük a régit használók felbontását, nagy hibát nem követünk el. De előfordulhat, hogy a program amit írni szeretnénk, más színeket, és más felbontást igényel mint amit a workbench nyújt Akkor

használhatunk olyaL, és annyi színt amilyet csak akarunk, olyan felbontásban amilyet a gép megenged, (lásd. Amiga felhasználói kézikönyv; Devs/Monitors) Nos. ebben az esetben **CUSTOM SCREEN-t** kell használnunk. A 2.0-ás rendszer feleltit használóknak, azonban rendelkezésükre áll egy ün **PubScreen** is. Ez nem kocsma screen-t jelent, hanem ún. Public, azaz nyilvános screen-t. Ez ánnyit tesz, hogy van egy-két program amit az userek nagy százaléka használ, és programunk erre a screen-re mintegy rá lud telepedni. Ennek a screen-nek meg tudjuk kerestetni a screen struktúrára mutató pointerét, és ebből minden1 megtudhatunk róla. valamint akár ablakot is nyithatunk rá, vagy rajzolhatunk rá. Az operációs rendszernek ez a fajta lehetősége azonban csak 2.0-tól létezik. Ilyen lehet például az Adpro. DirOpus, Fmaster stb. Azonban jegyezzük meg azt, hogy lehet hogy nem olyan jól néz ki a Workbench screenre megnyitott ablak, mint egy sajátra, de megvan az a nagy előnye, hogy nem foglalja a memoóriát feleslegesen egy screen. Ha tehát nem használjuk ki a saját screen lehetőségeit, inkább nyissuk az ablakunkat a Workbenchre.

Ahhoz, hogy Custom Screent használjunk, inicializálnunk kell egy **NewScreen** struktúrát, a megfelelő adatokkal. Majd meg kell hívnunk az **OpenScreenO** függvényt, amelynek meg kell adni argumentumnak az imént inicializált struktúra mutatóját. Mielőtt tovább mennénk tisztáznunk kell. hogy mit is jelentenek azok a definíciók, amit a screen megadásnál használnunk kell.

A felbontásnál van olyan hogy **High-resolution,** amely a nagy felbontásra utal. Ez a felbontás x irányában azaz szélességében 640 pixel, míg a **Low-resolution** screen csak 320 pixel széles. Ezek az adatok természetesen **PAL-**vagy-NTSC monitor esetén érvényesek. Minden monitor módnak van High. ill., Low resolution módja, csak ez más és más lehet. (Ezt megnéztük, és részleteztük az „Amigáról" részben)

A screennél meg kell adnunk, hogy a screen hány színt használjon, vagyis használhasson. Ezt úgy adjuk meg. hogy a megjelenítéshez használt BitPlane-ok számát adjuk meg. A neve is erre utal. ami a **DEPTH.** Minél több BitPlane-t használunk annál több színt is használhatunk. Úgy mint:

Depth	Színek száma	Színregiszter szám
1	2	0 - 1
2	4	0 - 3
3	8	0 - 7
4	16	0 - 15
5	32	0 - 31 (csak low - res ben és 2.0 alatt)
6	64	0 - 63 (csak 2.0-ától)
7	128	0 - 127
8	256	0 - 255

A 2.0 alatti gépekből csak különböző trükkökkel lehet kicsalni 64 ill. 4096 színt egyszerre a képernyőre, a 2.0 feletti gépeknél, amelyek ECS vagy AGA chip készlettel készültek, ennél jóval több szín is kicsalható.

Eypket a. trükköket HAM nak vagy Extra HalfBrighte-nak nevezik. Ahhoz, hogy a színeket beallilsuK, meg Ken mvnuim iy,y iuggwnj-i, >^ij..vh *„-.. - **SetRGB4fJ.** amellyel 4096 színű palettáról állíthatunk-be a színeket. Ha azonban olyan gépünk van amely ECS-t. vagy AGA-t tartalmaz, erre a célra

használhatjuk a SetRGB32() függvényt is. Ezek a függvények a graphics könyvtárban találhatóak. Ha a Screenünknek nem definiálunk külön színeket, akkor az a Workbench színeit fogja átvenni. Még akkor is. ha a screenünk 256 színű. Erre az esetre is van a rendszernek defaull beállítása, amivel a színregisztereket feltölti. Jobban mondva nem feltölti, hanem csak nem törli.

A függőleges felbontás is lehet különböző. Használhatunk ún. váltottsoros üzemmódot is. Ez abból következik, hogy ha nagyobb felbontást kívánunk használni,- és mivel a videó sávszélesség sajnos véges.- és át is akarjuk ezt vinni a monitor felé. akkor egy kis furfangra szorulunk. Nevezetesen ezt úgy oldják meg. hogy a képernyőn először csak minden második sort jelenítenek meg, tehát az 1. sor után nem a második jön. hanem a 3. Miután így elértek a képernyő aljára, kezdik kirakni a második félképet is, ami természetesen a 2. sorral kezdődik, és a 4.-kel folytatódik. Ezt ugyan mi villódzni látjuk, mivel a szem tehetetlenségéből adódóan csak az 50Hz-nél gyorsabb frekvenciájú képeket látjunk állni.

Ez a technika azonban igen jól használható abban az esetben, ha a kép tartalma állandóan változik (így működnek a TV-k is, valamint a mozifilm kockaváltása is 25Hz.). Tehát a képet villódzni látjuk, mert a korábbi frekvencia alatt (50Hz) egy képet raktak ki. míg ezzel az eljárással csak felet tesznek ki a képernyőre. Sajnos azonban a számítógépeknél nem változnak a képek, és bár Amigáról van szó. és lehetne multimédiázni, azonban ez nem általános, így a kép igen keveset változik. Ezért mi igencsak villódzni látjuk. Még akkor is. ha ECS. vagy AGA chipszettel van felszerelve a gépünk, és olyan monitorunk van amely minden felbontást elbír. Ez abból ered. hogy a szinkronjelek nem pontosan oda rakják a félképeket mint ahová kellene, és emiatt a vízszintes vonalak minden félképben (tehát csak 25Hz) mászkálnak a két érték között. A két értéket a félképek jelölik ki. Tehát hiába az AGA chipszet tudománya, hogy a frekvenciákat oda állítjuk ahová akarjuk, mert sajnos a beállított frekvenciák fele sohasem lesz SOHznél nagyobb. Ezt a trükköt nevezik INTERLACED-nek. Ennek a segítségével elérhetjük, hogy a függőleges felbontás 256-ról a duplájára nó'jjön, (Természetesen PAL monitor esetén. NTSC esetén csak 400.) Itt jegyezném meg, hogy a remegés vagy villódzás ellen lehet tenni. Méghozzá egy ún. FixerFlicker segítségével, ami egy olyan hardver, amely a képet saját memóriájában tárolja, majd olyan frekvenciával olvassa azt ki. és jeleníti meg, hogy nem látjuk villódzni. A villódzáson segíthetünk úgy is. ha az Interlaced screen-en kevésbé kontrasztos színeket használunk. így a villódzás nem lesz olyan szembeötlő. Például leketét szürkével a fehér helyett. Vagy monitorunkon az elektronsugár helyzetét megjelenítő foszforpor utánvilágítási ideje elég hosszú ahhoz, hogy a képet ne lássuk villódzni. Ez a monitortípus viszont hátrányos animáció nézéskor.

Ahhoz, hogy a képernyőnkön több színt használhassunk, két lehetőség ten. Az egyik a HAM, amely a Hold And Modify (tartás és módosítás) szavakból ered. Ez úgy működik, hogy a használható színek mellé csak olyan színeket használhatunk, amelyek nem teljesen különböznek a mellettük lévőktől. Ha belegondolunk abba. hogy a természetben kevés az éles kontúr, beláthajuk. hogy így kevesebb színnel is jó eredményt érhetünk el. Bár a képünk nem éri el az igazi 24 bites kép minőségét, de jól megközelíti azt. A 24 bites kép azt jelenti, hogy minden alapszínt 8 biten tárolunk, ami 3x8=24 elvén

24 bitet jelent. A három alapszín a következő, a piros (Red), a zöld (Green), valamint a kék (Blue). Ezen színek keveréséből kikeverhető bármely szín, amit szemünkkel érzékelni tudunk. Ezt használják aTV-technikában. sőt a fényképészetnél is. A másik az **EHB** amely az Extr Half Birighte (Extra Fél Fényerő) szavakból tevődik össze. Ez annyit jelent, hogy az alap 32 színből úgy nyernek 64 színt, hogy az alapszíneket félfényerővel megemelik. így létrejön a második 32 szín. Persze ezek az előző kissé világosabb változatai lesznek.

Ha a feladat úgy kívánja, használhatunk ún. **Dual Playfields** képernyőt is. Ez a típus abban tér el a többitől, hogy itt két képernyőt használunk egyszerre. Ebben a módban az egyik képmező közvetlenül a másik előtt jelenik meg.

Pédául egy akciójátékban az egyik képmező lehet a háttér, a másik képmezö pedig egy vezérlőpult. Mindkettőt változtathatjuk anélkül, hogy az egész képet kellene újraterveznünk. Ezenkívül mind a két képmezőt mozgathatjuk egymástól függetlenül is. Az egyes képmezőn kiválasztott átlátszó pixel az alatta lévő szín megjelenését eredményezi.

Azt, hogy a screen-ünkön milyen betűvel írhatunk, a screen **FONT-ja** állítja be. A Fontok azaz a betűtípusok a SYS:FONTS könyvtárában helyezkednek el. Van azonban két típus amit defaultnak is nevezhetnénk, ezek a TOPAZ_SIXTY: amely 9 pixel magas, valamint a TOPAZ_EIGHTY: amely rsak 8 pixel magas. Ezek a fontok be vannak építve a KickstartROMba. Ezek mindig rendelkezésre állnak, még ha nincs is FONTS' könyvtárunk. A screenünk méretét már meghatároztuk, de lehetőségünk van pozíciójának meghatározására is úgy, mint az ablak esetében. Gondoljunk arra. hogy egy kalandjátékot írunk, és a háttér elé akarunk rakni egy képernyőt, amely más színeket használ, és más felbontású is mint a hátért tartalmazó, és ebben az esetben általában kisebb felbontást alkalmazó háttérscreen. Ezt ne tévesszük össze a Dual Playfields móddal, ahol a két képernyő átlátszó lehet. így egymás színei átlátszanak a másikra. Ha a mi esetünkben az elől álló képernyőt felhúznánk a másik elé, akkor a háttérből nem látnánk semmit.

A screen fejlécének megadására is lehetőségünk van, mégpedig kétféle módon is. Az egyik amit már az ablakoknál megnéztünk, miszerint az ablak nál határozzuk meg. Valamint mi is definiálhatjuk azt a NewScreen struktúrában. Ezt lesz a default beállítás, amit aztán az ablak meg tud változtam.

Akkor nézzük meg. hogyan kell inicializálni a NewScreen struktúrát:

```
struct NeuiScreen
{
    SHORT LeftEdge, TopEdge, Width, Height, Depth;
    UBVTE DetailPen, BlockPen;
    USHORT UiewModes;
    USHORTType;
    structTextfIttr *Font;                                    *
    UBVTE *DefaultTitle;
    struct Gadget *Gadgets;
    struct BitMap *CustomBitMap;
};
```

JjeltEŰge; A &treeri kcídő ?4 koordinátájúi. Ez mindig O legyen

TopEdge: A screen kezdő y koodinátája.

Width: A screen szélessége. Ez általában low-resolution esetén 320, egyébként 640.

Height: A screen magassága. Nem interlaced üzemmódban, ez lehet 1-2OO(NTSC) vagy 1-256(PAL) módban. Interlaced esetén ez 1-400(NTSC), 1-512(PAL) módban.

Depth: A screen színeinek száma, (lásd fent) DetailPen: A screen szöveg megjelenítésének színe.

BlockPen: A screen szövegének háttérszíne (az észrevételeket lásd fent az ablaknál).

ViewModes: A megjelenítési mód flag-jei. Ha több flag-et is be kell kapcsolnunk használjuk a C-ben szokásos vagy művelet jelét " I" vagy a "+" jelet. A használható flag-ek a következők:

HÍRES: ezzel a flag-gel állíthajuk a screen-ünket High-resolution-ra. Az alap, a low-res.
SUPERHIRES: ezt a flag-et csak 2.0-ától használhatjuk, és képernyőnket Super high-resolutinra kapcsolja. Ami PAL monitor esetén 1280 pixel felbontást jelent vízszintes irányba.
- LACE: ezzel a screen-ünket váltottsoros módba kapcsolhatjuk. Az alap a Non-Interlaced.
SPRITE: ha használni akarunk Sprite-ot a screen-ünkön akkor ezt állítsuk be.
DUALPF: ha a screen-t DualPlayfields módúnak akarjuk használni, akkor használjuk ezt a flag-et.
HAM: ezzel a screenünk a HAM színek kezelésére is alkalmassá válik. (Nem AGA gépeken 4096 szín, AGA gépeken akár 262 144 szín egyszerre.)
EXTRAJKALFBRITE: a screen-ünk így extra halfbirte módban nyílik meg. (64 szín)
GENLOCKJVIDEO: ez egy nagyon éredekes lehetőség. Az Amigához könnyen lehet kapcsolni ún. genlock egységet, amelynek segítségével az Amiga képe keverhető lesz más videó forrásokkal. Ezt felhasználva könnyedén használhatjuk gépünket feliratozásra, és egyéb videós munkákra. Ha ezt beállítjuk, és kihasználjuk azt. hogy minden pixelről megmondhatjuk, hogy az átlátszó-e a videó számára avagy sem. akkor fog működni a dolog.

Type: Ezzel a flag-gel megadható a screen típusa az alábbiak szerint:
WBENCHSCREEN: ha a screenünket workbench screen-nek akarjuk, általában mi nem használjuk. A rendszer viszont annál inkább.
CUSTOMSCREEN: általában ilyen screen-t használjunk. Jelentése felhasználói screen.

PUBSCREEN: ez a fajta screen csak 2.0-ás rendszertől van. Ezzel olyan screent tudunk nyitni amely nyilvános lesz. Egyéb tekintetben, megegyezik a CUSTOMSCREEN-nel.
AUTOSCROLL: akkor érdemes használni, ha a felbontást nagyobbra állítottuk mint amit a monitortípus megadott. Ekkor a rendszer a monitortípus szerinti felbontásban nyitja meg a screenünket. és a nemláthato részt a képernyőre scrollo/za, ha az egérrel nógatjuk erre. Természetesen mi ebből nem látunk semmit, a rendszer elfedi előlünk. Programozásilag egy. a mi általunk megadott screen jón létre.
CUSTOMBITMAP: ez hasonló az ablaknál megbeszéltekkel, itt is nekünk kell foglalni a BitMap-ot.
SCREENBEHIND: ha a screenünket a többi screen mögé akarjuk tenni, használjuk ezt. Akkor előnyös a használata, ha két vagy több képernyőt használunk, és ameddig erre rajzolunk a másikat nézi az user.
SCREENQUIET: ha nem akarjuk, hogy az intuition számolja a gadget-eket és titléket a screenünkhöz.
SHOWTITLE: ezt beállítva a screen automatukusan meghívja a ShoWTitle() függvényt.
SCREENHIRES: A gadget-ek figyelembe veszik, hogy a screen Hires. (privát)

Font: megadhatjuk, a screen default fontjára mulató. (Ht TextAttr struktúra mutatót vár. A grafikus résznél tárgyaljuk részletesen.) Ha nem akarunk mást használni mint a default, akkor NULL.

DefaultTitle: a screen alapbeállítású fejléce. Ezt lehetőségünk van pl. az ablakkal is megváltoztatni.

Gadgets: Ezt nem használjuk, így ide NULL-t kell írni.

CustomBitMap: Ha CustomBitMap-ot akarunk használni, ide kell megadnunk a rámutató pointert. A type-nél azonban be kell állítanunk a CUSTOMBITMAP ílag-et. Ha nem használjuk. írjunk ide is NULL-t.

Mivel most már mindent tudunk. így nyithatunk egy screen t is. Ahhoz, hogy ezt megtehessük. nem kell mást csinálnunk csak inicializálnunk egy NewWindow struktúrát. Ha ezt megtettük, meghívjuk az **OpenScreenO** függvényt, amelynek argumentumként megadjuk a NewScreen struktúránk mutatóját. Visszatérési értékként a függvény tájékoztat bennünket arról, hogy sikerült-e neki a screen-t megnyitnia. Ha igen, egy pointert ad vissza a létrejött screen-ünkre. ha nem, akkor pedig NULL pointerrel tér vissza. De nézzük meg hogyan is hívjuk meg a NewScreenO függvényt.

```
my_screen = OpenScreen(&my_new_screen);
```

A my_screen deklarálva volt mint;

```
struct Screen *my_screen;
```

A my_new_screen deklarálva volt mint;

```
struct NewScreen my_neuj_screen;
```

Ezek után persze inicializáljuk a saját adatainkkal. Ha a screen-ünket be akarjuk csukni, azt a **CloseScreenO** függvénnyel tehetjük meg. Argumentumnak egy mutatót vár a nyitott screen-ünkre. A lemezmellékleten találhatunk ebben a témában két példaprogramot is. Az első esetben csak nyitunk egy alap screen-t. A másodikban egy kicsit megkomplikáljuk azzal, hogy a screen-ünknek definiálunk default fontot is. A függvényeket itt sem tárgyaljuk meg. hiszen ezt részletesen megtesszük a könyv 2. fejezetében (az Intuition része). Azért nézzük még meg pontosan, hogyan néz ki a screen struktúra, amit az OpenScreenf) visszaad.

```
struct Screen
{
    struct Screen *NentScreen; /*Mutató a köuetkező screenre.*/
    struct UJindow *FírstUJindou»;      /*Mutató az első ablakra a
                                                    screenen*/
  SHORT LeftEdge, TopEdge;  '              /*fl screen pozíciója*/
    SHORT lilídth, Heíght;   /*fl screen szélessége, és magassága*/
  SHORT MouseV, MouseH;       /*Rz egér pointerének koordinátái,
                          a screen bal felső sarkához ralatíuan*/
    USHORT Flags;                        /*R kiválasztott flag-ek*/
    UBVTE *Title;          /*Mutató a jelenlegi, fejléc szöuegre*/
    UBVTE *DefaultTitle;    /*Mutató az alap beállítású fejlécre*/

    BVTE BarHeight, BarUBorder,
        BarHBorder, MenuUBorder,
        MenuHBorder;
    BVTE UJBorTop, UJBorLeft,
        UJBorRíght, UJBorBottom;

    struct TeKtRttr *Font;             /*R screen default fontja*/
    struct UíeujPort UieujPort;          /*R screen uiewportja
                                          [lásd grafikai résznél]*/
    struct RastPort RastPort;
    struct BitMap BitMap;
    struct Layer_Info LayerInfo;
    struct Gadget *FirstGadget;
    UBVTE DetaílPen, BlockPen;
    USHORT SaueColorO;
    struct Layer *BarLayer;
    UBVTE *EKtData;
    UBVTE *UserData;
};
```

Hogy ebben a struktúrában részletesen melyik micsoda, egy későbbi fejezetben megnézzük, sőt használjuk is. Ez a fejezet az alacsonyszintű grafikával foglakozik majd.

1.5.3. *Csináljunk saját „CUSTOM" egérpointert az ablakunknak*

Mielőtt megnéznénk egy sokkal egyszerűbb módját annak, hogyan nyithatunk ablakot és képernyőt (a szépséghibája a fent említettnek, hogy csak 2.0-ás rendszertől része az Inuitionnnak a fenti eljárás. De nézzünk meg előtte egy érdekes dolgot.) Ha nyitottunk ablakot, lehetőségünk van az egérpointer megváltoztatására is. Természetesen addig él a mi pointerünk, ameddig az ablak aktív! Ezt a lehetőséget felhasználhatjuk például arra, hogy ha a programunk lefoglaja működésével a gépet, (számol, lemezen kavartatjuk, stb.) az usert tájékoztathatjuk úgy is, hogy a pointert átváltoztatjuk, mondjuk egy Zzz szimbólummá. Vagy ha rajzolóprogramot írunk, ecsetté, vagy szövegszerkesztésnél használhatjuk a toll szimbólumot is. Ahhoz, hogy ezt megtehessük, két dologra van seükségünk.

Nevezetesen, hogy inicializáljunk egy Sprite struktúrát, ahol megmondjuk a gépnek, hogy hogyan is nézzen ki a mi pointerünk. Valamint használjuk a **SetPointerO** függvényt. Mielőtt elkezdenénk programozni, el kell képzelnünk azt, hogy miként is fog kinézni a pointerünk. Miután ez megvan, tervezzük meg papíron, vagy hívjunk segítségül olyan rajzolóprogramot, amely ki tud menteni forrást is a képernyőről (ilyen pl. a PPaint is). Mivel pointert tervezünk, ne használjunk 16xl6-osnál nagyobb rajzot, és ne használjunk 3 színnél többet (4 szín az, csak a O-ás színt nem használjuk mert eiz a háttér). Tehát tervezzük meg. Nézzünk rá most egy példát. A vágyunk az, hogy egy nyilat rakjunk ki pointerként:

```
0000000200000000
0000002200000000
0000023200000000
0000231200000000
0002311200000000
0023111222222200
0231111133333320
2311111111111132
0211111111111112
0021111222221112
0002111200023112
0000211200023112
0000021200023112
0000002200023112
OOOOOOO^OOOZd112
0000000000022222
```

0: Átlátszó
1: Piros
2: Fekete
3: Fehér

Ahhoz, hogy ezt használni tudjuk, le kell fordítanunk a gép nyelvére. Az intuition itt egy Image-t vár. Mi ebből úgy kapunk image-t, hogy a képünket felbontjuk bitplane-ekre. Tehát ami átlátszó, az mind a két bitplane-en 0 lesz. Amely egyes színt használ, ott a O-ás bitplaneen egyes lesz az a bit. Ha 2-es színű az adott pixel, akkor csak az egyes bitplanen lessz az adott bit egyes. Ha a 3-as színű kell, akkor mind a két bitplanen 1-es lesz a neki megfelelő bit.

Szín	Bitplane 1-es	Bitplane O-ás	a szín regiszter száma
0	0	**0**	Bineárisan 00 = Decimálisán 0
1	0	**1**	Bineárisan 01 = Decimálisán 1
2	1	**0**	Bineárisan 10 = Decimálisán 2
3	1	**1**	Bineárisan 11= Decimálisán 3

A nyilunk tehát így néz, két bitplane-re lebontva.

```
        Bitplane O.ás                    Bitplane 1.-es

    0000 0000 0000 0000              0000 0001 0000 0000
    0000 0000 0000 0000              0000 0011 0000 0000
    0000 0010 0000 0000              0000 0111 0000 0000
    0000 0110 0000 0000              0000 1101 0000 0000
    0000 1110 0000 0000              0001 1001 0000 0000
    0001 1110 0000 0000              0011 0001 1111 1100
    0011 1111 1111 1100              0111 0000 1111 1110
    0111 1111 1111 1110              1100 0000 0000 0011
    0011 1111 1111 1110              0100 0000 0000 0001
    0001 1110 0000 1110              0010 0001 1111 0001
    0000 1110 0000 1110              0001 0001 0001 1001
    0000 0110 0000 1110              0000 1001 0001 1001
    0000 0010 0000 1110              0000 0101 0001 1001
    0000 0000 0000 1110              0000 0011 0001 1001
    0000 0000 0000 1110              0000 0001 0001 1001
    0000 0000 0000 0000              0000 0000 0001 1111
```

A fordítás második részében ezeket a bineáris adatokat célszerűen átalakítjuk Hexadecimálisra. Erre azért van szükség, mert C-ben ez sokkal kevesebb helyet foglal, és áttekinthető marad. A konverziónál használhatjuk segítségül az alábbi táblázatot:

```
0000 = O
0001 = 1
0010 = 2
0011 = 3
0100 = 4
0101 = 5
0110= 6
0111  = 7
```

1000 = 8
1001 = 9
1010 = **A**
1011 = **B**
1100 = **C**
1101 = **D**
1110 = **E**
1111 = **F**

Az eredmény **így néz ki:**

```
0.-ás:  1.-es:
0000    0100
0000    0300
0200    0700
0600    0d00
0E00    1900
1E00    31FC
3FFC    60FE
7FFE    c003
3FFE    4001
1E0E    21F1
0E0E    1119
060E    0919
020E    0519
000E    0319
000E    0119
0000    001F
```

Ahhoz, hogy ezt az Amiga is megértse, image formátumúra kell még igazítanunk. Mindezektől a problémáktól egyszerűen megkímél bennünket egy rajzolóprogram, ha tud forrást is menteni.

Tehát a sprite adatunkat az alábbiakkal kell még ellátnunk a szabályos inícializáláshoz (ne felejtsük, hogy a hexadecimális szám megadása C-ben a szám elé írt Ox-el kezdődik-.):

```
UWORD chip my_sprite_data[36]=
{
    0xBBBB, 0xBBBB,    (Csak az intuition használja)

    0xBBBB, 0xBBBB,
    0xBBBB, 0x8380,
    0xB2BB, 0x8788,
    0x86B8, 0x88B8,
    0xBEBB, 0x8819,
    0xE00F, 0x13F8,
    0x3FFC, 0xFE68,
    0xFFE7, 0x3CB8,
```

```
    8x3FFE, 0x4001,
    OHIEOE, OK21F1,
    OKOEOE, OKI 119,
    BK060E, QK0919,
    0K020E, 0KB519,
    OKOOOE, 0X0319,
    OKOOOE, BKO119,
    8x0000, 8x001F,

    0x0000, 8x0000 (Ez is csak az intuition miatt kell)
};
```

És végül hívjuk meg a SetPointer függvényt, hogy lássuk a pointerünket. Ezt az alábbi módon tehetjük meg.

SetPointer(az_ablakom, my_sprite_data, 16, 16, 0, -7);

az_ablakom: Pointer az ablakunkra.
my_sprite_data: Pointer a sprite adatunkra.
16: annak a szélessége.
16: annak a magassága. (Nem AGA, gépeken ennek 16 nak, vagy kisebbnek kell lennie!)
8: XOffset, pozíciója a pointer érző pixeljének. (Hot Spot)
-7: YOffset, 7 sorral lejebb.

A „Hot Spot" az a pixel amit a rendszer figyelembe vesz, ha lenyomjuk az egér szemét. Ez lesz az a pont, aminek az adott dolgon kell lennie, hogy a rendszer megnyomottnak vegye az egér szemét azon a ponton, és aktivizáljon mondjuk egy gadget-et. Ez alapesetben a pointerünk bal felső sarkában van. Mivel egy nyilat terveztünk, aminek a hegye nem a bal felső sarok felé néz, így nem ott van ahol várnánk. Elvárható, hogy az érzékelő pont a nyilunk hegyénél legyen, tehát rakjuk oda. Mivel ez a pont a 0,-7-es ponton van, így ezt adjuk meg az XOffset, és az Yoffset-nél.

Miután már nincs szükségünk a pointerünkre, és vissza akarunk térni a rendszeréhez, hívjuk meg egyszer a **ClearPointerfJ** függvényt. A függvény meghívása így néz ki.

```
ClearPointer(az_ablakom);'
```

Mint emlékszünk rá, az Amiga képet és hangot csak abból a memóriából képes lejátszani, **III.** kitenni a képernyőre, amit CHIP RAM-nak nevezünk. Mivel a Sprite is egy képernyő object, így ennek is ott kell lennie. Ezt a SAS/C jóvoltából igen egyszerűen elintézhetjük, nevezetesen, a típus deklaráció után egyszerűen csak be kell írnunk, hogy chip vagy__chip. Ami a fordítóval azt közli, hogy a változónak a CHIP RAM-ban foglaljon helyet.

Ez a programban így néz ki:

UWORD chip a_uáltozóm;

Ezzel a sorral létrehoztunk egy unsigned, azaz előjel nélküli word típusú változót „a_változóm" néven, amely a CHIP RAM-ban került elhelyezésre.

Erre is található egy példaprogram a lemezen. A programban két ablakot nyitunk, és mind a két ablaknak más a pointere, ha aktivizáljuk ókét.

1.5.4. Ablak és screen megnyitás V36, vagy magasabb Intuition esetén

1.5.4.1. Az ablakokról

Az új intuition lehetőséget ad arra, hogy ne kelljen nekünk inicializálni minden egyes adatot, ha meg akarunk nyitni egy ablakot vagy screen-t. Hanem csak olyan adatokat kell megadnunk, amelyekben a nyitandó ablakunk eltér a rendszerben definiált defaulttól. Ezt igen leegyszerűsíti, hogy a listában először meg kell adnunk, hogy mit akarunk megváltoztatni, és utána azt, hogy mire. Ez két függvénnyel vált lehetségessé. Az egyik az **OpenWindowTagsO,** valamint az **OpenScreenTagsO-** Mint nevükből is kitűnik, az egyiket az ablakok megnyitására **(OpenWindowTagsO),** míg a másikat a képernyők megnyitására használhatjuk. Mivel nyithatunk olyan ablakokat, és vagy képernyőket is amit már eddig is megtehettünk, ezért ne csodálkozzunk azon, hogy lesz egy-két ismerős is.

Az OpenWindo\VTags()-nál megadható paraméterek a következők:

WA_Left: Az ablak x koordinátája. \
WA_Top: Az ablak y koordinátája.
WA_Width: Szélessége.
WAJHeight: Magassága.
WA_DetailPen: A fejléc szöveg színe.
WA_BlockPen: A fejléc szöveg háttere.
WAJDCMP: Az IDCMP flag megadása, (majd a későbbiekben lárgyaljuk)
WA_Plags: A flag-ek megadása.
WA_Gadgets: A gadget-ekre mutató pointer megadása.
WA_CheckMark: Mutató egy Image struktúrára.
WA_Title: A fejléc szövegére mutató pointef megadása.
WA_ScreenTitle: A screen fejlécére mutató pointer. Akkor lesz látható amikor az ablak aktív.
WA_CustomScreen: Mutató a CustomScreen struktúrára.
WA_SuperBitmap: Ha használunk itt kell megadnunk.
WA_MinWidth: Az ablak minimális és maximális méreteit megadó flag-ek.
WA_MinHeight
WA_MaxWidth
WA_MaxHeiglit (lasa. Slzeoaüget ieircit.ci rciatct>t>.)
WA_IíinerWidth: A belső szélesség.

WAInnerHeight: A belső magasság a borderekhez. Ha az autoadjust flag-et beállíljuk, ezt elvégzi nekünk az intuitíon automatikusan.
WA_PubScreenName: Megadhatjuk a Pubsereen nevét.
WA_PubScreen: A Pubscreenre mutató pointert adhatjuk meg vele.
WA_PubScreenFallBack: Ez egy Boolean típusú változó, amelyben arról tájékoztat a rendszer, hogy ha nincs megadott PubScreen néven Pabscreen, akkor megnyissa-e azt a Workbench-re.
WA_Colors: Egy színeket tartalmazó tömböt lehet megadni, arra vonatkozóan, hogy amikor az ablak aktívvá válik, milyen színeket használjon.
WAjZoom: Egy olyan tömböt lehet megadni, amely 4 word-ből áll. és tartalmazza a Left/Top/Width/Height adatokat arra az esetre, ha az user aktivizálja a „Zoom" gadget-et.
WA_MouseQueue: Ezzel inicializálhatjuk az egér üzenetét, amit vissza ad ha az ablakból kiért.
WA_BackFillHook: A „backfill hook" amit az ablak layer használjon. (lásd. egy későbbi kötetben a layereknél)
WA_RptQueue: Ezzel a változóval inicializálható a billentyű ismétlési ill. visszatérési érték. (Boolean értékek, tehát mögötte csak TRUE vagy FALSE-1 kell megadni.)
WA_SizeGadget: Az ablakon legyen-e „Size gadget".
WA_DragBar: Legyen-e „Dragbar" az ablakon.
WA_DepthGadget: Legyen-e „Depth gadget".
WA_CloseGadget: Legyen-e „Close gadget".
WA_Backdrop: Az ablak backdrop legyen-e.
WA_ReportMouse: Kérünk-e tájékoztatást az egérről.
WA_Borderless: Az ablak bordér-nélküli legyen-e.
WA_Activate: Az ablak aktív legyen-e, mikor megnyílik.
WA_SimpleRefresh: Csak a TRUE érték esetén kell megadni.
WA_SmartRefresh: Csak a TRUE érték esetén kell megadni.
WA_BRight: Ha akarunk bordert a jobb oldalára is az ablaknak, a size gadget szélességében. Csak a Size gadgettel együtt érvényes.
WA_BBottom: Ha az ablak ajára is akarunk bordert (mint fent).
WA_AutoAdjust: (lásd WAJnnerHeight. WAJnnerWidlh-nél.)
WA_GimmeZeroZero: jelentése megegyezik a NewWindow-nál tárgyalt **WFLG_GIMMEZEROZERO-val.**
WAJWenuHelp: ha *a* menükön Help-et nyomunk, egy IDCMP_MenuHelp üzenetet kapunk a rendszertől.
WA_NewLookMenus: hatására új kinézetű menüink lesznek az ablakon.
WAJNotifyDepth: jelentése megegyezik az **IDCMP_CHANGEWINDO)V-all.**
WA_Pointer: az általunk definiáll egérpointert itt adhatjuk meg. Az OpenWindowTagsQ automatikusan meghívja a SetPointerO függvényt, ezzel az értékkel. Egyébként NULL. ha a defaultal akarjuk.
WA^BusyPointer: Ez egy Boolean, és TRUE ha saját pointert akarunk használni, ha foglalt a gép.
WA_PointerDelay: Ez is boolean változó, ami az egérpointerunk visszaállítását késlelteti.

WA_Tabletinformation: Boolean típusú, és akkor állítsuk TRUE-ra, ha üzenetet akarunk kapni a table-ról.
WA_HelpGroup: Segítségével lehetővé válik egy másik gadgetről is messaget kapni. így lehetőség van megtudnunk, hogy melyik gadetről kéri az user a Help-et (részletesebben a **GetGroupIDfJ.** és a **HelpControlO** rüggvényeknél.)
WA_HelpWindow: segítségével a fent említettekre lehetőségünk lesz más ablakra vonatkozóan is.

Amint ezekből kiderült, segítségével is minden megadható, de abban rejlik a nagyszerűsége, hogy ha csak néhányat is adunk meg, akkor is megnyithatjuk az ablakunkat.

Használatát a következő példával világítanám meg. Nyissunk egy ablakot, amely a következőket tudja: a WBScreen-re kerül. 320x200-as nagyságban, és fejlécében tartalmazza a "MI ABLAKUNK" feliratot, és amikor megnyílik aktívvá válik. Ez tehát így néz ki:

```
my_windouj = OpenUJindoujTags(NULL,        /*Ezzel kell kezdődnie*/
                UJfl_Width, 320,
                LUR_Height, 200,
                IDR_Title, "MI HBLHKUNK",
                LUH_Flags, UIFLG_HCTIUHTE,
                TRG_DONE                   /*Ezzel kell uégződnie*/;
```

Itt a my_window egy pointer típusú változó, ami a window-struktúrát címezi meg, ha sikerült megnyitnia az ablakot. Ha nem iskerült akkor NULL.

1.5.4.2. Az ablak biztonságos becsukása

Még mielőtt megnéznénk a screen-nek hasonló adottságait, beszéljünk az ablak bezárásáról néhány mondat erejéig. Vegyük figyelembe azt. hogy az Amiga operációs rendszere igazi multitaszkos rendszer. Az ablakhoz tartozhatnak olyan dolgok, amikről majd az 1.7.1 es részben beszélünk részletesen, de szervesen kapcsolja az ablakunkat a multitaszkos környezetbe. így belátható, hogy az ablak bezárásával (rósz helyen, rossz időben) igen komoly károkat vagyunk képesek okozni. Ezért létezik egy nem különösebben egyszerű, de biztos, a rendszert nem károsító ablakbezáró eljárás.

Ezt az eljárást ajánlják maguk a rendszer megíröi is, tehát biztosak lehetünk használhatóságában. Ezer szónak is egy a vége. az eljárás igazából nem tartalmaz semmi különöset, és érthetetlent.

```
/•Megadjuk a használt függuény prototípusát mielőtt definiáljuk
és használjuk azt.*/

UOIU MripilUUIfICISayCStsll-u^I MsgPort *mp, „«.-...-• IllinHnin *win);

/•Megadjuk az includokat ehhez a részhez.*/
```

```
#include "eHec/types.h"
#include "exec/nodes.h"
#include "eHec/lists.h"
#include "eKec/ports.h"
#include  "intuition/jntuitíon.h"

/• Ez az afügguény amely az ablak biztonságos bezárását uégzi. */

uoid CloseIDindoujSafely(struct Illindow *Luin)
{
    /* Először kikapcsoljuk a multitaszkingot, így az Intuition
    csak, uelünk kell, hogy foglalkozzon.*/
    ForbidO;

    /* Uísszaküldünk minden, az ablakunkhoz érkezett
    üzenetet, hiszen már nem reagállunk rájuk. */
    StripIntuiMessages(win->UserPort, win);

    /* Töröljük az UserPortot, igy az Intuiton nem használhatja azt.*/
    wín->UserPort = NULL;*/

    /* Megmondjuk az intutionnak, hogy ne küldjön több üzenetet */
    ModifyIDCMPIwin, OL);

    /* Uisszakapcsoljuk a multitaszkingot. */
    PermitO;

    /* és uégül ualóban bezárjuk az ablakot. */
    CloseUJíndowduin);
}

/* Ez a függuény eluégzi az összes, még uárakozó üzenet átuételét, és azokra a tõle telhetõén feguduariasabban uálaszol. */

uoid StripIntuiMessagesímp, min)
struct MsgPort *mp;
struct IUindoLu *LUin;
{
    struct IntuiMessage *msg;

    struct Node *succ;

    msg = (struct IntuiMessage *) mp->mp_MsgList.lh_Head;

    (uihileísucc =  msg->EKecMessage.mn_Node.ln_Succ)
```

```
            if(msg->IDCMPU)indow ==  min)
            {
                Remoueüstruct Node *)msg);
                ReplyMsgüstruct Message *)msg);
            }
            msg = (struct IntuíMessage *) succ;
        }
}
```

A programunkban használva, természetesen érdemes kiegészíteni a programban alkalmazott változók, objektumok, gadget stb. által foglalt memória-rész felszabadításával is. Ezt körültekintően végezzük, csak lefoglalt területet szabadítsunk fel.

1.5.4.3. A *screen-ekről*

Ezek után nézzük meg az OpenScreenTagsO függvény argumentumait is. melyek a következők:

SA_Left: a screen pozíciója (LeftEdge).
SAJTop: a screen pozíciója (TopEdge).
SA_Width: a screen szélessége.
SAJHeight: a screen magassága.
SA_Depth: a használni kívánt bitplanek száma.
SA_DetailPen: a fejléc szövegének színe.
SAJBlockPen: a fejléc háttérszíne.
SA__Title: a screen default fejléce.
SA_Colors: egy mutató a színeket tartalmazó tömbre.
(V39-től használjuk az SA_Colors32-t.)
SA_ErrorCode: egy lóng típusú változó, amely a hibakódra mutat.
A hibaködok az alábiakat takarják.
OSERRJVOMONITOR: a monitor típusa nem ismert.
OSERR_NOCHIPS: a custom chlpek nem megfelelőek
OSERR_NOMEM: nem állrendelkezesre elég normál memória a gépben.
OSERRJVOCHIPMEM: nem áll rendelkezésre elég Chipmem.
OSERR_PUBNOTUNIQUE: a használt public screen neve nem elérhető (nincs a rendszerben ilyen nevű PubScreen nyitva.)
OSERR_UNKNOWNMODE: ismeretlen módban akartuk megnyitni a screenünket.
OSERRJTOODEEP: nagyobb mélységre akartuk megnyitni. mint azt a hardver megengedné.
OSERR_ATACHFAIL: hiba a screen csatolásakor.
OSEKKJVOTftVAtLABI^E: a megnyílttal mód nem áll rendelkeüéore
SA_Font: megeggyezik a NewScreen.Font-tal.

SA_Type: megegyezik a NewScreen.Type-val. pl. CUSTOMSCREEN. PUBLICSSCREE (lásd. SA_I3ehind, SA_Quiet.)

SA_BitMap: megadható egy mutató a saját DitMapunkra.

SA_PubName: itt adható meg a PubScreen neve, ha screen-ünk az. Azonban ezt előtte adjuk meg mint a címét.

SA_PubSig:

PA_PubTask: az előzővel együtt a task ID signált küld. mikor az utolsó ablak is bezárult a PubScreenen.

SA_DisplayID: ez az adat új bővítés, és a használni kívánt monitor-típusról tájékoztatjuk. Ennek segítségével nyithatunk pl. DBLPAL képernyőt. Nem elég itt megadni, hanem léteznie kell a SYS:Devs/Monitors-ban is a monitor-típusának, hogy használni tudja. Az alábbiakat használhatjuk:
A PAL monitor ID-jei:

LORES_KEY
HIRES_KEY
SUPER_KEY
HAlVLKEY
LORESLACE_KEY
HIRESLACE.KEY
SUPER[J\CE_KEY
HAMLACE_KEY
LORESDPF_KEY
HIRESDPF.KEY
SUr³ERDPF_KEY
LORESLACEDPF_KEY
HIRESLACEDPF_KEY
SUPERLACEDPF_KEY
LORESDPF2_KEY
HIRESDPF2_KEY
SUPERDPF2_KEY
LORESLACEDPF2_KEY
HIRESLACEDPF2_KEY
SUPERIJ>iCEDPF2_KEY
EXTRA HAL FBRITE_KEY
EXTRAHALFBRITELACE_KEY
Csak AGA esetén (V39)
H1RESHAM_KEY
SUPERHAM_KEY
HIRESEHB_KEY
SUPEREHB_KEY
HIRESHAMIv\CE_KEY
SUPERHAMLACE_KEY
HIRESEHBLACE_KEY
SUPEREHBLACE_KEY

Kiegészítések V40-esetén, néhány jálék, és anini miatt
LORESSDBL_KEY
LORESHAMSDBL_KEY
LORESEHBSDBL_KEY
HIRESHAMSDBL_KEY

A VGA monitor ID-jei:

VGA_MONITORJD
VGAEXTRALORES_KEY
VGALORES_KEY
VGAPRODUCT_KEY
VGAHAM„KEY
VGAEXTRALORESIJ\CE_KEY
VGALORESIJ\CE_KEY
VGAPRODUCTIACE_KEY
VGAHAMLACE„KEY
VGAEXTRALORESDPF_KEY
VGALORESDPF_KEY
VGAPRODUCrDPF_KEY
VGAEXTRALORESIJ\CEDPF_KEY
VGALORESLACEDPF_KEY
VGAPRODUCrij\CEDPF_KE/
VGAEXTRALORESDPF2_KEY
VGALORESDPF2_KEY
VGAPRODUCrDPF2_KEY
VGAEXTRALORESI¡ACEDPF2_KEY
VGALORESIJ\CEDPF2_KEY
VGAPRODUCrLACEDPF2_KEY
VGAEX[r]rRAHAI.FBRITE_KEY
VGAEXTRAHALFBRI[r]PEIACE_KEY
Csak AGA estén (V39-tó'l)
VGAPRODUCTHAM_KEY
VGALORESHAM_KEY
VGAEXTRAI.ORES HAM_KEY
VGAPRODUCrHAMIJ\CE_KEY
VGALORES HAM IJKC E_KEY
VGAEX1^RALORESHAMIV\CE_KEY
VGAEXTRALORESEHB_KEY
VGAEXTRALORES EHBIJ\C E_KEY
VGALORESEHB_KEY
VGALORESEHBI^CE_KEY
VGAEHB_KEY
VGAEHBIJ\CE_KEY
VUAt-Al KftI>OKC^5Lí[illegible]_KEY
VGALORESDBL_KEY
VGAPRODUCrDBL_KEY

VGAEXTRALORESHAMDBL_KEY
VGALORESHAMDBL_KEY
VGAPRODUCTHAMDBL.KEY
VGAEXTRALORESEHBDBL_KEY
VGALORESEHBDBL_KEY
VGAPRODUCTEHBDBL_KEY

A Commodore A2024 monitor ID-jei:

A2024_MONITOR_ID
A2O24TENHERTZ_KEY
A2024FI rfEEN HERTZ_KEY

Az Euro 72 monitor ID-jei:

EURO72_MONIT0RJD
EURO72EXrRALORES_KEY
EURO72LORES_KEY
EURO72PRODUCT_KEY
EURO72HAM.KEY
EURO72EXTRALORESLACE_KEY
EURO72LORESLACE_KEY
EURO72PRODUCTLACE_KEY
EURO72HAMLACE_KEY
EURO72EXTRALORESDPF_KEY
EURO72LORESDPF_KEY
EURO72PRODUCTDPF_KEY
EUR072EXTRALORESLACEDPF_KEY
EURO72LORESLACEDPF_KEY
EURO72PRODUCTI,ACEDPF_KEY
EURO72EXTRALORESDPF2_KEY
EURO72LORESDPF2_KEY
EURO72PRODUCTDPF2_KEY
EURO72EXrrRALORESLACEDPF2_KEY
EURO72LORESLACEDPF2_KEY
EURO72PRODUCT[J\CEDPF2_KEY
EURO72EXTRAHALFBRITE_KEY
EURO72EXTRAHALFBRITELACE_KEY
Csak AGA eseten (V39-tol)
EURO72PRODUCTHAM_KEY
EURO72PRODUCTHAMLACE_KEY
EURO72LORESHAM_KEY
EURO72LORESHAMIJVCE_KEY
EURO72EXTRALORESHAM_KEY
EURO72EXrRALORESHAMlJ\CE_KEY
EURO72EXTRALORESEHB_KEY
EURO72EXTRALORESEHBLACE_KEY
EURO72LORESEHB_KEY

EURO72LORESEHBLACE_KEY
EURO72EHB_KEY
EURO72EHBLACE_KEY
EURO72EXTRALORESDBL_KEY
EURO72LORESDBL_KEY
EURO72PRODUCTDBL_KEY
EURO72EXTRALORESHAMDBL_KEY
EURO72LORESHAMDBL_KEY
EURO72PRODUCTHAMDBL_KEY
EURO72EXTRALORESEHBDBL_KEY
EURO72LORESEHBDBL_KEY
EURO72PRODUCrEHBDBL_KEY
EURO36_MONITORJD

SUPER72_MONITOR_ID
SUPER72LORESDBL_KEY
SUPER72HIRESDBL_KEY
SUPER72SUPERDBL_KEY
SUPER72LORESHAMDBL_KEY
SUPER72HIRESHAMDBL_KEY
SUPER72SUPERHAMDBL_KEY
SUPER72LORESEHBDBL_KEY
SUPER72HIRESEHBDBL_KEY

SUPER72SUPEREHBDBL_KEY

Ezek a monilor típusok csak V39-tó'l vannak:

A DBLNTSC monitor ID-jei:

DBLNTSC_MONITORJD
DBLNTSCLORES_KEY
DBLNTSCLORESFF_KEY
DBLNTSCLORESHAM_KEY
DBLNTSCLORESHAMFF_KEY
DBLNTSCLORESEHB_KEY
DBLNTSCLORESEHBFF_KEY
DBLNTSCLORESLACE_KEY
DBLNTSCLORESHAMIJ\CE_KEY
DBLNTSCLORESEHBLACE_KEY
DBLNTSCLORESDPF_KEY
DBLNTSCLORESDPFFF_KEY
DBLNTSCLORESDPFLACE_KEY
DBLNTSCLOREvSDPF2_KEY
DBLNTSCLORESDPF2LACEJKEY
DBLNTSCHIRES_KEY
DBLNTSCHIRE&FF-_KEY
DBLNrreCHIRESHAM_KEY
DBLNTSCHIRESHAMFF_KEY

DBLNTSC Hl RESLACE_KEY
DBLNTSCHI RESHAMLACE_KEY
DBLNTSCHIRESEHB_KEY
DBLNTSC Hl RESEHBFF_KEY
DBLNTSC Hl RES EHBLACE_KEY
DBLNTSCHIRESDPF_KEY
DBLNTSCHIRESDPFFFJÍEY
DBLNTSC H1RESDPFLACE_KEY
DBLNTSC Hl RES DPF2_KEY
DBLNTSCHIRESDPF2FF_KEY
DBLNTSCHIRESDPF2LACE_KEY
DBLNTSC EXTRALORES_KEY
DBLNTSCEXTRALORESHAM_KEY
DBLNTSCEXTRALORESEHB_KEY
DBLNTSCEXTRALORESDPF_KEY
DBLNTSCEXTRALORESDPF2_KEY
DBLNTSCEXTRALORESFF_KEY
DBLNTSC EXTRALORES HAMFF_KEY
DBLNTSCEXTRALORESEHBFF_KEY
DBLNTSCEXTRALORESDPFFF_KEY
DBLNTSCEXTRALORESDPF2FF_KEY
DBLNTSCEXTRALORESI^iCE_KEY
DBLNTSCEXPRALORESHAMLACE_KEY
DBLNTSCEXTRALORESEHBLACE_KEY
DBLNTSCEXTRALORESDPFLACE_KEY
DBLNTSCEXTRALORESDPF2LACE_KEY

A DBLPAL monitor ID-jei:

DBLPAL_MONITOR_ID
DBLPALLORES_KEY
DBLPALLORESFF_KEY
DBLF^ALLORESHAM.KEY
DBLPALLORESHAMFF_KEY
DBLPALLORESEHB_KEY
DBLPALLORESEHBFF_KEY
DBLPALLORESLACE^KEY
DBLPALLORESHAMIJ\CE_KEY
DBLPALLORESEHBIJVCE_KEY
DBLPALLORESDPF_KEY
DBLPALLORESDPFFF_KEY
DBLPALLORESDPPLACE_KEY
DBLPALLORESDPF2_KEY
DBLPALLORESDPF2FF_KEY
DBLPALLORESDPF2I^\CE_KEY
DBLPALHIRES_KEY
DBLPALHIRESFF_KEY
DBLPALHIRESHAM_KEY

DBLPALHIRESHAMFF_KEY
DBLPALHIRESLACE_KEY
DBLPALHIRESHAMLACE_KEY
DBLPALH1RESEHB_KEY
DBLPALHIRESEHBFF_KEY
DBLPALHIRESEHBLACE_KEY
DBLPALH1RESDPF_KEY
DBLPALHIRESDPFFF_KEY
DBLPALHIRESDPFLACE_KEY
DBLPALHIRESDPF2_KEY
DBLPALHIRESDPF2FF_KEY
DBLPALHIRESDPF2LACE_KEY
DBLPALEXTRALORES_KEY
DBLPALEXTRALORESHAM_KEY
DBLPALEXTRALORESEHB_KEY
DBLPALEXTRALORESDPFJÍEY
DBLPALEXTRALORESDPF2JÍEY
DBLPALEXTRALORESFF_KEY
DBLPALEXTRALORESHAMFF_KEY
DBLPALEXTRALORESEHBFF_KEY
DBLPALEXTRALORESDPFFF_KEY
DBLPALEXTRALORESDPF2FF_KEY
DB LPALEXTRALORESLACE_KEY
DBLPALEXTRALORESHAMLACE_KEY
DB LPALEXfRALORESEHBLAC E_KEY
DBLPALEXTRALORESDPFLACE_KEY
DBLPALEXTRALORESDPF2LACE_KEY

A felsorolt definíciók helyett azonban rugalmasabbnak tűnik, és ésszerűbbnek, ha programunk a screen megnyitása előtt megkérdezi egy „Modeld" requesterrel. hogy milyen monitorlípussal, és milyen felbontásban is szeretnénk látni. Természetesn megajánlva az aktuális WB screen adatait De erre majd az ASL libray tárgyalásánál látunk példát.

SA_DClip: definiálhatjuk azt a négyzetet, amely ún. élő része a képernyőnknek.

SA_Owerscan: ezzel, és az ezt megelőzővel tulajdonképpen ugyanazt tudjuk definiálni, mint a Prefs-ben az Overscan programmal, csak itt a saját képernyőnkre vonatkozóan.

SA_Obsolotel: obsolote SJV1ON1TERNAME.

Az itt következők csak boolean típusúak.

SA_ShowTitle: ugyanaz, mint a SHOWTITLE flag.

SA_Behind: ugyanaz, mint a SCREENBEHAIND flag.

SA_Quiet: ugyanaz, mint a SCREENQUIET flag.

SA_AutoScroll: ugyanaz, mint az AUTOSCROLL flag.

SA_Pens: mutató egy WORD tömbre, ameiyöen a ~o van. A Dwwinfo keresi. Ha ezt nem adjuk meg. a screen-únknek nem lesz olyan mint a többi 2.0-ás Screen-ek.

SA_FullPalette: segítségével inicializálható a színpaletta, amelyet a preferences-ből venne. **GetColorMapO** függvénnyel ez lekérdezhető.
SA_ColorMapEntries: segítségével felülírható egy eleme a ColorMap-nak a screenünkön. (csak V39-töl)
SAJParent: pointer a „parent" screenre. (csak V39-től)
SA_Dragable: boolean változó, akkor használjuk, ha a nem dragable-t akarjuk beállítani. Ne használjuk feleslegesen! (a defaultja a TRUE) (V39-IŐ1)
SA_Exclusive: boolean értékű, és megadásával a screen-ünk rejtett screen lessz. A default értéke FALSÉ. (V39-től)
SA_SharePens: boolean változó, amely a használt tollakat nyilvánossá teszi az inluilion számára. Ezért a default értéke a TRUE. (V39-IŐ1)
.SA_BackFill: megadhatjuk a „BackFill Hook"-ot a screen-ünkhöz. (V39-től)
SAJnterLeaved: boolean típusú, és arról tájékoztatja az intuitiont. hogy a használni kívánt BitMapunk interleaved. Defaultja a FAIJ3E. (V39-től)
SA_Colors32: ezzel a taggal beállíthatjuk, hogy a screen színeit 32 bitesen töltse fel. (csak V39-től)
SA_VideoControl: pointer egy tag listre. amelyet az intuition át ad a graphics.library/VideoControlO-nak, a screen létrejöttekor. (csak V39-töl)
SA_FrontChild: segítségével a „child" screen pointere adható meg, amely a család összes screen-je előtt fog mozogni mikor a screen megnyílik, (csak V39-től)
SA_BackChild: segítségével az a „rhild"-screen pointere adható meg. amely a család screen-jei mögé fog menni mikor a screen megnyílik, (csak V39-től)
SA_LikeWbrkbench: aktivizálni kell ahhoz, ha azt szeretnénk, hogy a screen-ünk pont olyan tulajdonságú legyen, mint a Workbench screen. (csak V39-től)

Szerintem a fentiekből kitűnik, hogy itt is beállíthatunk mindent, ha szükségünk van rá. Használata hasonlít az ablakoknál bemutatottá., de az egyszerűség kedvéért nézzünk rá egy példát is. (A lemezen is található példa.) A példában megnyitott screen 320x200, 3 bitplan-nel. és fejléce az alábbi lesz. "Az én Screen-em. "

```
my_screen = OpenScreenTags(NULL, /* Ez a kezdete */
                SfLJitle, "Rz én Screenem",
                SH_Pens, &DriPens[O],
                SH_UJidth, 320,
                SH_Height, 200,
                TflG_END /* Ez a uége */);
```

A my_screen természetesen egy mutató a Screen struktúrára. Ha értéke NULL, akkor nem sikerült megnyitnia. A sikertelen megnyitás okairól az

SA_ErrorCode-ból kapunk részletes információt. A **DriPensQ** egy olyan tömb. amely a screen külalakját határozza meg. Ha nem adjuk meg. akkor a screen-ünk olyan lesz. mintha régebbi Amigákon készült volna. (1.2. 1.3)

A lemezmellékleten minden részhez megtalálhatók a példaprogramok, melyek sok megjegyzést tartalmaznak, így nem okozhat problémát megértésük. Természetesen az Ablakok_és_Képernyők könyvtárban keresendőek.

A könyvtárba belépve két alkönyvtárba botlunk. Az egyik a C forrásokat, és lefordított állományokat tartalmazza, míg a másik az assembly-ben megírtakat. Mind a két részben elég csak a forrás ikonjára kattintani, és máris szerkeszthetjük azokat, vagy kipróbálhatunk más opciókat azok beírásával, és újrafordításával. Nem árt egy-két leírt dolgot kipróbálni.

Természetesen mindez csak abban az esetben van így, ha a SAS/C fordítót felinstalláltuk a gépre, és ugyanez a helyzet az ASM-One-val is. De az eddigi C- és leírásokról térjünk át a fent említettek megvalósítására assemblyben.

1.5.5 *Mindezek Assembly-ben*

Assemblyben sajnos sok kényelmes dologról nekünk kell gondoskodni, mint például arról, hogy hogyan indítottuk a programunkat, a struktúrák nehézkes kezelése, az egyes báziscímek megfelelő használata stb. Tehát nem lesz könnyű dolog szép és főleg hibátlanul működő programot írnunk.

Ez a rész ehhez nyújt egy kis segítséget.

Először is az AsmOne egy-két apróságáról szólnánk. Ajánlott a programunkban az eredeti C assembly include-okat használni, így sokkal könnyebb lesz. és az egyes rutinok hívásai is érthetőbbek lesznek, nem csak egy negatív számot fogunk látni. Ezen azt értem, hogy például ne -552 legyen az érték, hanem LVOOpenLibrary-t írjunk helyette. Ehhez persze kell egy LVO3.0 file csokor is, ami tartalmazza ezeket. A másik lényeges dolog az, hogy az include-okban a kis- és nagybetűk más-más szerepet játszanak (persze, hiszen C-hez vannak), így ha az AsmOne-ban nem kapcsoljuk be azt. hogy a kis- és nagybetűket megkülönböztesse, akkor Double Symbol hibával le is fog állni a fordítás.

Most áttérünk a konkrét programozási oldalra. A programunknak először is az ExecBase-t kell kiszedni a $4-es címről. Ez nem okozhat gondot, az összes példaprogramban benne van, és ezt már tárgyaltuk is. A következő bukkanó a könyvtárak megnyitása lesz. Valamilyen sorrendben a használt könyvtárakat meg kell nyitnunk, és ezek báziscímeit el kell raknunk. Ha valamelyik báziscím esetleg nulla lenne, akkor hiba történt. Ez még nem is olyan nagy baj. ki kell lépni a programból és ez már okozhat egy kis gubancot. Ez alatt azt kell érteni, hogy a már megnyitott könyvtárakat le kell zárni. Ezt úgy érdemes megcsinálni, hogy a megnyitási sorrendet megfordítva lezárjuk őket. így csak a megfelelő helyre kell beugrani, és a megnyitottakat sorban le is zárhatjuk.

Ezt így lehet egyszerűen szemléltetni:

megnyit dos.library
ha hiba ugrás A

megnyit graphirs.library
ha hiba ugrás 13

megnyit intuilion.library
ha hiba ugrás C

saját program

lezár intuition.library

C: lezár graphics.library

B: lezár dos.library

A: kilépés

Erre a legjobb példa a Elso_Ablakom.s program a lemezmellékleten, ebben semmi más nincs, csak néhány könyvtárnyitás és egy ablaknyitás.

Most. hogy már a könyvtárak meg vannak nyitva, folytassuk a screen megnyitással. Ehhez kétféle utat próbálhatunk meg bejárni, az egyik minden KickStart-ban megtalálható, az OpenScreen. a másik a 2.0-ás rendszertől, az OpenScreenTagList. Az OpenScreen-nek egy struktúrát kell létrehoznunk, melynek kötött a hossza, és az egyes elemek változó méretűek. Valahogy így néz ki:

```
my_neiu_screen: dc.iu   0                   ; LeftEdge
                dc.uj   0                   ;TopEdge
                dc.w    328                 ; Width
                dc.w    200                 ; Height
                dc.iu   3                   ; Oepth
                dc.b    0                   ; DetailPen
                dc.b    1                   ; BlockPen
                dc.iii  0                   ; UiewModes
                de.tű   CUSTOMSCREEN        ;Type
                dc.l    e                   ; Font
                dc.l    my_screen_name      ; Title
                dc.l    0                   ;Gadget
                dc.l    0                   ; BitMap

my_screen_name: dc.b    "Hz első screen-ünk!",0
```

Ez az egyik lemezmelléklet példájából van kiragadva (Screenl.s) és egy 320x200, 8 színű sereen-t nyit. Ennek a struktúrának a kezdőcímét az AO regiszterbe kell tenni és úgy kell meghívni az OpenScreen-t. Ez ha sikerült, akkor a DO regiszterbe egy mutatót ad vissza, mely egy Screen struktúrára mutat. Ezt gondosan tegyük el egy memóriacímre, mert később szükségünk lesz rá.

A másik mód az OpenScreenTagList. Itt assembly-ben nincs külön OpenScreenTags. ez csak C-ben van a kényelem kedvéért. Ennél is egy struktúra címét kell megadni, de ez a struktúra nem kötött méretű és csak long elemekből áll.

Például:

```
screen_tags:  de.l   SR_LUidth, 320
              de.l   SR_Heíght,200
              de.l   SR_Depth,4
              de.l   Sfl_Pens, pens
*             de.l   SR_Type.CUSTOMSCREEN
              de.l   SR_Quiet,1
              de.l   TRG_DONE
```

Mint ahogy írtuk, nem kell minden tag-ot inicializálni, mert mindegyiknek van egy defaull értéke. Az OpenScreenTagList is DO-ba ad vissza egy mutatót, ez is egy Screen struktúrára mutat.

A megnyitott screen lezárása a CloseScreen-el történik (mindkét esetben) és a megnyitáskor visszaadott Screen struktúra címét kell neki megadni a DO regiszterbe. A lezárással is vigyáznunk kell. mert a screen megnyitása nem biztos, hogy sikerült (tehát azt is figyelni kell) és nem ajánlatos megadni nullát.

Ha már megvan a screen. akkor nyithatunk rá ablakokat. Erre ismét két lehetőségünk van: OpenWindow és a OpenWindowTagList. Az OpenWindownál ismét egy kötött struktúrát kell megadnunk, ami így néz ki:

```
my_neuj_ujindou>: de.W  50                    ; LeftEdge
                  dC.LU 25                    ;TopEdge
                  de.ói 250                   ; lLMdth
                  dc.w  100                   ; Height
                  dc.b  0                     ; DetailPen
                  dc.b  1                     ; BlockPen
                  de.l  0                     ; IDCMPFlags
                  de.l  LUFLG_SMRRT_REFRESH   ; Flags
                  de.l  0                     ; FirstGadget
                  de.l  0                     ; ChetkMark
                  de.l  my_ujíndoLU_name      ; Title
                  de.l  0                     ;Screen
                  de.l  0                     ; BitMap
                  dc.w  0                     ; Minlüidth
                  de.ni 0                     ; MínHeight
                  dc.w  0                     ; MaKWidth
                  de.ni 0                     ; ManHeight
                  de.lúJ lilBENCHSCREEN       ; Type
my_iuindoui_name: dc.b  "Rz én első ablakom",0
```

Ha a Screen mező nulla, és a Type WBENCHSCREEN, akkor a workbench ablakra lesz nyitva az ablak, ha a Type CUSTOMSCREEN vagy PUBLICSCREEN. akkor a Screen mezőbe az OpenScreen vagy OpenScreenTagList által visszaadott értéket írjuk:

```
moue.l  #my_neui_UJindouj,a0
        moue.l  my_screen,30(aB)
```

Az ablak lezárása - bármelyik módon nyitottuk - a CloseWindow rutinnal kell lezárni. Itt már vigyáznunk kell a sorrendre, az ablakot a screen lezárása előtt kell becsukni.

Most már csak néhány tanácsot fogunk leírni, ami a nem működő pfogramok esetén jól jöhet:

- a báziscímet mindig az A6 regiszterbe tegyük és lehetőleg saját programunk azt ne kavarja el.
- minden megnyitott library-t zárjunk le! Ha ez nem történne meg. akkor nem lesz semmi baj. de a szabad memória kevesebb lesz.
- a library-k nevét mindig kis betűkkel kell írni és nullával kell lezárni - program végén a DO regiszter tartalmazza OS számára a visszatérési kódot, mely batch programot vezérelhet (elágazásoknál). Ha a Failat értékét meghaladja, akkor a batch program végrehajtása meg fog szakadni. Magyarul ha kilépünk egy programból akkor a DO legyen nulla, feltéve ha nem elágazást akarunk csinálni.
- mindig teszteljük le, hogy az egyes rutinoknál a visszatérési érték nem-e nulla (vagyis hiba történt).

1.6. Gadget-ek

Ebben a részben megismerkedünk részletesen az ún. „Gadgel"-ekkel. Ezek azok a kis bizgentyük. amin az user érezteti a géppel, hogy mit is akar.

E/ekre kell az egerünkkel ráklikkelnúnk ahhoz, hogy működjön a program, vagy opciókat érhetünk el. A gadget-ek létrehozása igen egyszerű Amigán

Szinte mindent megcsinál helyettünk az Intiiition. Nekünk csak közölnünk kell óhajainkat, és teljesülnek azok. Ahhoz, hogy az egér mozgását ne nekünk kelljen állandóan figyelnünk, és amikor a gadgetünk fölé ér. plusz még a gombja is megvan nyomva, akkor szóljon, hogy a/t az oda vonatkozó rutint indítsuk el.

Mivel azonban az egér mozgását nem nekünk kell figyelni, hanem csak egy üzenetet kapunk az Intuitiontól. hogy megnyomták a gadgel-ünket. ezért ismerkedjünk meg vele. hogy hogyan is működik, hiszen lépten-nyomon használnunk kell majd. Hacsak nem oldjuk meg a dolgot más úton-módon. Ezt az úzenetrendszert Amigán **IDCMP-nek.** azaz Intuition's Direct Communikations Messages Ports-nak hivják.

1.6.1 Az IDCMP

Mint már fent említettük, ezt használja az Intuiüon. hogy nekünk üzengessen ül. mi is csak ezt használhatjuk ha neki üzenünk valamit. Ha például az Amigába behelyezünk egy lemezt, vagy megnyomunk egy gadget-et. **1111.** bármi történik a gépen belül, akkor egy üzenet **(Message)** jön létre automatikusan. Ha ilyen történik, az Intuition végignézi, hogy melyik az a program amely érdeklődik a létrejött üzenet után. és elküldi annak. Ezt onnan deríti ki, hogy minden ablaknál megadható egy ún. IDCMP flag. Ezt beállítva megadhatjuk, hogy mi is érdekelje a sok történés közül a mi programunkat ami ezt az ablakot nyitotta.

Nézzük meg a 6. ábrát, am'ejyen ezt próbáltuk szemléltetni. Az Intuition először a keresést mindig az éppen aktív ablaknál kezdi, majd végignézi az összesei. Ha tehát beraktunk egy lemezt a gépbe, majd megnyomtunk egy gadget-et az aktív ablakon, akkor az Intuition először megnézi, hogy az aktív ablakos program, ami most a „B" program, vár-e lemezt. Ha igen, küld neki egy üzenetet. Jelen esetben nem ő várta a lemezt, hanem az „A" program.

Tehát az üzenetet az „A" programnak küldi el. A „B" programnak azt az üzenetet küldi, amelyben a gadget megnyomásáról esik szó. mivel a „B" program ezt várta. Tehát más programok nem szereznek tudomást az őket nem érintő eseményekről, amelyek a gépben történnek. Ha azonban mindegyik programot éredekelné a lemezbetélel. akkor természetesen minden program kapna róla értesítést. Utána a multitask elintézné, hogy mindegyik program hozzá is férjen a lemezhez. A programok ezek után megnézik, hogy az a lemez >•„„ ^ w „ t .„„„„•„-„ „„kik cvíik<5Pitv'ik van A multitaszkot a gép olyan jól elintézi, hogy meglehetjük akár azt is, hogy egy lemezt akár ket, vagy lODD prog-

rammal is lehet formázni, és lehet mindegyik formátum más és más. Az más kérdés, hogy ennek mi az értelme. De ilyet más gépeken nemigen látni. (Windows alatt ez elképzelhetetlen. Az MS-DOS-ról nem is beszélve.) De lássuk, hogy hogyan is történik ez a gyakorlatban. Ahhoz, hogy üzeneteket kapjunk, es adjunk, a legelső lépés az, hogy kreálnunk kell egy un. IDCMP Port-ot, amelyen keresztül majd mindez lezajlik. Ez igen egyszerű, mivel ez automatikusan létrejön, amikor megnyitunk egy ablakot. A NewWindow struktúrában megadhatjuk, hogy milyen üzeneteket vegyen a programunk. Pédául, ha megadjuk neki az IDCMP_GADGETDOWN flag-et. akkor minden esetben mikor az ablakon lévő gadget-el aklivizál]ák. üzenetünk fog érkezni. Ha már megnyitottuk az ablakot és esetleg módosítani akarjuk azt, hogy mire jöjjön az üzenet, akkor használjuk a **ModifyIDCMPO** függvényt. (Az Intuitíonban található.) Ha ez megvan, a következő lépés az. hogy meg kell várnunk azt. hogy érkezik-e nekünk szóló üzenet. Ezt kétféleképpen tehetjük meg. Az egyik esetben a várás passzív, mivel a programunk nem csinál addig semmit ameddig nem történik valami. Például játékban, addig nem kell csinálnunk semmit, ameddig az user nem lépett (azaz nem aktivizált egy gadget-et). A másik esetben a program működése nem áll meg. hanem a program dolgozik, csak éppen néha megnézi nem-e jött neki üzenet. Pédául egy fraktál számító programból ha nem ezt használnánk, igen sokára tudnánk kilépni, mert meg kellene várni azt. hogy a program kiszámolja a képet. Ezt az utat nevezik aktív várakozásnak. Ha már tudjuk hogyjött üzenet, akkor fel is kellene azt dolgoznunk. Ez azzal kezdődik, hogy átvesszük az üzenetet. Ezt a **GetMsgO** függvény meghívásával tehetjük meg. Ha nem érkezett üzenet a számunkra, a függvény visszatérési értéke NULL lesz. Ha azonban érkezett nekünk üzenet, akkor annak címére fog mutatni, ahol az üzenetet tartalmazó struktúrát találjuk. Ez a struktúra az IntuiMessage struktúra lesz. Ahhoz, hogy ennek a struktúrának alapján ki tudjuk hámozni az üzenet mibenlétét, ismernünk kell a struktúrát. Tehát a struktúra az alábbi:

```
struct IntuiMessage
{
    struct Message ExecMessage;
    ULGNG Class;
    USHORT Code;
    USHORT Qualifier;
    HPTR IfIddress;
    SHORT MouseH, MouseV;
    ULONG Seconds, Micros;
    struct WindOLU *IDCMPLUindow;
    struct IntuiMessage *SpecialLink;
};
```

Ezen belül a jelentésük a következő:

ExecMessage: e/.t csak az exec használja, ne nyúljunk hozzá.

Class: ez tartalmazza az IDCMP flag-ot. és így tájékoztat bennünket a történés mibenlétéről, (pl. GADGETDOWN. ha egy gadget meg lett nyomva]

Code: ez egyéb információt tartalmaz a lenti Class-t kiegészítendő. Például ha megadtunk az balakunknak egy ilyen IDCMP flag-ot mint a RAWKEY. vagy a VANILLAKEY, amelyek a billentyűzetről adnak tájékoztatást. Ez lesz az a hely, ahonnan megtudhatjuk, hogy melyik is volt lenyomva a billentyűk közül.

gualifier: ha a program ilyen üzenetet vár mint a RAWKEY (ez a nyers billentyűkódot adja meg, nem a keymap szerintit), akkor innen kapunk tájékoztatást arról, hogy a billentyű lenyomásakor azzal egyidejűleg le volt-e nyomva pl. a Ctrl. vagy a Shift stb.

- **MouseX:** az egér x koordinátája, az ablak bal felső sarkához relatívan.
- **MouseT:** az egér y koordinátája, az balak bal felső sarkához relatívan.

Seconds: a rendszer órájának, a másodpercek másolata.

Microns: ugyanaz, csak az ezredmásodperceké.

IAddress: ha flag-nak a GADGETDOWN-t adtuk meg, akkor itt találjuk az aktivizált gadget struktúrájára mutató pointert.

IDCMPWindow: pointer arra az ablakra, amelynek az üzenet el lett küldve.

SpecialLink: ezt csak az exec, és az Infuition használja, ne nyúljunk hozzá.

Miután megkaptuk az üzenetet, válaszolnunk is kell rá. A válasz célja csak annyi, hogy nyugtázzuk az üzenet megérkezését, hogy az Intuition ne aggódjon. Ezt a választ azonban célszerű minél előbb megtenni, mivel ennek hiányában az Intuition képes a csigánál is lassabban mozogni, mivel addig másik programot nem képes keresni, hogy átadhassa a neki szóló üzenetet.

Ezért célszerű ha az üzenetet lemásoljuk, majd válszoljunk az Intuítionnak. és elkezdhetjük az üzenet kiértékelését. A válasz nagyon egyszerű, mert csak meg kell hívnunk egy Inluition függvényt. Ennek a függvénynek a neve, a **ReplyMsgfJ.** Ezzel a függvénnyel mondjuk meg az Intuitionnak. hogy az üzenetének az olvasását befejeztük. Nagyon fontos az. hogy a válasz üzenetként elküldött IntuiMessage struktúra NE legyen megváltoztatva, mert azt más programok vagy az Intuition használhatja, és így a nem valós üzenet fennakadásokat okozhat a rendszer működésében.

Nézzünk meg néhány példát az IDCMP kezelésére a gyakorlatban: kezdjük talán a Passzív várakozásos példával. Először is tehát várunk az üzenet érkezésére.

```
UUait(1« my_uJindow->UserPort->mp_SigBit);
```

Ezzel a sorral addig várunk, ameddig egy üzenet nem érkezik a számunkra. A my_window egy pointer a már nyitott abalakunkra. Ezután át kell vennünk a küldeményt, ami így néz ki:

```
my_message = GetMsglmy_uiinüoLu->userPort);
```

A my_message egy pointer típusú változó, amely IntuiMessage típusú struktúrára mutat, ha az üzenetet sikerült átvennünk. Ezt ellenőrizzük is le. hiszen a kiértékelésnek csak akkor van értelme, ha sikerült átvennünk az üzenetet.

```
if(my_message)
  {
        Ide téued, ha sikerült az üzenetet átuennie.
        H harmadik lépés, hogy a minket érdeklő uáltozókat
        elmentsük a későbbi felhasználásra.
        Ilyen uáltozók lesznek például a Gadget-ek esetén
        a Class, a Code, és az IRddress.

        class=my_message->Class;
        code=my_message->Code;
        address=my_message->IfIddress;

        Ha ezt megtettük akkor uálaszoljunk neki.
        Ne felejtsünk el a lehető leggyorsabban
        uálaszolni az Intuition-nak.

        ReplyMsg(my_message);

        H uálasz után már kiértékelhetjük, hogy mi is érkezett.

        sujitch(class)
        {
            case IDCMP_GHDGETDOUJN:   itt köuetkezik a program
                                      gadget lekezelője,
                                      ha a gadget-et megnyomták.
            case IDCMP_GHDGETUP:      itt köuetkezik a program
                                      gadget lekezelője, ha a
                                      gadget-et felengedték.
            case IDCMP_MENUPICK:      es itt köuetkezik a
                                      programunk menü kezelő
                                      része. De ezt majd később,
                                      és a többi, ha uan.
        }

  }
```

Mint látjuk, a programunk már észre fogja venni a neki küldött üzeneteket. De mit tesz akkor, ha nem csak egy üzenet fog a futása alatt érkezni? Nos. arról már nem fog tudomást szerezni. Emiatt ezt az egész részt nyugodtan berakhatjuk egy while ciklusba. Ebben az esetben a programunk minden üzenet vételére alkalmas lesz. míg csak a program befejezésére üzenetet nem kap. Ez lehet mondjuk a CLOSE Gadget aktivizálása is. Mikor egy üzenetet vett a programunk, és végrehajtotta a kimért feladatokat utána rára-

kozní fog a következő üzenetre. Addig azonban ún. alvást „sleeping" végez, azaz nem csinál mást. mint hogy vár. csak vár. és még mindig vár. egészen addig, ameddig nem kap egy üzenetet. Ezalatt nem foglal különösebb taskidőt, csak memóriát maga a program. Tehát a többi program teljes sebességgel száguldozhat. (:-)

Akkor lássuk, hogy milyen üzeneteket várhatunk el a rendszertől.

A Gadget-ekre vonatkozó IDCMP flag-ek:

IDCMP_GADGETDOWN: ilyen üzenetet kapunk, ha ablakunkon aktivizálódott egy gadget. (az egérrel rá kattintottak, lenyomták

) **IDCMP_GADGETUP:** ezt az üzenetet kapjuk ha. egy gadget-et felengednek az ablakunkon.

IDCMP_CLOSEWINDOW: az Intuítion ezt adja vissza, ha az ablak-„CloseGadget"-jét aktivizálta az user.

IDCMP_GADGETHELP: arról tájékoztat bennünket, hogy az user melyik gadget-röl kér Help-et (segítséget). Csak V36-UM felfelé létezik.

IDCMPJDCMPUPDATE: tájékoztat a „boopsi" gadgetről. (V36-tól)

A Requester-ekre vonatkozó IDCMP flag-ek. (A Requester-ékről bővebben a 1.9-es részben):

IDCMP_REQSET: a program foga kapni ezt az üzenetet, ha nyilt egy requester az ablakra.

IDCMP_REQCLEAR: ez az üzenet érkezik, ha az összes requester törlődött az ablakról.

IDCMP_REQVERIFY: az üzenet akkor érkezik ha a requester aktívvá vált. (Csak megjegyzésképpen, ha egy requestert nyitunk az ablakunkra, az összes üzenetet az Intuition megszűri, és csak a requesterre vonatkozót továbítja a programunk felé.)

A következők arra az esetre vonatkoznak, ha menüt csatoltunk az ablakunkhoz, ezeket kaphatjuk (A menükkel a következő fejezetben foglalkozunk):

IDCMP_MENUPICK: ezt kapjuk vissza, ha az user aktivizált *eűy* menüt. Azt hogy melyiket is, a Code-ból tudjuk majd meg. (lásd. következő fejezet.)

IDCMP_MENUVERIFY: ezt az értéket kapjuk, ha a menü aktivizálódott, az user molyolt vele.

A code változóban az alábbiak szerepelhetnek:

MENUHOT: az Intuition ellenőrizni akarja vagy MENUCANCEL.

MENUCANCEL: HOT üzenet jön, ha a menük mégsem voltak aktivizálva, csak nézegették őket.

MENUWAITING: az Intuition egy ReplyMsg()-t akar. Ez akkor szokott előfordulni, ha egy másik ablakon éXKUVÍAcIltdiK n m e n ü l í r l .

IDCMPJMENUHELP: csak V36-oslói, és azt jelzi, hogy melyik menüről kéri az user a segítségei (Help).
Az egér 1DCMP fingjai.

IDCMPJMOUSEBUTTONS: programunk ezl az üzenetei kapja, ha az egér valamelyik gombjai lenyomták. A Code-ból olvashatjuk ki. hogy ez miként is történhet. Az alábbiakat kaphatjuk.

SELECTDOWN: a bal egérgomb lenyomott volt.
SELECTUP: a bal egérgomb volt realizálva.
MENUDOWN: a jobb egérgomb lenyomott volt.
MENUUP: a jobb egérgomb volt realizálva.
MIDDLEDOWN; a középső egérgomb volt lenyomva.
MIDDLEUP: a középső egérgomb volt realizálva.

(Figyelem! Ha az user lenyomta az egér bal gombját bárhol egy gadgeten belül, a program NEM kap erről üzenetet. Akkor sem kapunk ilyen üzenetet, ha az ablakon az egérrel egy menüt aktivizállunk. Ha akarunk kapni üzenetet a fenti esetben is. akkor be kell állítanunk a RMBTRAP flag-ol a NewWindow struktúránkban. De ezen az ablakon nem tudunk menükel is kezelni!)

IDCMP_MOUSEMOVE: az üzenet akkor érkezik, ha az egeret elmozdították. Ahhoz, hogy ez így is legyen, be kell állítani a REPORTMOUSE flag-ot. az ablak flagjai között (NewWindow struktúrában), vagy egy gadget szükséges az ablakban, a FOIXOWMOUSE falg-gal. az aktív mezőjében.
IDCMP_DELTAMOVED: ebben az esetben olyankor is szól. ha a pointerünkkel a mozgatás közben már elértük a képernyő borderét (keretét).

Az ablakai kapcsolatos IDCMP-ék:

IDCMP_NEWSIZE: a programunk ezt kapja, ha az ablak méretét megváltoztatja az user.
IDCMP_SIZEVERIFY: ezt kapjuk, ha az ablak méretét próbálta megváltoztatni.
IDCMP_REFRESHWINDOW: ezt akkor kapjuk, ha az ablakunkat szükséges frissítenünk. Ezt kaphatjuk a SIMPLE_REFRESH. vagy a SMART_REFRESH ablakok esetén. Akkor is kaphatunk, ha nem kell feltétlenül meghívnunk a **BeginRefreshfJ** és az **End RefreshfJ** függvényeket.
IDCMP_ACTIVEWINDOW: ilyen üzenetet kapunk, ha az ablakunkat aktivizálták.
IDCMP_INACTIVEWINDOW: ezt meg akkor kapjuk, ha az ablakunk inaktívvá vált. tehát az user más ablakot aktivizált.
IDCMP_CHANGEWINDOW: Ezt az üzenetet kaphatjuk, bármilyen változás történt az ablakunkon. (V36-os esetén!)
A Code változó az alábbiakat veheti fel:

CWCODE_MOVESIZE: ha az ablakot mozgatták, vagy a méretét változtatták.
CWCODE_DEPTH: az ablakon a depth-et aktivizálták. (V39-IÓ1)

A többi IDCMP-ek:

IDCMP_RAWKEY: A programunk akkor kapja ezt az üzenetei, ha egy billentyűt megnyomott az user. A RAWKEY kódokat megtaláljuk a könyv végén, a függelékben.

IDCMP_VANILLAKEY: ezt szintén a billentyű megnyomásakor kapjuk, csak a billentyűzetkód aszerint változik, hogy a gépen melyik billentyűzethez melyik karakter van rendelve (lásd. Keymap -> AmigaDos).

IDCMP_DISKINSERTED: ez az üzenet érkezik, ha egy lemezt helyeztek be a gép bármelyik meghajtójába. Az üzenetet hallani fogja az összes olyan program, amely erre be van állítva.

IDCMP_DISKREMOVED: ha kivették a lemezt, ilyen üzenet érkezik. Szintén eljut minden programhoz.

IDCMP_NEWPREFS: ilyen üzenetet kapunk, ha a rendszer Prefs-jében átállítottak valamit (vagy az user, vagy más program).

IDCMPJNTUITICKS: ilyen üzenetet másodpercenként megközelítőleg tizet kaphatunk.

IDCMP_WBENCHMESSAGE: ezt az üzenetet akkor kaphatjuk, ha egy program használja az OpenWorkBenchfJ **111.** a CloseWorkBench() függvényeket. Csak a rendszer használja!

A code változó ezeket az értékekel veheti fel:
WBENCHOPEN: Ha a Wb nyitott.
WBENCHCLOSE: Ha a Wb be van zárva.

Az IDCMP használatát a lemezen lévő példaprogramok jól szemlétetik. Mielőtt belefognánk a Gadget-ek ismertetésébe, meg kell néznünk, hogy hogyan is írhatunk az Intuition segítségével. Hiszen egy gadget-be bele kell írni, hogy például mire is szolgál. Persze ezzel a módszerrel lehetőségünk van az ablakunkra bármit is írni. A lemezmellékletünkön találhatunk egy-két példát az IDCMP kezelésére. A Gadget-ekkel és menükkel kapcsolatos példák a Gadget-eknél ill.a menüknél találhatók.

1.6.2. Az Intuition grafikus lehetőségei

Bár Amigán az Intuition nem felelős a grafikáért, azonban közvelve mégis kezelnie kell. hiszen a Gadget-ek, a menük stb. használhatnak grafikát.

Ezért az Intuition is képes kevés grafika kezelésére. Ezek a grafikai lehetőségek messze nem merítik ki az Amiga lehetőségeit. Nem is volt cél azt az Intuitionból elérhetővé tenni. Az Intuition rajztehetsége tehát csak keretrajzolásra használható, vonalak húzgálásából. Ili. szövegek megjelenítését szolgáló, vagy közvetlenül grafikai ikonok kezeléséből áll.

Ezekkel az eszközökkel mindent meg lehet valósítani, ami az Intuition feladatának ellátásához kell. Ezért nézzük meg őket részletesebben is.

1.6.2.1. Az IntuiText, avagy írjunk az Intuition-nal

Mivel Amigán nincs külön karakteres **111** grafikus képernyő mint a PC-n. így egy kicsit komplikáltabb a betűk megjelenítése. PC-n elég volt a kiírandó betű ASCII kódját a megfelelő memóriaterületre írni. a többit a hardver elintézte, így volt ez a jó öreg VIC20 esetén is.

Ott is elég volt a képernyő memóriájába egy ASCII-nek megfelelő számot írni. ami megjelent. Amigán ez egy kicsit másképpen megy. Ha belegondolunk, rögtön belátjuk, hogy a kiírásnál biztosan meg fogjuk adni azt, hogy melyik képernyőre, ül. melyik ablakra szeretnénk a szöveget kiírni.

Természetesen meg kell még adnunk azt is. hogy ez hová is történjen, és hogy milyen betűtípussal is. Valamit az sem mindegy, hogy milyen színnel.

Erre az Amiga szintén egy struktúrát használ, ami az IntuiText névre van keresztelve. Ebben a struktúrában a fentieken kívül megadhatunk még egy ugyanilyen struktúrát megcímző mutatót is. Ennek akkor van jelentősége, ha például az ablakunkra" több kiírást is szeretnénk tenni, de frissítéskor nem akarjuk mindet külön-külön újra kiíratni. De lássuk ezt részletesen:

```
struct IntuiText
{
    UBVTE FrontPen, BackPen;
    UBVTE Drau/Mode;
    WORD LeftEdge;
    WORD TopEdge;
    struct TentRttr *ITentFont;
    UBVTE *ITent;
    struct IntuiText *NeHtText;
}
```

Ahol is a **FrontPen** a szöveg színét, a **BackPen** a hátterének a színét jelöli ki. A DrawMode a rajzolási módot jelöli ki. (JAM1. JAM2 stb. lásd. Alacsonyszintű grafika (1.9) rész.) A **LeftEdge** a szöveg x koordinátáját, míg a **TopEdge** annak y koordinátáját adja meg. Az **ItextFont** mutató a kiírás szövegének betűtípusára. Az Itext mutató a kiírandó szövegre, aminek nullával kell végződnie.

Valamint a **NextText.** amely a következő ilyen struktúrát címzi meg. Használatával nagyon egyszerűen megjeleníthetünk árnyékolt szöveget is az ablakunkon, vagy screenünkön. A megjelenítés nagyon egyszerű, ha inicializáltuk a struktúránkat: csak meg kell hívni az Intuition **PrintITextO** függvényét, amely kirajzolja a szövegünket. A függvényről részletesen írunk az Intuition könyvtárnál hátul a könyvben.

1.6.2.2. A Borderek

A Border-ekkel lehetőségünk van keretek, ill. más célból vonalak húzására. Általában csak keret húzásra szokták használni, de ha van fantáziánk, másra is használhatjuk. Abban, hogy szinte korlátlan lehetőségünk van a vonalak húzgálásában. jelentős szerepet játszik az. hogy mint annyi mindent, ezeket is struktúrában adhatjuk meg. Természetesen a struktúrák egymásba ágyazhatóak. így kirakásukról egyetlen függvény gondoskodhat, vagy egy objektumként hozzárendelhető más Intuition objektumokhoz, mint például a Gadget-ekhez. Tekintsük meg a Bordér struktúrái:

```
struct Bordér
{
    SHORT LeftEdgeJopEdge;
    SHORT FrontPen, BackPen, DraiuMode;
    SHORT Count;
    SHORT *HY;
    struct Bordér *NextBorder;
};
```

Mint látjuk, a struktúra igen hasonló az InluíText struktúrára. Azzal a különbséggel, hogy itt nem szöveget adunk meg. így nincs a betű típusára utaló, sem a szöveget tartalmazó mező. De nézzük a mezők jelentését sorban. Az első a **LeftEdge,** ahol a rajzolás kezdeti x irányú koordinátát definiálhatjuk. A második a **TopEdge,** ahol természetesen az y koordinátát adhatjuk meg. A **FrontPen** a rajzolás színét van hivatva eldönteni. A **BackPen** a rajzolás háttérszínét határozza meg. A **DrawMode** a rajzolás módját mondja meg. Részletesen itt erre nem térünk ki, hiszen az alacsony szintű grafikánál ez fellelhető. A **Count** mezőben azt közöljük az Intuilionnal. hogy az XY koordinátákat tartalmazó tömb hány elemből áll

Mivel a Border-nek a vonalak húzásához szükséges koordinátákat párban adjuk meg (x.y), és ezek tárolására tömböt használunk. így ez a szám éppen a fele lesz a tömbben tárolt elemek számának. Az XY tehát erre a tömbre mutató pointert vár. A **NextBorder** mezőben pedig a következő Borderünk struktúrájára mutató pointert adhatjuk meg, ha van. Ha nincs, akkor NULL az adandó érték. A Bordér tulajdonképpen nem más. mint a régebbi gépeknél alkalmazott turtle (teknó'sbéka) grafika megvalósítása. Azaz a teknősnek az adott vonal húzásához a pont koordinátáit kell megadni. A vonal húzását az előző ponttól kezdi, és folyamatosan húzza ameddig a végére nem ért. Ha tehát például egy keretet akarunk rajzolni, azt a következő adatok megadásával érhetjük el. Pl. rajzolást kezdjük a bal felső saroktól és az óra mutatójával megegyező irányba rajzoljuk azt meg. A gépnek azaz a teknősnek tökmindegy, hogy merről rajzolja és hogy honnan kezdi. Tehát az első koordinátapárunk a 0.0 lesz. Ez a bal felső sarok koordinátája. A követ*icacro* magunkkal való megegyezés alaiiján a jobb felső sarok lesz. amely 100 pixeles szélességet feltételezve éppen 100,0 lesz. Ezek után a jobb alsó sarok következik. 100 pixeles magasságot megengedve ez 100.100 lesz. Már

csak a negyedik pont van csak hátra, amely a 0,1 OO-as koordináta-értéket kaphatja. Még nem vagyunk kés/en. hiszen a négyzetünk nem teljes, mivel az utolsó oldal hiányzik belőle. E/ért az utolsó koordináta a O.O-ás értéket kapja. Ne feledkezzünk meg arról, hogy a vonalakat egymással össze kell kötni. Ha ezt nem akarjuk, akkor több Bordér struktúrát láncoljunk össze (lásd. NextBorder mező a struktúrában). A példa alapján ne gondoljunk arra, hogy a vonal utolsó elemének összekötöttnek kell lennie az elsővel. Erre csak akkor van szükség, ha olyan zárt alakzatot akarunk létrehozni, mint a négyzet, téglalap, háromszög, stb. A tömbben szereplő 0,0-ás pont természetesen relatív a Bordérben definiált LeftEdge. és TopEdge által meghatározott pontjához képest. A 0.0 esetében az ott meghatározott koordinátájú ponttól kezdi. De lássunk egy példát erre a sok betűre. A pédában egy nem összefüggő alakzatot rajzolunk a bordér segítségével. A példa az alábbi:

```
/* Definiáltunk egy tömböt a koordináta párjainkat tárolandó */
USHORT pontjaink!] =
{
    10,10,    /* flz első pont K,y koord.-ja */
    30,10,    /* Ebből húzunk egy uonalat jobbra a 30,10 pontba. */
    30,40,    /* Ebből húzunk egy uonalat lefelé a 30,40 pontba. */
    20,40,    /* Ebből húzunk egy uonalat balra a 20,40 pontba. */
    34,-10,   /* Ebből felfelé ferdén egy uonalat a 34,-10 pontba.*/
};

struct Bordér Borderem =
{
          0,0,   /* Kezdjük a 0,0 ás ponttól. */
            3,   /* FontPen, fl narancs színnel rajzolunk (1.3),
                    ill. kékkel (2.0) */
            0,   /* BackPen, H 0-ás színregiszter a hátterünk. */
       JHM1,     /* DrawMode, fl rajz nem uáltoztatja meg
                    a hátteret. */
            5,   /* Count, 5 pár koordinátánk uan, az az 5 pontból
                    áll az ábra. */
&pontjaink,      /* KV, fl koordináta párokat tartalmazó tömb
                    pointere. */
          NULL,  /• NextBorder, Nincsen másik borderünk. */
```

Most már rsak azt kell elérnünk, hogy a Bordert kirajzolja a képernyőre. Ezt egy az Intuitionban található függvénnyel végeztethetjük el. Ez a függvény a **DrawBorderfJ** .Paramétereként először a rajzolási helyet (ablak, screen) várja, egy RastPort címét (ez az ablakunk struktúrájából egyszerűen kinyerhető: ablakunk->RPort). Majd egy pointer kell megadnunk, amely a Bordér struktúránkra mutat. Ezek után már csak a kirakás kezdeti koordinátáit adhatjuk meg. A koordinátákhoz természetesen relatívvá válik az egész.

1.6.2.3. Az Image-ek

Az eddigiekben megnéztük, hogy hogyan lehet szöveget kiíratni az Intuiüon segítségével, vagy hogyan tudunk vonalakat rajzolni. Most azt nézzük meg. hogy egy már megrajzolt ábrát hogyan tudunk kitenni a képernyőre. Ehhez először is meg kell terveznünk az ábrát. Ebben sokat segíthet az. ha egy rajzolóprogrammal tesszük ezt. Miután megrajzoltuk az ábránkat, olyan formátumúra kell hoznunk, hogy azt a gép is megértse. Ez a formátum tulajdonképpen nem különleges formátum, csak a megjeleníteni kívánt pixeleket tartalmazza olyan formában, hogy az első bitplane adatait követi a második bitplane-é és így tovább, mint ahogyan ezt az 1.6.3. részben már megnéztük. Természetesen az ott elmondottak ebben az esetben is állnak, miszerint az adatoknak a CHIP RAM-ban kell lenniük ahhoz, hogy meg tudjuk jeleníteni őket. Ezek után ezt az adatot meg kell adnunk magában az Image struktúrában, mert hogy az Image-hez is tartozik egy struktúra. A szerepe csupán a szokásos, megjelenítési formák közlése az Intuitionnal. Az Image struktúra a következő:

```
struct Image
{
    SHORT LeftEdge,TopEdge;
    SHORT lilídth, Height, Depth;
    SHORT *ImageData;
    UBVTE PlanePick, PlaneOff;
    struct Image *NextImage;
};
```

A struktúra tagok jelentése:

A **LeftEdge** és a **TopEdge** a kezdeti koordináták megadását várja. A **Width** és a **Height** a szélességét és a magasságát adja meg. valamint a **Depth** a szükséges bitplane-ok számát tartalmazza. Ezen értékeknek az ImageData-ra nagy befolyással bírnak, ugyanis a megadott értékeknek megfelelően próbálja erielemezni a megadott tömb tartalmát. Az ImageData pointer az adatokat tartalmazó tömbünkre. A **PlanePick** megadása azt közli az Intuitionnal. hogy a tömbben az l-essel inicializált bitek esetén mi is történjen. Ha például egy egy bitplanes képet szeretnénk megjeleníteni, és a tömb tartalmazza az adatokat, ennél a változónál deklarálhatjuk azt. hogy a tömben l-est tartalmazó részeken milyen színnel jelenjen meg. Ha például azt szeretnénk, hogy a kép narancs színű legyen és ezt a Workbnech képre rakjuk ki, akkor az egyes biteknek a Workbench képernyőn a 2 bitplanes képnek kellene lennie. Ennek ellenére a mi képünk csak 1 bitplane adataival rendelkezik. Ezzel a változóval azonban megadhatjuk, hogy a kép mégis narancs színben pompázva jelenjen meg a screenen. A megadás az alábbi logikn Qí'erint történik'

Bitplane amelyiken töténik:	PlanePick
Nem történik semmi	0000 = 0
Pláne 0	0001 = 1
Pláne 1	0010 = 2
Pláne 1 és 0	0011 = 3
Pláne 2	0100 = 4
Pláne 2 és 0	0101 = 5
Pláne 2 és 1	0110 = 6
Pláne 2,1 és 0	0111 = 7
Pláne 3 "	1000 = 8
Pláne 3 és 0	1001 = 9
Pláne 3 és 1	1010= A
és így tovább ...	

Ha azt akarjuk, hogy az 5,4.1 és a 0-ás színnel jelenjen meg. és az egész a 2 és a O ás bitplanen történjen, az alábbiak szerint járhatunk el:

000=0 szín, 001 = 1 szín. 100=4 szín. 101=5 szín

210bitplan 210bitplan 21ObitpIan 210bitplan

tehát a PlanePick változóba az 5 (0101) szám kerül. A PlaneOff változó hasonló az előzőhöz azzal a különbséggel, hogy a háttér színét állíthatjuk be a fent mégtárgyalt módon. Ha azt akarjuk, hogy az Image a 2 bitplanre kerüljön az 5 színnel, és a háttere az 1 színű legyen, akkor a következőket kell hogy beállítsuk: PalnePick 4 (0100) és az 1 es bitplane-t töltse fel 1, tehát PlaneOff 1 (0001). Ezen két változó variálásával el tudjuk érni egy terület törlését, ill. színezését is. Hiszen csak a méretet kell megadnunk, (TopEdge.LeftEdge.Width.Height), majd a Depthet 0-ra állítjuk (Nincs Bitplane), és a PlanePick változóba O-át írunk, valamint a PlaneOff változóba annak a színnek a számát adjuk meg. amellyel a színezést akarjuk végrehajtani. Az utolsó mező a **Nextlmage,** a következő Image struktúrára mutathat, vagy NULL.

Az Image megjelenítését egy függvény végzi el. amelynek a neve **DrawImagefj.** A függvénynek az ablak, screen rastport megadása után az Image struktúra címét kell átadnunk. Utána az X.Y koordinátáját a megjelenítés kezdetének.

Természetesen itt ettől relatívvá válik a struktúrán belő megadott x.y érték. Az V36-os Intuitionban található egy **EraselmageO** függvény is. amellyel a kirajzolt Image letörölhető.

1.6.3. A Gadget-ek

Azt hiszem ennyi ismeret után már tudjuk mi is az a gadget. de azért pontosítsuk. A/ok a grafikai object ek. amelyekre az egérrel rákattintva különböző, a program által meghatározott funkciók hajtódnak végre. Ebben a részben megtanulunk ilyen gadget-eket létrehozni. A Gadget-ek létrehozása szintén egyszerű dolog. Nagyban hasonlít az ablaknál és a screen-nél látot-

lakhoz, nevezetesen ill sem csinálunk mást. mint az inicializált struktúra mutatóját átpasszoljuk az Intuitionnak. ami ezek után elvégzi helyettünk a piszkos munkát. Alapjában véve csak két típusú gadgel létezik, nevezetesen a SYSTEM Gadget és a CUSTOM Gadgel. A SYSTEM Gadget-eket megbeszéltük az ablaknál. Amikkel itt foglalkozunk, azok a CUSTOM azaz felhasználói Gadget-ek lesznek. Ezeknek elég sok fajtájuk van. Az első a legegyszerűbb az ún. Boolean Gadget-ek melyek egy nyomógombszerűen csak bekapcsolt ill. kikapcsolt állapotot vehetik fel. A második a String Gadgef-ek, amelyekben az user szöveget pötyöghet be a programunknak. A harmadik típusba az Integer Gadget-ek tartoznak, amelyek hasonlítanak ugyan a String Gadgetre azzal a különbséggel, hogy csak integer adatok, azaz egész számok bevitelére alkalmas. A negyedik csoportba a Proportional Gadget-ek tartoznak. Ezek a Gadget-ek egy kis kapaszkodó segítségével (ahol megtudjuk ragadni őket) egy érték gyors beállítására szolgál. Ilyen Gadget-ek találhatók az ablakok szélein a szövegszerkesztőkben abból a célból, hogy az állományban gyorsan tudjunk lépkedni, valamint a színek beállításánál ilyeneket húzgálunk.

Ezenkívül léteznek még másfajták is. de ezeket pár oldallal arrébb tárgyaljuk, mert csak 2.0-ás rendszertől lehet használni őket (LislView gadget. Mpx gadget. stb.). Természetesen minden gadget megjelenését mi magunk' definiáljuk. így gadget-jeink olyanok lesznek amilyenre tervezzük őket. Lehet használnunk egyszerííbbeket, amikor csak kerete (Bordér) van. vagy használhatunk képeket (Image) is. De lehetőségünk van akár „láthatatlan" gadget használatára is. Erre, a legjobb példa a SYSTEM Gadget-eknél bemutatott Drag Gadget. A Gadget-ek tulajdonképpen bárhol elhelyezhetőek az őket megjelenítő eszközön (ablakok. Requester-ek. stb.). A pozíciójuk általában relatív az eszköz bal felső sarkához. Ezek után vizsgáljuk meg, hogyan is lehet megadni, ill. inicializálni egy gadget struktúrát.

Maga a struktúra így néz ki:

```
struct Gadget
{
    struct Gadget *NenGadget;
    SHORT LeftEdge, TopEdge, UJidth, Height;
    USHORT Flags;
    USHORT Rctiuation;
    USHORT GadgetType;
    RPTR GadgetRender;
    RPTR SelectRender;
    struct InutiTeKt *GadgetTent;
    LONG MutualExclude;
    RPTR SpecialInfo;
    USHORT GadgetID;
    RPTR UserData;
}
```

-ft ncxLuaagci: "&y mulató ti ROV^LKL^W £,*vaj^c . ^ . i t^ to,,o^ & - ^ ^L ^ i.~ , nálunk az ablakon, azokat össze kell fűznünk. A láncban nem érdekes semmiféle sorrend (koordináták szerint stb.). Ha az ablakunk megnyitása után

szeretnénk egy gadgel-eL megjeleníteni, azt ne a láncbafűzésével tegyük, mert így még nem lesz látható! Erre van egy speciális függvénye az Intuition-nak, azt használjuk.

A **LeftEdge. TopEdge** a gadget kooreiinátiáit definiálja.

A **Width, Height** a gadget szélességét és magaságát definiálja, úgy mint ahogyan azt már az ablaknál és a screen-nél megnéztük.

A **Flags,** a gadgetnek magadható flag-okat várja, amik az alábbiak: terijnészetesen ahol több megadható, ott a C-ben szokásos jelöléseket használhatjuk. (+.1). Az első csoportban azokat a flag-okat találjuk, amik a gadget megnyomott azaz választott módjában látszanak, mégpedig:

GFLG_GADGHCOMP: az összes gadget-ben található pixelnek a színét a Complemensére változtatja amikor a gadget aktívvá válik.

GFLG_GADGHBOX: körberajzolja a gadget-ünket. amikor kiválasztják a gadget-et.

GFLG_GADGHIMAGE: megjeleníti az általunk definiált Image-t vagy Borjier-t.

GFLG_GADGHNONE: nem csinál semmit ha aktivizáljuk a gadget-et. (no highlaited)

Ha a gadget-ünknek csináltunk Image-t, akkor állítsuk be a GADGHIMAGE flag-et, egyébként általában törölni szoktuk ezeket a flag-okat.

Ha azt akarjuk, hogy a gadget-ünk pozícója és/vagy nagysága kövesse az ablakunk ezen adatait, akkor használjuk a most következő flag-okat:

GFLG_RELBOTTOM: a Display TopEdge-jét fogja használni mint relatív koordináták alapja.

GFLG_RELRIGHT: a Display LeflEdge-je lesz a gadget-ünk y pozíciójának relatív viszonyítási alapja.

GFLG_RELWIDTH: a Display Width-je szolgál alpjául.

GFLG_RELRIGHT: a Display Hight-je lesz az alapja. Ezeket a flag-okat használhatjuk természetesen egyszerre is.

GFLG_SELECTED: ha a gadget-ünk toggle-select gadget (lásd. Acüvation flag-oknál). akkor ezt megadva, a gadget benyomott, azaz választott állapotban jelenik meg kirakásakor. Ha ennek a gadgetnek az aktivizálása során (IDCMP_GADGETDOWN) kíváncsiak vagyunk arra, hogy most benyomott-e avagy nem. akkor ezt a flag-ol kell csak lekérdeznünk.

GFLG_DISSABLED: ha azt akarjuk, hogy a kirakástól ez a gadget ne legyen aktív, akkor ezt a flag-ot állítsuk be. Van egypár flag ami a GadgetText mutatóját definiálja, hogy milyen típusú objectumra mutat.

GFLG_LABELITEXT: a GadgetText egy IntuiTextre mutat.

GFLG_LABELSTRING: a GadgetText egy (Char*)-re mutat.

GFLG_LABELIMAGE: a GadgetText egy Image struktúrára mutat.

Az alábbi flagak V37-LŐl részei a rendszernek:

GFLGJTABCYCLE: segítségével a gadgel részt vesz az aklivizációs ciklusban, amelyet aTab vagy a Shift-Tab.-bal érhetünk el. (V37!)
GFLG_STRINGEXTEND: ezt a Hagot sohase használjuk V34 alatt! A string gadget (lásd. Activation) STRINGEXTEND lesz a segítségével.
GFLG_IMAGEDISSABLE: ezt a flag-ot az Intuition állítja be nekünk automatikusan, így ezt mi ne tegyük! (V39!)
GFLG_EXTENDED: ez a flag jelöli ki azt. hogy ez a struktúra egy ExtGadget. Ha ezt nem állítjuk be. sohasem tudjuk majd olvasni az extre flag-okat. Az összes V39-es Boopsi Gadget-ek. ExtGadget-ek. (V39!)

Az Activationnal azt tudjuk beállítani, hogy mi történjék akkor amikor a gadget-ünket aktivizálják.

GACT_RELVERIFY: segítségével az Intuition üzen nekünk ha a gadgetet aktivizálták és a pointer a gadgel-et kiválasztó négyzeten belül található.
GACTJMMEDIATE: ha ez a flag be van állítva, akkor üzenetet küld az Intuition nekünk, ha a gadget-et megnyomták amikor aktív volt.
GACT_FOLLOWMOUSE: ha a gatget aktivizált állapotában üzenetet kívánunk kapni arról, hogy az egér pointer merre jár.

Ha a GACT_RELVERIFY Hag-ot nem állítjuk be ezeknél az utóbbi flag-oknál. csak egyszer kapunk vissza üzenetet. Ha beállítjuk, az üzenet követi az user minden megmozdulását.

GACTJTOGGLESELECT: ez azt állítja be, hogy a gadget olyan legyen mint egy kapcsoló, tehát hogy a megnyomásakor mindaddig megnyomva maradjon, ameddig az user újra meg nem nyomja. A megnyomott állapotát lekérdezhetjük a GFLG_SELECTED segítségével.
GACT_BOOLEXTEND: a gadget-hez Boollnfo csatlakozik, (lásd később.)

Ha a gadget ablakhoz csatolódik. lehetőségünk van arra. hogy a gadgetünk automatikusan kövesse az ablak bordér nagyságának változásait.

GACT_RIGHTBORDER: a gadget szélességét az ablak jobb oldali bordere alapján számolja ki.
GACTJJSFTBORDER: a gadget szélességét az ablak bal bordere alapján számolja.
GACTJTOPBORDER: a gadget y pozícióját az ablak felső bordere alapján számolja.
GACTJBOTTOMBORDER: a gadget y pozícióját az ablak alsó bordere alapján számolja.

A következő flag-okkal azt állíthatjuk be. hogy a String Gadget-be beírni ítívánt szövegünk hová legyen igazítva. Ha nem adjuk meg. akkor a bal szélére kerül. Valamint a String Gadget egyéb flag-jai következnek:

GACT_STRINGLEFT: ez a default beállítás, és a bal szélére igazítást jelenti.
GACT_STRINGRIGHT: ez a jobbra igazítást takarja.
GACT_STRINGCENTER: középre igazításnál használhatjuk.
GACT_LONGINT: String Gadget long int-re. így az user csak 32-bites előjeles egész számot vihet be.
GACT_ALTKEYMAP: alternatív keymap (bili.-kiosztás) használata.
GACT_STRINGEXTEND: a String Gadget-ünk StringExtend. Soha ne íiasználjuk V36-os alatt!

Egyéb flag-ek:

GACT_ACTIVATION: a gadget aktív lesz mikor először kijelzésre kerül.
GACT_BORDERSNIFFT: az összes Activation flag-ot bekapcsolja.
GACT_ENDGADGET: ha a gadget-ünk requesterhez csatlakozik, és ezt beállítottuk, akkor a gadget aktivizálásakor a requester automatikusan be-(psukódik (lásd. Requester-ek (1.8) résznél a könyvben.).

A Gadgeffype segítségével a gadget-ünk típusát adhatjuk meg, az alábbiak szerint:

GTYP_BOOLGADGET: a gadget-ünk boolean típusú lesz.
GTYP_STRGADGET: a gadget-ünk string gadget lesz.
GTYP_PROPGADGET: a gadget-ünk proporcionális gadget lesz.
GTYP_CUSTOMGADGET: a gadget a felhasználó által definiált gadget lesz.
GTYP_GADGET0002: a gadget OOO2-es típusú lesz.
GTYP_SYSGADGET: a gadget System Gadget lesz. A System Gadget-ek ltözül is hogy melyik, azt további flag-ok állítják be. Ezeket a flagokat itt nem Hészletezzük.
GTYP_GZZGADGET: ez a flag azt jelzi, hogy a gadget-ünk címe zero zero ablakhoz csatolódik. Beállításával a gadget-ünket a külső részébe teszi az ablaknak. így az nem fogja elrontani a belső ablaktérben lévő rajzunkat.
GTYP_REQGADGET: a gadget-ünk requesterhez csatolódik.
GTYP_SCRGADGET: a gadget-ünk nem ablakhoz, hanem screen-hez csatlakozik.

Az újabb rendszerben az ExtGadget esetén további flag-ok találhatóak, amelyeket az GFLG_EXTENDED flag beállítása után használhatunk, és aza,lábbiakat takarják (V39!):

GMORE_BOUNDS: az ExtGadget érvényességét állítja.
GMORE_GADGETHELP: ez a gadget válaszol a Gadget Helpre.
GMORE_SCROLLRASTER: ez a (Custom) gadget használja a ScrollRaster függvényt.

A GadgetRender egy mutatót tartalmaz arra az Image vagy Bordér struktúrára, amelyet a gadget-únk használ, ha beállítottuk a neki megfelelő flaget. Ha nem akarunk semmilyen grafikát kapcsolni a garíget-ünkhöz. akkor ezt állítsuk NULL-ra.

A **SelectRender** egy mutatót vár arra az objectumra. amely a gadget aktív állapotában jelenik meg. Ez lehet egy Image vagy Bordér struktúrát megcímző mutató. Természetesen a típusának meg kell egyeznie a Gadget-Rendernél beállítottakkal. Ahhoz, hogy használni tudjuk, be kell állítanunk a megfelelő flag-okat (pl.: GFLG_GADGH1MAGE stb.). Ha nem akarunk semmilyet használni, akkor ide is NULL-t kell írnunk, valamit a flag-okat ennek megfelelően beállítanunk (GFLG_GADGHNONE).

A **GadgetText** egy pointert vár a gadget-hez tartozó szövegre, amelynek IntuiText típusúnak kell lennie. Ha nem tartozik szöveg a gadget-hez. akkor ide NULL-t kell írnunk.

A **MutualExclude** azt a számot adhatjuk meg. hogy az első 32 gadgel közül mellyek azok a Gadget-ek, amelyek ezzel a gadget-tel együtt, mintegy egymást kiváltó kapcsolók működjenek. Tehál mikor ezt a gadget-et aktivizálják, a hozzá tartozó (MutualExcludedes) Gadget-ek automatikusan inaktív állapotba kerülnek. Ha például azt akarjuk, hogy a O-ás. 2. 5. és a 8-as Gadget-ek automatikusan inaktívvá váljanak, ide a 293 kell írnunk. (293=%100100101) Természetesen a dolog csak Toggle-select Gadget-ek esetében fog működni.

A **Speciallnfo** ide olyan struktúra mutatóját várja, amely a gadget típusától függően változhat. Ha a gadget proporcionális, akkor ide várja a Proplnfo struktúrát megcímző mutatónkat. Ha azonban String (Integer) Gadget, akkor ide a Stringlnfo struktúrát vár. Bool gadget esetében itt adhatjuk meg a Boollnfo struktúránkat, ha szükséges. Egyébként ide NULL-t írhatunk (ezekről részletesebben pár oldallal arrébb olvashatunk.).

A **GadgetID** olyan változó, amit az Intuition nem használ. Tehát saját magunk használhatjuk, például a gadget azonosítására. Ezt a lehetőséget pascalban (KickPascal) előszeretettel használják.

Az **UserData** szintén egy olyan pointer lehetősége, amit mi használhatunk, mert az Intuition ezt sem használja.

Ezek után nézzük meg hogyan is hozhatunk létre Gadget-eket saját magunk.

Először a legegyszerűbbet nézzük meg.

1.6.3.1. A Boolean Gadget-ek

Ahhoz, hogy ilyen gadget-et használhassunk, először is definiálnunk kell egy gadgel struktúrát majd inicializálnunk azt. Ezt így tegyük meg:

```
struct my_gadget =
{
    NULL,       /* Nincs másik gadget-ünk egyelőre */
    40,         /* H LeftEdge 40 pixelre */
    20,         /• fl TopEdge 20 sorral lejebb */
    60,         /* fl szélessége 60 pixel */
    20,         /* fl magassága 20 sor */
    GFLG_GfIDGHCOMP,/* H gadget flag-ját beállítottuk úgy, hogy */
                /* aktiuizálása esetén a gadget-ünk inuerz szi-
                /* nekben pompázzék. */
                /* fl színek az alábbiak lesznek: */
                /* R 0-ásból a 3-as lesz. (00 a negáltja a 11)
                /* flz l-esből a 2-es lesz. (01 negáltja az 10)
                /* fl 2-esböl az l-es lesz. */
                /* fl 3-asból a 0-áslesz. •/
    GHCT_IMMEDIRTE|RELUERIFY,
                /* flz actiuation flagot beállítjuk arra, hogy */
                /* kapjunk üzenetet az Intuitiontól, ha a
                   gadget-ünket kiuálasztják. */
    GTVP_BOOLGfIDGET, 7* H gadget Boolean típusú */
    Crny_border, /• mutató a bordér struktúránkra, részletesen
                 /* az alacsony színtű grafikai résznél. */
    NULL,       /* SelectRendert nem használjuk. */
    &my_text,  /• fl gadgetbe kerülő szöuegünk •/
    NULL,       /• R MutualExcludeot sem használjuk */
    NULL,       /* fl SpecialInfot sem használjuk */
    0,          /* Nem használjuk */
    NULL        /* Ezt sem */
};
```

Az első példánkban ezt működni is látjuk. Természetesen lehetőségünk /an arra is. hogy a bool gadget-ünknek ne az egész felületén lehessen aktivizálni így elérhetjük azt is. hogy a gadget-ünk ne csak négyszögletes legyen, hanem bármilyen sík alakzatot felvehessen, vagy csak akkor működjön ha bizonyos pontját aktivizálta az user. Ahhoz, hogy ezt alkalmazni tudjuk, meg kell ismerkednünk az ún. BoolInfo struktúrával. Maga a struktúra igen egyszerűen néz ki:

```
struct BoolInfo
{
    USHORT Ftags;
    UUI0RD  *Mask;
    ULONG  Reuersed;
}
```

A struktúra mint látjuk nem igazán nevezhető agyonkomplikáltnak. De nézzük mi mit is jeleni:

A **Flags-el** azt állíthatjuk be. hogy mi mire is szolgáljon. Egyenlőre csak egy flag. a **BOOLMASK** létezik hozzá, amellyel a Mask megadására utalunk.

A **Mask** már érdekesebb. Itt kell megadnunk azt a maszkot, amellyel kijelöljük az aktív területet. Ez egy pointer a maszkot tartalmazó UWORD típusú tömbre, ahol az aktív pontokat megadjuk.

A **Keserved,** mint a neve is mutatja egyenlőre fentartott. tudtommal még semmi nem használja.

A maszknál megbeszéltekre nézzünk egy példát, hogy jobban megérthessük megadásának menetét. A maszk egy bináris maszk, amely azon a helyen, ahol aktívvá válhat a gadget-ünk. egy l-est tartalmaz, egyébként O-át. A maszk mérete teljesen lefedheti a gadget-ünk méretét. Például nézzünk meg egy 16 pixel széles, és 8 pixel magas gadget maszkját, hogyan adjuk meg.

Maszk	16bites mamória szavak	Hexában
OOOOOO1111OOOOOO	0000 001 1 1100 0000	03C0
0000111111110000	0000 1111 1111 0000	OFFO
0011111111111100	0011 1111 1111 1100	3FFC
1111111111111111	1111 1111 1111 1111	FFFF
1111111111111111	1111 1111 1111 1111	FFFF
0011111111111100	0011 1111 1111 1100	3FFC
0000111111110000	0000 1111 1111 0000	OFFO
0000001111000000	0000 0011 1100 0000	03C0

Ez a gép nyelvén így adható meg:

```
WORD my_mask[]=
{
    0K03C0,
    OKOFFO,
    0K3FFC,
    OxFFFF,
    OKFFFF,
    0K3FFC,
    OKOFFO,
    8xQ3C0
};
```

Tehát ennek a tömbnek a címét kell átadnunk a gadget struktúrának, hogy csak ezen a részén lehessen aktivizálni a gadget-ünket. A lemezen természetesen találunk erre is példát.

1.6.3.2 A String és Integer Gadget

Mint már tudjuk ezen Gadgel-ek segítségével a programunkba közvetlenül vihetünk be adatokat, amelyek a string esetében karakterek lehetnek. míg az integer esetében csak egész számok. Használatukkal igen egyszerűvé válik ezen adatok bevitele. A bevitel során természetesen lehetőségünk van kisebbfajta szerkesztési feladatok ellátására, amit a rendszer automatikusan elvégez helyettünk (pl. a kurzormozgató billentyűk használata, Del gomb jhasználata, Back space használata stb.). A bevitel során korlátozhatjuk a bejvinni kívánt karakterek hosszát is. Ahhoz, hogy ilyen gadget-et használhassunk, nem elég csak a gadget struktúrát inicializálnunk, hanem egy StringInfo struktúrát is létre kell hoznunk, és címét megadnunk a gadget struktúrában. Ez a StringInfo az alábiak szerint néz ki:

```
struct StringInfo
{
    UBVTE *Buffer;
    UBVTE *UndoBuffer;
    SHORT BufferPosition;
    SHORT MaxChars;
    SHORT DíspPos;
    SHORTUndoPos;
    SHORT NumChars;
    SHORT DíspCount;
    SHORT CLeft, CTop;  •
    struct Layer *LayerPtr;
    LONG LongInt;
    struct KeyMap *RItKeyMap;
};
```

A mezők jelentése pedig álljon itt:

A **Buffer** egy nullára végződő string pointere. Itt kapjuk majd meg az user által bepötyögött karaktereket. Ha a gadget kirakásakor már itt található néhány karakter, azt az Intuition megjeleníti a gadget-tel együtt.

Az **UndoBuffer** színlén egy nullára végződő string pointerét kell hogy takarja, amelybe az Intuition bemásolja a Buffer tartalmát, ha az user kitörölte azt. és újra visszarakja a gadget-be. ha az user az Amiga+Q billentyűt használja.

A **MaxChars** -nál adhatjuk meg. hogy mennyi karaktert engedjen a gadgetbe írni.

A **BufferPos-ban** megadhatjuk, hogy a kurzort hová tegye az Intuition, mikor a gadget-ünket aktivizálják az user-ek.

A **DispPos** arra szolgál, hogy megadhassuk azt a pozíciót, amelytől a string-ünket kijelzi.

Az itt következőket az Intuiüon inicializálja, és használja.

Az **UndoPos** hasonló a fentiekhez, csak az Undo stringre vonatkozik.

A **NumChars** az aktuális karakter számát takarja a pufferben.

A **DispCount** a látható karakterek száma a keretben.

A **CLeft, CTop** a bal és a felső ofszett a keretben.

A **LayerPtr** pointer egy Layer struktúrára (egy későbbi könyvben tárgyaljuk a layer-eket).

A **LongInt** az user által beírt szám értéke, ha a gadget integer.

Az **AltKeyMap-ban** a használni kívánt alternatív billentyűzetkiosztást takaró KeyMap struktúra címét adhatjuk meg. Ennek segítségével lehetőségünk van korlátozni a bevihető adatokat. Ezt csak a GFLG_ALTKEYMAP beállítása esetén veszi figyelembe.

Ha String vagy Integer gadget-et használunk, ne feledjük el azt a programozói szólást, miszerint ha az user-re bízzuk az adat bevitelét, akkor az biztosan hibázni fog a bevitel során. Bár elég javítási lehetősége van bevitel közben, ez nem akadályozza meg abban, hogy hülyeségeket írjon be. Ennek a lekezelését azonban már nekünk kell megoldanunk. Bár jól leszűkíti a kört. ha a billentyüzetkiosztást a célnak megfelelően alakítjuk ki. De nézzünk konkrét pédát a string gadget definiálására és inícializálására:

```
UBVTE my_bufferl50];
UBVTE my_undo_buffer[50l;

struct Stringlnfo my_string_info=
{
    my_buffer; /* 0-ára kell hogy uégződjön. */
    my_undo_buffer;
    0,  /• H puffer pozíciónk 8 */
    50, /* fl maximálisan beuihetö hasznos karakterek száma 49 */
    0,  /* fl megjelenítést a O-ástól kezdi. */
        /* fl többit az Intuítion tölti ki, így ezek NULL-ok */
    0,  /* UndoPos */
    0,  /* NumChars •/
    0,  /• DispCount */
    0,0,   /* CLeft, CTop */
    NULL,  /* LayerPtr */
    NULL,  /* LongInt •/
    NULL   /* HltKeyMap */
};
```

Most jöhet maga a gadget megadása:

```
struct Gadget my_gadget=
{
     NULL,  /* Nincs másik gadget-ünk */
     68,    /* 68 pixelre uan kitéue */
     30,    /* 30 sorral lejjebb •/
     198,   /* fl szélessége 198 pixel */
     8,     /* fl magassága 8 sor. Itt nem árt figyelembe uennünk
               a használt Font méretét, mert egyébként csúnyán
               kilóghat a keretből. */
    UJFLG_GRDHCOMP, /* fl coplementjére uáltozik a színe, ha aktíu.
        /* fl kurzor színe is complementjére uáltozik. •/
        /* Leggünk arra is tekintettel, hogg ha ide
        GFLG_GfIDGHNONE kerül, akkor akiuizállás esetén az user
        nem fogja látni a kurzort. */
    GRCT_IMMEDIRTE|GRCT_RELUERIFY,
        /* fl programunk üzenetet fog kapni, ha aktiuizálták
        a gadget-ünket •/
    GTVP_STRGRDGET, /* fi gadget-ünk String típusú */
    &my_border,     /• H gadget-ünket körülueuó bordert
        definiáló tömb címe. */
    NULL,     /* Nem használjuk */
    &my_text, /* Mire is használjuk a gadget-ünket, itt nem árt */
        /* tájékoztam' az usert, mit is uárunk el tőle */
    NULL,     /* Nem használjuk */
    G-my_string_info, /* Ide kell tehát beírnunk a StringInfonk
        címét */
    O.NULL    /* Ezeket sem használjuk. */
};
```

Az Integer gadget annyiban tér el ettől, hogy az Activation mezőbe a OACT_LONGINT kerül. A bufferbe pedig az integer stringet másoljuk (strcpy(my_buffer, "0");).

Mint már fentebb említettem, az Intuition megengedi, hogy a bevinni kívánt karakterekel egy minimális segítséggel szerkeszthessük. Ezek a következők: a karakterek között mászkálhatunk a jobb. ill. a bal kurzort mozgató billentyűkkel, valamint ezeknek a Shift-tel való egyidejű lenyomásakor, a szerkesztett sor elejére ill. végére ugorhatunk. A Backspace-szel törölhetű«R^ karaktereket. A Del-lel pedig a kurzor alattiakat törölhetjük. Az Amiga+Q-ra visszaírja az ezt megelőző állapotot. Az Amiga+X-re kitörli a puffért. A return hatására elfogadottnak veszi a karaktereket és átadja a programunknak, tájékoztatva azt arról, hogy az user abbahagyta a bevitelt.

1.6.3.3. A proporcionális gadget

A proporcionális gadget-nél is, mint azt már megszokhattuk az Amiga esetében, itt is egy struktúrát kell inicializálni. Ez a struktúra mondja meg, hogy a proporcionális gadget-ünket vízszintesen, vagy függőlegesen kívánjuk-e húzgálni, vagy esetleg mindkét irányba. A proporcionális gadget egyébként nagyon hasonlít a hangerőszabályzó potenciométerre. Nem a Sokol rádió esetén. Tehát a tolópotméterre hasonlít. Azonban ennek a húzgálójának a méretét is állíthatjuk attól függően, hogy hány lépésben kívánjuk az értékek állítását. Például az Amiga hangerejét csak 64 lépésben lehet szabályozni, így kár lenne több lépést megengedni az usernek a rángatás közben mint 64, mert nekünk kellene lekezelnünk a magasabb értékeket. De inkább nézzük mit is tartogat számunkra a Proplnfo struktúra.

```
struct PropInfo
 {
     USHORT Flags;
     USHORT HorizPot;
     USHORT UertPot;
     USHORT HorizBody;
     USHORT UertBody;
     USHORT CUJidth;
     USHORT CHeight;
     USHORT HPotRes, UPotRes;
     USHORT LeftBorder;
     USHORT TopBorder;
 };
```

A mezők jelentése az alábbiakban alakul:

A Flags-oknál a következők adhatóak meg. Természetesen több is megadható ha szükséges.

FREEHORIZ: ha ez beállított, az user vízszintesen tudja mozgatni.
FREEVERT: ha ez beállított, az user függőlegesen tudja mozgatni.
PROPBORDERLESS: a proporcionálishoz nem számolódik keret.
KNOBHIT: beállítódik, ha a Knob (húzgálója) érintett.
PROPNEWLOOK: a proporcionálisunk új kinézetű lesz. (V36-tól!)
AUTOKNOB: ilyenkor az Intuition maga számolja ki a Knob kinézetét automatikusan.

A **HorizPot** tartalmazza az aktuális proporcionális vízszintes értékét. Ha az user 25%-ban jobbra húzta a Knob-ot, akkor az értéket a következőképpen kapjuk: az érték a MAXPOT 25%-ka lesz. Tehát a HorizPot 25%, és így ha a MAXPOT ot megszorozzuk 0.25-tel. akkor kapjuk az aktuális értéket, azaz 0x3FFF-et. A MAXPOT értéke OxFFFF.

A **VertPot** uayanaz mint a HorizPot, csak függőlegesen.

A **HorizBody** a Knob testeneK meretet amiiicujuK DC, így <*J. m i * COOH* annyit mozdulhat el, amennyit mi megadunk ennél a változónál. A változó megadására egy formulát mutatunk be. Ha például egy fájlkiválasztó abla-

kot csinálunk és az éppen látható fájlok száma 8 lehet, és 32 fájl van összesen, akkor a számítás így néz ki: HorizBody=MAXBODY * 8/32, azaz i0x3FFF. Ha a hangerőt szeretnénk a segítségével állítani, ami ugye 64 lépésiben lehetséges, akkor így néz ki: HorizBody=MAXBODY * 1/64 az az 0x03FF.

A **VertBody** ugyanaz mint a HorizBody, csak függőlegesen. A többi változót az Intuition használja és inicializálja.

A **CWidth** a keret szélessége.

A **CHeight** a keret magassága.

A **HPotRes, VPotRes** a pot felbontása.

A **LeftBorder, TopBorder** a keret pozíciója a border-től.

Ennyi tudás után nézzük meg, hogyan is inicializáljunk egy proporcionális gadget-et. A példában egy hangerőszabályzó prop. szerepel, amely 64 lépésben szabályoz. Először tehát megadjuk az Image struktúrát, amely definiálja a prop.-unk Knob-jának kinézetét (az Image-ról részletesen az alacsony szintű grafikánál). Ennek az image-nak a kinézetét az AUTOKNOB ílag beállítása esetén maga az Intuition számolja ki, nekünk csak egy ilyen típusú változó címét kell átadnunk a részére.

```
struct Image my_Image;

struct PropInfo my_prop_info =
{
    FREEHORIZIRUTOKNOB, /* R knob-ot uízszintesen akarjuk
                           mozgatni, és az Intuition számolja az
                           image-t a knobhoz •/
    0,              /* fl HorizPot kezdeti értéke. */
    0,              /* UertPot, 0 miuel nem lehet függőlegesen */
                    /* mozgatni. */
    MRKBODV * 1/64,    /* HorizBody, 64 lépésben •/
    0,              /* UertBody */
    /* R többit az Intuitionra bízzuk */
    0,              /* CUJidth */
    0,              /* CHeigth */
    0,0,            /* HPotRes, UPotRes */
    0,              /* LeftBorder •/
    0               /* TopBorder */
};

struct Gadget my_gadget =
{
    NULL,    /* Nincs másik gadget-ünk egyenlőre •/
    80,     /* 80 pixelre K irányban */
    30,     /* 30 sorral lejjebb */
    200,    /* UJidth, 200 pinel széles •/
    12,     /* Height, 12 sor magasan*/
    GFLG_GRDGHNONE, /* Flags, Ne csináljon semmit */
    GHCT_IMMEDIRTE|GfiCT_RELUERIFV,
            /* Rctiuation, ha aktiuizálja az user, szóljon erről
               nekünk is. */
```

```
    GTVP_PROPGHDGET,    /* GadgetType, proporcionális gadget */
    &my_image,      /* fl knob Image */
    NULL,           /* SelectRender, Ilyet nem használunk */
    {ymy_tent,      /* GadgetTent, a gadget-ünk szöuegének címe
                       (IntuiTeKt)*/
    NULL,           /* MutualExclude, ezt sem használjuk •/
    &my_prop_info, /* SpecialInfo, a PropInfo címe */
   0, NULL          /• GadgetID, UserData, ezeket sem használjuk */
};
```

Mint látjuk, ezek sem okozhatnak különösebb problémát. A lemezen található ehhez a részhez is példaprogram.

Feltűnhetett az, hogy ha 2.0-ás, vagy feletti gépünk van, akkor ezek a Gadget-ek nem egészen olyanok, mint amit a 2.0-ás rendszertől az Amigánk használ. Ez azért van, mert a 2.0-ás rendszernek már része a GadToolsLibrary. Ez a Library gondoskodik nekünk az új formájú Gadget-ekről. A következő részben ezt nézzük meg. Egyébként a fenti Gadget-ek létrehozása és belövése meglehetősen nehézkes volt segédprogram nélkül.

Szerencsére ez már másnak is szemet szúrt, és írtak ennek segítésére néhány segédprogramot. A programok jellemzően nemcsak a Gadget-ek elhelyezésében segítenek, hanem az ablak(ok), vagy a screen(ek), ill. a menük megszerkesztésében is nagy segítségünkre lehetnek. Leginkább azért, mert a legtöbb segédprogram komplett forráslistát ment a számunkra, amit nekünk csak fordítani, ill. futtatni kell. Természetesen a Gadget-ekre, ill. menükre válaszoló függvényeket nekünk kell megírnunk. Ezekről a programokról részletes leírást fogunk találni a könyvünk 3. fejezetében.

1.6.3.4. Gadget-ek a GadTools Library segítségével

A GadTools Library-val elérhetővé válnak számunkra a 8. ábrán látható Gadget-ek. Ezeket a Gadget-eket tulajdonképpen a mostani tudásunkkal mi is meg tudnánk csinálni, persze jóval bonyolultabban, mint ahogy ezt a GadTools használatával tehetjük. A képen jól látható, hogy minden gadget neve más, mint amit az eddigiekben láttunk. Ezeket a neveket nem árt megjegyezni, mert a segédprogramok is ezen a néven használják őket (pl. GadtoolsBox). Részletesen nem írjuk le a GadTools Library-val működő Gadget-ek létrehozását, mert a segédprogramok által mentett források jól érthetőek a fenti tudás birtokában. Annyit azonban meg kell jegyeznünk, hogy a Gadtools segítségével létrehozott Gadget-ek és menük IDCMP lekérdezésére is a GadTools függvényeit kell használnunk. A GadTools egyébként nem rendelkezik olyan sok függvénnyel mint az Intuition, de ez nem csoda, hiszen nem helyettesíti, inkább csak kiegészíti azt. Maga a library leírását megtaláljuk a könyv 2. fejezetében. Annyit azonban elárulhatunk, hogy a Gadget-ek létrehozásakor a NewGadget struktúrát is használja. A NewGadget struktúra az alaDDiaK szerint nez KI.

```
struct NewGadget
{
    UWORD ng_LeftEdge, ng_TopEdge;
    UWORD ng_Width, ng_Height;
    UBVTE *ng_GadgetText;
    struct TextRttr *ng_TeKtfIttr;
    ULDORO ng_GadgetID;
    ULONG ng_Flags;
    RPTR ng_UisulaInfo;
    RPTR ng_UserData;
};
```

Ahol az **ng_LeftEdge, ng_TopEdge** a gadget pozícióját adják meg, míg az **ng_Width,** és az **ng_Height** a gadget dimenzióját. Az **ng_GadgetText** egy mutató a gadget szövegére, amelyet állithatóan a gadget felé, alá. ill. jobb vagy bal oldalára helyezi automatikusan. A default elhelyezés a bal oldalsó. Az **ng_TextAttr** a gadget szövegének. TextAttribútumának címét várja. Minden-féleképpen adjuk meg. mert ha nem, vagy NULL-t adunk meg. a rendszer összeomlik, persze csak akkor, ha gadget szöveget is megadtunk. Az **ng_GadgetID** a gadget azonosítója. Az **ng_Flags-nél** azok a beállítások következnek, amik a gadget szöveg megjelenítésének helyére vonatkoznak. Ezek a következők:

PLACETEXT_LEFT: bal oldalra helyezi a szöveget.
PLACETEXT_RIGHT: jobb oldalra helyezi a szöveget.
PLACETEXTJVBOVE: középre és fölé helyezi.
PLACETEXTJBELOW: középre és alá helyezi.
PLACETEXTJN: középre és bele helyezi.
NG_HIGHLABEL: ha a gadget-et aktivizálják, a szöveg megvillan.

Mint látható, igen egyszerű a struktúra. A gadget létrehozását a **CreateGadgetAO** függvény végezheti el nekünk. A függvény első paraméterként a létrehozandó gadget típusát meghatározó számokat várja, amelyek az alábbiak lehetnek (a teljesség igénye nélkül):

BUTTON_KIND: a gadget button gadget legyen (új típusú, de tulajdonképpen a régi Bool Gadget).
CHECKBOX_KIND: a gadget check box gadget legyen (kipipálható).
INTEGER_KIND: integer gadget (egy (long) integer bevitelét lehetővé tevő).
LISTVIEW_KIND: a gadget listview gadget legyen (listánézegető).
MX_KIND: a gadget MX gadget (az egymást kiváltó kapcsolók).
NUMBER_KIND: egy szám megtekintését lehetővé tevő kis ablak (nem olyan mint amit az openwindow nyit!).
CYCLE_KIND: olyan gadget. amelyben a szöveget megelőzi egy kis kör alakú nyíl. mutatván, hogy ha rákattintunk, akkor a gadget más értéket vesz fel.
PALETTE_KIND: a screen színeit megjelenítő gadget. amellyel a színek kiválasztása érhető el.

SCROLLER_KIND: amelyet az ablakok szélén szoktak használni a bennük lévő információ scrollozására.
SLIDER_KIND: tulajdonképpen a régi proporcionális gadget megfelelője.
STRINGJÍIND: egy string bevitelét lehetővé tevő gadget.Tulajdonképpen csak külsőleg változott meg. a régi String Gadget helyett használható.
TEXTJKIND: egy tetszőleges szöveg megtekintését lehetővé tevő ablakocska (lásd NUMBER_KIND).

Ezek után egy gadget struktúra mutatót vár, és csak később adhatjuk meg a mutatóját a NewGadget struktúránknak. Legvégül a gadget típusától függő különböző beállítások felsorolása történhet a Gadtools tagok segítségével.
Ezek az alábbiak lehetnek (szintén a teljesség igénye nélkül):

A check box gadget tagjai:

GTCB_Checked: a check gadget állapotát állítja beállítottra.
GTCB_Scaled: (boolean) a megjelenő Image skálázott legyen-e? (V39!)

A Listview gadget tagjai:

GTLV_Top: a listview gadget-ben a lista száma, ahonnan látszik a gadgetben.
GTLV_Labels: a listát tartalmazó változóra (átalában tömb) mutató pointer.
GTLV_ReadOnly: a lista csak olvasható lesz. és nem lehet a részeit kiválasztani.
GTLV_Selected: a listában a kiválasztott elem számát adhatjuk meg.
GTLVJScrollWidth: a listát mozgató scroll gadget szélessége definiálható a segítségével.
GTLV_ShowSelected: a listában a kiválasztott elemet képes egy string gadget-nek átadni szerkesztésre, és itt annak pointerét várja. Természetesen NULL, ha nem akarjuk ezt használni.
GTLVJMakeVisible: (boolean) csináljon-e az item-nek visiblét? (V39!)
GTLVJtemHight: Az itemek a magassága. (V39t)

Az MX gadget tagjai:

GTMXJLabels: az MX gadget gombjaihoz tartozó szövegekre mutató pointer (megjegyzések, amiből az user rájon, hogy melyiket mikor nyomja meg).
GTMX_Active: melyik gombja legyen aktív az MX gadgetnek.
GTMX_Spacing: az MX gadget-et képező tagok között mennyi hézagot tegyen?
GTMX_Scaled:fboolean) a megjelenő Image skálázott legyen-e? V39!)

GTMXJTitlePlace: a gadget fejléce hová kerüljön. (V39!)

A Text gadget tagai:

GTTX_Text: az ablakocskában megjelenítendő szövegre mutató pointer.
GTTX_CopyText: (boolean) a megjelenítendő szövegünket bemásolja-e az ablakocskába.
GTTX_Border: (boolean) az ablakocskának legyen-e kerete.
GTTX_FrontPen: a megjelenítendő szöveg színe. (V39!)
GTTX_BackPen: a megjelenítendő szöveg háterrének színe. (V39!)
GTTX_Jastiflcation: a szöveg megjelenítésének módja. Ezeket az értékeket adhatjuk meg: GTJ_LEFT. GTJ_RIGHT. GTJ_CENTER (V39!)
GTTX_CHpped: (boolean) csináljon-e Clip text-et. (V39!)

A Nuber gadget tagjai:

GTNM_Number: az ablakocskában megjelenítendő szám.
GTNM_Border: (boolean) legyen-e kerete az ablakocskának.
GTNM_„FrontPen: a megjelenítendő szám színe. (V39!)
GTNM_BackPen: a megjelenítendő szám hátterének színe. (V39!)
GTNM_Jastification: a szám megjelenítésének módja. Ezeket az értékeket adhatjuk meg: GTJ_LEFT. GTJ_RIGHT. GTJ_CENTER (V39!)
GTNM_Format: mutató a formátumot tartalmazó string-re. (V39!)
GTNM_MaxNumberLen: a szám max. hossza. (V39!)
GTNM_Clipped: (boolean) csináljon-e Clip textet? (V39!)

A Cycle gadget tagjai:

GTCY_Labels: a cyclegadget-ben a lapozódó szövegekre (megjegyzéskor) mutató pointer. Nullával kell végződnie a felsorolásnak!
GTCY_Active: az aktív megadása, ha az nem a O-ás label-ű.

A Palette gadget tagjai:

GTPA_Depth: a használni kívánt Bitplane-ek száma.
GTPA_Color: a használni kívánt színek száma.
GTPA_ColorOffset: a színek felsorolását honnan kezdje (a default az a O.-tól).
GTPAJndicatorWidth: a színeket bemutató négyzet, ill. téglalap szélessége.
GTPAJndicatorHeight: a színekel bemutató négyzet ill téglalap magassága.
GTPA_NumColors: a megjelenyleni kívánt színek meghatározására szolgál. (V39!)

A Scroller gadget tagjai:

GTSCJTop: a scroller gadget max. elmozdulási értékét állíthatjuk be.
GTSC_Total: a scroller gadget mozgástartománya.
GTSC_Visible: a scroller gadget Knob-jának pozíciója.
GTSC_Arrows: a nyilak nagyságát definiálja.

A Slider gadget tagjai:

GTSL_Min: a slider gadget minimális mozgástartománya.
GTSL_Max: a slider gadget maximális mozgása, ameddig elmozdítható legyen.
GTSL_Level: a slider szintje.
GTSL_MaxLevelLen: az érték megjelenítésének maximális hossza.
GTSLJLevelFormat: az érték megjelenítésének formátuma (pl.: (ULONG) "%21d")
GTSLJLevelPlace: az érték megjelenítése hová kerüljön, (pl.: 2 jobb oldalára a gadgetnek.)
GTSL_DispFunc: meghív egy függvényt, amely a kívánt formátumot kiszámolja.
GTSL_MaxPixelLen: az érék kiírásának maximális hossza pixelben. (V39!)
GTSL_Jastiflcation: hogyan legyen az esetleges tördelése az érték kiírásának? (magadhatóak: GTJ_LEFI\ GTJ_R1GHT, GTJ_CENTER. /V39!/)

Valamint itt még megadható, ha szükséges, a gádget-ünknek az activationnál definiált tag-ok is (pl.: GA_RelVerify stb. ezek részletes felsorolását lásd később).

A String gadget tagjai:

GTST_String: a megjelenítendő szöveg mutatója (amit a megnyitáskor belemásol a String gadgetbe).
GTSTJVlaxChars: a bevihető maximális karakterek száma.

Az Integer gadget tagjai:

GTIN_Number: a megjelenítendő szám mutatója (amit a megnyitáskor belernásol az integer gadgetbe).
GTIN_MaxChars: a bevihető maximális számjegy.

Egyéb:

GT_Underscore: aláhúzást adhatunk meg a segítségével, amely a gadgetbe kerülő beírás egyik karaktere elé megadva az '_' jelet, annak a karakternek az aláhúzását jelenti.

Jól látható, hogy használatával igen egyszerűen, és mégis mindent meg tudunk adni fiadgel-ünk létrehozásához. A Gadget-ek álltai beállított értékeket, mint például a ListView gadget-ben a kivaias/aou iicm Mámöt. vn^y a. slider gadgetben beállított értéket, vagy az MX gadget aktivizált gombját, az IntuiMessage struktúra Code változója tartalmazza. Ne felejtsük el. hogy az

, IntuiMessage üzenet beolvasására, és a válasz elküldésére ne a szokásos lntuiüon függvényekel használjuk. Erre a GadTools Librarynak meg vannak **a** saját függvényei! (lásd a 2. fejezetben a Gadtoolsnál.) A létrehozott Gadget-eket az ablak becsukásakor nem árt felszabadítani. így nem foglalják tovább la memóriát (lásd **FreeGadgets(struct Gadget*)** függvény). Ezek szemléltetésére láthatunk a lemezen némi példát (lásd Gadget-ek_2.0/GadToolsDemo). Mégegyszer szeretnénk felhívni a figyelemét arra a tényre, miszerint ezen Gadget-ek létrehozásában igen jó szolgálatot tehet egy segédprogram használata (a lemezen található példa is így készült).

A GadTools Library nem csak a Gadget-ekben nyújt segítséget, hanem a grafikai tudását is kiegészíti az új megjelenésű Border-ek rajzolása segítségével. Ezt két függvény végezheti. A függvények a **DrawBevelBoxfJ** és a **DrawBevelBoxAfJ** nevű függvények. Ez utóbbinál a tag-lista megadását egy tömbben tehetjük meg. De nézzük a paraméterek listáját az elejétől kezdve.

Az első paraméterként az ablak rastport-jára vár egy mutatót. Ezután négy változóban a BevelBox méretei definiálhatóak, és végül a megjelenítés módját befolyásoló tag-ok felsorolása. A keret tulajdonságait befolyásoló tagok a következők lehetnek:

GTBB_Recessed: (boolean) a bemélyedó' hatást keltő keret rajzolása.
GTBB_FrameType: ennél a tag-nál adhatjuk meg a keret tulajdonságát. Ezek a következők:

BBFT_BUTTON: Gomb köré rajzolódó keret. (Szimpla keret)
BBFT_RIDGE: Dupla keret, a két keret közötti rész nincs.
BBFTJCONDROPBOX: Dupla keret, a két keret között 2 pixel távolság van.

A DrawBevelBoxra is találhatunk példát a lemezen.

1.6.3.6. A V39-es rendszer újdonságai

Ha valakinek A 1200-asa vagy esetleg A4000-ese van, (nagyon örüljön neki), valószínűleg találkozott már olyan Gadget-ekkel, amelyeket eddig nem tárgyaltunk. Esetleg nem is tudta, hogy az(ok) is gadget(ek). Gondolok itt például a Prefs könyvtárban tanyázó Palette programban megtalálható szín beállító karikára, és a mellette meghúzódó proporcionális gadgetre. amelynek háterében csak úgy csillognak a színek. Vagy esetleg feltűnhetett már egyeseknek az. hogy sok programban mintha egyforma Gadget-eket használnának a zenelejátszás elindítására, megállítására, előretekerés, stb.. valamint az animáció-lejátszásnál használt hasonló funkciók ellátására. Ez nyilvánvalóan nem lehet véletlen, mint ahogyan nem is az. Ezek a Gadget-ek mégsem részei a rendszernek abban az értelemben,\ámelyben az eddigi Gadget-ek voltak Ezek a Gadget-ek a V39 kiadott új rendszer koncepciójában vállaltak olyan szerepet, miszerint részei az ún. Boopsi. az objektum orientált Intuitionnak. Ennek megfelelően ilyen típusú Gadget-eket mi is létrehozhatunk és kapcsolhatunk a rendszerhez.

Gadget-ek

Ezért a rendszernek ezen Gadget-ek kezeléséhez külön függvényeket rendel, mivel elérhető ezeknek a Gadget-eknek az összes funkciója. Etaben az esetben is a tag technikára épül a rendszer, mint a GadTools-nál. Ezeket a tag-okat azonban két részre kell osztanunk. Az első részbe az olyan tag-ok tartoznak, amelyekkel a Gadget-ek általános tulajdonságait adhatjuk meg, mint például azt. hogy hová kerüljön az ablakban, mekkora legyen stb. A második részbe azok a tag-ok kerülnek, amelyek az adott gadget sajátjai, azaz gadget specifikusak. Ezek lehetnek például a színkarikánál (colorwheel) a szín össze tevők, vagy a magnószerű kézelőgomboknál (tapedeck) az elindulásnál benyomva megjelenő gadget száma. Ezek a Gadget-ek a sys: meghajtó Classes könyvtárának a Gadgets alkönyvtárában vannak definiálva. A bővítése igen leegyszerűsödik az újabb típusok installálása. A hírek szerint az újabb rendszereknek része lesz a **MUI** (MagicUserlnterface). amely igen szétkavarhatóvá teszi mind a Gadget-eket, mind az Intuition hatáskörébe tartozó object-eket (pl. ablakok, screen-ek. menük stb.). Nos ez a 39-es rendszer már láthatóan közelíti ezt a koncepciót. (Egy későbbi kötetben majd részletesen tárgyaljuk a MUI programozását is.) Nos, lássuk milyenek is ezek a Gadget-ek a gyakorlatban. Természetesen a lemezen ehhez is találhatók példaprogramok. Talán kezdjük a Gadget-ek létrehozásával. Ezeket a Gadget-eket az Intuitionban megtalálható NewObject függvénnyel hozhatjuk létre. A függvénynek paraméterként először egy pointert kell adnunk az IClass struktúrára. Ez többnyire NULL. Másodikként a ClasslD-t megadó pointert kell definiálnunk, ami a létrehozandó object nevére mutat. Ezen a nevek is. mint a library nevek csupa kisbetűvel íródnak, mint nemecseké. A jelen esetben ezek a nevek például az alábbiak lehetnek: **„colorwheel.gadget". „gradlentslider.gadget"** vagy **„tapedeck.gadget".** Ahol a colorwheel gadget a színbeállító karikát jelenti, mig a gradientslider a proporcionális gadgelet, melynek a hátterében színorgiák érhetőek el, valamint a tapedeck, amellyel a magnó gombját utánzó Gadget-eket hozhatjuk létre. Ezek után a Gadget-ek értékeit meghatározó tag-okat sorolhatjuk fel. Ezek az alábtaiak lehetnek (először az általános tag-okhoz tartozóak):

GAJLeft: a gadget y koordinátáját adhatjuk meg.
GA_RelRight: az ablak jobb borderéhez relatívan helyezi el a gadget-et.
GA_Top: a gadget x koordinátájának megadása.
GA_RelBottom: az ablak TopEdge értékéhez relatívan helyezi el a gadget-et.
GA_Width: a gadget-ünk szélessége.
GA_RelWldth: az ablak szélességéhez relatívan helyezi el a gadget-et.
GA_Height: a gadget-ünk magassága.
GA_RelHeight: az ablak magasságához relatívan helyezi el a gadget-et.
GA_Text: pointer a szövegünkre.
GAJmage: pointer az Image struktúránkra, amikor a gadget nem választott.
GAJBorder: pointer a gadget keretét meghatározó Bordér struktúránkra.
GA_SelectRender: pointer az aktivizált gadget Image-re.
GA__Highlight: (boolean) a gadget jelezze az aktivizálását.
liA_DJsaDlea: (Dixnccinj a gnagcL nem vnioa/iLh.aú.

GA_GZZGadget: (boolean) a gadget GimeZeroZero ablakhoz csatolt, és a külső részébe kerüljön az ablaknak.

GA_ID: a gadget ID-je.

GA_UserData: a gadget UserData-je adható meg.

GA_SpecialInfo: pointer a gadget-hez csatlakozó speciális struktúrára (prop esetében a PropInfora. String és Integer esetében a StringInfora stb.).

GA_Selected: (boolean) a gadget kiválasztott.

GA_EndGadget: (boolean) requesternél a megnyomására bezáródik a requester.

GAJmmediate: (boolean) lásd. GACTJMMEDIATE flag.

GA_RelVerify: (boolean) lásd. GACT_RELVERIFY nag.

GA_FollowMouse: (boolean) lásd. GACT_FOLLOWMOUSE flag.

GA_RightBorder: (boolean) lásd. GACT_RIGHTBORDER Hag.

GA_LeftBorder: (boolean) lásd. GACT_LEFTBORDER flag.

GAJTopBorder: (boolean) lásd. GACT_TOPBORDER flag.

GA_BottomBorder: (boolean) lásd. GACT_BOTTOMBORDER flag.

GA_ToggleSelect: (boolean) lásd. GACT_TOGGLESELECT flag.

GA_Previous: segítségével egy gadget fűzhető be a Gadget-ek sorába, és a befűzött gadget a gadget-ünk előtti lesz a sorban. Ha már az ablak, vagy a requester megnyitott, ezt ne használjuk! Használjuk helyette az **AddGListfJ** függvényt.

GA_Next: mint fent. csak a sorban utána fog következni.

GA_DrawInfo: néhány díszes gadget-nek szükséges a DrawInfo.

GA_IntuiText: az ItuiText struktúra megadására.

GA_LabelImage: egy Image-ra mutató pointer, amit a GadgetText helyeként használhatunk.

GAJTabCycle: (boolean) lásd. GFLG_TABCYCLE flag-nál. (V37!)

GA_GadgetHelp: (boolean) segítségével elérhető, hogy a gadget GadgetHelp üzenetet küldjön a prg-nek. (V39!)

GA_Bounds: pointer egy IBox struktúrára amely bemásolódik az extended gadget struktúrába. (V39!)

GA_RelSpecial: (boolean) jelzi, hogy ez a gadget egy „special relativity" tulajdonsággal rendelkezik. (V39!)

A proporcionális gadget osztály atributtumai (Erről a részről részletesen az 1.7.3.3 részben):

PGA_Freedom: csak az egyik használható a FREEVERT vagy a FREEHORIZ közül.

PGA_Borderless: (boolean) a prop.-unk keret nélküli legyen.

PGAJHorizPot: a Knob vízszintes helyzetét adhatjuk meg.

PGA_HorizBody: a Knob méretét állítja vízszintes irányban.

PGA_VertPot: a Knob függőleges helyzetét adhatjuk meg.

PGA_VertBody: a Knob méretét állítja függőleges irányban.

PGAJTotal: a Knob maximális elmozdulásának értékét adhatjuk meg.

PGAJVisible: A Knob pozíciója.
PGAJTop: A Knob maximálisan lehetséges elmozdulását adhatjuk meg.
PGA_NewLook: a prop.-unk új típusú lesz. (V37!)

A STRING gadget osztály atributtumai (erről a részről részletesebben az 1.7.3.2. részben.):

STRINGA_MaxChars: a maximálisan bevihető karakterek száma. Problémák adódhatnak, ha boopsi integer gadget-ben nagyobbat adunk meg, mint 15.
STRINGA_Buffer: pointer a Bufferre.
STRINGA_UndoBuffer: pointer az UndoBufferre.
STRINGA_WorkBuffer: a munka Bufferre.
STRINGA_BufferPos: a Buffer-beni pozíció.
STRINGA_DispPos: a kijelzési pozíció.
STRINGA_AltKeyMap: pointer az alternatív keymap-ra.
STRINGA_Font: pointer a használni kívánt betűtípusra.
STRINGA_Pens: a használni kívánt tollakra mutató pointer.
STRINGA_ActivePens: az éppen használt toll.
STRINGA_EditHook: az editálandó szöveg hook-ja.
STRINGA_EdltModes: az editáló módjai:
STRINGA_ReplaceMode (boolean)
STRINGA_FixedFieldMode (boolean)
STRINGA_NoFilterMode (boolean)
STRINGA_Justiflcation: a tördelési módnál a megadhatók a következők:
GACT_STRINGCENTER
GACT_STRINGLEFT
GACT_STRINGRIGHT
STRINGAJLongVal: a bevitt számot tartalmazó változómező
STRINGAJTextVal: a bevitt szöveget tartalmazó változómező.
STRINGA_ExitHelp: (boolean) segítségével kilép a szerkesztésből, ha közben mgnyomták a Help' gombot.

Mint láthatjuk, van egy pár olyan megadható tag, aminek a gadget létrehozásakor nincs igazán szerepe. Ez nem véletlen, mert a Gadget-ek állapotát egy függvény segítségével olvassuk ki. ha szükségünk van rá. Ennek a függvénynek viszont valamilyen módon meg kell hogy adjuk, mit is szeretnénk kiolvasni. Tehát ezeket a tag-okat ott fogjuk majd használni.

De ezek után következzenek a gadget specifikus részek. Először talán nézzük meg a colorwheel gadget-et. Ezért itt már nem írom ki külön-külön értelemszerűen, a tag-okat megadásra is, és a lekérdezésnél is lehet használni.

A COLORWHEEL gadget tagjai:

WHEEL_Hue: az árnyalat.
WHEEL_Saturation: a telítettség.
WHEEL_Brightness: világosság.
WHEEL_HSB: a ColorwheelHSB struktúra megadását várja, vagy adja.
WHEEL_Red: a vörös szín.
WHEEL_Green: a zöld szín.
WHEEL_Blue: a kék szín.
WHEEL_RGB: a ColorWheelRGB struktúra megadását várja, vagy adja.
WHEELJScreen: pointer a sreenre.
WHEEL_Abbrv: ha kevés a szín ahhoz, hogy az alapszíneket meg tudja jeleníteni, akkor a színeknek megfelelően betűket tesz az adott helyre, ami angol nyelv esetén a következő: „GCBMRY". De megadhatjuk magyarul is.
WHEEL_Donation: a színek, amit öröklődtek.
WHEELJBevelBox: a gadget egy BevelBoxba kerül berajzolásra. (A BevelBoxról lásd a grafikai részt. 1.1 l-es rész.)
WHEEL_GradientSIider: a gadget-hez tartozik egy gardientslider gadget ís, amit a GA_Previous-nál tudjuk megadni a rámutató pointert.
WHEEL_MaxPens: az allokálandó tollak max. száma.

Valamit a két hozzá tartozó struktúra:

```
struct ColorUJheelHSB
{
   ULONG cuj_Hue;
   ULONG cuj_Saturatíon;
   ULONG cuj_Brightness;
};

struct ColorlilheelRGB
{
   ULONG ciu_Red;
   ULONG cw_Green;
   ULONG cw_Blue;
};
```

Ezek a WHEEL_RGB-nél és a WHEEL_HSB-nél megadhatók, **111.** ilyet ad vissza. Maga a gadget elég intelligens, mivel ha nincs elegendő szín az alapszínek jelöléséhez, a színek szerint beosztja azokat területekre, és a területbe a színnek megfelelő karaktert helyezi. Ha azonban a screen 8 bitplane-es, akkor teljes színében tündököl.

Természetesen, ha kevesebb szín lehetséges a két véglet között, alkalmazkodik hozzá, .különböző trükkökkel. Tekintsük meg ezeket a példaprogramban, egyszerűen írjuk át a screen-ünk Dépth-jét kisebbre, és majd meglátjuk. A gadget Knob-jának minden mozdulatát érzékelni tudjuk, nem árt beállítanunk a GA_FollowMouse flag-et. A példa elég szemléletesen magyarázza, így nem részletezem tovább, hanem nézzük a következőt:

A gardientslider gadget tagjai:

GRADMaxVal: a slider maximális értéke.
GRAD_CurVal: a slider aktuális értéke.
GRAD_SkipVal: ha a Knob-ra kattintunk, az mennyit mozduljon.
GRADJKnobPixels: a Knob mérete pixelben.
GRAD_PenArray: a használt tollak színei (mutató egy tömbre).

Maga a gadget mint a neve is mutatja a slider gadget-ekkel van rokonságban így az ott szerzett tapasztalatunkat itt kamatoztathatjuk.

A harmadik a tapedeck gadget tag-jai:

TDECK_Mode: a megnyomott gomb értékét adja vissza, vagy az elinduláskor lenyomott gombot adhatjuk meg. A megadható lll. kapható értékek:

BUT_REWIND: visszapörgetés.
BUT_PLAY: lejátszás.
BUT_FORWARD: előrepörgetés.
BUTJ3TOP: állj.
BUT_PAUSE: pillanat állj.
BUT_BEGIN: kezdet.
BUT_FRAME: a frame slider-e.
BUT_END: vég.
TDECK_Paused: (boolean) a paused lenyomott-e, vagy ezután legyen lenyomva. (TRUE esetén)

TDECKJTape: (boolean) a gadget-et magnószerűen, vagy animáció kontrolként akarjuk-e használni (a defaultja a FALSE).
TDECK_Frames: a frame-k száma az animációban. Csak ha animáció kontrollként használjuk.
TDECK_CurrentFrame: az aktuális frame száma.

Ha már így mindent tudunk, nézzünk egy példát a tapedeck gadget-ekre, mintha animáció-lejátszóban használnánk. Tehát először is definiáljuk a szükséges változókat. Majd a könyvtárakat megnyitjuk, természetesen V39 es verziószámmal, mivel ezek a gadget-ek innen élnek. Utána, ha sikerült megnyitnia minden szükséges könyvtárat, próbáljuk meg a gadget-et is megnyitni, ami körülbelül így nézhetne ki:

```
struct Gadget* gad; /* definiáljuk a gadget-ünket jelentő uáltozót •/

gad = (struct Gadget*) NewObject(NULL,"tapedeck.gadget",
        Gfl_Top, 5B,  /* Mondjuk kerüljön az ablak 50,50 es
                         koordinátájára. */
        GflJ-eft, 50,

        TDECK_Tape, TRUE, /* Miuel animáció-lejátszóban
                             akarjuk használni. */
        TDECK_Mode, BU_STOP, /* flmikor megjelenik a gadget,
                                a STOP gombja leggen
                                megnyomua. */
        T0ECK_Frames, 144,  /• Ha már tudjuk, hogg háiig
                               frame-ből fog állni az animációnk,
                               (itt 144-ből) */
        TflG_END);
```

Majd a gadget-ünket befűzzük az éppen inicializálandó ablakunk struktúrájába, az ablak gadget-jeként. Az ablaknál adjuk meg az IDCMP flag-oknál a szükségeseket. Az IDCMP üzenetek a lekérdezése a szokásos módon történik, (nem kell a GadTools Library üzenet kezelőjét használnunk) csak a gadget-eink állapotát nem az **lnuiMessage->Code** mezőjéből nyerjük, hanem az Intuitionnak van egy függvénye, amely ezekből az object-ekből vissza tudja nyerni az információt. Ez a függvény a **GetAttrfJ** függvény.

A függvény paraméterként azt várja, hogy először megadjuk neki, hogy mit szedjen be, persze a tag-ok használatával mondjuk ezt meg neki. Valamint azt, hogy miből szedje be, és végül hová rakja azt. A míből-nél egy mutatót vár az object-ünkre, és a hová-nál annak a változónak a mutatóját, amibe rakja a kapott eredményt (ULONG változónk pointere). Ha menet közben tudjuk meg a frame-k számát, vagy ez megváltozott, egy függvénnyel változtathatjuk meg a gadget-ünk ezen értékét is. Ez a függvény a **SetAttrsfJ.** Első paraméterként egy pointert vár arra az objectre, amelyen a változásokat akarjuk eszközölni, majd a tag-lista követi, ahol a változások miéntjét tudjuk megadni. Ezek után már csak az object becsukásáról kell néhány szót mondanunk. Ez szintén egy függvény feleadata, mégpedig a **DisposeObjectfJ** függvényé. Ez a függvény csak egyetlen paramétert vár, a megszüntetendő object pointerét. Ezek után lezárhatjuk a megnyitott könyvtárakat is (az ablakot még az object lezárása előtt lezártuk), és kilépünk a programból.

Utólag most már beláthatjuk, hogy logikus és egyszerű a gadget-ek kezelése. A lemezen szinte mindenre találhatunk példát, ahol ezeket a száraz tényeket működve is bemutatjuk. A következő részben a menükről fogunk hasonló értekezést folytatni.

1.7. A Mentik

Ha az Amigán dolgozunk, igen sokszor és szinte elkerülhetetlenül használunk úgynevezett menüket. A menüket az egér jobb gombjának lenyomásával tehetjük láthatóvá az éppen aktív ablakunkon. A menük a screen bal felső sarkában jelennek meg. Bár néhány programot elindítva ez megváltoztatható olyan módon, hogy az egér aktuális helyzetébe tegye a menüt. Azonban a rendszer erre nem ad módot egykönnyen. A menüket igen sokféle módon és úton használhatunk. A rendszer ugyanúgy támogatja használatukat, mint a gadget-ek esetében, bár nincs annyi belőlük mint a gadget-ekből. Azért mi is igen sokfélét tervezhetünk. Rendszerszinten nem feltétlenül tudja azt, amit a többi grafikus rendszer menüi, de néhány trükkel ezek is megvalósíthatóak. De mitől lenne jobb, ha a hasonlítanának a Windows95-re, vagy egy Unix-os X Windows-ra. Szerintem sokkal inkább látványos a MUIban megvalósított menü, bár kétségtelen a SiliconGraphics-on látott 3D-s menü a nyerő (Amigán ezt már megvalósították, de valamiért nem látom elterjedni). Ezer szónak is egy a vége, azért nincs oka az Amigának szégyenkezni. (Leginkább azért, mert én még nem láttam a Windows alatt Amigás menüket, de még 3D-set sem.)

1.7.1 *Mit hogyan is hívnak*

Ennyi bevezető után lássuk, milyenek ezek a menük. Azt hiszem nehéz kitalálni, hogy itt is struktúrákat kell inicializálnunk, és utána az összes munkát rábízhatjuk az Intuition-ra, aki láthatóan nagy igyekezettel fog megfelelni az elvárásainknak. Először azonban nem árt, ha tisztázunk néhány menüvel kapcsolatos szakkifejezést. A legelsőt akkor látjuk meg, ha egy aktív ablak jelenlétében megnyomjuk az egér jobb szemét bárhol a screen-en. (... és nem fut egy olyan program, amely csak az ablakban lenyomott egérre.mutatja meg.) Tehát ilyenkor a screen fejlécében feltűnnek a menüket jelentő felírások. Például aWorkbench esetében a következők: Workbench Disk Special (1.3-as esetében), vagy Workbench Window Icons Tools (A 2.0-ás és 3.0-ás gépek esetében). Ezeket a menü neveket hívják az Amigán **Menü Strip**nek. Tulajdonképpen ezek a gyökér menük. Általában a felhasználói programoknál az első menü strip a Project névét kapja, ahová a file műveletekkel foglakozó menük kerülnek. Például az Open (betöltés). Savé (kimentés). Savé as (kimentés más néven), Print (nyomtatás), New (új adat) .About (a program készítőiről nyerhetünk információt), vagy a Quit ill. Exit (a programból való kilépés). Ha ezek után az egérrel a menüstrip felé érünk, láthatóvá válnak ezek a Menü Item-ek. Ha ezek után a kívánt menü ítem-en elengedjük ọ^cgunnh goi.iVjíM.. ^ „ , . „ ü t „ „ ;u«w\lt ,fcti™á tennünk P's El7 adótt Drqgfam végrehajtja a kijelölt dolgokat. Például kimenti a forrásunkat stb. Mielőtt továbbmennénk tisztáznunk kell. hogy az aktivizálásnak több módja is

A*

van. A fent említett az általános, de megkülönböztetünk még olyan esetet is, amikor a jobb egérgomb lenyomva tartása mellet a bal egérgombbal is kattintunk. Ezt menü acüon-nak nevezzük. A másik esetben a jobb és a bal gombot is lenyomva tartjuk. Ilyenkor a kiválaszthatóak közül mindet kiválasztottuk, és ezt menü drag-nak nevezik. A menü item-eken kívül még adhatunk meg al item-eket is, ha erre szükségünk van. Például olyan esetekben, ahol még el kell valamit döntenie az user-nek. Például a Savé menüitem esetében hová történjen a mentés, egy fájlba vagy nyomtatóra? A menü item-ek almenüjét subitem-nek nevezik. Több nem ágyazható egymásba, de erre nincs is szükség. Ezeket teszi szemléletessé a 8.ábra.

Mint ahogy azt már a programokban sokszor láthattuk, a menü item-ek mellett szerepel egy Amiga-gombra utaló ikon, és mellette egy másik gombra utaló karakter. Ezzel a formával hozza tudtunkra, hogy a menü által kiváltott funkciót a billentyűzetről is el lehet érni, a jobb Amiga gomb és vele együtt megnyomott másik gombbal. Ha menünkhöz ilyen megoldást akarunk csatolni, azt rendkívül egyszerűen megtehetjük: csak a kívánt gomb karakterét kell megadnunk a definiáló struktúrában, az Intuition elvégzi a többit. A billentyű-kombináció lenyomása esetén a programunk olyan üzenetet kap, mintha a menüt aktivizálták volna. A létrehozáskor azonban felmerülhet egy olyan probléma, hogy a menü item keretébe nem fér el az itemhez tartozó billentyű-kombinációt jelentő ikon. mivel a keret ehhez már kicsi. Ilyenkor egyszerűen a keretet meghatározó pixel méretéhez hozzá kell adnunk a COMMWIDTH konstanst, ami biztosítja számunkra azt, hogy mindig a megfelelő szélességű legyen az item kerete. Ha esetleg az ablakunk amin a menüt megjelenítjük, alacsony felbontású (LowResulution) screenhez csatlakozik, akkor ez a konstans érték a LOWCOMMW1DTH legyen (ezek a konstansok az Intuition.h-ban vannak definiálva).

A menü item-ek közül két fajtával találkozhatunk. Az egyik fajta az, amelyet annyiszor használhatunk, ahányszor csak akarunk, és a programunk mindegyikről kap visszajelzést. Ezt a fajtát Action Item-eknek nevezik. A másik fajta az Attribute Item-ek. Ezeket csak egyszer tudjuk aktivizálni, mert egymást kiváltó kapcsolóként működnek. Ha tehát akkor szeretnénk használni, amikor az egymást kizáró eseteket kell kezelnünk, akkor ezt a fajta menü item-et kell használnunk. Ez hasonló a gadget-ek működéséhez. Ahhoz, hogy használni tudjuk, a Mutual Exclude-dal kell megismerkednünk (a következő bekezdés). Amikor az ilyen típusú menü item-ünket aktivizálták, vagy aktív állapotba hozták, egy kis pipa szokta jelezni ezt az item szövege előtt. Természetesen ilyet mi magunk is tervezhetünk, a szokásos Image struktúrával megadhatjuk a rendszernek, hogy az ablakunk esetében mit is használjon. Ezt a NewWindow struktúrában kell megadnunk, a CheckMark tag-nál várja a pointert az Image struktúránkra.

Használatakor itt is felmerülhet az a probléma, miszerint nem férünk el a szövegnek fenntartott helyen a menü item keretében. Ebben az esetben csak egy **CHECKW1DTH** konstanst kell hozzáadnunk a már kiszámolt értékünkhöz. Ez a konstans tartalmazza a szükséges szélesség értéket (az Intuition.h-ban definiálva). Ha low resolution screen-en akarjuk használni, használjuk a **LOWCHECKWIDTH** konstanst.

A menük

A Mutual exclude-ról már beszéltünk a gadget-eknél is, és az imént egy pár sorral feljebb is. Most a menük esetében nézzük meg, hogy miképpen is használjuk. A mutual exclude-ra mint már említettük, akkor van szükségünk, ha egymást kiváltó menü item-eket szeretnénk készíteni. Például ha egy szövegszerkesztőt írunk és az user-nek változtathatóvá tesszük a kiírási betűkészlet megjelenését. Gondolok itt a dőltbetűs írásra, az aláhúzottra, és használhatjuk ezt a fajta menüt a normál, **III** a bold, azaz a dupla írásvastagság állításánál. Mind a kettő egymást kizárva létezhet csak, hiszen nem írhatjuk egyszerre vastagítva is, és normál vastagsággal is a betűket. Arról, hogy a menü item-ünk ilyen típusú legyen, egy egyszerű flag beállításával gondoskodhatunk. Ez a flag a **CHECKIT** flag a Menultem struktúrában. Ha ez beállított, az Intuition automatikusan elvégzi az aktivizálásakor aktuális más item-ek deaktivizálását. Hogy melyek legyenek ezek az item-ek. nos azt adhatjuk meg a Mutual Exclude segítségével. Itt egy 32 bites számot kell megadnunk, amelyben az egyes bitek jelentik az egyes item-eket. Ebből következően 32 ilyen item-ünk lehet egy menün belül. Ha például azt szeretnénk, hogy az első item ilyen legyen, és váltsa a második item-et, akkor az első item-nél a OxFFFFFFFE számot kell megadnunk (ami a következőnek felel meg binárisan: 11111111111111111111111111111110. azaz a legkisebb helyiértékű biten 0, a többin l-es). Az összes többi item-nél (ebben a menüben csak!) a 0x00000001 számot kell megadnunk és máris egymást váltva lehet aktivizálni a stílus menünkben az item-eket.

A menük használatát mint a gadget-eknél, először az őket inicializáló struktúrákkal kell kezdeni. A programban a menük megjelenítését a **SetMenuStripfJ** függvény végzi. Ne felejtsük el, hogy az ablak becsukása előtt a menüket le kell hogy szedjük az ablakról. Ezt a feladatot igen jól megoldja a **ClearMenuStripfJ** függvénye az Intuition-nak. Nézzük meg közelebbről azokat a struktúrákat. Először a Menü struktúrát vegyük szemügyre, amely az alábbi:

```
struct Menü
{
    struct Menü *NeKtMenu;
    SHORT LeftEdge, TopEdge, LUidth, Height;
    USHORT Flags;
    BYTE *MenuName;
    struct MenuItem *FirstItem;
    SHORT JazzH, üazzV, BeatX, BeatV;
};
```

Ahol a mezők jelentése a következő:

NextMenu: pointer a következő menüstruktúrára a Menü strip-ben. Értéljpiszerűen az utolsó Menü struktúrában ez NULL. A menüket is így láncolva éri el a rendszer.

LeftEdge: a menü X irányú koordinátája.

TopEdge: a menü Y irányú koordinátája. Ha azt akarjuk, hogy a menünk a screen tetején jelenjen meg, akkor itt mindig O-át használjunk. Általában Amigán ez az elterjedt. Régebbi Amigákon ezt a rendszer nem használja, és így mindig 0.

Width: a menü szélessége pixel-ben.

Height: a menük magassága. A régebbi rendszerek ezt szintén nem használják, ezért ez is mindig 0.

Flags: jelenleg csak két különböző flag létezik, és ezek a következők:

MENUENABLED: ha ez a flag beállított, az azt jelenti, hogy a menü választható, tehát az user használhatja. Ha nem beállított, akkor az Intuition úgy rajzolja ki, hogy csak minden második pixel látszik, jelezvén, hogy a menü item-ei nem választhatóak, azaz a menü szellemé vált. Ha ennek a flag-nak az állapotát menetközben szeretnénk változtatni, használjuk az Intuition könyvtár **OnMenuO 111, OffMenuO** függvényeit.

MIDRAWN: ezt a flag-ot csak az Intuition használja akkor, amikor az user megjelenítette a menüt.

MenuName: egy mutatót vár arra a szövegre, amely azt tartalmazza, aminek a menü strip esetén meg kell jelennie a képernyő tetején. A szövegnek 0-ra kell végződnie.

FirstItem: egy pointert vár a menühöz tartozó első menü item-re. A menü item struktúráját lásd a következő bekezdésben.

JazzX, JazzT, BeatX, BeatY: ezeket az érdekes nevű változókat az Intuition használja csak és kizárólag, tehát inícializáláskor 0-val kell **feltol**tenünk. (A rendszerben rajta lehet kapni a készítők humorát, vagy inkább sokrétűségét, de az is lehet, hogy csak a fiatalságuk játszott néha.)

Most lássunk egy példát arra, hogy egy menü struktúrát hogyan is inicializálhatunk saját magunk:

```
struct Menü my_menu =
    {
        NULL, /* Nincs köuetkező menünk, egyenlőre csak ez az egy. */
        0, /* LeftEdge, 0 pixel-lei jobbra a screen bal szélétől.*/
        0,     /* TopEdge, 0 pixel-lei lefelé a screen tetejétől. */
        60,    /* UJidth, 60 pixel szélesen. */
        0,      /* Hz Inutition átugorja az itt beállított */
                /* értéket, ezért íjunk ide O-át. */
        MENUENRBLED, /* Flags, a menü uálaszthaló legyen. */
        "Project",/* MenuName, a menünk megneuezése. */
        &First_item, /* FirstItem, a menühöz tartozó első */
                     /* item-ünk pointere. */
        0,0,0,0  /* JazzH, JazzV, BeatH, BeatV, ezeket csak az */
                 /• Intuition használja! */
    };
```

Láthatóan elég egyszerű struktúra. De most következzen a menü item struktúra:

```
struct MenuItem
    {
        struct MenuItem* NextItem;
        SHORT LeftEdge, TopEdge, LUidth, Height;
        USHORTFlags;
        LONG MutualExclude;
        RPTR ItemFill;
        RPTR SelectFill;
        BVTE Command;
        struct MenuItem *SubItem;
        USHORT NeKtSelect;
    };
```

A mezők jelentése a következő:

NextItem: a következő Itern-re mutató pointer. Ha az utolsó item-mel van dolgunk, értelemszerűen NULL-t kell írni.

LeftEdge: az X irányú pozíciója a kiválasztó keretnek, amely relatív a menü LeftEdge pozíciójához képest.

TopEdge: az Y irányú pozíciója az item-ünknek, amit az Intuition számol ki futás közben. Értékét az Intuition a használt font nagyságától, és az item-ek mennyiségétől függően számolja, ezért ezt a változót O-nak szoktuk megadni.

Width: az item kiválasztó keretének szélessége.

Height: az item kiválasztó keretének magassága.

Flags: ezeket a flag-okat használhatjuk, vagy az Intuition használja:

CHECKIT: ha azt akarjuk, hogy az item-ünk „attribute" item legyen, akkor ezt állítsuk be. Egyébként „action" lesz. Ha ez beállított, akkor az Intuition a kiírásunk elé fog tenni egy kis pipát (checkmark), ha aktivizált. Ilyen esetben figyeljünk arra, hogy legyen ehhez elég helye az Intuition-nak.

CHECKED: ha az item „attribute" (a CHECKIT flag beállított), az Intuition beállítja ezt a (lag-ot. amikor az user kiválasztotta azt. Ez a flag automatikusan törlődik ha a MutualExclude alapján kell neki. Ezt a ilag-ot kell használnunk akkor, ha a kirakáskor ezt az itemet szeretnénk hogy aktív legyen.

HIGHFLAGS: beállításával közölhetjük az Intuition-nal. hogy mi történjen, ha az user kiválasztotta az item-et. Az itt megadható további flag-ek a következők:

HIGHCOMP: kiválasztáskor a színeket a komplemensükre változtatja, mint a gadget-eknél.

HIGHBOX: a kiválasztáskor az item köré egy keretet rajzol.

HIGHIMAGE: a kiválasztáskor a mi általunk definiált image-t rajzolja ki, vagy írja ki az intuiText-kent megadott szöveget.

HIGHNONE: nem csinál semmit a kiválasztáskor.
ITEMENABLED: ez a flag állítódik be. ha az user kiválaszthatja az item-et. Ha a flag nem beállított, az user nem tudja az item-et kiválasztani. Az item „szellemként" jelenik meg. azaz minden második pixel-t rajzol ki az Intuition.
ITEMTEXT: a ílag-ot akkor állítsuk be. ha az item-hez szöveg tartozik (IntuiText). Ha Irnage-t akarunk, akkor töröljük. Ha beállított, a szöveget a ItemFill-nél és a SelertFill-nél kell megadnunk.
COMMSEQ: ha azt szeretnénk, hogy a menü item-et a billentyűzetről is el lehessen érni, ezt a flag-et be kell állítanunk. Ne felejtsük el azt a billentyűt megnevezni, amelyet együtt kell megnyomni majd az jobb Amigával. Ezt a Comrnand mezőnél adhatjuk meg az item struktúrában.
MENUTOGGLE: ha azt szeretnénk elérni, hogy az item-et mint egy kapcsolót lehessen működtetni. Tehát ha aktivizálom, addig marad aktív állapotában, ameddig újra nem aktivizálom, amikor is visszaáll nem aktív állapotába.

MutualExclude: az Intuition-nak itt adhatjuk meg azt, hogy az item-ek vagy subitem-ek közül melyek tartoznak bele az egymást kiváltó sorba. A megadása egy 32 bites számban történik, ahol minden egyes bit egy itemnek. vagy sub item-nek felel meg. Például, ha megadjuk azt a számot, hogy OxOOOOOOOB. azaz binárisan 00000000000000000000000000001011-t .akkor az azt jelenti az Intuition-nak. hogy az 1. a 2. valamint a 4-es item vesz ebben részt.

ItemFill: egy pointert vár ide. amely az ITEMTEXT flag beállítása esetén egy IntuiText-re mutat, vagy ha nincs ez a flag beállítva, akkor egy Image struktúrára. Az itt megadott szöveget vagy ikont a menü item akkor használja, amikor az user megnézi a menüket, tehát nem aktivizálta még az item-et.

SelectFill: szintén egy pointer mint fent. azzal a különbséggel, hogy ez akkor válik láthatóvá, amikor az user az egérpointert az item fölé mozgatta. Azaz aktivizálja. Ezt a mezőt csak akkor veszi figyelembe, ha a HIGH1MAGE flag beállított

Command: ha egy billentyű megnyomásakor is szeretnénk a menü item funkcióját elérni, itt kell megadnunk annak a billentyűnek a karakterkódját, amellyel a jobb Amiga gombbal együtt megnyomva a kívánt hatás létrejön. Az itt megadott parancs csak abban az esetben hatásos, ha a COMMSEQ flag beállított.

Subltem: egy Menultern struktúrára mutató pointert vár, amely az itemhez tartozó sub item-et definiálja. Ha nincs ilyenünk, akkor NULL-t kell megadnunk. Ha a menü nem aktivizálható, azaz szellem, akkor az Intuition ezt a részt átugorja.

NextSelect: az Intuition használja. Inícializáláskor a MENUNULL értéket kell itt megadnunk. Az Intuition ebben a mezőben tárolja a menü kiválasztásakor azt az egyetlent, amelyet kiválasztottak. Ha nem volt ilyen, akkor MENUNULL. (lásd. később).

Példaképpen nézzünk meg egy Menü Item struktúrát inicializálva:

```
struct MenuItem *elsö_item =
{
        második_item;  /* NextItem, pointer a második item */
                       /* struktúrájára. */
        0,             /* LeftEdge, 0 pixel-lei jobbra. */
        0,             /• TopEdge, 8 pixel-lei lejjebb. */
        150,           /• UJidth, 150 pixel széles. */
        10,            /* Height, 10 pixel magas. */
        CHECKITI       /* Flags, egy attribute item lesz, •/
        CHECKEDI       /* aktiuizált, amikor elindul, */
        COMMSEQI       /* bill.-röl is elérhető, */
        HIGHCOMPI      /* a kompiemesére uáltja a színeit*/
                       /* amikor aktiuizálják, •/
        ITEMENRBLED,   /* Hz item uálaszthatö, és nem szellem item. •/
        0x00000037,    /* MutualExclude, az 1,2, 3, 5 és a 6 •/
                       /* item-ekkel egymást kiuáltják. •/
        &my_image,     /* ItemFill, Miuel az ITEMTEHT flag nincs */
                       /* beállítua, így egy kis ikont teszünk a
                       menünkbe, •/
                       /• és az arra mutató pointert adjuk meg itt. */
        NULL,          /* SelectFill, Nem használunk alternatiu */
                       /* image-t, hiszen a komplemens színeket
                       állítottuk be a flag-oknél. */
        "0",           /* Command, R "0" betű gombjáual ha
                       egyszerre nyomjuk meg a jobb Hmiga
                       gombot ugyanaz fog történni mintha az
                       item-et aktiuizáltuk uolna. */
        NULL,          /* SubItem, Nem tartozik ehhez az item-hez */
                       /* subitem, ezért NULL. */
        MENUNULL       /* NextSelect, később állítja az Intuition. */
};
```

Ezen ismeretek birtokában már képesek vagyunk a menü struktúrą iniciallzálására. Azonban a menü kirakása nem olyan egyszerű - nem sokkal komplikáltabb - mint a gadget-ek esetében. Mivel az ablaknak nincs a struktúrájában menü mezője. így más úton kell ezt megoldanunk.

Segítségünkre lesz ebben az Intuition könyvtár **SetMenuStripO** függvénye. Mielőtt a függvériyt meghívnánk, az ablakunkat már meg kell hogy nyissuk. Mivel a függvény első paramétere egy pointer az ablakunkra, amelyhez a menüt csatolni szeretnénk. Belátható, hogy igen nehezen tudná az Intuition megvalósítani a menüt, ha ez a pointer NULL lenne, azaz az ablak még nem létezne. Második paraméterként egy pointer-t vár az első menünkre. Mivel a menük egymáshoz vannak láncolva (NextMenu), így az összes mejhü csatlakozik az ablakhoz. Ha valamil módosítani szeretnénk a menüink kinézetén, először a menüinket törölni kell az ablakról, majd ha a változta-

tást eszközöltük, kirakhatjuk újra a **SetMenuStripO-pel.** A menük törlését egy másik függvény végzi, szintén az Intuition könyvtárból. Ez a függvény a **ClearMenuStripO** függvény. Egyetlen paramétere van, mégpedig az ablak pointere, amiről a menüt el szeretnénk tüntetni. Ezt a függvényt kell meghívnunk, ha a program végére értünk és bezárnánk mindent. Ne felejtsük el azonban az ablak becsukása előtt meghívni a függvényt, mivel egy nem létező (az ablak pointere NULL) ablakról igen nehézkesen lehet törölni a menüket, és általában mindent.

7.2. *A Menük IDCMP jei*

Az eddigiekben már elég sokmindenről szóltunk, csak még arról nem, hogy ha kint vannak a menük az ablakon, miféleképpen lehet tudomást szerezni az user ilyetén ténykedéseiről. Az Amiga ezt nagyon egyszerűen oldja meg. még pedig a gadget-eknél már megtárgyalt IDCMP-kkel (lásd 1.6.1. rész). Mint már a gadget-eknél láttuk, a menünek három fontosabb IDCMP üzenete lehet. Az első az **IDCMP_MENUPICK,** amely arról tájékoztatja a programunkat, hogy ha valamelyik menüt, vagy egyiket sem aktivizálták.

Akkor is küld üzenetet, ha csak nézegették a menüket. Az **IDCMP_MENUVERIFY** üzenet mindig érkezik a programunknak, ha a rendszerben egy menüvel történt valami. Az **IDCMP_MENUHELP** csak a V36-os vagy magasabb kickstarttal rendelkező Amigákon elérhető. Akkor lehet rá szükségünk, ha programunkhoz olyan Help rendszert (Windows-on ez a súgó!) akarunk, amely a menükről is képes help-et adni. De nézzük őket sorrendben és részletesebben.

7.2.1. Az IDCMP_MENUPICK használata

Azt gondolom már mindenki kitalálta, hogy ezt az IDCMP flag-ot is, mint mindegyiket az ablak létrehozásakor kell megadnunk az ablak IDCMP-jei között. A lekérdezése ugyanaz mint más IDCMP üzenet esetén. Ha a programúnk kap egy **IDCMP_MENUPICK** üzenetet, akkor az user piszkálta a menüinket. vagy csak az egér jobb gombját nyomkodta. Arra, hogy eldöntsük melyik eset is állt fent, az érkezett IntuiMessage Code mezője tájékoztatni fog bennünket a menü aktivizálásakor. Mégpedig ezt akkor teszi, ha értéke egyenlő lesz a **MENUNULL** értékkel. Ilyenkor már biztosan tudhatjuk, hogy menüt aktivizált az user. de még nem tudjuk, hogy melyiket. Ha azonban használjuk az Intuition ItemAddressO függvényét a paraméterként megadott menü strip-ünkkel, a kezünkben lesz a választott item címe. A probléma ott kezdődik, hogy ha a programunknak nem csak egy menüt kell kezelni. Ilyenkor minden egyes menüre meg kell néznünk, hogy ő róla volt-e szó. Ez *S** probléma azonban csak látszólag komplikálja a helyzetet. A megoldás igen egyszerű és kézenfekvő. Ha egy menü item volt kiválasztva, akkor meg kell néznünk az item struktúra NextSelect mezőjében, hogy az ott található egyezik-e a MENUNULL-lal. vagy nem. Ha nem egyezik, akkor még egyszer meg kell hívnunk az ItemAddressO függvényt, és ezt addig ismételni, ameddig

nem egyezőt találunk, és már meg is kaptuk a kérdéses item címét. Az ismétlésnél természetesen a cimet kell mindig aktualizálnunk, amivel az ItemAddress-t meghívjuk. Az ezt elintéző programrész a következőképpen nézhet ki:

```
iftclass == IDCMP_MENUPICK)
{
      /* Rz user megnyomta a jobb egér gombot. •/

      /• fl menu_szama uáltozót inicializáljuk a code uáltozóual. */
      /* (IntuiMessage->Code) •/

        menu_szama = code;

      /* Hddig ebben a ciklusban uan ameddig nem egyezőt talál. */
      while(menu!= MENUNULL)
      {
      /* Beszedjük az Item címét */
      item_address = ItemflddressCí>elsö_menu, menu_szama);

      /* Ennek az Item-nek a feladatai ... */

      /• R köuetkező item menü számának beszedése. */
      menu_szama = item_address->NentSelect;
      }

      /* Rz összes item feladata amelyek kiuálasztua uoltak ... */
}
```

Ha nekünk a menü számára van csak szükségünk használjuk a SAS/C makróit erre a célra. A makrók megadják a választott menü/item/subitem számat. Ha O-át adnak vissza akkor az első menü volt választva, ha egyet akkor a második menü, és így tovább. De nézzük egyesével ezeket a makrókat. Mivel makrókról van szó így nagybetűvel írjuk őket a forrásban is.

MENUNUM(menu_száma): ha ez nullával egyenlő, akkor az első menü volt kiválasztva. Ha 1-gyel egyenlő, akkor a második, és így tovább. Ha a NOMENU konstanssal egyenlő, akkor nem volt menü aktivizálva.

ITEMNUM(menu_száma): ha ez nullával egyenlő, akkor az első item volt kiválasztva. Ha 1-gyel egyenlő, akkor a második, és így tovább. Ha a NOMENU konstanssal egyenlő akkor nem volt menü aktivizálva.

SUBNUM(menu_száma): Ha ez nullával egyenlő, akkor az első subitem g^olt kiválasztva. Ha 1-gyel egyenlő, akkor a második, és így tovább. Ha a NOMENU konstanssal egyenlő akkor nem volt menü aktivizálva.

1.7.2.2. *Az. IDCMP_MENUVERIFY használata*

Ezt az IDCMP üzenetet a programunk mindig kapja a rendszertől (persze ha beállított volt), ha az egér jobbgombját megnyomták, függetlenül attól, hogy melyik ablak volt az aktív. Tehát akkor is. ha nem a mi programunk menüjét aktivizálták. Szóval, ha arra van szükségünk, hogy ilyen üzenetet kapjunk a rendszertől, akkor állítsuk ezt be, az ablakunk IDCMP flag-jai között az IDCMP_MENUVERIFY-t. De lássunk inkább egy példát a használatára, ami sokkal beszédesebb mint bármi más.

```
/* Uárjuk a programhoz érkező IDCMP üzeneteket. */
UJait(1 « my_windoiu->UserPort->mp_SigBit);

/* Rddig ebben a ciklusban maradunk, ameddig az üzenetek összes-
ségét nem sikerül átuennünk. */
ujhile(my_message = GetMsg(my_uJindouj->UserPort))
{
    /* Miután sikerült átuennünk őket, elmentjük az */
    /* üzenetnek azt a részét, amelyre szükségünk lesz a •/
    /* feldolgozáshoz. */
    class = my_message->Class;
    code  = my_message->Code;

    /* Mielőtt uálaszolnánk az IIntuition-nak, meg kell néznünk,*/
    /* hogy nem érkezett-e IDCMP_MENUUERIFV üzenetünk. */

    ifíclass == IDCMP_MENUUERIFV)
    {
        /* Igen érkezett! Hz user aktiuizálta a menüket, */
        /* uagyis megnyomta a jobb egérgombot. Csak azt nem */
        /* tudjuk még, hogy a mi programunkon-e, uagy más  */
        /* programon. Le kell ellenőriznünk a code mezőjét az */
        /* üzenetnek. Ha az egyenlő a MENUWHITING-gal, akkor  */
        /* nem a mi programunk ablakán aktivizálták a menüt. */
        /* Ha azonban egyenlő a MENUHOT-tal, akkor a mi prg-nk */
        /• a kiuálasztott.                                  */

        ifícode == MENUUJHITING)
        {
            /* Tehát nem a mi ablakunkon aktiuizálták a menüt, */
            /* így befejezhetjük a rajzolást, uagy egyéb dolgunk, */
            /* mert az ablakban léuőket nem fogja zauarni */
            /* a menü, ugyanis a menük tönkreteszik az •/
            /* alacsony szintű grafikánkat az ablakunkban */
            /* lásd. 1.9-es rész). */
```

```
            /* H programunk így kész az üzenetre uálaszolni!  */
        }
        else
            if(code == MENUHOT)
            {
                /* R mi ablakunkon aktiuizálta az user a menüket. */

                /* Most képesek uagyunk egy kis időt nyerni,
                mielőtt az Intuitíon kirajzolná a menüinket. Uagy
                eluégezhetünk ualamít; ameddig a menüuel nem
                fejezte be az user a molyolást. Erre például akkor
                lehet szükségünk, ha rajzolóprogramot írunk,
                mert csak így tudjuk eldönteni, hogy az user
                törölni akar az egér jobb gombját aktiuizálua,
                uagy a menükhöz szeretne hozzáférni. •/

                /* Ezért le kell ellenőriznünk, hogy az egér pointere
                hol is tart. */

                if(my_iuindouj->MouseV < 10}
                {
                    /* flz V koordinátája az egér pointerének  */
                    /* kisebb mint 1B, az ablak LeftEdge-jéhez */
                    /* relatíuan. fl menü dolgai folytathatóak. */

                }
                else
                {
                    /* flz egér pointer az ablak kertén belül uan!
                    így törölhetők a menü dolgai az ablakról. */

                    /* Hhhoz, hogy ezt megtegyük a Code mezőt a */
                    /• MENUCflNCEL-re kell uáltoztatnunk.      */

                    my_message->MENUCHNCEL;
                }
            } /* iflcode == MENUHOT) */
    } /* iffClass == idcmp_menuujerify) */

/* Soha ne felejtsünk el uálaszt küldeni az érkezett IntuiMessage-
re! */
ReplyMsg(my_message);

}  /* while ... */
```

Ha beállítjuk az ablak IDCMP-jei között az IDCMP_RMBTRAP flag-ot. akkor az ablakunkhoz nem kapcsolhatunk menüket, mert az nem kezeli le azokat (lásd 1.6.1 rész a könyvben)! Viszont mindenkor pontosan követi a jobb egérgombbal kapcsolatos történéseket.

1.7.2.3. Az IDCMP_MENVHELP használata

Ezt az IDCMP flag-ot csak V37-es Intuition library-tól használhatjuk.

Segítségével a menüről is lehet Help-et kérni (Segítséget, Súgót, bár ez utóbbi Windows-os, ezért nem használnám). Ahhoz, hogy ilyen IDCMP-nk érkezhessen, az ablak nyitása előtt a tag-listában meg kell adnunk a **WA_MenuHelp** flag-ot is. Menü help üzenetet minden menü, item. subitem esetében kaphatunk. Ilyen üzenet csak a több választásos menüről nem érkezik. Tehát szellem menükről is érkezik, ezért segítséget adhatunk az usernek. hogy az a menü miért nem választható stb. Az IDCMPJV1ENUHELP esetében annak kiválasztása, hogy melyik menü, item. subitem volt kiválasztva hasonlóan tudható meg. mint az IDCMP_MENUPICK esetében.

Óvatosan bánjunk az IDCMP_MENUPICK-kel. ha az 1DCMP_MENUHELP is be van állítva. A lemezen található példaprogram azt hiszem egyértelművé teszi használatát.

1.7.3. Hogyan is működnek a menü számai

Azaz hogyan is kapjuk vissza az aktuális menü, vagy item, illetve subitem számát. Ezt a számot visszaadó szám 16 bites, és az első 5 bit tartalmazza a használt menü számát. A következő 6 bit a használt item számát tartalmazza, és a fennrnaradóak tartalmazzák a subitem számát, úgy ahogyan azt a 9. ábrán is láthatjuk.

Ezek után könnyen belátható, hogy az Intuition nem képes csak (!?) 31 ifnenü kezelésére. Valamint mindegyik menünek 63 item-e lehet, éf annak szintén 31 sub item-e. Nem gondolom azonban, hogy ennél többre szükségünk lenne programjaink során. Ha mégis, próbáljuk meg más szervezéssel áthidalni a problémát.

Nézzünk példát arra. hogy hogyan is értelmezzük a menü számát tartalmazó változó értékét. Példánkban az alábbi számot kapjuk: OxF8O3. ami binárisan az 11111 OOOOOO 00011 számnak felel meg. Ebből következően az eredményül kapott menü száma a 3. Az aktivizált item száma 0 (azaz első!), és a sub item-ei közül egy sem volt aktivizálva, mivel a kapott szám 31 volt. Ez a 31 megfelel a NOSUB konstans értékének, azaz nem volt subitem aktivizálva.

Ha az érték kiszámolására a makrókat használjuk, azt így tehetjük meg:

```
MENUNUM(0xF803) == 3 a negyedik menü.
ITEMNUM(0xF803) == 0 az első item.
SUBNUM(0xF803) ==31 (NOSUB).
```

Más út is lehetséges a menük számának lekérdezésére. Ebben segítségünkre lehetnek a SHIFrMENU, SHIFTITEM. és a SH1FTSUB makrók. Ha a programban használni akarjuk az **OnMenuO** és az **OffMenuO** függvényeket. akkor ezeknek a függvényeknek szükséges megadni paraméterként a kívánt menü számát. Ha például a második item-et a harmadik menüben „szellemé" kell tenni, a szükséges számítás az alábbi lehet:

```
menu_szama = SHlFrMENU(2) + SHIFTITEM(l) + SHIFTSUB(NOSUB);
                (harmadik menü) (második item-e)   (nincs-e sub item)
```

1.7.4. *A 2.0-ás rendszer újdonságai a menükben*

A 2.0-ás rendszer a menüben is sok újdonságot hozott. Leginkább a GadTools Library-nak köszönhetően. A legszembetűnőbb változás a külalakjukban történt. Pékiául a nem választható menüknél egy vonallal jelezhetjük a nem kiválaszthatóságát. stb. Ha ilyen új menüket akarunk használni programjainkhoz, az ablaknál ne felejtsük megadni a WA_NewLookMenus nevű flag-ol is, hogy valóban olyanok legyenek a menük. A GadTools Library a menük létrehozásához a NewMenu struktúrát használja. Ennek felépítése a következő:

```
struct NeiuMenu
{
    UBVTE nm_Type;
    STRPTR nm_Label;
    STRPTR nm_CommKey;
    UWORD nm_Flags;
    LONG nm_MutualEKdude;
    RPTR nm_UserData;
}
```

A struktúra mezőinek értelmezése a következő:

nm_Type: a típusok az alábbiak lehetnek:
NMjriTLE: menü fejléce.
NM_ITEM: szöveges Item.
NMJ3UB: szöveges Subltem.
IMJTEM: grafikus Item.
IM_SUB: grafikus Subltem.
NM_END: a NewMenu struktúrából álló tömbök végét jelző tag.

NMJGNORE: V39 alól indítva a Gadtools át fog ugrani néhány tag-ot az nm_Type mezőben.

nm_Label: ennél a mezőnél kell megadnunk a pointerünket arra szövegre, amelyet ha a menü szöveges akkor tartalmazni fog. Vagy a pointerünket az Image struktúrára kell megadnunk, ha a menü grafikus. Ha az Item. vagy **subitem nem valasztnato itt acmatjuK meg axt is,** vagy **1^1** s^poráim ^<>, ^^*l^t* akarunk bele rajzolni. A lemezen lévő példában a Style menüben a Plain ill. Bold között ilyen található. Ekkor az NM_BARLAI3EL-t kell itt megadnunk.

nm_CommKey: ez megegyezik a menü item-nél tárgyalt Command mezőjelentésével. Tehát ide azt az ASCII kódot írjuk, amellyel el szeretnénk érni a menü funkcióját billentyűzetről is.

nm_Flags: ez a mező megegyezik a Menu->Falgs. vagy a Menultem->Flags mezőjével. Erre a mezőre vonatkozó tag-ok a következők:

NM_MENUDISABLED: ha a menü nem választható. Mint a menüknél a MENUENAI3LED üag volt.

NMJTEMDISABLED: ha az Item nem választható.

NM_COMMANDSTRING: ha szöveges az Item-ünk vagy Sub Iteműnk. ennek a flag-nak a.beállításával közölhetjük a V39-es (!) Intuition-nal. hogy a megadott nm__CommKey egy olyan stringre mutat, ahol több lehetséges parancs bili. is szerepelhet.

NM_FLAGMASK: ennek a flag-nak a beállításával előre törölhetők a COMMSEQ. ITEMTEXT és a HIGH... flag-ok.

NM_FLAGMASK_V39: V39-es alatt a COMMSEQ falg-ot nem biztos, hogy törölnünk' kell. hiszen használható mint NM_COMMANDSTRIG. így ez a flag mindazt elvégz.i mint az előző a COMMSEQ kivételével. Használhatjuk a régi menü flag-okat. Mint a CHECK1T. MENUTOGLE. és a CHECKED. valamint ezeket egyszerre is, ha szükséges.

nm_MutualExclude: a MutualExclude ugyanaz, mint amit már a fentiekben tárgyaltunk. Ugyanazt a megadási módot várja el itt is (lásd. 1.6.1).

nmJUserData: pointer a felhasználói adatra. Ha a szokásos módon hozzuk létre a menünket, a rendszer mindig foglal helyet ennek a mezőnek is. Sőt még két makrót is a rendelkezésünkre bocsát ezek elérésére. Ezek a **GTMENU_USERDATAO** és a **GTMENUITEM_USERDATAO** makrók.

Az új menük meghívását a **CreateMenusfJ** függvénnyel végezhetjük. A függvény meghívásából három értékkel térhet vissza. Ezek a **GTMENUJTRIMMED,** ha túl sok menüt, item-et. vagy sub item-et akartunk létrehozni Vagy a **GTMENUJNVALID,** amelyet akkor kapunk, ha nem létező ,vagy nem jó NewMenu struktúrákból álló tömbbel hívatjuk meg a függvényt. Az utolsó a **GTMENUJVOMEM,** ha kevésnek bizonyult a rendelkezésre álló memória a menük létrehozásához.

Magának a függvénynek a következő paramétereket adhatjuk meg. Az első a létrehozandó menük NewMenu struktúrában megadott paraméterei, amelyek egy tömbben foglalnak helyet. Az első paraméter tehát ennek a tömbnek a címe.

Második paramétere a menük külalakjára vonatkozó tag-ok felsorolása.

Ezek a tag-ok a szokásos menü külalakját meghatározó tag-ok lehetnek. Ezek a következők:

GTMNJTextAttr: a menüben használt Font neve. (Természetesen TextAttr-ként megadva.

GTMN_FrontPen: a kirajzolás színe (A színregiszter száma).

GTMN_Menu: pointer a menüre, amit a LayoutMenuItemsfJ függvény használ.

GTMN_FullMenu: kérhetjük a CreateMenusQ függvényt, hogy érvényesítse az összes menüt a listánkban (V37!).

GTMN_SecundaryError: megadhatunk egy változót a CreateMenusfJ függvény futása során. A keletkezett hibát ide is berakja. A változó típusa ULONG kell hogy legyen.

GTMN_Checkmark: a checkmark-ot definiálhatjuk V39-es Intuílion esetén. Ha V36-V37-en adjuk meg. átugorja. Természetesen a Chkmark-ra mutató pointert vár. (&Image) GTMN_AmigaKey: Szintén V39-IÓl veszi figyelembe, és az Amiga bili.-t jelentő Image definiálható a segítségével.

GTMN_NewLookMenus: az új megjelenésű menük használatára kötelezzük. A flag boolean típusú. így csak a TRUE. vagy a FALSE megadása kell. Szintén V39-tól használható.

Megadáskor az egyes menükhöz tartozó tag-ok elválasztására a TAG_END konstans használható.

Magáért a menük külsejéért a GadTools Library **LayoutMenusfJ** függvényei felének. A függvényeknek meg kell adnunk első paraméterként a létező menükre mutató pointerünk, valamit egy pointert a Visulalnfo-ra. amelyet például a screen-ből nyerhetünk (A **GetVisualInfoO** függvényekkel, lásd Gadtools libray). valamint a külalakra vonatkozó tag-listánkat (ha használtuk a **GetVisualInfoO** függvényt, ne felejtsük el ezt is felszabadítani a **FreeVisualInfoO** függvény meghívásával). A menük kirakását a jó öreg **SetMenuStripfJ** függvény végezheti. Törlése ebből kifolyólag a **ClarMenuStripfJ** függvény feladata lesz. mint a normál menük esetében.

Mielőtt azonban kilépnénk a programunkból, szabadítsuk fel a **CreateMenusfJ** függvény és társai által lefoglalt memóriaterületet. Ezt a **FreeMenusO** függvénnyel végezhetjük el. Paramétereként csak a menünk mutatóját kell megadnunk. Mivel GadTools Library-t hívtunk segítségül a menük kezelésére, így ezt tartva az IDCMP üzenetek lekérdezésére is, a GadTools Library függvényeit használjuk. Különösebb problémát nem okozhat, mivel ha ilyen menüket használunk, általában gadget-eink is gadtools-os gadget-ek lesznek, és így a GadTools Library ezen függvényeinek használatát úgysem mellőzhetjük.

Egyébként a gadget-eknél ezt már megnéztük. Remélem a példaprogramok jó szolgálatot tesznek, és jól mutatják-í\z itt ismertetett lehetőségeket

1.8. A Requester-ek

Programozásaink során felmerülhet bennünk annak az igénye, miszerint az user-rel szeretnénk közölni bizonyos, dolgokat, vagy figyelmeztetni szeretnénk valamire, esetleg arra, hogy hülyeséget csinált. Azonkívül nem árt, ha visszakérdezünk egy törlő ill. fomázó parancs végrehajtása előtt, hogy tényleg azt akarja-e? De az is előfordulhat, hogy csak egy egyszerű kérdést intézünk hozzá, amire igennel vagy nemmel kell válaszolnia. Nos, ezeket a figyelmeztetéseket egyszerűen elintézhetjük az Amiga rendszerével. A figyelmeztetéseknek sok fajtáját biztosítja nekünk a rendszer. A legelső és legprimitívebb, amely csak niegvillantja a képernyőt és csippant egyet (3.0 ás gépeken a Prefs-ben beállított hangot játsza le). Ezt nevezik DisplayBeep-nek. Ennél fejlettebbek az ún. Alert-ek, amelyek igazán nem sok választást hagynak az user-nek. Ilyenek például a rendszer összeomlására figyelmeztető GURU piros keretben, a hiba felléptének helyével és okával, valamint a sokkal jobb kinézetű ablakokkal rendelkező Requester-ek. Ebben a fejezetben ezeket fogjuk áttekinteni.

1.8.1 A DisplayBeep

A DisplayBeep küldése az user felé. csak a nem igazán nagy hibáknál szükséges. Az effektust egy az Intuition library-ban lévő függvény végzi, ezzel a névvel. A függvény a következő:

Display Beep(Screen);

A függvénynek nincs visszatérő értéke. Paraméterként arra a screen-re rtiutató pointert vár, amelyen az effektust látni szeretnénk. Ha a programunk ablakból fut. és az ablak a Workbench screen-re van nyitva, akkor is el tudjuk érni ezt a mutatót. A Screen mutatója ugyanis az ablakunk struktúrájában megtalálható (ablakunk->WScreen). Meghívását kombinálhatjuk majd a Requester-ekkel a figyelem felkeltése miatt. A 3.0-ás rendszerrel rendelkező gépeken a hangjelzésére beállítható hangeffektust játssza le a gép. Az effektus beállítására szolgáló program a sys:prefs-ben található (Sound).

Ha az user olyan hibát követ el. amely egy egyszerű figyelmeztetéssel nem elintézhető, vagy esetleg a rendszer összeomlásához vezet, akkor használjuk az Alert eket ennek jelzésére.

1.8.2 Az Alert-ek

Az Alert-el is egy függvény meghívásával tudjuk aktivizálni. A függvény az Intuition library-ban található, igy minden körülmények között szinte pár mozdulattal elérhető. Annak ellenére, hogy egy függvény van. két fajta Alert

létezik. Az egyikben csak figyelmeztetünk, és a függvény visszatéréskor közli velünk, hogy az user mit választott. Ez a **RECOVERY.** A másik Alert-bó'l mindig a FALSÉ értékkel tér vissza. Ez a **DEADEND.** ebből a függvényből tulajdonképpen nincs is visszatérés, hiszen a rendszer újraindul a meghívása után lenyomott bármelyik egérgomb megnyomásával. Az újabb rendszereken a két Alert között megjelenési különbség is van, mégpedig a **RECOVERY** Alert narancs színben pompázik, míg a **DEADEND** pirosban.

A függvény szinopszisa a következő:

```
result = Dísplayfllertfnr, üzenet, magasság);
```

A függvény meghívásakor megadható paraméterek jelentése a következő:

nr: ezzel a változóval adjuk meg az Alert fajtáját, amely lehet **RECOVERY_Alert** vagy **DEADEND_Alert.**

üzenet: ez egy string (char *) változó, amelyben az üzenetet kell megadnunk, amit kiírva szeretnénk látni. A strir\g érdekesen van feloszva. Az első 2 byte-on a szöveg kiírásának x koordinátáját adhatjuk meg. míg az ezt követő byte-on a kiírás y koordinátáját várja. Ezután adhatjuk meg a tulajdonképpeni szöveget, ami a '\0' karakterig tart. Ha ezt a karaktert még követik további karakterek, azok a fenti bontásban a többi kiírandó szöveget jelentik. Ha nem követik, akkor ez volt az utolsó kiírandó szöveg.

magasság: az Alert keret magassága.

result: a függvény visszatérési értéke. Ha az Alert Recovery. akkor a visszatérési érték TRUE, ha az user a bal egérgombot nyomta meg. és FALSÉ, ha a jobb egérgombot Ha az Alert deadend típusú, akkor a visszatérési érték mindig FALSE.

Nézzünk példát a megadható üzenet megértéséhez. Azt az üzenetet szeretnénk kiíratni miszerint „ HIBA! Nincs elég memória. ". Ehhez a következőket kell megtennünk. Először definiálunk egy 32 karakteres tömböt. A tömbbe bemásoljuk a szöveget, majd az első byte-okat feltöltjük a kívánt értékekkel. Ez az alábbiakban nézhet ki forrásszinten:

```
char üzenet[32l;



strcpytüzenet, " HlBR! Nincs elég memória.");

üzenet[0]=0; /• az K pozición kisebb mint 256. */
üzenet[1]=32; /* 32 piKel az K. •/
iizenet[2]=16; /* 16 sorral lejjebb •/

/* és uéaül a szöueg utáni byte 0-ra állítása, miuel nincs több szöueg.*/

üzenet[31J=FRLSE;
```

Ha olyan üzenetet kívánunk megadni, amely több sorból áll, azt a következőképpen adhatjuk meg:

```
char üzenet[56];

    .
    .
    .
/* Hz üzenet elsó része. */
strcpyfüzenet, " HIBfl! Nincs elég memória. ");
/• flz üzenet második része. */
strcat(üzenet, " Uegyél egy böuítőt! ");

üzenet[O]=Q;
üzenet[1]=32;
üzenet[2]=16;

/•Hozzáadunk egy nullát a szöuegünk után, mert az letörlődött,*/
/*az strcatO függuény meghíuásakor.*/

üzenet[3B]="\0";

/* Majd az ezt köuető byte-ot igazra állítjuk, hiszen uan még */
/* egy kiírandó szöuegünk.*/

üzenet[31]=TRUE;

/* fl második üzenet koordinátái. */
üzenet[3Z]=0; /* Kissebb az H mint 256. */
üzenet[33]=32; /* 32 pixelre a szélétől. */
üzenet[34]=32; /* 32 sorral lejjebb. */

üzenet[55]=FHLSE; /* Nics több. */
```

A RECOVERY Alert-ról a lemezen található egy példaprogram.

Még annyit kellene hozzáfűzni, hogy egy. a rendszer által is ha&znált Alert is létezik. Ez egy másik függvény, mégpedig az exec library-ban. Ezzel a függvénnyel csak a hiba kódjára utaló számot vagyunk képesek kiíratni. A függvény neve az **AlertfJ,** és egyetlen paramétert vár. mégpedig egy long típusú számot a hiba kódjának megfelelően. Használatára akkor lehet szükségünk, ha például assemblyben programozunk, vagy olyan program írásakor amellyel nem nyitjuk nieg az Intuition-t. így a hibát vele írathatjuk ki. Maga az Alert külsőre megfelel a normál GURU Alert-eknek (Piros szín, DEADEND).

1.8.3 A Requester-ek

A Requester-eket akkor használjuk, ha a kiváltó tényező nem haladja meg az Alert-ek mértékét, de nagyobb mint az egyszerű figyelmeztetés. Természetesen a DisplayBeep-et használhatjuk a requester-ünk megjelenésekor figyelmeztetésre is. A requester tulajdonképpen egy ablak, amelyhez gadget-ek is tartozhatnak. Azzal a könnyebbséggel számolhatunk használatakor, hogy nem kell egy komplett NewWindow struktúrával számolnunk, és a Gadget-ek kezelésével sem. Természetesen komplikálhatjuk a Requester-ünket olyan mértékben is, hogy akár adatbevitelre is alkalmassá váljon. Összesen három fajta requesler-t különböztetünk meg. Az elsók a System Requester-ek. Ezeket - mint a neve is mutatja - a rendszer üzenetek küldésére használja (pl.: Helyezz lemezt a meghajtóba). A második csoport az Application Requester-ek. Ezt a fajta Requester-t használhatjuk mi is saját programjainkban, az üzenetek küldésére.

A harmadik a DuplaMenü requester. Ezt a requestert csak a jobb gomb kétszeri megnyomásával aktivizálhatjuk.

1.8.3.1 A System Requester

A System Requester-t az Intuitíon nyitja és vezérli, nincs beleszólásunk az intézésébe. Csak annyiban, hogy ha például DF1 :-re hivatkozunk, mondjuk fileművelet végzése során (írnánk vagy olvasnánk onnan), a rendszer küld egy ilyen rqeuester-t ha nincs lemez a meghajtóban, vagy nem az a lemez található benne, vagy nyomtatáskor a nyomtatóból kifogyott a papír. stb.

Az egyetlen fontos dolog, amit megjegyezhetünk csak annyi, hogy a System Reqeuster-t a többi ablak, vagy requesler elé. ill. mögé helyezhetjük, átméretezhetjük (csak 2.0-ás rendszertől lefelé), addig az őt küldő taszk áll és várakozik a requester-re adott válaszra, így akár kihasználva a multilaszk előnyeit, nyugodtan akár ki is tömöríthetjük a kért lemezt arra a meghajtóra, majd a Retry választ adva a program tovább futhat.

1.8.3.2 A Felhasználói Requester

Ezeket a Requester-eket használhatjuk a saját programjainban is. Ha csak annyira van szükségünk, hogy egy kérdésre vagy -ről eldöntse az user. hogy igaz. vagy hamis, illetve hogy Igen vagy Nem. akkor az **AutoRequestfJ** függvényt kell meghívnunk. Amennyiben a kérdés komplikáltabb, vagy adat bevitel is szükséges akkor a **RequestfJ** függvényt kell használnunk.

Fontos azonban megjegyeznünk, hogy az application Requester-ekel nem kezelhetjük olyan szabadon mint a rendszeréit. Nem méretezhetjük at, nem mozgathatjuk (csak 2.0 alatt. Felette a két requester teljesen egyforma, mármint az AuloRequestO általl nyitott). A két függvény közötti különbség csupán abban áll. hogy az egyszerű requesier-nez (Auume-qucsiQ) nem Kup =,«I ható gadget. Végül is mind a két requesler-hez lehet gadget-et is kapcsol-

nunk csak az egyszerű requester esetében ez kimerül két gadget kereteiben amelyeknek csak a mérete és a tartalmazandó szöveg mikéntje adható meg. míg a másik esetben szabadon definiálhatunk gadget-et. A Requester-ek kinézetére két eset van. Az egyik eset az egyszerűbb amikor az Intuiüon intéz mindent, nekünk csak azt kell közölnünk, hogy hogyan is nézzen ki. A másik esetben saját BitMap-ot kell használnunk és nekünk kell intéznünk mindent, az allokálástól kezdve. Az első esetben is közölhetjük vele az óhajtott színekkel border-t rajzoltathatunk és szövegeket írhatunk ki az IntuiText segítségével. Ha azonban saját BitMap-ot használunk és nekünk kell megadnunk mindent, akkor akár használhatunk olyan Border-t. Itext-et, vagy akár Image-t is. amilyet akarunk, az Intuition-t nem igazán fogja zavarni. A requester-eink kétféleképpen nyerhetik el a pozíciójukat a képernyőn. Az első esetben az ablakunkhoz lesznek relatívak a másik esetben az egér pointeréhez relatívan nyílhatnak meg. Végül még néhány szó a requester gadget-jeiről. A requester-ben használt gadget-eknek be kell állítani a GadgetType mezőjében a G1YP_REQGADGET flag-ot. Valamint a requeser-hez csatlakozó gadget-ek közül az egyikben biztosan be kell állítanunk a GACT_ENDGADGET flag-ot. hogy a reqester-ből vissza lehessen térni.

Ne felejtsük el. hogy a requester ablakában a gadget-eknek pozíciójuk van. Ez abban nyilvánul meg, hogy a kilépő gadget (CANCEL, RESUME) mindig a jobb oldalon található. Ebből következően az Igen választ adó gadget-ek (YES. OK. TRUE. RETRY) mindig a bal oldalon találhatóak. Fontos megjegyeznünk, hogy az Intuiüon nem ellenőrzi a gadget-ek tartalmát, ö csak a fenti szabálynak megfelelően kezeli azokat.

1.8.3.2.1 Az egyszerű (Simple) Requester

Mint már említettük egyszerű Requester-eket az olyan problémák megoldására használhatunk, ahol két dolgot kell eldöntenie az user-nek. Tehát a válasz Igen és Nem lehet csak. Ehhez az **AutoRequestfJ** függvényt használhatjuk. A függvény az Intuition-ban található mint minden requester-rel kapcsolatos dolog. Ebben az esetben a függvényen keresztül csak néhány dolgot kell megadnunk az Intuition-nak. például a kiírandó szöveget, annak kinézetét, valamint a pozitív válasz szövegét és a negatív válaszét. Ebből az Intuitinon megjeleníti, és a választól függő értékkel tér vissza. Ha az user az igenlő választ adta akkor a vjsszatérési értek a TRUE lesz. valamint ha a nemleges választ akkor FALSE. A függvényre lássunk egy példát:

```
result = RutoRequest (ablak, ínfo_teHt, igen_teHt, nem_teHt,
                      igen_IDCMP, nem_IDCMP, szélesség,
                      magasság);
```

Ahol a paraméterek jelentése a következő:

ablak: pointer az ablakunkra, ahová a requester tartozik. Ha nincs ilyen, akkor NULL.
info_text: az user-nek szánt tájékoztató szövegünk, amely természetesen intuiText formátumú és ide annak pointere kerül ("taody text").
igen_text: az igenlő választ tartalmazó szöveg, amely az Igenlő gadget-be kerül majd.
nem_text: a nemleges választ tartalma/ó szöveg, amely a Nem gadget-be kerül majd. Mind a kettő IntuiText pointer kell legyen.
igenJDCMP: az igen gadget IDCMP-je. A RELVERYFY mindig beállított.
nemJDCMP: a Nem gadget IDCMP-je. A RELVERIFY mindig beállított.
szélesség: hány pixel széles legyen a requester.
magasság: hány pixel magas legyen a requester.
result: a visszatérési érték. Ha az Igen gadget-et aktivizálták TRUE. ha a Nem gadget-et aktivizálták, akkor FALSE.

1.8.3.2.2 A Requester

Ha némiképpen komplikáltabb requester-re van szükségünk, mint a fentebb említett egyszerű requester akkor a következő utat kell végigjárnunk.
Először is a reqester struktúrát kell deklarálnunk és inicializálnunk, majd a requester-hen használni kívánt gadget-ekét, és végül meghívni a függvényt.
Ez a függvény lehet a **RequestfJ** függvény, vagy a kétszer megnyomott jobb egérgombra aktivizálódó DMrequester-t megnyitó **SetDMRequestO** függvény is.
Tehát a használatához meg kell ismernünk a Requester struktúrát Ez az alábbiakban tekinthető meg.

```
struct Requester
{
    struct Requester *OlderRequest;
    SHORT LeftEdge, TopEdge;
    SHORT UJidth, Height;
    SHORT RelLeft, RelTop;
    struct Gadget *ReqGadget;
    struct Bordér *ReqBorder;
    struct IntuiText *ReqTent;
    USHORTFlags;
    UBVTE BackFMI;
    struct Layer *ReqLayer;
    UBVTE ReqPad[32];
    struct MitMap *ImageBMap;
    struct UJindoiu *RWindouj;
   struct Image *ReqImage; /* Csak U36! esetében létezik */
    UBVTE ReqPad2[36];
};
```

A mezők jelentése:

OlderRequest: az IntuiLion használja. így nekünk NULL-t kell megadnunk.
LeftEdge, TopEdge: a Requesl relalív pozíriója az ablak bal felső sarkához, ha a PO1NTREL flag nincs beállítva.
Width, Height: a Requester nagysága.
RelLeft, RelHeight: a Requester pozíciója az egér pointeréhez képest, ha a PINTREL flag beállított.
ReqGadget: pointer az első gadget-ünkre a listában, amit a Requester -hez akarunk csatolni. Ne feledkezzünk meg arról, hogy az egyik gadget-nél az ENDGADGET flag-ot beállítsuk.
ReqBorder: pointer az első Border-ünkre. amit a Requester-be szeretnénk rajzoltatni.
ReqText: pointer a Requester-ben kiírandó szöveg(eink)ünkre.
Flags: a beállítható flag-ek a következők:

POINTREL: a Requester pozíciója relatív legyen az egér pointer-éhez képest. Azt. hogy mennyivel, a/t a RelLeft és a RelTop-ban állíthatjuk be.
PREDRAWN: ha saját BitMap-ot használunk, ezzel a flag-gel jelezzük az Intuition-nak, hogy ne rajzoljon semmit.
SIMPLEREQ: Ezzel a flag-gel jelezzük a Layer számára. hogy a Requester-ünk Simplerefresh-t használ.
NOISREQ: ha nem akarjuk, hogy a Requester-ünk szűrőt alkalmazzon a bemeneten.
USEREQIMAGE: Az Image-ünket kiteszi mielőtt a gadget-eket és az IntuiText-et. de a háttér színezése után (V36!-tól).
NOREQBACKFILL: Requesler-ünk mozgatásakor a backfill, pen nem fog zavarni benünket.

BackPill: annak a színregiszternek a számát várja, amellyel a hátteret kívánjuk kiszínezni, még mielőtt a szövegek vagy a gadget-ek megjelennének.
ReqLayer: pointer a Requester Layer struktúrájára. Az Intuition inicializálja, így nekünk ide NULL-t kell írnunk.
ReqPadl: az Intuition használja, állítsuk NULL-ra.
ImageMap: ha a PREDRAWN flag beállított akkor az itt megadott mutató alapján a mi általunk létrehozott és inicializált BitMap-ot fogja használni, egyébként NULL.
RWindow: az Intuition használja így NULL-nak adjuk meg.
Reqlmage: csak V36-os vagy magasabb [ntuition esetén adhatjuk meg, alatta nem szerepel a struktúrában. Ha a USEREQIMAGE flag beállított, ide várja az Image struktúránk mutatóját.
ReqPad2: az Intuition használja. így állítsuk NULL-ra.

Példaképpen inicializáljunk egy Reqester struktúrát.

```
struct Requester a_reqesterunk =
{
      NULL,     /* OldRequester, az Intuition használja. */
      40, 20,  /* LeftEdge, TopEdge, 40 pixelre és 20 sorral lejjebb. */
      320,100,/* UJidth, Height, 320 piHel széles, 100 sor magas. */
      0,0,      /* RelLeft, RelTop, H PINTREL flag nem beállított így */
                /* az Intuition átugorja ezt. */
      &elso_gadgetunk,/*ReqGadget, Pointer az első gadget-ünkre. */
      C-Borderunk,      /*ReqBorder, Pointer a Border-ünkre. */
      &TeKtunk,         /*ReqTeHt, Pointer a kiírást tartalmazó. */
                        /•szöuegünkre. •/
      NULL,     /* Flags, nem használunk. */
      3,        /* BackFill, a hátterünk narancs színű lesz (1.3-on). */
      NULL,     /* ReyLayer, az Intuition használja. */
      NULL,     /• ReqPadi, az Intuition használja. */
      NULL,     /* ImageBMap, nem használjuk. •/
      NULL,     /* RLUindoiD, az Intuition használja. */
      /* Ha U36-os uagy magasabb gépünk uan akkor még egy NULL */
      NULL      /* ReqPad2, az Intuition használja. •/
};
```

Hogyan használjuk akkor hát ezt a struktúrát? Nos, nagyon egyszerűen. A Requester-ünket a **RequestQ** függvény meghívásával érhetjük el. Vagy ha a Dupla Menü Requester-t akarjuk használni akkor a **SetDMRequsetQ,** függvény meghívásával. Mind a két függvény két paramétert vár, az első a pointer az inicializált Requester struktúránkra míg a második szintén pointer, de az ablakra mutató pointer, amihez a Requester csatlakozik. Mindkét függvény esetében a visszatérési érték ha sikerült a Requester-t megnyitnia TRUE. míg ha nem akkor FALSE. A Requester-ek megszüntetését nekünk kell kezdeményeznünk -a DMrequst esetében. Ezt a **ClearDMRequest** segítségével végezhetjük.

Paraméterként a csak az ablakunkra mutató pointer-t várja. Visszatérési értéknek akkor ad TRUE értéket ha az user aktivizálta azt és FALSE-t ha nem.

1.8.3.3 A Requester-ek IDCMP-i

Minden reqeuster müvelet az ablakunkon keresztül zajlik le. Ezért a Requesterünk-kel történt dolgokat is az ablakon keresztül tudhatjuk meg. mégpedig az IDCMP csatornán. Az lDCMP csatornának a Requester-re három féle jelzést küldhet az ablakunk felé. Az első akkor érkezhet ha a reqester aktívvá vált. Ehhez az iDCMP_REQSET flag-ot kell beállítanunk a NewWindow otrukinránkban: Ha arról akarunk üzeneteteket kapni amikor a Requester-ünket deaktivizálták. akkor az IDCMP_KEgci>ti.AK ruig-ot M I I,c<Ulitanunk.

Ezenkívül létezik még egy speciális flag. arra az esetre ha az user megpróbálja aktivizálni a DMrequester-ünket. Ez az üzenet akkor érkezik amikor még a Requester nem nyílt meg. Tehát addig nem is nyílik meg ameddig a programunk nem válaszol a **ReplyMsgO** függvénnyel az Intuition-nak. így a programunk nyugodtan befejezheti az elkezdett rajzot és egyéb tevékenységet. Ha tehát megjelenik a Requester a program már végzett a feladatával.

1.8.3.3 Az újabb Intuition Requester-jei

A V36-os Intuition-tól felfelé az íntuition-ban található a régi Requester-ek mellett egy újabb struktúra, és az erre épülő függvények. Az újabb struktúra segítségével a Requester megadás sokkal egyszerűbb lehet néhány esetben.

Természetesen, ha nem a rendszeréhez hasonlót akarunk csinálni akkor nem ússzuk meg a mindenféle inicializálgatásokat. Viszont lehetőség van immár a Requester-ben Image kitevésre is. nem csak a háttér színének automatikus fillezésére. Ezt az előbbi Requester struktúrában láthattuk is. A könnyebbség az új struktúra használatával adódhat. Nevezetesen nem kell gadget struktúrákat definiálgatnunk sem inicializálgatnunk. A Requester-be kiírandó szövegeket a C-ből ismert printf formátumban adhatjuk meg. Akkor nézzük azt a struktúrát amivel ez mind elérhető:

```
struct EasyStruct
{
    ULONG es_StructSize;
    ULONG es_Flags;
    UBVTE *es_Title;
    UBVTE *es_TentFormat;
    UBVTE *es_GadgetFormat;
};
```

A struktúra mezőj az alábbiak szerint alakulnak. Az **es_StructSize-val** a struktúra méretét kell megadni. Itt többnyire a **sizeof(a struktúránk)** sorral feltölthető. Az **es_Flags** egyenlőre nem megadható. Talán egy későbbi Intuition esetében lesz majd. Az **es_Title-vel** a Requester ablak fejlécét adhatjuk itt meg. Az **es_TextFonnat** a Requester-ben megjelenő információs szöveg, amely tartalmazhat a printf-ben megszokott konverziós karaktert (pl. %d a decimális számok részére, %S a string részére). A konverziós karakterek helyén természetesen az argumentumként átadott paraméterek kerülnek oda. és úgy ahogyan azt a konverziós karakter megadja. Természetesen erről nem a struktúra gondoskodik, hanem az őt felhasználó függvények. Az **es_GadgetFormat-tel** meghatározhatjuk a Requester-en megjelenő gadgetekben szereplő kiírásokat, amelyek meghatározzák a gadget-ek számát is. Nevezetesen amennyi gadgetbe kerülő szöveg van annyi gadget lesz. A gadget-ek neveit a " I" karakterrel választhatjuk el. A gadget-ekbe kerülő szövegben szerepelhet, vagy akár teljesen lehet az egész a konverziós karakterek által megadott argumentum is (lásd a lemezen a Requester8.c példaprogramot).

A Requester-ek

A struktúra önmagában persze csak lehetőség, ezt azonban ki tudjuk használni az Intuition által rendelkezésünkre bocsájtott függvények segítségével. Az egyik ilyen függvény az **EasyRequesterfJ** a másik pedig a **BuildEasyRequesterQ.** A függvények paramétereként megadható a meghívó program által létrehozott ablak pointer-e. Második argumentumként az inicializált EasyStruct pointer-ünket adhatjuk meg. A harmadik paraméterként az **EasyRequestfJ** függvény esetében egy pointer-t vár az IDCMP flag-okat tartalmazó változónkra (ULONG *). A másik függvénynél ezeket nem pointerként várja. A megadott IDCMP-nek megfelelően, amikor a kívánt esemény létrejött a Requester visszatér a meghívó programba anélkül, hogy az user hozzáért volna az egérhez (mint a rendszer Requester-jei).

Valamint megadhatóak további argumentumok amelyek a/ EasyStruct konverziós karakterei által mutatott helyre és formában kerülnek ki a Requester-ünkre.

Az EasyRequest visszatérési értéknek a lenyomott gadget számával tér vissza. A O-ás számú a jobb oldali gadget. majd balról az első lesz az első számú és így jobbra haladva (Ne feledkezzünk meg. hogy a pozitív válasz mindig a bal oldalon helyezkedik el. míg a negatív a jobb oldalon!).

A Build Requester annyiban jelent más megoldást, hogy visszatérési értékként egy Requester ablakot megcímző pointer-t kapunk. Ezen kívül van még egy pár függvény ami ezekkel kapcsolatban segiségünkre lehet.

Részletesebben a lásd az Intuition library leírását, és természetesen konkrét példákkal a lemezmellékletet.

1.9 Alacsony szintű grafika

1.9.1 Az alapok

Ebben a részben megpróbáljuk megmutatni, hogy a legegyszerűbb grafikai rutinokkal milyen könnyű grafikát kicsalni az Amigából. Nem sok olyan gép van. amely alapból (ROM-ból!) támogatja például a fillezett grafikát, pedig az Amiga már '85-ben megtette ezt.

Sorban bemutatjuk, hogy hogyan kell az egyes könyvtárrutinókat használni, de nem mindet, mert erre az egész könyv sem lenne elég (akkor mi maradna a könyv következő részébe?!), de azért annyit, hogy mindenki el tudjon indulni és a graphics library leírásával még többet is meg tudjon csinálni.

Először ejtsünk néhány szól arról, hogy az Amiga hogyan tárolja a képet a memóriában. Ez a kép csak a CHIP RAM valamely részén lehet, az elektronika csak innen tudja kiolvasni. Ennek a képnek BitPlane szervezésűnek kell lenni. Ezt úgy kell elképzelni, mintha több üveglapot raknánk egymásra. Az ECS chipset-ig maximum 6 ilyen "üveglap" vagyis BitPlane volt, az AGA gépeknél már 8. A bitplane-ek számától a színek száma függ. Erre vegyünk egy egyszerű példát. Legyen 2 darab bitplane-ünk.

Az első van hátul és a második elöl. Ekkor ha egyikre sem rajzolunk akkor kapjuk a O-ás színt. Ha az elsőre kirakunk egy pontot akkor az l-es paletta színével jelenik meg egy pont, ha" a másodikra, akkor a 2-es színnel. Ha mindkettőre ugyanarra a helyre rakunk egy pontot (vagyis fedik egymást) akkor kapjuk meg a 3-as színt. Ezzel akkor két bitplane-re négy különböző színnel tudunk rajzolni. Ha három bitplane-t nyitunk meg akkor már nyolc színünk lesz, nyolc bitplane esetén pedig 256 különböző színt használhatunk.

Példa 3 BitPlane használatára:

3 BitPlane	2 BitPlane	1 BitPlane	Színregiszter
0	**0**	**0**	0 (háttérszín)
0	**0**	**1**	1
0	**1**	**0**	2
0	**1**	**1**	3
1	**0**	**0**	4
1	**0**	**1**	5
1	**1**	**0**	6
1	**1**	**1**	7

Ha a graphics library rutinjait használjuk akkor ilyen dolgokkal nekünk nem kell foglalkozni, mert a rutinok elintézik ezt nekünk. Lehetőségünk van viszont arra. hogy ezekre a bitplane-ekre külön írjunk és akkor ezek lehetőségeit ki is tudjuk használni (most ezekkel nem foglalkozunk).

Létezik még néhány olyan üzemmód is, ahol a fent leírtak nem igazán teljesülnek. Az egyik ilyen mód az **ExtraHalfBright,** az extra félfényerő.

Ez a legelső OSC-nél volt egy kiegészítő hasznos funkció, amikor is 6 bitplane volt és csak 32 színt lehetett beállítani, a másik 32-t a hardver képezte az első 32-ből úgy, hogy azoknak a félfényerejű változatát vette. Ezt az ESC és AGA chip-set is tartalmazza, de már nem nagyon használja senki, főleg nem az operációs rendszer. A második ilyen lehetőség a DoublePlayfield. Ez OSC és ESC rendszernél két darab nyolc színű képet jelentett egymás mögött, egymástól teljesen függetlenül. Az egyik kép az 1,3.5 számú BitPlane-eket használta, a másik a 2.4,6 számúakat. Külön-külön lehetett ezeket scrollozni. külön színeket lehetett hozzárendelni.

Az AGA chip-set már két darab 16 színű képet tud kezelni, de egyébként nem sok értelme van.

A harmadik ilyen üzemmód a **HAM** volt. Ezt is az OCS-hez lett kifejlesztve azért, hogy 4096 színt egyszerre meg lehessen megjeleníteni kisebb megkötésekkel. Ez annak idején még megfelelt a kor elvárásainak, sőt túl is lőtt a célon, de ma már nagyon kevés az amit nyújt. Ennek utódjaként az AGA chip-set hozta a HAM8 üzemmódot, mely 262144 szín egyidejű megjelenítésére alkalmas. Ez már tényleg nagyon jó minőségű képet tud produkálni, főleg, hogy bármilyen felbontásban képes erre, míg az OSC és ESC chip-készletnél voltak komoly megkötések.

Visszatérve tehát, amikor megnyitunk egy screen-t. meg kell adnunk annak a mélységét (Depth), ez jelenti a BitPlane-ek számát. Ehhez a képernyőhöz tartozik egy ViewPort és egy RastPort struktúra. Egy screen-hez csak egy darab ViewPort struktúra tartozik, az ablaknak nincs ViewPort-ja. Ez mondja meg, hogy a Copper hogyan állítsa elő a képet, milyen .színeket használjon, a sprite prioritásokat, a screen módját, stb. Ezért van az, hogy a színbeállító (SetRGB4, SetRGB32. stb.) rutinok egy ViewPort-ot használnak és egy screen-en ha átállítunk egy színt, akkor az minden azon a screen-en levő ablakban minden színt is megváltoztat. ViewPort-ja tehát csak screen-nek van. Most lássuk magát a ViewPort struktúrát:

```
strii'ct UieiuPort
{
    struc   UieuiPort   *Nent;
    struct  ColorMap    •ColorMap;  /• színtáblázat ehhez a
                                       UieuiPort-hoz */
    struct  CopList     *DspIns;    /* a MakeUPortO használja */
    struct  CopList     *SprIns;    /* sprite-okhoz */
    struct  CopList     *ClrIns;    /* sprite-okhoz */
    strúct  UCopList    *UCopIns;   /* felhasználói copper lista */
    WORD    DUJidth.DHeight;
    WORD    DxOffset,DyOffset;
    UWORD   Modes;
    UBVTE   SDritePrioríties;
    UBVTE   EHtendedModes;
    struct  RasInfo *RasInfo;
};
```

RastPort-ja viszont van screen-nek is és window-nak is. A RastPort tartalmazza azokat az adatokat, melyekből kiderül, hogy a memóriában hol vannak a BitPlane-ek. hova lehet rajzolni, mi az aktuális rajzoló szín (pen). milyen betűkészletet kell használni, stb. Tehát ha minden screen-nek van RastPort-ja akkor arra is lehet rajzolni, nem kell nyitni rá ablakot. Ez viszont azt is jelenti, hogy IDCMP-ket nem tudunk honnan lekérdezni, vagyis mégis jó lenne rá egy ablak. A legegyszerűbb megoldás az, hogy nyitunk rá egy ablakot úgy, hogy keret nélküli, nem mozgatható, vagyis nem is látszik. Az user nem is tudja, hogy van ablak. Ezután nyugodtan lehet rajzolni a screen-re és IDCMP-ket is kapunk.

íme a RastPort struktúra:

```
struct RastPort
{
    struct  Layer *Layer;
    struct  BitMap *BitMap;
    UUJORD  *RreaPtrn;          /* mutató az areafill pattern-nek */
    struct  TmpRas *TmpRas;     /* mutató a TmpRas struktúrára az
                                   Fiiled Rrea-khoz */
    struct  fIrealnfo *fIrealnfo;
    struct  GelsInfo *GelsInfo;
    UBVTE   Mask;
    BVTE    FgPen;              /* elsődleges ceruza szín */
    BVTE    BgPen;              /* háttér ceruza szín */
    BVTE    ROlPen;             /* areafill outline szín */
    BVTE    DraujMode;          /* rajzolási mód a fill, lines, and text
                                   rutinokhoz: JRM1, JRM2, stb. */
    BVTE    RreaPtSz;
    BVTE    Mnpatcnt;
    BVTE    dummy;
    UUJORD  Flags;
    ULUORD  LinePtrn;           /* 16 bit a textúrázott uonalakhoz */
    WORD    cp_x, cp_u;         /* a grafikus kurzor helye */
    UBVTE   minterms[8];
    WORD    PenUJidth;
    UJORD   PenHeight;
    struct  TextFont *Font;     /* a használt betűkészlet címe */
    UBVTE   HlgoStyle;
    UBVTE   TKFlags;            /* text specific flags */
    UUJORD  TxHeight;           /* text height */
    U UJORD TKUJidth;           /* text nominal ujidth */
    UUJORD  TxBaseline;         /* text baseline */
    UJORD   TxSpacing;          /* text spacing (per character) */
    RPTR    *RP_User;
    ULONG   longreserued[2J;
~ #ifndef GFH_RHSTPORT_1_2
    UUORD ujordreserued[7];
    UBVTE  reserued[8];         /* fejlesztésre fenntartua */
#endif
};
```

Még ejtünk néhány szót a BitMap struktúráról is. Erre a RastPort struktúrában van egy mutató, innen tudjuk kiszedni, hogy a tényleges BitPlane-ek hol kezdődnek a memóriában.

A BitMap struktúra:

```
struct BitMap {
    UWORD    BytesPerRoui;
    UIDORD   Rows;
    UBVTE    Flags;
    UBVTE    Depth;          /* a BitPlane-ek száma */
    UWORO    pad;
    PLRNEPTR Planes[8];      /* mutatók a BitPlane-ekre
                                (a tényleges címek) */
};
```

A kép előállításához szorosan hozzátartozik a színek keverése. Az Amiga minden színt három alapösszetevőből kever ki. a pirosból, a zöldből és a kékből. Ezek aránya határozza meg a tényleges színt.

Ha azt mondjuk, hogy a rajzolási szín vagyis a ceruza szín kettő (majd később lesz a **SetAPen(rp,2)).** akkor ez azt jelenti, hogy a második színnel kell majd rajzolni. Ez még nem határozza meg azt, hogy az milyen színű legyen.

Azt, hogy ez sárga, vagy zöld, vagy valami más, azt külön hozzá kell rendelni a kettőhöz. Például a **SetRGB4(2,0,15,0)** azt jeleni, hogy a kettes színben a piros összetevő nulla (vagyis nincs), a zöld összetevő 15 (maximális) és a kék összetevő is nulla. Ennek eredménye az. hogy a kettes szín zöld lesz.

A táblázat néhány szín keverési arányát mutatja:

Piros	Zöld	Kék	Szín
0	0	0	Fekete
15	0	0	Piros
0	15	0	Zöld
0	0	15	Kék
15	15	0	Sárga
15	0	15	Lila
0	15	**15**	Cián kék
15	15	15	Fehér
8	8	8	Szürke
9	7	5	Barna
5	7	9	Pasztell kék
15	0	10	Ciklámen

Most. hogy meg vannak az alapok, először nyitunk egy screen-t. Megkeressük a RastPort és ViewPort struktúráját, mert ezek általában minden Graphics Library rutinhoz kellenek. Ezeket a sereen struktúrában találjuk meg úgy, hogy be vannak ágyazva, vagyis nem egy-egy mutató mutat a tényleges struktúrára.

Lássuk az assembly kódot:

```
                incdir    "ujork:sc/include/"

                include   "work:luo3.B/execJib.i"
                include   "work:luo3.0/Intuition_lib.i"
                include   "ujork:luo3.B/dos_lib.i"
                include   "ujork:luo3.0/graphicsjib.i"
                include   "Intuition/Intuition.i"
                include   "dos/dos.i"
                include   "graphics/gfH.i"

Start:          moue.l    $4.UJ,a6
                moueq     #0,dB
                moue.l    #IntName,a1
                jsr       _LU00penLibrary(a6)
                tst.l     dO
                beq.w     End
                moue.l    dOJntBase

                moue.l    $4.UJ,a6
                moueq     #0,d0
                moue.l    #DosName,a1
                jsr       _LU00penLibrarg(a6)
                tst.l     dO
                beq.w     Close_Intuition
                moue.l    dü,DosBase

                moue.l    $4.uj,a6
                moueq     #O,dO
                moue.l    #GfnName,a1
                jsr       __LU00penLibrary(a6)
                tst.l     dO
                beq.w     Closejos
                moue.l    dB,GfxBase

                moue.l    IntBase.aö
                moue.l    #0,a0
                moue.l    #screen_tags,a1
                jsr       _LU00penScreenTagList(a6)
                tst.l     dO
                beq.ui    Close_GfK
                moue.l    dfl,my_screen

                moue.l    my_screen,dO      ;kiszedjűk a Uieui
                                            Port-ot
                add.l     #44,dB
                moue.l    dO,up
```

```
                moue.l   my_screen,dO   ;kiszedjüka RastPorl-Dt
                add.l    #84,dO
                moue.l   dO.rp

;-------------------------

; ide jön majd a saját programunk

;-------------------------

                moue.l   DosBase,a6
                moue.l   #50*5,dl
                jsr      _LU0Delay(a6)  ;uárunk 5 másodpercet

Close_LUindouj: moue.l   IntBase,a6
                moue.l   my_screen,aO
                jsr      _LU0CloseScreen(a6)

Close_GfK:      moue.l   $4.uj,a6
                moue.l   GfxBase,al
                jsr      _LU0CloseLibrary(a6)

CloseJDos:      moue.l   $4.uj,a6
                moue.l   DosBase,al
                jsr  _LU0CloseLibrary(a6)

Close_Intuition : moue.l $4.uj,a6
                moue.l   IntBase,al
                jsr      _LU0CloseLibrary(a6)
end:            rts

screen_tags:    de.l     SRJDidth, 320
                de.l     SR_Height,28Q
                de.l     SH_0epth,2
                de.l     SR_Pens, pens
                de.l     SR_ Type.CUSTOMSCREEN
                de.l     SR_Quiet.1
                de.l     TRG_DONE

pens:           dc.uj    -1

my_screen:      —de.l    0

DosBase:        de.l     0
IntBase:        de.l     e
GfxBase:        de.l     0
rp:             de.l     e
up:             de.l     0
```

```
IntName:        dc.b    "Intuitíon.library",Q
DosName:        dc.b    "dos.library",0
GfxName:        dc.b    "graphics.líbrary",0
```

A Structure.olTsets file-ból szedtük ki, hogy miért épp 44 a ViewPort és 84 a RastPort struktúra offsel-je. C-ben ez egy kicsit könnyebb, ott nem kell ilyen számok után kutatnunk, íme:

```
#include <enec/types.h>
#ínclude < Intuítion/Intuítion.h>
#ínctude <Intuition/screens.h>
#include <clíb/dos_protos.h>
#include <clíb/enec_protos.h>
#include <clib/Intuition_protos.h>
#ínclude <clib/yraphícs_protos.h> #include <stdio.h>

struct GfxBase *Gf»Base;
struct IntuitionBase *IntuitionBase;
struct Screen *my_screen;
struct RastPort *rp;
struct UieuiPort *up;

UORD Pens[]={~0};

uoid main(uoíd)
{
  IntuitionBase=(struct IntuitionBase *)OpenLibrary("Intuition.library", OL);
  if (IntuitionBase)
  {
    GfnBase=(struct Gf»Base *)OpenLibrary("graphics.library",OL);
    if (GfKBase)
    {
       my_screen = OpenScreenTags(NULL,
                   SR_U)idth, 320,
                   Sfl_Height,2B0,
                   SR_Depth,2,
                   SR_Pens, &Pens[8],
                   SR_Quiet,TRUE,
                   Sfl_Type,CUSTOMSCREEN,
                   TR6_D0NE);
       if (my_screen)
       {
          up=&my_screen->UieujPort;
          rp=&my_screen->RastPort;

          /* a későbbi programok ide jönnek •/

          Delay(50*5);
```

```
        CloseScreen(my_screen);
    }
    CloseLíbraryüstruct Library *)GfxBase);
  }
  CloseLibrary((struct Library *)IntuitionBase);
 }
}
```

Ezzel az alapok megvannak. Nyitottunk egy képernyőt, melynek nincs fejléce (egy kicsit zavarónak találtam) és teljesen üres.

Ahhoz, hogy rajzoljunk is valamit, pixeleket kell kiraknunk ugyebár.

A legelső ilyen Graphics Library rutin amivel megismerkedünk, a **WritePixel.** Ha assembly oldalról nézzük, akkor ennek az Al regiszterbe kell egy RastPort, a DO regiszterbe egy X koordináta és a Dl regiszterbe pedig egy Y koordináta. Legyen! Az assembly forrásban van egy hely, ahova a programjaink kerülnek, oda írjuk be a következő néhány sort:

```
moue.l  GfxBase,a6
moue.l  rp.ai
moue.l  #30,dO
moue.l #30, dl
jsr     _LU0ùJritePixel(a6)
```

Ezzel a (30.30) koordinátára ki is raktunk egy pontot. C nyelvű forrásba is beszúrhatunk egy sort. itt egy ici-picit könnyebb a dolgunk:

```
UJrítePixel(rp,30,30);
```

Lehetőségünk van lekérdezni, hogy egy képpont milyen színű. Ezt a **ReadPixel** rutin végzi el, melynek Al-be egy RastPort kell, DO és Dl pedig a pont koordinátáit határozza meg. A rutin DO-ba adja vissza a színregiszter számát.

```
moue.l  GfxBase,a6
moue.l  rp.ai
moue.l  #30,dO
moue.l  #30,dl
jsr     _LU0ReadPixel(a6)
```

C nyeluen:

```
pen=ReadPixel(rp,3Q,3B);
```

Most. hogy már tudunk pixelt kirakni, meg is kellene határozni annak a színét. Erre a **SetAPen** rutin szolgál, mely csak azt mondja meg, hogy melyik „t eruzát" használjuk, nem a/t, hogy az milyen szinti (ezt adja vissza a ReadPixel) . Ennek is egy RastPort kell az Al regiszterbe és egy paletla szám a DO regiszterbe. Ez a sszám értelemszerűen csak O-tól a max. egyszerre

használható színek számáig terjedhet, például 3 mélységű képernyőnél O-tól 7-ig. Ha annál nagyobbat adunk meg. akkor sincs semmi baj, akkor is beállítja az annak megfelelő színt.

Létezik még egy **SetBPen** rutin is, mellyel a rajzolás hátterét tudjuk beállítani. A fillezésnél lesz komoly s/.erepe, így majd ott mutatunk használatára példát. A paraméterezése megegyezik a SetAPen rutinéval.

A következő példa a rajzolási színnek a hátteret állítja be (0), ha ezt beszúrjuk az előző példa elé akkor semmit sem fogunk látni a képernyőn, mert pont a háttér színével rajzol.

```
!       moue.l  GfxBase,a6
        moue.l  rp.ai
        moue.l  #0,d0
        jsr     _LU0SetflPen(a6)
```

C nyelven:

```
SetflPen(rp,O);
```

Most már itt az ideje, hogy a „ceruza" tényleges színét is beállítsuk.

Erre a **SetRGB4** és a **SetRGB32** rutin szolgál. A különbség az, hogy az első még a régi chipset-ekhez készült, csak 4096 szín közül válogathatunk, míg a SetRGB32 (csak AGA gépek!) 16777216 szín közül enged színt beállítani. Ezek AO-ba egy ViewPort-ot, DO-ba a paletta sorszámát, D1-D3 pedig az piros, zöld és kék összetevőt várják. A SetRGB32 az RGB színeket longword-ön balra igazítva várja, valahogy így: 0x12000000 vagy $12000000, állítsuk át a háttér színét feketére és az l-es ceruzát pirosra!

```
        moue.l  GfxBase,a6
        moue.l  up.aO
        moue.l  #Q,dO           ;háttér
        moue.l  #0,d1           ;piros
        moue.l  #B,d2           ;zöld
        moue.l  #Q,d3           ;kék
        jsr     _LU0SetRGB4(a6)

        moue.l  GfnBase,a6
        moue.l  up,aO
        moue.l  #1,dO           ;háttér
        moue.l  #15,d1          ;piros
        moue.l  #0,d2           ;zöld
        moue.l  #0,d3           ;kék
        jsr     _LU0SetRGB4(a6)
```

Vagy a SetRGB32-uel:

```
        moue.l  GfxBase,a6
        moue.l  up.aO
```

```
moue.l  #0,d0          ;háttér
moue.l  #Q,d1          ;piros
moue.l  #B,dZ          ;zöld
moue.l  #0,d3          ;kék
jsr     _LU0SetRGB32(a6)

moue.l  GfxBase,a6
moue.l  up.aB
moue.l  #1,dO          ;háttér
moue.l  #$ffBOOOOB,d1  ;piros
moue.l  #B,d2          ;zöld
moue.l  #B,d3          ;kék
jsr     _LU0SetRGB32(a6)
```

C nyelven:

```
SetRGB4(up,0,0,0,0);
SetRGB4(up,1,15,0,0);
```

A SetRGB32-vel:

```
SetRGB32(up, 0,0,0,0);
SetRGB32(up,1 ,BxffOQQQBB,O,O);
```

Most, hogy már ennyi mindent tudunk, csináljunk néhány szép ábrát! Az első 16 színű ferde csíkokat fog rajzolni. Ha az assembly program elsőre nem lenne világos, akkor nézzük meg a C forrást, az biztosan érthető! Tehát az assembly forrás:

```
              mouei   #0,d7
Color_again:  moue.l  GfxBase,a6
              moue.l  up,aO
              moue.l  d7,dfl
              moue.l  d7,d1
              moue.l  d7,d2
              moue.l  d7,d3
              jsr     _LU0SetRGB4(a6)
              add.l   #1,d7
              cmp     #16,d7
              bne.s   Color_again

              móueq   #0,d6

|9rauj_again2: mouec  #Q,d7

Draui_again:  moue.l  GfxBase,a6
              moue.l  rp.ai
```

```
moue.l   d7,dO
add.l    d6,dO
jsr      _LU0SetflPen(a6)

moue.l   GfxBase,a6
moue.l   rp.al
moue.l   d7,dO
moue.l   d6,d1
jsr      _LU0UJritePixel(a6)

add.l    #1,d7
cmp      #32fl,d7
bne.s    Drauj_again

add.l    #1,d6
cmp    #200,   d6
bne.s    Drauj_again2
```

Következzen a sokkal rövidebb és érthetőbb C forrás:

```
for (x=0;x<16;x++) SetRGB4(up,x,x,x,x);

for (y=O;y<20O;y++)
{
    for (x=B;x<320;x++)
    {
        SetRPen(rp,x+y);
        UJritePixel(rp,K,y);
    }
}
```

Láthatjuk, hogy itt már változókat is használunk, ezeket integer típusúnak kell definiálni, valahol a forrás elején (int x.y;).

Minden grafikus rutinhoz be lehet állítani a rajzolási módot, ezt a **SetDrMd** rutin végzi. Megadhatjuk az alábbiakat: JAM1. JAM2, COMPLEMENT és 1NVERSVID.

A rutin az Al regiszterbe egy RastPort-ot vár. és a DO-ba pedig a 8 biten megadott üzemmódot.

A **JAM1** azt jelenti, hogy csak az A pen-t használja, a háttért változatlanul hagyja. Ezt főleg szöveg kiírásakor látszik, ha szürke háttéren fekete szövegre ugyanazt a szöveget fehérrel akjuk rá egy pixellel odébb, akkor olyan lesz, mintha árnyéka lenne. így mindig csak egy színnel rajzol.

Ha ugyanezt **JAM2-es** módban próbáljuk megtenni, akkor a karakter üres részeit kitölti a B pen által meghatározott színnel. Ezt akor érdemes használni, ha felül akarunk írni egy szöveget, mert így az előzőleg kiírt szöveget nem kell letörölni. Ekkor mindig két színnel rajzol (persze ahol van értelme).

A **COMPLEMENT** mód a színregiszter számával végez egy komplementer müveletet.

Ez azt jelenti, hogy ha 4 színű képernyőnk van, és a háttérre rajzolunk, akkor a hármas színnel rajzol, ha l-es színű területre, akkor a 2-es színnel rajzol, ha 2-es színű területre, akkor az l-es színnel rajzol és végül ha 3-as színű területre, akkor a háttér színével rajzol.

Tehát teljesen mindegy, hogy a A és B pen mire van állítva. Ez arra jó, hogy ha kétszer ráengedjük egy területre, akkor ugyanazt kapjuk vissza (nézzük csak meg a op. rendszer kurzorát... az is így működik). A lemezmellékleten találhatunk erre egy szép példát, két ellipszisre engedi rá ezt az üzemmódot (LowLevel_ll.s). Az INVERSVID-et főleg a JAM1 és JAM2 módokkal együtt érdemes használni szöveg kiírására, máshol nem sok jelentősége van. A szöveget inverzben írja ki.

Menjünk tovább a rutinok ismertetésében. A következő a vonal rajzolás lesz, melyet a Draw rutin valósít meg. Ehhez el kell mondani, hogy létezik egy grafikus kurzor, mely nem látszik, csak egyszerűen van. A WritePixel ezt automatikusan oda állítja, ahová kirakja a pontot. A Draw ettől a ponttól kezdve rajzolja a vonalat a megadott végpontig, a SetAPen-el beállított színnel. A Draw az Al-be vár egy RastPort-ot, DO-ba és Dl-be pedig az X és Y koordinátákat. De ahhoz, hogy egy "vonalat rajzoljunk, a WritePixel túl extra lenne, hogy a grafikus kurzort átállítsuk, így arra is van egy külön rutin, a Move. Ez csak a grafikus kurzor mozgatására jó és a következőket kell neki megadni: Al-be egy RastPort-ot és DO,Dl-be pedig a koordinátákat. Ennek ismeretében már tudunk írni egy Line rutint, mely a megadott pontokat összeköti:

```
Line:   moue.l  GfKBase,a6
        moue.l  rp.ai
        jsr     _LU0Moue(a6)

        moue.l  GfnBase,a6
        moue.l  rp.ai
        moue.l   d2,dO
        moue.l   d3,d1
        jsr     _LU0Drauj(a6)
        rts
```

Ezt valahova az End címke után berakva, egy BSR utasítással meg tudjuk hívni szubriutinként az alábbi paraméterekkel:

```
DO - kezdőpont K koordinátája
Dl - kezdőpont V koordinátája
D2 - uégpont H koordinátája
D3 - uégpont V koordinátája
```

Példa:

```
moue.l  #0,d0
moue.l  #B,d1
moue.l  #319,d2
moue.l  *MQQ,d3
bsr.s   Line
```

így a bal felső sarkot összeköti a jobb alsó sarokkal.

A C nyelvű forráslista most egy kicsit hosszabb lesz, mert egyrész többet is csinál mint az előző assembly kód, másrészt az egészet leírjuk ide, mert túl sok már a változtatás benne:

```
#include <exec/types.h>
#include <Intuition/Intuition.h>
#include <Intuition/screens.h>
#include <clib/dos_protos.h>
#ínclude <clib/eHec_protos.h>
#include <clib/Intuition_protos.h>
#include <clib/graphics_protos.h>
#include <stdio.h>

/* Prototypes */

uoid Line(int,int,int,int);

/* Structures */

struct GfxBase *GfnBase;
struct IntuitionBase *IntuitionBase;
struct Screen *my_screen;
struct RastPort *rp;
struct UieujPort *up;

UWORD Pens[]={~0};

uoid Líneíint Hi,int yi.int K2,int y2) {
   Moue(rp,x1,y1);
   Draw(rp,x2,y2);
>
uoid main(uoid)
{
  int i;
IntuitionBase=(struct IntuitionBase *)OpenLíbrary("Intuition.library", QL);
  if (IntuitionBase)
   {
    GfxBase=(struct GfxBase *)OpenLibrary("graphics.library",QL);
    if (GfKBase)
    {
     my_screen = OpenScreenTags(NULL,
                    SR_UJidth, 320,
                    SH_Heíght,2G0,
                    SR„Depth,4,
                    SR_Pens, &Pens[0],
                    SR_Quiet,TRUE,
                    SR_Type,CUSTOMSCREEN,
                    TRG_BONE);
      if (my_screen)
```

```
    {
      up=Crny_screen->UieujPort;
      rp=&my_screen->RastPort;

      SetRGB4(up,0,0,0,0);      /* Fekete •/
      SetRGB4(up,1,15,15,15); /* Fehér */
      SetRPen(rp,1);            /• az l-es (fehér) színnel
                                rajzolunk) */

      for  (i=fl;K11;i++)
      {
        Line(160,i*10,168+i*10,188);
        Lineíi 60,i*10,16O-i*1G,10Q);
        Lined 60,280 i*10,160+í*10,10Ó);
        Line(160,200-i*10,168-1*10,100);
      }
      Delay(50*5);
      CloseScreen(my_screen);
    }
    CloseLibrary((struct Library *)GfnBase);
I  }
  CloseLibraryüstruct Library *)IntuitionBase);
 } }
```

A program egy csillagszerű alakzatot rajzol. Az assembly programot szerintem most már mindenki maga is meg tudja írni.

A vonal rajzolásához szorosan kapcsolódik a PolyDraw, mellyel vonalsorozatot tudunk készíteni. Az Al regiszterbe egy RastPort-ot vár, a DO regiszterbe a koordináták számát kell tenni és az AO regiszterbe pedig a táblázat címét.

A rajzolást az aktuális grafikus kurzor helyétől kezdi el, tehát előtte nem árt Move-al beállítani a kurzort.

Assembly nyelven:

```
            moue.l  GfnBase,a6
            moue.l  #5,dO
            moue.l  #lines,a0
            moue.l  rp.ai
            jsr     _LU0PolyDrauj(a6)
            .
            .
            .
lines:      de.LM    20,80
            dc.uj    48,80
            dc.iii   60,120
            de.  u>  40,120
            uu.ui    ZO.BO
```

C nyelven:

```
WORD lines1[]={  20,80,
                 40,80,
                 60,120,
                 40,120,
                 20,80
               };

        .
        .
        .
        PolyDrauj(rp,5,lines1);
```

A következő rutin a képernyő törlését végzi el. Ez a ClearScreen. Ennek csak az Al-be kell egy RastPort A törlést a grafikus kurzortól kezdi lefelé, tehát, ha az egész képernyőt törölni akarjuk, akkor a (0,0) pozícióba kell mozgatnunk a grafikus kurzort.

```
        moue.l   GfxBase,a6
        moue.l   #0,d0
        moue.l   #100,dl
        moue.l   rp.ai
        jsr      _LU0Moue(a6)

        moue.l   GfxBase,a6
        moue.l   rp.ai
        jsr      _LU0ClearScreen(a6)
```

Vagy C-ben:

```
        Moue(rp,0,0);
        ClearScreen(rp);
```

A képernyőt a **SetRast** rutinnal is letörölhetjük, ha paraméternek a háttér színt adjuk meg. Ez a megadott színnel feltölti a képernyőt.
Az Al-be RastPort-ot vár, a DO-ba pedig a szint.

```
        moue.l   GfxBase,a6
        moue.l   rp.ai
        moue.l   #1,dO
        jsr      _LU0SetRast(a6)
```

C:

```
        SetRast(rp,1);
```

Visszatérve a vonalakhoz. A vonal mintázatát is be tudjuk állítani, erre a RastPort struktúrába kell belenyúlni. Ezt egy 16 bites értékkel tudjuk beállítani, ahol az l-es bit az A pen-t jelenti, a nullás a B pen-t. Itt így már láthatjuk értelmét a SetBPen rutinnak. Assembly-ben így nyúlhatunk bele:

```
moue.l   rp.ai
moue.uj #%1100110011001100,34(a1) ;ez a LinePtrn
or.iü    ·#1,32(a1);      ;Flags - kell a mintázott
                          uonalrajzoláshoz
```

A 34 is a Structure.offsets file-ból a RastPort struktúrából néztük ki.

Ugyanez C-ben így néz ki:

```
rp->LinePtrn=ÖKCCCC;
rp->Flags|=FRST_DOT;
```

Most próbáljunk meg kört rajzolni. Erre a **DrawEHipse** rutin szolgál, mely ellipszist tud rajzolni, de ezzel értelemszerűen lehet kört is. Az Al regiszterbe egy RastPort-ot vár, a D0 és Dl regiszterbe az ellipszis középpontjának koordinátáit kell megadnunk, a D2 és D3 regiszterbe pedig a vízszintes és függőleges sugarakat.

```
moue.l   GfxBase,a6
moue.l   rp.ai
moue.l   #160,dű
moue.l   #100,dl
moue.l   #70,d2
moue.l   #40,(13
jsr       _LU0Moue(a6)
```

C nyelven:

```
DrawEllipse(rp,160,100,70,40);
```

Ezt a kört. vagy bármilyen zárt alakzatot ki is tudjuk fillezni. Ez a **Flood,** mely Al-be egy RastPort-ot vár. D2-be az üzemmódot. DO-ba és Dl-be pedig a koordinátákat. A koordináták a zárt alakzat belsejébe mutatnak.

Az üzemmód OutLine és Colour lehet. Ez még mind nem elég, mert be is kell állítani, hogy ha Outline módot használunk, akkor mely szín legyen az, ahol a fillezés megáll. Ezt ismét a RastPort struktúrában tudjuk beállítani:

```
moue.l   rp,a1
moue.b   #3,27(a1)   ;HOlPen = 3 - a hármas szín a határ
or.b     #$8,32(a1)  ;Flags - kell a kitöltéshez
```

Mindra C ben:

```
rp->flOlPen=3;
rp- >Flags|=HREflOUTLINE;
```

A következő kitöltött ábra a négyzet lesz. Ezt a **RectFill** rutinnal érhetjük el. Azért van külön ilyen rutin, mert a fillezése sokkal gyorsabb, mintha megrajzolnánk a körvonalait és ráengednénk a Flood fillezést.

Az Al regiszterbe egy - na mit is - RastPort-ot vár, a DO-D3-ig az xl.yl, x2,y2 bal felső és jobb alsó koordinátákat várja.

```
moue.l  GfxBase,a6
moue.l  rp.ai
moue.l  #18,dO
moue.l  #10,dl
moue.l  #7B,d2
moue.l  #4B,d3
jsr     _LU0RectFill(a6)
```

C-ben:

```
RectFilKrp.10,10,70,40);
```

Most, hogy a legtöbb alap grafikus rutint átnéztük, úgy gondolom, szöveget is ki kellene már írni. Ezt a **Text** rutin valósítja meg. Az Al-be egy RastPort kell, AO-ba mutató a szövegre, melyet nem kell nullával lezárni, mert a DO-ba úgyis meg kell adnunk a szöveg hosszát. A szöveget az aktuális grafikus kurzor pozíciójától kezdve írja ki az A pen színével. Ennél kipróbálhatjuk, hogy A SetDrMd-t INVERSVID-re állítjuk és JAMl-re.

```
moue.l  GfxBase,a6
moue.l  rp.ai
moue.l  #string,aO
moue.l  #string_end-string,dO
jsr     _LU0Text(a6)

.
.
.
string:     dc.b    "The Rmiga born a Champion!"
string_end:
```

A C forrás:

```
Text(rp,"The flmiga born a Champion!",26);
```

1.9.2 Füled Areas

Most egy speciális területe jön a graphics library-nek, az Area-k. Ezekkel fillezett alakzatokat készíthetünk. Ilyenkor a vonalakat egy speciális vektor táblázatban tárolja, melyben minden alakzatnak zártnak kell lenni.

Ehhez először is kell egy buffer. mely az alakzatokat tárolja. Egy vonalhoz 5 byte szükséges. Tehát 100 vonalhoz 500 byte kell, de a buffer-t word méretre kell igazítani (az elejét), így az 250 word:

```
my_buffer:      blk.w     250,0

ULUORD my_buffer[250];
```

Csinálnunk kell egy Arealnfo slrukLúrát:

```
my_area_info:  dc.l      0       ;UctrTbl
               dc.l      8       ;UctrPtr
               dc.l      0       ;FlagTbl
               dc.l      0       ;FlagPtr
               dc.uj     0       ;Count
               dc.LU     0       ;ManCount
               dC.LU     0       ;FirstH
               dC.LU     0       ;FirstV
```

vagy

```
my_area_info:  blk.b   24,0     ;az egészet egyben foglaljuk le
```

C-ben:

```
struct Rrealnfo my_area_info;
```

Az előző kettőt inicializálnunk kell egy **InitArea** rutinnal, melynek az A0-ba az Arealnfo címe **kell, Al-be** a buffer címe és DO-ba a maxvectors kell.

```
moue.l   GfnBase,a6
moue.l   #my_area_info,aO
moue.l   #my_buffer,a1
moue.l   #100,dO
jsr      _LU0lnitHrea(a6)
```

C-ben:

```
lnitnrea(&my_area_info,my_buffer,100);
```

Ezután létre kell hoznunk egyTmpRas struktúrát:

```
my_tmp_ras  dc.l    0
            dc.l    0
```

vagy

```
my_tmp_ras: blk.b   8,0
```

C-ben:

```
struct TmpRas my_tmp_ras;
```

Le kell foglalni egy akkora képernyőt, melyen a rajzolandó ábrák biztosan el fognak férni, normális esetben akkora mint a screen. melyet megnyitottunk.

Ezt az **AllocRaster** rutinnal tehetjük meg, mely DO-ba és Dl-be a lefoglalandó képernyő méreteit várja és DO-ba kapjuk vissza a lefoglalt terület kezdőcímét. Ezt a képernyőt mi nem fogjuk látni, ez csak a memóriában lesz valahol, átmeneti tárolóként használja a rendszer. Ha DO-ba nullát kapunk vissza, akkor nem sikerült lefoglalni, feltehetőleg memóriahiány miatt.

Ha sikerült, akkor a programunk végén a lefoglalt területet fel is kell szabadítani, ezt a **FreeRaster** rutinnal tehetjük meg. Az AO-ba kell a terület címe, DO-ba és Dl-be pedig a mérete.

A lefoglalás:

```
moue.l   GfnBase,a6
moue.l   #320,dO
moue.l  #200,  dl
jsr      _LU0nilocRaster(a6)
```

A felszabadítás:

```
moue.l   GfnBase,a6
moue.l   my_bit_pláne,aQ
moue.l   #320,d0
moue.l   #200,dl
jsr      _LU0FreeRaster(a6)
```

C-ben a lefoglalás:

```
PLHNEPTR my_bit_plane;

        my_bit_plane=nilocRaster(320,2Q0);
```

A felszabadítás:

```
        FreeRaster(my_bít_plane,320,2Qfl);
```

Ezek után ezt a kettőt is inicializálni kell az InitTmpRas rutinnal, AO-ba a TmpRas struktúra címe kell. Al-be a lefoglalt raster cím, DO-ba pedig a raster mérete-.

Assembly:

```
moue.l  Gf«Base,a6
moue.l  #my_tmp_ras,aO
moue.l  my_bit_plane,a1
moue.l  #16Q00,d0          ;(320/8)*2BQ
jsr     _LU0lnitTmpRas(a6)
```

C:

```
InitTmpRas(&my_tmp_ras, my_bit_plane,
RRSSIZE(32Q,ZeO));
```

Most már csak meg kell mondani a RastPort struktúrának, hogy amiket inicializáltunk, hol találja.

Assembly:

```
moue.l  rp.ai
moue.l  #my_area_info,16(a1)
moue.l #my_tmp_ras,12(a1)
```

C:

```
rp->flrealnfo=&my_area_info; rp->TmpRas=&my_tmp_ras;
```

Ha mindez megvan, most már rajzolhatunk. Itt is megvan a grafikus kurzor, melyet az AreaMove rutinnal tudunk mozgatni. Paraméterezése ugyanaz, mint a normál Move rutiné.

Vonalat rajzolni az AreaDraw rutinnal tudunk, ez is teljesen olyan mint a Draw rutin. Az AreaEllipse is olyan mint a DrawEllipse.

Ha megcsináltuk az ábrát, látni fogjuk, hogy nem látunk semmit. Ez azért van, mert az egészet még csak a vektor táblázatba rakta el a rendszer. Ha meg is akarjuk jeleníteni, akkor meg kell hívni az AreaEnd rutint, mely elvégzi a rajzolást úgy, hogy először a lefoglalt raster-en rajzolja meg a képet és csak ezután másolja rá a látható képernyőre. Ez azért kell, mert most a Blitter speciális fillező üzemmódját használta a rendszer, ezért is olyan gyors és ezért kellett egy másik kép, ahol a Blitter dolgozott. Az AreaEnd-nek csak az Al-be kell megadni a RastPort-ot.

Assembly:

```
moue.l  GfxBase,a6
moue.l  rp,a1
jsr     _LU0RreaEnd(a6) ;csak most kezdi el kirajzolni
```

C:

```
RreaEnd(rp);
```

Ha valaki rendszer alatt akar fillezett 3D vektorgrafikát csinálni, akkor ezt épp arra találták ki.

1.9.3 És a többi...

A legtöbb **Set** valami rutinhoz tartozik egy **Get** valami, mellyel megtudhatjuk annak az aktuális értékét. Például a SetAPen-hez tartozik a GetAPen, mely megadja, hogy épp milyen palettaregiszterrel rajzolunk. Egyébként ez csak a RastPort struktúrából a FgPen mező tartalmát olvassa ki. A többi ilyen ismertetésétől most eltekintünk, a Graphics Library leírásánál ezek ismertetése bővebben is megtalálható.

A másik dolog ami még font(os). a használt font beállítása. Ezt a **SetFont** rutin végzi, melynek egy RastPort-ot kell megadni az Al regiszterbe és egy TextFont struktúrát az AO regiszterbe.

Ezt a TextFont struktúrát pedig az **OpenFont** rutin adja vissza, mely egy TextAttr struktúrát vár az AO regiszterbe.

```
            moue.l  GfxBase,a6
            moue.l  #topaz8,aO
            jsr     _LU00penFont(a6)
            moue.l  dO.font

            moue.l  GfxBase,a6
            moue.l  rp.ai
            moue.l  font,aO
            jsr     _LU0SetFont(a6)

            .
            .
            .
font:       dc.l    0

topaz8:     dc.l    fontname
            dc.uj   8
            dc.b    0       ;style
            dc.b    1       ;flags   PFP_ROMFONT - 1

fontname:           dc.b    "topaz.fonf',0
```

C-ben:

```
struct TeHtfIttr topaz8 = { (STRPTR)"topaz.font",8,0,1}; struct Text-
Font* font;

    .
    .
    .
    font=0penFont(&topaz8);
    SetFontífp.font);
```

Ha lemezről szeretnénk megnyitni egy Font-ot. akkor azt a diskfont.library **OpenDiskFont** rutinjával tehetjük meg, melynek az AO regiszterbe egy TextAttr struktúra címe kell és a DO regiszterbe adja vissza a TextFont struktúra címét. Ezt is a CloseFont rutinnal kell lezárni.
A TextAttr struktúra:

```
struct TeHtfittr{
        STRPTR    ta_Name;   /* a font neue */
        UWORD     ta_VSize;  /* a font mérete */
        UBVTE     ta_Style;  •/* a font stílusa */
        UBVTE     ta_Flags;  /* font preferences és flags */
};
```

A font név végig kisbetű, nullával lezárva. Például: "topaz.font" A font stílusa és a flags a graphics/text.h file-ban megtalálható. ATopaz font általában a ROM-ban levő. a flags ezért PFP_ROMFONT vagyis 1.

2. Library-k

2.1 Az Exec Library

Elérkeztünk a második fejezethez, melyben a főbb library-k függvényeit ismertetjük a teljesség igénye nélkül. Minden library függvénynél feltüntetjük, hogy C-ben és assembly-ben hogyan használjuk, milyen paramétereket vár, mit ad vissza és a bázishoz képest mennyi az eltolási offset-je (ez csak az assembly-nél fontos). A leírásban találkozhatunk az ULONG vagy BYTE kifejezésekkel, ezek definícióit megtalálhatjuk az include/exec/types.h vagy types.i file-okban.

Supervisor - supervisor módú program végrehajtása

ULONG Superuisorf unsigned long (•userfunctionH));
Rx A5

Offset:- 30 ($1E)

Ezzel a gép teljes regiszterkészletét elérhetjük, programunk bármit megtehet, „rendszergazda" módban fut. Az A5 regiszternek tartalmaznia kell a gépi kódú programunk címét, melyet futtatni akarunk.

A programunkat RTE utasítással kell lezárnunk. A SupervisorO nem menti el a regisztereket, erről esetleg a mi programunknak kell gondoskodni.

Például a VBR regiszter tartalmát csak Supervisor módban tudjuk kiolvasni, vagy módosítani, de ez csak 68010-es processzortól van. A következő program ezt olvassa ki és rakja a DO adatregiszterbe:

```
; Superuísor

Start:  moue.l  $4.uj,a6
        moue.l  #Suprg,a5
        jsr     -30(a6)
        rts

Suprg:  mouec   ubr,dO
        moue    sr,d1
        rts
```

Ha nem akarjuk ezt a függvényt használni, lehetőség van aTRAP #x utasítással is Supervisor módba kerülni (a Supervisor is ezt használja):

```
; Trap superuisor

Start:  moue.l  $80,Oldtrap   ;elmentjük a régi TRflP #Q-t
        moue.l  #Suprg,$80    ;beállítjuk a saját programunkra
        trap    #0      ;szoftuer megszakítás, mint PC-n az INT
        moue.l  Oldtrap,$80   ;uisszaállítjuk a TRHP #0-t
        rts                   ;uége

Suprg:  mouem.l ubr,uecbase     ;ha a DO-ba akarjuk berakni,
                                 akkor mouec ubr.dB
        rte

Uecbase:dc.l  0
Oldtrap:dc.l  B
```

Azt, hogy Supervisor módban vagyunk, 68020-nál az SR 13. bit-je állapotából tudhatjuk meg, 68000-nél pedig a 14. bit-bői.

Alert - hiba jelzése a felhasználónak

```
uoid RlerUunsigned long RlertNum);
              D7
```

Offset:- 108 ($6C)

A D7-be megadhatunk egy számot, mely utalhat a hiba okára, de ez lehet magunknak is egy hibakód. A hiba típusa "Recoverable Alert", a gép nem akad ki, nem reset-el, a programunk futhat tovább.
Lássunk rá C nyelvű példaprogramot:

```
/* Rlert */

#include <clib/eKec_protos.h>

uoid main(uoid)
{
  Rlert(BxOBRDCODE);
}
```

íme az assembly forrás:

```
; Rlert

Start:  moue.l  $4.m,a6
        moue.l  #$badc0de,d7
        jsr     -188(a6)
        rts
```

Debug - a beépített debug rendszer indítása

uoid Debuglunsigned long flags);
DO

Ofrset:- 114 ($72)

Meghívja a beépített debugger-t, V39-tó'l ez a "SAD". a Simple Amiga Debugger, előtte pedig a ROM-WACK.

Disable - megszakítások feldolgozásának letiltása

uoid Disable(uoid);

Offset:- 120 ($78)

Letiltja a megszakítások feldolgozását ill. a folyamatban levők feldolgozását felfüggeszti egy EnableO meghívásáig. Nagyon körültekintően kell használni!! A DisableO meghívja a Forbit()-ot is. Meghívása után a regisztereket garantáltan nem változtatja meg.

Enable - megszakítások feldolgozásának engedélyezése

uoid Enable(uoid);

Offsel:- 126 ($7E)

Újra engedélyezi a megszakítások feldolgozását. Minden Disablef) után (miután persze a saját programunk lefutott) egy Enable()-t kell végrehajtani.

Forbid - task-ok letiltása

uoid Forbid(uoid);

Ofíset:- 132 ($84)

Minden más task frissítését letiltja egy Permilf) utasításig, de a megszakításokat nem tiltja le! A regisztereket garantáltan nem változtatja meg. Vigyázva használjuk!

Permit - task-ok engedélyezése

uoid Permit(uoid);

Offset:- 138 ($8A)

Újra engedélyezi a task-ok futását. Minden ForbidQ után használjuk.

SetSR - Status Register beállítása/lekérdezése

ULONG SetSR (unsigned long NeuiSR, unsigned long Mask);
DO DO D1

Offset:- 144 ($90)

A processzor Status Register-ét tudjuk vele lekérdezni lll. beállítani.
A közvetlen MOVE SR.Dx utasítást inkább kerüljük el. helyette ezt használjuk. A DO tartalmazza az új beállításokat, a Mask pedig azokat a biteket, melyeket át kell állítani. Azokat módosítja, melyek 1 értékűek.
A függvény a DO regiszterbe a Status Register régi állapotát adja vissza, ezért lehet lekérdezni is. úgy. hogy a NewSR-t és a Mask-ot nullázzuk.
Általában a processzor megszakítás! szintjének átállítására használjuk.

SetIntVector - új megszakításvektor beállítása

struct Interrupt *SetIntUector(long intNumber, struct Interrupt
DO DO AI
•interrupt};

Offset:- 162 ($A2)

Új megszakítási rutint tudunk beállítani valamelyik régi helyett, tehát lecseréljük azt. A DO regiszterbe a Paula valamelyik megszakitási számat kell raknunk, az Al-be pedig az Interrupt struktúra címét. DO-ba visszakapjuk az régi Interrupt struktúra címét, ha a mi megszakítási rutinunk nem kell már. akkor ezt állítsuk vissza. Csak nem láncolt megszakítást állítsunk be. ha láncolni akarjuk, akkor az AddlntServerO-t használjuk.

Az intNumber-t az alábbi táblázatból beállíthatjuk:

0 - Serial Port, az átviteli buffer üres
1 - Disk Blokk kész
2 - Szoftver megszakítási kérelem
3 - B/K portok és az időzítők
4 - Copper által kiváltott megszakítás
5 - Verticai Blank kezdete
6-A Blitter végzett
7-0. hangcsatorna végzett
8-1. Ivangcsatorna végzett
9-2. hangcsatorna végzett
10-3. hangcsatorna végzett
11 -A serial port Jogadó bujfer-e megtelt
12 - Disk újraszinkronizálva
13 - Külső megszakítás
14 - Master megszakítás

Olyankor akkor kapunk megszakítást, ha ezek közül valamelyik bekövetkezik.

Például ha közvetlen a hardver regiszterek címzésével elindítunk egy hangcsatornát, hogy játsszon le DMA-val valamit, akkor az magától nem fog leállni. A kijelölt memóriát (csak CHIP RAM lehet!) loop-olva, mindig elölről kezdi majd játszani, de mikor állítsuk le?! Amikor a blokk végére ér generálni fog egy megszakítást, ekkor lép életbe a mi megszakítási rutinunk és az például a hangerőt O-ra veszi, s a hangcsatorna elhallgat.

Ez persze így „nem elegáns," ha használjuk a rendszert és közvetlenül a hardver regisztereket is írjuk (lehet, de nem ajánlott!!).

Ha a rendszert alatt fut a programunk, akkor inkább az audio.device-t használjuk, azzal nem lesz semmi gondunk.

Az Interrupt struktúra tartalmaz egy Node-t, egy IS_DATA-t, mely a megszakítás által használt adatterületre mutat, és egy IS_CODE-t, mely a kódra mutat. Az Exec ezeket védett területre másolja, tehát nem nekünk kell allokálni memóriát a megszakításunknak!

A Interrupt struktúra Node részében be kell állítani a ln_Type-t NTJNTERRUPT-ra.

Végül egy táblázat, mely a megszakitási rutinba való belépéskor a regiszterek értékét mutatja:

DO - elállíthatjuk
Dl - aktív megszakítások, mint az INTENA és INTREQ, elállíthatjuk

A0- a cuslom regiszterek báziscime. $DFFOOO. elállíthatjuk
Al -az IS_DATA-ra mutat, elállithatjuk

A5 - ugrási vektor regisztere, elállithatjuk
A6- az Exec library bázisa, elállíthatjuk

Az összes többi regiszter értékét elállíthatjuk, de a megszakításból való kilépés előtt állítsuk azokat vissza.

AddIntServer - megszakítás táncolása

uoid RddIntSeruertlong intNumber, struct Interrupt *interrupt);
DO Al

Offset:- 168 ($A8)

A megszakitási láncba befűzi a megszakításunkat, prioritásának megfelelően.

A rendszer végre fogja haj lant ezeket prioritási sorrendben addig, míg a lánc végére nem ér. vagy valamelyik megszakítási rutin a Z (nulla) státusz bitet O-ba nem állítja. A Verticai Blank (5) visszatérhet 0 Z bit-el is.

Vigyázzunk arra. hogy nem minden magas szintű nyelv gondoskodik az előbbi feltételről! Használjunk inkább gépi kódú megszakítási rutint.

Az intNumber itt is a Paula valamelyik megszakítási számát jelöli.

A rutin hívásakor az alábbiak szerint vannak bizonyos regiszterek beállítva és némelyeket meg is változtathatunk, de a többit ha **elállítjuk,** akkor azokat állítsuk is vissza!

DO - elállíthatjuk
Dl - elállíthaljuk

AO - elállíthatjuk
Al - mutat az IS_DATA-ra, elállíthatjuk

A5 - ugrási vektor regisztere, elállílhatjuk
A6 - elállíthaljuk

A Node-ban be kell állítanunk az ln_Type-l NT_INTERRUFr-ra, a ln_Name-nek pedig mutatnia kell egy string-re. mely a megszakításunkat azonosítja.
A prioritást a ln_Pri állítja.

RemlntServer - megszakítás kiszedése a láncból

uoid RemlntSeruerdong intNumber, struct Interrupt *interrupt);
DO A1

Offset:- 174 ($AE)

A megadott megszakítást kiszedi a láncolatból, ha ez volt az utolsó akkor letiltja a megszakítási láncot.

AllocMem - memória lefoglalása

uoid •flllocMem(unsigned long byteSize, unsigned long require-
£>O DO D1
ments);

OlTsel:- 198 ($C6)

Megadott méretű és megadott követelményekkel ellátott memóriát foglalhatunk saját programunkból például adatok részére. A rendszer az egész memóriát átnézi, addig míg nem talál megfelelő méretű és típusú memória-blokkot. Ha sikerült, akkor a DO regiszterbe a memória- blokk kezdőcímét kapjuk vissza, ha nem akkor nullát. A lefoglalt memóriarészt fel **kelj** szabadítanunk a FreeMemfJ rutinnal.
Erre a területre más program - ha rendszerbarát - akkor nem írhat.
Megszakításból nem szabad meghívnunk! Nézzük meg. hogy milyen típusú memóriát foglalhatunk:

ME,ivir_Q;i-iiF. i&íiK a cnip KAM-üan loglal nelyet, ezt aKKor Kell nasználnunk. ha DMA-val akarjuk a területet elérni. A Copper-lista, a Blitter-nek az adatok, a hangok csak ilyen területen lehetnek.

MEMF_FAST: csak a Fást RAM-ben foglal helyei, ha nincs, akkor O-val tér vissza. A processzor az ilyen típusú memóriában a leggyorsabb, mert a Custom chip-ek ezt nem tudják elérni. Ha ez van beállítva, akkor a MEMF_CHIP-et nem állíthatjuk be.

MEMF_PUI31,1C: minden olyan memóriának melyet megszakítások ill. task-ok használnak (code és data is), ilyen típusának kell lennie.

MEMF_LOCAL: csak V36-tól van ilyen típusú memória, reset után nem veszik el a tartalma.

MEMF_24BITDMA: olyan memóriaterület, melyet a 24 bit DMA-val ellátott Zorro Il-es bus-al működő hardver egységek is el tudnak érni. tehát az alsó 16Mb memóriában foglal helyet. Csak V36-tól létezik.

MEMF_KICK: csak V39-től van ilyen típusú memória, melyet az Exec el tud érni a KickMem és a KickTag feldolgozása előtt és közben. Ez azt jelenti, hogy reset után ez a memória nem veszik el. Ezt használja a RAD: is.

Opciók:

MEMF_CLEAR: a lefoglalt memóriát feltölti nullával.

MEMF_REVERSE: a megadott memória végétől visszafele kezdi el keresni a memóriaterületet. V36-IÓ1 van. de bug-os, V39-IŐ1 jó.

MEMF_NO_EXPUNGE: V39-IŐ1 van. ha a memória lefoglalása nem sikerül, akkor semmilyen memóriaterületet nem fog törölni.

Ha nem állítunk be követelményt a memória típusára vonatkozóan, akkor a rendszer a „legjobb" memóriái próbálja meg nekünk lefoglalni, amit ha van. akkor a Fást Ram-ból próbál.

Ha az a lista, melyben a lefoglalt memóriaterülelek nyilván vannak tartva megsérül, akkor egy Alert-et fogunk kapni, $01000005 hibával (AN_MemoryCorrupt).

Példák:

fllocMemd Q24,MEMF_CMP|MEMF_CLEHR) - lefoglal 1K rész a CH1P RAM-ból és törli azt.

HllocMem(8192,MEMF_CHIP|MEMF_PUBLIC|CLEHR) - 8K Chip. Public és törölt.

AllocAbs - memória lefoglalása megadott helyen.

```
uoid *RllocRbs(unsígned long byteSize, RPTR location);
DO               DO                    Al
```

Orfset:- 204 ($CC)

A megadott memória címtől megpróbál megadott byte-ot lefoglalni, ha az a hely még szabad. Ellenkező esetben nullával tér vissza. Ezt is a FreeMem() rutinnal kell felszabadítani. A lefoglalt méret nem biztos, hogy egyezni fog, mert felfele kerekíti a legnagyobb lefoglalható blokkmérethez, de amennyit akartunk az meg lesz.

Végül is ha nem szabadítjuk fel, nem lesz semmi baj (nem omlik össze a rendszer), de a szabad memóriánk kevesebb lesz. és ezt mások esetleg nem nézik jó szemmel.

Egy assembly példa:

```
start:  moue.l  $4.uj,a6
        moue.l  #$1fOOOO,al
        moue.l  #10,d0
        jsr     -204(a6)        ;$1fOOOB-tól megpróbálunk
                                10 byte-ot lefoglalni
        tst.l   dO
        beq.s   end

        moue.l  dO.mem

free:   moue.l  $4,a6
        moue.l  #$1fOOBO,al
        moue.l  #IO,dQ
        jsr     -210(a6) ;fel is szabadítjuk azt
end:    rts

mem:    dc.l    0
```

FreeMem - memória felszabadítása i

uoid FreeMemíHPTR memoryBlock, unsigned long byteSize);
Al DO

Offset:- 210 ($D2)

A megadott memóriaterületet felszabadítja. Ha egy részt kétszer próbálunk meg felszabadítani akkor guru lesz a jutalmunk |$O1OOOOO9 AN_FreeTwice).

AvailMem - szabad memória

ULONG RuailMemíunsigned long requirements);
DO D1

Orfset:- 216 ($D8)

Megadja a kért típusú legnagyobb szabad memória méretét. A típusok az AllocMem-nél levő típusokkal megegyeznek, egy plusz van. a MEMF.LARGEST.

Példa:

HuailMem(MEMF_FfIST|MEMF_LHRGEST) megadja a legnagyobb lefoglalható szabad fást ram méretét.

FindTask - task keresése

```
struct Task *FindTask(UBVTE *name);
DO                    A1
```

Offiset:- 294 ($126)

A megadott nevű task-ot megkeresi. Ha névnek nullát adunk meg. akkor saját magát keresi meg. Ha sikerült, akkor a DO-ban egy mutatót kapunk egy Task struktúrára. Ha saját magát keresi, akkor nagyon gyors, míg ha nevet adunk meg, az igen lassú.

Assembly példa:

; **FindTask**

```
Start:  moue.l   $4.w,a6
        moue.l   #$0,a1        ;saját magát keresse meg
        jsr      -294(a6)
        moue.l   dO,aO
        moue.l   $a(aO),dO     ;dO-ba kerül egg mutató a task neuére
        rts
```

Ezt érdemes AsmOne-ból futtatni, mert csak így láthatjuk meg az eredményt, ahhoz hogy azt ki is írjuk, egy kicsivel hosszabb program kellene...

Futtatás után DO-ba kerül egy mutató a task nevére. Ha megnézzük azt* a memóriacímet, akkor láthatjuk, hogy az AsmOne neve van ott. Ez azért van. mert a futtatandó programnak nem nyit külön task-ot.

C példa:

```
#include <stdio.h>
#include <clib/enec_protos.h>
#inctude <eKec/tasks.h>
#include <enec/nodes.h>

struct Task *mytask;

uoid main(uoid)
{
   mytask=FindTask(NULL);
   printfC'Task  Name:%s\n",mgtask->tc_Node.ln_Name);
   printfC'Task prioritg:%d\n",mgtask->tc_Node.ln_Pri);
}
```

Mivel C-ben könnyebb kiírni a stdout-ra. ezért itt ki **is** írjuk a task nevel és a prioritását, ami minden bizonnyal 0.

A **FindTaskO** egy mutatót ad vissza, mely egy Task struktúrára mutat (az exec/lasks.h flle-ban meg lehet nézni). Ennek az elején áll egy Node struktúra, melyben benne van egy mutató a task nevére (ln_Name) és egy byte, mely a task prioritását jelenti (ln_Pri). A node felépítését az exec/nodes.h file-ban megnézhetjük.

SetTaskPrl - task prioritásának beállítása

```
BVTE SetTaskPrifstruct Task *task, long prioriity);
DO              AI                DO
```

Offset:-300 ($12C)

Beállíthatjuk egy lask prioritását ill. lekérdezhetjük azt. A visszaadott érték a task előző prioritása. Ha saját programunk prioritását akarjuk átállítani akkor Task struktúrának azt adjuk amit a FindTask(NULL) ad vissza.

Megjegyezzük, hogy 20-ra, vagy fölé ne nagyon állílsunk prioritást, mert 20-as prioritású az inpul.device, vagyis a gépünket azután már nem nagyon tudjuk op. rendszer szinten elérni billentyűzetről vagy egérről.

C példa:

```
#include <stdio.h>
#include <clib/exec_protos.h>
#include <exec/tasks.h>
#include <enec/nodes.h>

struct Task *mytask;
long old;

uoid main(uoid)
{
        mytask=FindTask(NULL);
        old=SetTaskPri(mytask,5);
        prIntfC'Old priority:%d\n",old);
        printf("Task priority:%d\n",mytask->tc_Node.ln_Pri);
}
```

Ez a program lekérdezi a task prioritását (feltehetőleg 0) és átállítja 5-re, majd ezeket ki is írja.

OldOpenLibrary - könyvtár megnyitása verziószám nélkül

```
struct Library *OldOpenLibrary(UBYTE *libName);
DO                              A1
```

OlTset:- 408 ($198)

A megadott könyvtárat megnyitja. Először a memóriában keresi, ha nincs, akkor az aktuális könyvtárban, végül a libs: könyvtárban. Ha sikerült megnyitni, akkor visszaad egy mutatót egy Library struktúrára, ellenkező esetben nulla a visszatérési érték. A library neve csupa kisbetű legyen és 0 álljon a végén. A név nem lehet ilyen: "Work:Kac/libs/gadget.library", csak a library nevét adhatjuk meg.

C példa a könyvtár megnyitására:

```
.
.
struct Library *DosBase;
.
.
OosBase=OldOpenLibrary("dos.library");
.
```

Assembly példa a név lefoglalására: **DosLibrary: dc.b "dos.library",0**

Érdekességképpen megemlítjük, hogy ez a rutin csak azért van a rendszerben, mert az 1.0-ás rendszerben ez hibás volt. Nem ellenőrizte a verziószámot, melyet a DO regiszterbe várt volna, ezért kompatibilitási szempontok miatt muszáj volt meghagyni. A hiba kiküszöbölésére rakták be a -552-es offset-nél levő új OpenLibrary-t, mely már normálisan működik.

Az OldOpenLibrary megegyezik az **OpenLibrary("ualami.library",O);** -val.

CloseLibrary - könyvtár lezárása

```
uoid CloseLibrarytstruct Library *);
                    A1
```

Offset:- 414 ($19E)

Az A1-ben megadott Library struktúrához tartozó library-t lezárja.

Minden megnyitott könyvtárat zárjunk le. Ha ezt nem tesszük meg a rendszer nem omlik össze, de a szabad memória mérete csökken.

OpenDevice - device megnyitása.

```
BVTE OpenDeuice(UBVTE *deuName, ULONG unit, struct lORequest
DO              AO              DO      Al
•ioRequest,
                ULONG flags);
                Dl
```

Offset:- 444 ($1BC)

Megnyitja a megadott device-t a megadott egység számmal és inicializálja a megadott struktúrát. A device-t először a memóriában keresi ha ott nincs akkor a devs: könyvtárban. Sikeres megnyitás esetén a megadott struktúrát inicializálja és nullát ad vissza.

A device nevét végig kisbetűvel és nullával lezárva kell megadni (pl.: "timer.device"). de ha a devs: könyvtárban van a device akkor ugyanolyan kis/nagybetű használatával kell megadni. A dos nem tesz különbséget a kis és nagybetűk között, de az exec igen.

A név megadás nem lehet ilyen: "Work:Kac/Devs/gadget.device".

A flags-ot csak bizonyos device-knál kell megadni, segédinformáció átadására használatos. Hibás megnyitás esetén hibakódot ad vissza.

CloseDevice - device lezárása

```
uoid CloseOeuice(struct lORequest (ioRequest);
                 Al
```

Offset:- 450 ($1C2)

A megadott struktúrához tartozó device-t lezárja. Be nem fejezett B/K művelet közben ne zárjuk le a device-t. Lezárás után a struktúra fel lesz szabadítva, újra lehet használni.

DoIO - B/K müvelet végrehajtása várakozással

```
BVIE DolOfstruct lORequest *ioRequest);
DO       Al
```

Offeet:- 456 ($1C8)

Végrehajtja az B/K struktúrában megadott B/K parancsot. Ez mindig megvárja míg az B/K művelet be nem fejeződik. A rutin egy byte-ot ad vissza, a hibakódol. Ez a lORequest struktúra io_Error mezőjének másolata

A lemezmellékleten a trackdisk.device használatára láthatunk egy példát, hogyan kell azt megnyitni, beolvasni a bootblock-ot a lemezről és hogyan kell lezárni a device-t.

SendlO - B/K művelet végrehajtása várakozás nélkül

```
uoid SendIO(struct lORequest *ioRequest);
            Al
```

Offset:- 462 ($1CE)

Ugyanaz mint a DoIOO. csak nem ad vissza semmit és nem várja meg, hogy az B/K müvelet befejeződjön.

WaitIO - várakozás B/K műveletre

BVTE WaitIO(struct IORequest *ioRequest);
DO A1

Offset:- 474 ($1DA)

A megadott B/K művelet befejezésére vár. Ha az B/K müvelet soha nem fejeződik be akkor a task-unk felfüggesztődik, ha már a rutin meghívása előtt befejeződött akkor azonnal kilép. A visszaadott byte a hiba típusát mutatja, ha az nem nulla.

OpenResource - resource megnyitása

APTR OpenResource (AFTR resource);
DO A1

Offset:- 498($1F2)

A megadott resource-t megnyitja, mely csak a memóriában levők egyike lehet. Éppen ezért nincs is olyan rutin, mellyel egy resource-t le tudnánk zárni, tehát senki ne keressen CloseResource-t, mert nincs.
Az A1-ben egy mutató kell a resource nevére, csupa kisbetű, végén nulla.
Ha nagyon akarjuk a gépünket rendszer alól hardverszinten programozni, akkor használjuk a resource-kat. Ha lefoglaltuk valamelyiket, akkor más (szintén így irt) program nem fogja azokat a hardver regisztereket piszkálni.

TypeOfMem - memória típusa.

ULONG TypeOfMemIRPTR address);
DO A1

Offset:- 534 ($216)

Az A1-ben megadott memóriacím típusát adja vissza (MEMF_CH1P, MEMF_FAST, stb.). Ha olyan címet adtunk meg. ahol nincs is RAM. akkor nullát ad vissza eredményül. Ide tartozik a ROM is és valamely expansion területe is. 3.0-ás Kickstart-on kipróbáltuk 68030 procival, de a $1000000-tól levő Fást RAM-ot egyáltalán nem volt hajlandó felismerni, ez alatt pedig csak azt tudta megmondani, hogy van-e ott RAM. A RAM-ra pedig chip/fast/24bitdma/public/kick típusokat adott.

OpenLibrary - könyvtár megnyitása.

```
struct Library *OpenLibrary(UBVTE *libName, unsigned long uersion);
DO                          Al              DO
```

OrišeL:- 552 ($228)

A javított OldOpenLibrary. Ellenőr/.i a megadott verziószámot is. mely nagyobb vagy egyenlő lehet. Ugyanúgy működik mint az OldOpenLibrary
Lehetőleg ezt használjuk, de így 1.0-ás gépen nem fog működni (van még olyan valakinek?!).

CopyMem - általános célú memória másolása.

```
uoid CopyMem(HPTR source, HPTR dest, unsigned long size);
             AO           Al         DO
```

Orfset:- 624 ($270)

Memóriablokk másolására jó. ha a/ nem átlapolt (például 1000-161 2000-ig 4000 byte-ot). A másolandó blokk mérete byte-ban értendő. Ha a méret nulla, akkor nem másol át semmit.

CopyMemQuick _ optimalizált gyors memóriamásolás.

```
uoid CopyMemQuícMflPTR address, unsigned long length,
                       AO              Al
unsigned long caches);
DO
```

Orfset:- 630 ($276)

Teljesen szélopiimalizáit másolás, nagyon gyors, viszont a forrás és cél helyet long-ra kell i<a/:ítani (lehes.sen a címet 4-gyel osztani)- A méretet byte-ban kell megadni, de a/ is 4-gyel osztható legyen. Átlapolási ez sem tartalmazhat.

ColdReboot - a gép hideg indítása.

uoid ColdReboot(uoid)

Offset:-726

Minden külső memóriát és perifériái, reset-el és a gépet újraindítja a bekapcsolási diagnosztikával. Ez a funkció soha sem tér vissza.

A diagnosztika rövid leírása időbeni sorrendben:

- törli az összes chip-et a régi adatok miatt,
- kikapcsol minden DMA-t és megszakítást a teszt idejére.
- törli a képernyőt.
- teszteli a hardvert, ellenőrzi a processzort.
- megváltoztatja a képernyő színét.
- minden Rom-on elvégzi az ellenőrző összeg kiszámítását és tesztelését.
- megváltoztatja a képernyő színét.
- teszteli a Chip Ram-ot,
- megváltoztatja a képernyő színét,
- ellenőrzi, hogy tud-e programot betölteni valamiről.
- megváltoztatja a képernyő színét,
- inicializálja (linkeli) a library-ket.
- ellenőrzi és linkeli az esetleges memóriabővítéseket.
- bekapcsolja a DMA-t és a megszakításokat.
- megnézi, hogy van-e 68010. 68020 vagy 68881.

Teszt közben az alábbi színeket kaphatjuk:

- sötétszürke: a hardver teszt sikerült.
- világos szürke: tud tölteni.
- fehér: az inicializálás megtörtént.
- piros: ellenőrző öss/eg hiba a Rom-ban.
- zöld: hiba a Chip Ram-ban,
- kék: hiba a CustomChip-ek valamelyikében.

- sárga: Guru előtt talált egy újabb hibát.

GetMsg - Bekér egy üzenetet a megadott üzenet portról,

slruct **Message** *GetMsg(struct MsgPort *port);

OlTset:- 372 ($174)

A függvény megadja a kimenetén a címét az üzenetnek ha érkezett olyan, egyébként NULI^t. A függvény tehát nem másolja az üzenetet csak a címét adja vissza.

Az üzenet átvétele után az üzenetre válaszképpen meg kell hívnunk a ReplyMsgO függvényt. A függvény nem vár az üzenet létrejöttére, csak bekéri azt. A várakozási a VVaiti). a WaitPortO függvénnyekkel végezhetjük.

Argumentumként az üzenet port címét várja.

PutMsg - Egy üzenet elküldése a megadott üzenet portra.

uoid PutMsgístruct MsgPort *port, struct Message *msg);

Offset:- 366 ($16E)

A függvény elküldi a megadott üzenetet címét (tehát nem másolja azt le) a megadott üzenet portra. A válasz akkor érkezett meg. ha azt a címzett task nyugtázza a ReplyMsgO függvény segítségével.

ReplyMsg - özenet nyugtázása.

uoid ReplyMsgístruct Message *msg);

Offsei "78($17A)

A függvény elküldi a megadott üzenetet az üzenetet küldő üzenet portnak, nyugtázási célból. Az üzenete átvételéhez hozzá tartozik az üzenet nyugtázása is. amelyet ezzel a függvénnyel végezhetünk el.

Wait - Várakozás egy vagy több jelző bitre.

ULONG WaitíULONG sígnalSet);

Offset:- 318 ($13E)

A függvény segítségével a programunk task-ját várakozásra kényszerilhejük, ameddig a kívánt jelzőbit(ek) a kívánt értékűek nem lesznek. Ha a jelzőbit(ek) felvették a kívánt értéket, akkor a Task (program) felébred és lovább folytatja a munkáját. A függvényt ne hívjuk meg Supervisor módban vagy megszakítás kezelő programban.

WaitPort - Várakozás ameddig a megadott port NULL.

struct Message *lüaitPort(struct MsgPort *port);

Offset:- 384 ($180)

A függvény addig alvásra kényszeríti a meghívó task-ot. ameddig a megadott port nem tartalmaz érvényes üzenetet, majd az első érvényes üzenet címével tér vissza. A függvény a Wait() függvén"! '-i'-ia meg. így a/ annál leírtak erre a függvényre is érvényesek.

2.1.1 Az Exec Library függvényei ojfszet sorrendben

-30 Supervisor
-72 InitCode
-78 InitStruct
-84 MakeLibrary
-90 MakeFuncüons
-96 FindResident
-102 InitResident
-108 Alert
-114Debug
-120 Disable
-126 Enable
-132 Forbid
-138 Permit
-144 SetSR
-150 SuperState
-156 UserState
-162 SetlntVector
-168 AddlntServer
-174 RemlntServer
-180 Cause
-186 Allocale
-192 Deallocate
-198AllorMem
-204 AllocAbs
-210 FreeMem
-216 AvailMem
-222 AllocEntry
-228 FreeEntry
-234 Insert
-240AddHead
-246 AddTail
-252 Remove
-258 RemHead
-264 RemTail
-270 Enqueue
-276 FindName
-282 AddTask
-288 RemTask
-294 FindTask
-300 SetTaskPri
-306 SeLSignal
-312 SetExrept
-318 Wait
-324 Signal
-330 AllocSignal
-336 FreeSignal
-342 AllocTrap
-348 FreeTrap
-354 AddPort
-360 RemPort
-366 PutMsg
-372 GetMsg
-378 ReplyMsg
-384 WaitPort
-390 FindPorl
-396 AddLibrary
-402 RemLibrary
-408 OldOpenLibrary
-414 CloseLibrary
-420 SelFunction
-426 SumLibrary
-432 AddDevice
-438 RemDevice
-444 OpenDevice
-450 CloseDevice
-456 DoIO
-462 SendlO
-468 ChecklO
-474 WaRIO
-480 AbortlO
-486 AddResource
-492 RemResource
-498 OpenResource
-522 RawDoFmt
-528 GetCC
-534 TypeOfMem
-540 Procure
-546 Vacate
-552 OpenLibrary
-558 InitSemaphore
-564 ObtainSemaphore
-570 ReleaseSemaphore
-576 AUemptSemaphore
-582 Obtain-Semaphoreüst
-588 Release-Semaphore List
-594 FindSemaphore
-600 AddSemaphore
-606 RemSemaphore
-612 SumKickData
-618 AddMemList
-624 CopyMem
-630 CopyMemQuick
-636 CacheClearl)
-642 CacheClearE
-648 CacheControl
-654 CreatelORequest
-660 DeletelORequest
-666 CreateMsgPort
-672 DeleteMsgPort
-678 ObtainSemaphore-Shared
-684 AllocVec
-690 FreeVec
-696 CreatePool
-702 DeletePool
-708 AllocPooled
-714 FreePooled
-720 AttemplSemaphoreShared
-726 ColdReboot
-732 StackSwap
-738 ChildFree
-744 ChildOrphan
-750 ChildStatus
-756 ChildWait
-762 CachePreDMA
-768 CacheE^ostDMA
-774 AddMemHandler
-780 ReniMemHandler

2.2 Az Intuition Library

A könyv első felében szinte csak az Intuition Library függvényeit használtuk, hiszen az ablakoktól a menükig szinte minden a felhasználóval kapcsolatot tartó dolog az Intuition része. Ebből kifolyólag talán a legnagyobb könyvtár az Amigában. A könyvtár részletes ismertetése meghaladná a könyv terjedelmeit, de a könyvben használt összes függvény megtalálható lesz ebben a részben, részletes leírással. Példákat nem mutatunk rá. hiszen azokat az eddigiekben már bemutattuk (lásd. lemezmelléklet). Ettől függetlenül azért lesznek olyan függvények is a leírásban, amelyekre példát nem mutattunk, vagy a függvény egyértelriiűségéből adódik, vagy a könyv terjedelme nem teszi ezt lehélővé. A nem ismert függvénnyekkel nyugodtan próbálkozzunk, sőt az ismertettekkel is.

ActivateGadget - Aktivizálja a (string vagy custom) gadget-et.

```
BOOL HctiuateGadgetfstruct Gadget *Gadget, struct Window
DO                    AO                    AI
•LUíndouu, struct Requester *Reqester);
A2
```

Offset:- 462 ($1CE)

A függvénnyel egy gadget-et aktivizálhatunk az ablakunkon ill. a Requesterünkön. Az ablaknak ill. a Requester-nek aktívnak kell lennie. Ne próbáljunk nem aktív, vagy nem engedélyezett ablakra ill. Requesterre a függvényt meghívni. Ha Requester esetén használjuk a gadget-nek a GTYP_REQESTER flag-jának beállítottnak kell lennie. A függvény visszatérési értéke akkor TRUE. ha a string gadget-et aktivizáltunk vele. egyébként FAI.SE.

ActivateWindow — Aktivizál egy [ntuition Ablakot.

```
uoid  flctiuatelilindoiii(struct  IDíndow *LUíndow);
```

Ofíset:- 450 ($1C2)

A függvénnyel egy Intuition Ablakot aktivizálhatunk, mintha az user rákattintott volna az egerével. Hatása megegyezik a WFLG_ACTIVATE flag beállításával.

AddClass - Létrehoz egy publikus class változót. (V36!)

```
uoíd RddClass(struct Class *Class);
          AO
```

Offset:- 684 ($2AC)

Hozzáad a belső nyilvános boopsi class listához egy class változói nyilvános használatra. Ezt a függvényt meg kell hívnunk a MakeClassQ függvény használata után. Paraméterként ennek a függvénynek a viszaadott értékét várja.

AddGadget - A gatlget listánkhoz hozzáfűz egy gadget-et.

```
UÜORD flddGadgettstruct LUindow *UJindow, struct Gadget *Gadget,
DO                AO                A1              DO
UIDORD pos);
```

Offset:- 42 ($2A)

A megadott gadget-et befűzi az ablak gadget listájába a pos paraméter-* ben megadott helytől A Pos paraméter jelentése: 0 a gadget a lista elejére kerül. 1 a gadget az első után kerül de a második elé ... Ha a pos változó nagyobb vagy nem valós értéket tartalmaz, a függvény a gadget-et a lista végére illeszti és a vissza térési értékként a gadget listában elfoglal helyéi kapjuk.

AddGList - Az ablakunkhoz vagy a requesterünkhöz egy gadget listát illeszt.

```
ULUORD RddGListlstruct UJindow *UJindoiu, struct Gadget *Gadget,
DO              AO                  A1
ULUORD pos, WORD num, struct Requester *Requester);
DO          D1        A2
```

Offset:- 438 ($1116)

Egy Oadget-ekból álló listát illeszt a/ ablakunkhoz, vagy requester-ünkhoz. Az illeszlés helyét megadhatjuk a pos paraméterben. A num paraméterben a befűzni kívánt Gadget-ek számát adhatjuk meg. Ha -1-et adunk meg a lista végéig befűz minden gadget-et. Visszatérési értékként a ténylegesen befűzésre került gadgetlista pozícióját adja vissza.

AllocRemember - AllocMemfl szerű függvény a memória egyszerű használatáért lett létrehozva.

```
HPTR HIlocRemembeHstruct Remember **, ULONG Size, ULONG Flags);
DO              AO              DO      D I
```

OITset:- 438 ($1136)

A struct Remember egy olyan struktúra, amelyben a rendszertói foglalt memóriaterületeket egyszerűen nyilvántarthatjuk, és a felszabadításukkor nem kell mindegyiket egyesével felszabadítanunk, hanem csak egyszer meghívnunk a FreeRememberO függvényt. A rerneber strukrűra az alábbi:

```
struct Remember
{
    struct Remember *NeHtRemember;
    ULONG RememberSize;
    UBVTE *Memory;
}
```

A függvény paramétereként egy ilyen struktúrára mutató pointer mutatóját várja. A függvény a remember struktúránkba belánrolja az új memóriaterületet.
A lefoglalni kívánt memória méretét a Size paraméterben adhatjuk meg. A lefoglalandó memória tulajdonságait a Flags paraméterben várja. (lásd. AllocMemf)). Visszatérési értékként a lefoglat memória cimét kapjuk.

Nézzünk rá egy példaprogramot:

```
struct Remember *RememberKey;
RememberKey=NULL;
buffer=RIlocRemember(í>RemeberKey> BUFSIZE, MEMF_CHIP);
if(buffer)
{
    /* Használjuk a buffert */
    ...
}
FreeRemeber(&RememberKey, TRUE);
```

AllocScreenBuffer - Létrehoz egy ScreenBuffert a dupla pufferes screen használatához. (V39!)

```
struct ScreenBuffer *RIlocScreenBuffer( struct Screen *Screen, struct
DO                                      AO                      AI
BitMap *BitMap, ULONG Flags);
```

DO Offset:- 396 ($171)

A függvény segítségével egy puffért allokálhatunk, amelyet a screen képernyőjeként két képernyős módban használhatunk. A Screenparameteben természetesen a Screenünkre vár mutatót. A BitMapnal a screen bítmapjára vagy a második bitmapra. amit váltani akarunk. A Flag paraméternél az alábbi falgokat használhatjuk ill. kombinálhatjuk egymással:

Ha nem CUSTOMBITMAP screenünk van. állítsuk SB_SCREEN_I3ITMAP flagot. hogy a ScreenBuffer váltakozzhasson a screen aktuális BitMapjával. SB_COPY_BITMAP hatására a screen bitmapját átmásolja a létrehozott pufferbe.

A függvény visszatérési értéke NULL ha nem sikerült lefoglani a kért területet, egyébként annak címével tér vissza.

AlohaWorkbench - Szándékosan nem publikálják.

```
uoid fllohalDorkbenchílong UJbport);
                           DO
```

Ofíset:- 402 ($192)

A függvényről szándékosan nem publikálnak semilyen információt.

AutoRequest - Automatikus Requester készítés.

```
BOOL  flutoRequestlstruct  UJindow *LUindow, struct IntuiTeKt
DO                  AO                        A1
•BodyText, struct IntuiTeKt *PosTent, struct IntuiTeKt *NegTeKt,
           A2                          A3
ULONG PosFlags, ULONG NegFlags, UJORD UJidth, UJORD Height);
DO              D1              D2            D3
```

Offset:- 348 ($15C)

A függvény egy Simple requestert hoz létre, a megadott paraméterekkel. Részletesen lásd. 1.8.3.2.1. *~

BeginRefresh - Az ablak frissítésének kezdete.

```
uoid BeginRefresh(struct IDindou) *UJindoiu);
                  AO
```

Offset:- 354 ($162)

A függvény egy optimalizált frissítési ciklus kezdete. Ha az ablakunknál beállítottuk a WFLG_SIMPLE_REFRESH flag-ol. akkor a frissítést végezhetjük ennek a függvénynek a segítségével, amikor egy IDCMF_REFRESHWINDOW üzenetet kapunk. A frissítés közben ne hívjuk meg a RefreshGadgetsfJ vagy a RefreshGList() függvényeket! A Gadget-eket az Inutition automatikusan frissíti, ha szükséges. A frissítés végét az EndRefresh() függvény meghívásával érhetjük el. A két függvény között azokat az ablakba rajzolt dolgainkat rajzolhatjuk újra. amelyek sérülhetnek. Például a Borderek. InutiTextek. stb.

Nézzünk példát a frissítés lefolyására a gyakorlatban:

```
...
switch(imsg->Class)
{
    ...
    case   IDCMP_REFRESHU)INDOUI:
           ii)indoiu=imsg-> IDCMPLUindoiu;

           /* Ez a sor csak a kétlépcsős frissítéskor kell. */
           LockLayerInfo(C'UJíndouj->IlJScreen->layerInfo);
           /* Hz Első lépcső */
           origclip = InstallClipRegionduindoLD->IDLayer, regioni);
           BeginRefresh(windouj);
           myRefreshRegioni(LUindoiu); /*Például az InutiteKtek újra*/
                                       /•kirajzolása stb. */
           EndRefreshtujindow, FHLSE);

           /* Második lépcső */
           InstallClipRegionduindouj-›Wlayer, regíon2);
           BeginRefresh(window);
           myRefreshRegion2(ujindouj);
           EndRefresh(window, TRUE);

           /• uisszaállítás, és unlock */
           InstallClipReyionduindouj-›UJLayer, origclip);
           UnlockLayerInfo(&UJindouj->IUScreen-›LayerInfo);

           break;
    ...
}
```

A példában egy két menetben végrehajtott frissítést láthatunk. Természetesen nem minden alkalmazásban van szükség a két lépcsős frissítésre.

BuildEasyRequestArgs - Egyszerű létrehozása rendszer request. (V36!)

BtiildEasyRequest - Mint a BuildEasyRequestArg, csak az argumentumokat egyesével adhatjuk meg. (V36!)

```
struct lilindoLU *BulidEasyRequestRrgs(struct LDindoiu* UJindouj, struct
DO                                    AI                          A2
EasyStruct *Es, ULONG IDCMPflags, HPTR Rrgsl;

struct LUíndow* BuildEasyRequest(struct UJindoui *UJindouj,
DO                                    AI
struct *EasyStruct
A2
*Fs, IIIONK mrMPflaos, HPTR Hrgi, ...):
     DO                    A3 ...
```

Offset:- 594 ($252)

A függvények segílségével a V36 os Intuiüonnal készítethetünk egyszerű requesler-t az EasyStruct segítségével. Részletesen lásd az 1.8.4 részben.

BuildSysRequest - Megcsinál és megjelenít egy rendszer (sys) rqeustert.

```
struct IDindoui *BuildSysRequest(struct UiindOLU *UJindouj, struct
DO                               AO                         Al
IntuinText
*BodyTeKt, struct InutiText *PosTent, struct InutÉTent *NegTent, ULONG
           A2                  •      A3                        DO
IDCMPFIags, liORD LUídth, UJORD Height);
            01           D2
```

Offeet:- 360 ($168)

A függvény segítségével egyszerűen használhatunk rendszer reqester-eket. Részletesen lásd 1.8.2.3. részben. Használata megegyezik az AutoRequester() használatával. A V36-os rendszereken mindig FALSÉ értéket kapunk visszatéréskor! Ha az ablak pointerének NULL-t adunk meg V36 a default pubscreenre nyílik meg, ami nem mindig a Workbench!

ChangeScreenBuffer - A srreen buffreinek megcserélése. (V39!)

```
ULONG ChangeScreenBuffertstruct Screen *Screen, struct ScreenBuffer
DO                              AO                A1
*Buffer);
```

Offset:- 780 ($30C)

A függvény a Screen két képernyőjét cseréli meg. így lehetőségünk van két képernyőt használni. Míg az egyikre rajzolunk, a másikban gfbnyörködik az user. Bemenetként a Screen struktúra mutatón kívül a ScreenBuffer mutatót várja. Ezt például az AllocScreenBufferíJ függvény meghívásával kaphatunk

Visszatérési értéke, ha a csere sikerült, akkor nem 0. ha nem sikerült a csere, akkor 0. A sikertelenség oka lehet az user is, ha éppen használta a Gadget-eket vagy a menüket.

ChangeWindowBox - Lecseréli az ablak pozícióját és vagy méreteit. (V36!)

```
Uoid ChangeLDindoLuBon(struct LUindoiu *lilindouj, UJORD Left,
                       AO                           DO
UJORD Top, UJORD UJidth, UJORD Height);
Dl         D2            D3
```

Offset- 486($1E6)

A függvény szimulája az user-t. mintha az ablakot megváltoztatta volna. Használatakor ha beállított, az IDCMP_CHANGEWINDOW üzenetet kapjuk.

•. **ClearDMRequest** - Törli az ablakról a DuplaMenü requestert.

BOOL ClearDMRequestístruct UJindow *U)indou;);
AO

Offset:- 48 ($30)

Törli az ablakról a DuplaMenü requestert. Vissza téréskor TRUE-t ad ha a Requester aktivizálva volt. ha nem akkor FALSÉ.

ClearMenuStrip - Törli a menüket az ablakról.

uoid CelarMenuStripístruct Windoiu *UJindoLu);

AO

Offset:- 54 ($36)

Törli az ablak menüit.

ClearPointer - Törli a felhasználói egérpointert az ablakról.

uoid ClearPointer(struct lilindow •LUindow);

AO

Offset:- 60 ($3C)

Törli a felhasználó által definiált és használt pointert, amit a SelPointerfJ függvénnyel rendelt az ablakhoz. A függvényt csak aktív ablak esetében hívjuk meg. Hatására visszaáll a Workbench default egérpointerre.

CloseScreen - Becsukja az Intuition screent.

BOOL CloseScreenístruct Screen *Screen);
DO AO
Offset:- 66 ($42)

A függvény becsukja az Intuition által megnyitott screent (OpenScreen(). OpenScreenTagO stb.) Csak V36-tól van visszatérési értéke, alatta void. A visszatéréskor TRUE értéket kapunk ha sikerült, és FALSÉ értéket ha van nyitott ablak a screen-en és emiatt nem tudta azt becsukni.

CloseWindow - Becsukja az Intuition ablakot.

```
uoid Closeli)indOLu(struct Ulindow *UJindow);
                    AO
```

Offset:- 72 ($48)

Egy Intuition álltai létrehozott ablak becsukását végzi. Az Intuiton felszabadítja a létrehozott memóriát és kiveszi a rendszerablak láncolatából. Amikor ezt a függvényt meghívjuk, az összes IDCMP üzenet elveszik ami akkor érkezik, amikor az Intuition deallokálja az ablakot, programunk nem tud választ küldeni azokra az Intuition-nak, így adatvesztés jöhet létre más taskok felé. Ezt a problémát kikerülendően használjuk a CloseWindowSafetlyO függvényt. Ennek leírását megtaláljuk az 1.6.4.2. részben. Ne feledkezzünk meg az ablak bezárása előtt az ablakhoz csatlakozó menük felszabadításáról. Ha az ablak „Visitor Window". be kell zárni mielőtt a screen bezáródna, mert a screen így bezárhatatlan lesz. Visitor Window akkor lehet az ablakunk, ha mondjuk egy Pub screen-re nyitottuk.

CloseWorkBench - Bezárja a Workbench screent.

```
LONG CloselDorkbench(uoid);
```

Offset:- 78 ($4E)

Ez a függvény bezárja a Workbench-et. ha egyetlen ablak sem nyitott (program ablak, nem meghajtó!). Ezután felszabadítja a speciális puffereket, a Workbench Screen-t, valamint az olyan programokat, amelyek például lemezműveletet várnak inaktivizálja stb. Ha sikerült a Workbench bezárása, TRUE értéket ad vissza, egyébként FALSE-val.

tíuirentTime - Az aktuális idő lekérdezése.

```
uíod CurrentTime(ULONG *SecOnds, ULONG *Micros};
                 AO              Al
```

Offset:- 84 ($54)

Bemásolja a paraméterekben megadott változókba az aktuális időt. Ez az idő körülbelül hatszor frissítődik egy másodpercen belül.

DisplayAlert - Kreál egy képernyőt az Alert üzenetnek.

```
BOOL Displayfllert(ULONG RlertNumber, UBVTE *String, UUJORD Height);
DO                 DO                 AO             Al
```

Offset:- 90 ($5A)

Részletes leírása az 1.9.2 részben. A V36-os rendszerektől az Alert-hez mindig a Topaz 8 típusú betű számolódik. (80 karakter egy sorban.)

DisplayBeep - A képernyő megvillantása.

```
uoid OisplayBeep(struct Screen *Screen);
                          AO
```

Orfset:-96 ($60)

A függvény megvillanLja a paraméterként megadott screent és V36-IÓ1 a beállított hangot lejátssza (sys:prefssound). Részletesen lásd. 1.8.1.
Amikor a Screen NULL, akkor mindeggyik screen megvillan.

DisposeObject - "boopsi" objektum tőrlése. (V36!)

```
uoid  DisposeObject(RPTR);
                      AO
```

Offset:- 642 ($282)

A függvény törli a boopsi objektumot és az összes kiegészítő adatot Ezeket az objektumokat a NewObjectO függvénnyel hozhatjuk létre, lásd 1.6.5. rész. Ha a iuggvényt NULL paraméterrel hívjuk meg. nem történik semmi. Paraméterként egy absztrakt pointert vár a boopsi objektumra.

DoGadgetMethodA - A boopsi Gadget-ek metódusai iák lekérdezése (V39!)

DoGadgetMethod - A boopsi gadget metódusának lekérdezése, a Method 1D Telsprolással (V39!)

```
ULONG DoGadgetMethod(struct Gadget *Gadget, struct lilindooi
DO                          AO                          A1
*LUindouj, struct Requester *Requester, Msg Messagp);
          A2                                A3
```

ULONG DoGadgetMethodtstruct Gadget *Gadget, struct LUíndom *LUindou), struct Requester *Requester, ULONG MethodID, ...);

Ofrset:- 810 ($32A)

Hasonlít a DoMelhodQ függvényre, de a/ összefüggési információkban es a Gadget Class eldöntésében az Intuiüon-hoz kapcsolható felhasználói gadget-ekröl a gadget leírása adja meg a kívánt támpontot. E/t a függvényt akkor célszerű használni, ha a Boopsi Gadel objektumokról kivánunk inlormáriót nyerni, amelyek propagáltak a gadgetről Az eltérés a fent említett DoMethoci-hoz képest annyi, hogy ez a függvény gondoskodik a Gadgetlnlo pointerről (ha ez lehetséges), amikor a metódus bekérődik. A függvény visszatérési értéke a megadott gadget melnocllD-iol lu^y. Ez a lekérdezendő metódustól és az objektumtól lügg. Ezek a metódusok a Gadget-ek-nél publikáltak (a felhasználó által aki a Gadget-el megírta!).

DoubleClick - leteszteli, hogy a két időpont megfelel-e a duplaklikk időnek.

```
BOOL DoubleClíckíULONG StartSecs, ULONG CurrentSecs,
DO                 DO                 DI
ULONG CurrentSecs, ULONG CurrentMicros);
D2                 D3
```

Offset:- 102 ($66)

Összehasonlítja a két időpontot, hogy az megfelel-e a Prefs-ben meghatározott duplaklikk időpontnak? Ha megfelelt, a visszatérési értéke TRUE. ha nem. FALSE. Azt hiszem beszédesebb ha használatára mutatok egy példát.

```
...
...
if(msg->Class == IDCMP_MOUSEBUTTONS)
{
    /• Hz egérgombja lenyomott uolt */
    if(msg->Code == SELECTDOUUN)
    {
        /* H bal egérgombja lenyomott uolt. •/
        /• elmentjük a régebbi időt */
        sec2 = secl;
        mic2 = sec2;

        /* Hz új idő */
        secl = msg->Seconds;
        micl = msg->Microns;

        /* Leellenőrizzük, hogy dupla klikk-e? */
        if(DoubleClick(sec1,mic1,sec2,mic2))
        {
            printTt" Dupla klíkk!\n");
            /* fl uáltozók törlése. */
            sed=0;
            mid=0;
        }
    }
}
...
...
```

DrawBorder - Egy bordér rajzolása a rastportra.

```
uoid DrawBorder(struct RastPort *RastPort, struct Bordér *Border,
                AO                          AI
WORD LeftOffset, WORD TopOffset);
DO               D1
```

Offsel:- 108 ($6C)

A bordér struktúrában megadott bordér kirajzolása a reastportra. A LeftOffset és a TopOffset a rajzolás kezdetét a O.O-ás kordinátát definiálja a rastport dimenziójában. A rajzolás mindaddig folytatódik, ameddig a bordér struktúra NextBorder mezőjében NULL nem található. Részletesen lásd. 1.6.2.2 részben.

DrawImage - Egy Image kirajzolása a rastportra.

```
uoid DraujImagetstruct RastPort *RastPort, struct Image • Image, WORD
              AO                          AI                    DO
LeftOffset, UJORD TopOffset);
                Dl
```

Orrset:- 114 ($72)

Az Image struktúrában definiált Image kirajzolása a rasztportra. A LeftOffset és a TopOffset a kirajzolás kezdeti koordinátáit definiálja. A kirajzolás mindaddig folyamatos, ameddig az Image struktúra Nexhnage mezője nem NULL-t tartalmaz. Részletesebben lásd az 1.6.2.3 részben.

DrawImageState - Image kirajzolása speciális megjelenítési opciókkal. (V36!)

```
uoid DrauiImageStateístruct RastPort *RastPort,
                        AO
struct Image *Image, WORD LeftOffset, UJORD TopOffset,
AI                   DO               D I
ULONG state, struct DraujInfo *DrawInfo);
D2           A2
```

Offset:- 618 ($25A)

A függvény egy Image-t rajzol a képernyőre, hasonlóan a Drawlmage függvényhez, csak itt beállítható a rajzolás módja. Ezeket a rajzolási módokat az alábbi konstansokkal adhatjuk meg:

IDSJMQRMAU mint a DrawimageQ
IDS_SELECTED: úgy fog kinézni, mintha kiválasztott Gadget Image lenne.
IDS_DISABLED. mintha „Szellem" gadget lenne (minden második pixel háttér színű).
1DSBUSY: egyenlőre nincs ilyen lehetőség, a jövő felhasználásáé.
IDSJNDETERMINATE: mint a fent említetté.
IDSJNACT1VENOKMAL: az ablak bordér gadget-ének.
IDSJNACTIVESELECTED: az ablak bordér gadget-ének.
IDSJNACTIVEDISABLED: az ablak bordér gadget-ének.

Csak az IDS_NORMAL megadásánál vár hagyományos Image struktúra mutatót. Az Objektum Orinetált Intuition (boopsi) által definiálható újabb Image osztályok által definiált objektumok is használhatóak. Természetesen ebben az esetben a felhasználót a használt objektumok lehetőségei kötik. A paraméterei a szokásosak. A state paraméternél adhatjuk meg a rajzolási módokat. A Drawinfo struktúrában a tollak használatát, a felbontást stb. adhatjuk meg. Erre a struktúrára mutató pointert a Sreen struktúra alapján nyerhetünk (lásd a Gadget-ek_2 programokban a lemezen. - csak a Drawlnfora példa).

EasyRequestArgs - Egyszerű alternatíva az AutoRequest()-re. (V36!)

EasyRequest - Mint a EasyRequestArgs. csak az argumentumok külön megadásával. (V36!)

```
LONG EasyRequestRrgs(struct LUindow *IDindow, struct EasyStruct
DO                   AO                       Al
•EasyStruct, ULONG *IDCMP_ptr, HPTR Hrgi);
               A2                A3
```

```
LONG EasyRequest(struct Windouj *Window, struct EasyStruct
DO                AO                      Al
•EasyStruct, ULONG *IDCMP_ptr, RPTR Rrgi, ...);
               A2                A3
```

Ori'set:- 588 ($24C)

Ez a függvény *egy* szimpla Requester létrehozására alkalmas, kinézetben hasonlít a System Requester-ek kinézetére A Requester a Screen Font-jától függő mérelü Requester-t hoz létre. Az EasyRequest függvény az előzőből az alábbiakban valósítható meg. Ezt a függvényt egyébként azért érdemes használni, mert a paraméterek megadása a C-hez közel áll.

EasyReuest(u), es, ip, argi)

```
struct UJíndouj *w;
struct EasyStruct *es;
ULONG *ip;
int argi;
{
    return(EasyRequestflrgs(LU, es, ip, &Rrg1));
}
```

A pédában egy tipikus felhasználást mutatunk be:

```
struct EasyStruct uolumeES =
{
    sizeofístruct EasyStruct),
    0,
    "Uolume Request",
    "Plase insert uolume %s in any driue.",
    "RetrylCancel".                    **
};

#define CRNCEL (0)

Uolume *
getUolume(uolname)
UBVTE *uolname;
{
    Uolume    *uptr;
    Uolume    *findllolume();
    UIDORD    reply;
    ULONG     iflags;

    iflags = IDCMP_DISKINSERTED;

    ujhile(((uptr = fíndUolume(uolname))==NULL) &&
   (EasyRequestfuj, &uolumeES, &iflags, uolname) != CRNCEL))
        7* Ciklus */

     /* megjegyzés: Rmikor benne uagyunk a ciklusban, újra
inicializálnunk kell az iflags érékét. Itt egy korrábbi értékkel uagy
új IDCMPflag uáltozóual. Ha több IDCMP flag-gal kombináljuk min-
dig csak egy IDCMP érékét kapjuk meg.*/

    return(uptr);
}
```

Részletesebben az 1.9.4. részben.

EndRefresh - Vége az optimalizált frissítésnek.

uoid EndRefresh(struct lilindooi *UJindouj, BOOL Comlette);
AO DO

Offset:-366 ($16E)

Ez a függvény az optimalizált frissítési ciklus végétjelenti. Használata, a BeginRefreshO függvény meghívása után. Miután a frissítések szükségessé válnak egy ablakon de nem engedjük azt megváltoztam, a függvény meghívásával az ablakon vissza állítjuk az eredeti állapotot. A függvény paramétereként megadható Complette paraméter jelntése az alábbi. Ez a paraméter BOOL típusú így értéke csak TRUE és FALSÉ lehet. Akkor adjunk meg TRUE értékel, ha a frissítést valóban befejeztük. Erre a multitask miatt van szükség. Csak *az* utolsó meghívásnál adjuk meg a FALSÉ értéket. Példát a BeginRefreshO függvénynél láthatunk.

EndRequest - Törli az aktuális requestert.

uoid EndRequestístruct Request *Request, struct lilindouj •IDindow);
AO Al

Offset:- 120 ($78)

A függvény törli a reqester-t az ablakról és annak csatolása^ is megszünteti az ablakhoz.

Eraselmage - Image törlése. (V36!)

uoid Eraselmagelstruct RastPort *RastPort, struct Image •Image,
AO Al
WORD LeftOffset, WORD TopOffset);
DO Dl

Offset:=- 630 ($276)

Egy Image törlése. Ha a megadott struktúra egy általunk definiált Imagera mutat, akkor az EraseRectO függvényt hívja meg (a törléshez a Layer Backfilljét használja az Image négyzetében. LeflEdgeTopEdgeWidthHeighl).

FreeClass -A MakeClassQ állal keráltboopsi class felszabadítása. (V36!)

```
BOOL FreeClass(struct IClass *ClassPtr);
DO              AO
```

Offset:- 714 ($2CA)

Megpróbálja felszabadítani a MakeClassfJ által létrehozott ciass-t. A függvény TRUE értékkel tér vissza, ha ez sikerült. Ha nem sikerült. FALSE értéket kapunk. Ha a class-unknak az adatait dinamikus memória-felhasználással hoztuk létre, akkor csak a függvény sikeres visszatérési értéke esetén szabadítsuk fel az adatok által foglalt helyet. (clJJserData). Erre a megoldásra láthatunk példát:

```
freeMyClass(cl)
struct IClass *cl;
{
    struct MyPerXlassData *mpcd;

    mpcd = (struct MyPerClassData *) cl->cl_UserData;
    if(FreeClass(cl))
    {
        FreeMemímpcd, sizeof(mpcd));
        return(TRUE);
    }
    else
    {
        return(FHLSE);
    }
}
```

FreeRemember - felszabadítja a memóriának azon részeit, amelyek a Remember struktúrában megtalálhatók.

```
uoíd FreeRemebeHstruct Remeber **, BOOL ReallyForget);
                    AO                  DO
```

Offset:- 408 ($198)

A függvény segítségével azokat a memóriarészeket szabadíthatjuk fel. amelyeket a paraméterként megadható Remember slruktúra taratalmaz. Ilyen struktúrát az AllocRememberO függvénnyel allokálhatunk. A függvény másik paramétere a felszabadítás mértékét határozza meg. Ha az értékét FALSE-ra állítjuk, csak a buffert szabadítja fel. míg ha TRUE. akkor magát a buffert hordozó Remember tag-ot is. Részletesen lásd az AllocRemeber()-nél.

FreeScreénBuffer - A Screen Puffer felszabadítása. (V39!)

uoid FreeScreenBufferístruct Screen *Screen,
AO
struct ScreenBuffer *ScreenBuffer);
Al

Orfset:- 774 ($306)

Felszabadítja az AllocScreenBufferQ függvénnyel lefoglal puffért. Azelőtt kell meghívnunk, mielőtt a Screen-t becsuknánk.

FreeScreenDrawInfo - befejeztük a Drawlnfo struktúra használatát. (V36!)

uoid FreeScreenDraiiiInfo(struct Screen *Screen, struct
AO Al
DrawInfo • DrawInfo);

Offset:- 696 ($2B8)

Befejeztük a Drawlnfo struktúránkkal a munkát. A Drawlnfo-l a GetScreenDrawInfo{) függvény visszatérési értékeként kaphatunk.

FreeSysRequest - A DuildSysRequestfJ függvény által létrehozott system request felszabadítása.

uoid FreeSysRequestístruct UJindouj *UJindow);
AO

Offset:- 372 ($174)

Ez a függvény letörli a BuildRequestQ által lefoglal területet. Ha a BulidRequest() egy Window mutatóval tér vissza, várnunk kell az érkező üzenetekre a message porton. Ha törölni akarjuk a Requester-t. hívjuk meg ezt a függvényt. A V36-os reendszer alatt a OK ha a függvénynek a TRUE vagy a NULL értéket adjuk át. így ez a függvény jól végzi a Requester-ek kezelését, ha használtuk a BuildRequesler() függvényt.

GadgetMouse - Kiszámolja a gadget relatív egér pozíciót. (V36!)

uoid GadgetMousefstruct Gadget *Gadget, struct GadgetInfo *GInfo,
AO Al
WORD *MousePoint);
A2

Ofrset:- 560 ($230)

Determinálja az aktuális tartózkodását az egérpoinlernek. relatívan a gadget bal felső sarkához képest. Tipikus használata egyedül a GM_HANDLEINPUT és a GM_GOACTIVE felhasználói Gadget-ek hook rutinjaiban. Paraméterként a Gadget-re mutató pointeren kívül mutatót vár a gadget info struktúrára, valamint mutatót két WORD-re. vagy a Point típus struktúrára. A kapott erednényt a két WORD-ben vagy a Point típus struktúrába helyezi el.

GetAttr - Attribútum érték bekérése egy objektumból. (V36!)

```
ULONG GetfltMULONG RttrID, HPTR Object, ULONG *StorageP1r);
DO              DO          AO          Al
```

Offsel:- 654 ($28E)

Adatbebvitel egy objektumból a megadott attribútum alapján. A paraméterként megadható AttrID jelenti az adatbevitel tárgyának meghatározását, a bekérendő objektum ezen paraméterei az objektum leírásában megtalálhatóak. A következő paraméter az objektumra mutató pointer, az utolsó paramétere pedig egy pointer a kérdésesre. Visszatérési értéknek vagy a kért adatot kapjuk, vagy O-t. ha a kért adat nem található az objektum osztályában. A lemezmellékleten a Gadget2.0-ban található példa a használatára.

GetDefaultPubScreen - A default Pubscreen nevének bekérése, p/36!)

```
uoid GetDeafaultPubScreentUBVTE * Namebuff );
                                  AO
```

Offeet:- 582 ($246)

Bekéri a paraméterként átadott NamebuH'er-be a default Pubscreen nevét. A paraméternek egy üres puffer-nek kell lennie. A mérete a MAXPUBSCREENNAME konstansban meghatározott. A függvény DO-ban visszaadja a pubscreen pointerét is. de nem lock-olja azt. így az bármikor bezárható. Ezért ezt a pointert ne használjuk fel az ablakunk nyitásakor, mivel egyáltalán nem biztos, hogy az ablak nyílásakor még létezik a pubscreen. Erre a feladatra a LockPubScreen() függvény az ideális. Ezzel a fugvénnyel először bekérjük a DefaulPubscreen nevét, majd a LockPubScreenO függvénnyel lelock-oljuk. és így már megnyithatjuk a „Visitor" ablakunkra.

GetDefPrefs - Az inution default preferencesének elmentése.

```
struct Preferences *GetDefPrefs(struct Preferences *PreBuffer,
DO                                AO
WORD Size);
Al
```

OHset:- 126 ($7E)-

Amikor a rendszer feléled, néhány változót is beállít, amit a Preferences-en keresztül tudunk befolyásolni. Ezeket a preferences-ben beállított adatokat egy struktúrában tárolja. Maga a Preferences struktúra az alábbiak szerint néz ki.

```
struct Preferences
{
    /* a default font magassága */
    BVTE FontHeight;

    /* konstans, hogy melyik portot használjuk nyomtatáskor •/
    UBVTE PrinterPort;

    /• H sors port Baud rate értéke */
    UUJORD BaudRate;
    /* uáltozók az időtartamokra. */
    struct timeual KeyRptSpeed;
    struct timeual KeyRptDelay;
    struct timeual Doubleclick;

    /* Intuition Pointer adatai */
    UUJORD PointerMatriK[POINTERSIZE];
    BVTE KOffset;
    BVTE VOffset;
    UUJORD colorl7;
    UUJORD colori 8;
    UUJORD colori 9;
    UUJORD PointerTicks;

    /* UJorkbench Screen színei */
    UUJORD colorO;
    UUJORD colori;
    UUJORD color2;
    UUJORD color3;

    /• Rz Intuition LJiew pozíció adatai */
    BVTE UiewKOffset;
    BVTE UieujVOffset;
    UJORO UieujlnítK, Uieuilnitv;

    /* R CLI rendelkezésére álló kapcsoló */
    BOOL EnableCLI;

    /* R nyomtató konfigurációi */
    UUJORD PrinterType;
    UBYTE PrinterFilename[FILENHME_SIZE];

    /• fl nyomtatási formátum és minőség config-ja */
    UUJORD PrintPitch;
    UUJORD PrintQuality;
    UUJORD PrintSpacíng;
    UUJORD PrintLeftMargin;
    UUJORD PrintRightMargin;
    UUJORD Printlmage;
    UUJORD Printflspect;
    UUJORD PrintShade;
    UJOHD PrintThreshold;
```

```
    /* H nyomtató papír beállításai */
    UUIORD PaperSize;
    UUJORD PaperLength;
    UUJORD PaperType;

    /* R soros port beállításai */
    UBVTE   SerRLDBíts;
    UBVTE   SerStopBuf;
    UBVTE   SerParShk;

    /* Egyéb */
    UBVTE   LaceUJB;
    UBVTE   Pad[ 12 ];

    UBVTE   PrtDeuName[DEUNRME_SIZE];
    UBVTE   DefaultPrtUnit;
    UBVTE   DefaultSerUnit;

    BVTE    RowSizeChange;
    BVTE    ColumnSizeChange;
    UUJORD    PrintFlags;
    UUJORD    PrintMaKUJidth;
    UUJORD    PrintMaKHeight;
    UBVTE    PrintDensity;
    UBVTE    PrintKOffset;
    UUJORD    uJb_LUidth;
    UUJORD    UJb_Height;
    UBVTE    wb_Depth;
    UBVTE    eKt_size;

};
```

A függvény erről a struktúráról készít másolatot számunkra, a paraméterként megadott méretben (Size). Figyeljünk arra. hogy a méretben megadottaknak eleget tegyen a pufferként megadott változónk is. Visszatérési értékként a default Prefs pointerét kapjuk. Ha ez NULL. nem sikerült a Prefset bolvasnia.

GetPrefs - Preferences beolvasása.

```
struct Preferences •GetPrefs(struct Preferences *PreBuff, WORD Síze);
DO                          AO                          DO
```

Offset:- 132 ($84)

Az aktuális Preferences beolvasása a Size szerinti méretben. A függvény a fentebb említetthez hasonlóan a PreBuffben adja vissza az eredményt. A függvény visszatérési értéke megegyezik a fentebb említettel.

GetScreenData - Másolatot csinál a screen adat struktúrájáról.

```
BOOL GetScreenDatatRPTR Buffer, UUJORD Size, UUJORD Type,
DO                AO          DO          DI
struct Screen *Scr);
Al
```

Offset:- 426 ($1AA)

Ezzel a függvénnyel másolatot tudunk készíteni egy screen típusáról. A másolat a buffer-be történik, és a Size-ben meghatározott méretben az Ser által mutatott screen-ről. A függvény a TRUE értékkel tér vissza, ha a művelet sikeresen végrehajtódott, és FALSE értékkel, ha a screen típus 'type' értéke nem volt megnyitva. A V36-os rendszereken ez a függvény nem használható az új Screen ID módok betöltésére. Erre a LockPubScreenQ függvény meghívása után a Graphics Libraryból a GetVPModelDQ függvényt használhatjuk. A függvénynek a Type argumentumaként a screen típusa adható meg (WORKBENCHSCREEN. CUSTOMSCREEN ...).

GetScreenDrawInfo - A rajzolási információ pointerének bekérése. (V36!)

struct Draujlnfo *GetScreenDrawlnfo(struct Screen *Scr);

Offset:- 690 ($2132)

Visszaadja a screen Drawlnfo struktúrájának pointerét. Ez az adat csak olvasható! Paraméterként csak a kívánt létező screen pointerét várja. Meg kell hívnunk a FreeScreenDrawInfofJ függvényt, ha már befejeztük a függvény visszatérési értékével a munkát. Ha a függvén}' nyilvános sereen-re hívjuk meg. ne feledkezzünk meg a sereen-t lock-olni a LockPubScreenO függvénnyel. A függvény hibája, hogy nem értesülünk arról, ha a screen Drawlnfója megváltozott.

HelpControll - engedélyezése/tiltása a Gadget Help-nek. (V39!)

uoíd HelpControKstruct Window *UJin, ULONG flags);

Offset:- 828 ($33C)

Ezzel a fügvénnyel ki ill. be tudjuk kapcsolni a Gadget Help-et az ablakunkon. Paraméterként annak az ablaknak a pointerét várja, amin az eseményt végre akarjuk hajtatni. A flag-oknáll a HC_GADGETHELP-et adhatjuk meg vagy nullát.

InitRequest - iníciali/ál egy Requesler strukturál.

```
uoid InitRequesterfstruct ílequcstt'r *Req);
                          AÜ
```

OlTset:- 138 ($8A)

Ez a függvény a korai időkben kerüli a/ Intuilion-ba. A függvénynek nieţí kell adni egy Requester pointerét, amit mi már a kívánt értékekkel felţöltöttünk. Elég csak azokat az értékeket megadnunk, amelyeket a Requester-bcn nekünk fontosak. A többi inícializálásál elvégzi a függvény helveüünk. Nem nagyon használják ezt a függvényt, az újabb Intuitionokban csak a kompatibilitáséri maradi benne.

IntuiTextLength - visszaadja (pixel szélességben) az InluiTexl hosszát.

```
LONG IntuiTentLenght(struct IntuiTent *It);
DO                        AO
```

Offset:- 330 ($14A)

A függvénynek egy pointert kell megadnunk arra az !ntuiText-re. amelynek a hosszára vagyunk kíváncsiak. A hossz értékét pixelben kapjuk ineí?.
Természetesen a kiírandó szöveg függvényében, an,elvre a használt Iont befolyással bír.

Intuition - a függvényt szándékosan nem publikálják.

```
uoid Intuitionístruct InputEuent *iEuent);
                  AO
```

OlTset:- 36 ($24)

A függvényről szándékosan nem jelentetnek meg semmiféle dokumentációt. Valószínűleg nem akarják, hogy használjuk.

ItemAddress - Visszaadja a Menü Ilemének címét.

```
struct MenuItem *ItemflddrcssCstruct Menü *, UWORD MenuNŰmber);
```

OlTset:- 144 ($90)

Ez a rutin visszaadja az aklivizáll menü slrip alapján a menii iteniének címét, amellyel a nienunumber-ből nyer. Tipikus felhasználása az ablakhoz csatlakozó menük értékelésekor. A megadható paraméterekre az alábbi összefüggések mondhatók el: ha a MenuNumber-kent megadott érték egyenlő a MENUNULL konstanssal, akkor a visszaidott érték ni'ndig NULL. Ha nem egyenlő, akkor megkapjuk az érvényes menü számát es az item s/amat is. valamint ha a menühöz csatlakozott SubMenü is. akkor annak a számát is.
Használatára a lemezen találunk példát a Menük könyvtárban.

LendMenus - kölcsön adja az ablak menüit egy másik ablaknak. (V39!)

uoíd LendMenusfstruct UJindow* fromUlin, struct Window *tolUin);

Offset:- 804 ($324)

Kölcsön adja a IVomWin ablak menüiL a toWin ablaknak. Ha a toWin ablakra mutáló pointer NULL-ra mulat, a fromWin menni megszűnnek.

LockIBase - Lezárja az intuitionBase szemaforjai használatát.

ULONG LockIBasetULONG LockNumber);
DO DO

Ofrset:-414($19E)

A függvény zárolja az Intuition által használt külső szemaforok használatát is. Ameddig a szemaforok zároltak, az Intuitin (és néhány graphics. layers. dos. valamint néhány magas szintű rendszerművelel is) működésében megáll.

A zárolás megszüntetését az UlockIBase() függvény végezheti el. Visszatérési étrékként egy olyan értéket kapunk, amit majd az UnloclBaseO függvénynek adhatunk át.

Paraméterként a zárolni kívánt szemafor számát várja. Ha a LockNumber paraméternek a nulla értéket adjuk, akkor minden szemafor zárolódik. A függvény hatása alatt nem hívhatunk meg másik zároló függvényt is, mint amilyen a I^ayer-böl a Layerlnfo zárolása.

LockPubScreen - Meggátolja a Pubscreen bezárását. (V36!)

struct Screen *LockPubScreen(UBYTE •Name);
DO AO

Offset:-51O($1FE)

Nem engedi a nyilvános screent bezárni. így nyithatunk a screenre 'VFsitor' abalkot. Ahhoz, hogy egy vendég ablakot nyithassunk az alábbi műveleteket kell elvégeznünk:

```
LockPubScreenO
...Infomációk bekérése a screenröl ...
OpenLUindoLuO a nyiluános screenre
UnlockPubScreenO
... az ablakunk használata ...
CIoseLUindouiO
```

Ha megnyitottuk az ablakunkat a pubscreenre. már nem szükséges zárolva tartani azt. hiszen az ablak megakadályozza annak becsukását. Ha a paraméterként a "Workbench"-et adjuk meg. vagy NULL értékkel hívjuk, a függvény a default Pubscreenre adja a visszatérési értéket. Ha a workbench screen sem nyitott automatikusan megnyitja azt. A visszatérési értéke a Lockoll screen. vagy NULL ha nem sikerült.

LockPubScreenList - megakadályozza a rendszer listájának megváltoztatását (V36!).

```
struct List *LockPubScreenList(uoid);
DO
```

OiTset:- 522 ($20A)

Zárolja a rendszer PubScreen listáját, ameddig gyorsmásolatot készíthetünk róla. A függvény ideális egy Public Screen Manager program írásához. A visszatérési érték egy pointer a rendszer PubScrennNode struktúrájára. Ajánlott a használatát összekötni a LockIBaseQ függvénnyel.

MakeClass - létrehoz és inicializál egy boopsi osztályt (class) (V36!)

```
struct IClass *MakeClass(UBVTE *ClassID, UBVTE *SuperClassID, struct
DO                       AO             A1                     A2

IClass* SuprClassPtr, ULUORD InstanceSize, ULONG Flags);
                      DO                   D1
```

Offset:- 678 ($2A6)

Ez a függvény létrehoz egy publikus. vagy magán boopsi osztályt. A SuperClass egy definiált másik boopsi Class: az összes class-nak a "rootclass"-ból leszámaztahatónak kell lennie. A SuperClass lehet nyilvános, vagy saját. Ha nyilvános, nekünk kell gondoskodnunk a nevéről és ID-jéről (nem használhatjuk a Commodore által már előr.e definiált nevel és ID-t.).

Miután befejeztük a létrehozását a nyilvános class-nak. meg kell hívnunk az AddClassf) függvényt. A visszatérési értéke a létrehozott class struktúrájára fog mutatni. Ha visszatérési érték NULL. az azt jelentheti, hogy nem volt elég memória a létrehozásra vagy már ezen a néven létezik egy osztály, vagy nem talája a nyilvános SuperClasst. A paraméterei a következők: az első a ClassID. amely NULL értékű ha a létrehozandó osztály privát. Egyébként a Nevét/ID-jét artalmazza. SuperClassID a neve vagy IDjaa SuperClassnak. és NULL ha privát. SuperClassPtr pointer a privát superdassunkra. ha a SuperClasslD nem NULL. Az InslanceSize a pélcla najíysága a classunkriak. Előtte definiálnunk kell a SuperClass objektumunkat. Használatára nézzünk egy rövid példát;

```
/* első ojektum példa adat definíció a classunknak. */
struct MyInstanceOata
{
    ULONG mid_SomeData;
};
```

```
/* néhány tábla megadása az összes objektumnak. */

UWORD myTablel] =
{
    5,4,3,2,1,0
};

struct IClass*
initMayClass(uoíd)
{
    ULONG _saueds myDispatcherO;
    ULONG hookEntryü; /* asm-ről C interface */
    struct IClass *cl;
    struct IClass *MakeClass();

     if(cl = MakeClass(NULL, SUPERCLRSSID, NULL /* fl superclass
publikus */
            sizeoflstruct MyInstanceüata), 0))
    {
        /* inicializáljuk a cl_Dispatcher Hookot •/
        cl->cl_Dispatcher.h_Entry = hookEntry;
        cl->cl_Dispatcher.h_SebEntry = MyDispatcher;
        cl->cLDispatcher.h_Data = (uoid •) OHFHCE; /* Nem használt •/

        cl->cl_UserData = (ULONG) myTable;
    }
    return(cl);
    )
```

MakeScreen - ay. Intuilionba integrált MakeVPortf).

```
LONG MakeSreenfstruct Screen *Scr);
DO                AO
```

Ofrset:- 378 ($17A)

A függvény egy a InUiiliónban megvalósított MakeVPortJ). Ez jó megoldás a custom screen viewPorLjának elkészítésére. Az alábbi műveleteket valósítja meg meg a függvény:

- várakozik, ameddig az Intuition a View Struktúrát nem kezdi el változtam.
- bállítja a view módot vagy korrigálja azt ha az (látható) inlerlased screenen.
- meghívja a MakeVPort()-ot ami átadja az Intuitionnak és a screenünknek a ViewPortot.
- felengedi az Intuition Viewet.

A függvény meghívása után szükség van arra.hogy meghívjuk a RethinkDisplayO függvényt is. hogy megvalósuljon az új ViewPorta Cusrom Screenünkön. és az Intuition képernyőn. Használata után az Intuition részére de-

terminálnunk kell a megváltozott globális interlace miatt szükséges a viewportot újra készítettni a RemakeDisplayO függvény meghívásával. A függvénynek visszatérési értéke csak V39! rendszertől van. Ha a függvény végrehajtásakor hiba keletkezett ezt a visszatérési értékben adja tudtunkra Tehát ha O-át ad vissza akkor minden OK.

ModifyIDCMP - megváltoztaja az ablak IDCMP-flag-jait.

```
BOOL ModifyIDCMP(struct Window *UJin, ULONG IDCMPFlags);
DO                AO                  DO
```

Offsel:- 150 ($96)

A függvény a megnyitott ablak IDCMP flagjait képes megváltoztatni az IDCMPFlags-ban megadottakra. Az IDCMP flagok jelentéséről részletesebben a könyv 1.7.1 részében található. A függvénynek csak V37 rendszerlől van visszatérési értéke amely BOOL típusú. NULL értékkel tér vissza, ha nem sikerült neki a MessagePortot létrehoznia.

ModifyProp - Megváltoztatja a Proporcionális gadget paramétereit.

```
uoid ModifyPropístruct Gadget *Prop, struct UJindow '"Iiin,
                          AO                    Al
struct Request *Req, UWORD Flags, UWORD HorizIPot, UUJORD UertPot,
A2                   DO           D I              D2
UUJORD HorizBody, UUJORD UertBody);
D3                D4
```

Offset:- 156 ($9C)

Módosítja a proporcionális Gadget paramétereit a paraméterként magadott értékekre. A folyamat során újra számolódik a Prop. Gadget kinézete (Knob stb.), és újra kijelzésre kerül, tehát frissító'dik. Ha a Gadget nem Requester Gadget. akkor a Req paraméternek NULL adható meg. A függvény egyéb paraméterei megegyeznek a Porplnfo struktúra azonos nevű mezőinek jelentésével (lásd 1.6.3. részben.). Ha nem akarjuk, hogy az ablakon használt összes Gadget-et frissítse, akkor használjuk inkább a NewModifyPropO függvényt.

MoveScreen - a screen mozgatása.

```
uoid MoueScreenIstruct Screen *Scr, UJORD DeltaX, WORD DeltaV);
                    AO                  DO          D I
```

Offset:- 162 ($A2)

; A megnyitott, már létező screen scrollozása a megidolt helyre (DelataX. DeltaY). A függvényre példát a lemezmellékleten a képernyők és Ablakok részken n> Ahinitok >« képernyők Demo-han láLhatunk. A me<lacioU mozgatási koordináták reletívak a Screen LeftEdge és TopEdge pontjaihoz, és az elmozdulás mértékét tartalmazzák.

MoveWindow - az ablak mozgatása.

```
uoid MoueùJindowístruct Window *UJin, UUJORD DeltaK, UUJORD DeltaV);
```

Offset:- 168 ($A8)

Ez a nitin egy üzenetet küld az Intuitionnak. amelyben felszólítja azt az ablak áthelyezésre a paraméterként megadott helyre. A paraméterként megadott értékek nem abszolút koordináták, hanem relatívak, az ablak aktuális helyéhe/, képest. Ha újab rendszer alatt (V36!) használjuk, a művelet befejezettségéről egy IDCMP_CHANGEWINDOW üzenettel szerezhetünk tudomást. A függvény használatára a fentebb említett példaprogramban találunk példát.

MoveWindowInFrontOf - Az ablak egy másik ablak elé helyezése. (V36!)

```
uoid MouelUíndoujInFrontOf(struct IDindoui *LUin, struct UJindoui
                           AO                      Al
•behindUJin);
```

Ofrset:- 48O($1EO)

Az ablakot a behindWinben megadott ablak elé helyezi. A függvény egy Intuitionban megvalósított MoveLayerlnForntOfO függvény. Nem használható Backdrop ablak esetén.

NewModifyProp - modifyPropO. de szelektív frissítéssel.

```
uoid NeiuModifyProp(struct Gadget *Prop, struct UJindoiu *Illin, struct
                          AO                  Al                  A2
Reqester *Req, UUJORD Flags, UUJORD HorízPot, UUJORD UertPot,
               DO            Dl              D2
UUJORD HorizBody, UUJORD UertBody, UUJORD NumGad);
D3                D4               D5
```

Orfset:- 468{$1D4)

A függvény a proporcionális gadget paramétereit változtatja meg a paraméterként kapott értékekre. Azzal a különbséggel a ModifyPropO függvény hez képest, hogy nem frissíti az összes gadget-et. csak a NumGad paraméterben megadott számút a gadget listában. Ha ezt az értéket -l-nek adjuk meg. a függvény úgy viselkedik majd. mint a ModifyPropO. Azaz a Gadget-ek frissítését a lista végéig hajtja végre. A paraméterként átadott értékek jelentésben megegyeznek a Proplnfo struktúrában megadható mezönevek jelentésével.

NewObjectA- létrehoz egy objektumot a class-ból. (V36!)

NemObject - egyenként paraméterezhetö változata a NewObjectAO-nak. (V36!)

```
RPTR NeuiObjectnístruct IClass *class, UBVTE * ClassID,
DO                     AO                 A1
struct TagItem *tagLíst);
A2

RPTR NeujObjectístruct IClass *class, UBVTE * ClassID,
DO                    AO                  AI
ULONGTagi,  ...);
DO
```

Offset:- 636 ($27C)

A függvények segítségével létrehozhatunk egy taoopsi objektumot a Classban definiált típusok közül. Ha a eláss pointere NULL-ára mutat, akkor a ClassID használatos. Visszatérési értékként egy pointert kapunk a létrehozott objektumra. A függvényre példát láthatunk a lemezmellékleten található Gadget2.0ás könyvtárban.

NextObject - megismételtet egy Objektumot az Exec listában. (V36!)

```
RPTR NeKtObject(RPTR objectPtrPtr);
DO              AO
```

Offset:- 666 ($29A)

A függvény csak a boopsi alkalmazásokra használható. Amikor a kollektíven állítjuk a boopsi objektumokat, az Execlistán lehívódik az OM._ADDMEMI3ER metódus. (Csak) Ezzel a függvénnyel vagyunk képesek helyrehozni. Nézzünk példát a használatára :

```
/* itt uan az OM_DISPOSE eset a class dispatcherből */
case OMJQISPOSE:
    /* fl Dispose tagok */
    object_state = mgdata->md_CollectionList.lh_Head;
    Luhile(member_object = NentObject(&object_state)I
    {
        DoMethod(member_object, 0M_REM0UE); /* törlése a
                                               listából. */
        DoMethodFKmember, msg); /* a dísposéuel egyült
                                   ualó átadása */
    }
```

A visszatérési értéke pointer mindegyik objektumra amelyik a listában következik, í 1l. NULL ha már nincs több.

NextPubScreen - a következő PubScreen azonosítása a ciklusban. (V36!)

UBVTE *NextPubScreen(struct Screen *Screen, UBVTE *NameBuff);
DO AO Al

• Orfset:-534 ($216)

Visszaadja a következő PubScreent. hogy a visitor ablak mindig át tudjon ugrani a következő PubScreen-re. (lásd PointerEyes program.) Paraméterként egy pointert vár a Pubscreenre. amelyre az ablakot nyitottuk, vagy NULL. ha nincs Pubscreen pointerünk. A NameBufferként megadott paramétert az Intuition tölti fel a következő Pubscreen nevével a ciklusban. Ennek a változónak a mérete MAXPUBSCREENNAME+1 karakterből állónak kell lennie.

ObtainGIRPort - a RastPort beállítása a felhasználói gadgetnek. (V36!)

struct RastPort *ObtaínGIRPort(struct GadgetInfo *GInfo);

Offset:- 558 ($22E)

Beállítja a RestPort-ot a felhasználói gadget hook rutinjának. Csak felhasználói gadgetnél használjuk. Ezt a függvényt mindannyiszor meg kell hívnunk, valahányszor a hook rutinnak a gadget megjelenítésének végrehajtásakor szükséges, és vele együtt használandó a ReleaseGIRPort(). Ha a hook függvény visszaadja a RastPort pointert például GM_RENDER. nincs szükség az ObtainGiRPortO függvény meghívására. Paraméterként a függvény egy pointert vár átadásra minden felhasználói gadget hook funkciójára. Visszatérési értékként egy mutatót kapunk a gadger megjelenítésére használt RastPort-ra. Ha ez NULL. nem történt megjelenítés.

OffGadget - A megadott gadget kikapcsolása.

uoid OffGadgetfstruct Gadget *Gad, struct LUindow *UJin,
AO Al
struct Requester *Req);
A2

Ofrset:- 174 ($AE)

A függvény a paraméterében megadott gadget kikapcsolását végzi. így azt az user nem aktivizálhatja. A függvény tulajdonképpen a gadget GFLG_DISABLAED flag-ját állítja be. Ezzel együtt a gadget „szellem"-szerü megjelenéséről is gondoskodik. Ha a gadget Requester-hez csatlakozik, a gadget-nek a G'1TP_REQGADGET flag-jának beálítottnak kell lennie.

OffMenu - a menü vagy menüitem kikapcsolását végzi.

```
uoid OffMenuístruct UJindow *UJin, UUJORD MenuNumber);
                AO                  DO
```

Offset:- 180 ($134)

Ez a függvény kikapcsolja a menüben az Item-el. vagy a Subltem-el ill. a menüt. Ezt a MenuNumber megadása alapján végzi el .A kikapcsolt menü nem választható. (Részletesen lásd. 1.7. részben, ill. a lemezemelléklelen.)

OnGadget - bekapcsolja a megadott gadget-et.

```
uoíd OnGadgetIstruct Gadget *Gad, struct UJinclow *Ii)in,
                AO                  AI
struct Requester *Req);
A2
```

Orfset:- 186 ($BA)

Bekapcsolja a gadget-et. amit a GFLGJ3ISABLED flag beállításával, vagy az OffGadget() függvény hatására kikapcsoltunk. így az user számára használhatóvá lesz. A függvény megszünteti a gadget „szellem" kinézetét is.

OnMenu - bekapcsolja a Menüt/MenüItemet/SubMenüt.

```
uoid OnMenuístruct UJindow *Uiin, UUJORD MenuNumber);
                AO                  DO
```

Offset:- 192 ($C0)

Visszakapcsolja a Menüt vagy Menültem-et ill. Subl temet, a paraméterként átadott MenuNumber alapján, így az user számára elérhetővé válik.

OpenIntuition — szándékosan nem publikált.

```
uoid OpenIntuition(uoid);
```

Ofíset:- 3O($1E)

A függvényről szándékosan nem publikálnak semmiféle dokumentációt.

OpenScreen - megnyit egy Intuition Screen-t.

```
struct Screen *OpenScreen(struct NeiuScreen *NeiuScreen);
DO                                AO
```

vagy

```
struct Screen *OpenScreen(struct ExtNewScreen *EKtNeujScreen);
```

Ofíset:- 198 ($C6)

A függvény megnyit egy Intuition Screen-t. amely tulajdonságait a NewSrreen struktúrából nyeri. Az újabb rendszerű gépeken (V36!) a Soreen megnyitásának két útja is van. Az egyszerűbb és újabb az OpenScreenTag-ListO függvény használata, ahol a leendő screen paramétereit kényelmesen egy Taglist megadásával érhetjük el. Vagy a hagyományos eljárás, ha ezt a függvényt nem a NewScreen struktúra inicializálása után annak mutatójának átadásával hívjuk meg. hanem egy ExtNewWindow struktúrával hívjuk meg. A könyv 1.6. fejezetében mindkettőről részletesebb leírás taláható. és ezt kiegészítve a lemezmellékleten mindkettőhöz társul némi példázat. Az alábbiakban azonban bemutatunk egy példát arra. amikor olyan screen-re **kell** ablakot nyitnunk, amely nem PubScreen. Ezt a megoldást használja a Dos is. amikor a Disk Full" szerű üzeneteket küldi, ez a részlet a következő:

```
#include "libraríes/dosextens.h"

    ...
struct Process *process;
struct UJindoiu *UJin;
HPTR  temp;

    ...
process = (struct Process •) FindTask(NULL);
temp = process->pr_UJindoLuPtr; (a régi érték elmentése)
process->pr_WindoujPtr= (RPTH) UJin;
/* használjuk a mutatót ablak nyitásra a screen-en. •/

    ...
    /* R saját kódunk helye. */

    ...
process->pr_UJindoii)Ptr = temp;
(uissza állítjuk a uáltozót mielőtt becsukjuk az ablakot. */
Closeli)indoui(üJin);
```

OpenScreenTagList - OpenScreeQ aTagltem tömb bővítéssel. (V36!)

OpenScreenTags - mint fent. csak egyenként megadható tagokkal (V36!)

```
struct Screen *OpenScreenTagList(struct NewScreen *NeujScreen,
DO                                AO
struct TagItem *TagItemek);
Al
```

```
struct Screen *OpenScreenTags(struct NewScreen *NeujScreen,
ULONGTagi, ...);
```

Orfset:- 612 ($264)

Mindkét függvény egy Intuition screen megnyitását végzi. A paramétereinek megadása opcionálisan lehet NewScreen struktúra mutató, avagy mutató a Tagi temeket tartalmazó tömbre. A tömb utolsó elemének aTAG_ENDnek kell lennie. A második esetben ez természetesen nem tömböt jelöl. A függvény részletesebben a könyv 1.6 fejezetében található.

OpenWindow - Egy Intuition ablak megnyitása.

```
struct UJindow •OpenLUindoiu(struct NeiuUJindoui *NewUJindoLu);
DO                            AO
```

Orrset:- 204 ($CC)

Egy Intuition ablak megnyitása a NewWindow struktúrában megadott értékek alapján. A könyv 1.6 fejezetében részletezve a megadható mezők és jelentésük.

OpenWindowTagList - intuition ablak megnyitásaTagitem bővítéssel. (V36!)

OpenWindowTags - mint fent. de a tagok egyesével való megadásával. (V36!)

```
struct Luindow *OpenlilíndoujTagList(struct NcLuUJíndow *NLU,
DO                                    AO
struct TagItem *);
Al
```

```
struct UJindouj *Openli)indoujTags(struct NewWindow *NW,
ULONGTagi, ...);
```

OlTset:- 606 ($25E)

Mindkét függvénnyel Intulton ablak nyitható, csak az első esetben az ablakokat meghatározó Tag-listát egy tömb mutatójával adhatjuk meg. míg a második esetben a tag-okat kulon-kulon aciiiajiuK meg. MintiKei ruggveny paraméterei opcionálisak. Visszatérési értékként természetesen a menyitott ablak pointerével tér vissza ha sikerült, egyébként NULL.

OpenWorkBench - megnyitja a Workbench Screen-t.

ULONG OpenLDorkbench(uoíd);
DO

Offset:- 510 ($D2)

A függvény újra megkísérli megnyílni a Workbench-et. Ha bármi nem sikerül, NULL értékkel tér vissza, egyébként a visszatérési értéke a Workbench srreen-re mutató pointer lesz. Ha a Wb már nyitott volt. annak rímével tér vissza.

PointImage - leellenőrzi, hogy a pontot tartalmazza-e az Image. (V36!)

BOOL PointInImageístruct Point Point, struct Image *Image);
DO DO AO

Offset:- 624 ($270)

Ellenőrzése, hogy az Image tartalmazza-e a pontot. A paraméterként megadott pont egy Long. amely két word-ot tartalmaz, az X és az Y koordinátákkal. A felső word-ben helyezkedik el az X koordinátát tartalmazó érték. Ha a második paramétert NULL-nak adjuk meg, akkor a visszatérési értéke mindig TRUE. Ez a paramétre egy mutató a normál Image-re.

PrintIText - IntuiText nyomtatása az Intuition képernyőre.

uoid PrintITextístructflastPort *RPort, struct IntuiTeKt *IT, **UUORD**
AO A1 DO
LeftOffset, WORD TopOffset);
ül

Offset:- 216 ($D8)

A függvény a paraméterként megadott IntuiText nyomtatását végzi, a szintén paraméterként megadott RastPort-ra. A paraméterként megadott IxflOffset és TopOffset az írás megkezdésének ofTszet-jét tartalmazhatják (a koordinátákat az IntuiText struktúrában adhatjuk meg.) Részletessebben az 1.7.2.1 részben.

PubScreenStatus -A státusz flag megváltoztatása a Pub Screenben. (V36!)

UUJORD PubScreenStatusístructScreen *Scr, ULUORD StatusFlags);
DO AO DO

Orfset:- 552 ($228)

Megváltoztatja a megadott nyilvános screen-nek a státusz flag-jait. Ne használjuk ezt a függvényt, ha a.programnak nem saját screen-jére hívjuk meg. és legfőképpen a Workbench screen-re ne hívjuk meg. Az első paraméterében a nyilvános screen pointerét várja, míg a másik paraméterben a flag-ot (pl.: PSNF_PR1VATE: a screen-re nem nyithatóak „visitor" ablakok). A visszatérési értékeként a legalsó bit használatos egyenlőre, a többi későbbi felhasználásra fentartva. Ez a bit is azt jelzi, ha a screen publikus volt-e avagy nem.

QueryOverscan - a standard oversoan régió lekérdezése. (V36!)

```
LONG QueryOuerscantULONG DisplayID, struct Fiectangle *Rect,
DO              DO          AO              D1
WORD OScanType);
```

Offset:- 474 ($IDA)

A függvény eldönti a kérdéses felbontáshoz tartozó Oversran területet. .1 megadott modelD által kijelölt felbontásban. A ModelD megadásáról részié tesen lásd az 1.5.4.3 részben vagy a <graphics/displayinfo.h>-ban. Az Oscan típusok megadható értékei:

OSCAN_TEXT: a preferences-ben megadható felhasználói oscan típus. A típus mindig megegyezik a STDSSCREENWIDTH és a STDCREENHEIGHT értékkel, a Left/Top mindig 0,0.
OSCAN_STANDARD: csak a monitoron látható felbontás, amit az user preferences-ben beállíthatunk.
OSCAN_MAX: a maximális méret, amire a képernyő képes és a legkisebb.
OSCANJVIDEO: az abszolút maximális felbontás, amit a graphics.library képes meghajtani.

A visszatérési értéknek adott változó akkor nulla, l^a nem létezik a megadott ID-jű Monitor specifikáció. Az Overscan felbontásokat a Rect-ként megadott struktúrába helyezi el. A megkapott értek közvetlenül átadható az ExtNewScreen struktúrának, az SA_SlclDClip-ként.

RefreshGadgets - frissítése (újrarajzolása) a Gadget-eknek.

```
uoid RefreshGadgets(struct Gadget *Gad, struct LUindow *IUín,
                    AO                  A I
struct Requester *Req);
A2
```

Offset:- 222 ($DE)

A megadott Gadget-ek frissítését végzi a megadott ablakon vagy requeí>ter-en. A Gadget-eknek természetesen láncolva kell lenniük. Ha a Gadget-ek kinézetét akarjuk megváltoztani (lecserélni a Texteket, Border-eket, vagy Image-ket). akkor először töröljük azt a gadget-listát (RemoveGListO). majd ha lecseréltük a szükséges elemeit, állítsuk vissza a listát (AddGListO). Ezek után frissítsük a képernyőt (ablakot), és ha szüséges hívjuk meg a függvényt. Akkor is így járjunk el. ha csak a GFI.G_SELECTED flag-ot módosítottuk csak. A szükséges gadget-et töröljük a listából RemoveGadgteí). majd ha lecseréltük a GFLG_SELECTED flag-ot. fűzzük vissza a listába az AddGadgetO függvénnyel, és hívjuk meg a függvényt.

RefreshGList - a Gadget Lista frissítése (újrarajzolása) a megadott számú gadget-től.

```
uoid RefreshGLisUstruct Gadget *Gad, struct Ulindouj *LUin,
                    AO                      Al
struct Requester *Req, LUORD NumGad);
A2                      DO
```

Offset:- 432 ($160)

A gadget-lista frissítése a megadott számú gadget-től. Ha a megadott érték -1. a gadget a lista végéig frissít (a lista végét a NULL jelzi a gadget->Nexl-Gadget mezőjében). Működését tekintve ugyanaz, mint a RefreshGadgetsO függvény.

RefreshWindowFrame - az Intuition utasítása az Ablak keretének frissítésére.

```
uoid RefreshLUindoiuFrameístruct LUindow *UJindouj);
                              AO
```

Orrset:- 456 ($1C8)

Az ablak keretének frissítése, benne foglalva a fejléc és az összes gadgetet is az ablakon.

ReleaseGIRPort - realizál egy felhasználói gadget RastPortot. (V36!)

```
uoid ReleaseGIRPortístruct RastPort *RPort);
                          AO
```

Offset:- 564 ($234)

Összefüggő függvény az ObtainGIRPorlfJ-al. Ez a függvény realizálja az Intuiton részére a Gadget-ek RastPort-ját.

RemakeDisplay - a teljes Intuition display újrakészítése.

```
LONG RemakeDisplay(uoid);
DO
```

Ofíset:- 384 ($15C)

Ez a függvény a teljes ViewPort-ot újrakészíti. Visszatérési értéke csak V39-es rendszertől van, és 0 ha sikerült. Hiba esetén a visszatérési értéke nem nulla. A hibákról részletesen a <graphics.library/ MakeVPorlO ill. MrgCopl) függvényeinél. A függvény megegyezik a MakeScreenO függvénnyel csak minden screen-re. és meghívja a benne megegyező függvényt a Rethink-DisplayO függvénnyel.

RemoveClass - érvénytelenítése egy boopsi classnak. (V36!)

uoid RemoueClassístruct IClass *classPtr);
AO

Offset:- 708 ($2C4)

Érvénytelenít egy nyilvános osztályt (class) a nyilvános elérésüek közül. A függvény belső osztály érvénytelenítésére nem használható. Paraméterként a MakeClassQ függvény által visszaadott érték adható át, mint az érvénytelenítésre kijelölt osztály neve.

RemoveGadget - Gadget törlése az ablakról.

UWORD RemoueGadgeUstruct UJindOLU *LUin, struct Gadget *Gad);
DO AO Al

Offset:- 228 ($E4)

A paraméterként megadott gadget (Gad) törlése a megadott ablakról (Win). Visszatérési értékként a törölt gadget pozícióját adja. Ha gadget nem törölhető, az ablakról az érték -1. Az érték akkor is -1. ha az ablakról a 65535 gadget-et töröltünk. A V37 rendszertől ha a törlendő gadgel aktív, megvárja míg azt inaktivizálják. s utána törli.

RemoveGList - a megadott helytől gadget törlése a listából és az ablakról.

UlÜORD RemoueGListlstruct Window *Win, struct Gadget *Gad,
DO. AO Al
UUJORD NumGad);
DO

Offset:-444 ($1BC)

A Numgad-ban megadott számú Gadget-eket töröl a listából a megadott ablakon. Természetesen ha a gadget Requesteré. a gadget-nek a GTYP_REQGADGET flag-jának beállítottnak kell lennie. A V36-os rendszertől az utolsó törlendő gadget-et a gadget NextGadget NULL értéke jelzi. V37-IÖl ha a törlendő gadget aktív, megvárja míg az user deaktiválja azt. és csak azután törli. Ha a NumGad értékének -1-et adunk meg, a lista végéig töröl.

Visszatéréskor a törölt gadget pozícióját adja. Ha nem szerepel a gadget a listában, vagy egyéb hiba lép fel. -1-et ad vissza.

ReportMouse - az InLuition tájékoztatása* hogy jelezze az egér mozgását.

```
uoid ReportMouse(B00L Boolean, struct LUindoiu *U)in);
                 DO            AO
```

Offset:- 234 ($EA)

Néhány fordító a függvény paramétereinek megadását fordított sorrendben várja. Tehát, vagy így ahogyan fent megadtuk, vagy ReportMouse(struct Window* Win. (ULONG) I3OOL boolean)-ként. Az első változat az. amelyet a Commodore az Amiga.lib-ben megad. A függvény automatikusan elvégzi a Win->Flags I = WFLG_REPORTMOUSE; és a Window->Falgs &=~WFLG_REPORTMOUSE műveleteket. A rendszer védelmében ez használja a Forbidf) **1ll.** PermitO függvényeket is. A függvény paramétereit a fent magadott regiszterekben várja assembly használatkor. A Dool paraméterben az egér üzenteinek küldését kapcsolhatjuk be (TRUE) vagy ki (FALSÉ) magadásává.

Request - aktivizál egy requestert.

```
BOOL Request(struct Requester *Req, struct LUindoiu *LUin);
DO           AO                     A1
```

Orfset:- 240 ($F0)

A függvénnyel a Req-ben megadott requester-t jeleníthetjük meg az ablakon (Win). Ez a rutin átugorja az ablak IDCMP_REQUVERIFY flag-ját. Ha a Requester-t sikerült megnyitnia, a visszatérési értéke TRUE. egyébként FALSÉ. Az ablakhoz maximum 8 Requester csatolható egyszerre. Részletesen az 1.9.3.2.2 részében.

ReSetMenuStrip - újra csatolja a menüket az ablakhoz. (V36!)

```
BOOL ResetMenuStripfstruct LUindouj *LUín, struct Menü *Menu);
DO                   AO                    Al
```

Offset:- 702 ($2BE)

A függvény egyszerűbb és gyorsabb útja a SetMenuStripO függvény által elvégzett menü újra kirakásnak. Ezt a függvényt csak abban az esetben használhatjuk, ha az ablakhoz már volt csatolva a menü (SetMenuStripO). A függvényt csak akkor használhatjuk továbbá, ha a menün' nem eszközöltünk akkora változtatást, hogy a kinézetét megváltoztassa. Tehát, csak az alábbi változtatások megengedettek: a CHECKED flag állítása, az 1TEMENABLED flag állítása. Ha mást is meg változtatunk, használjuk a SetMenuStripO függvényt. A menü ezen flag-jai változtatásának a miként je:

- OpenWindowO
- SetMenuStripO
  /* A változtatást kieszközlő rész */
  - ClearMenuStripO
  - változtatása a CHECKED vagy az ITEMENBLED riag(ok)nak.
  - ResetMenuStripO
  /* Vissza a normál működés hez. */
- ClearMenuStripO
- CloseWindowO

A függvény visszatérési értéke mindig TRUE.

RethinkDisplay - az egész Intuition display manipulációja.

LONG RethinkDísplay(uoid);
DO
Offset:- 390 ($186)

Az Intuition display-en végrehajtott globális visszaállítás. Magába foglalja az összes ViewPorl-ot és annak relációjában az egész display bázisú újraformázását. A függvény visszatérő értéke TRUE ha hibátlan, és FLASE ha valami hiba történt A visszatérési érték csak (V39!) alalt használható A hibákról a graphics.library MakeVPort() függvényénél.

ScreenDepth - A Screen mélységének szervezése extra vezérléssel. (V39!)

uoid ScreenDepthfstruct Screen *Scr, ULONG Flags, RPTR reserued);
AO DO Al

Offset:- 786 ($312)

Előtérbe, vagy a hátérbe helyezése a megadott Screen-nek. A megengedett vezérlése a mélység helyzetnek a Screen családok esetén. A Scr-nél a screenre mutató pointer adható meg, míg a Flagsnál az alábbi flag-ek: SDEPTH_TOFRONT vagy az SDEFrH_TOBACK, a screen előtérbe vagy háttérbe helyezésének Ha a screen egy csatlakoztatott screen. használhatjuk a SDEPTHJNFAMILY flag-et is. hogy a screen-t bevezessük egy screen családba. A reserved megadása egyenlőre a NULL értékben merül ki.

ScreenPosition - a screen mozgatása nagyobb vezérlési lehetőséggel. (V39!)

```
uoid ScreenPositionístruct Screen *Scr, ULONG Flag, ULONG xi,
                          AO                DO          Dl
ULONG yi, ULONG x2, ULONG y2);
D2          D3          D4
```

Offset:- 792 ($318)

A screen mozgatása a megadott pozícióra, vagy megadott méretűre növelése. Az értékek megadása pixelben történik. Az V36 felhasználóknak újdonsága, hogy az értékek lehetnek negatívak a screen LeftEdge, TopEdge értékeihez képest. A flag paraméternél a következők adhatók meg: SPOS_RELATIVE. SPOS_ADSOLUTE a megadott értékekre vonatkozóan, vagy az SPOS_MAKEVISIBLE. Lehetőség van továbbá az SPOS_FORCEDRAG megadására is. ha szükséges.

ScreenToBack - a megadott screen hátrahelyezése a képernyőn.

```
uoid ScreenToBackístruct Screen *Scr);
                    AO
```

Offset:- 246 ($F6)

A screent leghátulra helyezi a képernyőn.

ScreenToFront - a screen előre helyezése.

```
uoid ScreenToFrontfstruct Screen *Scr);
                    AO
```

Offset:- 252 ($FC)

A megadott screent a többi elé helyezi, azaz láthatóvá teszi.

ScrollWindowRaster - Intuition barát ScrollRasterBFO (V39!)

```
uoid ScrollUJindoujRaster(struct Uiindouj *UJin, IDORG dx, WORD dy,
                          AO                     DO       Dl
UJORD xmin, UJORD ymin, UJORD xmax, UJORD ümax);
D2          D3          D4          D5
```

Offset:- 798 ($31E)

A függvény meghívja a graphics.library/ScrollRasterBFO függvényét a végrehajtáshoz. A scroll a dx.dy ban megadott értékkel történik. Az xmin. ymin. és az xmax. ymaxban megadhatjuk a scroll érvényességi tartományát. Ha a dx. dy-ban megadott érték kimutatna ebből, akkor nem történik semmi. A scroll működése után automatikusan kijavítja a Gadget-eket, az ablak keretét stb.. és küld egy IDCMP_REFRESHWINDOW üzenetet az user-nek.

SetAttrsA - beállítja a megadott értékűre az objektumot. (V36!)

SetAttrs - mint fent, csak külön Tag megadással.

```
ULONG SetfIttrsflíflPTR Object, struct TagItem* TagList);
DO                AO              Al

ULONG SetfIttrs(RPTR Object, ULONG Tagi, ...);
```

Offset:- 648 ($288)

A megadott értékűre állítja az Objektum változóját. Az objektum változóiról az Objektum leírásából nyerhetünk információt. A visszakapott érték nem nulla ha sikerült megváltoztatnia az értéket, ha az objektum gadget és így szükséges a gadget-et frissíteni.

SetDefaultPubScreen - új Default Pubscreen megadása. (V36!)

```
uoid SetDefaultPubScreentUBYTE *Name);
                          AO
```

Offset:- 540 ($2IC)

Elfogadtatja az új Defaùlt Pubscreen-t a „visitor" ablakokkal. Ha a megadott név paraméter-mutatója NULL, az értelemzett a Workbench screen lesz. Ha a visitor ablak FALLBACK opciója kiválasztott, a függvény kikényszeríti, hogy az ablak rákérdezzen a Default Pubscreen-re.

SetDMRequest - DMRequest beállítása az ablakhoz.

```
BOOL SetDMRequesUstruct UJindow *LUin, struct Request *Req);
DO                    AO                    Al
```

Offset:- 258 ($102)

A függvény megkísérel beállítani egy a Req-ben megadott DMRequestert a Winben megadott ablaknak. Részletesebben a könyv 1.9.3.2.2 részében. A függvény visszatérési értéke TRUE. ha az user aktivizálta az ablakon és FALSÉ, ha nem.

SetEditHook - Globális string gadget beállítás. (V36!)

```
struct Hook *SetEditHook(struct Hook *NewHook);
DO                       AO
```

Offset:- 492 ($1EC)

Beállítja az összes String Gadget-re vonatkozóan a szerkesztési hook-ot. A függvény nem tesztelt, így ne nasznaljuK széleskörűen teljesített piogiom esetén. A paraméterben az új szerkesztési információt tartalmazó hook struktúrájának mutatóját várja. Visszatéréskor a régi hook címét adja.

SetGadgetAttrsA - megadott érték módosítása a Boopsi Gadget-en. (V36!)

SetGadgetAttr - mint fent, csak a paraméterek egyenkénti átadásával.

```
ULONG SetGadgetflttrsflístruct Gadget *Gad, struct Window *UJin,
DO                          AO                    Al
struct Requester *Req, struct TagItem *TagList);
A2                          A3

ULONG SetGadgetRttrsfstruct Gadget *Gad, struct UJindow *IKin,
struct Requester *Req, ULONG Tagi, ...);
```

Offset:- 660 ($294)

A megadott érték módosítása a megadott értékűre, a szintén megadott ablakon elhelyezkedő Boopsi Gadget-en. Az érték amit módosítani lehet és amivel lehet, kiderül a Gadget leírásából. A visszatérési érték az állítani kívánt Gadget-tó'l függ, de általában ha nem nulla a Gadget-en sikerült a változtás eszközölése. és a Gadget frissítése szükségessé vált.[1]

SetMenuStrip - menü(k) csatlakoztatása az ablakhoz.

```
BOOL SetMenuStripístruct UJindow *UJin, struct Menü *Menu);
DO                   AO                    Al
```

Offset:- 264 ($108)

A menü(k) csatolása az ablakhoz. A menü(k) így láthatóvá válnak, és használhatja az user a jobb egérgomb alkalmazásával. A paraméterként megadott menünek tartalmazni kell legalább egy Item-et! A függvény TRUE értékkel fog visszatérni ha sikerült, különben mindig TRUE értékkel tér vissza, mert addig vár, ameddig nem sikerül.

SetMouseQueue - megváltoztatja az egértől függő üzeneteket. (V36!)

```
LONG SetMouseQueueístruct Windoui *UJin, UILIORD QueueLength);
DO                        AO                    DO
```

Offset:-498 ($1F2)

Az egér üzenetének számmegváltoztatása, amit az Intuition a pointer ablakon kívül tartózkodása esetén küld. Paraméterként az ablak pointerén kívül az új érték magadását várja. Visszatérési értékként a régi értéket adja, vagy -1-et ha az ablak nem ismert.

SetPointer - egér Pointerlmage-ének megadása az ablakhoz.

```
uoid SetPointeKstruct IDíndoiu *lilín, UUJORD *Pinterlmage,
                    AO                    Al
UJORD Height, WORD Width, UIORD KOffset, LUORtl VOffset);
DO            D1          D2             D3
```

Offset:- 270 ($1EO)

Az ablakhoz rerjdeli a megadott Pointerlmage-t. A Pointerlmage-nak normál UWORD tömb mutatónak kell lennie. Részletesen lásd 1.5.3 rész a könyben.
Az ablak törlése előtt a pointerünket is törölni kell. Erre használható a ClearPointerQ függvény.

SetPrefs - az Intuition Preferences adatainak beállítása.

```
struct Preferences *SetPrefs(struct Preferences *PreBuffer,
DO                             AO
LONG Size, BOOL Inform);
DO         DI
```

Offset:- 324 ($144)

A függvény az Intuition Preferences értékeit módosítja a PreBufferben megadott érékekkel. A másolás csak a Size paraméterben megadott mértékben zajlik le. A Inform paraméterben magadott TRUE esetén IDCMP_NEWPREFS üzenetet küld minden olyan ablaknak, ahol a? IDCMP_NEWPREFS flag beállított az ablak IDCMPFlag-jei között. A Preferences struktúráról részletesen a GetPrefsfJ függvénynél már szóltunk. Ennél részletesebben az <Intuition/preferences.h>-ban olvashatunk. Visszatérési értéknek a megadott PreBuffer tartalmát adja.

SetPubScreenModes - globális beállítása a Pubsree n(ek) viselkedésének. (V36!)

```
UUJORD SetPubScreenModes(UWORD Modes);
DO                       Al
```

Offset:- 546 ($222)

Globális beállítása a nyilvános screen-ek viselkedésmódjának. A megadható paraméterbe bállítható a screen SHANGHAl viselkedése (Workbench ablakok nyithatóak a deíault Pubscreenre). vagy a POPPUBSCREEN lamikor visitor ablak nyílik a PubScreenre. előtérbe kerül a screen). Visszatérési értékként a régi beállítást adja.

SetWindowPointerA - egérpointer beállítása az ablakunkhoz. (V39!)

SetWindowPolnter - mint fent. csak a tag-ok egyenkénti megadásával. (V39!)

```
uoid SetüJindOLuPointerflístruct UJindouj *UJin, struct Tagitem *TagList);
                                  AO                    Al

uoid SetU)JndouJPointer(struct UJíndouj *LUin, ULONG Tagi, ...);
```

OffseL- 816 ($330)

A függvény a megadott kinézetűre cseréli az egér pointerét, amikor az ablakot aktivizálták. A paraméterként megadott Win az ablakra mutató pointer, míg a Tag-ok a következők lehetnek:

WA_Pointer (AFTR) - a pointer kinézetét definiáló struktúra címe. Ha értéke NULL, az ablak a default pointert fogja használni, ami a Prefs-ben megadott.
WA_BusyPointer (BOOL) - értékével a busy (elfoglat) pointer használatára lehet rábírni az Intuitiont. Alapértelmezése a FALSE.
WA_PoinlerDelay (BOOL) - haTRUE értékűre állítjuk, akkor késleteteti beállítását kapjuk a rövid megjelenítési időre.

A példában beállítunk egy BusyPointert egy kis késleltetéssel. Ami a pointer törlésekor fog működésbe lépni:

```
/* Egy Busg Pointer egy kis késletetéssel. •/
SetWindowPointer(LUín, UJH_BusyPointer, TRUE,
                 WH_PointerDelay, TRUE,
                 THG_DONE);

/• fl busy anyagát ide rakd. */

/* Most a töröljük a pointert •/

SetWindowPointer(LUin, TRG_DONE);
```

SetWindowTitles - Beállítja az ablak fejlécét is és a Screen-et is.

```
uoid SetUJindouiTitlesístruct Wíndow *Win, UBVTE *ùJindouiTítle,
                                AO                  Al
UBVTE *ScreenTitle);
A2
```

Offset:- 276 ($114)

A Screen és az ablak fejlécének beállítása a paraméterben megadottra. Ha a paraméterben -1-et. vagy O-át adunk meg. az előző fejléc kerül kijelzésre (C-ben a -1 megadásáar használhatjuk a pld. (UBYTE *) ~0-t).

ShowTitle - beállítja a screen fejlécének kijelzési módját.

```
uoid ShowTitle(struct Screen *Scr, BOOL Shoiuli);
                    AO                 DO
```

Ofiset:- 282 ($1 1A)

Ez a rutin beállítja a SHOWTITLE flag-ot a megadott screen-en. és akkor létrejön a screen és az ablak újrakijelzése. A screen TitleBar képes a WFLG_BACKDROP ablakokat az előtérbe ill. háltérbe helyezni. A Showlt paraméternél megadott TRUE a backdrop ablakok előtérbe helyezését végzi, míg a FALSÉ visszahelyezi őket a háttérbe. Tehát ha FALSÉ, a TitleBar az egész mögött jelenik meg (title bar - a screen fejlécének kerete, csíkja stb.).

SizeWindow - az Intuition utasítása az ablak átméretezésére.

```
uoid SizeLUindow(struct Windouj *Win, (KORD DeltaK, UORD DeltaV);
                    AO                 DO          D1
```

Offset:- 288 ($120)

Ez a rutin átméretezi az ablakot a megadott DeltaX. DeltaY méretekkel nagyobbra ill. kisebbre. Ha az ablak IDCMP_NEWSIZE flag-ja beállított küld egy ilyen üzentet is. V36-os rendszeren az Intuition ellenőrzi a végleges méretet, hogy megfelelő-e, az ablaknál megadott értekek közötti lesz-e az ablak (MinWidth. MinHeight. MaxWidth. MaxHeighl). Valamint átugorja a WFLG_SIZEGADGET nem beállításából adódó problémít.

SysReqHandler - a rendszer Requester „bemenetének" kezelése. (V36)

```
LONG SysReqHandler(struct UJindow *UJín,
DO                          AO
ULONG *IDCMPFlagsPtr, BOOL UJaitInput);
Al                          DO
```

Ofíset:- 6OO (($258)

Az ablak felé küldött IDCMP üzenetek kezelése, amit a BulidSysKequester() vagy a BuildEasyRequesterO függvény visszadna Ezzel a függvénnyel képessé válunk aszinkronizálni az EasyRequesK) vagy az AutoRequestO függvényeket.

Megadhatjuk, hogy a programunk ne fusson tovább addig, ameddig a Requester várakozik. Minden esetben amikor ezt a függvényt meghívjuk, a függvény átvesz IDCMP üzeneteket amik az ablakra érkeznek. Ha a Waitlnpul paraméter nem nulla, a függvény várakozni fog a beállított IDCMP üzenet megérkezéséig. A függvény visszatérési értéke hasonló az EasyRequeslfJ függvény értékéhez. Ha a gadget ID nagyobb akkor 0. ha nem akkor -1. és 1 ha más IDCMP üzenet érkezett. A függvénynek van még egy *-I* visszatérési értéke is, ha kapott üzenet nem a Requester-re vonatkozik. Egy példán mutatjuk be a használatát, amelyben az EasyRequestO függvénnyel együtt használva várakozó ciklust hozunk léire.

```
ujjndouj = BuildEasyRequestf...);
Luhile((retual = SysRequestHandler(windouj, ídcmp_ptr,
TRUE)) == -2)
{
    /* ciklus •/
}
FreeSysRequest(window);
```

Vagy egy konkrétabb alkalmazási példa, egy lemez bekérése, de a Requester nem lőhető le ha rossz lemezt helyeztek be.

```
struct EasyStruct uolumeES =
{
    sizeof(struct EasyStruct),
    0,
    "Uolume Requester",
    "Please insert uolume %s in any driue.",
    "Cancel"
};

Uolume *GetUolume(uolume)
UBVTE *uolume;
{
    struct UJindouj *UJin;
    Uolume *uolume =NULL;
    Uolume *findUolume();

    int    retual;

    UJin = BulidEasyRequest(NULL, t/uolumeES,

    IDCMP_DISKINSERTED, uolname);

    µjhile((retual = SysReqHandler(Wín, NULL, TRUE)) !=0)

        /• már nem lelőhető */

        /* amikor IDCMP_DISKINSERTED megnézzük melyik
        egység */
        iflfretual == -1) &C- (uolume = findUolume(uolume)))
            break;
    }
    FreeRequest(UJin);
    return(uolume);
}
```

TimedDisplayAlert - egy Alert kijelzése a kívánt ideig. (V39!)

```
BOOL TimedDisplayfilertlULONG RIertNumber, UBVTE *String,
DO                         DO                      AO
UIDORD Height, ULONG Time);
Dl                 Al
```

Offset:- 822 ($336)

Ez a függvény mindenben hasonlít a normál DisplayAlertO függvényre, azzal a különbséggel, hogy megadható az az idő. ameddig az Alert kijelzésre kerül.

Ha az user nem válaszol, akkor eddig az ideig látható az Alert. Az idő megadását videó frame-kben számolja. A függvény más paraméterei teljesen megegyeznek a DisplayAlertO függvényével. A visszatérési értéke szintúgy. Ha az user a beállított időn belül nem válaszol az egér jobb, vagy bal szemének lenyomásával, akkor az idő letelte után FALSE értékkel tér vissza (azaz mintha a jobb gombot nyomták volna meg).

UnlockIBase - felengedése az Intuition lezárásának, amit a LocklBase() ért el.

```
uoid UnlocklBase(ULONG);
                 AO
```

Offeet:- 420 ($1A4)

Felengedi az előzőleg LockIBaseO függvénnyel lezárt [ntuition-t. Ha ezt a függvényt nem a LockIBaseO használatával eredményezett lezárásra hívjuk meg, az a rendszer összeomlásához vezet. A paraméterként várt értéket a LockIBaseO függvény adja vissza. A paramétert nem a DO-ba várja!

UnlockPubScreen - a lezárt nyilvános srreen elengedése. (V36!)

```
uoid UnlocPubScreen(BYTE *Name, struct Screen *Screen);
                    AO          Al
```

Oflsel:- 516 ($204)

A függvény felszabadítja a zár alól a nyilvános screen-t. amelyet vagy a név megadásával, vagy ha ez NULL. a screen megadásával határozhatunk meg. Ebben az esetben a Screen pointer-ének annak a pointernek kell lennie, amit a LockPubScreenO függvény visszaad. Ha mindkét paraméter NULL. nem történik semmi.

UnlockPubScreenList - a PubScreen lista szemaforjának elengedése. (V36!)

```
uoid UnlockPubScreenList(uoid);
```

Offset:- 528 ($210)

Felengedi a LockPubScreenListO függvénnyel lezártakat.

ViewAddress - visszaadja az Intuition View struktúrájának pointer-ét.

struct Uieuj •UieuiHddress(uoid);
DO

Offset:- 294 ($126)

Ha használni akarunk grafikákat, szöveg kiírásokat, animáló „primitíveket", objektumokat, szükségünk lehet egy mutatóra a View struktúrájára az íntuitionnak. Ezzel a függvénnyel ehhez juthatunk hozzá.

ViewPortAddress - visszaadja az ablak ViewPort címét.

struct UiewPort *UiewPortfIddress(struct Window *U)in);
DO AO

Offset:- 3OO($12C)

Visszaadja a megadott ablak ViewPort címét. így képessé válunk rajzolni és egyebeket elkövetni a graphics.library segítségével, még ha a screen-t nem is ismerjük (pl. ha PubScreen-re nyitottuk az ablakot).

WBenchToBack - az összes screen mögé küldi a Workbench screen-t.

BOOL UJBenchToBack(uoid);

Orfset:- 336 ($150)

Az összes screen mógé helyezi a Workbench screen-t. de semmi más effektust nem csinál. Ha a Workbench screen nyitott volt, a függvény a TRUE értékkel tér vissza, egyébként FALSÉ.

WBenchToFront - a Workbench Screen előre helyezése.

BOOL WBenchToFront(uoíd);

Orrset:-342($156)

A Workbench Screen-t az összes screen elé helyezi, azaz ez lesz látható a képernyőn. Ha a Workbench Screen meg volt nyitva, a visszatérési érték TRUE. ha nem. akkor FLASE.

WindowLimits - beállítja az ablak kinyithatóságának minimális és maximális értékeit.

```
BOOL UJindowLimitsIstruct UJindouj *UJin, UWORD Minlilidth,
DO                          AO                  DO
UWORD MinHediht, UWORD MaxUJidht, UWORD ManHeiight);
Dl               D2               D3
```

Offset:- 318 ($13E)

Az ablak maximális kinyithatóságát és minimális összecsukhatóságát állíthatjuk be a segítségével. Ha nem akarjuk az összes értéket megváltoztatni. akkor a nem kívánt paraméterek helyére írjunk nullát. Ha a legnagyobb méretűre kívánjuk állítani amire csak lehetséges (a Screen felbontása befolyásolja), akkor a kívánt nagyság állító paraméterre adjuk meg a -1 (pld. --0). A visszatéréskor TRUE értéket ad ha az átállítás sikeres , és FALSE értéket, ha valamelyik megadott paraméter nem megfelelő volt.

WindowToBack - az ablak hátrahelyezése a többi ablak mögé.

```
uoid LUindoiuToBacktstruct Window *UJin);
                           AO
```

Offset:- 306 ($132)

A függvény utasítja az Intuition-t. hogy helyezze a megadott ablakot a többi, arra a screen-re nyitott ablak mögé. A WFLG_BACKDROP ablakot nem lehet sem az előtérbe, sem a háttérbe helyezni!

WindowToFront - az ablak előtérbe helyezése.

```
uoid UJíndoa»ToFront(struct UJindoui *UJin);
                             AO
```

Offset:-312 ($138)

A függvény utasítja az Intuiüon-t, hogy a megadott ablakot helyezze a többi ablak elé, amelyeket arra a screen-re nyitottak. A WFLG_BACKRDOP ablakot nem lehet sem az előtérbe, sem a háttérbe helyezni, helye mindig hátul van.

ZipWindow - Megváltoztája az ablak pozícióját és méretét. (V39!)

```
uoid ZipWindowístruct IDindoLU *UJin);
                          AO
```

Offset:- 504 ($1F8)

A függvény megvállozlaja az ablak helyét és méretét, amint azt a? ablak Zoom gadget-jével mi is megtehetnénk. A szükséges méreteket a WA_Zoom tag-item megadása után adhatjuk meg a Tag listában. Ha az ablakon beálh tott az IDCMP_CHANGEWINDOW flag. akkor ilyen üzenettel tudomást szerzünk a függvény működéséről.

2.2.1 Az Intuition Library függvényei qffszet sorrendben

Ebben a részben megadjuk az Intuition függvényeit offszet sorrend szerint, ezzel elősegítve a Disassambly utáni eligazodást a kódban.

-• 30 OpenIntuition
-- 36 Intuition
-• 42 AddGadget
-• 48 ClearDMRest
-- 54 ClearMenuStnp
-- 60 ClearPointer
-- 66 CloseScreen
-- 72 CloseWindow
-- 78 CloseWorkBench
-- 84 CurrentTime
-- 90 DisplayAlert
-- 96 Di&playDeep
-- 102 DoubleClick
-- 108 DrawBorder
-- 144 DrawImage
-- 120 EndRequest
-- 126GetDefPreis
-- 132GelPrefs
-- 138 InitRester
-- 144 JtemAddress
-• 150ModifyIDCMP
-• 156 ModifyProp
-- 162 MoveScreen
-- 168 MoveWindow
-- 174OffGadget
-- 188 OflMenu
-• 186 OnGadget
-- 192 OnMenu
-- 198 OpenScreen
-• 204 OpenWindow
-- 210OpenWorkBench
-•216 PnntlText
- 228 RefreshGadgets
- 228 RemoveGadget
-• 234 ReportMouse
- 240 Request
- 246 ScreenToBack
-• 252 ScreenToFi-ont
- 258 SetDMResquest
- 264 SetMenuStrip
-• 270 SetPointer
-• 276 SetWindowTHles
-- 282 ShowTitle
-- 288 SizeWindow
-- 294 ViewAddress
-- 300 ViewPortAddress
-- 306 WindovvToBack
-- 312 WindowToFront
--318 WindowLimits
-- 324 SelPrefs
-- 330 IntuiTextl>ength
-- 336 WBenchToBaok
-- 342 WBenchToFront
-- 348 AutoRequest
-- 354 BeginRefresh
-- 360 BuildSysRequest
-- 366 EndRefresh
-- 372 FreeSysRequest
-- 378 MakeScreen
-- 384 RemakeDisplay
-- 390 RethinkDisplay
-- 396 AllocRemember
-- 402 AlohaWorkbench
-- 408 FreeRemember
--414LockIBase
-- 420 UnlocklBase
-- 426 GetScreenData
-- 432 RefreshGList
-- 438 AddGList
-- 444 RemoveGList
-- 450 ActivateWindow
-- 456 RefreshWindowFranie
-- 462 ActivateGadget
-- 468 NewModifyProp
-- 474 QueiyOverscan
-- 480 MoveWindow InFrontOr
-- 486 ChangeWindowBox
-- 492 SetEditHook
-- 498 SetMouseQueue
-- 504 ZipWindow
-- 510 I-ockPubScreen
-- 516 UnlockPubScreen
-- 522 U>ckPubScreenList
- 528 UnloekPubSereenList
- 534 NextPubScreen
- 540 SetDefaultPubScreen
- 546 SetPiibSci-eenModes
- 552 PubScreenStatus
- 558 OblainGlRPort
- 564 ReleaseGIRPort
- 570 GadgetMouse
- 582 GetDelaultPubSereen
- 588 EasyRequestArgs
- 594 BuildEasy RequestArgs
- 600 SysReqHandler
- 606 OpenWindowTagüst
- 612 OpenScreenTagList
-618 DrawImageState
- 624 PointlnImage
- 630 EraseImage
- 636 NewObjectA
- 642 DisposeObject
- 648 SetAttrsA
- 654 GetAttr
- 660 SetGadgetAttrsA
- 666 NextObject
- 678 MakeClass
- 684 AddClass
- 690 GetScreenDrawInfo
- 696 FreeScreenDrawlnfo
- 702 ResetMenuStrip
- 708 RemoveClass
-714 FreeClass
- 768 AllocScreenBuffer
- 774 FreeScreenBuffer
- 780 ChangeScreenBuffer
- 786 ScreenDepth
- 792 ScreenPosition
- 798 ScrallWindowRaster
- 804 LendMenus
- 810 DoGadgelMethodA
- 816 SelWindowPointerA
- 822 TimedDisplayAlert
- 828 HelpControI

2.3 A Graphics Library

Az Amiga grafikájáért főleg a Graphics Library a felelős Innen tudjuk rendszer alól programozni a Blitter-t. csinálhatunk saját Copper listát, átrajzolhatjuk a sprite-okat és persze az alap grafikai rutinok is itt vannak. Itt is csak a könyvben található rutinokat írjuk le és még néhányat, ami egyértelmű.

BltBitMap - egy BitMap-en négyzet mozgatása

```
ULONG BltBitMapístruct BitMap* SrcBitMap , WORD SrcH, UJORD SrcV,
DO                     AO                    DO.16      D1:16
struct BitMap*, UJORD DstX, UJORD DstV , UJORD SizeK ,
Al              D2:16       D3:16        D4:16
UJORD SizeV , UBVTE Minterm, UBVTE Mask, [UUJCIRD * TempR]);
D5:16         D6:8           D7:8         [A2]
```

Offset:- 3O($1E)

A kijelölt négyzet alakú területei egy másik helyre másolja, mely, lehet másik bitmap-en is. Ha valami kilógna, akkor azt a rendszer nem figyeli, tehát a paraméterezéssel figyeljünk oda.

ScrBitMap.DestBitMpa - mutatók BitMap struktúrákra

SrcX.SrcY - a forrás X és Y koordinátája
DstX.DstY - a cél X és Y koordinátája
SizeX.SizeY - az átmásolandó terület szélessége és magassága
Minterm - a Blitter-nek a másolási függvénye, a könyv következő részében a Blitter leírásánál ezekre sokkal részletesebben kitérünk, most álljon itt néhány példa:
OxCO normál másolás
0x30 invertálja a forrás területet a másolás előtt
0x50 csak a cél területet invertálja
Mask - ez is a Blitternek kell a célterületre való iráíshoz. Általában OxFF az értéke, ki kell próbálni.
TempA - ha a másolandó területek között átfedés van. akkor ezt a Chip Ram-ba mutató területet felhasználja. Ha nulla Eikkor foglal saját magának területet, majd azt fel is szabadítja (érdemes nullának hagyni).

Például:

```
BltBitMap(C'Scr->BitMap,0,20,&scr->BitMap,0,12,640,1 QQ,0nCC,255,0);
```

ClearEOL - törlés a sor végéig

```
UOIQ ClearEOUstruct HastPort *rp);
                   Al
```

Offset:- 42 ($2A)

Lelöröl egy négyzetes területet a megadott RastPort-on. az aktuális grafikus kurzor pozíciójától. A négyzet magassága megfelel az aktuális karakterkészlet magasságának.

ClearScreen - képernyő törlése

uoid ClearScreenístruct RastPort *rp);
Al

Offset:- 48 ($30)

Az aktuális grafikus kurzor pozíciójától kezdve a sor végéig a ClearEOL rutinnal letörli a képernyőt, a többit (ami lentebb van) pedig teljesen letörli a képernyő aljáig.

TextLength - megadja egy szöveg hosszát pixelekben mérve

WORD TextLengthtstruct RastPort *rp, STRPTR string, WORD count);
DO Al AO D0:16

Offset:- 54 ($36)

A jelenlegi beállítások mellett megadja a megjelölt szöveg hosszát pixelekben mérve. Ezzel leellenőrizhetjük, hogy a grafikus kurzortól ha kiírnánk, akkor esetleg kilógna-e a képernyőről. Főleg arra használható, ha középre igazított szöveget szeretnénk kiírni, de a karakterkészlet betűi nem azonos méretűek.

Text - megadott szöveg kiírása

uoid Tewtístruct RastPort *rp, STRPTR string, WORD length);
Al AO D0:16

Offset:- 60 ($3C)

A grafikus kurzor pozíciójától kezdve kiírja az aktuális karakterkészlettel a megadott szöveget. A rendszer nem figyeli, hogy a szöveg kilóg-e a képernyőről. Ha ez épp a képernyő alján van. akkor bele is írhat egy másik programba, esetleg rendszerösszeomlást okozhat.

A szöveg kiírása után a grafikus kurzort a szöveg következő karakteréhez igazítja, folyamatosan írhatunk tovább.

SetFont - aktuális karakterkészlet beállítása

uoid SetFontfstruct RastPort *rp, struct TentFont *font);
Al AO

Offset:- 66 ($42)

A megadott RastPort-hoz átállítja a karakterkészletet és törli a régi attribútumokat. A Font mutató egy struktúrára, amit az OpenFontQ vagy OpenDiskFont() rutin adott vissza (lehetőleg nullát ne adjunk meg).

OpenFont - karakterkészlet megnyitása

```
struct TentFont* OpenFont(struct Tentflttr *te«tHttr);
DO                              AO
```

Offset:- 72 ($48)

Megkeresi a rendszerfont-ok között a megadottat és megnyitja azt. Amit visszaad, az kell a SetFont() rutinnak. Ajánlott a jó memóriagazdálkodás érdekében CloseFonl()-al a programunk végén lezárni. Ha nullát ad vissza, akkor nem talált ilyen Fontkészletet.

Példa a rendszer Topaz fontjának TextAttr struktúrájára:

```
struct TeKtHttr topaz8 = { (STRPTR)"topaz.font",8,0K00,0K01);
```

CloseFont - kerekterkészlet lezárása

```
uold CloseFont(struct TeHtFont *);
                  Al
```

Offset:- 78 ($4E)

Az OpenFonlO vagy OpenDiskFontO rutinnal megnyitott karakterkészletet zárhatjuk le vele.

AskSoftStyle - a Soft Style biteket kérdezhetjük le egy Font-ról

```
ULONG HskSoftStyleístruct RastPort *rp);
                        Al
```

Orfset:- 84 ($54)

A RastPort hoz tartozó fontkészlet sülusbeállítását adja vissza. Azok a bitek, melyek nincsenek definiálva, azok l-re lesznek'állítva.

SetSoftStyle - a Soft Style bitek (stílus) beállítása

```
ULONG SetSoftStyle(struct RastPort *, ULONG style, ULONG enable);
DO                  Al              DO          D1
```

OITset:- 90 ($5A)

A megadott RastPort-hoz tartozó fontkészlet stílusát állíthatjuk be.
A Style határozza meg a stílusú <\s. cnablc pedig *CWA* jelenti. Hogy nwiy biteket akarjuk átállítani (a többi változatlan marad). DO-ba az új Style-t kapjuk vissza.

DrawEllipse - ellipszis rajzolása

```
uoid DraiuEllJpse(struct RastPort *rp, SHORT ex, SHORT cy, SHORT a,
                  A1                   DO        D1        D2
SHORT b);
D3
```

Oflset:- 180 ($B4)

A megadott RastPort-hoz taroz'ó BitMap-ra (cx.cy) középpontú a és b sugarú ellipszist rajzol. A kép szélét nem vizsgálja.

AreaEllipse - fillezett ellipszis rajzolása

```
LONG RreaEllipsetstruct RastPort *rp, SHORT CK, SHORT cy, SHORT a,
DO                      A1                    DO        D1     *D2
SHORT b);
D3.
```

OfTset:- 186 ($BA)

A megadott RastPort-hoz tarozó Raster-re (cx.cy) középpontú a és b sugarú ellipszist rajzol. DO-ba ha -1-et ad vissza, akkor nem fért bele a vektorlistába. Használata az alacsony szintű grafika Fiiled Area részében részletesen le van írva'.

LoadRGB4 - paletta beállítása színtáblázatból

```
uoid LpadRGB4(struct UiewPort *up, UUJORD *colors, UJORO count);
              AO                   A1              D0:16
```

OÍTset:- 192 ($C0)

A megadott ViewPort-hoz tartozó palettákat beállíthatjuk egy színtáblázat segítségével, a nullás (háttér) színtó'l kezdve, mely az alábbi módon néz ki:

```
colors:   dc.ui $0000   ;fekete
          dc.iu $OfOQ   ;piros
          dc.LU $OOfO   ;zöld
          dc.LU $000f   ;kék
```

A maximális intenzitás 15 ($F). A count határozza meg a beállítandó színek számát, a colors pedig a táblázat címét tartalmazza.

WaitBlit - várakozás a ÜHUer-re

uoíd UJaitBlit(uoid);

Offset:- 228 ($E4)

Addig vár. míg a Blitter be nem fejezi a kijelölt müveletet.

Set Rast - teljes rajzolási terület kiszínezése megadott színre.

uoid SetRast(struct RastPort *rp, UBVTE pen);
Al DO

Orfset:- 234 ($EA)

A megadott RastPort-hoz tartozó BitMap-eket feltölti a megadott (0-255) színnel.

Move - grafikus kurzor mozgatása.

uoid Mouetstruct RastPort *rp, SHORT K, SHORT y);
Al DO D1

Oflset.- 240 ($F0)

A megadott RastPort-hoz tartozó grafikus kurzort mozgathatjuk. Ez a grafikus kurzor nem látszik! A Draw és Text rutin ezt használja kezdőpontnak.

Draw - vonal rajzolás.

uoid Drawístruct RastPort *rp, SHORT x, SHORT y);

Al DO D1

Offset:- 2461$F6)

Vonalat rajzol a grafikus kurzor pozíciójától kezdve a megadott (x.y) pontig.

AreaMove - az Area részen a grafikus kurzor mozgatása és polygon lezárása

LONG RreaMoue(struct RastPort *rp, SHORT K, SHORT y);
DO Al . D0:16 D1:16
Offset:- 252 ($FC)

Ezzel tudjuk a grafikus kurzort a lefoglalt Raster-en mozgatni ül. ha már elkezdtünk egy polygon-t rajzolni, akkor ez lezárja azt. c& újat kezdhetünk. Ha -1a visszatérési érték, akkor nem fért bele a vektortáblázatba.

AreaDraw - az Area részen vonal rajzolás

```
LONG RreaDrauJístruct RastPort *rp, SHORT K, SHORT y);
DO                A1                 DO       D1
```

Offset.- 258 ($102)

Az előző ponttól egy újabb vonalat tesz a vektortáblázatba. Ha -1 a visszatérési érték, akkor nem fért bele a táblázatba.

AreaEnd - a vektorláblázat lezárása és a fillezés elkezdése.

```
LONG RreaEndístruct RastPort *rp);
DO              A1
```

Offset:- 264 ($108)

Az elkészített vektortáblázatból a polygonokat és ellipsziseket elkezdi fillezni és a kész rajzot rámásolja a látható képre.

WaitTOF - várakozás a következő videó frame-re.

```
uoid WaitTOF(uoid);
```

OlTset:- 270 ($10E)

Várakozik az elektronsugár visszafutására és az összes erre láncolt megszakítás befejezésére. Ez időzítésre használható akkor, ha azt akarjuk, hogy a programunban a mozgások ne villogjanak.

InitArea - vektortáblázat inícializálása.

```
uoíd InitRreatstruct Rrealnfo *areainfo, uoid *buffer,
              AO                          Al
SHORT maxuectors);
DO
```

Offset:- 282 ($11 A)

A lefoglalt buffer-t és az Arealnfo struktúrát ezzel tudjuk inicializálni. Meg kell adni a maximális vektorok számát, ami még belefér a bufTer-be. Egy vektor (AreaMove/AreaDraw) 5 byte-ot foglal, egy AreaEllipse pedig 10 byte-ot. (lásd az Alacsony szintű grafika példaprogramjait)

SetRGB4 - palettaregiszter beállítása

```
uoíd SetRGB4(struct UieutPort *, SHORT n, UBVTE r, OBVTE g, UBVTE b);
               AO                 DO      D1:4    D2:4    D3:4
```

Ofrset:- 288 ($E4)

A megadott ViewPort-hoz tartozó n. palettaregisztert R. G, B színűre állítja. Az R (Red) a piros. G (Green) a zöld. a B (Blue) pedig a kék összetevő. Az n 0-31 tartományba eshet, az R. G. B pedig 0-15.

RectFill - fillezett négyzet rajzolása

```
uoid RectFilKstruct RastPort *rp, SHORT xmín, SHORT ymin,
              Al                  DO          D1
SHORT ymin, SHORT ymin);
D2          D3
```

OlTset:- 306 ($132)

A megadott RastPort-ra egy fillezett négyzetet rajzol. A xmin és ymin a bal felső sarok, az xniax és ymax a jobb alsó sarok koordinátái. A (xmax>=xmin) és (ymax>=ymin) relációnak teljesülni kell.

ReadPixel - pixel színének kiolvasása.

```
LONG ReadPixeKstruct RastPort *rp, SHORT x, SHORT y);
```

DO Al D0:16 1)1:16

Offset:- 318 ($13E)

A megadott (x.y) pont színét (pen) adja vissza.

WritePixel - pixel kirakása.

```
LONG WritePixeHstruct RastPort *rp, SHORT x, SHORT y);
```

DO Al DO D1
OlTset:- 324 ($144)

A megadott (x.y) pontra egy pixel-t rak ki az A pen színnel. Ha a visszatérési érték 0, akkor sikerült, ha -1 akkor a RastPort területén kívül van a meghatározott pont.

Flood - nilezés.

```
BOOL FloodUtruct RastPort *rp, ÜLONG mode, SHORT x, SHORT y);
```

DO A1 D2 DO D1

OlTset:- 330 ($14A)

A megadott ponttol kezdve elkezdi kifillezni az egész területet. Ha a mode 0. akkor OutLine módú, ha 1. akkor Colour módú lesz a fillezés.

PolyDraw - polygon rajzolása

```
uoid PolyDrawfstruct RastPort *rp, WORD count, UWORD *array);
                Al                    DO          AO
```

Ofi'set:- 336 ($150)

A grafikus kurzor pozíciójától kezdve a megadott tömbből elkezd folytonos vonalat rajzolni. A count a táblázat elemeinek száma.
A tömb:

```
UWORD [lines]={0,0,
               100,50,
               30,50
              };

      .
      .
      .
      PoiyDrau)(rp,3,&lines);
```

SetAPen - az A pen beállítása

```
uoid SetRPen(struct RastPort *rp, UBVTE pen);
                Al                   DO
```

Offset:- 342 ($156)

Az elsődleges rajzolási színt állítja be. Ez a 0-255 tartományba eshet. A Draw, Text, WritePixel és még sok más rajzoló rutin ezt használja, a beállított színnel rajzolnak. Megegyezik a RastPort struktúrában az FgPen-el.

SetBPen - a B pen beállítása.

```
uoid SetBPen(struct RastPort *rp, UBVTE pen);
                Al                   DO
```

OITset:- 348 ($15C)

Az másodlagos rajzolási színt állítja be. Ez a 0-255 tartományba eshet. A Text. Draw (ha be van állítva egy paüern) és fill rutinok ezt használják. Megegyezik a RastPort struktúrában a BgPen-el.

SetDrmd - rajzolási mód beállítása

```
uoid SetürMdístruct RastPort *rp, UBVTE mode);
                Al                   D0:8
```

Offset:- 354 ($162)

Beállítja a rajzolási módot a Draw. Fill és Text rutinoknak. A mode 0-255 tartományban lehet, a mintermeket állítja át. Használható alapbeállítások a JAM1. JAM2. COMPLEMENT és 1NVERSVID.

VBeamFos - az eleklronsugár vízszintes pozíciója.

LONG UBeaffiPusíuoídJ;
DO

OffseL:- 384 ($180)

Az elektronsugár az aktuális vízszintes pozícióját adja vissza, melyet a Custom regiszterekből szed ki.

WaitBOVP - vár. míg az elektronsugár eléri a megadott VievPort-ot.

uoid LUaitBOUP(struct UiewPort *up);
AO

Offset:- 402 ($192)

Addig vár. míg az elektronsugár el nem éri a megadott ViewPort tetejét. Ez is időzítésre jó. mint a WaitTOF.

OwnBlitter - a Blitter kisajátítása.

uoid OujnBlitter(uoid);

Offset:- 456 ($1C8)

A Blitter-t kisajátíthatjuk privát felhasználás céljából. Ha a Blitter szabad, akkor azonnal visszatér, ha nem. akkor addigTask-unkat várakoztatja. Mielőtt a Blitter regisztereit átírnánk, csináljunk egy WaitBlit rutint is. hogy az működését befejezze.

DisownBlitter - a Blitter visszaengedése a rendszernek.

uoid DisoujnBlitter(uoid);

Offset:- 462 ($1CE)

A/. OwnBlitter rutinnal lefoglalt Blittert visszaengedjük a rends/ernek és minden más programnak.

InitTmpRas - terület inícializálása az Area rutinokhoz.

uoid InitTmpRasístruct TmpRas *tmpras, uoid *buffer, ULONG size);
AO Al DO

Offset:- 468 ($1D4)

Az AllocRaster rutinnal lefoglalt raster-t inicializálja a tmpras struktúrába az Area rutinokhoz. A lefoglalt raster-területen fog a Blitter dolgozni, onnan másolja ra a kész llllezeU. raj/;ol a látható RsstPort ra. A sine a buffer mérete byte-okban megadva A buffer-nek a Chip Ram-ban kell lenni (de ha az AllocRaster rutinnal dolgozunk, akkor az elintézi ezt nekünk).

AllocRaster - helyeL foglal BitPlane-nek.

```
PLfiNEPTR RIlocRasteKULONG Lüidth, ULONG height);
DO                         DO          Dl
```

Orfsel:- 492 ($1EC)

Helyet foglal a Chip menióriában (a/ AllocMemO rutint hívja meg). A width és a height pixelben határozza meg a lefoglalandó terület nagyságát. Ha a visszatérési érték 0. akkor nem sikerült lefoglalni a kért nagyságú helyet, feltehetőleg memóriahiány miatt. A lefoglalt területet a FreeRasterO rutinnal kell felszabadítani.

FreeRaster - lefoglalt raster felszabadítása.

```
uoid FreeRasteKPLHNEPTR p, USHORT width, USHORT height);
                    AO          D0:16         D1:16
```

Ofíset:- 498 ($1F2)

Az AllocRaster által Chip Ram-ban lefoglalt helyet tudjuk felszabadítani. A lefoglalt méreteknek meg kell egyezni!!

GetRGB4 - színek lekérdezése.

```
ULONG GetRGB4(struct ColorMap *, LONG entry);
DO            AO                 DO
```

Offset:- 582 ($246)

A megadott ColorMap struktúrából (amit áViewPort struktúrából szedhetünk ki) az entry-vel meghatározott színt visszaadja. Ha -1 a visszatérési érték, akkor az entry nem volt jó.

GetVPModelD - megadott ViewPort azonosítójának lekérdezése.

```
ULONG GetUPModelDístruct UiewPort *up);
                         AO
```

Offset:- 792 ($318)

A megadott ViewPort modelD-jét adja vissza. Hiba esetén nem nullát, hanem INVALIDJD-t kapunk. A ID-k a inrlude/graphics/modes.h file-ban vannak definiálva (és a könyvben is megvan ez a táblázat...). (V36).

ModeNotAvailable - mód létezésének lekérdezése.

ULONG ModeNotRuailableíULONG modeID);
DO DO

OiTsel:- 798 ($31E)

Megnézi, hogy a megadott módot tudja-e a gép. Ha nem, azért meri esetleg nincs megfelelő chip-készlet. nincs olyan monitor beállítva, vagy a/ a mód nem rendelkezik genlock-al. Ha a visszatérési érték 0. akkor meg tudja jeleníteni, ha nem, akkor generál egy hibakodot, mely utal a meg nem jeleníthetőség okára (szép kifejezés...). (V36)

SetRGB32 - bővített palettaregiszter beállítása.

uoid SetRGB32(struct UiewPort *up, ULONG n, ULONG r, ULONG g,
AO DO D1 D2
ULONG b);
D3

Orfset:- 852 ($354)

Hasonló a SelRGB4-hez. de itt egy színt egy 32 bites szám ír le. AGA gépek esetén ez a szám csak 8 bites, de így is 16777216 színt tud kezelni. Az n 0-255-ig terjedhet, a r.g.b pedig balra igazított szám. Érthetőbben: ha rnox zöldet szeretnénk beállítani: OxFFOOOOOO. Az alsó biteket is ki lehet tõlteni. de azok most még nem számítanak (lehet, hogy majd az AGA utáni chipkészlet kihasználja). (V39!)

GetAPen - a/ elsődleges pen (A) lekérdezése.

ULONG GetHPen(struct RastPort *rp);
DO AO

Orfset-- 858 ($35A)

Az aktuális A pen értékét adja vissza. Ez a RastPort struktúrából is kiolvasható (FgPen). de lehet, hogy később újabb KickStart-nál ez máshova kerül. (V39!)

GetBPen - az másodlagos pen (13) lekérdezése.

ULONG GetBPenístruct RastPort *rp);
DO AO

Offset:- 864 ($360)

Az aktuális lS pen eneKei adja vin»/-,<i *cr.* u rtnouvn --t.uku.,.,i,„i ,« Uí,*I vasható (BgPen). de lehet, hogy később újabb KirkStart-nál ez ma&b.>va kerül. (V39!)

GetDrMd - az aktuális rajzolási mód lekérdezése.

```
ULONG GetDrMdistruct RastPort *rp);
DO                  AO
```

Offsel:- 870 ($366)

Az akluális rajzolási módot adja vissza. Ez a RastPort struktúrából is kiolvasható (DrawMode), de lehet, hogy később újabb KickStart-nál ez máshova kerül. (V39!)

LoadRGB32 - bővített paletta beállítása színtáblázatból.

```
uoid LoadRGB32(struct UiewPort *up, ULONG *table);
                     AO                  Al
```

Orfset:- 882 ($372)

Hasonlóan a LoadRGB4-hez. ez is egy táblázatból tölti be a színertékeket. de itt a table felépítése más:

1 word - a színek száma
1 word - az első színregiszter szàma
3 longword - az r.g.b színeket tartalmazó longword-ök (mint a SetRGB32-nél)
.
.
.
1 long - egy lezáró 0

A table-nak nem adhatunk meg NULL-t. (V39!)

GetRGB32 - színek lekérdezése.

```
uoid GetRGB32(struct ColorMap *cm, ULONG firstcolor,
                    AO                  DO
ULONG ncolors, ULONG *table);
DI                 Al
```

Oríset:- 900 ($384)

32 bites RGB értékekkel tölti fel a megadott táblázatot. A ColorMap struktúrát a ViewPort-ból szedhetjük ki. A firstcolor az első színregiszter számat jelenti, az ncolors pedig a lekérdezendő színek számát.
A táblázatnak ncolors*3 longword-öt kell tartalmaznia.

2.3.1 A Graphics Library függvényei qffszet sorrendben

-- 30 BHBitMap
-- 36 BltTemplale
-- 42 ClearEOL
-- 48 ClearScreen
-- 54 TextLenglh
-- 60 Text
-- 66 SetFonl
-- 72 OpenFont
-- 78 CloseFont
-- 84 AskSoftStyie
-- 90 SetSoftStyle
-- 96 AddBob
-- 102 AddVSprRe
-- 108 DoCollision
--114 DrawGList
-- 120 InitGels
-- 126 InilMasks
-- 132 Remi Bob
-- 138 RemVSprite
-- 144 SelCollision
-- 150 SortGList
-- 156 AddAnimOb
-- 162 Animate
-- 168 GetGBuITers
-• 174 initGMasks
-- 180 DrawEllipse
-- 186 AreaEllipse
-- 192 LoadRGB4
-- 198 InilRastPorl
--204 InitVPort
--21O MrgCop
--216 MakeVPort
--222 LoadView
-- 228 WaiLBlil
-• 234 SelRast
-- 240 Move
- 246 Draw
- 252 AreaMove
-- 258 AreaDraw
- 264 AreaEnd
- 270 WaitTOF
-• 276 QBlit
- 282 InitArea
- 288 SetRGB4
- 294 QBSBlit

-300 BUClear
-306 RectFill
-312 BltPaUern
-318 ReadPixel
- 324 WrkePixel
- 330 Flood
-336 PolyDraw
-342 SelAPen
-348 SelBPen
-354 SetDrMd
- 360 initView
- 366 CBump
-372 CMove
-378 CWaR
-384 VBeamPos
-390 initBHMap
-396 ScrollRaster
-402 WaitBOVP
- 408 GetSprite
-414 FreeSprite
- 420 ChangeSprite
- 426 MoveSprite
- 432 LockLayerRom
- 438 Unlockl^ayerRom
- 444 SyncSBHMap
- 450 CopySBitMap
- 456 OwnBÜUer
- 462 DisownBHUer
- 468 initTmpRas
- 474 AskFont
- 480 AddFont
- 486 RemFont
- 492 AllocRaster
- 498 FreeRasler
- 504 AndRectRegion
-510 OrRectRegion
- 516 NewRegion
- 522 ClearReclRegion
- 528 ClearRegion
- 534 DisposeRegion
-540 FreeVPortCopLists
- 546 FreeCopList
- 552 ClipBlit
- 558 XorReclRegion
-• 564 FreeCprUst

- 570 GelColorMap
- 576 FreeColorMap
- 582 GetRGB4
- 588 ScrollVPorl
- 594 UCopperListlniL
- 600 FreeGBuflers
-606 BltBitMapRnstRul
-612 OrRegionRegion
- 618 XorRegionRegion
- 624 AndRegionRegion
- 630 SetRGB4CM
-636 BltMaskBit-MapRastPort
- 642 GfxlnlernalI
- 648 Grxlnlernal2
- 654 AttemplLock-LayerRom
- 660 GfxNew
- 666 GrxFree
- 672 GfxAssociaLe
- 678 BiLMapScale
- 684 SoalerDiv
- 690 TexLExtenl
- 696 TexlFit
- 702 GixLookUp
- 708 VideoConlrol
- 714 OpenMonilor
- 720 CloseMonitor
- 726 FindDisplaylnfo
- 732 NexLDisplaylnlb
- 756 GelDisplaylnfo-Data
- 762 FonlExtent
- 768 ReadPixelLine8
- 774 WritePixelLineS
- 780 ReadPixelArrayS
- 786 WrileHxelArrayS
- 792 GetVPModelD
- 798 ModeNolAvailable
- Ö04 WeighTAMalch
-b\0 EraseReot
-í-16 ExtendFonL
- ^22 StnpFont
- 828 CalcIVG
- &34 AttachPalExtra

- 840 ObtainBestPenA
-•846 Glxlnlernal3
-•852 SetRGB32
-•858 GelAPen
-• 864 GetBPen
-• 870 GetDrMd
- 876 GelOutlinePen
- 882 LoadRGB32
- 888 SetChipRev
--894 SetABPenDrMd
- 900 GetRGB32
-•906 GlxSparel
-•918 AllocBilMap
- 924 FreeBRMap
- 930 GetExtSprileA
- 936 CoerceMode
- 942 ChangeVPBilMap
-• 948 ReleasePen
- 954 ObtainPen
-•960 GetBitMapAUr
- 966 AllocDBunnfo
- 972 FreeDBufTnfo
- 978 SelOutlinePen
-•984 SeLWriteMask
- 990 SetMaxPen
-996 SetRGB32CM
-1002 ScrollRasterBF
-1008 FindColor
-1014 GfxSpare2
-1020 AllocSpriteDataA
-1026 ChangeExtSpriteA
-1032 FreeSpriteDala
-1038SelRPAUrsA
-1044 GetRPAürsA
-1050 BestModelDA

2.4 Az Asl Library

Az Amigának eddig is jó Requester-ei vollak (pl.. Req.Library). de a 2 O-ás rendszerrel megjelent az Asl.Library. Ez a gyári Workbench lemezen laláihatő a libs: könyvtárban. A lile Requester szolgáltatásán kívül még Font **lll** a 3.0-ás rendszerhez adott library screenmode Requester-t is meg tud nyitni. Vannak olyan Tagltem-ek melyek BOOL típusúak. Ezeknél meg kell(!) adni. hogy az igaz (TRUE). vagy hamis (FAI^SE). nem elég csak beírni a listába, az még nem jelenti hogy igaz! Az Initial... értékek csak egy értéket adnak meg. ettől a hozzá tartozó gadget nem jeleníti meg. A megjelenítést a Do.... Tag-ok csinálják. Most pedig ennek a kis library-nek nézzük meg a rutinjait.

AllocFileRequest - Asl FileRequester struktúrának helyfoglalás.

struct FileRequester *RllocFileRequest(uoid);
DO

OlTset:- 3O($1E)

A Commodore által kiadott autodocs dokumentáció szerint ez a rutin idejét múlta, ne használjuk, helyette inkább az AllocAslRequest-et használjuk úgy, hogy típusnak ASL_FileRequest-et adunk meg.

FreeFileRequest - lefoglalt FileRequest felszabadítása.

uoid FreeFileRequest(struct FileRequester *)
AO

Offset:- 36 ($24)

Az AllocFileRequest rutinnal lefoglalt (ez persze nem igaz. mert a/ AllocAslRequest-et használtunk, ugye...) struktúrát tudjuk felszabadítani

RequestFile - FileRequesler megjelenítése.

BOOL RequestFiletstruct FileRequester *)
DO AO

Offsel:- 42 ($2A)

Mogjpipníti a FiieRequester-t. Az user kedvére válogathat. DO bn visszakapjuk, hogy választott-e, akkor DO mutat a lile nevére, vagy lm Cumd-t nyomott akkor a DO értéke nulla.

Sajnos nem tudjuk beállítani, hogy mi legyen a Drawer vagy a File mezőben, hiába próbálkozunk ákár a struktúra átírásával is. ami egyébként az Asl library-nél szigorúan tilos! Tehát inkább ezt se használjuk.

AllocAslRequest - Asl Requester-nek helyfoglalás.

```
fiPTR HllocHslRequester(ULONG reqType, struct TagItem *);
DO                      DO          AO
```

OíTset:- 48 ($30)

Ez már az a rutin amit nyugodtan használhatunk, teljesen el tudjuk állítgatni minden paraméterét. Először is DO-ba meg kell adni. hogy milyen típusú Requester-t szeretnénk megnyitni, amelyek az alábbiak lehetnek.

típus:	Ezt kapjuk vissza:
- ASL_FileRequest	mutatót egy FileRequester struktútára
- ASL_FontRequest	mutatót egy FontRequester struktúrára
- ASL_ScreenModeRequest	mutatót egy ScreenModeRequester struktúrára

Ha a visszakapott érték nulla, akkor valami gond van. nem sikerült a kért struktúrát lefoglalni. A ScreenModeRequest csak V38-tol van.

Ezután már csak a Requester paramétereit kell megadni, melyek mindhárom típus esetében mások. Ha nem adunk meg semmit, akkor sincs baj. mert minden értéknek van default beállítása. Még egyszer szeretném elmondani, hogy mindent megtalálunk a struktúrában, de ne azokat módosítsuk. azok az adatok csak olvashatóak! Tehát az AO-nak egy TagList-re kell mutatnia, mely tartalmazza a beállításokat, melyek az alábbiak lehetnek:

FileRequester esetén:

ASLFR_Window (struct Window *) - ehhez az ablahkoz kapcsolja majd a megnyitott Requester-t. (V36!)

ASLFR_PubScreenName (STRFPR) - megadhatjuk, hogy melyik Pubscreen-en nyissa meg az ablakot. Ha ez meg van adva. akkor semmisnek tekinti az ASLFRJWindow t. (V38!)

ASLFRJBcreen (struct Screen *) - melyik screen-en nyissa meg az ablakot. Az ASLFR3Vindow-t és az ASLFR_PubSereenName-t semmisnek tekinti, ha ezt megadjuk. (V38!)

ASLFR_PrivateIDCMP (I3OOL) - ha igazra van állítva, akkor az Asl lefoglal egy új IDCMP portot az ablak számára. Ha ez nincs beállítva és az ASLFR_Window meg van adva. akkor annak az ablaknak az IDCMP portját fogja használni. (V38!)

ASLFRJntuiMsgFunc (struct Hook *) - megadhatjuk, hogy mii hívjon meg ha egy ismeretlen Intuition üzenet érkezne a Requester ablak message portjára.
AO - (strurt Hook *)
Al - (struct intuiMessage *)
A2 - (struct FileRequester *) (V38!)

ASLFR_SleepWindow (BOOL) - ha be van állítva és megadtuk az ASLFR_Window-t. akkor „busy" egérpointert kapunk ha aktiváljuk azt az ablakot. Ilyenkor abban az ablakban minden gadget és menü inaktív lesz. (V38!)

ASLFR_UserData (AFrR) - megadhatunk egy 32 bites értéket, mely csak saját használatra van fenntartva. Az érték az fr_UserData mezőbe másolódik be. (V38!)

ASLFRJTextAttr (struct TexlAttr *) - megadhatjuk, hogy az ablakban milyen fontkészletet használjon. Ha nincs, vagy nem adtuk meg. akkor a képernyőhöz tartozó default fontot fogja használni. A megadott font-nak a memóriában kell lennie, tehát előtte nekünk meg kell nyitni az OpenDiskFont() rutinnal. (V38!)

ASLFR_Locale (struct Locale *) - megadhatjuk, hogy az ablak milyen „nyelvet" használjon, egyébként a default-ot fogja használni. Ez a nyelv vonatkozik a gadget-ekre és a menükre is. (V38!)

ASLFRJTitleText (STRITR) - megadhatjuk, hogy mi legyen az ablak neve. Ha nem adjuk meg. akkor az ablak név üres lesz. (V36!)

ASLFR_PositiveText (STRPTR) - megadhatjuk, hogy mi legyen az OK gadget helyett, de V38 előtt csak max 6 karakter lehet. (V36!)

ASLFR_NegativeText (STRPTR) - megadhatjuk, hogy mi legyen a Cancel gadget helyett, de V38 előtt ez is csak max 6 karakter lehet. (V36!)

ASLFRJnitialLeftEdge (WORD) - az ablak bal felső sarkának x koordinátája. (V36!)

ASLFRJnitialTopEdge (WORD) - az ablak bal felső sarkának y koordinátája. (V36!)

ASLFRJnitialWidth (WORD) - az ablak szélessége, de ha nem fér ki, akkor automatikusan kisebbre igazítja. (V36!)

ASLFRJnitialHelght (WORD) - az ablak magassága, ha ez sem férne ki.

ASLFR_InitialFile (STRPTR) - megadhatjuk, hogy az ablak megnyitásakor mi legyen a default filenév. (V36!)

ASLFRJnitialDrawer (STRPTR) - megadhatjuk, hogy mi legyen a default elérési útvonal. Ha nem adunk meg semmit, akkor az aktuális könyvtár listáját fogja megadni. Ha nem tudjuk a meghajtók neveit, akkor érdemes "RAM:" vagy "SYS:"-t megadni, ezek tuti, hogy vannak. (V36!)

ASLFRJnitialPattem (STRFrR) - megadhatjuk, hogy mi szerint készítse el a filelistát. Például a ".info"-ra végződő file neveket ne jelenítse meg. Az alapbeállítás, a "#?". ez mindent megjelenít. (V36!)

ASLFR_Flagsl (ULONG) - megadhatjuk a Requester opcióit, melyekre V38-tól külön Tagltem-k vannak. Ezek az alábbiak:

. FRF_FILTERFUNC
FRFJNTUIFUNC
FRF_DOSAVEMODE
FRF_PRIVATEIDCMP
FRF_DOMULTISELECT
FRF_DOPATPERNS A default érték 0. (V36!)

ASLFR_Flags2 (ULONG) - mint az előző:

FRF_DRAWERSONLY
FRF_FI LTERDRAWERS
FRF_REJECTICONS A default érték 0. (V36!)

ASLFR_DoSaveMode (BOOL) - ha igaz. akkor savé üzemmódba működik, csak egy file-t adhatunk meg (tehát nincs multiselect) és inverz lesz az egész. (V38!)

ASLFR_DoMultiSelect (BOOL) - ha igaz. akkor a shift lenyomása mellett több file-t is ki tudunk jelölni. A default értéke a hamis. (V38!)

ASLFR_DoPatterns - (BOOL) - ha igaz. akkor megjeleníti a pattern gadget-et. ekkor lehet megadni az ASLFR_lnitialPaUern-t. Alapértelmezés szerint nincs bekapcsolva. (V38!)

ASLFR_DrawersOnly - (BOOL) - ha igaz, akkor csak a könyvtárak neveit rakja be a listába. Alapértelmezés szerint nincs bekapcsolva. (V38!)

ASLFR_FilterFunc - (struct Hook *) - megadhatunk egy eljárást, melyet minden megtalált filebejegyzéskor meghív, és ha ez igaz értékkel tér vissza. akkor a file nevét beleteszi a listába. A következőket kell megadnunk (V38!):
AO - (struct Hook *)
Al - (struct AnchorPath *)
A2 - (struct FileRequester *)

ASLFR_RejectIcons (BOOL) - ha igaz, akkor nem mutatja meg a Workbench ikonjait. Alapértelmezés szerint nincs bekapcsolva. (V38!)

ASLFR_RejectPattern (UBYTE *) - egy az AmigaDOS által ismerL patter4 lehet megadni, mellyel egyező file-ok nem kerülnek be a listába. Az alapértelmezés szerinti pattern a -(#?). mely mindent megmutat. (V38!)

ASLFR_AcceptPattern (UBYTE *) - mint. az előző, csak épp ezeket rakja be a listába. Az alapértelmezés szerinti pattern a "#?". (V38!)

ASLFRJFilterDrawers (BOOL) - ha igaz, akkor a drawer nevekre is használja a háromféle szűrőt, (a pattern gadget, a reject/accept pattern). Ay alapértelmezés szerint nincs bekapcsolva. (V38!)

ASLFR_HookFunc (AIˈrR) - V36-nál a Flagsl és Flags2-ben megadott FRF_FILTERFUNC és FRFJNTU1FUNC mit hívjon meg. A meghívott résznek így kell kinézni:

ULONG function(ULONG mask, HPTR object, struct FileRequesteir *);

A mask a ASLFR_Flagsl. vagy a FRF^FILTERFUNC vagy a FRFJNTU1FUNC.

Az object mutató egy adat object-re. mely FRFJNTUIFUNC esetén egy (struct IntuiMessage *). FRF_F1LTERFUNC esetén pedig (btruct AnchorPath *). FRF_FILTERFUNC esetén ha a visszatérési érték nulla, akkor a file bekerül a listába, ha nem nulla, akkor nem.

FontRequester esetén: ASLFO_Window (strucl Window *) - ehhez az ablakoz kapcsolja majd a megnyitott Requester-t. (V36!)

ASLFO_PubScreenNatne (STR1TR) - megadhatjuk, hogy melyik Pubscreen-en nyissa meg az ablakot. Ha ez meg van adva. akkor semmisnek tekinti az ASLFR_Window-t. (V38!)

ASLFO_Screen (struct Screen *) - melyik screen-en nyissa meg az ablakot. Az ASLFR_Window-t és az ASLFR_PubScreenName-t semmisnek tekinti, ha ezt megadjuk. (V38!)

ASLFO_PrivateIDCMP (BOOL) - ha igazra van állítva, akkor az Asl lefoglal egy új IDCMP port-ot az ablak számára. Ha ez nincs beállítva és az ASLFO_Window megvan adva. akkor annak az ablaknak az IDCMP portját lógja használni (V38!)

ASLFOJntuiMsgFunc (struct Hook *) - megadhatjuk, hogy mit hívjon meg. ha egy ismeretlen Intuition üzenet érkezne a Requester ablak message port-jára.

AO - (stant Hook *)
Al - (strxict lntuiMessage *)
A2 - (struct FileRequester *) (V38!)

ASLFO_SleepWindow (BOOL) - ha be van állítva és megadtuk az ASLFR_Window-t. akkor „busy" egérpointert kapunk ha aktiváljuk azt az ablakot. Ilyenkor abban az ablakban minden gadget és menü inaktív lesz. (V38!)

ASLFO_UserData (AFFR) - megadhatunk egy 32 bites értéket, mely csak saját használatra van fenntartva. Az érték az fr_UserData mezőbe másolódik be. (V38!)

ASLFO_TextAttr (struct TextAttr *) - megadhatjuk, hogy az ablakban milyen fontkészletet használjon. Ha nincs, vagy nem adtuk meg, akkor a képernyőhöz tartozó default fontot fogja használni. A megadott font-nak a memóriában kell lennie, tehát előtte nekünk meg kell nyitni az OpenDiskFont() rutinnal. (V38!)

ASLFO_LocaIe (struct Locale *) - megadhatjuk, hogy az ablak milyen „nyelvet" használjon, egyébként a default-ot fogja használni. Ez a nyelv vonatkozik a gadget-ekre és a menükre is. (V38!)

ASLFOJTitleText (STRFFR) - megadhatjuk, hogy mi legyen az ablak neve. Ha nem adjuk meg. akkor az ablak név üres lesz. (V36!)

ASLFO_PositiveText (STRPTR) - megadhatjuk, hogy mi legyen az OK gadget helyett, de V38 előtt csak max 6 karakter lehet. (V36!)

ASLFO_NegativeText (STRPTR) - megadhatjuk, hogy mi legyen a Cancel gadget helyett, de V38 előtt ez is csak max 6 karakter lehet. (V36!)

ASLFOJnitialLeftEjdge (WORD) - az ablak bal felső sarkának x koordinátája. (V36!)

ASLFOJnitialTopEdge (WORD) - az ablak bal felső sarkának y koordinátája. (V36!)

ASLFOJnitialWidth (WORD) - az ablak szélessége, de ha nem fér ki akkor automatikusan kisebbre igazítja. (V36!)

ASLFO_InitialHeight (WORD) - az ablak magassága, ha ez sem lérne ki. akkor ezt is kisebbre igazítja. (V36!)

ASLFOJnitialName (STRPTR) - megadhatjuk, hogy az ablak megnyitásakor mi legyen a default font név. Ha nem adjuk meg. akkor nincs. (V36!)

ASLFO_InitialSize (UWORD) - megadhatjuk, hogy mi legyen a default font mérete. Alapértelmezés szerint 8. (V36!)

ASLFOJnitialStyle (UBYTE) - megadhatjuk, hogy milyen legyen a Iont stílusa. A deláult FS_NORMAL. A stílusokat a indude/graphies/text.h fileben találjuk. (V36f)

ASLFOJnitialFlags (UBYTE) - az Ib_Flags mező ériékét adhatjuk meg. Itt beállíthatjuk, hogy milyen legyen a font: RomFont. DiskFont. stb. A deľault az FPF_ROMFONT. (V36!)

ASLFOJnitialFrontPen (UBYTE) - beállíthatjuk, hogy a megjelenített példa font-ok kirajzolásához melyik színt használja (fo_FrontPen). A default érték 1. (V36!)

ASLFOJnitialBackPen (UBYTE) - a példa font-ok hátterének színe (foJiackPen). A default érték 0. (V36!)

ASLFCMnitialDrawMode (UBYTE) - milyen drawrnode-ot használjon a font-ok kirajzolásához. A defaul a JAM1. (V38!)

ASLFO_Flags (ULONG) - megadhatjuk a Requester opcióit, melyekre V38-IÓ1 külön Tagltem-k vannak. Ezek az alábbiak:

FOF_DOFRONTPEN
FOF_DOBACKPEN
FOF_DOSTYLE
FO F_ÜO DRAWM O D E
FOF_F1XEDW1DTHONLY
FOF_PR1VATE1DCMP
FOFJNTUIFUNC
FOF_FILTERFUNC A derault érték 0. (V36!)

ASLFO_DoFrontPen (BOOL) - ha igaz. akkor egy Front Color gadget-et is kirak. A default a hamis érték. (V38!)

ASLFO_DoBackPen (BOOL) - ha igaz. akkor egy Bark Color gadget-et is kirak. A defaul a haniií, érték. (V38!)

ASLFO_DoStyle (BOOL) - ha igaz. akkor egy Style Checkbox-ot is kirak. A default a hamis érték. (V38!)

ASLFO_DoDrawMode (BOOL) - ha igaz. akkor megjelenít egy Mode Cycle Gadget-et is, melyei a DrawMode-t tudjuk beállítani. A default érték a hamis. (V38!)

ASLFO_FixedWidthOnly (BOOL) - ha igaz. akkor csak a fix szélességű fontkészletek lesznek benne a listában. Például ilyen a topaz8 is. A delault érték a hamis. (V38!)

ASLFO_MinHeight (UWORD) - a minimális font magasság, amit az user még kiválaszthat. A default érték 5. (V36!)

ASLFO_MaxHeight (UWORD) - a maximális fonlmagasság, amit az user még kiválaszthat A default érték 24. (V36!)

ASLFO_FilterFunc (struct Hook *) - megadhatunk egy eljárást, melyet minden megtalált font bejegyzéskor meghív és ha ez igaz értékkel tér vissza . akkor a font nevét beleteszi a listába. A következőket kell megadnunk (V38!):
AO - (struct Hook *)
Al - (struct AnchorPath *)
A2 - (struct FileRequester *)

ASLFO_HookFunc (APTR) - V36-nál a ASLFO_Flags-ben mégadott FOF_FILTERFUNC és FOFJNTUIFUNC mit hívjon meg. A meghívott résznek így kell kinézni:

ULONG functioníULONG mask, RPTR object, struct FileRequester *);

A mask a ASLFR_Flags. vagy a FOF_FILTERFUNC. vagy a FOFJNTUI-FUNC Az object mutató egy adat object-re. mely FOFJNTUIFUNC esetén egy (struct IntuiMessage *). FOFJ^ILTERFUNC esetén pedig (struct TextAttr *). FOFJ^ILTERFUNC esetén ha a visszatérési érték nulla, akkor a lile bekerül a listába, ha nem nulla, akkor nem.

ASLFOJWaxFrontPen (UI5YTE) - a maximum Pen-ek száma, melyet az user megadhat az ASLFOJJoFrontPen tag-nál. A default érték 255. (V40!)

ASLFOJVIaxBackPen (UBYTE) - a maximum Pen-ek száma, melyet az user megadhat az ASLFOJJoBackPen tag-nál. A default érték 255. (V40!)

ASLFOJVIodeList (STRPTR *) - átírhatjuk, hogy a drawmode-nál megjelenített szövegek mik legyenek. Eredetileg „Text". „Text+Field" és „Complement". Ezek a JAM1. JAM2 és COMPLEMENT módok. A megadott lista első eleme tartalmazza a gadget nevét, majd a következő string-ek pedig a megjelenítendő szövegeket tartalmazza. Egy NULL string-el zárjuk le a listát. (V36!)

ASLFOJFrontPens (UBYTE *) - mutató egy táblázatra, mely Pen értékeket tartalmaz, hogy az ASLFO_DoFrontPen mely színeket használja. A táblázatban levő értékek szániának meg kell egyezni a megjelenítendő színek számával. A deiaul egy NULL pointer. (V40!)

ASLFOJ3ackPens (UBYTE *) - mint az előző, csak az ASLFOJ3oBackPen által használt színekre vonatkozóan. A deláult itt is egy NULL pointer. (V4O!) ScreenModeRequester esetén (csak V38-tól):

ASLSMJWindow (struct Window *) - ehhez az ablakoz kapcsolja majd a megnyitott Requester-t (V36!)

ASLSM_PubScreenName (STRPTR) - megadhatjuk, hagy melyik pubscreen-en nyissa meg az ablakot. Ha ez meg van adva. akkor semmisnek tekinti az ASLFR_Window-t. (V38!)

ASLSM_Screen (struct Screen *) - melyik screen-en nyissa meg az ablakot. Az ASLFR_Window-t és az ASLFR_PubScreenName-t semmisnek tekinti, ha ezt megadjuk. (V38!)

ASLSM_PrivateIDCMP (BOOL) - ha igazra van állítva, akkor az Asl lefoglal egy új IDCMP port-ot az ablak számára. Ha ez nincs beállítva és az ASLSM_Window meg van adva. akkor annak az ablaknak az IDCMP portját fogja használni. (V38!)

ASLSM_IntuiMsgFunc (struct Hook *) - megadhatjuk, hogy mit hívjon meg. ha egy ismeretlen Intuition üzenet érkezne a Requester ablak message port-jára.
AO - (struct Hook *)
Al - (struct IntuiMessage *)
A2 - (struct FileRequester *) (V38!)

ASLSM_SleepWindow (BOOL) - ha be van állítva és megadtuk az ASLFR_Window-t. akkor „busy" egér pointert kapunk ha aktiváljuk azt az ablakot. Ilyenkor abban az ablakban minden gadget és menü inaktív lesz. (V38!)

ASLSM_UserData (AITR) - megadhatunk egy 32 bites értékel, mely csak saját használatra van fenntartva. Az érték az fr_UserData mezőbe masolódik be. (V38!)

ASLSMJTextAttr (struct TextAUr *) - megadhatjuk, hogy az ablakban milyen fontkészletet használjon. Ha nincs, vagy nem adtuk meg. akkor a képernyőhöz tartozó Default Font-ot fogja használni. A megadott Font-nak a memóriában kell lennie, tehát előtte nekünk meg kell nyitni az OpenDiskFont() rutinnal. (V38!)

ASLSM_Locale (struct Locale *) - megadhatjuk, hogy az ablak milyen „nyelvet" használjon, egyébként a delault-ot fogja használni. Ez a nyelv vonatkozik a Gadget-ekre és a menükre is. (V38!)

ASLSMJTitleText (STOPFR) - megadhatjuk, hogy mi legyen az ablak neve. Ha nem adjuk meg. akkor az ablak név üres lesz. (V36!)

ASLSM JPositiveText (STRFPR) - megadhatjuk, hogy mi legyen az OK gadget helyett, de V38 előtt csak max. 6 karakter lehet. (V36!)

ASLSMJVegativeText (STRPrR) - megadhatjuk, hogy mi legyen a Cancel Gadget helyett, de V38 elölt ez is csak max. 6 karakter lehet. (V36!)

ASLSM_InitialLeftEdge (WORD) - az ablak bal felső sarkának x koordinátája. (V36!)

ASLSMJnitialTopEdge (WORD) - az ablak bal felső sarkának y koordinátája. (V36!)

ASLSMJnitialWidth (WORD) - az ablak szélessége, de ha nem fér ki akkor automatikusan kisebbre igazítja. (V36!)

ASLSMJnitialHeight (WORD) - az ablak magassága, ha ez sem férne ki, akkor ezt is kisebbre igazítja. (V36!)

ASLSMJnitialDisplayID (ULONG} - beállíthatjuk, hogy mi legyen az eredetileg megajánlott Srreenmode (smDisplaylD). A defaul értek a 0. mely a LORES_KEY. "(V38!)

ASLSMJnitialDisplayWidth (ULONG) - beállíthatjuk, hogy mi legyen a Width Gadget-ben az érték, ez a képernyő szélessége (sm_DisplayWidth). A default érték 640. (V38!)

ASLSMJnitialDisplayHeight (ULONG) - beállíthatjuk, hogy mi legyen a Height gadget-ben az érték, ez a képernyő magassága (sm_DisplayHeight). A default érték 200 (V38!)

ASLSMJnitialDisplayDepth (UWORD) - beállíthatjuk, hogy mi legyen a Colors Gadget-ben az érték, ez a képernyő mélysége, vagyis a színek száma (sm_DisplayDepth). A default érték 2. (V38!)

ASLSMJnitialOverscanType (UWORD) - beállíthatjuk, hogy mi legyen a Overscan Type Cyde Gadget-ben (sm_OverscanType).

V38-nál:

0	: Regülar Size
OSCAN_TEXT	: Text Size
OSCAN_STANDARD	: Graphics Size
OSCAN_MAXIMUM	: Maximum Size

V39-töl:

OSCAN_TEXr	: Text Size
OSCAN_STANDARD	: Graphics Size
OSCAISLMAXIMUM	. Extrémé Size
OSCAINLVIDEO	• Maximum Size

Az OSCANJVIDEO V39-től van. es a 0 itt már OSCAN_TEXT-et jelenti.

ASLSM_InitialAutoScroll (BOOL) - ha igaz. akkor az Autoscroll Gadget-ben az autoscroll be lesz állítva (sm_AutoScroll). A default az igaz érték. (V38!)

ASLSM JnitialInfoOpened (BOOL) - Ha igaz. akkor nyit egy kis ablakot, melyben infókat kapunk az épp kijelölt képernyőmódról. A default a hamis érték. Ehhez nem tartozik Do.... rutin! (V38!)

ASLSM JnitialInfoLeftEdge (WORD) - az info ablak bal felső sarkának x koordinátája. (V38!)

ASLSMJnitialInfoTopEdge (WORD) - az info ablak bal felső sarkának y koordinátája. (V38!)

ASLSMJJoWidth (BOOL) - ha igaz. akkor kirakja a Width Gadgei-et. A default a hamis érték. (V38!)

ASLSMLDoHeight (BOOL) - ha igaz, akkor kirakja a Height Gadgel-et. A default a hamis érték. (V38!)

ASLSM_DoOverscanType (BOOL) - ha igaz. akkor kirakja az Overscan Type Cycle Gadget-et. A default a hamis érték. (V38!)

ASLSM_DoAutoScroll (BOOL) - ha igaz. akkor kirakja az AutoScroll checkbox gadget-et. A default a hamis érték. (V38!)

ASLSM_PropertyFlags (ULONG) - A lehetséges módok mellett megjeleníti azt is, hogy milyen plusz módban tudja azokat megnyitni. Például HAM. ekkor a színek száma 4096 és 16M lehet (ha van Depth gadget), vagy lehet EHB, ekkor a színek száma 64, lehet DPF vagy DPF2 is. Az ASLSM_Property-Mask-ben beállított biteket veszi csak figyelembe. A default érték DIPF_IS_WB, csak azt jeleníti meg, amit workbench képernyőként is meg lehet nyitni. (V38!)

ASLSMJVHnWidth (ULONG) - a minimális képernyőszélesség, amit az user beállíthat. A default érték 16 (V38!)

ASLSMJMaxWidth (ULONG) - a maximális képemyőszélesség. amit az user beállíthat. A default érték 16384. (V38!)

ASLSMJMinHegiht (ULONG) - a minimális képernyőmagasság, amit az user beállíthat. A default érték 16. (V38!)

ASLSM_MaxHeight (ULONG) - a maximális képernyőmagasság, amit az user beállíthat. A derault érték 16384. (V38!)

ASLSM JMinDepth (UWORD) - a minimális képernyőmélység, amit az user beállíthat. A derault érték 1. (V38!)

ASLSM_MaxDepth (UWORD) - a maximális képernyőmélység, amit az user beállíthat. A default érték 24. (V38!)

ASLSM_FilterFunc (stmct Hook *) - megadhatunk egy eljárást, melyet minden megtalált mód bejegyzéskor meghív és ha ez igaz értékkel tér vissza, akkor a font nevét beteszi a listába. A következőket kell megadnunk (V38!):

AO - (struct Hook *)
Al - (ULONG) mode td
A2 - (struct FileRequester *)

ASLSM^CustomSMList (stmct List *) - megadhatunk saját listát is melyből válogathat az user. Ez a List DisplayNode típusú node-okból áll. Ezek voltak a megadható Tag-ok.

FreeAslRequest - lefoglalt Request felszabadítása.

uoíü FreeRslRequest(HPTR);
AO

Offset:- 54 ($36)

Az AllocAslRequestf) által lefoglalt struktúrát és rendszer resource-ket szabadítja fel. Amit már felszabadítottunk, már nem használhatjuk az Asl-Request()-el.

AslRequest - Asl Requester megnyitása.

BOOL RslRequestíflPTR, struct TagItem *);
DO AO A1

Orfset:- 60 ($3C|

A megadott Requester-t tudjuk megnyitni. AO tartalmazza azt. amit az AllocAslRequest visszaadott. Al mutathat egy TagList-re. itt is megadhatjuk a Requester-ünk tulajdonságait. Ha az user cancel,-t nyomott, akkor nulla értékkel tér vissza. Ha nem nulla, akkor a struktúrából kiszedhetjük, hogy az user mit választott.

Végül pedig álljon itt néhány példaprogram. Az assembly nyelvű források a lemezmellékleten vannak, helytakarékosság miatt a könyvben csak a C nyelvű forrásokat közöljük. Az első a File Requesler megnyitására mutat példát, és arra. hogy hogyan lehet megtudni, mit választott az user.

```
/*
 flsl - File Requester
 írta: Prieuara Zsolt 1995.10.05
*/

#include <stdio.h> #include <stdlib.h> #include <clib/asl_protos.h> #include <clib/enec_protos.h> #include <libraries/asl.h> #include <exec/libraries.h>

/* Prototypes */

uoid main(uoiü); uoid CIoseflIK);

/* Structures */

char___stdiou)inl]= "C0N:195/244/445/156/0utput Rblak";

struct Library *RslBase; struct FileRequester 'req;

uoid CloseRII(uoid) {
   if (RslBase) CloseLibrary(HslBase); }

uoid main(uoid) {
   RslBase=OpenLibrary ("asl.library ",39L);
   if (!RslBase)
   {
      printfC'Can't open asl.library U39.\n");
      CIoseRlK);
      exit(0);
  .}
   /* kiírjuk az asl library uerziószámát */
   printfC'Rsl library uersion:%d.%d\n",flslBase->lib_Uersion,Rsl-
Base->lib_Reuision);

   req=fillocRslRequestTags(RSL_FileRequest,
                  RSLFR_TitleTent,"Válassz egy file-t!",
                  HSLFR_lnitialLUidth,3QQ,
                  RSLFR_InitialHeight,300,
                  RSLFR_PositiueTeKt,"Ez az!",
                  RSLFR_NegatiueTent,"Nem!",
                  RSLFR_InitíalDrauier,"RRM:",
                  RSLFR_DoPatterns,TRUE,
                  RSLFR_InitialPattern,"~(#?.info)",   /* a .info
file-okat nem jeleníti meg •/
                  TRG_DONE);
   if(!req)
```

```
{
   printfC'Can't allocate asl request.Xn");
   CloseflllO;
   eKít(O);
}
if (!HslRequestTags(req,THG_DONE))
{
   printfC'fl Nem! gadget-et nyomtad meg!\n");
}
else
{
   printf("Drawer:%s\n",req->fr_Draiuer);
   printfC'Selected file:7os\n",req->fr_File);
}
FreeflslRequest(req);
CloseflllO; }
```

A második példa a ScreenMode Requester megnyitát mutatja meg. Úgy írtam meg. hogy kiszedi a Workbench felbontását és üzemmódját, ezeket ajánlja meg a Requesterben. Ehhez kell a **LockPubScreenfJ,** melynek meg kell adni a screen nevét ("Workbench"). Ez visszaad egy mutatót (persze csak ha létezik az a screen) a workbench screen struktúrájára. Ebből kiszedem, hogy milyen módban van. mekkora a felbontása (width. height) és a színek számát, a Depth. Ezek alapján kitöltőm a struktúrát a megfelelő értékekkel. Ha az user választott egy felbontást, akkor a program megnyit egy olyan képernyőt és kiír arra egy szöveget, majd kilép.

```
/*

 flsl - ScreenMode Requester

 írta: Prieuara Zsolt 1995.10.05

*/

#include ‹stdío.h)
#include <stdlib.h>
#include <clib/asl_protos.h>
#include <clib/enec_protos.h>
#include <clib/Intuition_protos.h>
#include <clib/graphics_protos.h>
#include <libraries/asl.h>
#include <eKec/libraries.h>
#include <Intuítion/screens.h>
/* Prototypes */
```

```
uoid main(uoid);
uoid CloseRlU);

/* Structures and Uariables */

char___stdioujin[]= "C0N:195/244/445/156/0utput flblak";

struct Library *HslBase;
struct ScreenModeRequester *req;
struct Screen *scr;
struct IntuitionBase *IntuitionBase;
struct GfHBase *G.f«Base;
struct RastPort *rp;

struct TeHtRttr topaz8 = { (STRPTR)"topaz.font",8,0H00,0KQ1 };

unsigned long id;       /• a UJorkbench képernyömódja */
unsigned char depth;    /* a UJorkbench mélysége/bítplanes */
unsigned short tuidth;  /* a UJorkbench szélessége */
unsigned short heíght; /* a UJorkbench magassága */

int i;

UUJORD DriPens[]={ 65535 };

uoid CloseRII(uoid) {
    if (ser) CloseScreen(scr);
    íf (req) FreeflslRequest(req);
    if (RslBase) CioseLibrary(HslBase);
    if (IntuitionBase) CloseLibrary((struct Library *)IntuitionBase);
    if (GfxBase) CIoseLíbrarydstruct Library *)(ifxBase);
}

uoid maín(uoid) {
    IntuitínnBase=(struct IntuitionBase *)0penLib-
rary("Intuítion.library",37L);
    if (MntuitionBase)
    {
        príntfC'Can't open Intuition.library u37.\n");
        CloseHlK);
        e»it(O);
    }

    RslBase=OpenLibrary("asl.library",39L);
    if CRslBase)
    {
        printfC'Can't ooen asl.library u39.\n");
        CloseHlK);
        CKÍt(O);
```

```
}

    GfnBase=(struct GfxBase *)OpenLibrary("graphics.lib-
rary",37L);
if (IGfKBase)
{
   printfC'Can't open graphics.library u37.\n"j;
   CloseHlK);
   exít(O);
}

scr=LockPubScreen("UJorkbench");
id=GetUPModelD(&scr->Uieu>Port); /* a wb képernyőmódja */
depth=scr->BitMap.Depth; /* a wb mélysége */
ujídth=scr->UJidth;  /* a wb szélessége */
height=scr->Height;  /* a wb magassága */
UnlockPubScreen("Workbench",scr);

req=HllocRslRequestTags(nSL_ScreenModeRequest,
        HSLSM_TitleTeKt,"Select ScreenMode",
        RSLSM_InítialIüidth,250,      /* Szélesség */
        HSLSM_InitíalHeight,240,      /* Magasság */
        RSLSM_InitialDisplayID,id, /• ezt szedtük ki a wb-ból •/
        RSLSM_InitíalDisplayDepth,depth, /* a ujb mélysége */
        flSLSM_InitialDísplayIlJidth,iDidth, /* a wb szélessége */
        RSLSM_InitíalDisplayHeight,heíght, /* a wb magassága •/
        RSLSM_Inítial0uerscanType,OSCRN_TEKT, /* ouerscan
                                                 típusa •/
        RSLSM_InitíalRutoScroll,TRUE, /* legyen-e autoscroll */
        HSLSM_DoOuerscanType,TRUE,
        RSLSM_DoDepth,TRUE,
        RSLSM_Golllidth,TRUE,
        RSLSM_DoHeight,TRUE,
        RSLSM_DoflutoScroll,TRUE,
        RSLSM_InitialInfoOpened,TRUE, /* legyen info ablak •/
        flSLSM_InitialInfoLeftEdge,400, /* info ablak »
                                           koordinátája */
        TRG_D0NE);

if(!req)
{
   printfC'Can't allocate asl request.Nn");
   ClosefllK);
   eKit(O);
```

```
    }

    if  (!RslRequestTags(req,THG_DONE))
    {
       printf("Cancel-t nyomott az userAn");
       CloseHIK);
       eKit(O);
    }
    else
    {
       scr=OpenScreenTags(NULL,
             SR_Title,"Gadget Screen U1.0",
             SR_Depth,req->sm_DisplayDepth,
             SR_Left,O,
             Sfi_Top,B,
             SR_Type,CUSTOMSCREEN,
             SR_UJidth,req->sm_DisplayWidth,
             Sfl_Height,req->sm_DisplayHeight,
             SR_DisplayID,req->sm_DisplayID,
             SR_Ouerscan,req->sm_OuerscanType,
             SR_Font,G'topaz8,
             SR_Behind,Q,
             SR_Quiet,O,
             SR_Shou)Title,1,
             SR_RutoScroll,req->sm_RutoScroll,
             R_Pens,&DriPens[01,
             RG_DONE);
       if (!scr)
       {
          printfC'Can't open screen.\n");
          CloseRIIO;
          eKit(O);
       }

       rp=&scr->RastPort; /* kiszedjük a RastPort struktúra címét */

       SetRGB32(f>scr->UiewPort,G,0x90000000,0x9001)0000,0K90D00000);
       SetRGB32(&scr->UiewPort,1,0x00000000,0x00000000,0x00000000);
       SetRGB32(&scr->UieiuPort,2,OxfQOOQ0BO,QxfQflGOOQO,QxfflOOOOBO);

       SetRPen(rp,1);  /* az I-es színnel írunk majd */
       Moue(rp,fl,18);  /* grafikus kurzor beállítása */
       Tent(rp,"Uárj egy kicsit és bezáródik a képernyő",39):;

       for(i=0;i<3B0;i++) printf("UJait\n");  /* nem elegáns, de uá-
rakozni jó */
       CloseScreen(scr);
    }
    CloseRIIO; }
```

A harmadik példa a Font Requester-re mutat példát. Azt, hogy mit választott az user csak a req struktúrából tudhatjuk meg.

```
/*

 Rsl - Font Requester

 írta: Príeuara Zsolt 1995.10.05

*/

#include <stdio.h>
#include <stdlib.h>
#include <clib/asl_protos.h>
#include <clib/enec_protos.h>
#include <libraries/asl.h>
#include ‹eKec/líbraríes.h>

/* Prototypes */

uoid main(uoid); uoid CloseHllf);

/* Structures and Uanables */

char___stdiowin[] = "C0N:195/244/445/156/0utput flblak";

struct Library *RslBase; struct FontRequester *req;

uoid CloseRlHuoid) {
   if (HslBase) CloseLibrary(flslBase); }

uoid main(uoíd) {
   flslBase=0penLibrary("asl.library",39L);
   if OflslBase)
   {
      príntfC'Can't open asl.library u39.\n");
      ClosefillO;
      enit(O);
   }

   printf("Rsl library uersion:%d.%d\n",flslBase->lib_Uersion,Rsl-
Base->lib_Reuisíon);

   req=flllocflslRequestTags(RSL_FontRequest,
                     RSLFO_TitleTeKt,"Uálassz egy font-ot!",
                     RSLF0_lnitialUJidth,300,
                     RSLF0_lnitialHeight,300,
                     RSLFO_PositiueTeKt,"Ez az!",
                     RSLFO_NegatiueTeKt."Mégsem",
```

```
                        HSLFO_DoStyle,TRUE,  /* stílusok: plain, bold,
                                             italic, underlmed */
                        HSLFO_DoFrontPen,TRUE,
                        nSLFO_DoBackPen,TRUE,
                        HSLFO_DoDraiuMode,TRllE,   /« draiumodes:
text, text+fíeld, complement */
                        THG_D0NE);
    if (!req)
    {
        printfC'Can't allocate asl request.W);
        ClosenilO;
        eKÍt(O);
    }
    if  (!HsinequestTags(req,TRG_DONE))
    {
        printf("Cancel-t nyomott az userAn");
    }
    else
    {
        príntfC'Kiuálasztott egy fontkészletet az user.Xn");
    }

    FreeHslRequest(req);

    ClosefllO; }
```

2.4.1 Az Asl *Library függvényei offszet sorrendben*

- 30 AllocFileRequest	- 48 AllocAsiRequest	-54 FreeAslRequesl
- 36 FreeFileRequest	- 42 RequestFile	- (50 AslRequest

2.5 A GadTools Library

A 2.0-ás rendszer külső megjelenéséhez nagyban hozzájárulnak a GadTodls Library lehetőségei. Ezért fontosnak tartottuk a Library leírását is közzétenni. A leírás ugyan nem lesz részletes, de igyekeztük jól használhatóra megszerkeszteni. A leírás formailag követi az ebben a fejezetben leírt Library-k leírását.

CreateContext - létrehozza és elhelyezi a Gadtools contexl adatait. (V36!)

struct Gadget *CreateContext(struct Gadget **Gaget);
DO AO

Offset:- 114 ($72)

A függvény létrehozza és elhelyezi a context (összekötő szöveg) adatokat amik- az ablaknak szükségesek. A függvényt azelőtt kell meghívni-, mielőtt a Gadget-eket létrehoznánk. A függvény paraméterként egy pointert vár a gadget-listánk pointerére. Visszatérési értékként a Context gadgetpointerével tér vissza, vagy NULL. ha hiba következett be. A függvényre bemutatunk egy tipikus alkalmazást is:

```
struct Gadget *Gad;
struct Gadget *glist = NULL;

gad = CreateContextf&glist);

/* fl Gadget-eket létrehozó rész itt köuetkezik. •/

if(gad)
{
    myNewUJindOLU.FírstGadget = glist;
    iffmyNeuiúJindoui = OpenUJindouiC&myNeLuuJindoui))
    {
        GT_RefreshLUindouj(rnýuJindoijj, NULL);
        /* más egyéb */
        CloseUJíndoLu(myUJíndow);
    }
}
FreeGadgets(glist);
```

CreateGadgetA - allokál és inicializál egy GadTools-os Gadget-et. (V36!)

CreateGadget - minL fent. csak a tagok eggyessével megadhatók. 0/36!)

```
struct Gadget *CreateGadgetR(ULONG Kind, struct Gadget *preunous,
DO                                DO          AO
struct NewGadget *newgad, struct TagItem *taglist);
A1                                A2
```

```
struct Gadget *CreateGadgetR(ULONG Kind, struct Gadget *preuious,
struct NeuiGadget *newgad, Tag, ...);
```

Offset:- 30 ($1E)

A függvény allokálja és létrehoz egy gadget-et amit aztán hozzáfűz a previous argumentumként megadott gadget-hez. A létrehozandó gadget tulajdonságai a Kind paraméterben megadhatók. Ezenkívül a gadget-et a megadott NewGadget struktúra tartalma is befolyásolja. A Kind argumentumnál megadottakat a Taglistában (vagy a tagok) paraméterezhetjük. A Kind-ok leírását megtaláljuk az 1.6.4 részben. A függvény sikeres létrehozás esetén a létrehozott gadget címét adja vissza, ha nem sikerült, akkor NULL-t.

CreateMenusA - allokálja és kitölti a menüstruktúrát. (V36!)

CreateMenus - mint a fentebbi, csak a tag-ok egyesével adhatók meg. (V36!)

```
struct Menü *CreateMenusfl(struct NewMenu "neujmenu,
DO                         AO
struct TagItem *TagList);
A1
```

```
struct Menü *CreateMenusfl(struct NeuiMenu *neujmenu, Tag, ...);
```

Offset:- 48 ($30)

A függvény allokálja, és a megadott taglista (tagok), és NewMenü inicializáló tömb alapján kitölti a menü struktúrát. A menükhőz kapcsolódó tag-ok listáját megtalálhatjuk az 1.7.4 részben. A függvény visszatérési értéke természetesen az allokált es kitöltött menü címe lesz. Ha nem sikerült a/, allokáció, a NULL értékkel ter vissza.

DrawBevelBoxA - kirajzol egy „háromdimenziós"· keretet. (V36!)

DrawBevelBox - mint fent, csak a tag-ok egyesével adhatók meg. (V36!)

```
uoid DrawBeuelBoxR(struct RastPort *Rp, WORD left, LUORD top,
                   AO                   DO        D I
WORD wídth, LUORD height, struct TagItem *TagList);
D2          D3            A1
```

```
uoid DrawBeue1RDu(struct RastPort *RD, LUORD left, WORD top,
LUORD width, WORD height, Tag, ...);
```

Offset:- 120 ($78)

A függvény segítségével a megadott rastport-ra rajzolhatunk a tag-oknál megadható vonal fajtával, az egyéb argumentumként megadható dimenzióban. A A tag-okról részletesen az 1.6.3.4 részben.

FreeGadgets - Gadget lista felszabadítása. (V36!)

```
uoid FreeGadgetsfstruct Gadget *glist);
                    AO
```

Offest:- 36 ($24)

Felszabadítja az összes GadTools Gadget-et a megadott-tói kezdve. Minden memóriarészt felszabadít, amit a CreateGadgetAO függvénnyel lefoglaltunk. A paraméterében az első felszabadítandó gadget címét várja.

FreeMenus - felszabadítja az CreateMenuQ által lefoglat területeket. (V36!)

```
uoid FreeMenusístruct Menü *menu);
                  AO
```

Offset:- 54 ($36)

Felszabadítja az CrealeMenu() által lefoglat területeket. Ha a függvényt NULL paraméterrel hívjuk, nem okoz problémát. Paraméterként pointert vár a menü struktúrára, vagy az első' Menultem-re amit a CreateMenusAO adott vissza.

FreeVisualInfo - visszaadja a GetVisualInfof) által adott adatokat. (V36!)

```
uoid FreeUisualInfoíRPTR ui);
                    AO
```

Offset:- 132 ($84)

Ezt a függvényt csak akkor, kell meghívnunk amikor befejeztük a Gadgetekkel végzett munkánkat és bezártuk az ablakot, de a Screen még létező, (pl., előtte a CloseScreen()-nek vagy az UnlockPubScreen()-nek.) A paraméterként megadott vi a GetVisualInfofJ által visszadott érték kell. hogy legyen.

GetVisualInfoA - bekéri a Gadtools-nak szükséges kinézeti információkát. (V36!)

GetVisualInfo - mint fent. csak a tagok egyesével megadhatók. (V36!)

```
HPTR GetUisualInfoflfstruct Screen *Scr, struct TagItem *taglist);
DO                AO                A1

HPTR GetUisualInfoRístruct Screen *Scr, Tagi, ...);
```

Ofrset:- 126 ($7E)

Bekéri a Screen információit (melyik kiírás milyen színű stb.). amik a GadTools-nak szükségesek ahhoz, hogy a Gadtools által 'étrehozott objektumok (Gadget-ek . menük stb.) is úgy nézzenek ki mint a rendszer. Miután meghívtuk a CloseWindowfJ függvényt a programunkban, meg kell hívnunk a FreeVisuallnfofJ függvényt, ha használtuk ezt a függvényt. A paraméterként megadott screenről szerzi be az információkat. A Tag-oknál egyenlőre még nem lehet megadni semmit, mert még nincs ilyen Tag, így itt NULL t adhatunk meg. A visszatéréskor egy pointert kapunk, ami az információkra mutat. Ezt a visszakapott értéket kell megadnunk a FreeVisuallnfol) függvénynek is. valamint a CreateMenuAfJ. CreateGadgetAfJ függvényeknek a Tag-listán keresztül.

GT_BeginRefresh - frissítés kezdete GadTools barát környezetben. i,V36!)

```
uoid GT_BeginRefresh(struct litindow *iuin);
                              AO
```

Offset:- 90 ($5A)

A függvény hasonló az Intuition library BeginRefreshO függvényéhez. Azzal a különbséggel, hogy az ablak frissítését a GadTools-nak megfelelően teszi. A frissítét a GT_EndRefresh() függvény fejezi be. A függvény meghívása az IDCMP_REFRESHW1NDOW üzenet érkezésekor válik aktuálissá.
Nézzünk rá egy alkalmazást:

```
case IDCMP_REFRESHIDINDOW:
     GT_BeginRefresh(win);
     GT_EndRefresh(ujin, TRUE);
break;
```

GT_EndRefresh - a frissítés vége GadTools barát környezetben. (V36!)

```
uoid GT_EndRefresh(struct Ulindow *win, B001. comlete);
                          AO              DO
```

orfset:- 144 ($60)

A függvény hasonlít az Intuition Library EndRefreshQ függvényhez, és annak leírása megegyezik erre a függvényre is.

GT_FilterIMsg - szűrő az intuiMessage Gadtools-ot érintő részének ki szűrésére. (V36!)

```
struct IntuiMessage *GTJFilterlMsg(struct IntuiMessage *imsg);
DO                                  Al
```

Offset:- 102 ($66)

Ennek a függvénynek a használatára csak néhány szélsőséges esetben lehet szükség. A programjainkban használjuk inkább a GT_GeUMsg() és a GT_ReplyIMsg() függvényeket, és ne a GT_FilterIMsgO és GT_PostFilterIMsgfJ-t. Paraméterként normál intuiMessage pointert vár, melyet az ablakunk struktúrájából az UserPort mezőből nyerhetünk. A függvény ebből a paraméterből kiválasztja a GadToolsra vonatkozó üzeneteket. A függvény NULL értékkel tér vissza, ha az üzenet csak a Gadtools-ra vonatkozik. Ha nem csak arra, akkor a módosított 1DCMP üzenetre. Használjuk a GT_PostFilterlMsg() fügvényt az eredeti IMsg visszanyerésére. Ha a függvény a NULL értékkel tér vissza, az üzenetet nyugtázhatjuk a ReplyMsgO fügvénnyel. A V39-es rendszer használatakor a függvény ExtlntuiMessage struktúra mutatót ad vissza. de a prototípust a kód kompatibilitási okok miatt nem változtatták meg.

GT_GetGadgetAttrsA - bekéri a GadTools Gadget attributtumait. (V39!)

GT_ GetGadgetAttrs - mint fent, de aTag-okat egyenként adhatjuk meg. (V39!)

```
LONG GT„GetGadgetfIttrsfi(struct Gagdet *Gad, struct lüindoui *li)in,
DO                    AO          A1
struct Requseter *Req, struct TagItem *taglist);
A2                    A3

LONG GT_GetGadgetnttrsR(struct Gagdet *Gad, struct UJindoui *Win,
struct Requseter *Req, Tagi, ...);
```

Offset:- 174 ($AE)

A gadget bekéri a megadott gadget azon attributtumát, amelyet szintén megadtunk a paraméterekben. Visszatérési értékként a kívánt adatot adja. A Taglistában megadhatók a következők:

BUTTON_K1ND:
GA_Dissabled (BOOL) -TRUE, ha a gadget nem választható, egyébként FALSÉ. (V39!)

CHECKI3OX_KIND:
GA_Dissabled (BOOL) -TRUE, ha a gadget nem választható, egyébként FALSÉ. (V39!)

GTCB_Checked (I3OOL) - TRUE. ha a gadget választott, egyébként FALSÉ. (V39!)

CYCLE_KIND:
GA_Dissabled (BOOL) -TRUE, ha a gadget nem választható, egyébként FALSÉ. (V39!)

GTCY_Active (UWORD) - a megadott sorszámú (nullától kezdődően) tagja aktív-e a Cycle Gadget-ben. (V39!)

GTCY_Labels (STRPTR •) - az éppen aktuális felajánló stringet ami választható (V3QI)

INTEGER_KIND:
GA_Dissabled (BOOL) -TRUE, ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTIN_Number (ULONG) - Az integer gadget száma. (V36!)

LISTVIEW_KIND:
GA_Dissabled (BOOL) - TRUE. ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTLVJTop (WORD) - a Telső választható item száma. (V39!)
GTLV_Labels (struct List *) - a lista, amely az In_Name mezőben található és a kijelzés alatt áll. (V39!)
GTLV_Selected (UWORD) - az aktuálisan választott item sorszáma. ~0 ha egy sem választott. (V39!)

MX_KIND:
GAJDissabled (BOOL) -TRUE, ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTMX_Active (UWORD) - a megadott sorszámú (nullától kezdődően) tagja aktiv-e az MX gadget-ben. (V39!)

NUMBER.KIND:
GAJDissabled (BOOL) -TRUE. ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTNMJVumber - előjeles integer, amely csak olvasható és a kijeUett számot kapjuk vissza, (default 0) (V39!)

PALETTE_KIND:
GA_Dissabled (BOOL) - TRUE. ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTPA_Color (UBYTE) - a kíválsztott szín a palettáról. (V39!)
GTPA_ColorOffset (UBYTE) - az első használt paletta szín. (V39Ü
GTPA_ColorTable (UBYTE *)- Pointer a tollakat tartalmazó táblázatra. amely szerkesztés alatt áll a Palette Gadget-ben. Lehet NULL is akkor, ha 1-ből 1 látható. (V39!)

SCROLLER_KIND:
GA_Dissabled (BOOL) - TRUE. ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTSC_Top (WORD) - a legfelső látszódó állapot. (V39!)
GTSCJTotal (WORD) - az összes a scroll mezőben. (V39Í)
GTSCJVisible (WORD) - A látszódó állapot száma. (V39!)

SLIDER_KIND:
GA_Dissabled (BOOL) - TRUE. ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTSL_Min (WORD) - A minimum értéke a Knob-nak. (V39!)
CTSL_M<n IWORDI o Knob nioximum éltéke. |VO9!)
GTSLJLevel (WORD) - a Knob akluállis értéke. (V39!)

&TRING_KIND:
GAJMssabled (DOOL) - TRUE ha a gadget nem választható, egyébként FALSÉ. (V39!)
GTST_String (STRPTR)- pointert ad vissza a String Gadget puITerére. (V39!)

TEXT_KIND:
GTTXJText - Pointer a string-re amely osak olvasható a szövegkijelző gadget-ben. (V39!)

Következzen itt egy alkalmazási példa:

```
longtop=0;
long selected = 0;
long uálasz;

ualasz = GT_GetGadgetRttrs(listuiew_gad, min, NULL,
            GTLU_Top, &top,
            GTLU_Selected, &selected,
            TfIG_DONE);
if(ualasz!=2)
{
    printfC'Hiba történt! \n");.
}
```

GT_GetIMsg - kér egy IntuiMessage-t GadTools felhasználásra bővítve. (V36!)

```
struct IntuíMessage *Gt_GetIMsg(struct MsgPort *MsgPort);
DO                                  AO
```

Offset:- 72 ($48)

Ezt a függvényt az Exec Library GetMsgO függvény helyett használjuk akkor, amikor olvasni akarunk egy intuiMessage-et az ablak UserPort-járól. Ha a GadTools segítségével gadget-eket és menüket használunk, akkor ezt a függvényt használhatjuk az üzenetek átvételére. Paraméterében a Window->UserPort-ját várja. A visszatérési értéke akkor NULL. ha csak a GadTools Library kapott üzenetet. A szükséges választ az Intuitionnak a GT_ReplyIMsg() függvénnyel adhatjuk meg. Ha szükségünk van a régi Message átvételére is. akkor használjuk a GT_Filter!Msg() függvényt.

GT_PostFilterIMsg - visszaadja a nem szűrt üzeneteket a GT_FilterlMsgl) meghívása után. (V36!)

```
struct IntuíMessage *GT_PostFílterIMsg(struct IntuiMessage *modímsg);
DO                                  Al
```

Offset:- 108 ($6C)

Ezt a függvényt csak néhány extrém program esetében ajánlatos használni. Helyette használjuk inkább a GT_GetIMsg() és a GT_ReplyIMsg() függvényeket és ne a GT_Fi)terIMsg() és a GT_PostFilterIlVIsg()-t. A többit lásd a GT_FilterIMsg() függvénynél. A függvény paramétere a módosított IMsg, és ebből visszatéréskor az eredeti IMsg-t adja.

GT_RefreshWindow- frissíti az összes GadTools-os gadget-et az ablakon. (V36!)

```
uoid GT_RefreshUJindouj(stnjct UJíndouj *UJin, struct Requester *Req);
                        AO                     Al
```

Orfset:- 84 ($54)

A függvény a GadTools Gadget-ek frissítését végzi el. Az ablak megnyitása után meg kell bogy hívjuk ezt a függvényt. Ha az ablakot a gadget-ek megjelenítése nélkül kell megnyitnunk, használjuk az Intuition AddGListO függvényét a gadget-ek csatolását az ablakhoz. Jelenítsük meg a RefreshGList() függvény segítségével és utána hívjuk meg ezt a függvényt.

Más egyéb esetekben nem szükséges a függvényt meghívni. Paramétereként egyenlőre csak az ablak pointerét várja, amelyre a Gadtools-os Gadgeteket akarjuk kitenni. A Req paraméternek egyenlőre csak a NULL-t kell megadnunk, későlobi felhasználásra fenntartva.

GT_ReplyIMsg - válasz a GT_GetIMsg() által bekért IDCMP üzenetre. (V36!)

```
uoid GT_ReplylMsg(struct IntuiMessage *imsg);
                  AO
```

Offset:- 78 ($4E)

A függvény válaszol az Intuitionnak az IDCMP üzenetre amit az küldött és amit a GT_GetIMsg() függvény segítségével bevettünk. A függvény paramétereként a GT_GetIMsg() függvény által visszaadott módosított IMsg-ét várja. A fügvény hasonló az Exec.Library ReplyMsgQ-éhez. használata megegyezik azzal.

GT_SetGadgetAttisA - megváltoztaja a GadTools Gadgetattribúttumát. |V36!)

GT_SetGadgetAttrs - mint fent. csak a tag-ok egyenként adhatók meg. |V36!)

```
uoid GT_SetGadgelRttrsH(struct Gadget *gad, atruct lilindoui *ujin,
                        AO                  Al
struct Requester *Req, struct TagItem *TagList);
A2                     A3
```

```
uoid GT_SetGadgetRttrs(struct Gadget *gad, atruct UJindow *iuín,
struct Requester *Req, Tagi, ...);
```

Offset:- 42 ($2A)

A függvény a megadott gadget attributtumait állítja be a megadott értékűre a megadott gadgeten, a szintén megadott atributtomon. A függvény paraméterei azt hiszem egyértelműek. A megadható Tagok a/, alábbiak:

BUTTON_KIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott, egyébként FALSE. (V36!)

CHECKBOX_KIND:
GA_Dissabled (BOOL) - Állítsuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V36!)
GTCB_Checked (BOOL) - a Check Box állapota. (V36!)

CYCLE_KIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V37!)
GTCY_Active (UWORD) - a megadott sorszámú (nullától kezdődően) tagja aktív lesz a Cycle Gadget-nek. (V36!)
GTCY_Labels (STRPTR *) - az éppen aktuális felajánló string-et ami választható. (V37!)

INTEGER_KIND:
GA_Dissabled (BOOL) - álliítsuk TRUE-ra ha a gadget nem választott, egyébként FALSE. (V36!)
GTIN_Number (ULONG) - az Integer Gadget új értéke. (V36!)

LISTVIEW_KIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott, egyébként FALSE. (V39!)
GTLV_Top (WORD) - a felső választható item száma. (V36!)
GTLV_MakeVisible (WORD) - annak a tag-nak a száma, amely gyorsan a látható területbe scrollozódjon. (V39!)
GTLVLabels (struct List *) - a lista, amely az ln_Name mezőben található és a kijelzés alatt áll. (V36!)
GTLV_Selected (UWORD) - az aktuálisan választott item sorszáma.(V39!) ~O értéket kapjuk ha (V36!) alól hittatjuk.

MX_KIND:
GA_Dissabled (BOOL) - Állisuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V36!)
GTMX_Actlve (UWORD) - A megadott sorszámú (nullától kezdődően) tagja aktív legyen az MX gadget-ben. (V39!)

NUMBER_KIND:
GTNM_Number - előjeles integer amely legyen kijelzett. (default 0) (V36!)
GTNM_FrontPen (UBYTE) - a megjelenítéskor ezzel a színnel jelenjen meg. (V39!)
GTNM_BackPen (UBYTE)- a kijelzés hátterének színe adható meg. (V39!)
GTNM_Justiflcation (UBYTE) - a megjelenítéskor használt igazítás megadása, (lásd. 1.6.4 részben.) (V39!)

GTNM_Format (STRITR) - C stílusú megjelenítési formátum a megadható konverziós karakterűre konvertálódik a megadott érlek. (V39!)

PALETTE_KIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V36!)
GTPA_Color (UBYTE) - a kiválasztott szín a palettáról. (V36!)
GTPA_ColorOffset (UBYTE) - az első használt paletta szín. (V39!)
GTPA_ColorTable (UBYTE *)- Pointer a tollakat tartalmazó táblázatra. amely szerkesztés alatt áll a Palette Gadget-ben. Lehet NULL akkor, ha 1-ből 1 látható. (V39!)

SCROLLER_K1ND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V36!)
GTSC_Top (WORD) - a legfelső látszódó állapot. (V36!)
GTSCJTotal (WORD) - a/, összes a scroll mezőben. (V36!)
GTSC_Visible (WORD) - a látszódó állapot száma. (V36!)

SLIDERJKIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha'a gadget nem választott. egyébként FALSE. (V36!)
GTSL_Min (WORD) -A minimum értéke a konbnak. (V36!)
GTSLJMax (WORD) - A maximum értéke a konbnak. (V36!)
GTSL_Level (WORD - Az aktuállis értéke a knobnak. (V36!)
GTSL_LevelForaiat (STRITR) - C formátumú megjelenítést biztosít a scroller értékének. (A príntf konverziós karaktereit használhatjuk.) (V39!)
GTSL_DispFunc (LONG (* funcUon) (StructGadget *. WORD)) - függvény megadása az érték kiszámításához. A függvény első paraméterének a gadget pointerének kell lennie, míg a második a szintet jelző WORD. (V39!)
GTSL_Justiflcation (UBYTE) - A szintet jelző kiírás a gadget-hez képes való megjelenítésének módja, [lásd 1.6.4 részben.) (V39!)

STRING_KIND:
GA_Dissabled (BOOL) - állítsuk TRUE-ra ha a gadget nem választott. egyébként FALSE. (V36!)
GTST_String (STRITR)- a String Gadget új értékkel történő inicializálása. Ha NULL-t adunk meg. üres gadget-et eredményez. (V36!)

TEXTJÍIND:
GTTX_Text (STRITR) - a String Gadget uj értékkel történő inicializálása. Ha NULL-t adunk meg, üres gadget-et eredményez. (V36!)
GTTX_FrontPen (UBYTE) - a megjelenítéskor ezzel a színnel jelenjen meg. (V39!)
GTTX_BackPen (UBYTE)- a kijelzés hallereneK színe aunaui uicg.rvoo!)
GTTX_Justiflcation (UBYTE) - a megjelenítéskor hasznait igazítás megadása (lásd. 1.6.4 részben) (V39!).

LayoutMenuItemsA - az összes menüitem pozíciói. (V36!)

LayoutMenuItems - mint fent. csak a tagok egyenként megadhatók. (V36!)

BOOL LayoutMenultemsflístruct Menultem *menuitem, HPTR ui,
DO AO AI
struct TagItem *TagList);
A2

BOOL LayoutMenultemsHístruct Menultem *menuitem, RPTR ui, Tagi,...);

Offset:- 60 ($3C)

A függvény a menüitemek és subitemek kinézetét állítja be, a megadott visualinfo (vi) segítségével, ill. a Tag-ok segítségével. A függvény segítségével a menüket sorbarendezhetjük (formailag) és/vagy átrendezhetjük. A tag-oknál a következők adhatók meg:

GTMN_TextAttr (struct TextAttr *) - a TextAttr. amelyet a menük használjanak a kijelzés során. A Fontnak már megnyitottnak kell lennie. (OpenFontfJ) (V36!)
GTMN_NewLookMenus (BOOL) - ha az ablakon beállított a WA_NewLookMenus, akkor ez a tag ajánlott. Ez informálja a GadTools-t a helyes checkmark ill. Amiga bili. kép használatára. (V39!)
GTMN_CheckMark (struct Image *) - ha saját készítésű képet akarunk használni a checkmark helyett, akkor itt adhatjuk meg a pointerét az őt tartalmazó Image-nak. (V39!)
GTMN_AmigaKey (struct Iamge *) - ha saját készítésű képet akarunk használni az Amiga bili. helyett, akkor itt adhatjuk meg az őt tartalmazó Image pointerét. (V39!)
GTMN_FrontPen (ULONG) - ez a tag a használni kívánt kiírási színt határozza meg. Ha a menün GTMNJVewLookMenuk beállított, a default értéke a screen BARDETAILPEN-e lesz. egyébként semmi. (V39!)

A visszatérési értéke FAI^SE. ha nem sikerült (pl. a TexLAUr nem volt megnyitva.' stb.) és TRUE ha sikerült.

LayoutMenusA - az összes menü pozíciói. (V36!)

LayoutMenus - mint fent csak a tagok egyenként megadhatók. (V36!)

BOOL LayoutMenusfHstruct Menü •menü, RPTR ui,
DO AO AI
struct TagItem *TagLíst);
A2

BOOL LayoutMenus(struct Menü *menu, HPTR ui, Tagi, ...);

Offset:- 66 ($42)

A függvény a menük es azok lteme-einek. és Subltem-einek kinézetét állítja be. a megadott visualinfo (vi) ill a tag-ok segítségével A függvény segítségével a menüket sorbarendezhetjuk (formailag) és/vaj^y átrendezhetjük. A tag-oknál a következők adhatók meg:

GTMNJTextAttr (struct TexlAttr *) - a textAttr. amelyet a menük használjanak a kijelzés osrán. A Font-nak már megnyitottnak kell lennie. (OpenFontf)) (V36!)
GTMN_NewLookMenus (BOOL) - ha az ablakon beállított a WA_NewLookMenus, akkor ez a Tag ajánlott. Ez az informálja a Gadtools-t a helyes checkmark ill. Amiga bili. kép használatára. (V39!)
GTMN_CheckMark (struct Image *) - ha saját készítésű képet akarunk használni a checkmark helyett, akkor itt adhatjuk meg az őt tartalmazó Image pointerét. (V39!)
GTMN_AmigaKey (struct Iamge *) - ha saját készítésű képet akarunk használni az Amiga bili. helyett, akkor itt adhatjuk meg az őt tartalmazó Image pointerét. (V39!)
GTMN_FrontPen (ULONG) - ez a tag a használni kívánt kiírási színt határozza meg. Ha a menün GTMN_NewLookMenuk beállított, a default értéke a screen BARDETAILPEN-e lesz. egyébként semmi. (V39!)

A visszatérési értéke FALSÉ ha nem sikerült (pl. a TextAttr nem volt megnyitva, stb.), és TRUE ha sikerült.

2.5.1 A GadTools Library függvényei offszet sorrendben

-30 CreateGadgetA	-66 LayoutMenusA	-114 CreateContext
-36 FreeGadgets	-72 GT_GetIMsg	-120 DrawBevelBoxA
-42 GT_SetGadget-Attrs	-78 GT_ReplyIMsg	-126 GetVisualInfo
-48 CreateMenusA	-84 Gr_RefreshWindow	-132 FreeVisualInfo
-54 FreeMenus	-90 GT_BeginRefresh	-144 GT_EndRefresh
-60 LavoutMermltemsA	-102 GT_FilterIMsg	-174 GT_GetGadget-AttrsA
	-108 GT_PostFilter	

3. Fejlesztői rendszerek

3.1 PowerWindows

Az Inovatronics cég 1987-ben adta ki a PowerWindows 2.0-ás verzióját, mely annak idején nagyon jó program volt. Az 1.2 és 1.3-as Kickstart-al rendelkező gépekhez grafikusan tudunk tervezni magunknak ablakokat és azokba szép (?!) gadget-eket. Mindezek után a program tud Assembly és C (Lattice és Manx) forrásokat menteni, melybe nekünk már csak az egyes gadget-ek megnyomásakor vagy bármilyen használatkor szükséges rutinokat kell elhelyeznünk. A könyvbe azért vettük bele ennek a programnak a leírását, mert sokan nem is ismerik, és 1.3-as géphez szinte csak ez az egy ilyen prg. van.

A program indításkor nyit magának egy ablakot, de e* csak azért kell. hogy a menüket elhelyezze valahova. Ha elkezdünk vele dolgozni, akkor először nem árt egy screen-t definiálni a **Screen/Define** Screen Type menüponttal.

Ebben beállíthatjuk, hogy a Screen-ünk milyen típusú legyen: WBENCHSCREEN vagy CUSTOMSCREEN. Ha CUSTOMSCREEN-t választunk, akkor egy új screen-t nyit, melynek a színeit ekkor már be tudjuk állítani a **Screen/Screen Palette** menüponttal. Be tudjuk állítani, hogy a screen milyen legyen: Custombitmap. Dual Playfield. Hold & Modify (vagyis HAM) és legyenek-e sprite-ok. A következő a Screen Title. ahol a screen nevét tudjuk meghatározni. A Screen Gadgetlist pointer a NewScreen struktúrában a Gadgets-hez ezt írja be. A screen méreteit a Screen Dimensions gadget-ek közül állíthatjuk be. A Pen numbers-nél a Detail és a Block pen-t tudjuk beállítani. A Number of BitPlanes a BitPlane-ek számát jelenti, ami az egyszerre megjelenítendő színek számát takarja. Ha a Screen Flags-nél Dual PlayField-et vagy Ham üzemmódot választottunk akkor ezt állítsuk 6-ra. A következő a Text Font, ami lehet Default, Topaz8 és Topaz9. Már csak a Custombitmap pointer maradt, ami a NewScreen struktúra CurstomBitMap-ját jelenti, de csak akkor állítsuk be, ha a Cutsombitmap-ot beállítottuk a Flags-nél.

Most. hogy már van screen-ünk tegyünk rá egy ablakot, mely a **Current Window/Open** new window menüpont. Ezzel meg is jelenik a „Your new window" nevű ablak. Lehetőségünk van ablakot is „lopni" a Grab window menüpont kiválasztásával. Ez kiszedi nekünk az ablakot, de még a benne levő gadget-eket is (2.0-és ablak grabbolásakor igen érdekes dolgokat tud csinálni). Az ablakunk paramétereit az Edit window characteristics menüponttal tudjuk editálni. Első ránézésre igen nagy káosz uralkodik a megjelenő ablakban, a másodikra meg méginkább. Itt tudjuk beállítani, hogy az Intuition milyen történéseket figyeljen és továbbítsa azt az ablakunk message portjára (vagyis az IDCMP-ket itt tudjuk beállítani). Ezek leíása néhány fejezettel

előbb már megtörtént és a GadToolsBox program leíásánál is megvannak, do, ott több, így most ezeket nem írjuk le. Ugyanígy vagyunk a Winclow Fiags-al is, ezekkel az ablakunk kinézetét tudjuk beállítani (legyen-e size gadget. close gadget, átb.) Ebben a részben tudjuk az ablak nevét is átállítani, ez aTitle.

Az ID egy azonosító, a type pedig a típusa: Window vagy Requester. Itt tudjuk számszerűen beállítani az ablakunk helyét és nagyságát ül. a minimum és maximum méreteit is. Már csak a Pen-s van hátra, az ablakon belüli rajzoló színek.

A Current Window menüben még néhány pont, mellyel ablakot tudunk törölni, szöveget tudunk kiírni és ezeket törölni.

Most már csak a lényeg van hátra a gadget-ek. Ezeket a Gadgets menü Add gadget menüpontjával tudjuk lerakni. A lerakott gadget-eket a Define Gadget-el tudjuk editálni. Alapvetően háromféle gadget típust tudunk beállítani. Az első a Boolean. Proportional és String.

A Boolean csak a nyomkodható, a proportional a húzogatható és a string az amelybe szöveget írhatunk. A proportional gadget-nél beállíthatjuk, hogy az vízszintes hagy függőleges legyen-e (Freehoriz/ freevert). hány részre legyen osztva (Hbody/Bbody) és azt. hogy legyen-e kerete (Propborderless). A Slring gadget-nél beállíthatjuk a tárolandó szöveg hosszát (Length). azt. hogy legyen-e Undobuffer, a szöveg középre (StringCenler) vagy jobbra (StringRightl legye-e igazítva (alapesetben balra van). A Longint csak számokat fogad el.

A gadgel pozícióját és méreteit a Gadget Position és Gadget Size-nál tudjuk beállítani pontosan. A bal alsó sarokban levő gadget-ekkel pedig a gadget Flag-eket tudjuk beállítani, ezekről már írtunk. A jobb oldalon a forrásba kerülő azonosítók neveit tudjuk beállítani.

A Gadgets menüben még az alábbiak találhatók:

Move a gadget - gadget-et tudunk mozgatni
Clone a gadget - egy gadget-ről tudunk másolatot készíteni, és azt máshol lerakni
Re-slze a gadget - átméretezhetünk egy gadget-et
Work on gadget borders - egy új menüben a kijelölt gadgetnek tudunk új border-t készíteni
Work on gadget text - szöveget tudunk egy gadget-hez hozzárendelni
Work on gadget image - IFF képet tudunk egy gadget hátterébe belenni
Move gadget to start of list - a gadget listában az első helyre tehetjük
Specify gadget successor - két gadget sorrendjét határozhatjuk meg
Move gadget to end of list - a gadget listában az utolsó helyre tehetjük
Delete a gadget - egy gadget törlése

A propgramunkhoz még menükéi is tudunk gyártani a Menus/Append a menü menüponttal. Ezzel csak menüt tudunk létrehozni, a Work on rnenuitems for ... kiválasztásával tudunk menüpontokat kreálni (kész vagyok már ettől a rengeteg menütől!). Ekkor egy új menüsor jelenik meg. melyből az Append a menüitem a legfontosabb. Az Item Text jelenti a nevet, ez kerül be a menü listába. ül. a Coinmand Sequence Key is fontos, mert ez jelenti art a billentyűt, amely a menüi tem mellé log kerülni. ÉkKor a R00lU foi Aiihcy-i nem art bekapcsolni, mert így a Command Sequence Key-nek is hagy helyet. A Room for Check pedig akkor kell, ha kipipálhatónak állítjuk be a menüpontot. Továbbá be lehet még állítani.

hogy a menüpont milyen DrawMode-al legyen kiírva, milyen betűkkel, kiválasztáskor inverzbe váltson (Highromp) vagy egy keret jelenjen meg (HighBox) vagy semmi (Highnone). Azt, hogy kipipálható legyen a Checkit-el tudjuk beállítani. Az egyes menuitem-ekhez image-et is rendelhetünk a Work on images for... menüponttal. Itt külön a kiválasztott állapothoz és a nem kiválasztott állapothoz is rendelhetünk rajzot. A többi menüpont most már magáért beszél.

Most már csak a **Preferences** menü maradt, melyben az alábbiak vannak:

OK prompts - ha be van kapcsolva, akkor minden esemény előtt rákérdez.
FW backdrop - ha be van kapcsolva, akkor a PowerWindows ablaka backdrop típusú, nem tudjuk előtérbe tenni és mozgatni sem.
Autó Redraw- automatikus újrarajzolás.
Autó Select - automatikus kiválasztás, ha be van kapcsolva, akkor egy gadget kiválasztása után a PowerWindows ablaka azonnal aktív lesz
Mouse coordinates - az egér koordinátáinak megjelenítése.
Gadget Collision Check - két gadget ütközésének vizsgálata, ha be van kapcsolva, akkor nem engedi, hogy egy gadget belelógjon a másikba.
Image color remapping - egy betöltött kép színeit az aktuális színekhez állítsa-e.
Image compression - a képeket lömörítse-e (nem a forrásban).
Trim extra bitplanes - ha egy image több színű mint a mi képernyőnk. akkor megpróbálja lekonvertálni (ez nem szokott sikerülni...).
Source code comment - a kimentett forrást ellássa-e comment-ekkel.
Source code spacing - a forrásban tab-ot vagy space-ket használjon.

Végül az első menü. a **Project:**

Generate Source - forráskód készítése assembly, Lattice C vagy Manx C-hez.

Az alábbiak generálását állíthatjuk be:
Screen - a NewScreen képernyő struktúrája
Window - a NewWindow ablak struktúrája
Menü - menü-k struktúrája
Render - az Image-k
Palette - a beállított színek
Gadgets - a gadget-et siniktúrWindowvText - az ablakokhoz lí-ndelt szöveget *
Evén Is - az IDCMP kezelő

Ha valamelyiket nem mentené akkor az össze többit kapcsoljuk ki és próbáljuk meg újra. vagy éppen olyat most nem is tud menteni (pl.: WI3ENCHSCREEN-hez screen-t). Meg kell adni a kimentendő file nevét, ez az Output.

Savé screen and/or windows - adatok kimentése

Load screen and/or windows - adatok betöltése

Kill everything- az összes adat kitörlése a memóriából, új lappal indulunk

Quit - azt hiszem mindenki tudja....

3.2 GadToolsBox (37.300)

A GadTools Box hasonló a PowerWindows programhoz, de sokkal több funkciója van és persze már ismeri a 2.0-ás Kickstart-hoz tartozó gadget-eket.
A program már nem egy mai gyerek, de még mindig az egyik legjobb.
Tehát a programmal alapvetően szép kinézetű képernyőket tudunk szerkeszteni melyekhez C, Oberon és Assembly forrásokat tud generálni. A generált forrásokba pedig be tudjuk rakni a saját programunkat. Olyan mint a manapság elterjedőben levő Visual fejlesztői rendszerek, de annál azért több programozási alapot feltételez, mert a végén úgyis C-ben vagy más nyelven fogjuk a programunkat befejezni. Sokat segíthet a beállítások megértéséhez. ha a hozzá tartozó fejezetet át tanulmányozzuk.

Néhány fontos dolog, amire szükségünk lesz:

-a Shift lenyomása mellett több gadget-et tudunk kijelölni.
-egy gadget-en a double click a Gadgets menü Edit funkcióját jelenti.
-egy click-el megfoghatjuk a gadget-et és mozgathatjuk.
-F8 előhozza a Image Bank-ot.
-F10 redraw funkció, vagyis frissíti a képernyőt.
-egy gadget-et a szélén levő nyolc kis valamivel lehet átméretezni.
-F7 ugyanaz mint a Project/Open.

Ennyi kis kitérő után lássuk, hogy mik vannak a menükben, a nagyobb részeket pedig a menük után mutatjuk be.

A Project menü:

New - teljesen előröl kezdhetjük a munkát.
Open - egy előzőleg elmentett project-et tölthetünk be.
Savé - kimentés.
Savé as - kimentés új névvel.
Generate Source.
C - C nyelvű forráskód generálása.
Assembler – assembly nyelvű forráskód generálása.

Oberon - Oberon nyelvű forráskód generálása.

Preferences

Main - a program általános funkcióit tudjuk itt beállítani.
C Source - a C nyelvű forráskód generálásába tudunk itt beleszólni.
Asm Source - mint az előbb, csak az assembly forráskódhoz.

Close Workbench - a Workbench képernyő bezárása (több memória...)

About - a programról némi info.
Quit - ez nagyon rémes... nem is tudom, hogy mondjam el...

Ez lett volna az első menü, ebből csak a preferences menüpontot érdemes kivesézni, a többi magáért beszél.

Preferences/Main

User Name - felhasználó neve, a kimentett forrás fejlécébe írja bele.
Icon Path - az ikonok megtalálási helye.
Crunch - a kimentett .gui file-ben levő adatokat tömörítse-e.
Password - kódot tehetünk a .gui file-ra.
Buffer - a tömörítéshez használt buffer mérete.
Type - a tömörítés típusa, mennyit tömörítsen a file-on.
Coordinates - az egér koordinátáit megjelenítse-e a ablak fejlécében.
Write Icon - icon-t ment a file-okhoz.
GZZ **adjust** - GimmeZeroZero-t használjon-e.
Overwrite - létező file-t felülírja vagy rákérdezzen.
ASL filerequest - Asl library-t használjon a file ablakokhoz.
Font adaptive - relatív font méretet engedélyez.
Close Workbench - indításkor bezárja-e a Workbench képernyőt.

Preferences/C source

Static - a változókat static-nak definiálja
Generate OpenFont - generáljon-e olyan részt, mely megnyitja a DiskFont-ot.Ha használunk olyan font-ot amely a Fonts: könyvtárban van akkor érdemes ezt beállítani.
Include Pragmas - beszerkeszti a pragma-kat is.
Aztec C - az Aztec C szintaktikait követi.
Generate IDCMP handler - a beállított IDCMP-khez megfelelő class szerinti kódot generál. Tehát külön rész lesz minden IDCMP-hez. például az ablak becsukáshoz (IDCMP_CLOSEWINDOW). meg a többihez, nekünk már tényleg csak a saját kódunkat kell hozzá megírni. Érdemes bekapcsolni, főleg a kezdő programozóknak.

A Gadgets menü

Kind - a lerakandó Gadget típusát tudjuk beállítani.
Gadgets - ha bekapcsoljuk, akkor nem rak le újabb gadget-et. ki tudjuk próbálni a már lerakottakat.
Copy - a kijelölt gadget-et vagy gadget-eket tudjuk másolni, ugyanolyat tudunk máshova lerakni.
Delete - a kijelölt gadget-ek vagy gadget-eket törli.
Edit - a gadget beállításait hívja elő. ez a double click is.
Align - a kijelölt gadget vagy gadget-ek egy utólag meghatározott gadget-hez igazítása.
Clone - a kijelölt gadgel vagy gadget-ek egy utólag meghatározott gadget méreteibez állítása.

Spacing - több kijelölt gadget között egységes helyköz megadása.
Spread - két meghatározott szél közé elhelyezi a kijelölt gadget-eket. például az ablak két széle közé x db gadget-et egyenletes helyközökkel.
Transform - MX gadgel-böl Cycle gadget-et csinál és fordítva. Az eredmény lehet, hogy túl kicsi lesz. nem kell megijedni, hogy mi az a kis piszok ott?!
Edit TabCycle Order - az olyan gadget-eknél (1NTEGER/STRING). melyeknél vanTabCycle funkció, beállíthatjuk ezek sorrendjét
Load - kimentett gadget-eket tölt be, az összes előzőt letörli.
Savé Selected - csak a kijelölt gadget-et menti ki.
Savé - az összes gadget-et kimenti.

A Window menü

New -jelenlegi ablakunkat elteszi és újat hoz létre (a jelenlegi nem veszik el!)
Delete - ha több ablak van akkor lehet törölni valamelyiket
Other - ha több ablak van akkor ezzel lehel kiválasztani az aktívat

Edit Data

Project Name - a forráson belül ezen a néven fog az ablak és annak dolgaira hivatkozni lll. az Other-ben is Start ID from - a start azonosítót ettől kezdi Min/Max/X/Y - az ablak minimális es maximális nagyságának értékei, csak akkor, ha van SizeGadget
Print Info - nemt'om, mert nincs nyomtatóm...
Load - kimentett ablakot tudunk betölteni, az Other listába teszi be, tehát a jelenlegi ablakunkat nem törli
Savé - ablakot lehet kimenteni
Edit Flags - az ablak beállításai, itt már nem részletezzük.
SIZEGADGET - a jobb alsó sarokban az átméretező gadgel.
DEPTHGADGET- a jobb felső sarokban levő háttérbe/előtérbe helyező gadget.
SIZEDR1GHT- a size gadget-el a jobb keretben helyezi el
SMART_REFRESH - az Intuition kezeli az összes ablakfrissítést.
SUPER_BITMAP - saját BitMap-unk lesz.
BACKDROP - háttér ablak lesz.
GIMMEZEROZERO - a/, ablak kerete és a gadget-ek egy extra layer-en lesznek.
ACTIVATE - aktív lesz az ablak mikor megnyitjuk.
DRAGBAR - mozgatható lesz az ablak.
CLOSEGADGET - a bal felső ablakban lesz a becsukó gadget.
SIZEBBOTTOM - a size gadget-et az alsó keretben helyezi el.
S1MPLE_REFRESH - a mi programunk végzi a teljes ablakfrissítést.
OTHER_REFRESH - más egyéb frissítés használata.
REPORTMOUSE - az egér x.y koordinátáit átadja az ablaknak, ezt az IDCMP_MOUSEMOVE-val együtt használjuk.
RMBTRAP - a jobb egérgomb lenyomásának vizsgálata, az IDCMPJVIOUSEBUTrONS a MENUUP és MENUDOVVN-al együtt.

Edit IDCMP - mely IDCMP-k legyenek beállítva, ezt sem részletezzük (lásd 1.6.1 rész.)

SIZEVERIFY - megváltozik az ablak mérete.
MOUSEMOVE - az egér megmozdult.
GADGETUP - valamelyik gadget fel lett endedve.
REQCLEAR - Requester törölve lett.
MENUPICK - valamelyik menüpont ki lett választva.
CLOSEWINDOW - becsukta az user az ablakot.
NEWPREFS - ha a SetPrefs()-el megváltozik a rendszer beállítás.
DISKREMOVEÜ - lemez lett kivéve valamelyik meghajtóból.
INACTIVEWINDOW - inaktív lett az ablak.
VANILLAKEY- billentyűlenyomás -> a jelenlegi character map szerint.
IDCMPUPDATE - a Boopsi Custom gadget-ekliez IDCMP-kiégészítés.
CHANGEWINDOW - az ablak mérete vagy a helye megváltozott.
NEWSIZE - az user átméretezte az ablakot (tehát be is fejezet!)
MOUSEBUTrONS - egérgomb lenyomás vagy felengedés történt.
GADGETDOWN - valamelyik gadget le lett nyomva.
REQSET - Requester lett nyitva az ablakban.
REQVERIFY - Requester lesz megnyitva.
MENUVERIFY- sper menü kezelés ellenőrzése.
RAWKEY - billentyűlenyomás -> saját kód szerint, alt és az F-ek is.
DISKINSERTED - lemez lett berakva valamelyik meghajtóba.
ACTIVEWINDOW - aktív lett az ablak.
DELTAMOVE - az elmozdulás relatív legyen a MOUSEMOVE-ban.
INTUITICKS - másodpercenként tízszer bekövetkezik
MENUHELP- Help billentyű menü közben.

Edit Tags - az ablak néhány fontos paraméterét itt tudjuk beállítani

InnerWidth - ha be van kapcsolva akkor WA_Width helyett WAJnner Width tag-ot használ a forrás generálásakor.
innerHeight - mint az előző.
MouseQueue - WAJVIouseQueue-t is beteszi az ablak tag-listájába és annak értékét is beállíthatjuk.
RtpQueue — WA_RplQueue-t is beteszi az ablak tag-listájába.
WindowTitle - itt állíthatjuk be. hogy mi legyen az ablakunk neve.
Screen Title - itt állíthatjuk be. hogy mi legyen a screen fejlécében ha aktív az ablakunk.
AutoAdjust - a WA_AutoAdjust tag-ot beteszi az ablak tag-listájába és azt TRUE-ra állítja.
FallBack - a WA_PubScreenFallBack tag-ot beteszi az ablak tag-listájába és azt TRUE-ra állítja.

Edit Grid- rács beállítása a gadget-ekhez (nagyon hasznos funkció).

Gríd X size - a rács X mérete.
Grid Y size - a rács Y mérete.
Grid On - a rács bekapcsolása.

Edit Offset - keret offset beállítása - egy gadget max ilyen közel vihető a kerethez.

Horiz offset - horizontális érték.
Vert offset - vertikális érték.
Offsets on - bekapcsolása.

Texts - szövegek elhelyezése.

Add - megadott szöveg elhelyezése a képernyőn.
Edit - a kiválasztott szöveg paramétereit tudjuk újra beállítani.
Delete - a kiválasztott szöveget törli.
Move - a kiválasztott szöveget mozgatni tudjuk.
Load - szövegeket tölthetünk be, az meglevők után is lehel fűzni.
Savé - szövegeket menthetünk ki.

BevelBoxes - keretek elhelyezése.

Add - keret elhelyezése.
Move - keret mozgatása.
Size - keret átméretezése.
Delete - keret törlése.
Flip Recessed - mintha benyomnánk/kihúznánk.
Flip Dropbox - dupla keretesre változik/vissza.
Load - keretek betöltése.
Savé - keretek kimentése.

A Seréén menü

Palette - színpaletta beállítása, ha a sereen-ünk CUSTOMSCREEN.
Select Font - beállíthatjuk, hogy milyen karakterkészletet akarunk használni.
Edit DriPens - ha a sereen-ünk CUSTOMSCREEN akkor beállíthatjuk. hogy az egyes rajzolási funkciókhoz melyik paletta elemet rendeljük.
Edit Tags - beállíthatjuk a sereen-ünk típusát és a Title-t. A Titie is csak akkor számít, ha CUSTOMSCREEN típus.t állítottunk be. Ez az SA_Title lesz.
Change Type - a sereen üzemmódját tudjuk beállítani, ennek is csak a CUSTOMSCREEN módban van értelme.

A Menüs menü

Edit - itt tudjuk megszerkeszteni a menüinket.
Test - le is tudjuk tesztelni, hogy hogyan működnek, kilépés ESC.
Loaü — Iliciiú Ijetíjltéoc.
Savé — menü kimentése.

A Szövegekről

Igen szép ablakot tudunk csinálni, ha a szövegeknek árnyéka van. Ehhez egy fekete szövegre egy JAM1-es módú fehér szövegei tegyünk rá és kész is van az immár sokkal szebb szöveg.

A JAM2-es módot akkor érdemes használni, ha egy szöveget később programunk- ból meg akarunk változtatni. Ekkor a régi szöveget le kellene törölni és csak utána írhatnánk ki az új szöveget. Ez a mód viszont elintézi ezt nekünk, csak a szöveget kell megváltoztatni és csak a hosszra kell vigyázni.

A Gadget típusokról

Általában a gadget-eknél az alábbi „valamiket" kell megadnunk (zárójelben a jellemző gadget neve található):

Label - a forrásban ilyen azonosítóval fog rá hivatkozni.
Text - magyarázó szövegei helyezhetünk el.
Place - a Text hol legyeir

In - belől
Left - bal oldalt
Kight -jobb oldalt
Above - felette
Below - alatta

Underscore - ha egy betű elé az aláhúzás jelet rakunk, akkor azt alá- hu/Ai pl „Gadget - ekkor a G betűt húzza alá.
HighLabel - a Text a DrlPens-nél HIGHLIGHTTEXTPEN által meghatá- Mi/oll színnel lesz kiírva.
Disabled - a gadget nem engedélyezett.
Checked - már ki legyen pipálva (CHECKBOX).
TabCycle - az ilyen gadget-ek között választhatunk a tabulátor megnyomásával (INTEGER).
Piacement - a benne levő számot hova igazítsa.
Center - középre
Left - balra
Rifiht -jobbra
MaxChars - max ennyi karaktert fog a gadget elfogadni.
Number - kezdőérték (INTEGER).
Spacing - a sorok közolt ennyi pixelt hagy ki (LISTVIEW).
Read Only - nem lehel kiválasztani, csak szöveg megjelenítésre való (LISTVIEW).
ShowSelected - kiválasztáskor nem csak megvillan, hanem úgy is marad (LISTVIEW)
Ser. Width - scroller szélessége (LISTVIEW).
Active - alapban melyik legyen aktív (MX/CYCLE).
Bordér - legyen-e kerete (NUMBER).
Default - alapérték (NUMBER).

Depth - színek száma 1-2 szín. 2-4 szín. 3-8 szín, 4-16 szín, stb... (PALETTE)
Color - aktív szín (PALETTE).
Freedom - horizontális vagy vertikális legyen (SCROLLER/SLIDER).
Top - kezdeti érték (SCROLLER).
Totál - ennyi részre van felosztva (SCROLLER).
Visible - a huzigálható bizgentyü mérete (SCROLLER).
Arrows - nyilak legyenek-e és megadható a mérete is (SCROLLE'.R).
RelVerify - IDCMP_GADGETUP küldése (ha nem is húztuk el a gadgetet) (SCROLLER/SLIDER).
Immediate - IDCMP_GADGETDOWN küldése(SCROLLER/SLIDER).
Min - minimális érték (SLIDER).
Max - maximális érték (SLIDER).
Levél - kezdőérték, lehetőleg a min és max közé essen... (SLIDER) .
Length - a Format-ban megjelenítendő max karakterek száma (SLIDER).
Formát - standard C formátumban megadhatjuk hogy a slider értékét hogyan írja ki, pl "Az értéke:%ld" (SLIDER).
Position - a Formát hova kerüljön, mint a Place (SLIDER).
MaxChars - a max elfogadható karakterek száma (STRING).
String - a default szöveg (STRING).
Exithelp - a help billentyű lenyomására külön IDCMP_GADGETLJP. kódot küld. így hozzárendelhetünk help szöveget (INTEGER/STRING)

A menü szerkesztésről

Az első oszlopban a menük nevét adhatjuk meg, a másodikban a menüben szereplő szövegeket, a harmadikban pedig az almenüket.
A második és harmadik oszlopba be lehet tenni elválasztó vonalat a Barlabel gadget segítségével.

Az alábbi check gadget-eket találjuk még:

Disabled - a menü/menüpont nem kiválasztható, tiltott.
Menutoggle - a menü ki/be kapcsolható legyen, (mint egy lámpa kapcsoló).

Checkit - ki lehet pipálni.
Checked - alapból ki van választva.

Többet erről nem érdemes szólni, mindenki hamar rá fog jönni a technikájára.
Ennyi lett volna a GadToolsBox leírása, mindenkinek ajánljuk a használatát, nagyon jó és könnyen használható.

4. Függelék

4.1 RAW kód táblázat

Ebben a részben csak a billentyűzet RAW kódjait szeretnénk megadni, hogy programozás közben ne kelljen más egyéb utat keresgetni ezen számok ismeretéhez. Mint tudjuk a RAW kódok közvetlenül a billentyűzetet meghajtó mikro kontroller által küldött kódok. így ezek a kódok nem mennek keresztül semmiféle procedúrán. A leütött gombról akkor is érkezik jel amikor lenyomott és akkor is amikor felengedett. A leírást nehezíti az a tény. miszerint minden gombhoz szinte más.funkcióI rendelhetünk. így egy adott billentyű meghatározására nehéz támpontot találnunk. Mi a német billentyűzetet használjuk támpontként a leíráshoz, mivel véleményünk szerint ez van több " (annak ellenére, hogy mindenki utalja) és ezen a bili. több gomb is található mint az angol billentyűzeten. Figyelem! A két bili. alapvetően az Y és a Z betű felcserélésében különbözik! Tehát a számok mellel található betű vagy egyéb ismertető jegy a német billentyűzetre vonatkozóan azon az ismertető jeggyel ellátott billentyűt jelöli (például az ö bili. az alulról a harmadik sorban a negyedik billentyűt jelenti, az Enter természetesen attól, hogy két sorban van. bár egy bili. mindkét sorba beleszámított ennek kiszámításába). Nos. tehát a táblázat, amelyben minden billentyűhöz két szám tartozik. Részletesen a RAW kódról az 1.6.1 rés/ében olvashatunk a könyvnek.

Lenyomott		Felengedett		Bili.
69	$45	197	$C5	Esc
80	$50	208	$DO	FI
81	$51	209	$D1	F2
82	$52	210	$D2	F3
83	$53	211	$D3	F4
84	$54	212	$D4	F5
85	$55	213	$D5	F6
86	$56	214	$D6	F7
87	$57	215	$D7	F8
88	$58	216	$D8	F9
89	$59	217	$D9	F10
0	$00	128	$80	`
1	$01	129	$81	1
2	$02	130	$82	2
3	$03	131	$83	3
4	$04	132	$84	4

5	$05	133	$85	5
6	$06	134	$86	6
7	$07	135	$87	7
8	$08	136	$88	8
9	$09	137	$89	9
10	$0A	138	$8A	0
11	$0B	139	$8B	fi
12	$0C	140	$8C	'
13	$0D	141	$8D	\
65	$41	193	$C1	Back Space
66	$42	194	$C2	Tabulátor
16	$10	144	$90	9
17	$11	145	$91	W
18	$12	146	$92	E
19	$13	147	$93	R
20	$14	148	$94	T
21	$15	149	$95	Z az angolnál ez Y
22	$16	150	$96	U
23	$17	151	$97	I
24	$18	152	$98	0
25	$19	153	$99	p
26	$1A	154	$9A	ö
27	$1B	155	$9B	+
68	$44	196	$C4	Enter vagy return
99	$63	227	$E3	Ctrl
98	$62	226	$E2	Caps Lock
32	$20	160	$A0	A
33	$21	161	$A1	S
34	$22	162	$A2	D
35	$23	163	$A3	F
36	$24	164	$A4	G
37	$25	165	$A5	H
38	$26	166	$A6	J
39	$27	167	$A7	K
40	$28	168	$A8	L
41	$29	169	$A9	ő
42	$2A	170	$AA	/
43	$2B	171	$AB	#
96	$60	224	$E0	Bal Shift
48	$30	176	$B0	<
49	$31	177	$B1	y
50	$32	178	$B2	X
51	$33	179	$B3	C
52	$34	180	$B4	V
53	$35	181	$B5	B
54	$36	182	$B6	N
55	$37	183	$B7	M
56	$38	184	$B8	.

Raw kód táblázat

• 57	$39	185	$B9	.
58	$3A	186	$BA	-
97	$61	225	$E1	Jobb Shift
100	$64	228	$E4	Bal Alt
102	$66	230	$E6	Bal Amiga
64	$40	192	$C0	Space
103	$67	231	$E7	Jobb Amiga
101	$65	229	$E5	Jobb Alt
70	$46	198	$C6	Del
95	$5F	223	$DF	Help
• 76	$4C	204	$CC	Kurzor Fel
77	$4D	205	$CD	Kurzor Le
78	$4E	206	$CE	Kurzor Jobbra
79	$4F	207	$CF	Kurzor Balra
				Numerikus Billentyűzet.
90	$5A	218	$DA	[(NumL)
91	$5B	219	$DB	] (ScrL)
92	$5C	220	$DC	/ (SysRq)
93	$5D	221	$DD	* (PrtScr)
61	$3D	189	$BD	7 (Home)
62	$3E	190	$BE	8 (Kurz or fel)
63	$3F	191	$BF	9 (PgUp)
74	$4A	202	$CA	-
45	$2D	173	$AD	4 (Kurzor Balra)
46	$2E	174	$AE	5
47	$2F	175	$AF	6 (Kurzor Jobbra)
94	$5E	222	$DE	+
29	$1D	157	$9D	1 (End)
30	$1E	158	$9E	2 (Kurzor Le)
31	$1F	159	$9F	3 (PgnDn)
67	$43	195	$C3	Enter
15	$0F	143	$8F	0 (Ins)
60	$3C	188	$BC	. (Del)

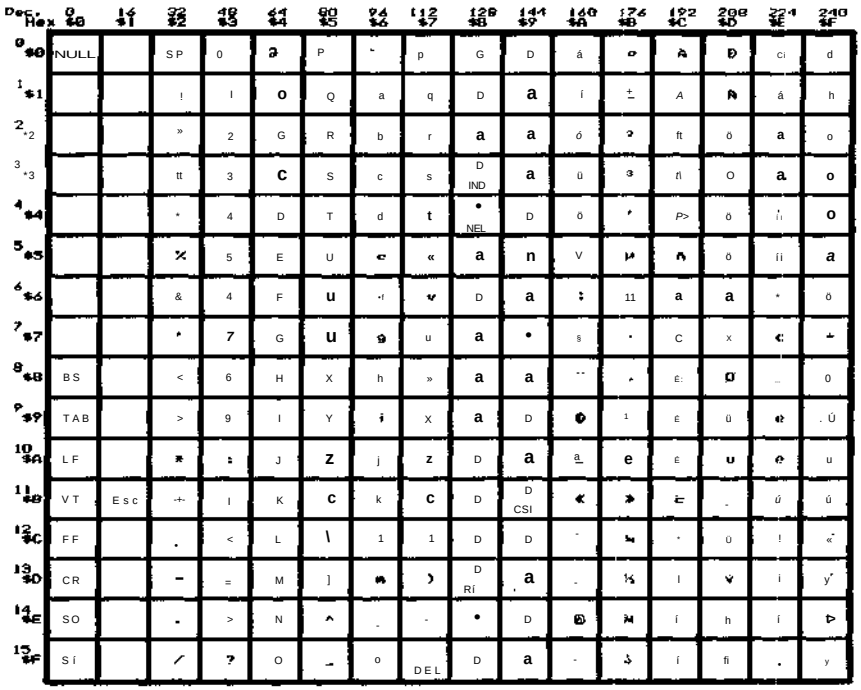
A vezérlő gombokkal való együtt lenyomást a Qualiíier változó tartalmazza, és ennek értékei az alábbiakban változhatnak.

Qualifler		Vez. Billentyű
32796	$8001	Bal Shift
32770	$8002	Jobb Shift
32772	$8004	Caps Lock
32776	$8008	Ctrl
32784	$8010	Bal Alt
32800	$8020	Jobb Alt
32832	$8040	Bal Amiga
32896	$8080	Jobb Amiga
33024	$8100	Num. lásd. szöveg.
33280	$8200	Ismétlés lásd. szöveg.

Valamint ha a Numerikus billentyűzeten megnyomjuk valamelyik billentyűt, akkor a 33024-es kódot kapjuk, és mindaddig ezt kapjuk, ameddig az user csak a numerikus billentyűzetet használja. Ha az user keze visszatéved a normál billetyüzetre ez a kód megszűnik.

A normál billentyűzeten (kivéve a numerikus), azok közül bármelyik lenyomva tartása mellet, ha a billentyűzet elkezdte azt ismetelni, akkor a következő értéket kapjuk: 33280. Természetesen, ha ezt a gombot egyszerre tartjuk lenyomva egy. vagy több vezérlő karakterrel, az értékhez hozzáadódik a vezérlő karakternek megfelelő érték (bitenként vagy művelettel).

4.2 Az Amiga ASCII karakterkészlete



BS: Back Space
VT: Vertikális Tabulátor
SO: Shift Out
NULL: 0 karakter.
LEN: Return Line Feed
TAB: Tabulátor
FF: Form Feed
Sí: Shift In
DEL: Delete
IND: Line Feed
LF: Line Feed
CR: Carriage Return
Esc: Esacape
Rí: Reverse Line Feed

4.3 A DOS hibaüzenetei

103 insufficient free store
Nincs elég memória.

121 file is not an object modulé
A fájl nem futtatható program.

202 object in use
Éppen egy másik program által használt az elérni kívánt file.

203 object already exist
A megadott néven már létezik egy file.

204 directory not found
A könyvtár (directory) nem létezik, ill. nem erérhető.

205 object not found
Nincs meg a keresett file. program.

210 invalid stream component name
Nem érvényes file név.

213 disk not validated
A DOS nem képes a lemezről olvasni (vagy mert hibás a lemez, vagy mert kivettük a lemezt a meghajtóból amikor dolgozott vele).

214 disk wxite-protected
A lemez írásvédett.

218 device not mounted
Az egység nem létező, vagy nincs csatlakoztatva, vagy nem üzemképes, vagy egyszerűen nem mount-ált.

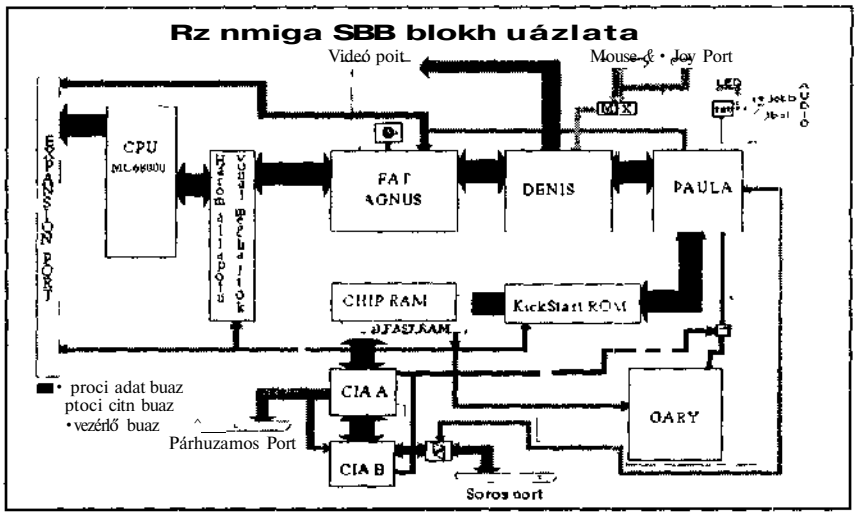
221 disk full
A lemez megtelt.

222 file is protected from deletion
A file védelmi attributtuma nem engedélyezi a törlést.

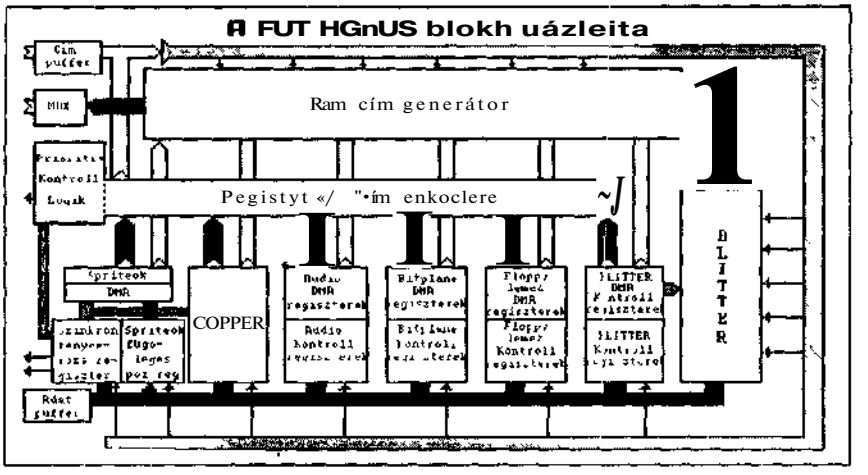
225 not a DOS disk
A DOS számára ismeretlen típusú lemez.

226 no disk in drive
Nincs lemez a meghajtóban.

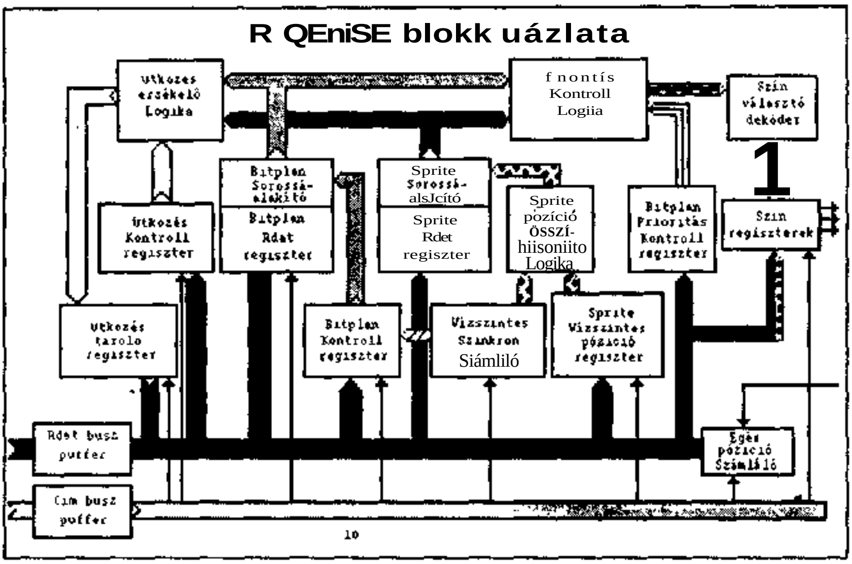
4.4 Ábrák



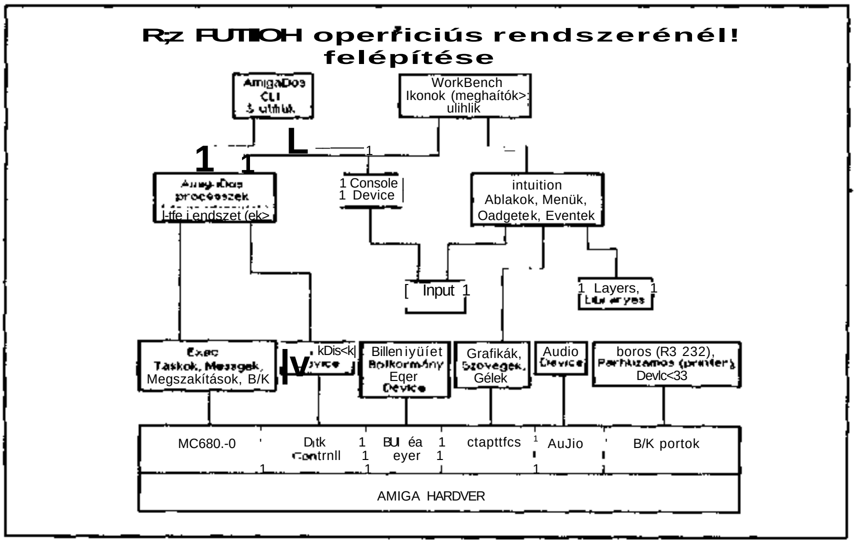
1 ábra



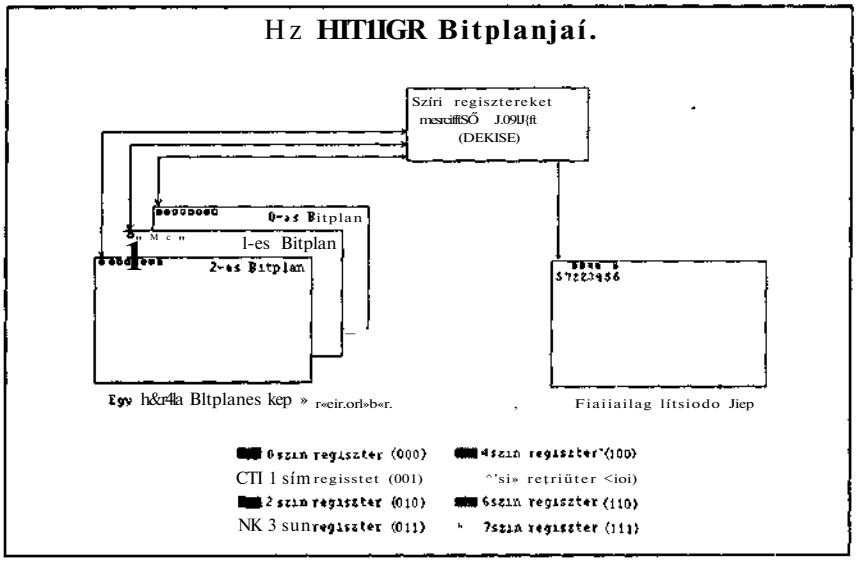
2 ábra


Sprite
Sprite
Kontroll
Sprite
Rdet
regiszter
Sprite
pozíció
Logika
Bitplan
Kontroll
regiszter
Egér
pozíció
Számláló
Rdat busz
puffer
Cím busz
puffer

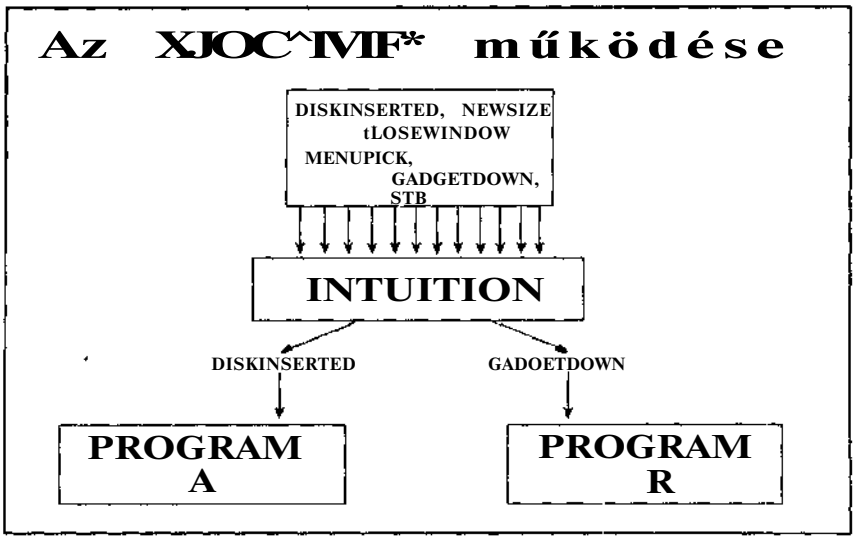
3. ábra


AmigaDos
CLI
WorkBench
Console
Device
intuition
Ablakok, Menük,
Eventek
Input
Layers,
Exec
Megszakítások, B/K
Grafikák,
Gélek
Audio
Device
AMIGA HARDVER

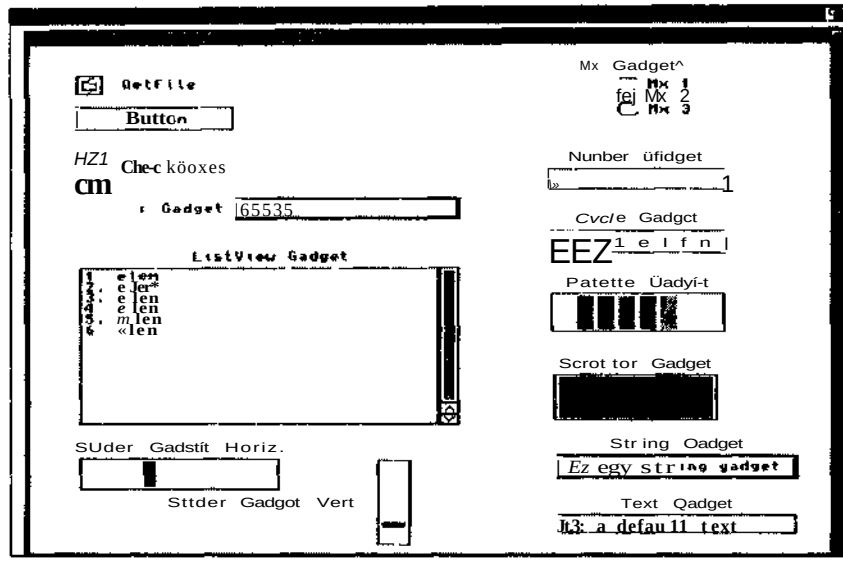
4. ábra


Hz HITIIGR Bitplanjaí.
Szíri regisztereket
(DEKISE)
0-as Bitplan
l-es Bitplan
2-as Bitplan
Egy h&r4la Bltplanes kep »
Fiaiiailag lítsiodo Jiep
0szin regiszter (000)
CTI 1 sím regisztet (001)
2 szin regiszter (010)
NK 3 sun regiszter (011)
4szin regiszter (100)
6szin regiszter (110)
7szin regiszter (111)

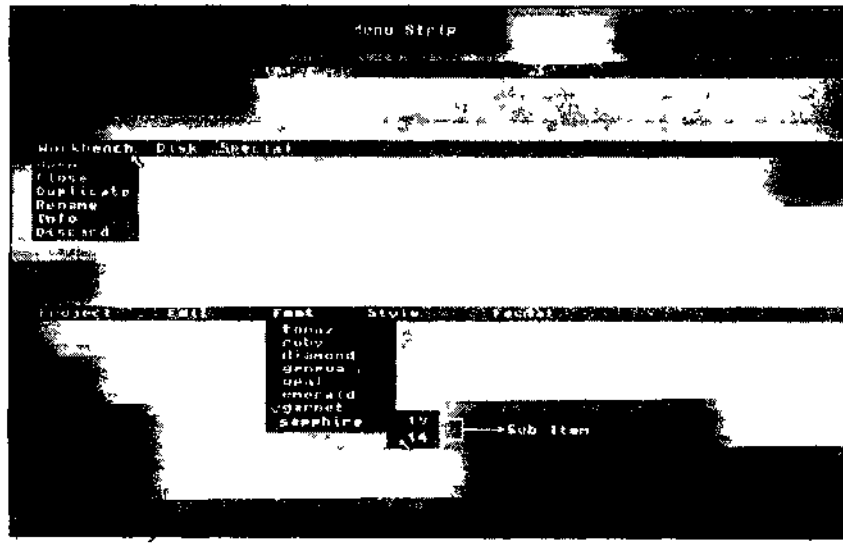
5. ábra


Az XJOC^IVIF* működése
DISKINSERTED, NEWSIZE
tLOSEWINDOW
MENUPICK,
GADGETDOWN,
STB
INTUITION
DISKINSERTED
GADOETDOWN
PROGRAM A
PROGRAM R

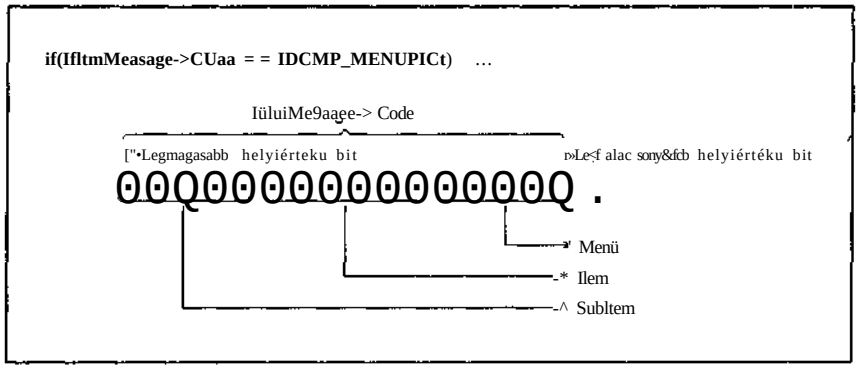
6 ábra


GetFile
Button
HZ1
Che-c köoxes
Gadget
65535
ListView Gadget
1. elen
2. e.Jer*
3. e len
4. e len
5. m len
6. «len
SUder Gadstít Horiz.
Sttder Gadgot Vert
Mx Gadget^
Mx 1
Mx 2
Mx 3
Nunber üfidget
1
Cvcle Gadgct
Patette Üadyí-t
Scrot tor Gadget
Str ing Oadget
Ez egy string gadget
Text Qadget
It3: a defau 11 text

7 ábra


Menu Strip
Workbench Disk Options
Close
Duplicate
Rename
Info
Password
Project Edit Font Style Text
Sub Item

8 ábra


if(IfltmMeasage->CUaa = = IDCMP_MENUPICt) …
IüluiMe9aaee-> Code
["•Legmagasabb helyiérteku bit
p»Le<:f alac sony&fcb helyiértéku bit
00Q0000000000000Q .
Menü
-* Ilem
-^ Subltem

9. ábra

4.5 Tárgymutató

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

u

V

W

5.0 Búcsúzunk

Ez a könyv azzal a céllal íródott, hogy egy jó kézikönyvet készítsünk' amely nagy segítséget nyújthat az Amiga programozásában Azt gondoljuk, hogy látszik a könyv tartalmából, közel sem teljes. Ezért úgy tervezzük, hogy folytatjuk és szeretnénk megjelentetni egy második - vagy több - részt.

Ehhez viszont az kell. hogy sokan megvegyék a könyvet és ne csak LEMÁSOLJÁK a lemezmellékletet. Ezt a könyvet nem anyagi javak szerzése miatt irtuk, hanem azért, hogy az Amiga éljen és túléljen. Tehát ha megfelelően nagy lesz az érdeklődés akkor elkészítjük a második részt is. mert reménykedünk abban, hogy vannak még Amigások...

A következő rész tervezett tartalmából:

- az Amiga sprite, bob 1ll. animációs lehetőségei (rendszer alól is!)
- a Dos library
- a Workbench library
- az IFF A-tól Z-ig
- a rendszer devices
- a resources
- a 68882-es kooprocesszor programozása
- az Amiga hangja és modul lejátszás (renszer alól is!)
- a Custom regiszterek (OSC. ECS. AGA)
- a Dice C bemutatása
- a Designer bemutatása
- demók készítése immár nem rendszer alól
- 3D vektorgrafika
- Blitler
- Copper

A regisztrációs kártyáról

Mint tudjuk, egy könyvben azért van regisztrációs kártya, hogy a szerzők értesüljenek az olvasói kör véleményéről (mert köztudottan nem elterjedt a iró-olvasó találkozó a szakmai könyvek körében), óhajairól és esetleges, megrendeléséről. További előnye, hogy az olvasót közvetlenül tudjuk értesíteni bármilyen új könyv megjelenéséről. Felmerült bennünk az a gondolat is, hogy ha bárkinek bármilyen Amiga programozással kapcsolatos gondja támadna, akkor nyugodtan keressen meg minket, és biztos, hogy mi a legjobb tudásunk szerint próbálunk segítséget nyújtani.