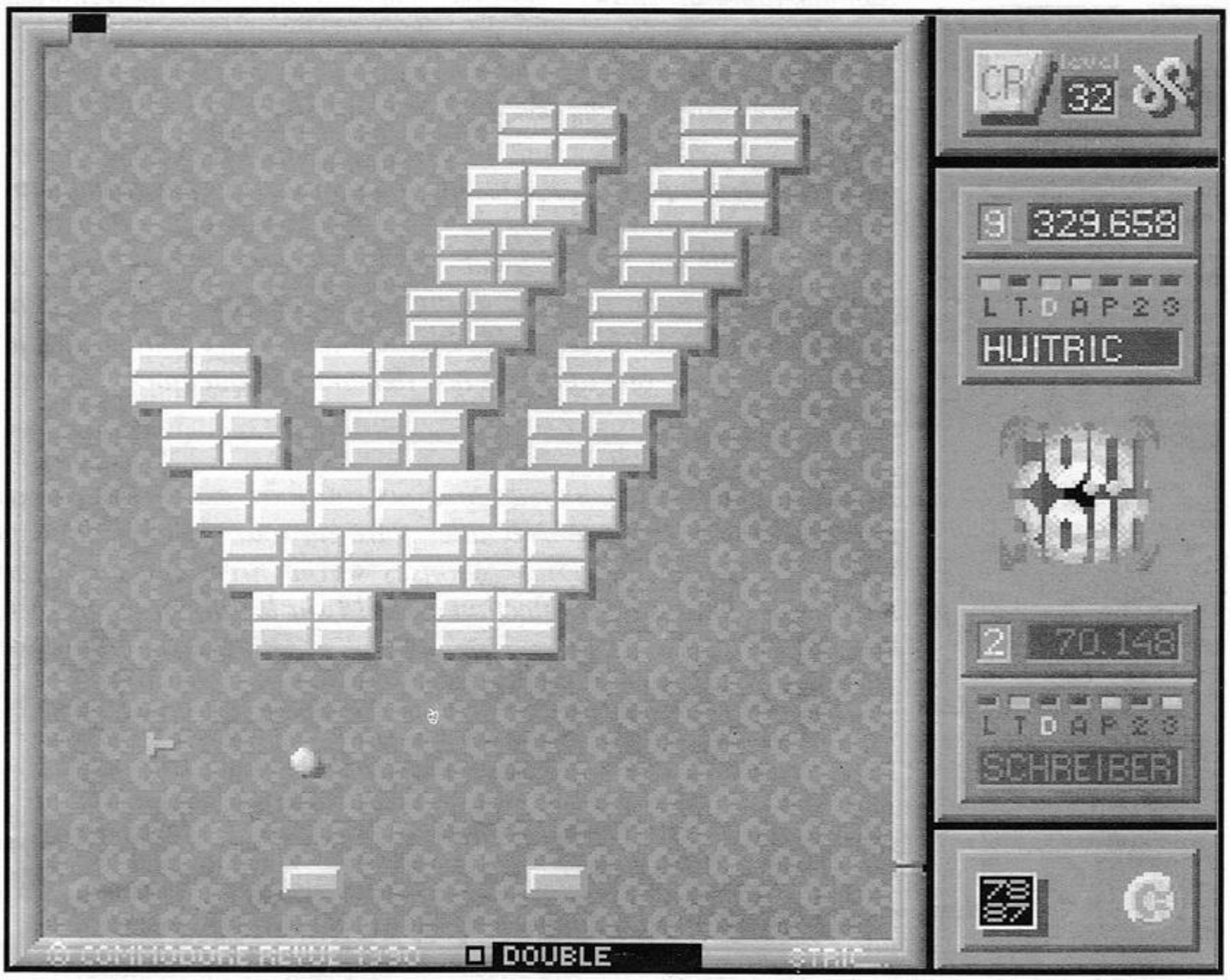
LE MAGAZINE DES PROGRAMMEURS SUR AMIGA - N°20 - MARS 1991

AMIGA NEWS-TECH



Page écran de ComRoïd, premier tableau du logiciel développé sous Amos dans l'Ant ancienne formule. Cette initiation continue dans les colonnes d'*AMIGA* Revue (voir rubrique programmation).

Graphismes Huit
Programmation Septh

Ce numéro 20 de l'Amiga News-Tech a été tiré à 3000 exemplaires (2417 abonnés enregistrés le 10.02.91).
Ce magazine est une publication Commodore Revue Sarl. Téléphone de la rédaction : (1) 42 469 290

OURS

Tiens, je savais bien qu'il y en aurait au moins un pour venir s'abimer les yeux à lire les petits caractères. Alors ça, ça s'appelle l'Ours et ça parle de nous, du journal et des détails qu'on est obligés de donner. Nous, c'est Amiga News Tech (ANT), une publication de Commodore Revue S. A. R. L. L'adresse, c'est le 5, rue de la Fidélité, 75010 Paris. Le directeur de la publication s'appelle Jean-Yves Primas et on peut le joindre au **(1) 42. 47. 12. 16** tous les jours entre 12 h 00 et 12 h 02. La charmante Christine Robert est sa secrétaitre de direction, y'en a vraiment qui s'embêtent pas. La rédaction, maintenant. Un seul numéro : **(1) 42. 46. 92. 90.** Yves Huitric est le rédacteur en chef, courageusement assisté de Stéphane Schreiber rédacteur en chef adjoint.
Stéphane Schreiber, qui sert également de secrétaire de rédaction. Plusieurs fous furieux ont collaboré à ce numéro : **Pascal Amiable, Franck Charlet, François Lionet, Max, Frédéric Mazué, Antoine Mussard, Denis Obriot et Stéphane Schreiber** (encore lui). La maquette est signée Yves Huitric. Pour abonner vos copains, passez donc un coup de fil à Claude au **(1) 43. 78. 08. 21.** Voilà, c'est tout, y'a d'autres pages plus intéressantes à lire.

Ah non, j'allais oublier de dire qu'Amiga est une marque déposée de Commodore Business Machines et qu'Amiga News-Tech est une publication totalement indépendante de la société Commodore France.

Ce numéro 20 d'Amiga News Tech a été réalisé sur un Amiga 2000 (7 Mo de mémoire, 68030, FlickerFixer) avec les logiciels Deluxe Paint III, Excellence! 2.0 et Professional Page 2.0, quelques nuits blanches et beaucoup de patience. Du coup, le 2 et le 0 sont nos chiffres porte-bonheur. Bisous à tous les crapitons du monde.

36.15 ANT LE 16.03.91

EDIT OH !

Voilà, ça y est, vous le tenez entre vos mains : l'Amiga News Tech nouvelle formule, le petit frère d'Amiga Revue devenu grand, a enfin pris son envol solitaire.

Depuis le temps que l'on désirait faire de ce "journal dans le journal" un magazine à part entière... Rendez-vous compte, l'ANT devient le seul journal en langue française entièrement dédié à la programmation sur Amiga. Jusqu'à présent, de tels magazines américains ou anglais existaient, mais difficiles à se procurer de par chez nous. Je pense notamment au Transactor, qui avait donné son nom à l'une de nos rubriques.

Au sommaire de ce numéro 20 (déjà) : rien que du bon. Les rubriques habituelles, bien sûr : c'est ainsi que vous retrouverez le pantagruélique Frédéric Mazué et ses utilitaires fous-fous-fous, le germanique Doktor Von GlutenStimmelImDorf alias Mister Graphics, l'ex-militaire Max, spécialiste ès assembleur et retardataire de première, François "Amos" Lionet et tous les autres.

Mais également d'autres rubriques, nouvelles, qui profitent des 16 pages supplémentaires qui nous ont été généreusement offertes : avec d'abord le concours listing permanent (voir page 16), l'invité du mois, le Forum, ou encore le Coin-coin des DemoMakers (page 12). Les abonnés à la disquette y trouveront les sources et exécutables de tous les programmes publiés dans le journal, ainsi que quelques surprises supplémentaires (démos, domaine public...). Par exemple, la disquette de mars (ANT N°21) contiendra le compilateur C de Mat Dillon complet !

L'ANT nouvelle formule, c'est également un serveur Minitel accessible par le 3615 code ANT dès le 16 mars, sur lequel vous pourrez retrouver tous nos collaborateurs, télécharger, échanger des idées, dialoguer avec des programmeurs célèbres...

L'ambition de ce nouvel ANT est de devenir la référence dans le domaine de la programmation sur Amiga. Malgré des débuts difficiles (saviez-vous qu'une carte Passerelle AT plantait irrémédiablement dans un Amiga 2000 équipé de plus de 5 Mo de RAM ? Nous l'avons appris à nos dépends) et quelques défauts de jeunesse ("nobody's perfect"...).

Et puisque le meilleur moyen de vous satisfaire est encore de savoir avec exactitude ce que vous attendez de l'ANT, allez faire un tour du côté de la page 27 où un superbe sondage vous attend. Trois réponses seront tirées au sort, pour faire gagner à leurs auteurs des heures de connection en 3614 sur le futur serveur ANT.

Stéphane Schreiber

NB : Nous annoncions dans notre dernier numéro d'Amiga Revue, le Guide le l'Amiga 91 en cadeau d'abonnement... Celui-ci étant encore en cours de fabrication, vous ne le recevrez que le mois prochain, avec le numéro 21 de l'ANT.

COMPILATEUR GFA- BASIC 3.5

Depuis le temps qu'on l'attendait, celui-là... Micro-Application vient tout juste de le sortir en France, il ne devrait pas tarder à débarquer chez vos marchands de frites favoris. L'interprêteur 3.5 se fait lui toujours attendre, puis la dernière version officielle est la 3.41.
Micro-Application, 58, rue du Faubourg Poissonnière, 75010 Paris.

LA NOUVELLE BIBLE

La Bible de l'Amiga nouvelle édition de luxe est également disponible, mais pas depuis suffisamment longtemps pour pouvoir vous en parler plus longuement dans ce numéro de l'ANT. En tout cas, on m'a assuré chez Micro-Application qu'elle avait été ré-écrite pour éliminer tous les bugs de la précédente édition (voilà qui devrait faire taire Frédéric Mazué). Des chapîtres supplémentaires sur le système ont également été ajoutés. On vous en dira plus le mois prochain.

ADAPTEZ-VOUS !

Plus qu'un nouvel assembleur pour Amiga, Adapt est un ensemble de d'utilitaires pour les programmeurs en langage-machine, incluant un macro-assembleur 68040 (!), un linker simple passe (les deux possédant un port ARexx), un analyseur de modules et un profiler, le tout pour moins de 50 dollars. Car en effet, il s'agit d'un produit américain, que nous allons nous dépêcher de tester pour vous.
Lake Forest Logic, Inc., 28101 Ballard Rd. Unit E, Lake Forest, Il. 60045, USA.

LE RETOUR DU PROJET D

Ben Fuller, le bidouilleur fou docteur ès-pistes et secteurs, frappe encore en annonçant la version 2.0 de son fameux Projetc D. Pour ceux qui ne connaîtraient pas cet utilitaire, il se présente d'un programme maître chargeant plusieurs modules utilitaires baptisés "Tools". La version 2.0 comprend le BackupTool, un copieur soft de grande qualité, l'OmniTool, capable de lire et écrire des disquettes au format MS-DOS ou TOS, l'EditorTool, un éditeur de secteurs, ainsi que le CatalogTool, un éditeur et optimiseur de répertoires. La version 2.0 est entièrement compatible avec le Workbench 2.0 des Amiga 3000.
Fuller Computer Systems, Inc. Tel : (19)-1-800-874-3475.

L'EXPO

La troisième édition d'AmigaWorld Expo (ex-AmiEXPO) se tiendra du 15 au 17 mars prochains à l'hôtel New York Hilton, à New York, donc. En plus du salon lui-même, plusieurs conférences de programmeurs sont prévues sur les thèmes les plus variés, comme par exemple la vidéo, le multimédia, la musique ou le Workbench 2.0. Si vous avez les moyens et le temps d'y faire un tour, contactez AmigaWord Expo au (19)-1-800-322-6442 ou écrivez leur à l'adresse suivante : AmigaWorld Expo, 465 Columbus Avenue, Ste. 285, Valhalla, NY 10595.

FAUX JUMEAUX

Le magazine américain AmigaWorld lance à peu près en même temps que nous l'ANT, l'AmigaWorld Tech Journal, un magazine en langue anglaise, dédié aux professionnels de la programmation. La parution est bimestrielle et l'abonnement magazine+disquette coûte 99,95 dollars pour l'Europe.
The AmigaWorld Tech Journal, P. O. Box 802, 80

NE RATEZ PAS LES DISQUETTES DE FICHIERS "ARP" DU CAHIER PRATIQUE D'AMIGA REVUE

LANGAGE : AMT : AMOS Technique

Nouveau journal, nouvelle rubrique AMOS, plus technique et plus axée sur le fonctionnement de l'interpréteur et ses astuces. Nous allons voir aujourd'hui le principe de programmation d'une extension au basic....

AMOS est un langage extensible : on peut facilement ajouter de nouvelles instructions aux 600 déjà présentes. Il suffit de respecter quelques règles simples dans la construction du programme.
Pour assembler une extension, vous devez récupérer le fichier "EQU.S" se trouvant sur la disquette "Extras", dans le dossier "Extensions". Ce fichier contient la définition de toutes les variables système de l'AMOS. Vous devez l'inclure dans votre programme.

Bien que ce ne soit pas indispensable, je vous conseille d'écrire du code relogeable, pour deux raisons :

- vous n'aurez pas à modifier les routines pour le compilateur AMOS qui, lui, n'acceptera QUE du relatif pc ;
- vous ressentirez l'immense satisfaction de produire un tel code!

CHARGEMENT DE L'EXTENSION

Au démarrage, AMOS explore la liste des extensions contenue dans le fichier AMOS1_2.Env, et charge tous les fichiers cités à l'aide de la fonction AmigaDOS LoadSeg(). Une extension est donc un programme normal, pouvant contenir plusieurs hunks. Immédiatement après le chargement, AMOS effectue un JSR à la première adresse du premier hunk. Une extension doit débuter impérativement par la routine d'initialisation à froid (ou "Cold Start").

AMOS transmet plusieurs paramètres à l'extension :

- A0 pointe sur la table d'adresse des routines AMOS ;
- A1 contient l'adresse de base de l'intuition.library ;
- A2 contient l'adresse de base de la graphics.library ;
- A3 contient l'adresse de base de la diskfont.library ;
- D0 contient l'adresse du rastport de l'écran courant ;
- A5 pointe sur la zone de datas du Basic ;
- A6 pointe sur la liste des tables d'instructions de toutes les extensions ;

De son côté, l'extension doit impérativement retourner plusieurs adresses

- dans A0, la table des instructions de l'extension ;
- dans A1, le message de bienvenue ;
- dans A2, l'adresse de la routine "Warm Start" ;
- dans A3, l'adresse de la routine "Quit" ;
- dans D0, 0 si tout va bien, -1 si une erreur quelconque est survenue (provoque le retour au CLI) ;
- dans D1, le numéro de l'extension dans la liste ;
- dans D2, 0 en temps normal, ou l'adresse de la routine "Bank Check";

L'extension peut réserver de la mémoire, charger des fichiers, procéder à des affichages, faire du bruit, etc... Veillez cependant à ne pas trop ralentir le chargement de l'AMOS. L'extension rend la main à l'AMOS par un simple RTS.

INITIALISATION A CHAUD (WARM START)

Cette routine est appelée au cours de l'initialisation de l'écran, lors d'un RUN ou d'un DEFAULT. L'extension peut ici remettre a zéro toutes ses variables internes (si nécessaire).

ROUTINE DE FIN (QUIT)

Cette routine est appelée lors d'un retour au système. Elle doit faire un ménage complet, et ne laisser en mémoire aucune trace du passage de l'extension.

LA TABLE DES INSTRUCTIONS

Dans cette table sont codées toutes les nouvelles instructions et fonctions qu'ajoute l'extension. Cette table contient aussi les pointeurs sur les routines de traitement.

La table doit débuter par :
```
dc.w 1,1
dc.b $80,-1
```

Suivent ensuite les instructions et fonctions. En premier, un pointeur relatif sur la routine de traitement, par rapport au début de la table (généralement baptisée Tk) :

Si c'est une instruction :
```
        dc.w Routine-Tk,1
```

Si c'est une fonction :
```
        dc.w 1,Routine-Tk
```

Suit le nom de l'instruction en lettres minuscules, la dernière lettre étant signalée par l'addition de 128 (bit 7 à 1) :

```
        dc.b "no","m"+$80
```

Un point d'exclamation au DEBUT du nom sert de MARQUEUR sur ce nom. Un nom VIDE ($80 seul) indique a l'AMOS de rechercher le marqueur précédent. Voir plus loin pour comprendre l'utilisation de ce système.

Il faut maintenant donner a l'AMOS toutes les indications nécessaires pour le test de la syntaxe :

Le type de l'instruction :

- Instruction : "I"
- Fonction retournant:
  - un entier "0"
  - un réel "1"
  - une chaîne "2"

La liste de paramètres, terminée par -1 :
- Aucun : -1 seul
- Un paramètre
  - entier : "0"
  - réel : "1"
  - chaîne : "2"
- Un séparateur, seuls sont autorisés :
  - la virgule, codée par ","
  - TO, codé par "t"

Quelques exemples vont rendre tout cela plus clair.

```
; Instruction ESSAI A,B
        dc.w    AdEssai-Tk,1
        dc.b    "essa","i"+$80,"I0,0",-1

; Instruction ANT A$,A TO B
        dc.w AdANT-Tk,1
        dc.b "an","t"+$80,"I2,0t0",-1

; Fonction entière =AMIGA(A,B,C,D). A, B, C et D sont entiers
        dc.w 1,AdAMIGA-Tk
        dc.b "amig","a"+$80,"00,0,0,0",-1

; Fonction réelle =LIONO(A). A est un réel
        dc.w    1,AdMAX
        dc.b    "lion","o"+$80,"11",-1
```

Certaines instructions ou fonctions peuvent accepter différents nombres de paramètres, par exemple SCREEN COPY. Il faut alors donner chaque syntaxe à la suite l'une de l'autre, en terminant la liste des paramètres par -2. La dernière liste se terminera par -1. Pour éviter de taper à chaque fois le nom de l'instruction, on peut utiliser les marqueurs vus plus haut. Voici la définition de SCREEN COPY dans AMOS :

```
; SCREEN COPY A TO B
        dc.w    Scop2-Tk,Synt-Tk
        dc.b    "!screen cop","y"+$80,"I0t0",-2

; SCREEN COPY A,X1,Y1,X2,Y2 TO B,X3,X4
        dc.w Scop8-Tk,Synt-Tk
        dc.b $80,"I0,0,0,0,0t0,0,0",-2

; SCREEN COPY A,X1,Y1,X2,Y2 TO B,X3,X4,M
        dc.w Scop9-Tk,Synt-Tk
        dc.b $80,"I0,0,0,0,0t0,0,0,0",-1
```

Notez bien que vous devez avoir une entrée de routine pour CHAQUE jeu de paramètres.

TRAITEMENT D'UNE INSTRUCTION

Lors de l'interprétation, AMOS appelle votre routine. Les paramètres sont poussés dans la pile (A3), en ordre inverse. Votre routine doit dépiler TOUS les paramètres demandés, (sous peine de plantage !).

Vous ne devez pas modifier les registres A4, A5 et A6. A5 pointe sur la zone de données du basic, l'extension a donc un accès direct à tout l'AMOS (voir à ce sujet le fichier EQU.S).

TRAITEMENT D'UNE FONCTION

Le déroulement est pratiquement identique à celui d'une instruction. La valeur retournée doit être poussée en -(a3).

LA TABLE D'ADRESSES

Lors du démarrage de l'extension, AMOS fournit une donnée importante en A0 : la table d'adresses du Basic. Cette table contient des sauts à d'importantes routines d'AMOS.

Pour y accéder, il suffit de faire :

```
Move.l     Table(pc),a0
Jsr   Fonction(a0)
```

Voici une rapide vue d'ensemble des fonctions les plus intéressantes.

- Table + $04 : routine de traitement des erreurs normales. Cette routine provoque une erreur. Vous devez lui transmettre en D0 le numéro de l'erreur souhaitée. Tous les numéros se trouvent dans le "Handy Index" de l'AMOS. Cette fonction ne revient jamais, inutile de faire un JSR.

* Paramètres d'entrée :
  - D0.l = numéro de l'erreur

* Exemple d'appel, pour provoquer un "Illegal Function Call" :

```
Move.l Table(pc),a0
Moveq #23,d0
Jmp (a0)
```

Les valeurs contenues dans les registres A4-A5-A6 DOIVENT être identiques à celle fournies lors de l'appel de l'instruction.

La pile A3 est restaurée par le système lors d'un appel d'erreur : vous pouvez donc tester les erreurs lors du dépilement des paramètres (très utile lorsqu'il y en a beaucoup!) sans avoir à vous en pré-occuper.

- Table + $08 : routine de traitement des erreurs spécifiques. Cette routine vous permet d'avoir vos propres messages d'erreurs.

* Paramètres d'entrée:
  - D0.l = numéro de l'erreur dans la nouvelle liste ;
  - D1.l = numéro de l'erreur à partir duquel ON ERROR fonctionnera.

Vous pouvez ainsi avoir des erreurs non détournables par le Basic (par exemple, en cas de crash imminent!). Dans l'exemple, toutes les erreurs de l'extension sont détournables.
  - A0.l = adresse de début des messages. Chaque message se termine par un octet à zéro. Exemple de liste :

```
ErrList:
```

```
    dc.b "Attention clavier trop chaud!",0
    dc.b "Moniteur sur le point d'imploser",0
    dc.b "Coca-Cola détecté entre les touches",0
  ; etc...
```

* Exemple d'appel :

```
  Moveq #Erreur,d0
  Moveq #0,d1
  Lea ErrList(pc),a0
  Move.l Table(pc),a1
  Jmp 8(a1)
```

- Table + $0C : trouve l'adresse d'une banque. Petite routine utile si vous désirez récupérer des données dans une banque de mémoire.

* Paramètres d'appel :
- D3.l = numéro de la banque ;

* Paramètres de sortie :
- A0.l = adresse de la banque si réservée ;
- D0.l = longueur de la banque ;

La routine provoquera une erreur "Bank not reserved" si elle ne peut trouver la banque demandée.

- Table + $10 : routine de "tests". AMOS appelle cette routine avant tous les branchements et boucles. C'est elle qui provoque le rafraichissement des bobs, sprites et écrans, qui teste le CONTROL-C, appelle les menus etc... Vous devez appeller cette routine régulièrement si votre extension garde la main pendant plus de 1/50ième de seconde. Elle ne modifie aucun registre à part D0 et A0 et ne nécessite aucun paramètre d'appel.

- Table + $14: Wait. Attend un certain nombre de balayages écran tout en appelant la routine de tests.

* Paramètres d'appel:
  - D3.l = nombre de balayages à attendre ;

- Table +$18 : adresse d'un écran. Trouve les adresses d'une structure écran AMOS. Cette structure contient toute la définition interne de l'écran : position, adresses des bitmaps etc...

* Paramètres d'appel :
  - D1.l = numéro de l'écran ou bien fonction spéciale (=LOGIC(n), =PHYSIC(n)) représentant l'écran logique ou physique.

* Parametres de sortie:
  - A0.l = adresse de la structure écran ;
  - D0.l = adresse dans cette structure des adresses des bitmaps. Si un simple numéro d'écran ou la fonction =LOGIC est envoyé à la routine, D0 pointe les bitmaps LOGIQUES. Si vous demandez =PHYSIC, vous pointerez l'écran PHYSIQUE.

* Exemple d'appel :

```
  Move.l (a3)+,d1
  Move.l Table(pc),a0
  Jsr $18(a0)
  Move.w NbPlan(a0),d1* Defini dans le fichier EQU.S
  Subq.w #1,d1
  Move.l d0,a1 * Adresse des adresses
Loop:
  Move.l (a1)+,a2
  Move.l #-1,(a2)* Pour s'amuser !
  dbra d1,Loop
```

- Table + $1C: adresse ou banque mémoire? Un certain nombre de fonctions AMOS vous permettent de représenter l'adresse de début d'une banque mémoire par le simple numéro de cette banque.

Exemple : Bload "Essai.Bin",10 et Bload "Essai.Bin",Start(10) sont deux instructions équivalentes.

La routine numéro $1C fait cette operation pour vous. Elle dépile automatiquement le paramètre de la pile (a3).

* Paramètre d'entrée :
  - (A3).l paramètre P à vérifier ;

* Paramètre de sortie:
  - D3.l adresse de la banque si P<16, ou même valeur si P>16 ;

- Table + $20 : effacement de banque mémoire.

* Paramètre d'entrée :
  - D3.l numéro de la banque à effacer. Vous pouvez ainsi créer dans une extention un instruction effaçant toutes les banques d'une seul coup (bien pratique ça, pourquoi que je ne l'ai pas mis dans le Basic dès le départ, hein?). Cette routine ne provoque pas de message d'erreur si la banque n'est pas réservée.

- Table + $24 : demander un espace de chaîne. Cette routine n'est accessible qu'à partir de la version 1.23 (disponible maintenant !). Son but est de réserver un espace dans la zone des variables pour retourner une chaîne de caractères. Elle gère automatiquement la "garbage collection" (ou ramassage des miettes).

* Paramètre d'entrée:
  - D3.l taille de la chaîne demandée ;

* Paramètres de sortie:
  - A0.l et A1.l adresse de stockage autorisée.

* Exemple d'appel: un routine inversant majuscules et minuscules :

```
******* =MAJMINS(A$)
MajMin
        move.l (a3)+,a2     * Adresse de la chaine
        moveq #0,d3
        move.w (a2)+,d3     * Taille
        move.l Table(pc),a0
        jsr $24(a0)         * Demande la taille

        move.w d3,(a1)+     * Taille de la réponse
        beq.s Mm4           * Si chaine vide
        subq.w #1,d3
Mm1     move.b (a2)+,d0
        cmp.b #"A",d0       * Majuscule -> Minuscule
        bcs.s Mm3
        cmp.b #"Z",d0
        bhi.s Mm2
        add.b #$20,d0
        bra.s Mm3
Mm2     cmp.b #"a",d0       * Minusucule -> Majuscule
        bcs.s Mm3
        cmp.b #"z",d0
        bhi.s Mm3
        sub.b #$20,d0
Mm3     move.b d0,(a1)+
        dbra d3,Mm1

Mm4     move.l a0,d3        * Retourne le résultat
        moveq #2,d2
        rts
```

Voilà, on s'arrête là pour ce mois-ci. Mais on ne se quittera pas comme ça, puisque je vous livre le listing d'une extension AMOS qui ajoute l'instruction ODD LOKE et la fonction ODD LEEK. Ca n'est pas grand chose, mais ça vous montrera comment il faut faire.

```
* Exemple simple d'extension au basic AMOS
* Assemble avec GENAM2
* Ajoute une instruction: Odd Loke adresse,donnée : un LOKE
* qui marche pour des adresses impaires
* et une fonction: =Odd Leek(Adresse)


*       INCLUSION DES VARIABLES SYSTEME
Include "Equ.s"

*  NUMERO DE L'EXTENSION DANS LA LISTE
ExtNumb  equ 10

*       ROUTINE APPELLEE LORS DU CHARGEMENT (COLD START)
move.l  a0,d0
lea     Tk(pc),a0
move.l  d0,Table-Tk(a0)
lea     Welcome(pc),a1
lea     Warm(pc),a2
lea     End(pc),a3
moveq   #0,d2
moveq   #ExtNumb-1,d1
moveq   #0,d0
rts

*       SCREEN RESET (rien ici!)
Warm    rts
*       QUIT (idem)
End     rts


*       INSTRUCTION "OddLoke A,B"
Odd_Loke
move.l  (a3)+,d0
move.l  (a3)+,d1
move.l  d1,a0
btst    #0,d1
bne.s   Lok1
move.l  d0,(a0)
rts
Lok1    moveq   #3,d1
Lok2    rol.l   #8,d0
move.b  d0,(a0)+
dbra    d1,Lok2
rts

*       FONCTION "Odd Leek(A)"
Odd_Leek
move.l  (a3)+,d0
move.l  d0,a0
btst    #0,d0
bne.s   Lik1
move.l  (a0),d3
rts
Lik1    moveq   #3,d1
Lik2    rol.l #8,d3
   move.b  (a0)+,d3
dbra    d1,Lik2
rts


*       LISTE DES INSTRUCTIONS ET FONCTIONS
Tk:     dc.w    1,0
dc.b    $80,-1
dc.w    Odd_Loke-Tk,1
dc.b    "odd lok","e"+$80,"I0,0",-1
dc.w    1,Odd_Leek-Tk
dc.b    "odd lee","k"+$80,"00",-1
dc.w    0

*       ADRESSE DE LA TABLE DU BASIC
Table   dc.l    0

*       TITRE DE L'EXTENSION
Welcome dc.b    27,"Y",48+15,"Lisez l'ANT!!!",0

*
dc.l    0
```

NE MANQUEZ PAS LE DEMARRAGE DU 36.15 ANT LE 16 MARS 1991 A PARTIR DE 20hOO

LE MOIS PROCHAIN DANS L'ANT

L'AGENDA DE PRESENCE (jours et heures) DE NOS JOURNALISTES SUR LE 3615 ANT

L'INVITE DU MOIS : FRED FISH

UTILITAIRE : RAJOUTEZ DES MENUS DEROULANTS A VOTRE SHELL

TOOLBOX : POWER-WINDOWS

DEMOS : UNE ROUTINE DE VECTOR-BALLS

DES NOUVELLES DU DEVKIT

ET BIEN PLUS ENCORE...

TOOLBOX : Devpac 2 : le Kit des Cats

La version 2 de Devpac, entièrement ré-écrite pour l'Amiga, permet la programmation en assembleur avec autant de facilité qu'un langage interprété genre Basic.

Le kit fourni par HISOFT comprend l'éditeur de textes sources, l'assembleur, le déboggueur et un linker du domaine public, BLink version 7.2, le tout dans un environnement intégré permettant l'écriture, la compilation et l'exécution sans passer par le CLI.

GenAm2, l'éditeur, est loin d'être aussi performant que d'autres du domaine public, comme *Az* ou même *MicroEmacs* : pas de macros re-définissables, édition d'un seul fichier en même temps... Il est cependant amplement suffisant pour des programmes assembleur et extrèmement rapide, aussi bien dans la gestion du texte que dans les recherches/remplacements. Il est de plus paramétrable, de sorte que le programmeur peut choisir la taille du source qu'il va assembler (jusqu'à 3 Mo !), la création ou non de fichiers backups (.BAK), l'auto-indentation des lignes (très pratique) ou encore le passage à la ligne automatique (c'est une question de goût).
Quasiment toutes les fonctions disponibles sont accessibles soit par les menus déroulants, soit par le clavier, qu'il s'agisse du déplacement du curseur (avec une émulation des touches de *Wordstar* s'il vous plaît !), du chercher-remplacer ou encore de la sauvegarde/chargement du source. A noter que pour ces deux dernières fonctions, **GenAm2** utilise le sélecteur de l'arp.library, à condition bien sûr que celle-ci se trouve dans le répertoire LIBS: du *WorkBench* (elle est de toute manière fournie sur les disquettes d'origine de *Devpac*). Dans le cas contraire, un bête requester demandant le nom du fichier est utilisé. La souris quant à elle ne sert qu'à positionner directement le curseur, bien que l'on apprenne rapidement à s'en passer.

Le plus indiscutable de **GenAm2** est de permettre l'appel de l'assembleur et du déboggueur sans quitter l'éditeur : le cycle de développement usuel édition/assemblage/déboggage en est grandement accéléré. On retrouve là la philosophie qui a fait le succès de *Borland* dans le monde PC, philosophie d'ailleurs reprise de-ci de-là et notamment par *Lattice* avec la version 5 de son compilateur C.

L'ASSEMBLEUR

L'assembleur **GenIm2** suit entièrement les définitions énoncées par *Motorola* pour le 68000 ainsi que pour les 68010 et 68020, et accepte, comme beaucoup de ses confrères, quelques petites améliorations, comme par exemple la possibilité d'utiliser l'instruction MOVE au lieu de MOVEA ou encore DBRA au lieu de DBE.
GenIm2 permet l'assemblage du texte source soit en fichier objet prêt à être linké, soit directement en programme exécutable. Il est malheureusement impossible de créer un fichier binaire pur, pour inclusion, par exemple, au sein d'un programme Basic (il faut pour ce faire passer par le déboggueur). **GenIm2** peut être rendu résident et être appelé depuis le CLI ou **GenAm2**. Dans ce dernier cas, c'est l'éditeur qui se charge de lui transmettre toutes les options nécessaires, ce qui réduit son appel à l'appui sur une simple touche.
L'assemblage peut se faire soit en mémoire pour permettre un déboggage immédiat, soit sur disque, soit enfin ne pas avoir lieu du tout, histoire de vérifier la syntaxe du programme, le tout suivant deux modes appelés Fast et Slow, le second utilisant moins de mémoire (utile pour les *Amiga* avec seulement 512 Ko de RAM).

Outre les directives maintenant classiques sur Amiga (inclusion d'un autre fichier avec INCLUDE, assemblage conditionnel avec IFC et consoeurs), on en trouve quelques unes qui facilitent grandement la tâche du programmeur : ce sont INCBIN (inclusion d'un fichier binaire, par exemple une image-écran, au sein du programme) ou encore SECTION (qui permet d'imposer le type de mémoire, Chip, Fast ou Public).

Autre point fort de l'assembleur, la possibilité d'inclure des labels "locaux", c'est-à-dire valables uniquement entre deux labels "globaux". L'avantage est évident : on pourra trouver plusieurs fois dans le source une étiquette ".BOUCLE" sans que celà ne provoque de message d'erreur. Les macros sont également pleinement supportées et sont même récursives (une macro peut s'appeler elle-même !), ce qui offre des possibilités très intéressantes (voir l'exemple de factorielle dans le manuel de *Devpac II*, page 66).

Autre excellent bon point pour **GenIm2**, il optimise sur simple demande le code produit. Celà va du forçage des branchements courts (BRA.S au lieu de BRA) quand celà est possible au remplacement de certaines instructions par leur forme "QUICK", en passant par l'adressage court et la suppression des déplacements nuls dans les modes d'adressage du type indirect avec déplacement: ***d(An)*** et indirect avec indexation et déplacement : ***d(An,Ri)***.

Enfin, tout fichier binaire produit, qu'il soit objet ou exécutable, peut intégrér des informations de déboggage que **MonAm2**, ou tout autre déboggueur au format AmigaDOS, pourra par la suite exploiter.

LE DEBOGGUEUR

Il s'agit sans doute du plus surprenant et du plus puissant des trois programmes composant le *Devpac*. S'intégrant parfaitement dans l'univers multitâche de l'Amiga, il permet de traquer impitoyablement les bogues et les gurus sans disturber les autres tâches actives. Autrement dit, même **MonAm2** chargé, si une autre tâche vient à "planter", le Guru se manifestera tout de même.

MonAm2 est un déboggueur symbolique, c'est-à-dire capable de reprendre et d'utiliser les labels définis dans le fichier source, si ceux-ci existent. Dans le cas contraire, les adresses hexadécimales sont utilisées.
L'utilisation de **MonAm2** n'est possible qu'au clavier. Même s'il ouvre son propre écran (au sens Intuition du terme), aucun menu déroulant n'est disponible. Ce qui pourraît paraître à priori gênant se révèle rapidement fort agréable, d'autant plus que toutes les commandes ont été choisie de manière "parlante" (T pour Trace, L pour Load, etc.).

L'écran est découpé en trois fenêtres : registres, désassemblage et dump mémoire. La première affiche donc le contenu de tous les registres du 68000 (en hexadécimal) ainsi que, pour les registres d'adresses, les 4 premiers octets de l'adresse qu'ils pointent. Il est évidemment possible de modifier leur valeur à tout moment.

La fenêtre de désassemblage est par défaut pointée sur le PC, l'instruction en cours étant repérée par un signe supérieur (>), mais il est bien entendu possible de la faire pointer sur n'importe quelle partie de la mémoire, voire même de la scinder en deux pour comparer deux routines. Comme on l'a déjà dit, les informations de déboggage sont utilisées si elles existent et, plus fort encore, les appels aux libraries du système sont affichés en clair (JSR _LVOOpenScreen(a6) au lieu de JSR -$C6(a6)). Enfin, le désassemblage peut-être obtenu sur imprimante ou sur disque en vue d'un ré-assemblage ultérieur par **GenIm2**.
La fenêtre de dump mémoire est également scindable en deux, afin de comparer deux

zones de mémoires. Il est bien entendu possible d'éditer la zone pointée pour en modifier le contenu. Autre aspect intéressant, la fenêtre peut être "liée" à un registre donné, afin d'afficher en permanence la zone de mémoire qu'il pointe. Les opérations de remplissage, recherche et remplacement intelligent (c'est-à-dire pouvant couvrir deux zones se chevauchant) sont évidemment supportées.

Le controle du déroulement du programme en cours de dégoggage est assez précis. **MonAm2** connaît cinq types de points d'arrêt (breakpoint), à savoir : simple (effacé après avoir été atteint), permanent, compté (arrêt du programme après un certain nombre de passages), conditionnel (arrêt du programme si une condition est remplie) ou... involontaire ("plantage" du 68000 après une erreur d'adressage, une instruction illégale, un TRAP, etc.). Dans ce dernier cas, il est possible de reprendre l'exécution du programme comme si rien ne s'était passé, simplement en changeant la valeur du PC. Enfin, la trace (déroulement pas à pas du programme) offre quatre possibilités : exécution de l'instruction pointée par le PC en traçant ou non les sous-programmes (BSR et JSR), exécution du corps d'une boucle (DBcc) jusqu'à sa fin, ou enfin incrémentation du PC sans exécuter l'instruction en cours.

Enfin, il est possible de quitter **MonAm2** en interrompant le programme tracé (mais attention à la mémoire réservée qui n'est pas libérée, aux écrans, fenêtres et fichiers non fermés, etc.) ou bien en le laissant continuer sa tâche comme s'il avait été lancé depuis le CLI.

PAS MAL, NON ?

Le pack de développement d'HiSoft offre tout ce qu'il faut pour réaliser depuis les petits projets jusqu'aux programmes de haut-niveau. Il est certes avant tout ciblé "système" - les programmeurs de démos et de jeux ont peut-être été un peu oubliés - mais reste pour l'instant le meilleur outil de développement en assembleur qui soit sur l'*Amiga*. L'intégration du cycle édition-assemblage-déboggage est un plus indéniable qui permet un gain de temps considérable. A noter qu'une version "Pro" de *Devpac* existe - au moins en Angleterre - qui permet notamment le déboggage depuis une autre machine (*Amiga* ou *Atari ST*).

Denis Obriot
et Stéphane Schreiber

K-SEKA : c'est ka..

Frédéric Mazué et Max n'ont jamais caché leur antipathie vis-à-vis de l'assembleur de Kuma Computers, qui a pourtant toujours réussi à gagner les faveurs des démomakers, quelque soit la machine concernée.

C'est vrai qu'il a un look préhistorique, le K-Seka. Il fonctionne par "commandes", c'est-à-dire qu'une fois chargé, il attend patiemment les ordres du programmeur, qui peuvent être d'éditer le texte source, l'assembler, désassembler ou dumper une partie de la mémoire. On se croirait sous un Debug amélioré sous MS-DOS.

L'éditeur est sans doute le plus mauvais qu'il m'ait été donné d'utiliser sur un ordinateur. On y accède par la touche Esc et ne mérite même pas que l'on s'y attarde : aucune fonction n'est directement accessible (chargement, sauvegarde, chercher, remplacer...), il faut tout d'abord retourner au mode "commandes". Les capacité multitâches de l'Amiga permettent heureusement de palier à ce défaut, en utilisant un autre éditeur, K-Seka ne servant alors plus que d'assembleur.

ARGHHH, ORG

Le macro-assembleur ne présente aucune particularité ou fonctionalité particulière ; il se contente d'assembler le plus fidèlement possible le texte source, ce qui est quand même la moindre des choses ! Il peut produire un code binaire selon deux formats : code objet pour linkage ultérieur ou code image pur, c'est-à-dire sans aucune information de relogement. Ce mode est idéal pour créer de petites routines relogeables à intégrer, par exemple, au sein de programmes Basic, et c'est d'ailleurs celui qui manque à Devpac. L'inconvénient est qu'il oblige à donner une adresse de chargement du programme (sauf dans le cas de programmes écrits en mode relatif au PC) à l'aide de la directive ORG. Une telle pratique est bien entendu à proscrire totalement dans un environnement multitâche, ou un programme peut être chargé par le DOS à n'importe quelle adresse libre et susceptible de l'accueillir. Celà convient par contre parfaitement aux démos qui ont besoin de se loger en CHIP-RAM pour fonctionner correctement... Celà dit, la directive SECTION de Devpac propose une solution élégante à ce problème, et c'est celle que nous avons adoptée dans l'ANT (regardez la rubrique "démos" pour vous en convaincre).

Une commande particulière permet d'inclure un fichier binaire dans le code produit, très utile pour charger des écrans, musiques ou autres données externes dans le programme. Il suffit pour celà de réserver à l'assemblage suffisamment de place (avec la directive "blk", l'équivalent du "dcb" prôné par Motorola). Par contre, la directive INCLUDE, très utile pour programmer des applications "amiga-friendly", n'a pas été implémentée.

Enfin, il est à noter que notre pourfendeur de bugs devant l'éternel, Frédéric Mazué, assure que le K-Seka en grouille, acceptant par exemple des instructions ne faisant pas partie du jeu du 68000, et refusant au contraire d'en assembler de parfaitement valides !

MINI-MONITEUR

Le K-Seka possède également un moniteur-débogueur intégré, qui, s'il est loin d'être aussi puissant que d'autres existant sur l'Amiga, permet d'explorer la mémoire, de tracer le programme instruction par instruction en visualisant le contenu des registres, de positionner des points d'arrêt, etc. Il n'est malheureusement pas symbolique, c'est-à-dire qu'il n'affiche pas les labels utilisés dans le code source, mais des adresses hexa-décimales (sur 32 bits, bien sûr, ce qui rend la lecture assez difficile). Dans l'ensemble, ce débogueur est le seul point intéressant du K-Seka.

K-SEKA VS THE OTHERS

K-Seka souffre à l'évidence d'un défaut incorrigible : il date. Sa conception même découle de principes mis au point voilà des années, conception aujourd'hui désuettes. Il donne l'impression assez désagréable de piloter un PC ou un Atari ST (une version du K-Seka existe sur chacune de ces machines) car ne tirant absolument pas partie des capacités propres à l'Amiga (fenêtres, etc.). De plus, l'impossiblité de créer des programmes directement exécutables sans passer auparavant par un compacteur quelconque, le rend plutôt inaccessible au débutant en assembleur sur Amiga. Il est en tout cas hors de question de l'utiliser pour concevoir un projet de large envergure (même le Métacomco est plus efficace), mais pour de petites routines, pourquoi pas ?

ExecBase : les bibliothèques publi

Devinette : quel est le moyen à la fois le plus simple, le plus efficace et le plus sûr d'accéder à toute la puissance du système d'exploitation de l'Amiga ? Tous ceux qui ont répondu "les bibliothèques" ont le droit de lire la suite de cet article.

Bien sûr, on peut s'en passer, des bibliothèques. Il existe même certains cas où c'est obligatoire, notamment lorsque l'on se fiche du multitâche comme de l'an 40, et que tout ce que l'on souhaite, c'est faire de zoulis effets avec le Copper et/ou afficher des zénormes bobs avec le Blitter. Messieurs les Demo- et GameMakers, vous m'aurez compris.

Mais dès que l'on se penche un peu plus avant sur la cohabitation de plusieurs tâches au même instant dans la même machine (subtil résumé du terme "multitâche"), les choses deviennent un peu plus hardues, voire franchement impossibles. Heureusement, l'Amiga met à notre disposition plusieurs séries de fonctions regroupées par thèmes dans ce que l'on appelle, justement, les bibliothèques ("libraries" en anglais).

CE QU'IL FAUT SAVOIR

Qu'est-ce qu'une bibliothèque, concrètement ? Il ne s'agit ni plus ni moins que d'un ensemble de fonctions, auxquelles on accède via une adresse de base. Pour clarifier les choses, l'adresse de base représente l'adresse d'une table de pointeurs sur les fonctions de la bibliothèque. Ces pointeurs sont relatifs à l'adresse de base et négatifs, car en fait l'adresse de base pointe sur la fin de la table. Pour accéder à une fonction particulière, il faut en connaître son numéro d'offset (c'est-à-dire sa position dans la table) ainsi bien sûr que l'adresse de base de la bibliothèque. L'appel ne peut se faire qu'à partir de l'assembleur, via l'instruction JSR. La règle veut que ce soit le registre A6 qui contienne l'adresse de base de la librarie (règle d'ailleurs obligatoire et immuable, étant donné que la majorité des fonctions accède à des données propres à la bibliothèque qui les contient, en utilisant A6 comme index). Un exemple typique d'appel d'une fonction serait :

```
        movea.l   Adresse_de_base,a6
        jsr       Offset_de_la_fonction(a6)
```

Pour qu'un programme puisse accéder aux fonctions d'une bibliothèque donnée, il doit d'abord lui en demander la permission. En jargon de programmeurs, on appelle ça "ouvrir la bibliothèque". Et comment fait-on pour ouvrir une bibliothèque ? On appelle une fonction d'une bibliothèque, tout simplement et paradoxalement en même temps.

Tout celà mérite une explication. Il existe en fait une et une seule bilbiothèque à laquelle on peut accéder sans rien demander à personne : il s'agit d'exec.library ; c'est elle qui se charge de la gestion du multitâche au sein de l'Amiga (partage du temps du microprocesseur, allocations de mémoire, communication entre tâches, etc.). L'adresse de base d'exec.library est la seule adresse "fixe" du système d'exploitation, garantie par Commo- dore-Amiga de le rester jusqu'à la saint Glin-Glin. On la trouve à l'adresse 4 :

```
ExecBase  EQU     4

          ...
          movea.l   ExecBase,a6
          jsr       Fonction_Exec(a6)
          ...
```

Partant de là, on peut ouvrir toutes les autres libraries que l'on souhaite.

Y'A QUELQU'UN ?

Les conventions mises au point par les gens de chez Commodore-Amiga indiquent que le passage de paramètres aux fonctions des bibliothèques se fait par l'intermédiaire des registres du microprocesseur. C'est en effet la méthode la plus rapide, mais également la plus gourmande en mémoire, puisqu'elle oblige à sauvegarder les registres utilisés par les fonctions. Heureusement, ces mêmes conventions limitent les dégâts, en garantissant que seuls les registres a0, a1, d0 et d1 risquent d'être modifiés par les fonctions. Tous les autres sont préservés. Enfin, un usage "intelligent" des registres est fait, puisque toutes les adresses sont transmises par l'intermédiaire des registres d'adresse a0 à a5 (a6 étant occupé par l'adresse de base de la bibliothèque) et toutes les autres données, par l'intermédiaire des registres... de données, oui. Une exception à cette règle : la dos.library, qui n'utilise que les registres de données. Enfin, dernière règle, quand une fonction doit retourner une valeur, c'est le registre d0 qui la contient.

UN EXEMPLE !

Pour illustrer celà, nous allons utiliser l'exec.library pour ouvrir l'intuition.library (celle qui contient toutes les fonctions de gestion des fenêtres, menus déroulants et autres requesters). Le code présenté ici est écrit en assembleur avec Devpac II, et en C avec le compilateur Lattice. Jugez-vous même de la différence entre les deux langages.

```
;; Flash.s
; Cet exemple ouvre la bibliothèque "intuition.library" et
; appelle la fonction DisplayBeep() qui fait flasher l'écran
;
  INCLUDE"exec/exec_lib."
  INCLUDE"intuition/intuition_lib.i"

main  leaIntName(pc),a1
; Nom de la librairie à ouvrir
  moveq  #0,d0 ; N° de la version demandée
  CALLEXEC  OpenLibrary
; Appel de la fonction
  move.l  d0,_IntuitionBase
; Sauve l'adresse de base
  beq  NoInt ; sauf en cas d'erreur

  move.l  d0,a6 ; Adresse de base dans a6
  suba.l a0,a0  ; Paramètre pour DisplayBeep
  CALLINT DisplayBeep
; Appel de la fonction

  move.l  a6,a1  ; Paramètre pour CloseLibrary
  CALLEXEC  CloseLibrary
; Appel de la fonction

NoInt moveq  #0,d0
; Retour au CLI
  rts

_IntuitionBase:
```

```
  dc.l 0

IntName dc.b "intuition.library",0
  even

  END
```

Le même en C, maintenant.

```
/*
* Flash.c
* Cet exemple ouvre la bibliothèque "intuition.library" et
* appelle la fonction DisplayBeep() qui fait flasher l'écran
*/

#include <exec/types.h>
#include <exec/libraries.h>
#include <intuition/intuitionbase.h>

VOID main(void)
{
struct IntuitionBase *IntuitionBase;

  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary("intuition.library", 0L);
  if (IntuitionBase != NULL)
  { DisplayBeep(NULL);
    CloseLibrary((struct Library *)IntuitionBase);
  }
}
```

Reste encore un point à éclaircir : où trouve-t-on les numéros d'offsets des fonctions et comment sait-on quels paramètres elles attendent et, pour les programmeurs en assembleur, dans quels registres les passer ? Il existe heureusement de la documentation spécialisée, notamment les fameux Rom Kernal Manuals, dont Frédéric Mazué vous dit du bien chaque fois qu'il le peut. Il s'agit en effet de la documentation officielle de Commodore-Amiga sur les routines du système d'exploitation. Mais l'on peut également en trouver une liste (sans aucune explication, hélas) dans la Bible de l'Amiga, dont Frédéric Mazué vous dit du mal chaque fois qu'il le peut.

SPECIAL ASSEMBLEUR

Alors que le langage C fournit un appel direct aux fonctions des bibliothèques (les références étant résolues lors du 'linkage), les programmeurs en langage-machine doivent en définir les offsets dans chacun de leurs programmes. Commodore - et Hisoft dans le cas de Devpac II - vient fort heureusement à leur rescousse, en incluant dans leur assembleur un pool de fichiers baptisés "fichiers include", qu'il suffit d'inclure dans le source pour pouvoir les utiliser. Ces fichiers contiennent la liste des offsets de toutes les bibliothèques ainsi que des macros facilitant leur utilisation. Pour reprendre l'exemple précédent, la macro CALLEXEC est définie comme suit :

```
CALLEXEC MACRO
  move.l (_SysBase).w,a6
  jsr _LVO1(a6)
  ENDM

_SysBase  EQU 4
```

Sachant que dans le fichier "exec/exec_lib.i", on trouve l'EQUate que voici :

```
_LVOOpenLibrary EQU -552
```

on en déduit que le programme assemblé ressemblera à quelque chose comme :

```
Start: lea IntName(pc),a1
  moveq  #0,d0
  move.l 4.w,a6
  jsr -552(a6)
  move.l _IntuitionBase,d0
  ...
```

(note : les trois lettres "LVO" signifient Library Vector Offset", soit en français dans le texte, Offset de Vecteur de Bibliothèque).

DERNIER POINT IMPORTANT

Et même très important : lorsque l'on ouvre une bibliothèque dans le but d'utiliser ses fonctions, il convient de la refermer avant de terminer le programme et de retourner au CLI ou au Workbench. Pour vous en

ADRESSES

Les adresses de base des bibliothèques portent des noms particuliers, qui sont utilisés par le linker dans le cas du langage C, et par les macros dans le cas de l'assembleur. Voici un mémento des principales bibliothèques avec le nom symbolique de leur adresse de base.

Bibliothèque	Nom en Assembleur	Nom en C
exec.library	_SysBase ou ExecBase.	SysBase ou AbsExecBase
intuition.library	_IntuitionBase.	IntuitionBase
graphics.library	_GfxBase.	GfxBase
dos.library	_DOSBase.	DOSBase
layers.library	_LayersBase.	LayersBase
icon.library	_IconBase.	IconBase

iconvaincre, examinons de plus près le fonctionnement de la fonction OpenLibrary().

1) dans un premier temps, elle recherche dans la Ram si la bibliothèque demandée n'a pas été déjà ouverte par une autre tâche (c'est souvent le cas, puisque le Workbench et AmigaDOS utilisent eux-mêmes plusieurs bibliothèques). Si c'est le cas, elle passe directement au point 5 ;

2) dans le cas contraire, elle recherche si la bibliothèque demandée est située en Rom et si oui, elle saute au point 5 ;

3) sinon, c'est que la bibliothèque se trouve sur disque (généralement dans le tiroir LIBS: de la disquette Workbench), auquel cas elle la charge en mémoire et passe au point 5;

4) si on arrive là, c'est que la bibliothèque demandée n'existe pas, et OpenLibrary retourne une valeur indiquant l'erreur (en l'occurence, il s'agit de la valeur NULL) ;

5) une fois la biliothèque trouvée (ou chargée depuis le disque), OpenLibrary saute à une routine interne à la bibliothèque, qui s'occupe d'initialiser différents paramètres qui lui sont propres. Celà peut fort naturellement inclure des réservations de mémoire. De plus, une même bibliothèque pouvant être utilsée par plusieurs tâches "en même temps", un compteur interne est incrémenté, qui comptabilise le nombre de "clients" de la bibliothèque ;

6) enfin, l'adresse de base de la bibliothèque nouvellement ouverte et initialisée est renvoyée au programme appelant dans le registre d0.

Lorsqu'on ferme une bibliothèque via CloseLibrary(), ce sont les opérations inverses qui sont effectuées :

1) saut à une routine interne à la bibliothèque, qui libère la mémoire allouée lors de l'ouverture et décrémente le nombre de "clients" ;

2) si le nombre de "clients" est égal à zéro, la bibliothèque est tout bonnement effacée de la Ram. Il faudra la recopier depuis la Rom ou la recharger depuis le disque lors d'un prochain appel à OpenLibrary().

On voit donc qu'oublier de fermer une bibliothèque conduit immanquablement à des "pertes de mémoire", qui peuvent tôt ou tard s'avérer critiques pour le système.

A noter que si vous avez chargé le Workbench en incluant la commande "LoadWB -debug" dans votre Startup-Sequence, le menu "invisible" supplémentaire propose l'option "flushlibs", qui permet de remédier à une "oubli" de fermeture.

THE END

Utiliser les bibliothèques présente deux avantages : dans un premier temps, on s'évite toutes les galères propres au multitâche que d'autres ont déjà résolues pour nous. Et dans un deuxième temps, on est sûr de rester compatible avec toutes les versions futures et à venir du système d'exploitation... à condition toutefois de respecter les règles imposées par Commodore-Amiga et de ne pas taper "n'importe comment n'importe où". Mais celà est une autre histoire...

WE NEED YOU : l'ANT est sans cesse à la recherche de nouveaux talents pour compléter son équipe pourtant déjà nombreuse et compétente, notamment pour la rubrique ExecBase. Si vous êtes spécialiste dans un domaine particulier (infographie, vidéo, démos, multitâche ou que sais-je encore ?), n'hésitez surtout pas à nous contacter pour de fructueuses relations, et plus si affinités. Ecr. au jnl. qui trsm.

La solution est extrèmement simple et ne présente avec le Copper qu'un seul trait commun : son nom de "rasters". En effet, pour créer cet effet impressionnant, il ne faut pas utiliser le Copper mais le Blitter. Je ne m'attarde pas à vous présenter le Blitter et ses attributs, ceci a déjà été fait dans cette revue (cf. Commodore Revue 21 à 24). Passons donc tout de suite à l'explication : il suffit d'afficher tout d'abord 1 bob de 3 pixels de hauteur sur 16 de largeur et de l'animer de droite à gauche. L'astuce est de recopier ce bob quelques lignes plus bas, tout en modifiant ses coordonnées pour créer un effet de vague, le Copper se chargeant ensuite de recopier la dernière ligne des bobs sur ce qui reste de l'ecran, en modifiant les valeurs modulo des bitplanes.

Le programme d'exemple qui suit anime 80 barres de ce type. Si vous désirez redessiner le bob qui nous sert de modèle, vous devrez prendre en considération les contraintes suivantes quant à sa taille :

- 16 pixels de largeur (1 mot) ;
- 3 lignes de hauteur ;
- 3 bitplanes (8 couleurs).

Notez bien que cette routine peut être très nettement optimisée, de façon à prendre encore moins de temps pendant le VBL (80 lignes de raster).

```
;---------------------------- *
; ROUTINE DE BARRES DE "COPPER" VERTICAL
; PROGRAMME SOUS DEVPAC V2.14 PAR CHARLET FRANCK
;---------------------------- *

WIDTH     EQU       120       ;3 PLANS DE 40 OCTETS DE LARGE
HEIGHT    EQU       256
RASSIZE   EQU       WIDTH*HEIGHT

;--- INITIALISATIONS MACRO ASSEMBLEUR *
INITPIC:  MACRO
          MOVE.L    #1,D0
          LEA       2,A0
          MOVE.W    D0,6(A0)
          SWAP      D0
          MOVE.W    D0,2(A0)
          ENDM

VIDEPIC:  MACRO
          LEA       1,A0
          LEA       2,A1
VIDE@:    MOVE.W    A0,-(SP)
          CLR.W     (A0)+
          MOVE.W    (SP)+,D0
          MOVE.W    D0,$DFF180
          CMP.L     A1,A0
          BLO.S     VIDE@
          ENDM

; -- DEBUT DU PROGRAMME *
START:    LEA       GFXNAME,A1
          MOVEQ     #0,D0
          MOVE.L    4.W,A6
          JSR       -552(A6) ;OPENLIBRARY
          MOVE.L    D0,A1
          MOVE.L    38(A1),OLDCOP
          JSR       -414(A6) ;CLOSELIBRARY
          LEA       $DFF000,A0
          MOVE.W    2(A0),OLDDMA
          MOVE.W    $1C(A0),OLDINT
          BSR       PRG       ;SAUT AU PRG PRINCIPAL
          LEA       $DFF000,A0
          MOVE.W    OLDDMA(PC),D0
          BSET      #15,D0
          MOVE.W    D0,$96(A0)          ;REMET DMACON
          MOVE.W    OLDINT(PC),D0
          BSET      #15,D0
          MOVE.W    D0,$9A(A0)          ;REMET INTENA
          MOVE.L    OLDCOP(PC),$80(A0) ;REMET LA COPPERLIST
```

les Rasters Verticaux

Créer des barres verticales d'un pixel de précision avec le Copper... Impossible ! me direz-vous, et vous aurez raison. Pourtant de nombreuses démos possèdent cet effet (comme la Mégademo de Dragons).

```
        MOVE.W    D0,$88(A0)          ;SYSTEME
        RTS
OLDDMA: DC.W      0
OLDINT: DC.W      0
OLDCOP: DC.L      0

GFXNAME: DC.B     "graphics.library",0
        EVEN
;--- INITIALISATIONS GENERALES *
PRG:    LEA       $DFF000,A6
        VIDEPIC   PLAN,PLAN+RASSIZE
        INITPIC   PLAN,CBP1
        INITPIC   PLAN+40,CBP2
        INITPIC   PLAN+80,CBP3
        MOVE.W    #$7FFF,$9A(A6)      ;VIDE $DFF09A (INTENA)
        MOVEQ     #0,D0     ;VIDE D0 (ON NE SAIT JAMAIS!)
INIT:   MOVE.B    #$87,$BFD100        ;STOPPE LE DRIVE
        NOP
        NOP
        MOVE.B    #$FF,$BFD100
;--- INITIALISATION DE LA COPPERLIST *
        MOVE.L    #COPPERLIST1,$80(A6)        ;NOTRE COPPERLIST
        MOVE.W    D0,$88(A6)                  ;EST VALIDEE.
        MOVE.W    $1C(A6),D0
        AND.W     #$F000,D0
        ADD.W     D0,D0
        MOVE.W    D0,$9A(A6)          ;SUPRIME LES INTERUPTIONS
;--- BOUCLE DU PRG *
RASTERBEAM:
        CMP.B     #80,6(A6);BOUCLE A LA LIGNE 262
        BNE.S     RASTERBEAM
        BSR.S     EFFACE    ;EFFACE D'ABORD L'ECRAN
        BSR.S     BARRES    ;MAINTENANT ON PEUT AFFICHER NOS BARRES
        BTST      #6,$BFE001          ;ATTEND LE CLICK GAUCHE
        BNE.S     RASTERBEAM
        RTS
;--- EFFACE L'ECRAN DES BARRES *
EFFACE: BTST      #14,$DFF002         ;ATTEND LE BLITTER
        BNE.S     EFFACE
        MOVE.L    #PLAN,$54(A6)       ;POINTE SUR L'ECRAN
        MOVE.L    #$1000000,$40(A6)  ;ET VIDE LE!
        MOVE.W    #(1*64)+60,$58(A6) ;DIMENSION
        RTS
;--- ROUTINE DES RASTERS VERTICAUX *
BARRES: LEA       PLAN+2,A3
        LEA       BLOCKRASTER,A4
        LEA       TABLE,A5
        MOVE.W    4(A4),D0
        MOVEQ     #79,D7    ;80 BARRES
BOUCLE: MOVE.W    (A5,D0.W),D6        ;AJOUTE   D0   (VITESSE)   A   LA
TABLE
        MOVE.W    D6,D5
        LSR.W     #3,D6     ;= DIVU #8,D6 MAIS EN + RAPIDE
        MOVE.L    A3,A2
        ADD.W     D6,A2
        AND.W     #$F,D5    ;D5 NE DOIT PAS DEPASSER $000F
        ROR.W     #4,D5     ;ROTATION DE 4 BITS
        MOVE.W    D5,D6
        OR.W      #$FE2,D6  ;MINTERN A $FE2    (ABCD)
        SWAP      D6
        MOVE.W    D5,D6
WAITBLT1:
        BTST      #14,2(A6);ATTEND LE BLITTER
        BNE.S     WAITBLT1
        MOVE.L    D6,$40(A6)
        MOVE.W    #$FFFF,$44(A6)      ;MASQUE LA MOITIE DU BOB
        CLR.W     $46(A6)
        MOVE.W    #36,$64(A6)         ;MODULOS SOURCES ET DEST
        MOVE.W    #36,$62(A6)
        MOVE.W    #36,$60(A6)
        MOVE.W    #36,$66(A6)
        MOVE.L    #SOURCEBOB,$50(A6)
        MOVE.L    #SOURCEMASK,$4C(A6)
        MOVE.L    A2,$48(A6)
        MOVE.L    A2,$54(A6)
        MOVE.W    #(3*64)+2,$58(A6)  ;DIMENSION DU PLAN 1
WAITBLT2:
        BTST      #14,2(A6)
        BNE.S     WAITBLT2
        MOVE.L    #SOURCEBOB+120,$50(A6)
        MOVE.L    #SOURCEMASK,$4C(A6)
        ADD.W     #40,A2
        MOVE.L    A2,$48(A6)
        MOVE.L    A2,$54(A6)
        MOVE.W    #(3*64)+2,$58(A6)  ;DIMENSION DU PLAN 2
WAITBLT3:
        BTST      #14,2(A6)
        BNE.S     WAITBLT3
        MOVE.L    #SOURCEBOB+240,$50(A6)
        MOVE.L    #SOURCEMASK,$4C(A6)
        ADD.W     #40,A2
        MOVE.L    A2,$48(A6)
        MOVE.L    A2,$54(A6)
        MOVE.W    #(3*64)+2,$58(A6)  ;DIMENSION DU PLAN 3
WAITBLT4:
        BTST      #14,2(A6)
        BNE.S     WAITBLT4
        MOVE.W    #$9F0,$40(A6)       ;COPIE   LA   BARRE   QUELQUES
LIGNES
        CLR.W     $42(A6)   ;PLUS BAS
        MOVE.L    #-1,$44(A6)
        MOVE.L    #$40004,$64(A6)
        MOVE.L    A3,$50(A6)
        ADD.L     #120,A3
        MOVE.L    A3,$54(A6)
        MOVE.W    #(3*64)+18,$58(A6)
        ADD.W     (A4),D0   ;COPIE LA BARRE+L'ECART
        AND.W     #$7FE,D0
        DBF       D7,BOUCLE
        MOVE.W    2(A4),D0  ;ANIME!
        ADD.W     D0,4(A4)
        AND.W     #$7FE,4(A4)         ;CES   5   LIGNES   EVITENT   DES
COMPARAISONS
        AND.W     #$7FE,2(A4)         ;SUR LA TABLE
```

Avertissement

Nous nous sommes toujours attachés, dans l'ANT, à vous présenter une programmation "propre", c'est-à-dire respectueuse de la machine et de son système d'exploitation, et par voie de fait, compatible avec tous les modèles de l'Amiga, quelle qu'en soit la configuration (extensions, drives supplémentaires...). Aviez-vous remarqué par exemple, que la directive "ORG" était absente de tous nos listings assembleur, car bannie de notre vocabulaire ?

Les démos échappent la plupart du temps à ces règles, parfois pour des questions d'optimisation des routines, souvent par fainéantise, ignorance ou nostalgie des 8 bits des "coders". Nous essaierons cependant tant que faire se peut, de "limiter les dégâts" en proposant des programmes les plus concis possible, et en insistant sur le fait que l'algorythme mis en oeuvre reste finalement plus important que la routine elle-même.

D'accord!

Merci

```
      AND.W  #$7FE,(A4) ;(ET C'EST PLUS RAPIDE!)
      RTS

;--- SECTION DES VARIABLES *
;--- LES 2 PREMIERES VALEURS DONNENT L'ECART ENTRE LES BARRES *
;--- ET LA VITESSE DE DEPLACEMENT *
;--- VOUS POUVEZ EVIDEMMENT LES CHANGER A VOLONTE!!! *
;--- ESSAYEZ PAR EXEMPLE 1010 POUR L'ECART
BLOCKRASTER:
      DC.W 2020  ;ECART ENTRE LES BARRES
      DC.W 10  ;VITESSE
      DC.W 10

;--- TABLE DU DEPLACEMENT SUR X (BON COURAGE!)
TABLE: DC.W  $0080,$0080,$0081,$0082,$0083,$0083,$0084,$0085
      DC.W $0086,$0087,$0087,$0088,$0089,$008A,$008A,$008B
      DC.W $008C,$008D,$008E,$008E,$008F,$0090,$0091,$0091
      DC.W $0092,$0093,$0094,$0095,$0095,$0096,$0097,$0098
      DC.W $0098,$0099,$009A,$009B,$009B,$009C,$009D,$009E
      DC.W $009E,$009F,$00A0,$00A1,$00A2,$00A2,$00A3,$00A4
      DC.W $00A5,$00A5,$00A6,$00A7,$00A7,$00A8,$00A9,$00AA
      DC.W $00AA,$00AB,$00AC,$00AD,$00AD,$00AE,$00AF,$00B0
      DC.W $00B0,$00B1,$00B2,$00B2,$00B3,$00B4,$00B5,$00B5
      DC.W $00B6,$00B7,$00B7,$00B8,$00B9,$00BA,$00BA,$00BB
      DC.W $00BC,$00BC,$00BD,$00BE,$00BE,$00BF,$00C0,$00C0
      DC.W $00C1,$00C2,$00C2,$00C3,$00C4,$00C4,$00C5,$00C6
      DC.W $00C6,$00C7,$00C8,$00C8,$00C9,$00Ca,$00CA,$00CB
      DC.W $00CB,$00CC,$00CD,$00CD,$00CE,$00CF,$00CF,$00D0
      DC.W $00D0,$00D1,$00D2,$00D2,$00D3,$00D3,$00D4,$00D5
      DC.W $00D5,$00D6,$00D6,$00D7,$00D7,$00D8,$00D9,$00D9
      DC.W $00DA,$00DA,$00DB,$00DB,$00DC,$00DC,$00DD,$00DD
      DC.W $00DE,$00DE,$00DF,$00E0,$00E0,$00E1,$00E1,$00E2
      DC.W $00E2,$00E3,$00E3,$00E4,$00E4,$00E4,$00E5,$00E5
      DC.W $00E6,$00E6,$00E7,$00E7,$00E8,$00E8,$00E9,$00E9
      DC.W $00EA,$00EA,$00EA,$00EB,$00EB,$00EC,$00EC,$00EC
      DC.W $00ED,$00ED,$00EE,$00EE,$00EE,$00EF,$00EF,$00F0
      DC.W $00F0,$00F0,$00F1,$00F1,$00F1,$00F2,$00F2,$00F2
      DC.W $00F3,$00F3,$00F3,$00F4,$00F4,$00F4,$00F5,$00F5
      DC.W $00F5,$00F6,$00F6,$00F6,$00F6,$00F7,$00F7,$00F7
      DC.W $00F8,$00F8,$00F8,$00F8,$00F9,$00F9,$00F9,$00F9
      DC.W $00FA,$00FA,$00FA,$00FA,$00FA,$00FB,$00FB,$00FB
      DC.W $00FB,$00FB,$00FC,$00FC,$00FC,$00FC,$00FC,$00FC
      DC.W $00FD,$00FD,$00FD,$00FD,$00FD,$00FD,$00FD,$00FE
      DC.W $00FE,$00FE,$00FE,$00FE,$00FE,$00FE,$00FE,$00FE
      DC.W $00FE,$00FE,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF
      DC.W $00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF
      DC.W $00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF
      DC.W $00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FF,$00FE
      DC.W $00FE,$00FE,$00FE,$00FE,$00FE,$00FE,$00FE,$00FE
      DC.W $00FE,$00FE,$00FD,$00FD,$00FD,$00FD,$00FD,$00FD
      DC.W $00FD,$00FC,$00FC,$00FC,$00FC,$00FC,$00FC,$00FB
      DC.W $00FB,$00FB,$00FB,$00FA,$00FA,$00FA,$00FA,$00FA
      DC.W $00FA,$00F9,$00F9,$00F9,$00F9,$00F8,$00F8,$00F8
      DC.W $00F8,$00F7,$00F7,$00F7,$00F6,$00F6,$00F6,$00F6
      DC.W $00F5,$00F5,$00F5,$00F4,$00F4,$00F4,$00F3,$00F3
      DC.W $00F3,$00F2,$00F2,$00F2,$00F1,$00F1,$00F1,$00F0
      DC.W $00F0 $00F0 $00EF,$00EF,$00FE,$00EE,$00EE,$00ED
      DC.W $00ED,$00EC,$00EC,$00EC,$00EB,$00EB,$00EA,$00EA
      DC.W $00EA,$00E9,$00E9,$00E8,$00E8,$00E7,$00E7,$00E6
      DC.W $00E6,$00E5,$00E5,$00E4,$00E4,$00E4,$00E3,$00E3
      DC.W $00E2,$00E2,$00E1,$00E1,$00E0,$00E0,$00DF,$00DE
      DC.W $00DE,$00DD,$00DD,$00DC,$00DC,$00DB,$00DB,$00DA
      DC.W $00DA,$00D9,$00D9,$00D8,$00D7,$00D7,$00D6,$00D6
      DC.W $00D5,$00D5,$00D4,$00D3,$00D3,$00D2,$00D2,$00D1
      DC.W $00D0,$00D0,$00CF,$00CF,$00CE,$00CD,$00CD,$00CD
      DC.W $00CB,$00CB,$00CA,$00CA,$00C9,$00C8,$00C8,$00C7
      DC.W $00C6,$00C6,$00C5,$00C4,$00C4,$00C3,$00C2,$00C2
      DC.W $00C1,$00C0,$00C0,$00BF,$00BE,$00BE,$00BD,$00BC
      DC.W $00BC,$00BB,$00BA,$00BA,$00B9,$00B8,$00B7,$00B7
      DC.W $00B6,$00B5,$00B5,$00B4,$00B3,$00B2,$00B2,$00B1
      DC.W $00B0,$00B0,$00AF,$00AE,$00AD,$00AD,$00AC,$00AB
      DC.W $00AA,$00AA,$00A9,$00A8,$00A7,$00A7,$00A6,$00A5
      DC.W $00A5,$00A4,$00A3,$00A2,$00A2,$00A1,$00A0,$009F
      DC.W $009E,$009E,$009D,$009C,$009B,$009B,$009A,$0099
      DC.W $0098,$0098,$0097,$0096,$0095,$0095,$0094,$0093
      DC.W $0092,$0091,$0091,$0090,$008F,$008E,$008E,$008D
      DC.W $008C,$008B,$008A,$008A,$0089,$0088,$0087,$0087
      DC.W $0086,$0085,$0084,$0083,$0083,$0082,$0081,$0080
      DC.W $007F,$007F,$007E,$007D,$007C,$007C,$007B,$007A
      DC.W $0079,$0078,$0078,$0077,$0076,$0075,$0075,$0074
      DC.W $0073,$0072,$0071,$0071,$0070,$006F,$006E,$006E
      DC.W $006D,$006C,$006B,$006A,$006A,$0069,$0068,$0067
      DC.W $0067,$0066,$0065,$0064,$0064,$0063,$0062,$0061
      DC.W $0061,$0060,$005F,$005E,$005D,$005D,$005C,$005B
      DC.W $005A,$005A,$0059,$0058,$0058,$0057,$0056,$0055
      DC.W $0055,$0054,$0053,$0052,$0052,$0051,$0050,$004F
      DC.W $004F,$004E,$004D,$004D,$004C,$004B,$004A,$004A
      DC.W $0049,$0048,$0048,$0047,$0046,$0045,$0045,$0044
      DC.W $0043,$0043,$0042,$0041,$0041,$0040,$003F,$003F
      DC.W $003E,$003D,$003D,$003C,$003B,$003B,$003A,$0039
      DC.W $0039,$0038,$0037,$0037,$0036,$0035,$0035,$0034
      DC.W $0034,$0033,$0032,$0032,$0031,$0030,$0030,$002F
      DC.W $002F,$002E,$002D,$002D,$002C,$002C,$002B,$002A
      DC.W $002A,$0029,$0029,$0028,$0028,$0027,$0026,$0026
      DC.W $0025,$0025,$0024,$0024,$0023,$0023,$0022,$0022
      DC.W $0021,$0021,$0020,$001F,$001F,$001E,$001E,$001D
      DC.W $001D,$001C,$001C,$001B,$001B,$001B,$001A,$001A
      DC.W $0019,$0019,$0018,$0018,$0017,$0017,$0016,$0016
      DC.W $0015,$0015,$0015,$0014,$0014,$0013,$0013,$0013
      DC.W $0012,$0012,$0011,$0011,$0011,$0010,$0010,$000F
      DC.W $000F,$000F,$000E,$000E,$000E,$000D,$000D,$000D
      DC.W $000C,$000C,$000C,$000B,$000B,$000B,$000A,$000A
      DC.W $000A,$0009,$0009,$0009,$0009,$0008,$0008,$0008
      DC.W $0007,$0007,$0007,$0007,$0006,$0006,$0006,$0006
      DC.W $0005,$0005,$0005,$0005,$0005,$0004,$0004,$0004
      DC.W $0004,$0004,$0003,$0003,$0003,$0003,$0003,$0003
      DC.W $0002,$0002,$0002,$0002,$0002,$0002,$0002,$0001
      DC.W $0001,$0001,$0001,$0001,$0001,$0001,$0001,$0001
      DC.W $0001,$0001,$0000,$0000,$0000,$0000,$0000,$0000
      DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
      DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0000
      DC.W $0000,$0000,$0000,$0000,$0000,$0000,$0000,$0001
      DC.W $0001,$0001,$0001,$0001,$0001,$0001,$0001,$0001
      DC.W $0001,$0001,$0002,$0002,$0002,$0002,$0002,$0002
      DC.W $0002,$0003,$0003,$0003,$0003,$0003,$0003,$0004
      DC.W $0004,$0004,$0004,$0005,$0005,$0005,$0005,$0005
      DC.W $0005,$0006,$0006,$0006,$0006,$0007,$0007,$0007
      DC.W $0007,$0008,$0008,$0008,$0009,$0009,$0009,$0009
      DC.W $000A,$000A,$000A,$000B,$000B,$000B,$000C,$000C
      DC.W $000C,$000D,$000D,$000D,$000E,$000E,$000E,$000F
      DC.W $000F,$000F,$0010,$0010,$0011,$0011,$0011,$0012
      DC.W $0012,$0013,$0013,$0013,$0014,$0014,$0015,$0015
      DC.W $0015,$0016,$0016,$0017,$0017,$0018,$0018,$0019
      DC.W $0019,$001A,$001A,$001B,$001B,$001B,$001C,$001C
      DC.W $001D,$001D,$001E,$001E,$001F,$001F,$0020,$0021
      DC.W $0021,$0022,$0022,$0023,$0023,$0024,$0024,$0025
      DC.W $0025,$0026,$0026,$0027,$0028,$0028,$0029,$0029
      DC.W $002A,$002A,$002B,$002C,$002C,$002D,$002D,$002E
      DC.W $002F,$002F,$0030,$0030,$0031,$0032,$0032,$0033
      DC.W $0034,$0034,$0035,$0035,$0036,$0037,$0037,$0038
      DC.W $0039,$0039,$003A,$003B,$003B,$003C,$003D,$003D
      DC.W $003E,$003F,$003F,$0040,$0041,$0041,$0042,$0043
      DC.W $0043,$0044,$0045,$0045,$0046,$0047,$0048,$0048
      DC.W $0049,$004A,$004A,$004B,$004C,$004D,$004D,$004E
      DC.W $004F,$004F,$0050,$0051,$0052,$0052,$0053,$0054
      DC.W $0055,$0055,$0056,$0057,$0058,$0058,$0059,$005A
      DC.W $005A,$005B,$005C,$005D,$005D,$005E,$005F,$0060
      DC.W $0061,$0061,$0062,$0063,$0064,$0064,$0065,$0066
      DC.W $0067,$0067,$0068,$0069,$006A,$006A,$006B,$006C
      DC.W $006D,$006E,$006E,$006F,$0070,$0071,$0071,$0072
      DC.W $0073,$0074,$0075,$0075,$0076,$0077,$0078,$0078
      DC.W $0079,$007A,$007B,$007C,$007C,$007D,$007E,$007F
;-- DONNES EN CHIP RAM OBLIGATOIREMENT
      SECTIONCHIP,DATA_C ;MERCI DEVPAC!
COPPERLIST1:
      DC.W$008E,$2C81,$0090,$18C1   ;DIMENSION DE L'ECRAN
      DC.W$0092,$0038,$0094,$00D0
CBP1: DC.W$00E0,$0000,$00E2,$0000   ;POINTEUR DES PLANS
CBP2: DC.W$00E4,$0000,$00E6,$0000
CBP3: DC.W$00E8,$0000,$00EA,$0000
      DC.W$0100,$3200    ;3 PLANS+MODE COULEURS
      DC.W$0102,$0000,$0104,$0000
      DC.W$0108,$0050,$010A,$0050   ;+80 AU MODULOS
      DC.W$0120,$0000,$0122,$0000
      DC.W$0124,$0000,$0126,$0000
      DC.W$0128,$0000,$012A,$0000   ;PAS DE SPRITES
      DC.W$012C,$0000,$012E,$0000
      DC.W$0130,$0000,$0132,$0000
      DC.W$0134,$0000,$0136,$0000
      DC.W$0138,$0000,$013A,$0000
      DC.W$013C,$0000,$013E,$0000
      DC.W$0182,$0FFF,$0184,$0DDF   ;COULEURS DES BARRES
      DC.W$0186,$0BBF,$0188,$099F
      DC.W$018A,$077F,$018C,$055F
      DC.W$018E,$033F,$0180,$0000
      DC.W$7C09,-2
; -- LA LIGNE SUIVANTE PERMET DE RECOPIER LES BARRES SUR TOUT
L'ECRAN:
; -- EN METTANT LES MODULOS A -40, LES BITPLANES POINTENT TOUJOURS
; -- SUR LA MEME LIGNE!
      DC.W$0108,$FFD8,$010A,$FFD8
      DC.L-2    ;LE COPPER A TERMINE SON BOULOT!
; -- DESSIN DU BOB
SOURCEBOB:
      DC.W$A995 ;1ER MOT=DONNEES DU BOB
      DCB.W  59,$0000 ;PUIS 59 MOTS (118 OCTETS)
      DC.W$CE73 ;IDEM 2EME LIGNE DU BOB
      DCB.W  59,$0000
      DC.W$F00F ;ET 3EME LIGNE DU BOB
      DCB.W  59,$0000
; -- MASQUE DU BOB
SOURCEMASK:
      DC.W$FFFF
      DCB.W  59,$0000
;--- DONNES NON INITIALISEES EN CHIP
      SECTIONCHIP,BSS_C

PLAN: DS.BRASSIZE  ;RESERVATION MEMOIRE POUR L'ECRAN

      END
```

Interview : Franck Ostrowski

Pour fêter la sortie de la version PC du GfA-Basic, Micro-Application avait invité son auteur en personne à venir passer quelques heures sur leur stand, au Forum PC. L'occasion pour nous d'aller recontrer ce jeune et sympathique allemand, véritable petit génie de l'informatique

C'est donc par un froid glacial que je débarquai au Parc des Expositions de la Porte de Versailles à Paris, où se tenait ledit salon. Et déjà, permettez-moi un petit apparté pour vous dire que Commodore France y avait un stand ma foi plutôt grand et bien placé, juste devant l'entrée. Le PC Forum étant avant tout un salon professionnel (l'entrée est tout de même à 200 balles !), on pouvait y admirer surtout des PC (belle mentalité !) et... un Amiga 3000. Fin de l'aparté.

Me voici arrivé devant le stand de Micro-Application, où un pot de bienvenue est offert à divers journalistes qui traînaient dans le coin. Hop, je me précipite sur un verre de whisky et demande à ce que l'on me présente le génie de service... Au lieu de ça, j'ai droit à un bonhomme assez bizarre mais tout-à-fait charmant et sympathique, à l'accent germanique assez prononcé : il s'agit de l'une des têtes pensantes de GfA SystemTechnik GmbH, qui accompagne son programmeur-vedette et lui sert de traducteur. Lequel programmeur-vedette est tranquillement assis derrière un PC, à tapoter comme un malade sur son clavier tout neuf - mais qui ne va pas le rester longtemps s'il continue comme ça. L'interview peut commencer.

Moi : Bonjour, Franck.
Lui : Bonjour.
Moi : Commençons par les questions bêtes mais obligatoires... Comment en es-tu arrivé à devenir programmeur chez GfA ?
Lui : Après avoir eu mon Bac, j'ai voulu m'inscrire à l'université pour étudier l'informatique. Malheureusement, je n'avais pas les moyens de vivre et de payer mes études en même temps. Alors je vendais des listings pour Atari 800 XL à des journaux pour gagner un peu d'argent. Un jour, j'ai publié dans Happy Computing un programme nommé Turbo-Basic, qui accélérait le Basic de cet ordinateur et y ajoutait quelques fontions. En lisant le journal, le directeur de GfA SystemTechnik a eu l'idée de m'appeler et m'a proposé de développer un Basic spécifique pour le nouvel Atari de l'époque, le 520 ST, qui disposait d'un très mauvais interprêteur. J'ai tout simplement accepté cette proposition.
Moi : Et le GfA-Basic était né, et il a connu le succès que l'on sait.
Lui : Oui, les versions ST ont été vendues à plus de 600.000 examplaires à travers le monde.
Moi : Pas mal ! Et la version Amiga ?
Lui : Bof... Seulement 35.000 examplaires. Le problème sur l'Amiga, c'est qu'il y a beaucoup de piratage. On a estimé à GfA SystemTechnik que pour un GfA-Basic Amiga vendu, 6 personnes au moins le copiaient...
Moi : C'est à cause de ça que les efforts n'ont pas été poursuivis ?
Lui : Non, pas du tout ! Mais on a eu tellement de problèmes pendant le développement de la version Amiga, et puis j'ai du travailler sur la version PC, tout en pensant au compilateur... C'est très difficile de tenir à ce rythme, mais c'est aussi très excitant !
Moi : Des problèmes ? Quel genre de problème ? C'est en rapport avec le système d'exploitation ?
Lui : Oui et non. Notre grosse erreur a été de confier le travail de l'adaptation à un autre programmeur, afin que je puisse me consacrer à d'autres choses. Il a travaillé pendant presque un an dessus, au bout desquels son truc ne marchait toujours pas ! Il a donc fallu que je reprenne tout à zéro ! Quant à l'AmigaDOS, c'est vrai que c'est un système difficile à aborder quant on est habitué aux 8 bits ou même au ST. Le multitâche impose tellement de contraintes... Mais je dois reconnaître que celà m'a donné quelques bases solides pour la version PC sous Windows qui est également multitâche, et pour une future version UNIX System V, déjà en préparation.
Moi : Je suppose que tout a été écrit entièrement écrit en langage-machine. Tu ne travailles qu'avec ce langage ?
Lui : Tu supposes bien, oui. Mais je programme également en C, voire même en Basic quand il s'agit de faire de petits programmes de test pour les algorythmes. Je gagne beaucoup de temps comme ça.
Moi : Micro-Application vient de sortir le compilateur 3.5, mais la l'interprêteur 3.5, lui, n'est pas encore là... Celà signifie-t-il que GfA SystemTechnik a décidé d'arrêter sa production sur Amiga ?
Lui : Non, nous continuerons à sortir de nouvelles version, notamment pour enlever quelques bugs qui traînent encore. Nous pensons également sortir des extensions spécifiques, comme une "DataBase ToolBox" ou un "TextEditor ToolBox". Mais ça, ça n'est encore qu'à l'état de projet. Comme je le l'ai déjà dit, le piratage sur Amiga freine beaucoup d'éditeurs de logiciels. Et c'est bien dommage, car c'est vraiment une machine qui mériterait de se développer encore plus.
Moi : A qui le dis-tu ! Encore deux petites choses : quels utilitaires de programmation utilises-tu, et que reproches-tu le plus à l'Amiga ?
Lui : J'utilises des assembleurs internes à GfA SystemTechnik, dont certains tournent sur PC. Quant à l'Amiga, le plus gros reproche que je puisse lui faire en tant que programmeur, c'est le Guru : pourquoi faut-il donc qu'il plante toute la machine ? Sur ST par exemple, on a des bombes qui d'affichent à l'écran, mais le blocage de l'ordinateur est rare. Sous Windows, quand une tâche plante, elle est tout simplement désactivée. Sur Amiga, deux fois sur trois, la machine se bloque pour un rien, et si on a pas pris la précaution de sauvegarder, tout est perdu ! Ca m'est déjà arrivé des dizaines de fois !
Moi : Heu... Tu sais que des utilitaires comme GOMF permettent de remédier à ce (léger) défaut ?
Lui : Maintenant, oui, mais à l'époque, je ne le savais pas ! Quand je me suis mis sur l'Amiga, je ne le connaissais absolument pas, et j'ai du faire avec ce que j'avais sous la main, c'est-à-dire les Rom Kernal Manuals en tout et pour tout !
Moi : Ok. Pour terminer, un petit mot sur ce que tu souhaiterais faire ou voir venir dans le futur ?
Lui (après mure réflexion) : Ce que j'aimerais, c'est que la maladie du multitâche contamine tous les constructeurs d'ordinateurs. Je pense que c'est la voie naturelle que va suivre l'informatique dans l'avenir. En tout cas, c'est quelque chose dont on ne peut plus se passer une fois qu'on y a goûté.

La conversation continua quelques minutes, puis je pris congé de monsieur Ostrowski, qui replongea illico dans le programme qu'il avait abandonné pour moi quelques instants plus tôt. In-cor-ri-gi-ble !

Propos recueillis par Stéphane Schreiber

fontes vectorielles

L'Amiga souffre d'un grave défaut au niveau des fontes de caractères : il ne connaît pas les polices vectorielles. Certains programmes, comme ProPage, implémentent donc les leurs, qui suivent leur propre format. En voici un exemple.

La différence entre une police "Bitmap" et une police vectorielle, tient essentiellement en ce que la police vectorielle est affichée par des calculs mathématiques, et non grâce à des matrices de points. Cependant, le calcul revient à transformer une police vectorielle en une matrice affichable, donc pour les petites tailles, il n'y aura pas de différence entre les deux types de polices. Mais l'avantage de la police vectorielle est qu'elle peut générer elle-même toutes ses tailles, sans limite théorique (seulement l'affichage), contrairement à la police Bitmap qui doit posséder chaque taille de corps de caractère, sous peine de représenter le caractère avec un effet d'escalier assez désagréable, en procédant à une multiplication des points de la matrice.

LA THEORIE

Pour représenter la police vectorielle, j'ai renoncé aux coordonnées dites rectangulaires, où un point a pour coordonnées les valeurs x et y par rapport à l'origine, au profit des coordonnées dites polaires, où un point est défini par un rayon et un angle par rapport à l'origine. L'avantage des coordonnées polaires est essentiel lorsque l'on désire procéder à une rotation du caractère, car il suffit d'ajouter l'angle de rotation désiré à l'angle du point et le tour est joué !

Cependant, les instructions graphiques du GfA ne connaissant que les coordonnées rectangulaires (x,y), il faudra procéder à une conversion :

Conversion Polaire à Rectangulaire :
x = r * COS(alpha)
y = r * SIN(alpha)

Conversion Rectangulaire à Polaire :
r = SQR(x~2 + y~2)
alpha = ATN(y/x)
Dans mes routines, j'utilise les fonctions SINQ() et COSQ() qui travaillent dix fois plus vite que SIN() et COS(), même si la précision est moins bonne, elle est amplement suffisante.

LA PRATIQUE

Avant de pouvoir afficher une police, il faut en posséder les données. A cette fin, chaque caractère de la police est formé par un certain nombre de vecteurs, théoriquement illimité, mais moins il y en aura, plus rapide sera l'affichage.
Par soucis de simplification, vous définirez vos caractères à l'aide de coordonnées rectangulaires, puis les données seront converties par le programme 2 en coordonnées polaires, sous la forme de DATA, que vous insérerez ensuite dans le programme 1.

Donc en premier lieu, écrivez vos définitions de caractères dans un fichier appelé "VectorFont.data" (utilisez n'importe quel éditeur de textes) sous la forme caractère, x1, y1, x2, y2, ..., xn, yn, -1,-1

Par exemple : A, 1, 0, 1, 7,...., -1, -1

-1,-1 indique la fin de la définition du caractère.
Chaque caractère est défini à l'aide de segments de droite, il faut donc deux couples de coordonnées pour chaque segment (x1,y1) et (x2,y2). L'origine des points se trouve en haut à gauche de la matrice.
Enfin, vous devez prévoir la taille de la matrice de la définition d'un caractère dans les variables font_base% (largeur, en principe 8) et font_height% (hauteur, en principe 8).

Ces données sont converties en coordonnées polaires pour ne pas avoir à le faire pendant l'exécution du programme, ce qui gagne du temps. Les DATA sont lues et chaque segment est stocké dans un tableau nommé Font_data#(), un autre

OURS-COUCOU-CONCOURS-COU

Envie de participer à notre concours listing ? Rien de plus simple, il suffit de nous envoyer vos créations quelqu'en soit le domaine (jeu, utilitaire, démo ou intro...) et quelque soit le langage utilisé (AmigaBasic, GfaBasic, C, Assembleur, AMOS). Le règlement est tout aussi simple et se plie aux contraintes de la mise en page du journal : le total article + listing ne doit pas dépasser deux pages, soit en moyenne 3000 caractères de texte (soit 50 lignes de 60 caractères) pour 200 lignes de programme (à raison de 75 caractères maximum par ligne). Ces proportions correspondent à 1 colonne de texte pour 3 colonnes de programme.

Qui dit concours dit cadeau : le gagnant se verra décerner non seulement nos plus sincères félicitations, mais également le DevKit ANT, composé de : **- l'assembleur Devpac Amiga version 2 - le compilateur C Lattice version 5 - la documentation officielle Commodore "Amiga Rom Kernal Manual" (3 volumes).**

Le choix du vainqueur sera effectué par la rédaction de l'Amiga News Tech, selon plusieurs critères parmi lesquels l'intérêt du programme bien sûr, mais aussi la qualité de la programmation. Tout le monde n'étant pas graphiste ou musicien, ces deux points ne joueront qu'un infime rôle dans la décision finale. L'article quant à lui ne jouera absolument aucun rôle.

Envoyez vos chefs d'oeuvres : article, source ET exécutable ainsi que tous les fichiers annexes nécessaires à la compilation (images, sons...), accompagnés d'une petite lettre vous décrivant, vous et votre configuration Amiga, à :

Amiga News Tech, 5, rue de la Fidélité
75010 Paris, France . Le cachet de La Poste ne fera foi de rien du tout.

tableau nommé Font%() servant de table d'index pour retrouver les segments des caractères. Il est à 2 dimensions : le premier indice varie de 0 à 255, équivaut au code ASCII. Le second peut prendre les valeurs 1 et 2 : 1 indique l'indice de départ pour Font_data#(), 2 indique le nombre de segments pour le caractère demandé. De fait, rien n'oblige à construire une police entière ; on peut se contenter de définir les caractères dont on a réelleent besoin (d'ailleurs, à titre d'exemple, je vous donne la définition des chiffres 0 à 9 seulement).
Les routines qui suivent sont relativement simples et suffisamment comentées. Vous pourrez même les améliorer, en permettant par exemple d'utiliser plusieurs polices en même temps...

Programme 1 : gestion des fontes vectorielles.

```
@init_vecfont
OPENS 1,0,0,640,512,2,32772  ! Ecran Hires Lace
TITLES #1,"Test de Fontes Vectorielles"
OPENW #1,0,0,640,512,262144,256+2048+4096
r=1
FOR i%=0 TO 360 STEP 5
  CLS
  VSYNC
  @draw_line("023",320,220,r,r,i%)
  r=r+0.2
NEXT i%
PROCEDURE init_vecfont  ! Initialisions pour la fonte vectorielle
  '
  font_base%=8            ! Largeur d'un caractère
  font_height%=8          ! Hauteur d'un caractère
  '
  DIM font%(255,2),font_data(400,4)
  '         |                  |
  '         |__Code ASCII      |__Nombre de segments pour toutes les
lettres
  '                              (à augmenter si nécessaire)
  '     font%(...,1) = Indice de départ pour Font_data#()
  '     font%(...,2) = Nombre de segments constituant la lettre
  '
  '     font_data#(...,1) = Angle
  '     font_data#(...,2) = Rayon/ Début du segment
  '     font_data#(...,3) = Angle
  '     font_data#(...,4) = Rayon/ Fin du segment
  '
  RESTORE segments
  cpt_data%=1    ! Compteur pour l'indice de Font_data#()
  a$=""
  DO
    READ a$
    EXIT IF a$="Fin"
    font%(ASC(a$),1)=cpt_data%
    cpt_n_data%=0
    DO
      READ ang,ray
      EXIT IF ang=-1 AND ray=-1
      font_data(cpt_data%,1)=ang
      font_data(cpt_data%,2)=ray
      READ ang,ray
      font_data(cpt_data%,3)=ang
      font_data(cpt_data%,4)=ray
      INC cpt_data%
      INC cpt_n_data%
    LOOP
    font%(ASC(a$),2)=cpt_n_data%
  LOOP
  '
  ' Format des DATA = "X" (caractère défini),angle,rayon,...,-1,-1
(Fin de définition)
  '
  segments:
  '     insérer ici les DATA
  DATA "Fin"
RETURN
PROCEDURE draw_caract(car$,x%,y%,coef_x,coef_y,angle%)
  '
  ' car$   = chaine contenant le caractère à afficher (ex: "A")
  ' x,y    = coordonnées pour afficher le caractère dans l'écran
  ' coef_x = coefficient de déformation horizontale (multiplication)
  '           (si 1 alors pas de changement)
  ' coef_y = coefficient de déformation verticale (multiplication)
  '           (si 1 alors pas de changement)
  ' angle  = valeur en degrés (0 à 360) pour incliner le caractère
  '           (si   0 alors pas de changement,
  '            si 180 alors le caractère sera à l'envers)
  '
  LOCAL x1%,y1%,x2%,y2%,k%
  a%=font%(ASC(car$),1)              !   Indice   de   départ   pour
font_data#()
  b%=a%+font%(ASC(car$),2)-1         !   Indice   d'arrivée   pour
font_data#()
  '
  ' Lecture des segments et conversion des Coor Polaires en Coor
Rectangulaires
  '
  FOR k%=a% TO b%
    x1%=ROUND(font_data(k%,2)*coef_x*COSQ(font_data(k%,1)+angle%),0)
    y1%=ROUND(font_data(k%,2)*coef_y*SINQ(font_data(k%,1)+angle%),0)
    x2%=ROUND(font_data(k%,4)*coef_x*COSQ(font_data(k%,3)+angle%),0)
    y2%=ROUND(font_data(k%,4)*coef_y*SINQ(font_data(k%,3)+angle%),0)
    LINE x%+x1%,y%+y1%,x%+x2%,y%+y2%
  NEXT k%
RETURN
PROCEDURE draw_line(phrase$,x%,y%,coef_x,coef_y,angle%)
  LOCAL largeur%,k%,xx%,yy%
  largeur%=0
  '
  ' Utilise aussi les conversions Coor Pol/Rec pour aligner les
caractères
  '
  FOR k%=1 TO LEN(phrase$)
    xx%=x%+largeur%*COSQ(angle%)
    yy%=y%+largeur%*SINQ(angle%)
    @draw_caract(MID$(phrase$,k%,1),xx%,yy%,coef_x,coef_y,angle%)
    largeur%=largeur%+font_base%*coef_x
  NEXT k%
RETURN
```

Programme 2 : conversion des coordonnées cartésiennes en coordonnées polaires.

```
CLS
PRINT "Programme pour convertir les coordonnées rectangulaires des
caractères"
PRINT "en coordonnées polaires (angle,rayon)"
OPEN "i",#1,"VectorFont.data"  ! Source
OPEN "O",#2,"VectorFont.lst"   ! Destination
'
WHILE NOT (EOF(#1))
  INPUT #1,a$      ! Caractère défini
  PRINT #2,"DATA ";CHR$(34);a$;CHR$(34);",";
  PRINT
  PRINT "Définition du caractère : ";a$
  PRINT
  DO
    INPUT #1,x$,y$  ! Couple de coordonnées rectangulaires
    EXIT IF x$="-1" AND y$="-1"
    PRINT "Rec(";x$;",";y$;")";
    x=VAL(x$)
    y=VAL(y$)
    rayon=SQR(x^2+y^2)
    IF x=0 THEN
      x=1.0E-12
    ENDIF
    alpha=DEG(ATN(y/x))
    PRINT #2,STR$(alpha);",";STR$(rayon);",";
    PRINT " --> Pol(";alpha;",";rayon;")"
  LOOP
PRINT #2,"-1,-1"
WEND
CLOSE
```

Exemples 3 : coordonnées cartésiennes, à convertir avec le programme 2 et à insérer dans le programme 1 au label "segments:".

```
1,4,0,4,8,4,0,2,2,-1,-1
2,2,2,2,0,2,0,6,0,6,0,6,3,6,3,2,8,2,8,7,8,-1,-1
3,1,0,7,0,7,0,7,8,7,8,1,8,7,4,3,4,-1,-1
4,5,8,5,0,5,0,1,5,1,5,7,5,-1,-1
5,7,0,3,0,3,0,1,3,1,3,7,3,7,3,7,8,7,8,1,8,-1,-1
6,7,0,1,0,1,0,1,8,1,8,7,8,7,8,7,4,7,4,1,4,-1,-1
7,1,0,7,0,7,0,1,8,-1,-1
8,1,0,7,0,7,0,7,8,7,8,1,8,1,8,1,0,1,4,7,4,-1,-1
9,1,0,7,0,7,0,7,8,7,8,1,8,1,0,1,4,1,4,7,4,-1,-1
```

2 utilitaires sinon rien!

Vous connaissez sans doute les raccourcis claviers Commodore-N et Commodore-? qui permettent de passer l'écran du Workbench en avant ou en arrière de la pile d'écrans. Seulement voilà, il n'est pas possible, dans le cas où trois écrans sont utilisés, de faire apparaître de cette manière celui du "milieu".

Le programme que je vous propose installe un nouveau raccouci clavier permettant la permutation circulaire des écrans. De cette manière, chaque écran deviendra accessible sans qu'il soit nécéssaire d'utiliser cette vilaine petite souris toujours ensevelie sous une montagne de documents. Et puisque c'est la mode, nous allons agrémenter ce programme d'un blanker. En supposant que les touches du raccourci soient choisies, voyons d'abord ce qu'il est possible de faire afin de tester si celle-ci sont actionnées.
Il y a énormément de possibilités sur un Amiga. Il serait possible d'utiliser les messages Intuition, mais l'inconvénient serait alors la nécéssité de la présence d'une fenêtre active. Le même inconvénient se présenterait avec les devices CON: et NEWCON:. Enfin la méthode "je programme Amstrad ou Atari" qui consisterait à lire en permanence les registres hardware afin de savoir quelles touches sont pressées, ne vaut pas un clou sur Amiga, car cela ruinerait les performances du multitâche.

Considérant que notre programme devra être une véritable tâche de fond totalement transparente, voici les deux bonnes possibilités qu'il nous reste : utiliser le console.device ou utiliser l'input.device. Sans raisons particulières, j'ai choisi l'input.device.

INPUT.DEVICE

L'input.device est une tâche qui est toujours lancée dès que le système démarre. Elle communique en permanence avec le clavier et les gameports afin d'obtenir des évènements "crus" en provenance du clavier, de la souris ou du joystick. De plus, le timer.device travaille en étroite collaboration avec l'input.device pour la gestion de la vitesse de répétition des touches. Comme tous les devices, l'input.device est programmé à l'aide des routines système, Opendevice(), CloseDevice(), DoIO(), SendIO() et AbortIO().
Les commandes traditionelles START, STOP, INVALID et FLUSH sont implémentées et donc valides. Les autres commandes standard ne sont pas reconnues. En revanche, l'input.device supporte les commandes spécifiques suivantes : IND_WRITEEVENT, IND_ADDHANDLER, IND_REMHANDLER, IND_SETTHRESH, IND_SETPERIOD, IND_SETMPORT, IND_SETMTRIG et IND_SETMTYPE.

Ne paniquez pas, voici l'explication de ce charabia.

IND_WRITEEVENT : nous commençons là par une commande tout à fait curieuse qui permet à l'utilisateur non pas de lire un événement reçu par l'input.device, mais de lui en envoyer un fabriqué maison qui sera reconnu et traité de la même façon que tout évènement extérieur. Application : il est de cette façon possible de s'amuser en enregistrant par exemple les événements souris à partir de l'input.device, puis de les "rejouer" en renvoyant les événements à l'aide de cette commande dans le même input.device.

IND_ADDHANDLER permet d'ajouter un handler à la chaîne. Késako ? Et bien un peu plus haut, j'ai dit qu'il était mauvais de lire en permanence l'état des registres-machine. J'ai également dit que l'input.device communiquait en permanence avec le clavier et les gameports... Je vois des vilains avec un sourire en coin qui pensent que je ne sais même pas ce que je dis. Mais si, car l'input.device surveille le clavier en permanence, mais à partir de routines

```
/*****************************************************************************
* Programme: CRBlanker.c                                                     *
* Fonctions: - Bteindre moniteur si ordinateur inutilisé après               *
*              délai réglé par utilisateur                                   *
*            - Permuter écrans Intuition                                     *
*                                                                            *
* Utilisation: Ctrl - Help pour régler délai                                 *
*              Amiga gauche - Help pour permuter écrans                      *
* Compilateur: Lattice C 5.10 Compilation: lc -cist -v                       *
* Linkage:    FROM LIB:cback.o+"CRBlanker.o"+"blankerhandler.o"              *
*             TO "CRBlanker"                                                 *
*             LIB LIB:lc.lib LIB:amiga.lib                                   *
*             NODEBUG                                                        *
*             SMALLCODE                                                      *
*             SMALLDATA                                                      *
*             DEFINE __main=__tinymain                                       *
*                                                                            *
* Auteur: F MAZUE pour ANT le 18/11/90                                       *
*****************************************************************************/

#include <exec/types.h>
#include <exec/nodes.h>
#include <exec/lists.h>
#include <exec/memory.h>
#include <exec/ports.h>
#include <exec/io.h>
#include <exec/interrupts.h>
#include <exec/devices.h>
#include <libraries/dos.h>
#include <devices/timer.h>
#include <devices/input.h>
#include <devices/inputevent.h>
#include <intuition/intuitionbase.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <graphics/gfxmacros.h>
#include <hardware/custom.h>
#include <hardware/dmabits.h>
#include <string.h>

#ifdef LATTICE
#include <proto/exec.h>
#include <proto/dos.h>
#include <proto/intuition.h>
#include <proto/graphics.h>
#endif

/*****************************************************************************
*                        DECLARATION POUR CBACK                              *
*****************************************************************************/

#ifdef LATTICE
extern BPTR _Backstdout;
char *_procname = "Blanker";
LONG _BackGroundIO = 1;
LONG _stack = 4000;
LONG _priority = 20;
#endif

/*****************************************************************************
*                                  CONSTANTES                                *
*****************************************************************************/

#define PORTNAME "Blanker_Port"
#define TICK 5L                         /* interval de 5 secondes */
#define Message "CRBlanker 1.0 par F Mazué"

/*****************************************************************************
*                    VARIABLES GLOBALES                                      *
*****************************************************************************/

struct IntuitionBase *IntuitionBase;
struct GfxBase       *GfxBase;

/*****************************************************************************
*            StringInfo pour gadget de chaîne                                *
*****************************************************************************/
```

d'interruptions. Autrement dit, la surveillance est permanente mais ne s'effectue pas tout le temps, seulement au rythme des interruptions de la machine. De même, l'utilisateur qui veut avoir connaissance et pouvoir traiter les événements, doit lui même ajouter un handler d'interruption qui sera chaîné avec les autres selon sa priorité. La rapidité d'éxécution étant de rigueur dans ce genre d'activité, il est de bon ton d'écrire le handler en assembleur. De plus, les conditions d'entrée (registres a0 et a1) sont telles que l'utilisation du C ou d'un autre langage nécessiterait un exercice de style tout à fait inutile, comme on peut sans rendre compte en examinant le source de l'éternel PopCLI fourni avec toutes les versions passées, présentes et à venir du Lattice. A titre indicacheveu (c'est tout de même plus poli qu'indicatif) les handlers d'Intuition ont une priorité de 50 et ceux du console.device, une priorité de 0. Ainsi, avec un handler de priorité 51, il est possible de bidouiller les événements avant qu'Intuition n'y touche pour par exemple inverser les mouvements de la souris (cf. le programme du RKM "libraries and devices" page 700. Oh quelle belle idée de virus !).

IND_REMHANDLER permet de supprimer un handler de la chaîne. C'est la commande inverse de la précédente (si!).
IND_SETTHRESH permet de fixer le délai de répétition des touches.
IND_SETPERIOD permet de fixer le délai à partir duquel commence la répétition des touches.
IND_SETMPORT permet de déterminer le port souris.
IND_SETMTRIG fixe les conditions qui généreront un événement souris.
IND_SETMTYPE fixe le device associé au port de la souris.

COMMENT ECRIRE UN HANDLER ?

C'est très simple : il suffit de savoir qu'en a0, est passé le premier événement et qu'en a1, est passé le champ Data de la structure d'interruption correspondant à votre handler. Pour ceux qui n'auraient pas tout compris, l'article "Sentinelle" dans Commodore Revue 29 explique les interruptions avec suffisamment de détails. Les bienheureux qui possèdent les RKM pourront également s'y reporter.

L'événement dit "InputEvent" répond d'une structure dont le premier élément pointe sur le prochain, s'il existe. Ceci permet très facilement d'examiner tous les événements en attente et d'agir en conséquence.
Comme condition de sortie, il suffit de placer la valeur initiale de a0 dans d0 et c'est tout.

Voyons un peu la structure InputEvent.

```
struct InputEvent {
  struct InputEvent *ie_NextEvent;
  UBYTE ie_Class;
  UBYTE ie_SubClass;
  UWORD ie_Qualifier;
  union {
    struct {
      WORD ie_x;
      WORD ie_y;
    } ie_xy;
    APTR ie_addr;
  } ie_position;
  struct timeval ie_TimeStamp;
};
```

Le premier élément est déja expliqué. Ensuite :
- **ie_Class** indique la provenance de l'évènement : clavier, souris, gadget, menu, etc. ;

```
UBYTE Buffer[2]="";
struct StringInfo MyStringInfo =
{
  Buffer,
  NULL,
  1,
  2,
  0,
  0,
  0,
  0,
  0,0,
  NULL,
  NULL,
  NULL,
};

/*****************************************************************************
*                                  Gadget de chaine                          *
*****************************************************************************/

struct Gadget MyGadget=
{
  NULL,                                 /* pas d'autre gadget */
  95,48,                                /* position dans la fenêtre */
  20,10,                                /* largeur et hauteur */
  GADGHCOMP,                            /* flags */
  RELVERIFY,                            /* activation */
  STRGADGET,                            /* type de gadget */
  NULL,
  NULL,
  NULL,
  NULL,
  (APTR)&MyStringInfo,                  /* pointeur sur StringInfo */
  0,
  NULL
};

/*****************************************************************************
*                          fenêtre pour réglage delai                        *
*****************************************************************************/

struct NewWindow Reglage =
{
  100,100,                              /* position */
  250,70,                               /* dimensions */
  1,3,                                  /* couleur de fond et crayon */
  GADGETUP,                             /* flag evenement */
  ACTIVATE|GIMMEZEROZERO,               /* flags */
  &MyGadget,                            /* pointeur sur gadget de chaîne */
  NULL,
  "Blanker 1.0 par F MAZUE",
  NULL,
  NULL,
  250,70,                               /* dimensions mini */
  250,70,                               /* dimensions maxi */
  WBENCHSCREEN
};

struct NewScreen NewScreen =
{
  0,0,320,12,1,
  0,0,
  NULL,
  SCREENQUIET,
  NULL,
  NULL,
  NULL,
  NULL
};


char *text_reglage[] =
{
  "(1-9) nb de minutes avant",
  "extinction des feux",
  "(x ou X) arrête programme"
};

BYTE InputSig =-1;
BYTE PermutSig =-1;
BYTE InitSig =-1;

struct Task *task = NULL;

LONG TimeSigMask = NULL;
```

- **ie_SubClass** nous intéresse peu aujourd'hui : ce champ concerne essentiellement Intuition et les menus ;
- **ie_Code** donne le code de la touche clavier ou du bouton de souris responsable de l'évènement ;
- **ie_Qualifier** indique quelle touche spéciale (Shift, Ctrl, Alt, Amiga) est actionnée ;
- **ie_position** est une union dans laquelle on trouvera suivant le type d'événement, soit la position du pointeur de la souris, soit une structure correspondant aux événements temps.

Pour connaitre le code de touches vous pouvez vous reporter au RKM "libraries and devices" page 658. Quelques valeurs intéressantes :

Help = $5f
Esc = $45
Tab = $42
F1 à F10 = $50 à $59
flèches = $4c à $4e

Pour en terminer avec l'input.device, une petite précision peut être nécessaire : il ne faut pas confondre la chaîne de handlers avec la chaîne d'évènements. Chaque handler de la chaîne est une routine à laquelle est passée la chaîne de tous les évènements. Autrement dit, les évènements sont tous examinés par tous les handlers. Ce qui justifie le besoin de rapidité de traitement.

Finalement et pour résumer, ce que nous voulons faire, ce n'est que rajouter un handler au système, cette routine devant elle aussi examiner toute la chaine d'évènements et prendre ses dispositions en conséquence. Grâce à tout ceci, nous allons pouvoir écrire notre programme très facilement.

LE TIMER.DEVICE

Que vient-il faire ici celui-là ? Et bien dans notre programme de permutations d'écrans, je vous ai proposé d'ajouter un blanker, c'est à dire une routine qui se chargera d'éteindre l'écran si durant une certaine période, aucun évènement clavier ou souris n'est apparu. Ceci permet d'économiser le phosphore du tube cathodique.

Pour ce faire, nous allons utiliser le timer.device. Nous n'en dirons pas grand chose aujourd'hui car ce n'est pas notre propos principal. Le timer.device a pour vocation d'envoyer un message au port avec lequel il a été ouvert pour signaler la fin d'un délai programmé.

Il est à remarquer que chaque fois que le timer.device décrémente le délai, un évènement est engendré, ce qui signifie que l'on pourrait intercepter tout ceci dans notre routine de traitement des évènements. Si cette possibilité peut être parfois intéressante, dans le cas qui nous occupe, la simplicité veut tout bonnement attendre de façon on ne peut plus classique, le signal sur le port message.

DEUX TIMERS SINON RIEN

Il y a en effet deux timer.devices dans notre cher Amiga :
- **UNIT_VBLANK.timer**, synchronisé avec le blanc vertical, de bonne précision, et généralement suffisant pour la plupart des applications ;
- **UNIT_MICROHZ.timer**, doué d'une précision d'enfer pour ceux qui auraient cassé leur chronomètre.

Les deux timers se programment de la même façon, si ce n'est que UNIT_VBLANK compte en secondes et UNIT_MICROHZ en microsecondes. Tout ceci étant dit, voyons un peu...

LE PROGRAMME

La routine assembleur du handler tout d'abord. Là encore, pas de grosses explications (je suis si fatigué). L'observateur attentif y verra que si un évènement souris est reçu, un signal est envoyé au

```
LONG InputSigMask = NULL;
LONG PermutSigMask = NULL;
LONG InitSigMask = NULL;
LONG Sig = NULL;

LONG Time_Limite = 24;      /* soit deux minutes */
LONG Time_Count = NULL;

LONG Active = NULL;

struct MsgPort *port = NULL,
*inputdevice_port = NULL,
      *timedevice_port = NULL;

struct IOStdReq *inputreq = NULL;
struct timerequest  *timereq =NULL;

struct Interrupt *InputHandler;

extern struct Custom custom;
struct Custom *cust = &custom;
#define custom (*cust)

/*****************************************************************************
*                    DECLARATION DES PONCTIONS                               *
*****************************************************************************/

extern VOID MyHandler();
VOID Clean_Exit();
VOID LaunchTimer(struct timerequest *tr,LONG secondes);
LONG InitLimite();

/*****************************************************************************
*                              PROGRAMME PRINCIPAL                            *
*****************************************************************************/

VOID main()

{
  struct Screen *BlankScreen = NULL;

  if (port=(struct MsgPort *)FindPort(PORTNAME))
  {
     Active = 1L;
     Clean_Exit();
  }

  #ifdef LATTICE
  if (_Backstdout)
  {

     Write(_Backstdout,Message,sizeof(Message));

Close (_Backstdout);  /* on libère le CLI */
     _Backstdout=0;
  }
  #endif

/*****************************************************************************
*                                  Suis-je déjà là ?                          *
*****************************************************************************/

port = (struct MsgPort *)CreatePort(PORTNAME,0L);
if (!port) Clean_Exit();

/*****************************************************************************
*                    Ouvertures et Initialisations                            *
*****************************************************************************/

  if (!(IntuitionBase = (struct IntuitionBase *)
  OpenLibrary("intuition.library",0))) Clean_Exit();

  if (!(GfxBase = (struct GfxBase *)
  OpenLibrary("graphics.library",0))) Clean_Exit();

  if (!(inputdevice_port = CreatePort(NULL,NULL))) Clean_Exit();
  if (!(inputreq = (struct IOStdReq *)
  CreateExtIO(inputdevice_port,sizeof(struct IOStdReq)))) Clean_Exit();
  if (OpenDevice("input.device",NULL,
  (struct IORequest *)inputreq,NULL)) Clean_Exit();

  if (!(timedevice_port = CreatePort(NULL,NULL))) Clean_Exit();
  if (!(timereq = (struct timerequest *)
  CreateExtIO(timedevice_port,sizeof(struct timerequest))))
Clean_Exit();
  if (OpenDevice(TIMERNAME,UNIT_VBLANK,
  (struct IORequest *)timereq,NULL)) Clean_Exit();
```

programme principal. Donc une petite touchette sur la souris suffira à réallumer l'écran. Si vous jugez que cette sensibilité est excessive, il vous suffit de supprimer ce morceau de programme.

Ensuite, dans le cas d'un évènement clavier, on regarde si la touche Help (S5f) est pressée. Dans ce cas, on examine alors si l'une des touches Ctrl ou Commodore (Amiga-gauche) est pressée. Le signal correspondant sera envoyé au programme principal le cas échéant, sinon ce sera un signal clavier "simple". Si l'évènement est d'une autre nature, aucun signal n'est envoyé et on passe à l'examen de l'évènement suivant s'il y en a un.
Le programme principal maintenant : il est écrit en C car C plus facile. Voyez d'abord dans l'entête, pour le linkage, le #DEFINE __main=__tinymain (notez bien les deux caractères soulignés). Ceci sert à remplacer le _main du C par une routine qui ne s'occupera d'aucune entrée/sortie. C'est une des manières d'éviter, lorsque le programme est lancé depuis le Workbench, l'ouverture d'une fenêtre par défaut.
Viennent ensuite les déclarations pour le cback du Lattice, qui permet au programme de se décrocher du CLI. Si votre compilateur ne dispose pas d'une telle option, ça n'est pas grave, il vous suffira de lancer votre programme par Run.

Après les diverses déclarations, le main() : d'abord, le programme teste s'il a déjà été lancé ; on recherche pour celà un port du nom défini par le programme. Si ce port n'existe pas, celà signifie que le programme est lancé pour la première fois. Dans ce cas, on crée le port et l'on continue. Sinon, si le port existe déjà, c'est que le programme, est déjà lancé et on quitte immédiatement.
Ensuite, les librairies et devices nécéssaires sont ouverts et les signaux alloués. Puis le handler est installé et le timer est lancé une première fois. Le programme se met alors en attente des différents signaux.

L'extinction de l'écran est réalisée comme suit : on ouvre un écran Intuition de très petite hauteur (12) afin d'éviter le problèmes de mémoire. L'encre de fond est mise à 0, couleur noire. Il reste à éteindre le pointeur de souris. Il est possible d'utiliser pour ce faire les routines SetPointer() et ClearPointer() d'Intuition. J'ai choisi une autre solution qui consiste à couper les directement DMA dans le hardware avec la macro OFF_DISPLAY, histoire de vous donner un exemple de programmation des registres-machine en C.

UN BUG (ENORME ET MONSTRUEUX) DU LATTICE

La boucle qui attend les signaux est une boucle sans fin. Normallement, une telle boucle s'écrit en C soit for(;;) soit while(1) mais ne le faites pas ici, car le compilateur vous fabriquerait n'importe quoi sans pour autant vous envoyer de message d'erreur (normal, le programme n'en comporte pas). Pas même un petit warning. Rien.

Ce bug se manifeste comme suit :
- si les fonctions sont déclarées avant main() et définies après, le bug apparaît - si les fonctions sont exprimées avant le main(), le programme est parfois compilé correctement..

Pour remédier infailliblement à celà, j'utilise while(task), task étant l'adresse de base de la tâche. C'est une constante ultra-sûre. En faisant ainsi, le while n'est plus sans test et le programme est toujours compilé correctement. Ce n'est bien sûr pas la seule solution et de loin, on aurait même pu écrire le programme tout autrement, mais je n'ai pas pu résister à la tentation de laisser libre cours à ma médisance. M'enfin, le Lattice n'en reste pas moins (à mon avis) le meilleur compilateur C disponible sur Amiga. Pour le moment.

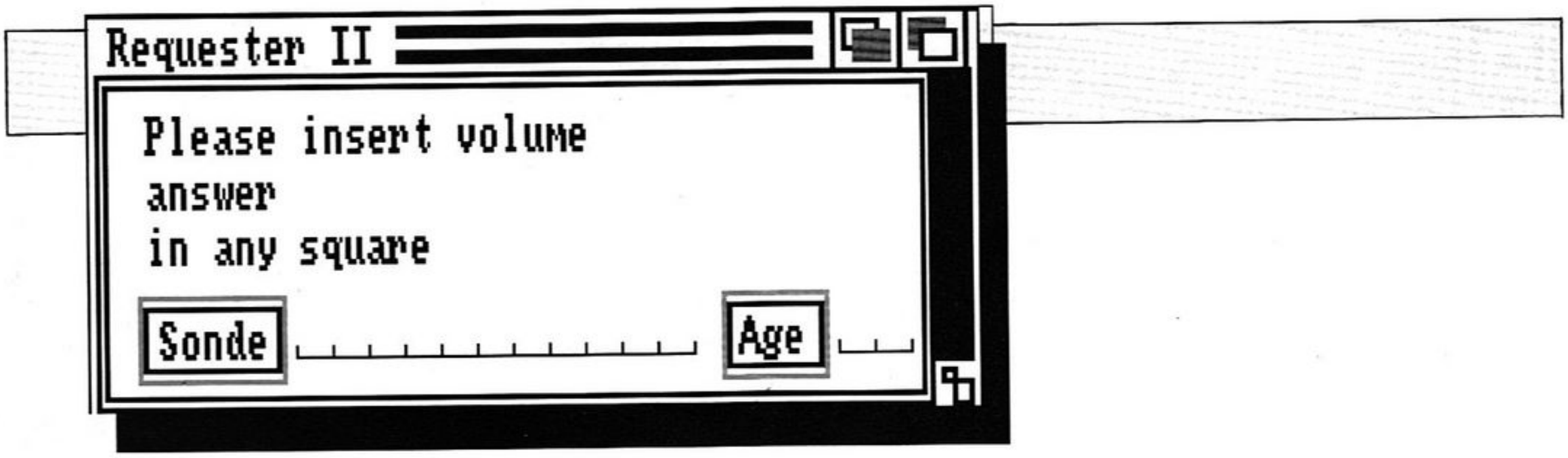
Cet énôôôôrme bug du Lattice 5.04 n'a pas été corrigé dans le 5.10

```
  TimeSigMask = (1L << timedevice_port->mp_SigBit);

  if((InputSig = AllocSignal(-1L)) == -1) Clean_Exit();
  InputSigMask = 1L << InputSig;

  if((PermutSig = AllocSignal(-1L)) == -1) Clean_Exit();
  PermutSigMask = 1L << PermutSig;

  if((InitSig = AllocSignal(-1L)) == -1) Clean_Exit();
  InitSigMask = 1L << InitSig;

  task = (struct Task *)FindTask(NULL);

/*****************************************************************************
*               Installation du Handler                 *
*****************************************************/

  if(!(InputHandler =(struct Interrupt *)
  AllocMem(sizeof(struct Interrupt),MEMF_PUBLIC|MEMF_CLEAR)))
Clean_Exit();

InputHandler->is_Code=(VOID (*)())MyHandler;
  InputHandler->is_Data=NULL;
  InputHandler->is_Node.ln_Pri=51;
  InputHandler->is_Node.ln_Name="MyInputHandler";
  inputreq->io_Data=(APTR)InputHandler;
  inputreq->io_Command=IND_ADDHANDLER;
  DoIO((struct IORequest *)inputreq);

LaunchTimer(timereq,TICK);

/*****************************************************
*                                                    *
*   l'éternité c'est long ... surtout vers la fin    *
*                                                    *
*****************************************************/

  while(task)
  {

    Sig = Wait(TimeSigMask|InputSigMask|PermutSigMask|InitSigMask);

    if ((Sig & InputSigMask) && (BlankScreen == NULL)) Time_Count = NULL;

    if ((Sig & InputSigMask) && (BlankScreen != NULL))
    {
        CloseScreen(BlankScreen);
       ON_DISPLAY;
        BlankScreen = NULL;
        Time_Count = NULL;
    }

    if (Sig & InitSigMask)
    {
        if (BlankScreen != NULL)
        {
          CloseScreen(BlankScreen);
          BlankScreen = NULL;
         ON_DISPLAY
        }
        WBenchToFront();
        Time_Limite = InitLimite();
       Time_Limite *= 12;
        Time_Count = NULL;
    }

    if (Sig & TimeSigMask)
     {

       if (BlankScreen == NULL)
     {
          Time_Count++;
          if (Time_Count > Time_Limite)
          {
            BlankScreen = (struct Screen *)OpenScreen(&NewScreen);
            if (BlankScreen != NULL)
            {

SetRGB4(&(BlankScreen->ViewPort),0,0,0,0);
             OFF_DISPLAY
            }
          }
       }
        LaunchTimer(timereq,TICK);
     }

    if (Sig & PermutSigMask)
    {
```

```
      if (BlankScreen)
      {
        CloseScreen(BlankScreen);
        BlankScreen = NULL;
       ON_DISPLAY
      }
     Time_Count = NULL;
      ScreenToBack(IntuitionBase->FirstScreen);
    }
  }
}

*************************************************************************
*                          FONCTIONS                                    *
*************************************************************************

VOID Clean_Exit()
{
  if (timereq != NULL)
  {
    if (timereq->tr_node.io_Device != NULL)
    {
      CloseDevice((struct IORequest *)timereq);

    }
  DeleteExtIO((struct IORequest *)timereq);
    }

  if (inputreq != NULL)
  {
    if (inputreq->io_Device !=NULL)
{
       inputreq->io_Command = IND_REMHANDLER;
      inputreq->io_Data= (APTR)InputHandler;
      DoIO((struct IORequest *)inputreq);
      CloseDevice((struct IORequest *)inputreq);
    }
  DeleteExtIO((struct IORequest *)inputreq);
}

  if (InputHandler) FreeMem(InputHandler,sizeof(structInterrupt));
  if (InitSig != -1) FreeSignal(InitSig);
  if (PermutSig != -1) FreeSignal(PermutSig);
  if (InputSig != -1) FreeSignal(InputSig);
  if (timedevice_port)  DeletePort(timedevice_port);
  if (inputdevice_port) DeletePort(inputdevice_port);
  if (GfxBase) CloseLibrary((struct Library *)GfxBase);
  if (IntuitionBase) CloseLibrary((struct Library *)IntuitionBase);
  if ((port) && (!Active)) DeletePort(port);
  exit(0);
}

VOID LaunchTimer(struct timerequest *tr,LONG secondes)
{
  tr->tr_node.io_Command = TR_ADDREQUEST;
  tr->tr_time.tv_secs = secondes;
  tr->tr_time.tv_micro = 0;
  SendIO((struct IORequest *)tr);
}

LONG InitLimite()
{
struct Window *MyWindow;
LONG nb_minute = NULL;
char dummy_string[2];

if ((MyWindow = (struct Window *)OpenWindow(&Reglage)) == NULL)

Clean_Exit();

Move(MyWindow->RPort,10,10);
Text(MyWindow->RPort,text_reglage[0],strlen(text_reglage[0]));
Move(MyWindow->RPort,10,20);
Text(MyWindow->RPort,text_reglage[1],strlen(text_reglage[1]));
Move(MyWindow->RPort,10,40);
Text(MyWindow->RPort,text_reglage[2],strlen(text_reglage[2]));

do
{
     strcpy(dummy_string,"+0");
     ActivateGadget(&MyGadget,MyWindow,0);
Wait( 1L << MyWindow->UserPort->mp_SigBit);

if (Buffer[0] =='x' | Buffer[0] =='X')
{
CloseWindow(MyWindow);
Clean_Exit();
}

     dummy_string[1] = Buffer[0];
nb_minute = atoi(dummy_string);

}while((Buffer[0] > '9')  | (Buffer[0] < '1'));

CloseWindow(MyWindow);

   return(nb_minute);
}
```

Programme : Blankerhandler.s
Fonction : routine assembleur pour le programme CRBlanker
Assembleur : Devpac 2
Assemblage : produire un code linkable sous le nom blankerhandler.o
Auteur : Frédéric Mazué pour ANT le 18/11/90

```
include "exec/exec_lib.i"
include "exec/io.i"
include "devices/inputevent.i"

XREF _task
XREF _InputSigMask
XREF _PermutSigMask
XREF _InitSigMask

XDEF _MyHandler

_MyHandler

move.l a0,-(sp)                 ;empiler chaîne d'événement

Loop
move.b ie_Class(a0),d1
move.l _InputSigMask,d0
cmp.b #IECLASS_RAWMOUSE,d1      ;événement souris ?
bne no_mouse
bra envoie_signal               ;oui -> envoi signal

no_mouse
cmp.b #IECLASS_RAWKEY,d1
bne Next

move.w ie_Code(a0),d1
cmp.w #$5f,d1                   ;touche HELP pressée
bne envoie_signal               ;non -> envoi signal clavier simple

move.w ie_Qualifier(a0),d1
btst #IEQUALIFIERB_CONTROL,d1 ;touche contrôle pressée ?
beq suite
move.l _InitSigMask,d0
bra envoie_signal               ;envoi signal pour
                                ;ouvrir fenêtre de réglage
suite
btst #IEQUALIFIERB_LCOMMAND,d1;touche amiga gauche pressée ?
beq envoie_signal               ;non -> envoi signal clavier simple

move.l _PermutSigMask,d0        ;pour envoi signal
                                ;permutation d'écran

envoie_signal
move.l a0,-(sp)
move.l _task,a1
CALLEXEC Signal
move.l (sp)+,a0

Next
move.l (a0),d0                  ;pour positionner flag Z
move.l d0,a0
bne Loop                        ;prochain événement
move.l (sp)+,d0                 ;récupérer liste événement dans D0
rts
```



Hé oui, déjà un sondage, dès le premier numéro ! Comprenez-nous : il faut bien que l'on sache avec exactitude ce que vous vous attendez à trouver dans l'ANT, pour que l'on puisse vous offrir le meilleur journal de programmation sur Amiga de la galaxie (banlieue comprise). Faites un petit effort : un tirage au sort parmi toutes les réponses reçues attribuera à trois gagnants, 5 heures de connection chacun en 3614 sur le futur serveur minitel de l'ANT (ouverture le 16 mars 1991).

1 . Classez les rubriques suivantes selon votre ordre de préférence de 1 à 12:

News ..... :
Langages ..... :
L'invité du mois ..... :
Tool Box ..... :
ExecBase..... :
Sub Way ..... :
Concours listing permanent .... :
Le Coin des DemoMakers ..... :
L'utilitare du mois ..... :
Forum (dès le mois prochain !)
Requester ..... :
Le Club Dorotohée ..... :

2. - Quelle(s) rubrique(s) vous semble(nt) trop importante(s) ou carrément de trop, et pourquoi ? .....
.....
.....
.....
.....
.....
.....
.....

3. Quelle rubrique aimeriez-vous voir se développer ou apparaître dans l'ANT ? .....
.....
.....
.....
.....

4. Sachant qu'un train A part de la Gare de Lyon à 5 h 45 et qu'un avion B décolle de l'aéroport d'Orly à 18 h 12, quel est l'âge du capitaine C ?
.....
.....

5. Denise, la fourmi-mascotte de l'ANT vous paraît :

Rigolotte ..... :
Superflue..... :
Pourquoi une fourmi ? ..... :
Hein ? ..... :

6. Ce sondage est nul, vous n'avez pas pu dire tout ce que vous aviez sur le coeur, alors profitez-en : .....
.....
.....
.....
.....
.....
.....
.....

7. Renseignements personnels

Nom :.....
Prénoms : .....
Adresse : .....Code Postal : .....
Ville : ..... Pays : .....
Force : ..... Intelligence : ..... Sagesse : ..... Points de vie : .....

8. J'ai choisi la formule d'abonnement suivante :

5 mois sans la disquette ☐
5 mois avec la disquette ☐
11 mois sans la disquette ☐
11 mois avec la disquette ☐
Je le lis chez un abonné et je copie sa disquette. ☐

et j'explique les raisons de mon choix : .....
.....
.....
.....

9. J'ai un avis précis sur le contenu de votre disquette, et je ne vais pas me gêner pour vous le donner : .....
.....
.....
.....
.....

10. Et enfin, je possède :

Unité centrale : .....
Mémoire : .....
Périphériques : .....
.....
.....
Je programme en : .....

Voilà, ça n'était pas bien sorcier ? Il ne vous reste plus qu'à découper ou à photocopier (merci de ne pas recopier à la main) ce sondage et à nous l'expédier le plus rapidement possible à :

Amiga News Tech
5, rue de la Fidélité
75010 Paris

l'Amiga et ses écrans

En tant qu'ordinateur multitache, l'Amiga possède une gestion d'écran assez particulière, mais ô combien efficace. Qui n'a jamais été amusé ou surpris, la première fois, de faire défiler l'écran du Workbench du bout de sa souris ?

Cette série d'articles a pour but de vous faire mieux connaître deux des bibliothèques (libraries dans la langue de Shakespeare) de l'Amiga, à savoir Graphics et Intuition.

La graphics.library contient toutes les fonctions de gestion des objets graphiques de base (écrans, sprites, bobs) ainsi que les primitives de dessin (points, droites, cercles, remplissage...). L'intuition.library quant à elle est une plate-forme permettant le développement d'interfaces conviviales à base de menus, gadgets variés et autres fenêtres.

Cette étude sera formée d'une partie théorique et d'une série d'exemples, pour la plupart en langage C. Une connaissance du C (ou de tout autre langage structuré) sera donc très utile, ainsi parfois que des rudiments d'assembleur.

Enfin, un certain nombre de termes anglo-saxons difficilement traduisibles en français seront utilisés tout au long de cet article. En cas d'incompréhension, reportez-vous au lexique de fin.

LES CONCEPTS DE BASE

Lorsque l'on crée une zone de visualisation, il faut prendre en compte deux éléments essentiels, qui sont le playfield et les sprites. Le playfield représente la partie statique de la zone de visualisation, alors que les sprites peuvent se déplacer et interférer entre eux ou avec le playfield. Nous étudierons les sprites en détails dans un prochain article.

Pour projeter son affichage, L'Amiga utilise une technique de balayage vidéo connue sous le nom de "Raster Display". Cette image est formée d'un certain nombre de lignes horizontales, créées par le faisceau électronique du moniteur. En mode normal, c'est-à-dire non entrelacé, il est possible d'afficher 256 lignes horizontales (200 en Mode NTSC). En mode entrelacé, on peut afficher 512 lignes (400 en NTSC). Comment est-ce possible ? C'est assez simple : pour afficher 512 lignes, l'Amiga commence par afficher en 1/60éme de seconde une première série de 256 lignes représentant les lignes paires de l'image à afficher, puis il affiche les 256 lignes impaires après avoir décalé la position verticale de départ d'une demie épaisseur de ligne. Il existe toutefois un inconvénient à ce mode : pour tracer une image en mode entrelacé, l'Amiga met deux fois plus de temps, soit 1/30ème de seconde. Or ce délai est trop long pour obtenir un affichage de qualité visuelle standard, ce qui explique le scintillement, si disgracieux, du mode entrelacé.

Chaque ligne visualisée est composée d'un certain nombre de points. Deux modes de précision horizontale sont également proposés. En basse résolution, chaque ligne compte 320 points, alors qu'en haute résolution, il est possible d'avoir 640 points par ligne. L'affichage en haute résolution ne permet plus que d'obtenir 16 couleurs à l'écran alors que nous en avions 32 en basse résolution. A noter la présence des modes EHB et HAM qui permettent respectivement d'obtenir 64 et 4096 couleurs simultanément à l'écran avec toutefois quelques contraintes, mais nous y reviendrons plus tard.

Pour dessiner à l'écran, il est nécessaire d'écrire dans une portion de mémoire, portion qui représente la zone de visualisation qui va être affichée sur le moniteur. Cette zone de mémoire est organisée sous

```
/* ------------------------------------------------------------------------*/
/*     Ouverture d'un écran sous graphics...                               */
/*     Auteur Pascal AMIABLE (c) 1991                                      */
/* ------------------------------------------------------------------------*/

#include "exec/types.h"
#include "graphics/gfx.h"
#include "graphics/rastport.h"
#include "graphics/copper.h"
#include "graphics/view.h"
#include "graphics/gels.h"
#include "graphics/regions.h"
#include "graphics/clip.h"
#include "exec/exec.h"
#include "graphics/text.h"
#include "graphics/gfxbase.h"
#include "hardware/dmabits.h"
#include "hardware/custom.h"
#include "hardware/blit.h"

#define PLAN 2   /* Nombre de Bitplane associés à l'écran */
#define COLOR 4    /* Nombre de couleurs disponibles */
#define HORIZONT 640  /* Largeur de l'écran en pixels */
#define VERT 512  /* Hauteur de l'écran en lignes */
#define MODE_V  LACE  /* Mode de visualisation du View (entrelacé) */
#define MODE_VP HIRES+LACE   /* Mode de visualisation du ViewPort */
       /* Haute résolution et entrelacé */
int i;

struct View view;    /* View associé à l'écran */
struct ViewPort viewport;     /* ViewPort associé au View */
struct RasInfo rasinfo;       /* Structure RasInfo liée au ViewPort */
struct BitMap bitmap; /* BitMap liée au ViewPort */
struct RastPort rastport;     /* Le RasPort avec lequel on dessine */

struct GfxBase *GfxBase;      /* Pointeur sur la graphics library */

struct View *ecran_sauvegarde;   /* pointeur sur une structure View afin */
         /* de sauvegarder l'écran courant        */

UWORD colortable[COLOR] = { 0x000,0xf00,0x0f0,0x00f };
         /* Table des couleurs, couleur0 = NOIR */
         /* couleur1 = ROUGE, couleur2 = VERT   */
         /* couleur3 = BLEU                     */

char *bouton_gauche = (char *) 0xbfe001;/* Port A du CIA dont le bit */
       /* correspond au bouton gauche de la souris  */

void init(),ouvrecran(),libere();


/* ------------------------------------------------------------------------*/
/*             Programme principal                                         */
/* ------------------------------------------------------------------------*/

void main()
{
  init();
  ouvrecran();
  Move(&rastport,100,100);
  Draw(&rastport,200,200);
  while(!((*bouton_gauche & 0x40)-64)); /* tant que le bouton  */
    /* gauche de la souris n'est pas enfoncé */
  LoadView(ecran_sauvegarde); /* restauration du View sauvegardé */
  libere(); /* libération des ressources utilisées */
}

/* ------------------------------------------------------------------------*/
/*     Init() Ouvre la bibliothèque et sauvegarde l'ancien écran           */
/* ------------------------------------------------------------------------*/
void init()
{
  if((GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",0)) ==
```

forme de "bitplanes". Un bitplane représente une zone virtuelle de traçage, permettant d'afficher une seule couleur (mise à zéro ou à un de chaque bit de la zone). L'affichage avec plusieurs couleurs est possible en superposant plusieurs bitplanes. On obtient alors 2^N couleurs, où N représente le nombre de bitplanes. Cette gestion des couleurs par superposition de bitplanes est réalisée grâce à un circuit spécialisé portant le doux nom de Denise. Un maximum de 6 bitplanes superposés est autorisé, soit dans l'absolu 2^6 couleurs (64). Cette assemblage de bitplanes s'appelle une bitmap (cf Fig 1).

STRUCTURES ET FONCTIONS D'AFFICHAGE

L'Amiga produit une zone d'affichage à partir d'une série d'instructions organisée sous la forme d'une structure C et baptisée "View". Cette zone d'affichage peut être décomposée en plusieurs parties distinctes, séparées par au minimum une ligne horizontale (une ligne de "raster") : ce sont les "ViewPorts", qui forment donc des zones rectangulaires (cf. Fig 2). Le ViewPort correspond au CustomScreen d'Intuition, que nous étudierons plus tard.

Pour définir un ViewPort, il faut dans un premier temps initialiser quelques paramètres : sa taille (hauteur et largeur), sa profondeur (le nombre de bitplanes utilisés) qui indique le nombre de couleurs, son mode de visualisation (basse ou haute résolution, interlacement...), l'adresse de la zone de mémoire utilisée pour l'affichage, et enfin sa position par rapport au coin haut et gauche de l'écran.

La hauteur est associée au champ DHeight de la structure ViewPort. Elle représente le nombre de lignes verticales du ViewPort.

La largeur est contenue dans le champ DWidth. Elle représente le nombre de pixels (points) par lignes du ViewPort. Il est possible de définir pour chaque ViewPort des largeurs différentes.

Le nombre de Bitplanes utilisés pour la mémoire d'affichage, conditionne le nombre de couleurs affichables dans le ViewPort sous la forme déjà citée : nombre_couleurs = 2^nombre_bitplanes. A noter le cas particulier du mode HAM qui autorise avec une profondeur de 6 bitplanes, l'affichage de 4096 couleurs, avec toutefois des contraintes de proximité. La profondeur est codée dans le champ utilisé Depth.

A chaque ViewPort est associée une palette de couleurs. Cette palette est spécifiée dans une structure baptisée ColorMap et rattachée au ViewPort par son adresse. Nous verrons cela un peu plus loin.

Il existe 9 modes possibles de visualisation, le mode par défaut étant la basse résolution. Pour choisir un autre mode il faut remplir le champ Modes avec les valeurs adéquates, suivant la liste ci-dessous :

- basse résolution (NULL) : dans ce mode, une ligne normale

```
NULL)
    exit(1); /* La bibliothèque graphics ne veut pas s'ouvrir */

  ecran_sauvegarde = GfxBase->ActiView; /* sauvegarde du View actif */
}

/* ------------------------------------------------------------------*/
/*     ouvrecran() initialise et ouvre un écran                      */
/* ------------------------------------------------------------------*/
void ouvrecran()

{
  InitView(&view); /* Initialisation de la structure View */
  InitVPort(&viewport); /* Initialisation de la structure ViewPort */
  view.ViewPort = &viewport; /* On lie le ViewPort au View */

  view.Modes = MODE_V; /* On indique le mode visualisation du View */

  InitBitMap(&bitmap,PLAN,HORIZONT,VERT); /* On initialise la bitmap */

  rasinfo.BitMap = &bitmap; /* liaison de la Bitmap avec le RasInfo */
  rasinfo.RxOffset = 0;
  rasinfo.RyOffset = 0;
  rasinfo.Next = NULL; /* pas d'autre structure */

  viewport.DxOffset = 0; /* On remplit la structure ViewPort */
  viewport.DyOffset = 0; /* cf l'article ci-dessus */
  viewport.DWidth = HORIZONT;
  viewport.DHeight = VERT;
  viewport.RasInfo = &rasinfo; /* liaison entre le ViewPort et */
        /* le RasInfo                          */
  viewport.Modes = MODE_VP; /* mode de visualisation du ViewPort */
  viewport.Next = NULL; /* Pas d'autre ViewPort */
  viewport.ColorMap = (struct ColorMap *)GetColorMap(COLOR);
/* Initialisation de la ColorMap du ViewPort */
  LoadRGB4(&viewport,&colortable[0],COLOR);
/* Chargement des couleurs dans la ColorMap */

  for ( i=0;i<PLAN;i++) /* Allocation des bitplanes */
  { if ((bitmap.Planes[i] = (PLANEPTR)AllocRaster(HORIZONT,VERT)) == NULL)
      exit(2);
    BltClear ((UBYTE *)bitmap.Planes[i],RASSIZE(HORIZONT,VERT),0);
      /* effacement de bitplanes au Blitter */
  }

  InitRastPort(&rastport); /* initalisation du RastPort */
  rastport.BitMap = &bitmap; /* on lie le RastPort à la */
    /* bitmap dans laquelle on dessine            */

  MakeVPort(&view,&viewport); /* On crée la zone de visualisation */

  MrgCop(&view); /* et la CopperList associée */
  LoadView(&view); /* on affiche l'écran sur le moniteur */
}

/* ------------------------------------------------------------------*/
/*     libere() libere la mémoire utilisée                           */
/* ------------------------------------------------------------------*/
void libere()
{
  for(i = 0;i<PLAN;i++) /* on libère la mémoire des bitplanes */
    FreeRaster(bitmap.Planes[i],HORIZONT,VERT);

  FreeColorMap(viewport.ColorMap);   /* on libère la mémoire de la */
          /* ColorMap                    */
  FreeVPortCopLists(&viewport);      /* libération de la CopperList*/

  CloseLibrary(GfxBase);  /* Fermeture de la bibliothèque graphics */
}
```

Overscan :
- haute résolution (HIRES) : la résolution horizontale passe ici à 640 pixels, ou 704 en Overscan ;
- entrelacé (LACE) : l'affichage s'effectue en mode entrelacé, soit 512 lignes par écran au lieu de 256. Attention ! Lorsque vous spécifiez le mode LACE dans le ViewPort, vous devez impérativement le spécifier également dans le champ View.Modes.
- Hold & Modify (HAM) : mode spécial permettant l'affichage de 4096 couleurs simultanées à l'écran ;
- double playfield (DUALPF) : le ViewPort est traité comme possédant deux zones d'affichage superposées et indépendantes ;
- avec Sprites (SPRITES) : cette valeur est obligatoire pour indiquer à l'Amiga que l'on va utiliser des sprites hardwares ;
- playfield 2 avant 1 (PFBA) : cette valeur est liée au mode DUALPF et permet de positionner le playfield 2 devant le premier ;
- ViewPort caché (VP_HIDE) : indique que le ViewPort est caché et par là même, que son contenu n'est pas visualisé à l'écran ;
- mode 64 couleurs (EXTRA_HALFBRITE) : dans ce mode, 64 couleurs sont affichées, les 32 dernières étant dérivées des 32 premières.

Pour obtenir, par exemple, un écran en haute résolution entrelacée, on mettra HIRES+LACE dans le champ Modes.

Comme le Viewport est de taille égale ou inférieure à la zone de visualisation, il est nécessaire d'indiquer sa position à l'écran. Cette position est définie par les distances DxOffset et DyOffset entre le coin supérieur gauche de l'écran et le coin supérieur gauche du ViewPort (cf. Fig 3). DxOffset doit être compris entre -16 et +352 (-32 et +704 en HIRES), DyOffset entre -16 et +256 (-32 et +512 en mode LACE) sachant que (0,0) positionne le ViewPort strictement dans le coin Haut et Gauche de l'écran. DxOffset et DyOffset sont deux champs de la structure ViewPort.

La zone de mémoire réservée pour l'affichage peut être plus grande que l'écran de visualisation et ce jusqu'à la taille limite de 1024x1024 points. C'est justement pour cela qu'il faut spécifier la position du ViewPort dans cette zone d'affichage, suivant le même principe que ci-dessus et à l'aide des champs RxOffset et RyOffset (cf. Fig 4). Ces deux valeurs sont stockées dans les champs de la structure RasInfo portant le même nom. Cette structure contient également un pointeur sur la Bitmap d'affichage (structure BitMap).

Nous en savons maintenant assez pour créer notre propre écran. L'exemple ci-joint va nous permettre de passer en douceur de la théorie à la pratique, en créant un simple ViewPort d'une taille de 320*200 avec une bitmap de même taille. Il démontre également la manière de placer ce ViewPort par rapport à la structure View. Le mois prochain, nous verrons la génération de playfields avec Intuition, ce qui nous permettra d'étudier les différences entre les deux méthodes.

Herr Doktor Von GlutenStimmelImDorf.

Fig. 1 - Organisation de l'affichage sur Amiga.
Fig. 2 - Décomposition d'une zone d'affivchage.
Fig. 3 - Position du ViewPort à l'écran.
Fig. 4 - Position du ViewPort par rapport à la BitMap.

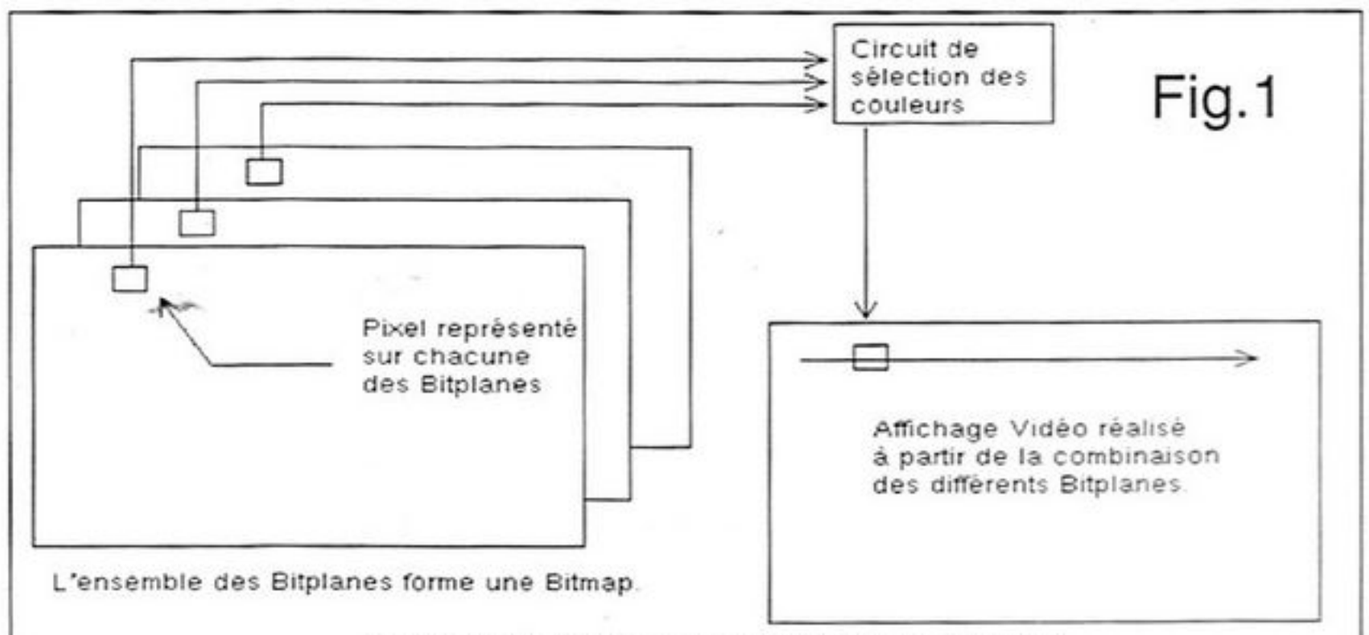


Fig.1

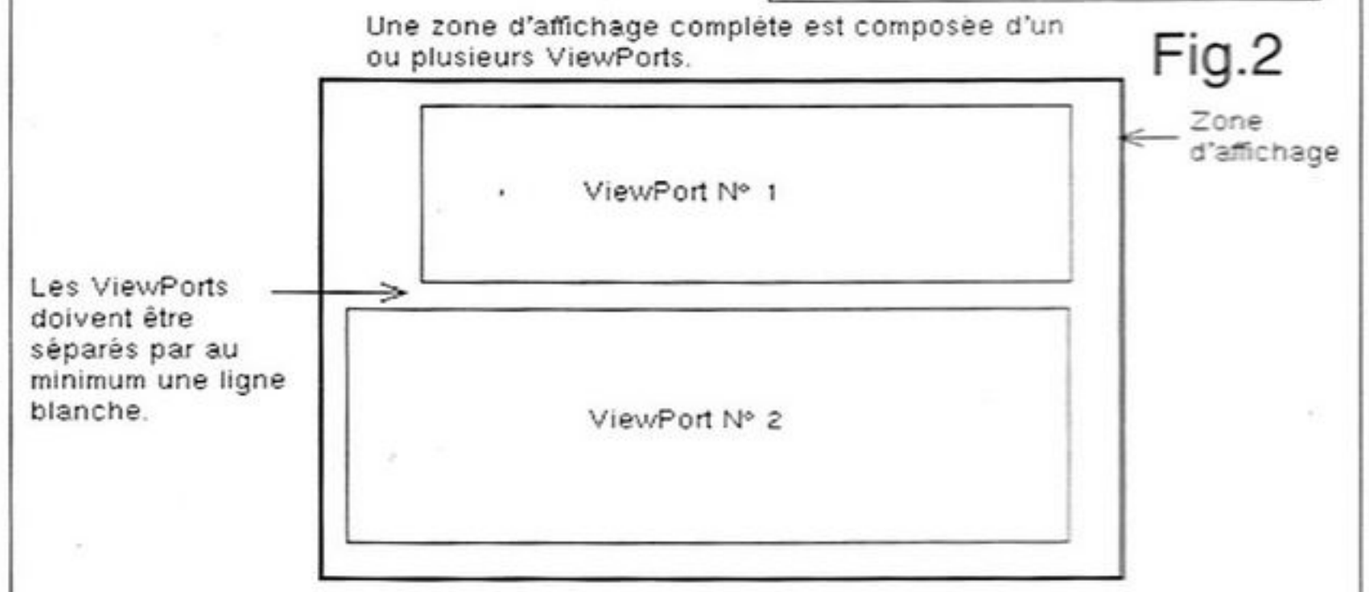


Fig.2

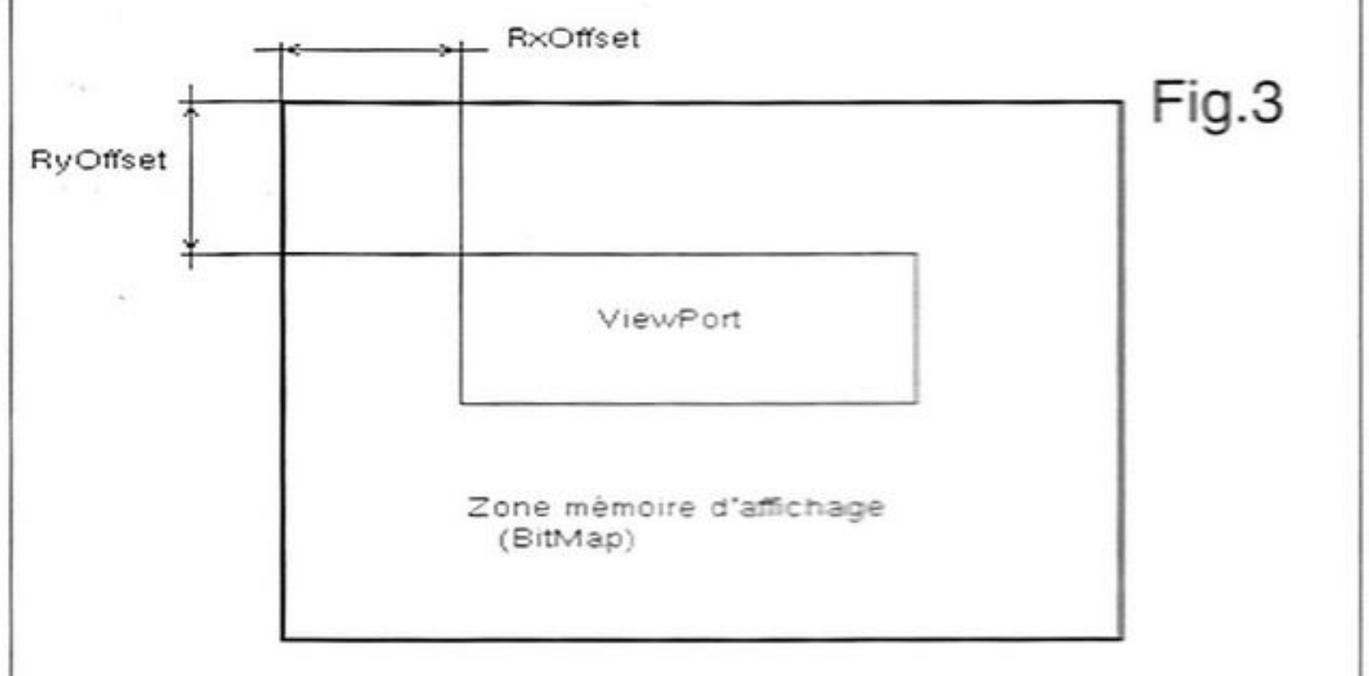


Fig.3

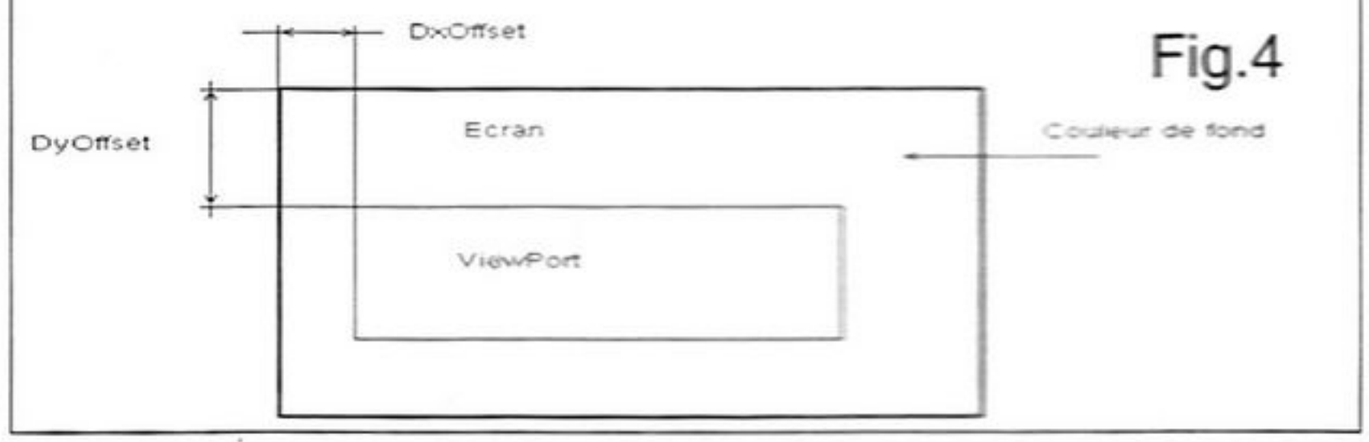


Fig.4

PETIT LEXIQUE

Bitmap : assemblage de bitplanes, permettant de créer un écran pour le traçage en multi-couleurs.

Bitplane : littéralement, plan de bits. Il s'agit d'une zone de mémoire organisée sous forme rectangulaire, permettant de représenter un écran de traçage virtuel.

Bob : alors que le sprite est indépendant du playfield, le bob en fait partie intégrante. Ce qui signifie que l'on doit gérer son interaction avec le playfield. C'est le principe des brosses dans un logiciel de dessin.

Copper : processeur spécialisé de l'Amiga. Synchronisé sur le spot vidéo, il a été conçu pour gérer la partie affichage de l'Amiga grâce à son jeu d'instructions propres.

Library : en français, bibliothèque. Ce terme est utilisé pour désigner les bibliothèques de fonctions du système d'exploitation de l'Amiga, qu'elles soient en ROM ou sur disquette (ex: "graphics.library").

Sprite : lutin en français. Il représente un objet graphique indépendant du playfield, qui peut se déplacer en superposition de celui-ci sans en modifier le contenu (idéal pour représenter, par exemple, un petit vaisseau se déplaçant sur un fond

free men. 2417

ANTDos

Beaucoup de lecteurs nous écrivent afin de connaître l'adresse du club informatique le plus proche de leur domicile. Nous ne répondons pas à ces demandes car nous ne connaissons pas les adresses de tous les clubs informatiques de France et de Navarre. Nous vous suggérons donc de vous renseigner auprès du syndicat d'initative de votre ville. Quant aux clubs qui désirent se faire connaître du grand-public, qu'ils n'hésitent surtout pas à nous écrire.

L'idée de vendre l'ANT séparément de Commodore Revue est une bonne idée, pourtant je déplore deux points : premièrement, le fait que l'on soit obligé de s'abonner pour le recevoir et deuxièmement, le fait qu'il coûte le même prix que Commodore Revue (...).
Voici en fait l'objet principal de ma lettre. Ayant acheté "le Livre du Langage-Machine", je fis quelque chose déconseillé par celui-ci, c'est-à-dire lire l'adresse $DFF00A en faisant q$DFF00A avec SEKA. Je sais bien que la curiosité est un vilain défaut, mais je n'ai pas pu m'empêcher de le faire, et depuis, chaque fois que j'enlève la souris du port 1, le personnage contrôlé par l'intermédiaire du port 1 (avec le joystick) est irrémédiablement dirigé vers le haut, et ce pour tous les jeux. (FM : notre lecteur de nous raconter dans le détail et avec désespoir tous les déboires auxquels il doit faire face).

Roger Humbert (Firminy)

Bien sûr, nous aurions nous aussi préféré une diffusion en kiosque pour l'ANT... Mais il faut être réaliste : le mondre de la programmation sur Amiga n'est pas encore suffisamment développé pour permettre la création d'un journal de 200 pages, tout en couleurs, que l'on vendrait 10 francs à 50.000 exemplaires. Si nous avons choisi la formule de l'abonnement, c'est en espérant bien que tôt ou tard, nous pourrons arriver à une diffusion nationale en kiosque. Quant au prix, nous essayons de le compenser par des cadeaux et des offres promotionnelles incomparables. Vous n'avez pas encore tout vu !
Je vous rassure tout de suite, votre curiosité n'a pas détruit votre ordinateur. Si le simple fait de lire une adresse pouvait casser une machine, où irions nous ? Les bons conseils de ce livre sont bien entendu erronés, mais après tout, pourquoi y aurait-il moins de bêtises ici que dans la Bible (NLDR : ceci est une opinion personelle de F. Mazué qui n'engage que lui) ? Il est parfaitement possible d'accéder auxdites adresses... C'est en fait K-SEKA qui en est incapable et qui plante lamentablement, comme vous pourrez vous en rendre compte en utilisant un autre utilitaire (comme MonAm2 de Devpac, tout-à-fait au hasard). Mais je l'ai toujours dit, K-SEKA c'est caca (il y a tout de même des noms prédestinés !).
Pour ce qui est de votre panne, peut-être faut-il faire réparer votre machine, mais il est probable qu'un simple dépoussiérage résoudra votre problème.

Enfin la bonne nouvelle de l'ANT. En même temps que ce courrier, j'envoie mon abonnement. Bravo !
Par contre, je suis déçu de ne pas avoir eu de nouvelles de la lettre que j'avais destinée à Max, il y a deux mois. D'autant plus que j'ai même glissé une enveloppe timbrée pour la réponse. Je demandais tout simplement la marche à suivre pour faire tourner le "lecteur IFF", ce n'est pas grand chose, hein !? Sniff (NDMax : Sn-IFF ?) !
Comment fait-on avec l'Amiga pour programmer une fonction de nombres aléatoires ? Est-ce une routine dans une librairie ou doit-on utiliser les timers et le VHPOS par exemple, et faire quelques algorithmes savants nous-mêmes ?

Ce vieil habitué et impatient de Patrick Hurtrel (Vichy)

Salut Patrick. Ton enthousiasme nous fait toujours plaisir, mais il faut être patient et indulgent car :
- les réponses paraissent dans le Requester environ deux à trois mois après réception du courrier. En ce moment, tu lis le numéro de février, mais nous préparons déjà celui de mai ;
- nous ne pouvons hélas pas répondre à toutes les lettres. Il faut comprendre que nous ne manquons ni de timbres ni d'enveloppes, mais bel et bien de temps. Et crois bien, ainsi que vous tous les autres déçus, que nous en sommes désolés.
Tu as raison, pour programmer une fonction aléatoire, il faut utiliser les timers ou le VHPOS. Il n'y a rien dans les libraries, mais il est très intéressant de passer par le timer.device. Quant à l'algorythme savant pour obtenir une suite de nombres aléatoires, voila celui que j'utilise : c'est la relation de congruence

$$X[n+1] = (a * X[n] + c) \bmod m$$

Ainsi, on peut "tirer" un nombre à partir d'un précédent, sans continuité mathématique apparente, ce qui donne un bon aspect aléatoire. a et c sont des constantes choisi par le programmeur. m permet d'obtenir l'intervalle dans lequel se situeront les nombres. Exemple : avec m=7, on obtient des nombres entre 0 et 6. On filtre le 0 et ainsi on peut simuler le jet d'un dé. J'envisage de vous faire prochainement un article là-dessus.

Réponses fournies par Frédéric Mazué

SUB.way : Console-Device

Le port IDCMP (Intuition Direct Communication Message Port pour les ignares) de toute fenêtre Intuition est un moyen simple mais pas très efficace de recevoir des caractères en provenance du clavier et d'en écrire dans ladite fenêtre.

En effet, le flag RAWKEY ne renvoie que les codes clavier des touches, nous obligeant pour la conversion en ASCII à l'emploi de tables minimisant la compatibilité internationale du programme, et le flag VANILLAKEY renvoie bien le code ASCII, mais oublie au passage de prendre en compte les touches spéciales telles que F1-F10 ou encore Help. La seule solution lorsque l'on désire traiter la console avec tous les égards qu'elle mérite consiste donc à utiliser le device qui lui est dédié, autrement dit le console.device.

DEVICE ? VOUS AVEZ DIT DEVICE?

Avant tout, un bref rappel de ce qu'est un device semble s'imposer. Un device a la même structure qu'une librarie : il est composé d'un bloc de données et d'un bloc de sauts à des routines internes. Vous trouverez la définition de la structure Device dans le fichier 'exec/devices.h' pour les programmeurs en C, et 'exec/devices.i' pour les 68000eurs.

Pour communiquer avec un quelconque device, on utilise une autre structure définie dans 'exec/io.h' (resp. 'exec/io.i') :

```
struct IOStdReq
{ struct Message io_Message;   /* structure message*/
  struct Device  *io_Device;   /* réservé au device*/
  struct Unit *io_Unit;        /* réservé au device*/
  UWORD io_Command;            /* Commande à effectuer*/
  UBYTE io_Flags;              /* Flags particuliers*/
  BYTE  io_Error;              /* <>0 en cas d'erreur*/
  ULONG io_Actual;             /* nombre d'octets transférés*/
  ULONG io_Length;             /* ""    "" à transférer*/
  APTR  io_Data;               /* pointeur sur le buffer IO*/
  ULONG io_Offset;             /* offset particulier*/
};
```

Il est à noter que certains devices utilisent des structures IO étendues ; c'est le cas du trackdisk.device par exemple, qui utilise une structure nommée IOExtTD et définie dans 'devices/trackdisk.h'.

La structure Message est celle qui permet au device de communiquer avec la tâche qui l'a ouvert. La première étape à l'ouverture d'un device est donc la création d'un 'message port'. La fonction CreatePort() de l'Amiga.lib fait ça très bien pour nous :

```
struct MsgPort *messageport;
messageport=(struct MsgPort *)CreatePort("MonPort", 0);
```

Ce qui donne en assembleur :

```
XREF _CreatePort

  ...
  CLR.L -(A7)
  PEA NomDuPort(PC)
  JSR _CreatePort
  ...

NomDuPort:
```

```
/*************************************************
* Compilation (Lattice 4.0) :
*   lc -csuw -d -r -b Console.c
*
* Linkage :
*   FROM LIB:c.o,Console.o TO Console
*   LIBRARY LIB:lc.lib,LIB:amiga.lib
*   SMALLCODE
*   SMALLDATA
*   VERBOSE
*   NODEBUG
*************************************************/

#include <exec/types.h>
#include <exec/io.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <libraries/dos.h>
#include <devices/console.h>
#include <proto/all.h>

#ifdef LATTICE
int CXBRK(void) { return(0); }   /* Ciao CTRL-C*/
#endif

/*** Commandes ANSI du console.device */
/*** Note : 033 est la valeur octale de ESC (27 décimal) */
#define RESETCON"033c"
#define CURSOFF "033[0 p"
#define CURSON  "033[ p"
#define DELCHAR "033[P"

#define COLOR01 "033[31m"
#define COLOR02 "033[32m"
#define COLOR03 "033[33m"
#define ITALICS "033[3m"
#define BOLD   "033[1m"
#define UNDERLINE "033[4m"
#define NORMAL  "033[0m"

/*** Prototypes des fonctions */
void main(int, char **);
void cleanexit(UBYTE *,LONG);
void cleanup(void);
BYTE OpenConsole(struct IOStdReq *, struct IOStdReq *, struct Window *);
void CloseConsole(struct IOStdReq *);
void QueueRead(struct IOStdReq *, UBYTE *);
UBYTE ConGetChar(struct MsgPort *, UBYTE *);
LONG ConMayGetChar(struct MsgPort *, UBYTE *);
void ConPuts(struct IOStdReq *, UBYTE *);
void ConWrite(struct IOStdReq *, UBYTE *, LONG);
void ConPutChar(struct IOStdReq *, UBYTE);

/*** Variables globales */
struct NewWindow nw =
{ 10, 11, 620, 180,
  -1, -1,
  CLOSEWINDOW,
  WINDOWDEPTH|WINDOWDRAG|WINDOWSIZING|WINDOWCLOSE|
    SMART_REFRESH|ACTIVATE|RMBTRAP,
  NULL, NULL, "Console Test", NULL, NULL,
  100, 45, 640, 256,
  WBENCHSCREEN
};

struct Library  *IntuitionBase = NULL;
struct Window   *win = NULL;
struct IOStdReq *readReq = NULL, *writeReq = NULL;
struct MsgPort  *readPort = NULL, *writePort = NULL;
BOOL OpenedConsole = FALSE, FromWB;

/*** C'est parti ! ********************/
```

```
  dc.b "MonPort",0
  even
```

Il ne faudra pas oublier de détruire ce port une fois qu'on n'en aura plus besoin :

```
DeletePort(messageport);
```

La seconde étape consiste à initialiser correctement une structure IOStdReq - ou une structure IO étendue pour les devices en utilisant une - ce qui se fait au moyen de la fonction CreateExtIO() de l'amiga.lib :

```
#define EXTSIZE ((LONG)sizeof(struct IOStdReq))

struct IOStdReq *request, *CreateExtIO();
request=(struct IOStdReq *)CreateExtIO(messageport, EXTSIZE);
```

Cette structure devra elle aussi être éliminée lorsqu'on n'en aura plus besoin :

```
DeleteExtIO(request);
```

Ce n'est qu'alors que l'on pourra appeler la fonction OpenDevice() d'exec.library :

```
BYTE erreur;
erreur = OpenDevice("nom.du.device", unit, request, flags);
```

- "nom.du.device" décrit devinez quoi ? Le nom du device que l'on veut ouvrir, oui. Ce peut etre "trackdisk.device", "console.device", "keyboard.device", etc. ;
- "unit" désigne l'unité concernée. Par exemple, le trackdisk.device contrôle quatre unités, à savoir DF0: à DF3:. Le keyboard.device, lui, n'en contrôle qu'une (et pour cause : l'Amiga ne possède qu'un seul clavier !). Dans le cas du console.device, l'unité pourra être 0 ou -1 ;
- "request" est le pointeur sur la structure IOStdReq - ou bien, encore une fois, sur une structure IO étendue - créée plus tôt ;
- "flags" sera les trois quarts du temps mis à 0.

CONSOLE !

"Le terme "console" désigne en informatique le ou les périphérique(s) destiné(s) au dialogue homme/machine, généralement le clavier pour les entrées et l'écran pour les sorties" (Petit Robert, édition 91).

Le console.device permet donc la saisie de caractères depuis le clavier et leur édition à l'écran, plus précisémment dans une fenêtre Intuition - car n'oublions pas que toute fenêtre est créée et gérée par Intuition, même les CON:, RAW: et autres NEWCON:.

Un programme peut désirer ne prendre connaissance que des entrées et effectuer les sorties lui-même au travers de la graphics.library et de la fonction Text(). C'est le cas de ProWrite ou de GenAm2, l'éditeur du Devpac, par exemple. Dans ce cas, on ouvre l'unité -1. Mais il est également possible de laisser le console.device éditer le texte dans la fenêtre. Dans ce dernier cas, il faudra initialiser deux structures IOStdReq et deux message-ports, un pour chaque sens de la communication.

Pour lire le console.device :

```
struct MsgPort *readPort;
struct IOStdReq *readReq;

if ( (readPort = CreatePort("conRead.port", 0)) == NULL )
   clean_exit("Impossible de créer le port de lecture.");

if ( (readReq = CreateExtIO(readPort, EXTSIZE)) == NULL )
```

```
void main(argc, argv)
int argc;
char **argv;
{
struct IntuiMessage *winmsg;
ULONG signals, conreadsig, windowsig;
LONG lch;
SHORT InControl = 0;
BOOL Done = FALSE;
UBYTE ch, ibuf, obuf[200];
BYTE error;

  FromWB = ((argc == 0) ? TRUE : FALSE);

/*** Ouvre intuition.library */
  if (!(IntuitionBase=(struct Libray *)
    OpenLibrary("intuition.library", 0)))
    cleanexit("Intuition", RETURN_FAIL);

/*** Crée le WritePort et le WriteRequest */
  if (!(writePort=(struct MsgPort *)
    CreatePort("consoleWrite.port", 0)))
      cleanexit("write port", RETURN_FAIL);

  if (!(writeReq=(struct IOStdReq *)
    CreateExtIO(writePort, (LONG)sizeof(struct IOStdReq))))
      cleanexit("write request", RETURN_FAIL);

/*** Crée le ReadPort et le ReadReq */
  if (!(readPort=(struct MsgPort *)
    CreatePort("consoleRead.port", 0)))
      cleanexit("read port", RETURN_FAIL);

  if (!(readReq=(struct IOStdReq *)
    CreateExtIO(readPort, (LONG)sizeof(struct IOStdReq))))
      cleanexit("read request", RETURN_FAIL);

/*** Ouvre la fenêtre */
  if (!(win=(struct Window *)OpenWindow(&nw)))
    cleanexit("window", RETURN_FAIL);

/*** Attache la console à la fenêtre */
  if (error = OpenConsole(writeReq, readReq, win))
    cleanexit("console.device", RETURN_FAIL);
  else
    OpenedConsole = TRUE;

/*** On écrit quelques trucs */
  ConPuts(writeReq,"Texte normal !");

  sprintf(obuf, "%s%sCouleur 3, italiques", COLOR03, ITALICS);
  ConPuts(writeReq, obuf);

  ConPuts(writeReq, NORMAL);
  ConPuts(writeReq, "On va effacer cette astérisque : -*-");
  Delay(50);
  ConPuts(writeReq, "\b\b");
  Delay(50);
  ConPuts(writeReq, DELCHAR);
  Delay(50);

  QueueRead(readReq, &ibuf); /* envoie un request de lecture */

  ConPuts(writeReq, "Maintenant, on lit la console...");
  ConPuts(writeReq, "Fermez la fenêtre pour terminer.");

  conreadsig = 1 << readPort->mp_SigBit;
  windowsig = 1 << win->UserPort->mp_SigBit;

  while(!Done)
  { signals = Wait(conreadsig | windowsig);

    if (signals & conreadsig) /* Ca vient de la console */
    { if ((lch = ConMayGetChar(readPort, &ibuf)) != -1)
      { /* La routine ci-dessous filtre les   */
        /* caractères de contrôle qui peuvent */
        /* être tapés directement au clavier  */
        ch = lch;
        if (InControl == -1)
        { if (ch=='[') InControl=2;
          else InControl=-2;
        }
        if ((ch==0x9b) || (ch==0x1b))
        { InControl=(ch==0x1b) ? 1 : 2;
```

clean_exit("Impossible de créer le request de lecture.");

Pour écrire dans le console.device :

```
struct MsgPort *writePort;
struct IOStdReq *writeReq;

if ( (writePort = CreatePort("conWrite.port", 0)) == NULL )
  clean_exit("Impossible de créer le port d'écriture.");

if ( (writeReq = CreateExtIO(writePort, EXTSIZE)) == NULL )
  clean_exit("Impossible de créer le request d'écriture.");
```

On ouvre ensuite la fenêtre, puis le console.device lui-même :

```
if ( (win = OpenWindow(&newWindow)) == NULL )
  clean_exit("Impossible d'ouvrir la fenêtre.");

writeReq->io_Data = (APTR)win;
writeReq->io_Length = sizeof(struct Window);
if ( OpenDevice("console.device", 0L, writeReq, 0L) )
  clean_exit("Impossible d'ouvrir le console.device.");
```

Pour terminer, on relie les 2 requests ensemble sur le même device :

```
readReq->io_Device = writeReq->io_Device;
readReq->io_Unit  = writeReq->io_Unit;
```

Partant de là, on peut commencer la lecture le port 'readPort' et l'écriture sur le port 'writePort'.

OPERATIONS D'ENTREES/SORTIES

Quel que soit le device concerné, deux types d'échanges sont possibles : d'une part l'échange "synchrone" (le device ne rend la main qu'une fois sa mission accomplie) et d'autre part l'échange "asynchrone" (le device travaille parallèlement au programme, en multi-tâche). Cette dernière possibilité est particulièrement utile lorsque, par exemple, on veut animer une présentation à l'écran tout en chargeant des données depuis le disque. La tâche incombe toutefois au programme de se renseigner auprès du device s'il a terminé son travail.

Pour en revenir au console.device, il est préférable d'effectuer les sorties (c.à.d l'édition de caractères) en mode synchrone. Les d'entrées (lecture de caractère(s) en provenance du clavier) peuvent être faites en n'importe quel mode, suivant que l'on désire attendre un caractère ou simplement le lire "au vol". Dans ce dernier cas, il faudra déjà avoir lancé une requête asynchrone, de façon à pouvoir examiner le port. Les fonctions C suivantes peuvent être utilisées :

Lancement d'une requête asynchrone :

```
void QueueRead(readreq, whereto)
struct IOStdReq *readreq;
UBYTE *whereto;
{
  readreq->io_Command = CMD_READ;
  readreq->io_Data   = (APTR) whereto;
  readreq->io_Length = 1;
  SendIO(readreq);
}
```

Lecture "au vol" d'un caractère (sans attente ; retourne -1 si aucun caractère n'a été frappé) :

```
LONG ConMayGetChar(msgport, whereto)
struct MsgPort *msgport;
UBYTE *whereto;
```

```
      ConPuts(writeReq, "=== Control Seq ===");
    }

    if (((ch>=0x1f) && (ch<=0x7e)) || (ch>=0xA0))
      sprintf(obuf, "Reçu : hex %02x = %c", ch, ch);
    else
      sprintf(obuf, "Reçu : hex %02x", ch);
    ConPuts(writeReq, obuf);

    if ((InControl==3) && ((ch>=0x40) && (ch<=0x7e)))
    { InControl=-1;
    }

    if (InControl == 2) InControl=3;

    if (InControl < 0)
    { InControl = 0;
      ConPuts(writeReq, "=== End Control ===");
    }
  }
}

  if (signals && windowsig) /* Ca vient de la fenêtre */
  { while (winmsg = (struct IntuiMessage *)GetMsg(win->UserPort))
    { switch(winmsg->Class)
      { case CLOSEWINDOW :
          Done = TRUE;
          break;
        /* Il pourrait y en avoir d'autres, */
        /* comme par exemple MENUPICK       */
        default :
          break;
      }
    ReplyMsg((struct Message *)winmsg);
    }
  }
}

if (!(CheckIO(readReq)))
  AbortIO(readReq);
WaitIO(readReq);

cleanup();
exit(RETURN_OK);
}

void cleanexit(s, n)
UBYTE *s;
LONG n;
{
  if (*s & (!FromWB)) printf("Erreur d'ouverture : %s\n", s);
  cleanup();
  exit(n);
}

void cleanup()
{
  if (OpenedConsole)CloseConsole(writeReq);
  if (readReq)   DeleteExtIO(readReq);
  if (readPort)    DeletePort(readPort);
  if (writeReq)    DeleteExtIO(writeReq);
  if (writePort)   DeletePort(writePort);
  if (win)    CloseWindow(win);
  if (IntuitionBase)CloseLibrary(IntuitionBase);
}

BYTE OpenConsole(writereq, readreq, window)
struct IOStdReq *writereq;
struct IOStdReq *readreq;
struct Window *window;
{
BYTE error;

  writereq->io_Data = (APTR) window;
  writereq->io_Length = sizeof(struct Window);
  error = OpenDevice("console.device", 0, writereq, 0);
  readreq->io_Device = writereq->io_Device;
  readreq->io_Unit   = writereq->io_Unit;
  return(error);
}


void CloseConsole(writereq)
struct IOStdReq *writereq;
```

```
{
register temp;
struct IOStdReq *readreq;

  if (!(readreq = (struct IOStdReq *)GetMsg(msgport)))
    return(-1);

  temp = *whereto;
  QueueRead(readreq, whereto);
  return(temp);
}
```

Lecture d'un caractère (avec attente) :

```
UBYTE ConGetChar(msgport, whereto)
struct MsgPort *msgport;
UBYTE *whereto;
{
register temp;
struct IOStdReq *readreq;

  WaitPort(msgport);
  readreq = (struct IOStdReq *)GetMsg(msgport);
  temp = *whereto;
  QueueRead(readreq, whereto);
  return((UBYTE)temp);
}
```

Ecriture d'un caractère sur la console :

```
void ConPutc(writereq, c)
struct IOStdReq *writereq;
BYTE c;
{ writereq->io_Command = CMD_WRITE;
  writereq->io_Data   = (APTR)&c;
  writereq->io_Length = 1;
  DoIO(write_io);
}
```

Ecriture d'une chaine de caractères (terminée par '0') sur la console :

```
void ConPuts(writereq, string)
struct IOStdReq *writereq;
UBYTE *string;
{
  writereq->io_Command = CMD_WRITE;
  writereq->io_Data   = (APTR) string;
  writereq->io_Length = -1;
  DoIO(writereq);
}
```

Ecriture d'un buffer de X caractères sur la console :

```
void ConWrite(writereq, string, length)
struct IOStdReq *writereq;
UBYTE *string;
LONG length;
{
  writereq->io_Command = CMD_WRITE;
  writereq->io_Data   = (APTR) string;
  writereq->io_Length = length;
  DoIO(writereq);
}
```

A SUIVRE...

Voilà. La deuxième et dernière partie de cet article traitera des commandes particulières du console.device, mais en attendant, je vous propose un petit programme de démonstration très largement inspiré de l'exemple publié dans les RKM. A bientôt pour de nouvelles aventures.

```
{
  CloseDevice(writereq);
}


void ConPutChar(writereq, character)
struct IOStdReq *writereq;
UBYTE character;
{
  writereq->io_Command = CMD_WRITE;
  writereq->io_Data    = (APTR) &character;
  writereq->io_Length  = 1;
  DoIO(writereq);
}


void ConWrite(writereq, string, length)
struct IOStdReq *writereq;
UBYTE *string;
LONG length;
{
  writereq->io_Command = CMD_WRITE;
  writereq->io_Data    = (APTR) string;
  writereq->io_Length  = length;
  DoIO(writereq);
}


void ConPuts(writereq, string)
struct IOStdReq *writereq;
UBYTE *string;
{
  writereq->io_Command = CMD_WRITE;
  writereq->io_Data    = (APTR) string;
  writereq->io_Length  = -1;
  DoIO(writereq);
}


void QueueRead(readreq, whereto)
struct IOStdReq *readreq;
UBYTE *whereto;
{
  readreq->io_Command = CMD_READ;
  readreq->io_Data    = (APTR) whereto;
  readreq->io_Length  = 1;
  SendIO(readreq);
}

LONG ConMayGetChar(msgport, whereto)
struct MsgPort *msgport;
UBYTE *whereto;
{
register temp;
struct IOStdReq *readreq;

  if (!(readreq = (struct IOStdReq *)GetMsg(msgport)))
    return(-1);

  temp = *whereto;
  QueueRead(readreq, whereto);
  return(temp);
}


UBYTE ConGetChar(msgport, whereto)
struct MsgPort *msgport;
UBYTE *whereto;
{
register temp;
struct IOStdReq *readreq;

  WaitPort(msgport);
  readreq = (struct IOStdReq *)GetMsg(msgport);
  temp = *whereto;
  QueueRead(readreq, whereto);
  return((UBYTE)temp);
}
```

AMIGA NEWS-TECH : MODE D'EMPLOI

Magazine, sources et exécutables uniquement par abonnement...

Parmi les nouvelles rubriques de l'ANT, certaines vous ont peut-être intrigué, d'autres nécessitent votre collaboration pour vivre. L'occasion pour nous de les présenter, ainsi que celles que vous retrouverez dans nos prochains numéros.

Les news : hé oui, il en faut bien. Mais pas n'importe lesquelles, hein, seulement celles en rapport direct avec la programmation, cela va de soi. D'ailleurs, si vous avez connaissance de quelque nouveau produit intéressant à venir, n'hésitez surtout pas à en faire profiter les autres lecteurs.

Les langages : nous avons choisi de parler des plus couramment utilisés sur l'Amiga, à savoir : le Basic (Amos et GfA), le C, l'assembleur et... Arrexx.

L'invité du mois : il y a des noms qui reviennent souvent, surtout des américains ou des canadiens, mais aussi quelques français. Le but de cette rubrique est de vous faire découvrir les personnalités qui se cachent derrière ces noms, comme John Toebes (auteur entre autres du Lattice C 4.0), Fred Fish (père de la collection de DP qui porte son nom), Jean-Michel Forgeas (auteur de la série FLamTel), etc.

ToolBox : les cordonniers étant toujours les plus mal chaussés, les meilleurs programmeurs ne disposent que rarement de tous les outils adaptés à leur travail. Dans Tool Box, nous vous présentons les outils qu'il faut posséder pour développer efficacement.

ExecBase : cette rubrique-ci, à l'inverse de cette rubrique-là, se veut plus générale que les aures. C'est beaucoup plus la "philosophie" de l'Amiga, sa conception même qui y sont abordées, que des routines toutes faites et prêtes à l'emploi.

SUB.way : tiens, voilà que l'on parle du métro dans l'ANT, maintenant ? Que nenni, le titre "SUB.way" désigne ce que l'on pourrait appeler la "face cachée" de l'Amiga : "SUB" étant à prendre dans le sens de "procédure", cette rubrique parlera... de la programmation système de l'Amiga (libraries, devices, resources, multitâche, etc.).

Le concours : là, nous avons vraiment besoin de vous, puisque c'est vous qui alimentez de vos création nos pages centrales. Ce mois-ci, une mise en oeuvre de fontes vectorielles en GfABasic. Rendez-vous vite en page 16 pour plus de détails.

Le Coin-coin des Démo Makers : ben oui, il faut être honnête, l'Amiga n'est pas seulement une machine multitâche, c'est également une superbe console de jeux et de démos ! Cette rubrique est donc faite pour ceux qui préfèrent ce type de programmation, au grand désespoir de Frédéric Mazué ! N'hésitez surtout pas à nous contacter si vous pensez pouvoir y contribuer, même occasionnellement.

Utilitaire : tiens, quand on parle du loup... Vous retrouverez dans cette rubrique notre ami Mazué dans ce qu'il sait le mieux faire, c'est-à-dire des tas de petits programmes pour simplifier la vie des fainéants. Indispensable !

Le Forum : c'est une véritable tribune d'échange que ce Forum, réservée à tous ceux qui ont envie de partager leur savoir ou qui aimeraient bien qu'on les aide un peu. Des questions que vous posez auxquelles vous répondez, ainsi que nous-mêmes et l'invité du mois si possible.

Le Requester : alias le courrier des lecteurs. Pourquoi pas ?

Hein ? Il vous manque quelque chose ? Une rubrique supplémentaire vous plairait ? Ou au contraire, celle-ci vous semble de trop ? Vous n'aimez pas la maquette du journal ? Vous avez trop mangé hier soir ? N'hésitez pas : notre sondage est là pour vous... sonder (ben oui). Dites tout ce qui vous passe par la tête et même plus, on ne vous en voudra pas. De plus, cela peut vous permettre de gagner des heures en 3614 sur le serveur minitel de l'ANT, alors qu'est-ce que vous attendez ?