

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT

ExecBase

Déboguer avec Monam (II)
par D. Jarril (p.2)

Interruptions sous Exec
par S. Schreiber (p.22)

La Mémoire du Monde
par P. Vautrin (p.28)

AMOS

Interpréteur d'expressions
par F. Lionet (p.4)

ARexx

Travail Pratique (II)
par F. Gueugnon (p.6)

ToolBox

ADebug
par E. Brunet et F. Fleuret
(p.8)

Awk
par Herr Doktor (p.22)

SubWay

Les Layers (II)
par Max (p.18)

Utilitaire

ExecMaster
par Max (p.12)

Transactor

Le bon octant
par J. S. Valette (p.18)

Demos

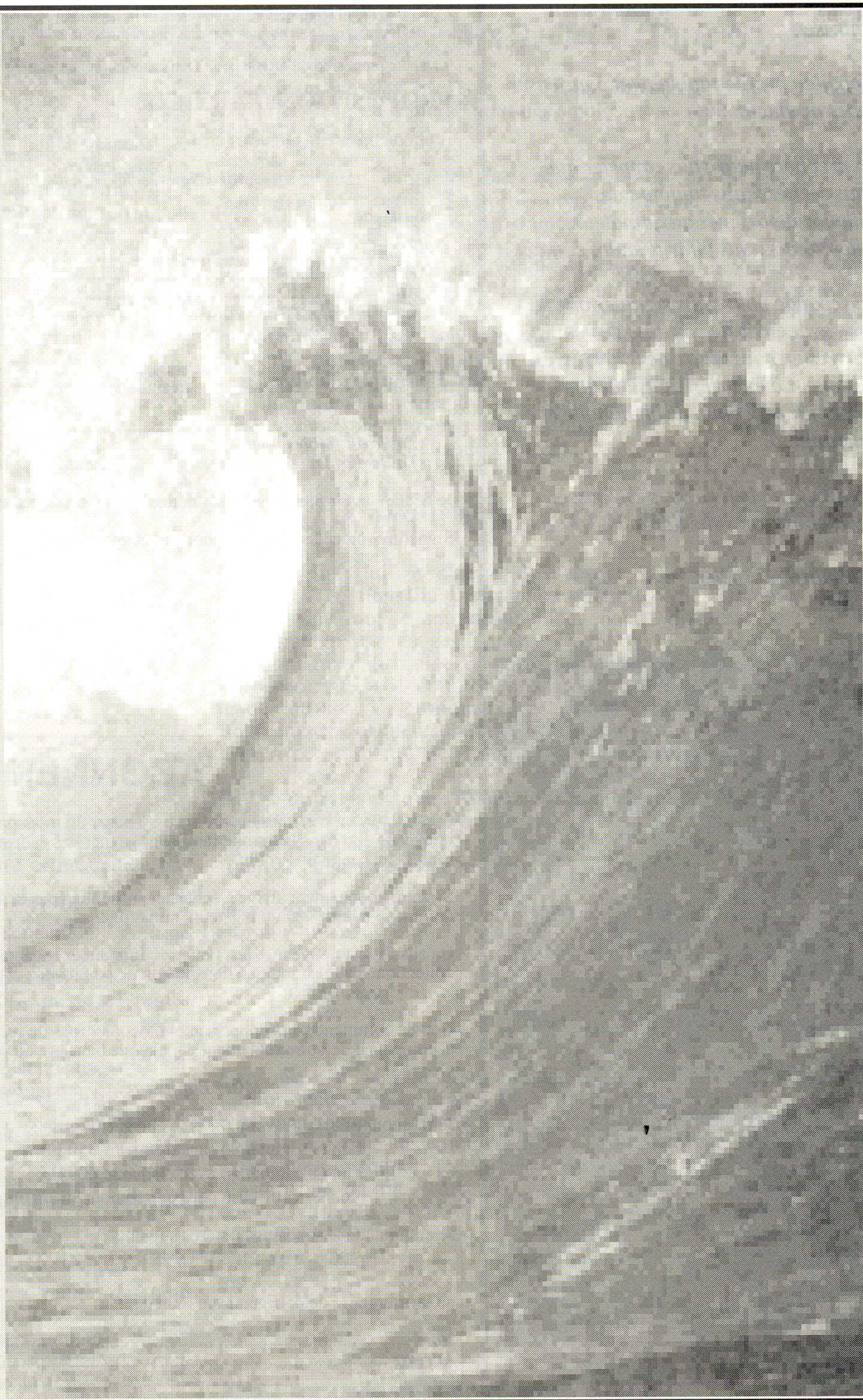
Scrolling "Kick-Off"
par J. Etienne (p.24)

Requester

par Ok et Cancel (p.30)

Hardware

Les disques chouettes (III)
par L. Far (p.32)



45 F.

Ce numéro 27 de l'Amiga NewsTech a été tiré à 3500 exemplaires.
Amiga NewsTech est une publication Commodore Revue SARL 5, rue de la Fidélité, 75010 PARIS

EDITO

Vous avez vu ? Ce numéro est en retard. Et pour une fois, on ne pourra accuser ni les PTT, ni le mauvais temps, ni même le destin. Non, nous avons tout simplement tout changé à notre système de PAO, passant de Professional Page à Publishing Partner Master. Résultat des courses, on a lamentablement merdoyé avant de réussir à sortir quelque chose d'imprimable. Toutes nos excuses donc, ça ne devrait plus se reproduire. On maîtrise.

Hop, j'en profite rapidement pour caser deux annonces.

La première pour vous rappeler que nous recherchons toujours des collaborateurs, avis aux amateurs. Téléphonez-nous pour prendre contact.

La seconde pour vous signaler que si vous avez envie de voir une de vos oeuvres en couverture de l'ANT, c'est possible. Là encore, il suffit de nous passer un petit coup de bigophone et on s'arrangera.

A part ça, hé ben ça va, s'il s passe quek'chose, on vous l' dira.

Stéphane Schreiber

O U R S

Amiga NewsTech est une publication mensuelle et vendue uniquement par abonnement, de Commodore Revue SARL, 5, rue de la Fidélité, 75010 Paris.

Rédaction
5, rue de la Fidélité, 75010 Paris.
Tel : (1) 42.46.92.90. Fax : (1) 47.70.08.21.

Directeur de la publication : Jean-Yves Primas
Rédacteur en chef : Stéphane Schreiber
Rédacteur en chef adjoint : Stéphane Schreiber
Adjoint du red'chef adjoint : Stéphane Schreiber
Premier Secrétaire de rédaction : Stéphane Schreiber
Secrétaire de rédaction : Stéphane Schreiber
Chef de rubrique : Stéphane Schreiber
A collaboré à ce numéro : Stéphane Schreiber

Fabrication
Flashage par ColorWay, 5, rue de la Fidélité, 75010 Paris.
Tel : (1) 47.70.87.87. Fax : (1) 48.24.97.77

Imprimé par INO-Montrouge, 9, rue Auber, 92120 Montrouge.
Tel : (1) 46.56.27.69.

Chef de fabrication : Stéphane Schreiber
Premier maquettiste : Stéphane Schreiber
PAO : Stéphane Schreiber
Support technique : Yves Huitric
Support moral : Get 27, Vodka Eristoff, Lucky Strike
Support physique : 3615 Ulla

Publicité
Chef de publicité : Stéphane Schreiber
Assistan : Stéphane Schreiber
Dessous de table : Stéphane Schreiber

Abonnements, anciens, présents et futurs numéros
Gérés par MCM Europe, 16 Quai Jean-Baptiste Clément, 94140 Alfortville. Tel : (1) 43.78.92.10. Fax : (1) 43.78.99.89.

L'intelligence, c'est comme les parachutes : quand on n'en a pas, on s'écrase. (Pierre Desproges)

TRANSACTOR

- 1.) La rubrique Transactor de l'Amiga NewsTech est la rubrique écrite par les lecteurs, pour les lecteurs. Son but est de leur permettre de s'exprimer librement sur tout sujet qui les passionne, dans quelques langage de programmation que ce soit. Seule restriction : la Rédaction doit avoir la possibilité de tester le bon fonctionnement de tous les programmes fournis, complets ou simples routines d'illustration. Non mais.
- 2.) Tout lecteur désirant participer à la rubrique Transactor devra envoyer son article sur disquette au format Amiga, en respectant les quotas définis aux paragraphes 4 et 5. Les disquettes ne seront pas retournées, sauf demande expresse et écrite de l'auteur, et encore.
- 3.) Tout article publié sera rémunéré au tarif forfaitaire de 1, 500 F HT les deux pages ; un article de plus de deux pages sera publié en plusieurs fois, à concurrence de trois épisodes maximum. La paiement a lieu le mois suivant celui de la parution. Les auteurs des articles retenus pour parution seront directement prévenus par téléphone.
- 4.) Les articles doivent parvenir au format ASCII standard (utilisez un éditeur de textes ou l'option de sauvegarde text only ou ASCII de votre traitement de texte). Les programmes illustrant les articles, qu'ils soient complets ou de simples routines d'exemple, doivent également être fournis sous forme ASCII, et éventuellement en format brut le cas échéant (Basic, AMOS...). Le(s) fichier(s) exécutable(s) doivent se trouver sur la disquette. L'auteur doit fournir à l'ANT la possibilité de recompiler ou d'interpréter ses programmes sans difficulté (joindre au besoin une version run-only du langage utilisé).
- 5.) La taille totale du fichier ASCII donne le nombre de signes de votre article ; une page de l'ANT contient en moyenne 7, 000 signes de texte. Une colonne de listing contient 80 lignes de 65 caractères de large. Un programme de 160 lignes occupe donc une page entière. Les programmes en assembleur particulièrement longs peuvent être publiés en trois colonnes de 40 caractères chaque (240 lignes par page).
- 6.) Toute proposition d'article pour Transactor devra être accompagnée du bon ci-dessous, découpé ou photocopié, mais en aucun cas reproduit à la main.
- 7.) Toute proposition d'article pour Transactor entraîne l'acceptation par l'auteur que les programmes joints soient placés sur la disquette d'accompagnement de l'ANT. L'auteur doit préciser dans des commentaires en tête de listing s'il place son programme dans le domaine public ou s'il désire conserver son copyright.
- 8.) Ni l'ANT, ni Commodore Revue SARL, ni les auteurs collaborant à la rubrique Transactor ne sauraient être tenus pour responsables en cas de dommages quelconques survenus à l'ordinateur, suite à une mauvaise utilisation des documents (textes et programmes) publiés dans cette rubrique.
- 9.) Toute proposition d'article pour Transactor entraîne l'entière acceptation par l'auteur de ce règlement.

PROPOSITION D'ARTICLE

A photocopier et à retourner à :

Amiga NewsTech (Transactor), 5, rue de la Fidélité, 75010 Paris, France

Je soussigné,

Nom
Prénom
Adresse
Code Postal **Ville** **Pays**
N° de téléphone

déclare être l'auteur de l'article et du (des) programme(s) ci-joints. Je désire les soumettre à l'appréciation de la rédaction de l'Amiga NewsTech, pour une éventuelle publication dans le cadre de la rubrique Transactor.

Sujet de l'article
Taille du fichier texte **signes**
Taille du (des) programmes(s) **lignes**

Je déclare avoir pleinement pris connaissance du règlement ci-dessus et l'approuver entièrement.

Signature (précédée de la mention "lu et approuvé") :

ABONNEMENTS A L'ANT

C'est nouveau, c'est pas encore sorti : à partir du mois prochain, si vous êtes déjà abonné, vous pourrez parrainer un nouveau venu dans la grande famille et gagner des cadeaux offerts par CIS. Je ne veux pas gâcher la surprise, vous en saurez plus en temps voulu.

Si votre abonnement arrive à échéance, n'oubliez pas de le renouveler le plus tôt possible. Ce serait quand même dommage de rater un ou plusieurs numéros !

Ca aussi c'est nouveau, vous pouvez commander les anciens numéros de l'ANT à la pièce, afin de compléter votre collection. Tout est indiqué dans le bon de commande ci-dessous.

Bon de commande à découper ou photocopier (pas de recopie manuelle SVP) et à retourner correctement rempli et accompagné de votre règlement par chèque bancaire ou postal libellé à l'ordre de à :

MCM Europe, 16, Quai Jean-Baptiste Clément, 94140 Alfortville, France

Je m'abonne à Amiga NewsTech. Je choisis la formule

- ☐ 1 an (11 numéros) sans disquette 350 FF
☐ 1 an (11 numéros) avec disquette 600 FF
☐ (cochez cette case pour un ré-abonnement)

Je commande les anciens numéros suivants

- ☐ n°20 ☐ n°21 ☐ n°22 ☐ n°23 ☐ n°24 ☐ n°25 ☐ n°26

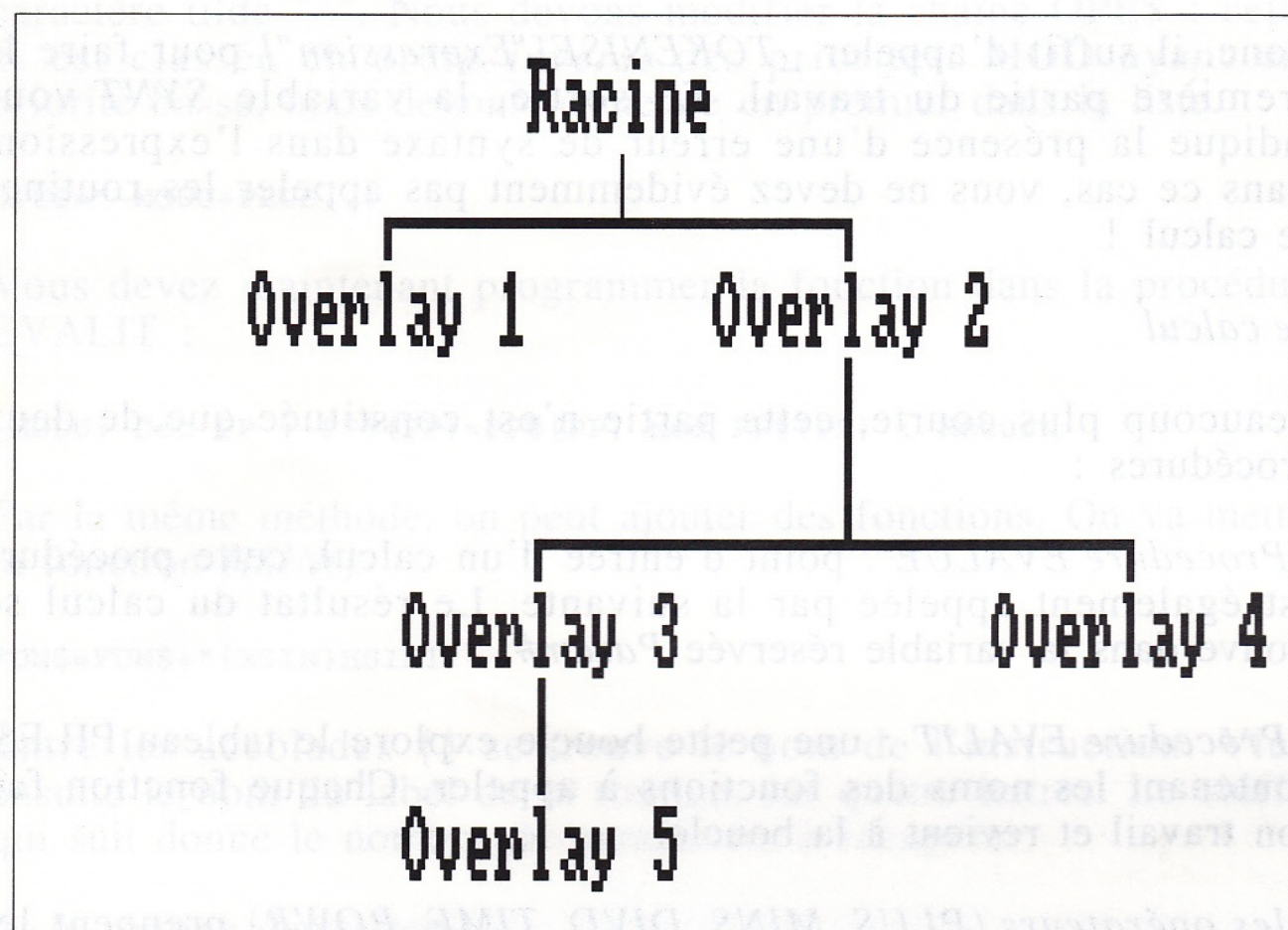
au tarif unitaire de 45 F, pour un total de F.

Nom **Prénom**
Adresse
Code Postal **Ville** **Pays**

La technique de programmation par "overlays" permet de disposer d'une application de plusieurs centaines de kilo-octets sur disque mais en utilisant beaucoup moins en mémoire centrale. Comment ça marche ?

C'est très simple, tout du moins dans la théorie : on ne charge les routines en mémoire qu'au fur et à mesure de leur utilisation, et on les libère lorsqu'elles se terminent. Vous vous doutez certainement que pratiquement, c'est plus compliqué : une routine du programme principal doit s'occuper de libérer et de charger les nouveaux overlays au moment opportun. Fort heureusement, le compilateur Lattice Lc et son linker associé BLink réduisent au strict minimum le travail du programmeur : celui-ci n'a plus qu'à simplement prévoir quelles fonctions de son application seront ou ne seront pas des overlays. Avouez que c'est déjà quand même plus simple.

Une représentation classique des overlays est sous forme d'arborescence, comme dans la figure 1 ci-dessous.



La *Racine* est le programme principal, le premier à être chargé. Quoiqu'il arrive, il réside toujours en la mémoire et ne pourra pas en être viré par le gestionnaire d'overlays.

Les *Overlay 1* et *Overlay 2* s'exécutent indépendamment l'un de l'autre, un seul à la fois : le programme principal ne peut pas appeler de fonctions de l'*Overlay 1* puis de l'*Overlay 2* sans que le gestionnaire ne le charge d'abord. De cet arbre, quelques règles simplissimes découlent :

- si l'*Overlay 1* est chargé, les 2, 3, 4 et 5 ne le sont pas.
- si l'*Overlay 2* est chargé, le 1 ne l'est pas, mais 3, 4 et 5 peuvent l'être (ce n'est pas obligatoire).
- si l'*Overlay 3* est chargé, le 2 l'est aussi, les 1 et 4 ne le sont pas, et le 5 peut l'être.
- si l'*Overlay 4* est chargé, les 1, 3 et 5 ne le sont pas.
- si l'*Overlay 5* est chargé, les 2 et 3 aussi, mais les 1 et 4 ne le sont pas.

Malheureusement, quelques restrictions existent :

- la *Racine* peut appeler des routines de n'importe lequel des cinq overlays.
- les 5 overlays peuvent appeler des routines de la *racine*.
- * toutes les routines de l'*Overlay 1* peuvent appeler d'autres routines de l'*Overlay 1* ainsi que toutes celles de la *Racine*.
- toutes les routines de l'*Overlay 2* peuvent appeler d'autres

routines de l'*Overlay 2* ainsi que celles de la *Racine* et des *Overlays 3* et 4.

- toutes les routines de l'*Overlay 3* peuvent appeler d'autres routines de l'*Overlay 3* ainsi que celles de la *Racine* et des *Overlays 2* et 5.
- etc...

On le voit, deux noeuds de même niveau sont mutuellement exclusifs : l'*Overlay 1* ne peut pas appeler des routines de l'*Overlay 2*.

LES OVERLAYS, POUR QUI ?

Le mécanisme des overlays n'est pas à utiliser n'importe quand. On le réserve en général pour les applications réellement énormes (plus de 100 Ko).

Imaginons par exemple que vous écriviez un programme de traitement de texte. Le programme principal - la *Racine* - contiendra toutes les routines d'entrée de caractères, d'édérations, de manipulations de bloc... Un premier overlay pourrait contenir les routines d'impression, qui n'ont pas besoin d'être chargées en permanence. Un second overlay pourrait s'occuper de l'impression en PostScript, qui est totalement différente de l'impression matricielle ou laser (on pourrait également partager le premier overlay en deux, l'un pour PostScript, l'autre pour les matricielles). Enfin, un troisième overlay pourrait gérer l'analyse du document (comptage du nombre de mots, de caractères, repérage des répétitions, histogrammes...). C'est uniquement au programmeur de décider de tout cela.

LES OVERLAY, COMMENT ?

C'est BLink qui se charge de presque tout, lors de la phase d'édérations de liens du programme définitif : il remplace chaque appel d'une routine appartenant à un noeud différent de celui en cours par un saut au gestionnaire d'overlays. Celui-ci décide alors, en fonction du niveau des deux noeuds concernés dans l'arborescence, s'il doit d'abord charger quelque chose depuis le disque ou non. Le gestionnaire est inclus dans les bibliothèques du Lattice, mais on peut au besoin le remplacer par un fait maison. Ce qui ne présente pas réellement beaucoup d'intérêt.

BLink utilise un fichier WITH spécial dans sa ligne de commande qui décrit l'arborescence des overlays. Le format de ce fichier est strictement défini et doit être respecté à la lettre. L'arbre de la figure 1 pourrait être décrit dans le fichier suivant :

```

OVERLAY
Overlay1
Overlay2
*Overlay3
**Overlay5
*Overlay4
#
  
```

Le mot-clé *OVERLAY* introduit le premier niveau de l'arborescence et le signe dièse (#) termine la liste. La *Racine* n'est pas incluse : BLink utilise comme *racine* le premier fichier objet de sa ligne de commande (le fichier *FROM* ou *ROOT*). Une astérisque introduit un niveau supplémentaire d'overlay, pas d'astérisque du tout correspondant au niveau 1. Remarquez enfin que l'*Overlay 5* a été placé tout de suite derrière le 3, dont il dépend dans l'arbre.

Par Denis Jarril

Bonjour les z'amis, aujourd'hui je m'en vais vous proposer un programme d'évaluation d'expressions. Holà, ça va être raser ça. Je vous vois déjà tourner la page pour aller lire les excellents jeux de mots de Max...

Non, restez, ce que je vous propose est complètement hypra-dément (excusez du peu). Parmi les 10,385,129 questions posées sur AMOS à ce jour, une revient très souvent (10,385,128 fois exactement), je cite : "Pouvez-vous mettre une fonction EVAL, pour que je puisse faire un traceur de courbes ?". Ma réponse étant invariablement, je cite également : "NON !".

Le rôle d'une telle fonction est de permettre à l'utilisateur d'entrer une équation, et de pouvoir la calculer à l'intérieur du programme.

C'est très pénible à réaliser dans un interpréteur : la fonction EVAL fait appel à des routines de l'éditeur, se trouvant à un niveau totalement différent de celui de l'interpréteur. De plus, la compilation d'une telle fonction est tout-à-fait impossible.

La seule solution pour programmer un traceur de courbe est de réaliser une fonction EVAL en AMOS. C'est ce que je vous propose aujourd'hui.

LE PRINCIPE

Le principe du programme que je propose, est totalement pompé sur les routines d'évaluation d'expressions de l'AMOS et du compilateur. Je vais me gêner, tiens ! Vous allez voir que c'est complètement récursif : les procédures n'arrêtent pas de s'appeler l'une-l'autre.

Il faut, pour tracer une courbe sur un écran HIRES, calculer la même expression au moins 640 fois. Le programme a donc intérêt à être rapide, pour ne pas finir à la poubelle. Or, explorer une chaîne de caractères n'est pas des plus performant. C'est pourquoi le programme d'aujourd'hui se divise en deux parties :

- analyse de la chaîne de caractères et fabrication d'une liste des fonctions à appeler pour le calcul. Cette partie, la plus lente, s'appelle la tokenisation (transformation en "tokens"). Elle n'est effectuée qu'une fois.
- exploration de la liste des fonctions fabriquée par la tokenisation et appel des routines correspondantes. C'est la partie rapide, appelée 640 fois pour dessiner la courbe.

LE PROGRAMME

Comme aujourd'hui je veux vous faire comprendre le principe de l'évaluation, hé bien, je garde le traceur de courbes pour le mois prochain.

Initialisation

- la variable *MXE* contient la taille maximum des expressions. Chaque élément d'une expression compte pour une unité. Ainsi, *1+1* prend trois positions dans les tableaux.
- suit la définition des variables. *PILE()*, *PILE#()* et *PILE\$()* sont les tableaux importants, contenant les noms des routines à appeler.
- la variable *OPE\$* contient les opérateurs disponibles. Voir plus loin pour rajouter vos propres opérateurs.
- la variable *FUN\$* contient, elle, les fonctions disponibles. Voir également plus loin pour ajouter vos propres fonctions.

- suit une petite boucle d'initialisation de variables internes, à titre d'exemple.
- la boucle principale du programme se contente de demander une expression et d'afficher le résultat.

LA TOKENISATION

Nous n'allons pas trop nous éterniser dessus. Elle est constituée de quatre procédures :

- *Procédure _TOKENISE[a\$]* : c'est la procédure à appeler pour lancer la tokenisation. Elle initialise les variables, et appelle la procédure suivante.
- *Procédure _TEVALUE* : se charge de digérer une expression entière, comme "1+1". Elle peut être appelée plusieurs fois, pour les expressions entre parenthèses (quand je vous disais que c'était récursif !).
- *Procédure _TOPERAND* : se charge d'appréhender un opérande, un chiffre ou une fonction. Dans le dernier cas, elle appelle la procédure *_TEVALUE* pour récolter les paramètres, etc.
- *Procédure CHAR* : ramène les lettres de la chaîne à tokeniser, une par une.

Donc, il suffit d'appeler *_TOKENISE["Expression"]* pour faire la première partie du travail. En sortie, la variable *SYNT* vous indique la présence d'une erreur de syntaxe dans l'expression. Dans ce cas, vous ne devez évidemment pas appeler les routines de calcul !

Le calcul

Beaucoup plus courte, cette partie n'est constituée que de deux procédures :

- *Procédure EVALUE* : point d'entrée d'un calcul, cette procédure est également appelée par la suivante. Le résultat du calcul se trouve dans la variable réservée *Param#*.
- *Procédure EVALIT* : une petite boucle explore le tableau *PILE\$*, contenant les noms des fonctions à appeler. Chaque fonction fait son travail et revient à la boucle.
- les opérateurs (*PLUS*, *MINS*, *DIVD*, *TIME*, *POWR*) prennent les valeurs sur la pile *PP#()*, effectuent leur opération, et empilent le résultat.
- les fonctions appellent la routine d'évaluation pour récupérer les paramètres, puis poussent le résultat sur la pile. Seule exception, la fonction *=PI*.

FONCTIONNEMENT

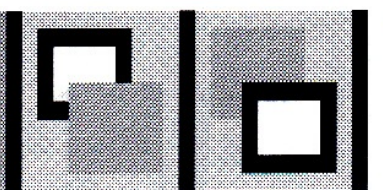
Le programme affiche les noms des fonctions appelées par la procédure EVALIT. Pour mieux comprendre, voici quelques expressions et la liste des fonctions appelées pour chacune.

1

```
CONST 1 : pousser la valeur 1 sur la pile
END 0 : (le paramètre n'est pas utilisé)
```

4/2+3>

```
CONST 4 : pousse 4 dans la pile
CONST 2 : pousse 2 dans la pile
DIVD 0 : (paramètre non utilisé), divise les deux
derniers chiffres de la pile, ici 4/2,
et pousse le résultat. La pile contient
donc 2
```

```
CONST 3 : pousse 3 dans la pile
PLUS    : additionne 3 et 2
END      : le résultat est donc 5
```

La logique de calcul utilisée ici est donc la logique polonaise inverse, que tous les utilisateurs de calculatrices HP connaissent bien. Plus compliqué :

```
SQR(2)+3
SQR 4 : le paramètre n'a pas d'importance ici. La
       routine SQR appelle EVALUE, pour
       récupérer le paramètre
CONST 2 : pousse le 2 de la racine carrée
END 0 : fin de l'évaluation du paramètre, on
       revient à la fonction SQR, qui pousse le
       résultat dans la pile
CONST 3 : pousse le 3
PLUS 0 : fait l'addition finale
END 0 : fin de l'expression
```

AJOUTER DES FONCTIONS

Il est très facile d'ajouter ses fonctions et ses opérateurs.

Par exemple, ajoutons l'opérateur *Modulo*, qui sera codé par le caractère tilde "~". Nous devons modifier la chaîne OPE\$: celle-ci est classée en ordre *inverse* des priorités. MOD ayant une priorité basse, nous devons le mettre en premier dans la liste.

```
OPE$="~MODU+PLUS...
```

Vous devez maintenant programmer la fonction dans la procédure EVALIT :

```
_MODU: Dec PP : PP#(PP)=PP#(PP) Mod PP#(PP) : Return
```

Par la même méthode, on peut ajouter des fonctions. On va mettre la fonction *HSIN()*.

```
FUN$=FUN$+"{HSIN}HSIN1"
```

Entre les accolades {} se trouve le nom de l'instruction. Vient ensuite le nom du label de la routine, sur quatre lettres. Le chiffre qui suit donne le nombre de paramètres à récupérer.

Et la routine de traitement :

```
_HSIN: EVALIT : PP#(PP)=Hsin(PP#(PP)) : Return
```

Voilà, c'est tout pour aujourd'hui, j'ai déjà dépassé la taille autorisée par mon rédacteur en chef adoré (sur tranche). Le mois prochain, Daisy fera le traceur de courbes, et moi je parlerai d'autre chose. A plus !

```
' Evaluation d'expression en AMOS
'-----
Screen Open 0,640,200,4,Hires
'
' Taille maximum des expressions
MXE=25
'
' Definition des variables
Dim PILE(MXE),PILE#(MXE),PILE$(MXE),PP(MXE),PP#(MXE)
Global OPE$,FUN$,EV$,PEV,LEV,SYNT,PILE(),PILE$( ),PILE#( ),PPILE
Global PP(),PP#( ),PP,NPAR,CHAR$
'
' Definition des operateurs, en ordre inverse
OPE$="+PLUS-MINS*TIME/DIVD^POWR"
'
' Definition des fonctions
FUN$="{PI}PIPI{VAR()}VARS2{SIN()}SINU1{COS()}COSI1{TAN()}TANG1"
```

```
FUN$=FUN$+"{SQR()}SQRR1{LOG()}LOGG1{EXP()}EXPP1{LN()}LNNN1"
FUN$=FUN$+"{ABS()}ABSS1{INT()}INTT1{SGN()}INTT1{RND()}RNDD1"
```

```
' Example de variables internes
Dim VAR#(10,10)
Global VAR#()
For A=0 To 10
  For B=0 To 10
    A$=Str$(A)+". "+Str$(B)
    VAR#(A,B)=Val(A$)
  Next
Next
'
' Petite boucle de tests
Do
 Cls
  Print "Evaluation d'expression en AMOS."
  Print
  Line Input "Entrez une expression S.V.P :";A$
  If A$="" : Exit : End If
  '
  ' Va tokeniser
  _TOKENISE[A$]
  '
  ' Calcule le resultat
  Print
  If Not SYNT
    EVALUE
    Print "Resultat: ";Param#
    '
    ' Pour voir le procédé de calcul
    Print
    For N=0 To PPILE
      Print PILE$(N);PILE#(N)
    Next
  Else
    Print "Syntax error!"
  End If

  Print
  Print "Pressez une touche..." : Wait Key
Loop
End
'-----
' PROCEDURES DE TOKENISATION
'
Procedure _TOKENISE[A$]
  EV$=Upper$(A$) : PEV=1 : LEV=Len(EV$) : PPILE=0 : SYNT=0
  _TEVALUE[0] : PILE$(PPILE)="_END" : Inc PPILE
  If NPAR : SYNT=-1 : End If
End Proc[SYNT]
'
Procedure _TEVALUE[N]
  If SYNT : Pop Proc : End If
  P=0 : NPAR=N
  Do
    Inc PP : PP(PP)=P
    Do
      _TOPERAND
      Do
        CHAR : P=Instr(OPE$,CHAR$)
        Exit If P>PP(PP),2
        Dec PEV : P=PP(PP) : Dec PP
        Exit If P=0,3
        PILE$(PPILE)="_"+Mid$(OPE$,P+1,4) : Inc PPILE
      Loop
    Loop
    Loop
    If CHAR$=")" : Dec NPAR : Inc PEV : End If
  End Proc[RES]
```

(suite page 31)

AREXX: TRAVAIL PRATIQUE (II)

Dans le chapitre précédent, nous avons écrit et expliqué le premier module de notre projet d'édition du catalogue du disque de notre choix. Nous nous attaquons aujourd'hui au second module.

Rappelons que ce premier module permettait d'obtenir de manière quasi-certaine, c'est-à-dire sans risque d'erreur, le nom du disque dont on désire obtenir le catalogue, ce nom pouvant être logique (par exemple, Arexx:) ou bien physique (par exemple, DH0:). Mais afin de permettre l'interfaçage entre les deux modules, nous proposons une petite, toute petite, modification. Il s'agit d'ajouter juste avant l'instruction EXIT les deux instructions PUSH suivantes :

```
PUSH device
PUSH 'rx ram:Traitement'
EXIT /* Ligne 48 du module précédent */
```

Ces deux instructions "poussent" dans la pile de la console, d'une part la variable ARExx device (qui contient la définition du disque choisi) et d'autre part, une instruction du type CLI, Shell ou ConMan.

Voici alors ce qui va se passer (on supposera que le nom du programme décrit ci-après est *Traitement* et qu'il est placé en RAM:).

Il faut d'abord lancer le premier programme, qui va obtenir le nom légal du disque à analyser. A la fin de ce programme, juste avant EXIT, ces deux instructions vont pousser dans la pile le nom du disque à analyser et celui du programme qui devra le faire. Notons en passant que ces deux instructions appartiennent chacune à des mondes différents. Dès la sortie par EXIT, la main revient à l'hôte, ici la console. L'instruction rx ram:Traitement, insérée en dernier par PUSH, est celle qui va être rappelée la première (différence entre PUSH et QUEUE). C'est une instruction légale pour la console, elle lance donc le programme ARExx Traitement.

Dans ce programme qui prend alors la main, la première instruction est PULL nomvariable ; cette instruction extrait de la pile celle qui reste, à savoir le nom du disque à traiter. Voici donc une particularité intéressante qui permet de passer à peu près n'importe quoi d'un programme à un autre, sans égard pour la différence des types de programme.

PROGRAMME TRAITEMENT

Ce programme permet d'obtenir le répertoire du disque précédemment défini avec la longueur de chaque fichier. Voici son mécanisme :

- en utilisant la fonction *dir opt a* du DOS, création d'un fichier en RAM: nommé *repertoire*.
- analyse ligne par ligne de ce fichier et obtention de la taille de chaque entrée.
- création d'un fichier en RAM: nommé *resul* comprenant les indications nécessaires au prochain programme pour afficher notre forme personnelle du répertoire.

Note : on aurait pu utiliser les fonctions spécialisées d'ARExx pour obtenir le répertoire désiré (en l'occurrence, la fonction *DIR*). Nous avons bien précisé que nous ne cherchions qu'à montrer des fonctionnements et des mécanismes !

Pour analyser le fichier placé en RAM, il faut en connaître la structure. C'est un fichier texte, dans lequel chaque ligne peut

contenir soit un nom de répertoire, soit deux noms de fichiers. Le décalage des répertoires entre eux, selon leur rang, est de 5 caractères, le premier étant à 3 caractères de la marge gauche. Les fichiers d'un répertoire sont décalés de 2 caractères à droite.

C'est le calcul de la position du répertoire qui donne son rang et les fichiers qui lui sont affectés. Il faut donc retenir les différentes positions pour affecter correctement les fichiers aux répertoires, surtout en revenant d'un emboîtement de répertoires ! Les fichiers, eux, sont placés à raison de 2 par ligne, lorsque le nombre est pair. Dans le cas contraire il n'y en a qu'un. Un intitulé de fichier peut contenir des espaces ce qui leurre ARExx, qui l'interprète comme un séparateur. En conséquence, l'extraction des fichiers se fera par position, car elle reste heureusement constante.

Enfin, le fichier résultat aura une forme particulière. Ce sera une chaîne de caractères contenant toutes les informations nécessaires, concaténées sans espaces et avec des séparateurs spécifiques. Cette chaîne sera construite par concaténations consécutives des valeurs utiles dès leur obtention.

Avertissement : une erreur de programmation initiale fait que la chaîne finale n'a pas tout-à-fait la régularité souhaitée. Au moment d'effectuer la correction, nous avons pensé tirer parti de la chose en transformant un bug en *feature* ! En effet, dans le programme suivant, nous verrons comment filtrer et corriger l'erreur. Ceci pour montrer la souplesse des programmes qui peuvent traiter des logiciels extérieurs sans s'occuper vraiment de la logique matérielle de ces programmes.

Ce programme fait partie d'un tout. On peut toutefois le lancer seul (par exemple, *rx ram:traitement*) ; l'instruction PULL en tête attend alors un argument qui aurait été placé dans la pile par le programme précédent. Comme il n'y est pas, il faut le fournir à partir du clavier. Par exemple, *DFI:* et return.

LISTING DU PROGRAMME

Ce programme est listé ci-contre. Rappelons que les numéros de lignes n'ont été mis que pour faciliter la lecture ; il ne faut pas les taper lors de la recopie du programme. Les explications nécessaires ont été directement incluses dans le listing, mais il semble toutefois utile d'apporter quelques compléments.

Certaines lignes de programme sont trop longues pour la largeur de la page de l'éditeur utilisé. Dans ce cas, il est possible de couper la ligne sans affecter son intégrité logique, en plaçant une virgule à l'endroit de la coupure, suivie bien sûr d'un retour-chariot.

La création d'une chaîne de caractères continue pour matérialiser le résultat est totalement arbitraire. Mais cela représente une bonne introduction à la lecture de chaînes qui n'ont pas de sens ASCII, c'est-à-dire du code ou tout autre chose, dans lesquelles la notion de ligne n'a pas de sens. C'est un des aspects sympathiques et uniques d'ARExx de traiter uniquement des caractères.

Dans ce programme, nous voyons comment nous pouvons d'une part acquérir des noms de répertoires ou de fichiers, les traiter, mélanger, construire et leur faire subir tout ce qu'on peut effectuer sur des chaînes de caractères et d'autre part, pourvu qu'ils aient une forme légale, les utiliser comme commandes par exemple pour changer de répertoire !

Enfin pour passer des éléments : variables, chaînes, valeurs, tableaux, commandes, etc., nous avons utilisé et nous utilisons encore cette fois-ci, le passage par la pile. Cependant, nous avons ici utilisé concurremment une autre méthode qui s'apparente aux POKE et PEEK du Basic, avec cette différence essentielle toutefois que l'opération se fait au travers de l'allocateur général de l'Amiga et qu'en conséquence, il n'y a pas de risque d'écraser

un espace occupé par quelqu'un d'autre, non plus que de risquer pour soi les mêmes déboires. Dans le présent programme, nous n'avons pas pris la précaution de vérifier que l'allocation avait été correctement effectuée en regardant le RC (pour *Return Code*), ce qui est imprudent en milieu multitâche.

Cette allocation survient à la fin du programme demandeur d'allocation. Cela impose de gérer la désallocation lorsque il n'est plus utile de conserver cet espace. L'instruction est *FREESPACE(adresse,taille)*, d'où la nécessité de passer ces valeurs aux programmes utilisateurs.

```

/*début de la boucle principale qui se termine en ligne 37 */
1 /*TRAITEMENT*/
2 /*call trace i r*/ /*pour la mise au point uniquement*/
3 PULL disque /*extrait le nom du disque de la pile*/
4 x=PRAGMA('D',disque) /*installe le répertoire courant*/
/*commande CLI pour mettre le sous forme de fichier texte*/
5 ADDRESS COMMAND 'dir > ram:repertoire opt a'
6 x=OPEN('lec','ram:repertoire','R') /*ouvre le fichier*/
7 x=OPEN('ecr','ram:resul','W') /*création du fichier
résultat */
8 posancien=1 /*préparation de variables */
9 soustotal=0 /*idem*/
10 total=0 /*idem*/
11 taquet=0 /*idem*/
/*préparation d'une ligne de taille déterminée chaine blanche*/
12 d='
13 file=STRIP('| 'disque' |') /*obtention d'une chaine sans
caractères blancs en tête et en queue*/
/*boucle de lecture des lignes du fichier repertoire en ram*/
14 DO k=1 BY 1 WHILE ~EOF('lec')
15 ligne=READLN('lec')
16 pos=WORDINDEX(ligne,1)
17 IF pos=0 THEN DO
18 pos=posancien-1
19 ITERATE k
20 END
/*sélection du traitement selon que c'est un répertoire ou un
fichier, la clause étant de trouver ou non la chaine 'dir' dans
la ligne analysée*/
21 IF INDEX(ligne,'(dir)',1) ~= 0 THEN CALL repertoire
22 ELSE CALL fichier
23 END
24 CALL traitement
25 SAY 'Le grand total est:'total
26 SAY 'J'ai terminé'
27 SAY file
28 x=WRITELN('ecr',file)
/*préparation d'un espace mémoire pour stocker résul en dehors de
ce programme afin de l'utiliser dans le programme d'affichage*/
29 say LENGTH(file) /*calcul de la taille nécessaire*/
30 alpha=GETSPACE(5000)
31 EXPORT(alpha,file) /*stockage dans la mémoire réservée*/
32 PUSH alpha /*place l'adresse dans la pile*/
33 PUSH 5000 /*met la taille dans la pile*/
34 PUSH disque /*place le nom du disque dans la pile */
35 PUSH 'rx RAM:affichedir' /*prépare le lancement du
troisième et dernier module */
37 EXIT /*fin du programme avec retour à la console*/
/*premier sous-programme: traitement des répertoires*/
40 repertoire: /*nom du sous-programme*/
43 p=(pos-1)%5
44 PARSE VAR ligne courantdir '(' . /*analyse de la ligne
jusqu'au séparateur désigné: ( */
45 dir.p=STRIP(courantdir) /*l'indice du tableau est le rang
du répertoire*/
46 dir=disque
/*boucle de création du chemin de sélection lorsque il y a
plusieurs répertoires imbriqués*/
47 DO i=1 TO p
48 dir=dir||dir.i||'/'
49 END
50 dir=STRIP(dir,'B','/')
51 x=PRAGMA('D',dir)
52 file=STRIP(file '<'dir p'>')
53 CALL traitement /*appel à un sous programme*/
54 CALL imprime 'Le dir en cours est:'PRAGMA('D') /*appel à
un sous-programme avec passage d'une chaine*/
55 soustotal=0
56
57 RETURN /*fin du sous-programme*/
58
59
/*sous-programme de traitement des fichiers*/
60 fichier: /*nom du sous programme*/
61
62 /*exemples d'imbrications de fonctions*/
63 if STRIP(ligne) ~= '' THEN

```

```

fichier1=STRIP(SUBSTR(ligne,pos,32))
64 if STRIP(ligne) ~= '' THEN
fichier2=STRIP(SUBSTR(ligne,pos+33,32))
65
66
67
68
69 IF pos < posancien THEN DO/*fin de la boucle DO ligne 82*/
70 posit=(pos-3)%5
71 dir=disque
72 IF posit ~=0 THEN DO i=1 TO posit /*fin de la boucle DO
ligne 74*/
73 dir=dir||dir.i||'/' /* la concaténation sans espace est
due à: || */
74 END
75
76 dir=STRIP(dir,'B','/') /*élimination en tête et queue de:
/ */
77
78 x=PRAGMA('D',dir)
79 file=STRIP(file '<'dir posit '>')
80 CALL traitement
81 CALL imprime 'Le dir en cours est:'PRAGMA('D')
82 END
83
84
85 /*STATEF permet d'obtenir divers renseignements sur le
fichier défini par son nom et bien entendu par son chemin
d'accès*/
86 IF fichier1 ~= '' THEN nombre1= '('
WORD(STATEF(fichier1),2) ')'
87 ELSE nombre1= ''
88 IF fichier2 ~= '' THEN nombre2= '('
WORD(STATEF(fichier2),2) ')'
89 ELSE nombre2= ''
90 IF nombre1 ~= '' THEN a1=STRIP(STRIP(nombre1,'b','()'))
91 IF nombre2 ~= '' THEN a2=STRIP(STRIP(nombre2,'b','()'))
92 /*calcul des cas particuliers selon l'existence des
fichiers*/
93 SELECT
94 WHEN a1 ~= '' & a2 ~= '' THEN soustotal=soustotal+a1+a2
95 WHEN a1 ~= '' & a2 = '' THEN soustotal=soustotal+a1
96 WHEN a1 = '' & a2 ~= '' THEN soustotal=soustotal+a2
97 WHEN a1 = '' & a2 = '' THEN soustotal=soustotal
98 END /*pas de sortie OTHERWISE ici non nécessaire c'est
imprudent*/
99 l1=OVERLAY(fichier1,d,10) /*OVERLAY surimpose des
chaines en lieux et tailles bien déterminés*/
100 l1=OVERLAY(nombre1,l1,50)
101 CALL imprime l1
102 l2=OVERLAY(fichier2,d,10)
103 l2=OVERLAY(nombre2,l2,50)
104
file=STRIP(file '|||'STRIP(fichier1)STRIP(nombre1)STRIP(fichier2)S
TRIP(nombre2)'|||')
105 CALL imprime l2
106 taquet =1
107 fichier1= ''
108 fichier2= ''
109 nombre1= ''
110 nombre2= ''
111 posancien=pos
112
113 RETURN /*fin du sous-programme*/
114
115
116 imprime:
117
118 ARG texte /*récupère la chaine transmise à l'appel de ce
sous-programme*/
119
120 SAY texte
121 RETURN
122
123 traitement: /* création de la chaine résultat*/
124
125
126
127
128 l3=OVERLAY(soustotal,d,48)
129 l3=OVERLAY('SOUS-TOTAL=',l3,37)
130 CALL imprime l3
131 total=total+soustotal
132 l3=OVERLAY(total,d,48)
133 l3=OVERLAY(' TOTAL=',l3,37)
134 file=STRIP(file '///soustotal','total'///)
135 CALL imprime l3
136
137 RETURN

```

Par F. Gueugnon

Contrairement à ce que voudrait la théorie, le programmeur moyen sur des machines telles que l'Amiga passe la moitié de son temps à taper des lignes de code et le reste à essayer d'en faire quelque chose d'utilisable.

Pour assister les pauvres développeurs que nous sommes, la société Arobas va bientôt mettre sur le marché un nouveau débogueur : Adebug, version Amiga d'un programme Atari, qui laisse loin derrière lui les différents outils déjà existant... Les débogueurs que l'on trouve sur Amiga ne sont pas très nombreux, les principaux étant Monam2 du Devpac, Metascope, AsmOne (éditeur, compilateur, débogueur), ainsi que des moniteurs intégrés à de petits assembleurs (K-Seka et autres mutants qui en découlent) ou des petites brutes écrites par des traumatisés du mode trace (RomCrack et autres).

Ils se répartissent en deux principaux groupes : ceux qui sont évolués et conservent le multitâche et Intuition (Monam2, Metascope, ou même Profimat) et ceux qui apprécient le calme d'un ordinateur débarrassé de son système d'exploitation. Adebug se situe à l'intersection de ces deux ensembles puisqu'il élimine le multitâche lorsque l'on utilise son écran de contrôle, mais remet la configuration courante (niveau IPL, registres hardware...) lorsqu'on désire tracer des instructions. Enfin faisant office de relique, K-Seka est décidément positionné sur le marché de l'assembleur pour jeunes...

PRESENTATION

Adebug se démarque comme un utilitaire externe, contrairement à Monam, AsmOne, Profimat et K-Seka qui sont inclus dans l'éditeur-assembleur. Une fois le lancement effectué, le programme va chercher sur le disque les préférences définies lors de la cession précédente et après quelques flash blancs (épileptiques s'abstenir) on se retrouve avec un écran noir rappelant celui de Monam2. Il faut bien avouer que le premier contact avec ce genre de débogueur n'est pas en général en sa faveur, puisqu'il faut abandonner le doux confort d'Intuition pour l'aridité d'un éditeur avec ligne de commande. On se retrouve en fait avec une gestion clavier du type de celle du Shell, avec possibilité de remonter l'historique des commandes déjà exécutées. Mais ceci ne doit en aucun cas décourager le lecteur, la majorité des commandes ne nécessitant l'appui que d'une ou deux touches (les mêmes que Monam2, d'ailleurs), ce qui est, après un court apprentissage, extrêmement pratique et rapide.

Finalement que demande-t-on à l'interface d'un débogueur ? C'est de permettre une utilisation efficace et rapide des opérations les plus courantes, ce qui n'est sûrement pas le cas d'interfaces plus visuelles à base de fenêtres et de menus déroulants. L'utilisation exclusive du clavier n'est pas un handicap pour un logiciel destiné aux programmeurs qui, jusqu'à preuve du contraire, tapent leurs listings manuellement.

L'aspect de l'écran se rapproche fortement de celui de Monam2, dont le look caractéristique est basé sur la division de l'écran en zones (jusqu'à 5 fenêtres de taille variable) auxquelles sont associées un type d'information précis (listage des registres, désassemblage, dump mémoire hexa et ASCII). On peut sélectionner la fenêtre sur laquelle on compte agir grâce à la

touche TAB et faire agir dessus diverses commandes d'édition.

Les informations fournies par le système de fenêtrage sont suffisamment complètes pour permettre des débogages relativement complexes. La zone d'affichage des registres montre non seulement les valeurs de ces derniers, mais aussi le contenu des adresses pointées par ceux-ci, ce qui épargne au programmeur de nombreux dumps-mémoire. La fenêtre de désassemblage ne se contente pas non plus d'un simple affichage de code assembleur mais grâce à de petits dessins, indique la position du PC, l'endroit où il se retrouvera après l'exécution d'une instruction de branchement, ainsi que les différents points d'arrêt.

Adebug est un débogueur symbolique, ce qui veut dire que les lignes désassemblées utilisent au maximum les étiquettes définies dans le code source, ainsi que celles que l'on peut définir dans des fichiers de définition de variables. Il n'est plus étonnant dans ces conditions de voir des programmes en ROM avec leurs labels. Un petit détail qui a son importance : quand on appuie sur la touche de curseur haut, le pointeur de début de fenêtre remonte d'une ligne désassemblée et non de deux octets comme c'est le cas dans certains débogueurs moins soucieux du confort de l'utilisateur. Il faut noter que toutes les sorties graphiques sont dynamiques, c'est-à-dire qu'elles sont réactualisées en permanence. Adebug peut de plus rediriger son affichage sur un Minitel (mode 80 colonnes), permettant donc d'afficher à la fois les données du débogueur et l'écran du programme.

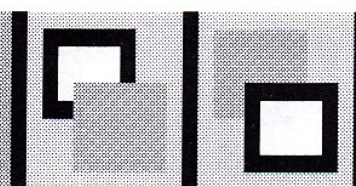
Le travail est facilité par un certain nombre de raccourcis clavier. Par l'appui d'une touche on peut demander de voir le contenu de la mémoire pointée par n'importe quel registre (a0 à a6, sp, ssp et pc). On peut aussi définir une macro (oui, une !), sorte de script contenant les commandes tapées au clavier. Ce système est indispensable, lors de débogages répétitifs, à la santé mentale du pauvre codeur...

LE TRAVAIL AVEC ADEBUG

Bien que cette interface soit relativement perfectionnée, le degré de "bestialité" du programme tracé n'influence en aucun cas le confort d'utilisation. Ainsi, un programme tracé en niveau d'IPL 7 se présentera de la même manière qu'une routine Intuition (bien évidemment utilisée en IPL 0) de façon à conserver les interruptions du multitâche. Adebug permet de choisir de cette manière un niveau de résistance aux agressions diverses, telles que celles que l'on a l'habitude de rencontrer en traçant une démo (rip, rip) ou un jeu (crack, crack).

Mais Adebug n'est pas une forteresse coupée du monde extérieur, il permet de voir la tâche se dérouler avec son propre écran si celle-ci a une CopperList. Il suffit d'en spécifier le label et Adebug se charge de l'afficher à chaque traçage d'instruction.

La fenêtre active est celle sur laquelle on va faire agir des fonctions. Ces dernières sont plus ou moins classiques : on retrouve tout d'abord les habituels copie de bloc mémoire, remplissage, édition... Le principal défaut des fonctions d'édition est l'absence d'assembleur-ligne, qui aurait par exemple permis de remplacer directement un SUBQ.W #1,d0 par un NOP, au lieu de passer par l'éditeur du dump hexadécimal. Par contre, la fonction de recherche est très complète puisqu'elle prend en charge plusieurs types de données : octets, mots, mots longs, chaînes ASCII et instructions (par exemple, chercher un LEA \$70000,a0 dans un code objet de VcC).



Que ce soient des labels ou des valeurs définies d'une autre manière, Adebug considère toute variable de la même manière, et ceci n'est rendu possible que par l'existence d'un puissant évaluateur d'expressions, qui constitue la principale innovation de Adebug puisque sa seule présence prend en charge une bonne partie de la tâche qui incombait jusque là au programmeur, à savoir la prise de décisions. Nous verrons dans la suite à quels endroits précis ce système démontre clairement sa puissance.

L'évaluateur possède une syntaxe classique qui ne dépaysera pas le programmeur habitué à un langage évolué quelconque. Il permet de faire des opérations arithmétiques (+, -, *, /), logiques (EOR, AND, OR, EOR), des décalages (LSL, LSR) sur des valeurs numériques ou sur des variables. Ces dernières sont soit définies par l'utilisateur, soit réservées à l'avance par Adebug. En effet, une grande partie des variables internes sont accessibles. Les adresses des points d'arrêt et de début des fenêtres, le contenu des registres... ont des noms réservés. De plus, une fonction autorise la lecture ou l'écriture en mémoire d'octets, de mots ou de mots longs. Certaines variables sont en fait des expressions à évaluer à chaque pas de programme qui est tracé. Il devient aisé de faire suivre à une fenêtre le contenu d'un registre.

Un exemple d'expression serait $w2=4*d0+\{TableBase\}.L$, ce qui revient à demander de positionner la deuxième fenêtre sur un mot long d'une table dont le numéro est fourni par le registre d0. On réalise aisément l'aide réelle apportée par un évaluateur puissant associé à une gestion correcte des variables.

Le mode trace autour duquel est construit tout débogueur qui se respecte peut se voir grandement amélioré par quelques fonctions peu courantes. Tout d'abord, Adebug trace sans problème la ROM de l'Amiga grâce à une recopie automatique en RAM des instructions. Il est tout aussi capable de tracer une routine de gestion de mode trace. Pour ce qui est des aides à la vie courante, Adebug émule le fameux mode trace du 68020 qui ne s'arrête que sur les instructions interrompant le flux (branchements conditionnels ou non). Il peut aussi sauter une instruction ou même forcer un branchement. Tout ceci est consigné dans un petit historique, tenu à jour en permanence.

Mais la grande nouveauté est le traçage conditionnel, qui fait appel à l'évaluateur d'expression. On peut avec ce dernier demander à Adebug de tracer un programme jusqu'à ce qu'une condition spécifiée (par exemple $d0=10$ et $a0<>5$) devienne vraie. Evidemment, ce système est très gourmand en temps-machine puisqu'il faut évaluer l'expression à chaque instruction exécutée. Heureusement, une technique plus rapide est mise à notre disposition grâce aux points d'arrêts conditionnels. Rappelons qu'un point d'arrêt est une sorte de marque que le débogueur positionne à l'endroit spécifié par l'utilisateur de façon à récupérer la main quand le microprocesseur "passe" dessus. Cette marque est en général une instruction du type Line A ou Trap (selon les besoins du moment) qui bloque le 68000 et remet en route Adebug. Jusque là, les points d'arrêts bloquaient à coup sûr le programme, mais avec l'évaluateur d'expressions, ceux-ci peuvent être associés à une condition. Le point d'arrêt et la condition seront affichés en permanence dans les fenêtres de désassemblage.

Une autre fonction ouvre de nouveaux horizons dans l'art de la recherche d'erreurs. En effet, des routines externes peuvent être exécutées par Adebug à chaque pas du mode trace. Ces routines, créées par chaque utilisateur, reçoivent comme paramètre une structure reflétant l'état du débogueur et de la tâche tracée. On retrouve bien entendu toutes les valeurs des registres mais aussi les adresses des fenêtres.

ET ALORS ?

Finalement, le principe général d'Adebug est de répondre point par point aux besoins des programmeurs. Ces besoins ne peuvent être recensés qu'à l'usage du programme, et le fait qu'Adebug vienne tout droit du monde Atari lui permet de bénéficier d'une plus grande maturité dès sa sortie. Espérons que ce logiciel saura évoluer dans le sens d'une plus grande simplicité d'emploi, par

1/REGS															
D0:*00000001	↑	A0:*00C06634	0A30	2499	FFFF	FFFF	0030	1960	00C0	65E8	IPL0				
D1:*00C06634	AF4	A1:*00C06740	0030	E121	0000	03E8	0030	1980	0000	0001	INTS				
D2:*00000FA0	↓	A2:*00C05F98	0000	0096	00FF	9008	00FF	495E	00FF	4A90	603C				
D3:*00000FA8	↓	A3:*00C38484	0030	2D6D	6008	202D	4172	662E	202D	48E7	MC:				
D4:*00000001	↑	A4:*00C4F8A0	0000	0FA0	00C0	69C8	00C3	8488	00C5	48C0	AA:N				
D5:*0000003E	↓	A5:*00FF4134	265F	48F1	1A00	08F4	D3C0	48D1	001E	4ED4	AB:N				
D6:*0030368D	06%	A6:*00FF4128	4CE9	0A00	FFF4	2869	FFFC	4ED3	265F	48F1	BA:N				
D7:*00C0634C	AcL	A7:*00C4F89C	00FF	492E	0000	0FA0	00C0	69C8	00C3	8488	BB:N				
SR:*0000	0	SSP:*00C80000	002E	03D0	8000	3EA7	FFFF	2D56	0404	F101					
PC:*00C00276	DC.W	\$C0													
2/DIS								3/HEXA							
00C00276	↑	DC.W	\$C0					00C00276	00C0	1916	A96				
00C00278		MOVE.B (A6),-(A4)						00C0027A	00C0	03F0	A08				
00C0027A		DC.W	\$C0					00C0027E	0900	00FC	0	u			
00C0027C		DC.W	\$3F0					00C00282	00A8	0400	0	u			
00C0027E		BTST D4,D0						00C00286	0276	024C	0	u			
00C00280		DC.W	\$FC					00C0028A	0022	0002	0	u			
00C00282		ORI.L #54000276,\$24C(A0)						00C0028E	00FC	0018	u	8			
4/ASC								5/HEXA							
A96								00C00276	00C0	1916	A96				
A08								00C0027A	00C0	03F0	A08				
u								00C0027E	0900	00FC	0	u			
0								00C00282	00A8	0400	0	u			
0								00C00286	0276	024C	0	u			
0								00C0028A	0022	0002	0	u			
u								00C0028E	00FC	0018	u	8			
u								00C00292	672D	0000	9-				

l'adjonction d'un certain nombre de petites fonctions qui ne seraient pas du luxe : un sélecteur de fichiers, un classement alphabétique des labels ou un affichage des préférences sur un panneau de contrôle.

On pourra, plus sérieusement, regretter l'absence de version cartouche telle qu'elle existe sur le ST. Un débogueur de ce type serait d'une efficacité redoutable s'il n'avait plus besoins de cohabiter en mémoire avec les programmes erronés.

Adebug est donc un excellent programme qui s'adressera à des programmeurs ayant une bonne connaissance du langage assembleur et qui se sont déjà retrouvés confrontés aux insuffisances de leur débogueur habituel. La configuration nécessaire au déploiement de ce char d'assaut aura tout intérêt à comporter une extension mémoire, la taille du programme étant actuellement de 97600 octets !

Par Eric Brunet et François Fleuret



SUBWAY: LES LAYERS (II)

Puisque vous savez maintenant tout ou presque sur les layers, il est temps de passer à la seconde partie de notre découverte de ces petites bêtes sympathique et d'aborder une autre notion tout autant mystérieuse : les régions.

Mais avant que de commencer, il semble important d'apporter quelques précisions à l'article du mois dernier, le mécanisme de verrouillage des layers et de la layer_info ayant été odieusement passé sous silence, alors qu'il s'agit d'une particularité fort intéressante. En vous présentant toutes mes excuses, je vous invite donc à un petit détour par le monde étrange et pénétrant des Sémaphores...

C'EST MA FAURE

De quoi s'agit-il Auguste ? Tout simplement et en simplifiant quelque peu, de permettre à une tâche donnée d'empêcher une autre tâche quelconque d'accéder à une zone de mémoire particulière. Bref, c'est à un véritable "verrou de la mémoire" que nous avons à faire. Ce verrou étant purement logiciel (aucun matériel hardware n'intervenant dans l'histoire), cela suppose évidemment que la seconde tâche soit disciplinée et accepte de se soumettre à ce procédé. Ce qui est souvent le cas, parfois même à son nez et à sa barbe. Les exemples les plus frappants étant d'une part la structure IntuitionBase, que l'on peut verrouiller en appelant la fonction LockIBase() et, donc, nos layers.

La principale utilité des sémaphores est donc d'arbitrer l'accès à la mémoire entre les différentes tâches. Imaginez par exemple que vous soyez en train d'ajouter un noeud dans une liste publique, tandis qu'au même moment, une autre tâche parcourt cette même liste à la recherche d'un autre noeud ; il y a alors une chance sur deux pour que l'insertion de votre noeud fasse se mélanger les pinces à l'autre tâche. Le cas est encore pire si deux tâches essaient chacune d'ajouter un noeud à une même liste en même temps. Dans tous ces cas, interdire le multitâche avec Fobrid() ou même Disable() ne suffit plus, étant donné que l'autre tâche peut avoir déjà commencé son travail avant que vous ne lui coupiez les vivres. Bien sûr, lorsqu'il s'agit de données "statiques" (non modifiables) comme la Rom, un mécanisme de verrouillage devient superflu.

Les sémaphores sont entièrement basés sur le double système des listes et des signaux. Lorsqu'une tâche a obtenu avec succès un sémaphore, toutes les autres tâches qui essaieraient d'y accéder sont mises en état d'attente (avec Wait()) jusqu'à ce que celui-ci soit libéré. De cette manière, on est sûr que deux tâches n'accéderont jamais aux mêmes données en même temps.

Si le sujet vous intéresse, nous le décrirons sans doute plus avant dans ces mêmes pages, mais pour l'heure, il est temps de revenir à nos moutons, les layers, que l'on peut donc verrouiller.

LA CLEF DES CHAMPS

Cela se fait très simplement avec les fonctions adéquates de la layers.library, qui se charge toute seule comme une grande de l'interface avec les sémaphores, nous libérant de tout ce travail. Il est ainsi possible de verrouiller soit un layer particulier, soit une layer_info complète, et donc tous les layers qui en dépendent. Par exemple, Intuition verrouille la layer_info de l'écran concerné lorsque vous déplacez ou agrandissez une fenêtre, ou encore lorsque vous déroulez un menu. Par autre exemple, le Workbench verrouille la layer_info de son écran lorsque vous déplacez une icône.

Voici listées pour vous et dans le tableau ci-dessous, les fameuses "fonctions adéquates" dont il est question dans le paragraphe précédent. Rendez-vous tout de suite après pour les explications de rigueur.

LockLayer()	verrouille un layer donné
UnlockLayer()	Déverrouille ledit layer
LockLayers()	verrouille les layers d'une layer_info
UnlockLayers()	Déverrouille ces même layers
LockLayerInfo()	verrouille la layer_info elle-même
UnlockLayerInfo()	Déverrouille ladite layer_info

Avec LockLayer(), vous verrouillez un layer donné afin qu'aucune autre tâche ne puisse y effectuer de sortie graphique en même temps que vous. Rappelons que ce verrouillage étant basé sur les sémaphores, votre tâche sera mise en état d'attente si d'aventure le layer concerné était déjà verrouillé par une autre tâche. Si vous désirez verrouiller plusieurs layers dans un même écran mais pas tous, il vous faudra entourer tous les appels à LockLayer() par une paire de LockLayerInfo() et UnlockLayerInfo(), ceci afin d'éviter un blocage intempestif du système.

LockLayers() vous permet de verrouiller en une seule fois tous les layers d'un écran donné, sans toucher à la structure layer_info associée. De fait, une autre tâche peut toujours la modifier, même si les layers eux-mêmes sont protégés.

Enfin, LockLayerInfo() est la plus puissante des trois fonctions de verrouillage, puisque non contente de bloquer tous les layers de l'écran, elle verrouille également la layer_info elle-même. De cette manière, vous êtes sûr que rien d'imprévu ne peut arriver à vos layers pendant que vous les travaillez au corps à corps.

Ces trois fonctions possèdent chacune leur vis-à-vis, qu'il convient d'appeler absolument lorsque le travail désiré a été effectué : n'oubliez pas que d'autres tâches peuvent attendre d'avoir le droit d'accéder au(x) layer(s) verrouillé(s).

Dernier point intéressant à noter, le fait que la graphics.library dispose de deux fonctions similaires à LockLayer() et UnlockLayer(), s'appellant respectivement LockLayerRom() et UnlockLayerRom() et qui peuvent donc permettre, dans certains cas ma foi assez limités, d'éviter d'avoir à ouvrir la layers.library.

LES REGIONS

Cette fois-ci, tout ce qui pouvait être dit sur les layers l'a été, nous pouvons donc aborder sans plus attendre ce nouveau concept qu'est la Région. La layers.library est maintenant oubliée, c'est la graphics.library qui se charge de tout (à une exception près, que nous verrons plus loin). Franchement, je n'ai jamais compris pourquoi elle ne se chargeait pas également des layers, mais bon, hein, c'est pas moi qui ai écrit le système (Dieu merci !).

Tout comme les layers, les régions font intervenir la notion de *rectangle de clipping*. Le principe est pourtant assez simple : alors que les layers servent à limiter les sorties graphiques à une zone particulière d'une BitMap, les régions servent à les limiter à une zone particulière d'un layer. Pourquoi donc se compliquer ainsi la vie ? N'oubliez pas que les layers peuvent se chevaucher entre eux, changer de taille, être déplacés... L'utilité des régions réside donc dans la capacité qu'elles sont de limiter les sorties graphiques aux parties visibles d'un layer donné.

Lorsque plusieurs layers se chevauchent, le système maintient et met à jour une liste des régions perdues, appelée la DamageList. Ainsi, plus tard, lorsque l'application concernée voudra rafraîchir sa fenêtre, le système limitera les sorties graphiques aux régions contenues dans cette liste, accélérant d'autant l'opération. Ceci est tout-à-fait transparent pour l'application, qui se contente de redessiner "bêtement" le layer entier.

La graphics.library met à notre disposition plusieurs routines de création et de gestion des régions au sein d'un layer. Celles-ci

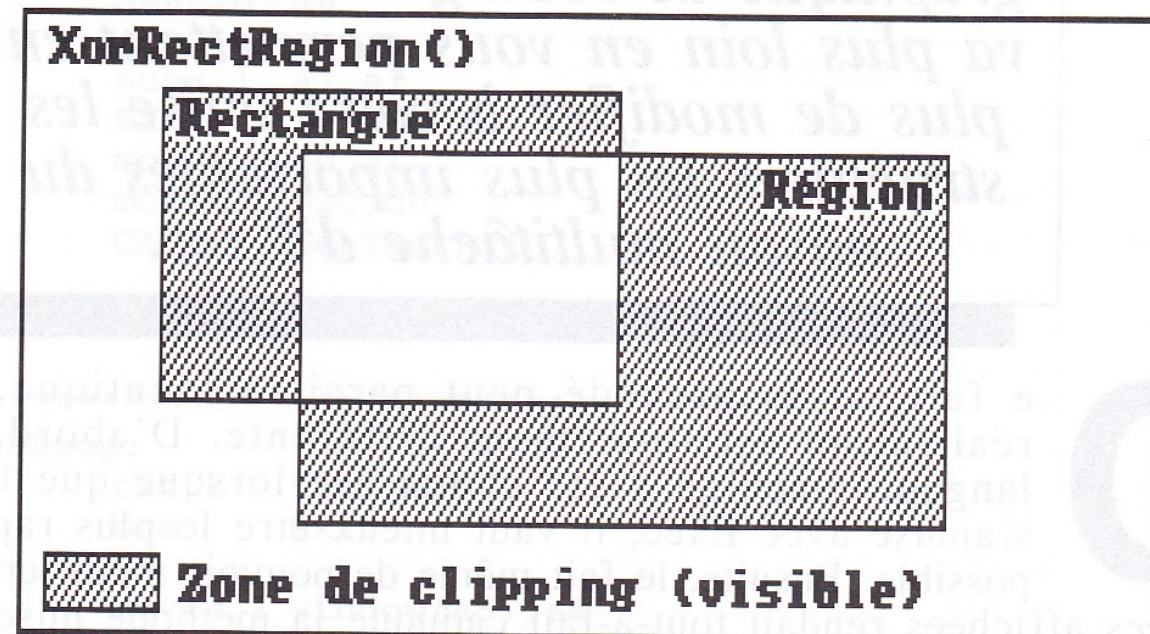
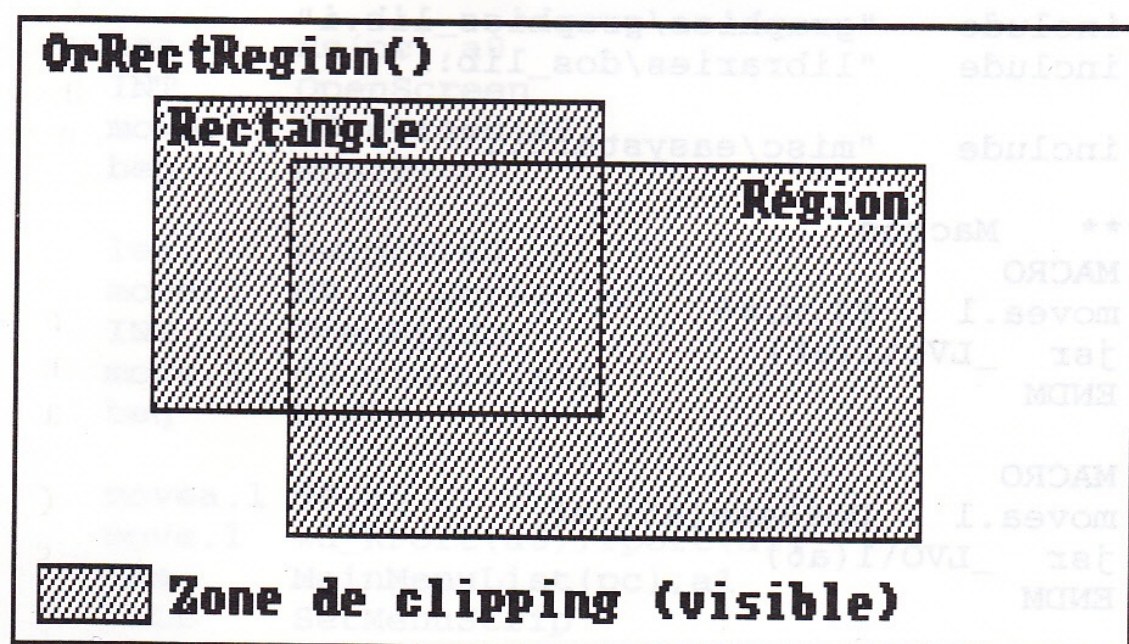
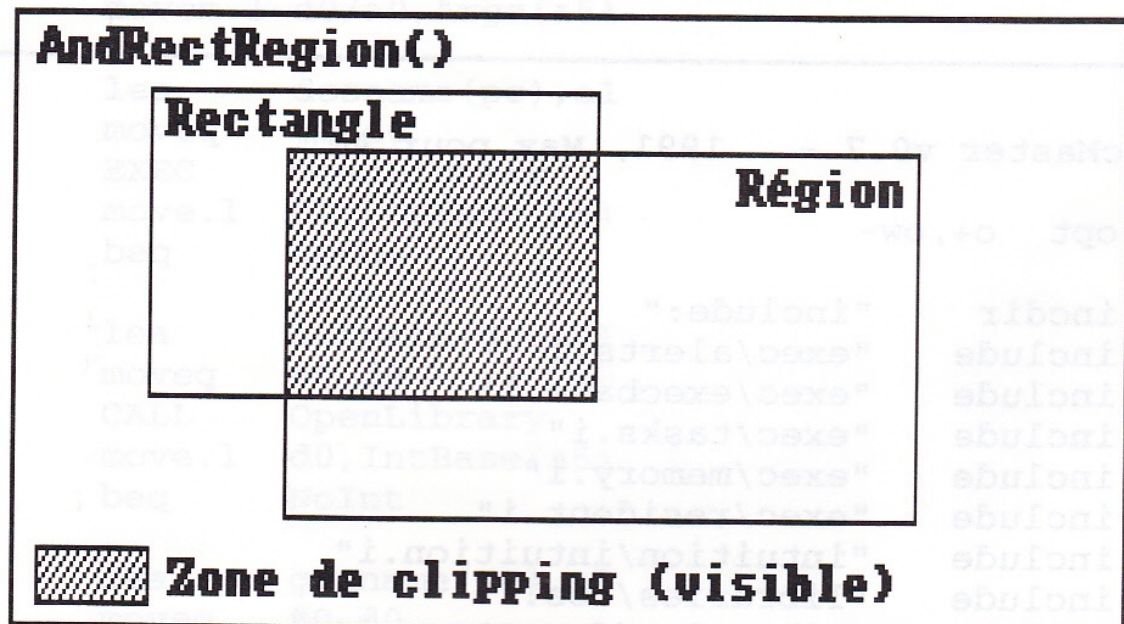
sont résumées dans le tableau ci-dessous, que je vous laisse examiner avant de continuer.

<code>NewRegion()</code>	Crée une nouvelle région dans le layer
<code>DisposeRegion()</code>	Libère ladite région
<code>ClearRegion()</code>	Réinitialise une région
<code>AndRectRegion()</code>	
<code>OrRectRegion()</code>	Gestion des régions conjointement avec des Rectangles
<code>XorRectRegion()</code>	
<code>ClearRectRegion()</code>	
<code>AndRegionRegion()</code>	
<code>OrRegionRegion()</code>	Gestion des régions conjointement avec d'autres régions
<code>XorRegionRegion()</code>	

Enfin, la `layers.library` offre une seule fonction de gestion des régions, à savoir `InstallClipRegion()`, qui installe effectivement la région dans le layer. Tout cela semble un peu bordélique, mais c'est finalement assez logique.

Avec `NewRegion()`, on alloue dynamiquement et initialise une structure `Region` que l'on pourra par la suite lier au layer. Cette région est "vierge", c'est-à-dire qu'elle ne permet pas encore de sortie graphique. L'allocation étant dynamique, la fonction `DisposeRegion()` sert à libérer la mémoire utilisée.

Les fonctions suivantes, `AndRectRegion()`, `OrRectRegion()` et `XorRectRegion()` servent à mettre en place la région. Elles utilisent une structure supplémentaire, connue sous le nom de structure `Rectangle`, qui définit les coordonnées de la zone de clipping effective. Ce n'est qu'une fois l'une de ces trois fonctions appelée que la région sera opérative et pourra être installée dans le layer avec `InstallClipRegion()`. La différence entre ces trois fonctions tient dans la manière dont elles calculent la zone effectivement visible : `AndRectRegion()` limite la sortie aux parties du layer visibles à la fois dans le `Rectangle` et dans la `Région` ; `OrRectRegion()` étend la `Région` aux parties visibles dans le `Rectangle` ; `XorRectRegion()` limite la sortie aux parties visibles dans le `Rectangle` ou dans la `Région`, mais pas dans les deux. Arrivé ici, un schéma sera peut-être plus clair.



Enfin, `ClearRectRegion()` supprime un `Rectangle` donné de la région considérée.

Quant aux trois dernières fonctions, à savoir `AndRegionRegion()`, `OrRegionRegion()` et `XorRegionRegion()`, elles agissent de la même manière que les précédentes, mais jonglent avec deux régions plutôt qu'avec une région et un rectangle.

Finalement, lorsque vous avez décidé et correctement initialisé votre `Région`, vous l'installez dans le layer avec `InstallClipRegion()` - qui, rappelons-le encore une fois, fait partie de la `layers.library` - et enfin, toutes les sorties graphiques dans ce layer seront limitées à cette région.

L'EXEMPLE

Le programme qui suit est l'exemple d'utilisation des layers en dehors d'Intuition, promis le mois dernier. Pour les régions, c'est un peu plus difficile de réaliser un programme dont l'effet visuel doit évident... C'est pourquoi je vous propose à la place une petite routine, d'ailleurs adaptée de celle fournie dans les *RKM Libraries & Devices 1.3*, pages 516 et 517, qui permet de clipper une fenêtre Intuition à ses bords (ce qu'Intuition ne fait pas elle-même tout seule ; si vous avez déjà essayé de dessiner en dehors d'une fenêtre, vous aurez pu remarquer que les bords font partie intégrante du layer, sauf évidemment dans le cas d'une fenêtre de type GIMMEZEROZERO, mais ça, c'est une autre histoire).

Par Max

Hé, pssiittt...
On a failli oublier de
vous rappeler que le
3615 ANT
et le 3615 ComRev
vous attendent 24
heures sur 24 pour
dialoguer en direct,
télécharger, jouer...

UTILITAIRE: ExecMaster

Avec l'Espion de la Mémoire, vous avez pu explorer tout le système graphique de l'Amiga. ExecMaster va plus loin en vous permettant en plus de modifier à votre guise les structures les plus importantes du noyau multitâche d'Exec.

De fait, si le procédé peut paraître identique, la réalisation est totalement différente. D'abord, le langage machine a été choisi car lorsque que l'on s'amuse avec Exec, il vaut mieux être le plus rapide possible. Ensuite, le fait même de pouvoir modifier les données affichées rendait tout-à-fait caduque la méthode mise en oeuvre dans l'Utilitaire de l'ANT numéro 25.

ExecMaster vous est proposé ici dans sa version 0.7, c'est-à-dire qu'il s'agit quasiment de ce que les professionnels de l'informatique nomment une *bêta-version*. Le listing est déjà suffisamment conséquent comme ça (jugez par vous-même...), à tel point d'ailleurs que nous avons choisi de le publier en deux fois. Vous trouverez donc ce mois-ci le programme principal, qui vous permettra au moins de voir comment tout cela fonctionne, et le mois prochain, les définitions de l'écran, des fenêtres, des gadgets et des menus utilisés.

PRESENTATION

Le menu principal Affichage propose de visualiser les différentes tâches en service dans le système, les bibliothèques, devices et ressources présents en mémoire, les ports publics connus d'Exec, les modules résidents, les listes-mémoires et les interruptions actives. Toutes ces données ne sont pas obligatoirement modifiables, aussi un second menu apparaît-il en fonction de ce qui est en cours de visualisation.

ExecMaster permet de changer la priorité d'une tâche, de la "geler" provisoirement (c'est-à-dire de suspendre son exécution) et de la "réchauffer" (la ré-introduire dans la liste des tâches candidates à l'utilisation du CPU). Vous pouvez également ouvrir et fermer une bibliothèque particulière, ou éliminer de la mémoire toutes celles qui ne sont pas présentement utilisées. Quant aux ports, vous pouvez les "cacher" (les supprimer de la liste des ports publics, mais sans les détruire) ou bien les "révéler" (les ré-introduire dans la liste). Enfin, vous pouvez changer la priorité d'un type de mémoire donné, par exemple pour faire en sorte que le système alloue de la mémoire CHIP en priorité. Tout cela s'effectue à la souris, via les menus adéquats.

PRINCIPE

Dans tous les cas, étant donné que l'on parcourt des listes publiques modifiables par une autre tâche à n'importe quel instant, il convient d'interdire d'abord le multitâche et même carrément les interruptions, de lire les données de la liste en cours et de les copier dans un buffer supplémentaire, pour finalement les afficher après avoir rétabli le multitâche. Une routine, *AffList*, se charge de tout cela : elle parcourt la liste pointée par a2 jusqu'à sa fin, en appelant à chaque itération la routine pointée par a3 et chargée de mettre en forme les données trouvées. *AffList* est appelée quasiment par toutes les routines d'affichage, sauf celles qui ne dépendent pas d'une liste (au sens Exec du terme), comme par exemple les interruptions ou les modules résidents.

Geler une tâche est très simple : il suffit de la retirer de sa liste actuelle (*Ready* ou *Waiting*) et de l'insérer dans une autre, interne au programme. Ainsi, Exec ne sait plus que cette tâche existe et elle "s'arrête" tout bonnement de tourner. Le processus inverse est utilisé pour la "réchauffer". Notez qu'ExecMaster refuse de se

geler lui-même (c'est la moindre des choses !) et qu'il ne vérifie pas, en quittant, si des tâches sont encore gelées. Faites donc attention.

Ouvrir et fermer une bibliothèque particulière ne pose aucun problème, on peut peut-être seulement se demander comment la virer de la mémoire si elle n'est pas actuellement utilisée. Très simple : il suffit d'essayer d'allouer un très grosse quantité de mémoire, et le tour est joué ; Exec supprime alors tout ce qui peut l'être : bibliothèques, devices, ressources, fontes...

Pour "cacher" et "révéler" un port, on procède de la même manière que pour geler et réchauffer une tâche : on le supprime de la liste des ports publics pour l'insérer dans une autre, interne au programme et inversement. Cela ne détruit pas le port lui-même, même si *FindPort()* ne peut plus le trouver. Là encore, ExecMaster ne fait aucune vérification quant au nombre de ports encore cachés avant de quitter, donc prudence...

AMELIORATIONS

Beaucoup sont possibles (je vous rappelle que ce n'est là que la version 0.7), comme par exemple terminer définitivement une tâche donnée (gare aux allocations mémoire qui ne seront pas libérées) ou carrément la désassembler depuis son PC actuel... On peut également envoyer des signaux et/ou des messages aux ports, lister d'autres structures (input-handlers, vecteurs Reset...) avec toutes les applications qui en découlent. L'imagination est la seule limite aux possibilités d'un tel programme (sans me vanter).

A SIGNALER

... que l'idée d'ExecMaster ne vient pas de moi, mais de l'allemand *Werner Günther*, auteur du domaine public *Xoper* dont la version 2.2 est un véritable petit bijou (moins évident à utiliser car entièrement au clavier) que tout programmeur se doit de posséder. D'ailleurs, je vous l'ai glissée quelque part sur la disquette d'accompagnement de ce numéro de l'ANT. Merci qui ?

```
;
; ExecMaster v0.7 - 1991, Max pour ANT
;

opt  o+,ow-

incdir      "include:"
include     "exec/alerts.i"
include     "exec/execbase.i"
include     "exec/tasks.i"
include     "exec/memory.i"
include     "exec/resident.i"
include     "intuition/intuition.i"
include     "libraries/dos.i"
include     "libraries/dosexterns.i"

include     "exec/exec_lib.i"
include     "intuition/intuition_lib.i"
include     "graphics/graphics_lib.i"
include     "libraries/dos_lib.i"

include     "misc/easystart.i"

; ***** Macros
EXEC MACRO
movea.l     $4.w,a6
jsr         _LVO\1(a6)
ENDM

DOS MACRO
movea.l     DosBase(a5),a6
jsr         _LVO\1(a6)
ENDM

INT MACRO
```



```

        movea.l IntBase(a5),a6
        jsr     _LVO\1(a6)
        ENDM

GFX      MACRO
        movea.l GfxBase(a5),a6
        jsr     _LVO\1(a6)
        ENDM

CALL     MACRO
        jsr     _LVO\1(a6)
        ENDM

; *****
        rsreset
XItem    rs.b    0          ; Structure MenuItem étendue
xi_Item  rs.b    mi_SIZEOF  ; Item standard
xi_Func   rs.l    1          ; Fonction à exécuter ou NULL
xi_SIZE   rs.w    0

        rsreset
Ligne    rs.b    0
lg_Num    rs.w    1          ; Numéro de la ligne
lg_Data   rs.b    38         ; Données pour RawDoFmt()
lg_Str     rs.b    80         ; Buffer ASCII pour
RawDoFmt()
lg_SIZE   rs.b    0

        rsreset
Args      rs.l    2
WBmsg     rs.l    1
DosBase   rs.l    1
IntBase   rs.l    1
GfxBase   rs.l    1
screen    rs.l    1
window    rs.l    1
rport     rs.l    1
tasks     rs.b    LH_SIZE    ; Liste des tâches "gelées"
ports     rs.b    LH_SIZE    ; Liste des ports "cachés"
ligne_y   rs.w    1
lignes    rs.w    1
fini      rs.w    1
VARSIZE   rs.w    0

; *****
Start     lea     VARS(pc),a5
        movem.l d0/a0,Args(a5)

        lea     dosname(pc),a1
        moveq   #0,d0
        EXEC    OpenLibrary
        move.l  d0,DosBase(a5)
        beq     NoDos

        lea     intname(pc),a1
        moveq   #0,d0
        CALL    OpenLibrary
        move.l  d0,IntBase(a5)
        beq     NoInt

        lea     gfxname(pc),a1
        moveq   #0,d0
        CALL    OpenLibrary
        move.l  d0,GfxBase(a5)
        beq     NoGfx

        lea     ns(pc),a0
        INT     OpenScreen
        move.l  d0,screen(a5)
        beq     NoScr

        lea     nw(pc),a0
        move.l  d0,nw_Screen(a0)
        INT     OpenWindow
        move.l  d0>window(a5)
        beq     NoWin

        movea.l d0,a0
        move.l  wd_RPort(a0),rport(a5)
        lea     MainMenuList(pc),a1
        CALL    SetMenuStrip

        lea     tasks(a5),a0

```

```

        lea     ports(a5),a0
        NEWLIST a0

        suba.l  a1,a1
        EXEC    FindTask
        movea.l d0,a1
        moveq   #20,d0
        CALL    SetTaskPri

        sf      fini(a5)

; *****
MainLoop:
        tst.b   fini(a5)
        bne     Exit

        movea.l window(a5),a0
        movea.l wd_UserPort(a0),a0
        EXEC    WaitPort

AllMsgs   movea.l window(a5),a0
        movea.l wd_UserPort(a0),a0
        EXEC    GetMsg
        tst.l   d0
        beq.s   MainLoop

        movea.l d0,a1
        move.l  im_Class(a1),d3
        move.w  im_Code(a1),d2
        CALL    ReplyMsg

        cmpi.l  #MENU_PICK,d3
        bne.s   AllMsgs

DoMenus   moveq   #0,d0
        move.w  d2,d0
        lea     MainMenuList(pc),a0
        INT     ItemAddress
        tst.l   d0
        beq     AllMsgs
        movea.l d0,a1
        move.l  xi_Func(a1),d0
        beq.s   .nofunc
        movea.l d0,a0

        movem.l d0-d7/a0-a6,-(sp) ; Sauve les registres
        movea.l $4.w,a6           ; Prépare a6
        lea     txtbuff,a4        ; et a4
        jsr     (a0)              ; Saute à la routine
        movem.l (sp)+,d0-d7/a0-a6 ; Récupère les
registres

        .nofunc move.w  mi_NextSelect(a1),d2
        bra.s   DoMenus

; *****
Exit      move.l  window(a5),d0
        beq.s   .nomenu
        movea.l d0,a0
        INT     ClearMenuStrip

        .nomenu movea.l window(a5),a0
        INT     CloseWindow

NoWin     movea.l screen(a5),a0
        INT     CloseScreen

NoScr     movea.l GfxBase(a5),a1
        EXEC    CloseLibrary

NoGfx     movea.l IntBase(a5),a1
        EXEC    CloseLibrary

NoInt     movea.l DosBase(a5),a1
        EXEC    CloseLibrary

NoDos     moveq   #0,d0
        rts

; *****
SetMenu   move.l  a0,-(sp)
        movea.l window(a5),a0
        INT     ClearMenuStrip

```



```

movea.l window(a5),a0
lea MainMenuList(pc),a1
move.l (sp)+,mu_NextMenu(a1)
INT SetMenuStrip
rts

; *****
PTitre move.l a0,-(sp)

moveq #0,d0 ; Efface l'écran
movea.l rport(a5),a1
GFX SetRast

moveq #3,d0
movea.l rport(a5),a1
CALL SetAPen

moveq #0,d0 ; Affiche le titre
moveq #20,d1
movea.l rport(a5),a1
CALL Move
movea.l (sp)+,a0
moveq #0,d0
move.b (a0)+,d0
movea.l rport(a5),a1
CALL Text

moveq #1,d0
movea.l rport(a5),a1
CALL SetAPen

moveq #0,d0 ; Positionne le curseur
moveq #30,d1 ; pour la 1ère ligne
move.w d1,ligne_y(a5)
movea.l rport(a5),a1
CALL Move
rts

; *****
NewLine addq.w #8,ligne_y(a5) ; Descend d'1 ligne de
texte
move.w ligne_y(a5),d1
moveq #0,d0
movea.l rport(a5),a1
CALL Move
rts

; *****
; Affiche la liste pointée par a2 en appelant la
; routine de formatage pointée par a3
AffList move.w #$4000,$dff09a
.all movea.l LH_HEAD(a2),a2
tst.l LN_SUCC(a2)
beq.s .endlst
lea lg_Data(a4),a1
jsr (a3)
bsr.s DoFmt
bra.s .all
.endlst move.w #$c000,$dff09a
rts

; *****
; Vérifie si le noeud pointé par a1 se trouve
; dans la liste pointée par a0
ChkList move.w #$4000,$dff09a
.all movea.l LH_HEAD(a0),a0
tst.l LN_SUCC(a0)
beq.s .end
cmpa.l a0,a1
bne.s .all
.end move.w #$c000,$dff09a
cmpa.l a0,a1
rts

; *****
; Vérifie si d0 est un numéro de ligne valide et si oui,
; renvoie l'adresse du Noeud concerné dans a1
CheckID move.l d0,d1
bmi.s .non
cmp.w lignes(a5),d0
bcc.s .non
mulu #lg_SIZE,d0
movea.l lg_Data(a4,d0.1),a1

move.b #4,CCR ; Positionne Z
rts
.non movea.l screen(a5),a0
INT DisplayBeep
move.b #0,CCR ; Efface Z
rts

; *****
DoFmt movem.l d0-d3/d7/a0-a3/a6,-(sp)
moveq #-1,d7
move.w lignes(a5),lg_Num(a4)
addq.w #1,lignes(a5)

movea.l a4,a1
lea .putch(pc),a2
lea lg_Str(a4),a3
EXEC RawDoFmt
cmpi.b #80,d7
bls.s .ok1
moveq #80,d7
.ok1 move.l d7,d0
lea lg_Str(a4),a0
movea.l rport(a5),a1
GFX Text
bsr NewLine
lea lg_SIZE(a4),a4
movem.l (sp)+,d0-d3/d7/a0-a3/a6
rts

.putch move.b d0,(a3)+
addq.l #1,d7
rts

; *****
GetNum movem.l d1-d7/a0-a6,-(sp)
lea w3gad(pc),a1
ori.w #LONGINT,gg_Activation(a1)
bra.s DoGadget

GetStr movem.l d1-d7/a0-a6,-(sp)
lea w3gad(pc),a1
andi.w #~LONGINT,gg_Activation(a1)

DoGadget:
movea.l gg_SpecialInfo(a1),a1
clr.w si_BufferPos(a1)
clr.w si_DisPos(a1)
clr.w si_NumChars(a1)
clr.l si_LongInt(a1)
movea.l si_Buffer(a1),a1
clr.b (a1)

moveq #-1,d4
move.l a0,d2

.ok1 movea.l window(a5),a0
INT ClearMenuStrip
lea nw3(pc),a0
move.l screen(a5),nw_Screen(a0)
move.l d2,nw_Title(a0)
CALL OpenWindow
move.l d0,d2
beq .nowin

.Loop movea.l d2,a0
movea.l wd_UserPort(a0),a0
EXEC WaitPort
.all movea.l d2,a0
movea.l wd_UserPort(a0),a0
EXEC GetMsg
tst.l d0
beq.s .Loop
movea.l d0,a1
move.l im_Class(a1),d3
CALL ReplyMsg

cmpi.l #GADGETUP,d3
beq.s .gotit
cmpi.l #ACTIVEWINDOW,d3
bne.s .all

.activ lea w3gad(pc),a0
movea.l d2,a1

```



```

suba.1 a2,a2
INT ActivateGadget
bra .all

.gotit move.1 strinfo+si_LongInt(pc),d4

.close movea.1 d2,a0
INT CloseWindow

.nowin movea.1 window(a5),a0
lea m1(pc),a1
CALL SetMenuStrip

move.1 d4,d0
movem.1 (sp)+,d1-d7/a0-a6
rts

; *****
Affiche_taches:
lea m2(pc),a0
bsr SetMenu
lea ttitre(pc),a0
bsr PTitre
clr.w lignes(a5)

movea.1 $4.w,a6
lea AffTask(pc),a3
movea.1 ThisTask(a6),a2
lea lg_Data(a4),a1
jsr (a3)
bsr DoFmt

lea TaskReady(a6),a2
bsr AffList

lea TaskWait(a6),a2
bsr AffList

lea tasks(a5),a2
bra AffList

AffTask lea tfmt(pc),a0
move.1 a2,(a1)
move.1 TC+LN_NAME(a2),18(a1)
bne.s .ok1
move.1 #Default,18(a1)

.ok1 move.1 #tasktxt,4(a1)
move.1 #Default,14(a1)
cmpi.b #NT_PROCESS,TC+LN_TYPE(a2)
bne.s .ok2
move.1 pr_CLI(a2),d0
beq.s .ok2
addq.1 #4,4(a1)
move.1 pr_TaskNum(a2),14(a1)
lsl.1 #2,d0
movea.1 d0,a0
move.1 cli_CommandName(a0),d0
lsl.1 #2,d0
movea.1 d0,a0
tst.b (a0)+
beq.s .ok1.1
move.1 a0,18(a1)

.ok1.1 lea tfmt2(pc),a0
.ok2 moveq #0,d0
move.b TC_STATE(a2),d0
bpl.s .ok1.2 ; Si le bit 7 est positionné,
moveq #7,d0 ; alors la tâche est "gelée"

.ok1.2 lsl.w #3,d0
add.1 #tasktxt,d0
move.1 d0,8(a1)
move.b TC+LN_PRI(a2),d0
ext.w d0
move.w d0,12(a1)
rts

ttitre dc.b .len-*
dc.b " N° Adresse Type Etat Pri Num No"
.len dc.b "m"
even

tfmt dc.b "%3d %08lx %.4s %-7.7s %4d %.3s %s",0
tfmt2 dc.b "%3d %08lx %.4s %-7.7s %4d %3ld %s",0

```

```

even
tstxt dc.b "Invalid Added Running Ready "
dc.b "Waiting Except Removed Frozen "
tasktxt dc.b "TaskProc"
even

Pri_tache:
lea pritxt(pc),a0
bsr GetNum
bsr CheckID
bne.s .err
lea TaskReady(a6),a0
bsr ChkList
beq.s .id_ok
lea TaskWait(a6),a0
bsr ChkList
beq.s .id_ok
lea tasks(a5),a0
bsr ChkList
bne.s .err
.id_ok lea pritxt2(pc),a0
bsr GetNum
EXEC SetTaskPri
.err lea txtbuff,a4
bra Affiche_taches

pritxt dc.b "N° de tâche :",0
even
pritxt2 dc.b "Priorité : ",0
even

Gele_tache:
lea geltxt(pc),a0
bsr GetNum
bsr CheckID
bne.s .err
cmpa.1 ThisTask(a6),a1
beq.s .err
lea TaskReady(a6),a0
bsr ChkList
beq.s .ok_id
lea TaskWait(a6),a0
bsr ChkList
bne.s .err
.ok_id CALL Disable
bset #7,TC_STATE(a1) ; Pour Affiche_taches
move.1 a1,-(sp)
CALL Remove ; Enlève le noeud de sa liste
lea tasks(a5),a0
movea.1 (sp)+,a1
CALL AddHead ; et le met dans la notre
CALL Enable
.err lea txtbuff,a4
bra Affiche_taches

geltxt dc.b "Geler N° :",0
even

Chauffe_tache:
lea tasks(a5),a0
IFEMPTY a0,.err

lea chautxt(pc),a0
bsr GetNum
bsr CheckID
bne.s .err
lea tasks(a5),a0
bsr ChkList
bne.s .err
.ok_id CALL Disable
lea tasks(a5),a0
move.1 a1,-(sp)
CALL Remove
lea TaskWait(a6),a0
movea.1 (sp)+,a1
bclr #7,TC_STATE(a1)
cmpi.b #TS_WAIT,TC_STATE(a1)
beq.s .ok1
lea TaskReady(a6),a0
.ok1 CALL AddTail
CALL Enable
.err lea txtbuff,a4

```



```

bra      Affiche_taches

chautxt dc.b      "Réchauffer N°:",0
even

; *****
Affiche_libraries:
    lea      m3(pc),a0
    bsr      SetMenu
    lea      ltitre(pc),a0
    bsr      PTitre
    clr.w    lignes(a5)

    movea.l   $4.w,a6
    lea      LibList(a6),a2
    lea      AffLibs(pc),a3
    bra      AffList

AffLibs move.l   a2,(a1)
        moveq  #0,d0
        move.b LIB_VERSION+1(a2),d0
        move.w d0,4(a1)
        move.b LIB_REVISION+1(a2),d0
        move.w d0,6(a1)
        move.b LIB_FLAGS(a2),d0
        ext.w  d0
        move.w d0,8(a1)
        move.b LIB_OPENCNT+1(a2),d0
        move.w d0,10(a1)
        move.l LIB+LN_NAME(a2),12(a1)
        bne.s  .ok1
        move.l #Default,12(a1)
.ok1    lea      lfmt(pc),a0
        rts

ltitre  dc.b      .len-*
        dc.b      " N° Adresse Version Flg Cnt No"
        .len      dc.b      "m"
        even
lfmt    dc.b      "%3d %08lx %3d.%-3d $%02x %3d %s",0
        even

Ouvre_lib:
    lea      libtxt(pc),a0
    bsr      GetStr
    lea      strbuf(pc),a1
    moveq    #0,d0
    EXEC     OpenLibrary
    tst.l    d0
    bne.s    .ok
    movea.l   screen(a5),a0
    INT      DisplayBeep
.ok      lea      txtbuff,a4
        bra      Affiche_libraries

libtxt  dc.b      "Nom library :",0
        even

Ferme_lib:
    lea      libtxt(pc),a0
    bsr      GetStr
    movea.l   $4.w,a6
    lea      LibList(a6),a0
    lea      strbuf(pc),a1
    CALL     FindName
    tst.l    d0
    beq.s     .err
    movea.l   d0,a1
    CALL     CloseLibrary
.err      lea      txtbuff,a4
        bra      Affiche_libraries

Purge_lib:
    move.l    $ffffff,d0
    moveq     #1,d1
    EXEC     AllocMem
    lea      txtbuff,a4
    bra      Affiche_libraries

; *****
Affiche_devices:
    suba.l    a0,a0
    bsr      SetMenu

```

```

    lea      ltitre(pc),a0
    bsr      PTitre
    clr.w    lignes(a5)

    movea.l   $4.w,a6
    lea      DeviceList(a6),a2
    lea      AffLibs(pc),a3
    bra      AffList

; *****
Affiche_ports:
    lea      m4(pc),a0
    bsr      SetMenu
    lea      ptitre(pc),a0
    bsr      PTitre
    clr.w    lignes(a5)

    movea.l   $4.w,a6
    lea      PortList(a6),a2
    lea      AffPort(pc),a3
    bsr      AffList

    lea      ports(a5),a2
    bra      AffList

AffPort move.l   a2,(a1)
        move.l   MP+LN_NAME(a2),4(a1)
        bne.s    .ok1
        move.l   #Default,4(a1)
.ok1    moveq     #0,d0
        move.b   MP_FLAGS(a2),d0
        bpl.s    .ok1.2
        moveq     #4,d0
.ok1.2  lsl.w     #3,d0
        add.l    #pftxt,d0
        move.l    d0,8(a1)
        move.b    MP_SIGBIT(a2),d0
        andi.w    #$ff,d0
        move.w    d0,12(a1)

        move.l    #Default,14(a1)
        move.l    MP_SIGTASK(a2),d0
        beq.s     .ok3
        movea.l   d0,a0
        cmpi.b    #NT_PROCESS,TC+LN_TYPE(a0)
        bne.s     .ok2
        move.l    pr_CLI(a0),d0
        beq.s     .ok2
        lsl.l     #2,d0
        movea.l   d0,a0
        move.l    cli_CommandName(a0),d0
        lsl.l     #2,d0
        movea.l   d0,a0
        tst.b     (a0)+
        bne.s     .ok4
        movea.l   MP_SIGTASK(a2),a0
        movea.l   TC+LN_NAME(a0),a0
        move.l     a0,14(a1)
        bra.s     .ok3
.ok4    move.l     TC+LN_NAME(a0),d0
        beq.s     .ok3
        move.l     d0,14(a1)
        lea      pfmt(pc),a0
        rts

ptitre  dc.b      .len-*, " N° Adresse"
        dc.b      " Nom                      Flags      Bit Tâch"
        .len      dc.b      "e"
        even
pfmt    dc.b      "%3d %08lx %-16.16s %.7s %2d %s",0
        even
pftxt   dc.b      "Signal SoftInt Ignore "
        dc.b      "Action Hidden "
        even

Cache_port:
    lea      portxt(pc),a0
    bsr      GetNum
    bsr      CheckID
    bne.s     .err
    lea      PortList(a6),a0
    bsr      ChkList
    bne.s     .err

```



```

CALL    Disable
bset    #7,MP_FLAGS(a1) ; Pour Affiche_port
move.l  a1,-(sp)
CALL    Remove
lea     ports(a5),a0
movea.l (sp)+,a1
CALL    AddHead
CALL    Enable
.err    lea     txtbuff,a4
        bra     Affiche_ports

```

```

portxt  dc.b    "N° du port :",0
even

```

Revele_port:

```

lea     ports(a5),a0
IFEMPTY a0,.err
lea     portxt2(pc),a0
bsr     GetNum
bsr     CheckID
bne.s   .err
lea     ports(a5),a0
bsr     ChkList
bne.s   .err
CALL    Disable
move.l  a1,-(sp)
lea     ports(a5),a0
CALL    Remove
lea     PortList(a6),a0
movea.l (sp)+,a1
bclr    #7,MP_FLAGS(a1)
CALL    AddTail
.err    lea     txtbuff,a4
        bra     Affiche_ports

```

```

portxt2 dc.b    "Montrer N° :",0
even

```

; *****

Affiche_memoire:

```

lea     m5(pc),a0
bsr     SetMenu
lea     mtitre(pc),a0
bsr     PTitre
clr.w   lignes(a5)

```

```

movea.l $4.w,a6
lea     MemList(a6),a2
lea     AffMem(pc),a3
bra     AffList

```

```

AffMem  move.l  a2,(a1)
        move.l  MH_LOWER(a2),4(a1)
        move.l  MH_UPPER(a2),8(a1)
        move.l  MH_FREE(a2),12(a1)
        move.w  MH_ATTRIBUTES(a2),16(a1)
        move.b  MH+LN_PRI(a2),d0
        ext.w   d0
        move.w  d0,18(a1)
        move.l  MH+LN_NAME(a2),20(a1)
        bne.s   .ok
        move.l  #Defaut,20(a1)
.ok     lea     mfmt(pc),a0
        rts

```

```

mtitre  dc.b    .len-*, " N° Node      "
        dc.b    "Lower  Upper  Free  Attr "
        dc.b    "Pri No"
.len    dc.b    "m"
even
mfmt    dc.b    "%3d %08lx %08lx %08lx "
        dc.b    "%8ld  %2d %3d %s",0
even

```

Pri_memoire:

```

lea     memtxt(pc),a0
bsr     GetNum
bsr     CheckID
bne.s   .err
lea     memtxt2(pc),a0
bsr     GetNum
movem.l d0/a1,-(sp)
CALL    Disable

```

```

CALL    Remove
lea     MemList(a6),a0
movem.l (sp)+,d0/a1
move.b  d0,LN_PRI(a1)
CALL    Enqueue
CALL    Enable
.err    lea     txtbuff,a4
        bra     Affiche_memoire

```

```

memtxt  dc.b    "N° du noeud :",0
even

```

```

memtxt2 dc.b    "Priorité :",0
even

```

; *****

Affiche_resources:

```

suba.l  a0,a0
bsr     SetMenu
lea     ltitre(pc),a0
bsr     PTitre
clr.w   lignes(a5)

movea.l $4.w,a6
lea     ResourceList(a6),a2
lea     AffLibs(pc),a3
bra     AffList

```

; *****

Affiche_modules:

```

suba.l  a0,a0
bsr     SetMenu
lea     otitre(pc),a0
bsr     PTitre
clr.w   lignes(a5)

```

```

movea.l $4.w,a6
movea.l ResModules(a6),a2
AffMods lea     lg_Data(a4),a1

```

```

.next  move.l  (a2)+,d0
        beq.s   .ret
        bgt.s   .1
        bclr    #31,d0
        movea.l d0,a2
        bra.s   .next
.1     movea.l  d0,a0
        move.l  a0,(a1)
        move.b  RT_PRI(a0),d0
        ext.w   d0
        move.w  d0,4(a1)
        move.b  RT_VERSION(a0),d0
        ext.w   d0
        move.w  d0,6(a1)
        move.b  RT_TYPE(a0),d0
        andi.w  #15,d0
        lsl.w   #2,d0
        move.l  ottab(pc,d0.w),8(a1)
        move.l  RT_INIT(a0),12(a1)
        move.l  RT_NAME(a0),16(a1)
        bne.s   .ok1
        move.l  #Defaut,16(a1)
.ok1    lea     ofmt(pc),a0
        bsr     DoFmt
        bra.s   AffMods
.ret    rts

```

```

otitre  dc.b    .len-*, " N° "
        dc.b    "Adresse Pri Ver Type      Init      No"
.len    dc.b    "m"
even
ofmt    dc.b    "%3d %08lx %3d %3d %-8s %08lx %s",0
even
ottab   dc.l    Defaut,otxt1,0,otxt2,0,0,0,0
        dc.l    otxt3,otxt4,0,0,0,0,0,0
otxt1   dc.b    "Tâche",0
otxt2   dc.b    "Device",0
otxt3   dc.b    "Resource",0
otxt4   dc.b    "Library",0
even

```

(suite page 29) 

Lorsque l'on veut tracer des droites au blitter, la recherche de l'octant dans lequel se trouve la droite est souvent problématique, surtout lorsque les points définissant la droite sont calculés.

On peut toujours utiliser la routine de Frédéric Mazué, parue dans l'ANT N° 14, mais je vous propose d'étudier la façon dont est conçu le code de l'octant pour mettre au point une autre routine un peu plus rapide : on n'a plus à piocher dans un table, ce qui réclame beaucoup de cycles de bus (l'instruction move.w d(pc),an en réclame 24 avec un 68000!). La composition du code de l'octant apparaît comme étant assez compliquée, mais en fait elle est régie suivant une loi simple.

Soient (x1,y1) et (x2,y2) les coordonnées des point définissant la droite. Posons $DX=|X1-X2|$ et $DY=|Y1-Y2|$ et étudions le tableau suivant.

CODE EN BINAIRE	Y1 PAR RAPPORT A Y2	X1 PAR RAPPORT A X2	DX PAR RAPPORT A DY
000	Y1<Y2	X1<X2	DX<DY
100	Y1<Y2	X1<X2	DX>DY
010	Y1<Y2	X1>X2	DX<DY
101	Y1<Y2	X1>X2	DX>DY
001	Y1>Y2	X1<X2	DX<DY
110	Y1>Y2	X1<X2	DX>DY
011	Y1>Y2	X1>X2	DX<DY
111	Y1>Y2	X1>X2	DX>DY

L'analyse des deux premières lignes nous montre que si DX est supérieur à DY, alors le troisième bit du code de l'octant est positionné. La ligne 3 semble nous montrer que le second bit correspond à X1 par rapport à X2, mais la ligne 4 fout toute cette jolie théorie par terre. Par contre, ces deux lignes confirment l'hypothèse faite sur le troisième bit. Les quatre dernières lignes nous montrent que cette hypothèse est valable.

Nous codons donc pour l'instant l'octant sous la forme Dxx, où D est positionné selon DX et DY. Ce qui nous amène à séparer le précédent tableau en deux (c'est un très bon moyen pour y voir clair).

en supposant DX>DY			en supposant DX<DY		
code bin	Y1,Y2	X1,X2	code bin	Y1,Y2	X1,X2
000	Y1<Y2	X1<X2	100	Y1<Y2	X1<X2
010	Y1<Y2	X1>X2	101	Y1<Y2	X1>X2
001	Y1>Y2	X1<X2	110	Y1>Y2	X1<X2
011	Y1>Y2	X1>X2	111	Y1>Y2	X1>X2

Et que voit-on ? Tout simplement que l'octant peut être codé sous la forme 1AB ou 0BA ! Et comment le voit-on ? Très simple.

Pour A d'abord. Dans le tableau de gauche, on constate que sur les deux premières lignes le premier bit est à 0 alors qu'il est mis pour les lignes 3 et 4, ce qui correspondrait à A=1 si Y1>Y2 et A=0 si Y1<Y2. En se ruant sur l'autre tableau, on s'aperçoit que le premier bit ne joue plus ce rôle et l'on regarde, à tout hasard, si le second ne correspond pas... Et là, ô miracle, il correspond. Tout s'enchaîne alors et l'on s'aperçoit que le bit restant évolue aussi selon la loi magique :

B=1 si X1>X2 et B=0 si X1<X2.

Avec un peu de recul, on décide plus généralement que le code de l'octant est de la forme XYZ avec $X=F(DX,DY)$, $Y=G(Pdelta)$ et $Z=H(Gdelta)$, où Pdelta et Gdelta sont respectivement le plus

petit et le plus grand de DX et DY.

Enfin, voici un extrait de source mettant tout ceci en application. On peut le mettre à la place de la routine Recherche_octant du programme de l'ANT N°14.

Note : la routine Recherche_octant que je vous propose a été modifiée pour être compatible avec le programme de Frédéric Mazué. Les changements sont indiqués en commentaire avec les anciennes valeurs.

```
Recherche_octant ;trouve l'octant et le renvoie dans D0
Move.w X1,D0 ;D3 contient PDelta et D2 GDelta
Move.w Y1,D1 ;D0 à D3 sont modifiés
Move.w X2,D2 ;L'octant est déjà decalé de deux sur
Move.w Y2,D3 ;la gauche (on a plus qu'à faire un or.w
D0,...)
Sub.w D1,D3
Bpl.s Y1_inf_Y2 ;si Y1 est plus petit
Neg.w D3 ;DY>0
Moveq #4,D1 ;A=1 (cf article),si l'on veut utiliser
cette routine
Bra.s Test_X ;a la place de celle de l'ANT N°14 mettre
1 dans D1
Y1_inf_Y2
Moveq #0,D1 ;si Y1<Y2 alors A=0
Test_X
Sub.w D0,D2
Bpl.s X1_inf_X2 ;Idem pour X
Neg.w D2
Moveq #4,D0 ;B=1,on peut mettre 1 dans D0 pour utiliser
cette
Bra.s Test_D ;routine dans le programme de l'ANT N°14
X1_inf_X2
Moveq #0,D0
Test_D
Cmp.w D3,D2
Bpl.s DY_sup_DX
Lsl.b #1,D1
Or.b D1,D0
Or.b #16,D0 ;1AB dans D0,mettre Or.b #4,D0 si l'on
met cette
;routine dans le Programme de l'ANT N°14
; Move.b D0,octant ;modification pour assurer
; Move.w D2,GDelta ;la compatibilité,ajoutez
; Move.w D3,PDelta ;ces trois lignes pour utiliser
; Rts ;le programme de l'ANT N°14
DY_sup_DX
Exg.w D3,D2
Lsl.b #1,D0
Or.b D1,D0 ;0BA dans D0
; Move.b D0,octant ;cf ci dessus
; Move.w D2,GDelta
; Move.w D3,PDelta
Rts

;
;
; tracer de droite au blitter,programme de l'ANT 14
; la routine recherche_octant a ete modifiée
;
;
;
; **exec.library**
OldOpenLibrary equ -408
CloseLibrary equ -414
AlloocMem equ -198
FreeMem equ -210
SuperState equ -150
UserState equ -156
; **graphics library**
OwnBlitter equ -456
DisownBlitter equ -462
; **intuition library**
OpenScreen equ -198
CloseScreen equ -66
; **CHIPS**
BLTSize equ $58
BLTCPth equ $48
BLTBPTH equ $4C
BLTAPTH equ $50
BLTDPth equ $54
BLTCMOD equ $60
BLTBMOD equ $62
BLTAMOD equ $64
BLTDMOD equ $66
BLTCON0 equ $40
BLTCON1 equ $42
BLTAFWM equ $44
BLTALWM equ $46
BLTDDAT equ $00
BLTCDAT equ $70
```



```
BLTBDAT equ $72
BLTADAT equ $74
DMACON equ $96
DMACONR equ $02
```

Debut

```
Move.l $4,A6
Lea graf_name,A1
Jsr OldOpenLibrary(A6)
Move.l D0,graf_base
Lea int_name,A1
Jsr OldOpenLibrary(A6)
Move.l D0,int_base
Move.l int_base,A6
Lea ecran,A0
Jsr OpenScreen(A6)
Move.l D0,canal_ecran
lea $dff000,A5
Bsr Blitter_Busy
Move.l graf_base,A6
Jsr OwnBlitter(A6)
Bsr Recherche_octant
Bsr adresse_depart
Move.l D0,BLTCPATH(A5)
Move.l D0,BLTDPATH(A5)
Move.w #40,BLTCDMOD(A5)
Move.w #40,BLTDMOD(A5)
Clr.l D0
Clr.l D1
Clr.l D2
Clr.l D6
Clr.l D7
Move.w PDelta,D0
Move.w GDelta,D1
Muls #4,D0
Move.w D0,BLTBMOD(A5)
Move.w D0,D2
Muls #2,D1
Sub.w D1,D0
Roxl.w #7,D7
Move.l D0,BLTAPATH(A5)
Move.w D2,D0
Muls #2,D1
Sub.w D1,D0
Move.w D0,BLTAMOD(A5)
Move.w #$8000,BLTADAT(A5)
Move.w $ffff,BLTBDAT(A5)
Move.w $ffff,BLTAFWM(A5)
Move.w $ffff,BLTALWM(A5)
Move.w X1,D0
And.w #$000f,D0
Ror.w #4,D0
Or.w D0,D6
Or.w D0,D7
Or.w #0bca,D6
Move.w D6,BLTCON0(A5)
Move.b octant,D0
Ext.w D0
; Ror.w #2,D0
Bset #0,D0
Or.w D0,D7
; Bset #1,D7
Move.w D7,BLTCON1(A5)
Move.w GDelta,D0
add.w #01,D0
Muls #64,D0
Add.w #00002,D0
Move.w D0,BLTSIZE(A5)
Bsr Blitter_Busy
Jsr DisownBlitter(A6)
souris
Lea $bfe001,A5
Btst #6,(A5)
Bne.s souris
Move.l int_base,A6
Move.l canal_ecran,A0
Jsr CloseScreen(A6)
Move.l $04,A6
Move.l int_base,A1
Jsr CloseLibrary(A6)
Move.l graf_base,A1
Jsr CloseLibrary(A6)
Rts
Blitter_Busy
Btst #14,DMACONR(A5)
Bne.s Blitter_Busy
Rts
Recherche_octant ;trouve l'octant et le renvoie dans D0
Move.w X1,D0 ;D3 contient PDelta et D2 GDelta
Move.w Y1,D1 ;D0 à D3 sont modifies
Move.w X2,D2
Move.w Y2,D3
Sub.w D1,D3
Bpl.s Y1_inf_Y2 ;si Y1 est plus petit
Neg.w D3 ;DY>0
```

```
Moveq #4,D1 ;A=1 (cf article),si l'on veut eviter le
decalage
Bra.s Test_X ;lors de la preparation de BLTCON1 mettre 4
dans D1
Y1_inf_Y2
Moveq #0,D1 ;si Y1<Y2 alors A=0
Test_X
Sub.w D0,D2
Bpl.s X1_inf_X2 ;Idem pour X
Neg.w D2
Moveq #4,D0 ;B=1,on peut mettre 4 a la place pour eviter
le decalage
Bra.s Test_D ;vers la gauche lors de la preparation de
BLTCON1
X1_inf_X2
Moveq #0,D0
Test_D
Cmp.w D2,D3
Bpl.s DY_sup_DX
Lsl.b #1,D1
Or.b D1,D0
Or.b #16,D0 ;1AB dans D0,si on a fait les
modifications lors de
;la preparation de D1 et D0 alors mettre or.b #16,D0 au lieu de
or.b #4,D0
Move.b D0,octant ;modification pour assurer
Move.w D3,PDelta ;la compatibilitee,enlevez
Move.w D2,GDelta ;ces trois ligne et traitez
;le resultats directement
DY_sup_DX
Exg.w D3,D2
Lsl #1,D0
Or.b D1,D0 ;0BA dans D0
Move.b D0,octant ;modification pour assurer
Move.w D3,PDelta ;la compatibilitee,enlevez
Move.w D2,GDelta ;ces trois ligne et traitez
;le resultats directement
Rts
adresse_depart
Move.w X1,D0
Ext.l D0
Divs #8,D0
And.l $ffff,D0
Move.w Y1,D1
Muls #40,D1
Add.l D1,D0
Move.l canal_ecran,A1
Move.l $c0(A1),D1
Add.l D1,D0
Rts
image Dc.l 0
canal_ecran Dc.l 0
int_base Dc.l 0
graf_base Dc.l 0
int_name Dc.b 'intuition.library',0
Even
graf_name Dc.b 'graphics.library',0
Even
ecran
Dc.w 0
Dc.w 0
Dc.w 320
Dc.w 200
Dc.w 2
Dc.b 0
Dc.b 1
Dc.w 2
Dc.w 15
Dc.l 0
Dc.l titre_ecran
Dc.l 0
Dc.l 0
titre_ecran
Dc.b 'Little STORM Blitter demo',0
Even
octant dc.b 0
Even
X1 Dc.w 0
Y1 Dc.w 150
X2 Dc.w 300
Y2 Dc.w 100
GDelta Dc.w 0
PDelta Dc.w 0
end
```

Par Jean-Sébastien Valette

Les différentes interruptions que nous avons vues jusqu'à présent étaient toutes générées par Paula, suite à un évènement hardware. Mais Exec permet également la génération d'interruptions totalement logicielles.

On ne voit pas très bien à priori à quoi des interruptions logicielles peuvent bien servir... Alors qu'il est souvent primordial de pouvoir réagir au quart de millionième de seconde près au VBL ou à la fin d'une transmission série, un petit délai (dû à un grand nombre de tâches tournant concurremment) dans un protocole de communication entre deux processus n'est que très rarement pénalisant. En fait, c'est justement là leur utilité : permettre de synchroniser le plus justement possible deux tâches entre elles.

Le principe est très simple : étant donné que les messages sont chaînés les uns à la suite des autres dans un port, un message particulier peut mettre un certain temps avant d'être traité par la tâche réceptrice. Or, dans certains cas, il est primordial que ce message soit immédiatement pris en compte, quitte à ce que les autres patientent un peu, bien au chaud qu'ils sont dans leur liste d'attente. Pour ce faire, on décide de suspendre provisoirement le système multitâche - grâce à une interruption, donc - et de gérer tout de suite l'évènement intervenu. On le voit, les interruptions logicielles agissent exactement comme les interruptions matérielles, à ceci près qu'elles sont générées non pas par le hardware, mais par le software (en fait, elles utilisent le niveau d'interruption 1, bit 2 de INTENA).

MISE EN OEUVRE

Une interruption logicielle s'initialise exactement de la même manière qu'une interruption matérielle, au moyen d'une structure Interrupt correctement remplie. Attention ici à bien spécifier dans le noeud, le type NT_INTERRUPT et non NT_SOFTINT, qui est un flag interne à Exec. Seules cinq priorités sont possibles : -32, -16, 0, +16 et +32 ; toute autre valeur est illégale. Les champs is_Code et is_Data sont initialisés comme pour une interruption matérielle (cf. le mois dernier).

Une fois déclenchée, l'interruption logicielle s'exécute dans un environnement identique à celui des interruptions matérielles : mode superviseur, retour au handler d'Exec par l'instruction RTS et protection obligatoire des registres A2-A4 et D2-D7. Si elle survient en cours de fonctionnement "normal" du système (mode multitâche), l'interruption est exécutée immédiatement ; sinon, si elle est déclenchée au cours d'une autre interruption logicielle, elle devra attendre que celle-ci se termine ; enfin, si elle est déclenchée au cours d'une interruption matérielle, elle devra attendre que tous les serveurs éventuels restant à exécuter aient fini leur travail. Ceci permet par exemple à une interruption de haut-niveau de se décharger d'une tâche ingrate et gourmande en temps-machine sur une autre interruption de priorité moindre.

Il existe deux moyens de déclencher une interruption logicielle : soit en appelant la fonction Cause(), soit en postant un message dans un port de type PA_SOFTINT.

Dans le premier cas, on transmet à Cause() l'adresse de la structure Interrupt concernée et l'interruption se déclenche suivant les conditions énoncées plus haut.

Dans le second cas, on initialise un MsgPort comme illustré ci-dessous :

```
struct MsgPort SoftIntPort = {0};
struct Interrupt SoftInt = {0};
```

```
/* Initialisation de l'interruption */
SoftInt.is_Node.ln_Type = NT_INTERRUPT;
SoftInt.is_Node.ln_Pri = 0; /* par exemple */
SoftInt.is_Code = (VOID (*)())SoftCode;
SoftInt.is_Data = (APTR)SoftData;

/* Initialisation du port */
SoftIntPort.mp_Node.ln_Type = NT_MSGPORT;
SoftIntPort.mp_Node.ln_Name = "SoftInt port";
SoftIntPort.mp_Flags = PA_SOFTINT;
SoftIntPort.mp_SigTask = (struct Task *)&SoftInt;
AddPort(&SoftIntPort);
```

Ensuite, chaque message envoyé sur ce port déclenchera l'interruption logicielle concernée. Notez qu'en assembleur, il existe un champ supplémentaire dans la structure MsgPort, baptisé MP_SOFTINT, qui permet de différencier à l'écriture du programme un port d'interruption d'un port de message. Sa valeur est égale à celle de MP_SIGTASK.

EMPLOI AVEC LES DEVICES

Comme les interruptions matérielles, les interruptions logicielles ne peuvent appeler qu'un nombre réduit de fonctions du système. Une particularité est à noter concernant les devices : normalement, il est interdit d'utiliser DoIO() et BeginIO() dans une routine d'interruption, mais le timer.device et l'audio.device font exception à cette règle. Cela permet par exemple de relancer directement une requête de l'un de ces devices depuis l'interruption, sans que le programme principal n'ait besoin d'intervenir.

L'EXEMPLE

Comme le veut la tradition, nous allons terminer cette étude des interruptions logicielles par un petit exemple. En l'occurrence, il ne s'agit ni plus ni moins que d'un petit compteur de 1/2 secondes. On aurait certes pu le programmer d'une autre manière, mais c'est la seule idée de programme qui me soit passée par la tête...

```
;
; Exemple (tout-à-fait inutile) d'interruption
logicielle.
;
; L'interruption compte les 1/2 secondes...
; Le programme principal attend simplement un CTRL/C puis
; affiche le résultat du comptage.
;
; S. Schreiber - 171190
;
```

```
incdir      "include:"
include     "exec/interrupts.i"
include     "devices/timer.i"
include     "libraries/dos.i"
```

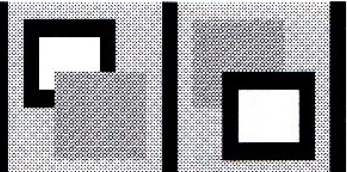
```
include     "exec/exec_lib.i"
include     "intuition/intuition_lib.i"
include     "libraries/dos_lib.i"
```

```
OFF EQU 0 ; Compteur OFF
ON EQU 1 ; Compteur ON
STOP EQU 2 ; Compteur arrêté
```

```
DELAIEQU 50000; Nb micro-secondes (1/2 seconde)
```

```
; *****
Startlea dosname(pc),a1; Ouvre la dos.library
moveq#0,d0
```


SOUS EXEC (II)



```
CALLEXEC OpenLibrary
move.l    d0,_DOSBase
beq    NoDos

CALLDOS    Output; Pour communiquer...
move.l    d0,stdout

move.l    d0,d1; Salut !
move.l    #text,d2
moveq#tlen,d3
CALLDOS    Write

lea    tireq(pc),a1; Ouvre le timer.device
move.b    #NT_MESSAGE,IO+MP+LN_TYPE(a1)
move.l    #SoftIntPort,IO+MN_REPLYPORT(a1)
lea    tiname(pc),a0
moveq#UNIT_MICROHZ,d0
moveq#0,d1
CALLEXEC OpenDevice
tst.b d0
bne    NoTimer

lea    SoftIntPort(pc),a1
CALLEXEC AddPort

bsr    begintr    ; Lance la 1ère requête...

*****    Programme principal : attente de CTRL/C (!)

Main move.l    #SIGBREAKF_CTRL_C,d0
CALLEXEC Wait

*****    C'est tout !!!

move.l    counter(pc),longout

move.b    #OFF,iflag; Demande à l'interruption de
Exit cmpi.b    #STOP,iflag; s'arrêter et attend qu'elle
bne.s Exit    ; le fasse

lea    nbfmt(pc),a0; Affiche le compteur
lea    counter(pc),a1
lea    putche(pc),a2
lea    charout(pc),a3
CALLEXEC RawDoFmt

lea    SoftIntPort(pc),a1
CALLEXEC RemPort

lea    tireq(pc),a1; Ferme le timer.device
CALLEXEC CloseDevice

NoTimer movea.l    _DOSBase(pc),a1; Ferme la
dos.library
CALLEXEC CloseLibrary

NoDos moveq#0,d0    ; Retour au CLI
rts

; *****
putche    movem.l    d0-d3/a6,-(sp); J'avais pas envie de
bufferiser !
move.b    d0,(a3)
move.l    stdout(pc),d1
move.l    a3,d2
moveq#1,d3
CALLDOS    Write
movem.l    (sp)+,d0-d3/a6
rts

; *****
; *****    Routine d'interruption logicielle
SoftCode:
addq.l    #1,counter; Incrémente le compteur

cmpi.b    #ON,iflag; On s'arrête ?
beq.s begintr    ; Pas encore, relance la requête
move.b    #STOP,iflag; Oui, prévient Main
rts

;
; Lancement d'une requête timer pour DELAI micro-
secondes.
;
; ATTENTION :
; -----
; Ca ne marche qu'avec le timer.device ou l'audio.device
!
; On ne peut pas appeler les autres devices depuis une
; interruption, même logicielle !
;
begintr    lea    tireq(pc),a1
move.w    #TR_ADDREQUEST,IO_COMMAND(a1)
move.l    #DELA, IOTV_TIME+TV_MICRO(a1)
BEGINIO
rts

; *****
; Variables
_DOSBase dc.l    0    ; dos.library
stdout    dc.l    0    ; fenêtre CLI
counter    dc.l    0    ; compteur
longout    dc.l    0    ; buffer pour RawDoFmt
charout    dc.b    0    ; ""
iflag dc.b    ON    ; flag d'arrêt

; MsgPort de type PA_SOFTINT
SoftIntPort:
dc.l    0,0    ; ln_Succ, ln_Pred
dc.b    NT_MSGPORT,0; ln_Type, ln_Pri
dc.l    pName    ; ln_Name
dc.b    PA_SOFTINT,0; mp_Flags, mp_SigBit
dc.l    SoftInt    ; mp_SigTask
dcb.b LH_SIZE    ; mp_MsgList

; structure Interrupt
SoftInt    dc.l    0,0    ; ln_Succ, ln_Pred
dc.b    NT_INTERRUPT,0; ln_Type, ln_Pri
dc.l    siName    ; ln_Name
dc.l    0,SoftCode; is_Data, is_Code

tireq dc.b    IOTV_SIZE ; IORquest pour le timer.device

dosname    DOSNAME
tiname    TIMENAME
pName dc.b    "Horloge.Port",0
siName    dc.b    "Horloge.interrupt",0
text dc.b    "L'interruption logicielle compte les demi-"
dc.b    "secondes...",10,"Pressez CTRL/C pour
arrêter.",10
tlen EQU    *-text

nbfmt dc.b    "Résultat du comptage : %ld",10,0
even

; *****
END
```

Par Stéphane Schreiber

ToolBox: GAWK

Nous entamons ici une nouvelle série dont le but est de vous initier à l'utilisation de programmes compatibles Unix et de vous faire découvrir un certain nombre d'utilitaires du Domaine Public qui de par leur qualité, valent bon nombre de produits du commerce.

Nous allons commencer ce mois-ci par GAWk (abréviation de GNU Awk), un langage de traitement de fichiers ASCII. C'est l'aboutissement de la réécriture de l'utilitaire Unix Awk dans le cadre du projet GNU (d'où sont nom de GAWk), projet visant à améliorer les fonctionnalités et la portabilité des commandes Unix sur d'autres systèmes d'exploitation, tout en restant compatible avec les anciennes.

Le langage Awk a été écrit à l'origine par trois célèbres acteurs du monde Unix, à savoir Aho, Weinberger et Kernighan (vous savez, un des papas du C). GAWk est compatible avec la version System V release 4 d'Unix.

FONCTIONS ET SYNTAXE

La documentation Unix officielle décrit la fonctionnalité de Awk (et par extension de GAWk) comme une commande permettant de réaliser des recherches sur des lignes spécifiques d'un fichier ASCII et d'effectuer des traitements en fonction du résultat de ces recherches.

Pour expliquer un peu plus cette définition, je dirais que GAWk est un logiciel de traitement des chaînes de caractères d'un fichier ASCII, plus puissant que grep (1), dans la mesure où il autorise la réalisation de traitements sur ces chaînes de caractères là où grep se contente de les visualiser.

Les différentes fonctionnalités de GAWk sont :

- Traitement de données numériques.
- Utilisation de variables et de tableaux unidimensionnels.
- Contrôle des flots de données d'entrée.
- La sélection et utilisation de patterns (2) de recherche.
- Besoin d'aucun compilateur d'aucune sorte pour l'analyse des traitements à effectuer.

GAWk lit chaque fichier d'entrée dans l'ordre indiqué lors du passage de la commande (cf. description de la syntaxe ci-dessous). Si le nom du fichier est '-' où s'il n'existe aucun nom de fichier alors GAWk lit l'entrée standard (typiquement l'entrée clavier).

Voyons de manière rapide comment cela fonctionne.

La commande GAWk travaille ligne par ligne. Pour chaque ligne, elle cherche à détecter la correspondance entre la ligne et la liste des patterns définies par l'utilisateur. Quand elle trouve l'occurrence d'une pattern dans la ligne, elle exécute l'action associée à cette pattern et renvoie le résultat sur la sortie standard (par défaut l'écran).

La syntaxe de la commande telle qu'elle est décrite dans le manuel est la suivante :

GAWk [-F caractère] [-f Programme] [--] [fichier...]

- *caractère* permet d'indiquer le caractère séparateur de champs.
- *programme* indique le nom du programme Awk à exécuter.
- *indique* la fin des options. Ceci est très utile pour autoriser les noms des arguments des programmes sources à commencer par le caractère '-'.
- *fichier* correspond à la liste des fichiers à traiter.

LANGAGE OU PAS LANGAGE ?

GAWk ressemble plus à un véritable langage qu'à un simple utilitaire dans la mesure où l'on est amené à programmer des actions précises devant survenir si certaines conditions sont réunies, et ce à l'aide d'un langage structuré basé sur le couple pattern/actions. Pour nous en persuader, voyons plus en détails le mécanisme de traitement de GAWk et la manière dont on peut le programmer.

GAWk commence par lire et contrôler la syntaxe de toutes les paires "pattern/actions" du programme. Si aucune erreur de syntaxe n'est détectée, il transforme le programme en une série d'instructions internes (aucun résultat de cette pseudo compilation ne sort de la commande). A ce moment, le traitement ligne par ligne des fichiers spécifiés peut commencer.

GAWk lit une ligne du fichier d'entrée et la compare à chaque pattern du programme, effectuant l'action associée dans le cas où le test de comparaison est vrai. Il passe ensuite à la prochaine ligne, et continue ainsi jusqu'à la dernière du dernier fichier d'entrée.

La commande GAWk considère chaque ligne du fichier d'entrée comme une suite de champs séparés par un caractère particulier et se terminant par une fin de ligne. Par défaut, le caractère servant de séparateur entre deux champs est le caractère <Espace>. Il est toutefois possible de le changer par celui de votre choix.

En effet, ce séparateur est contenu dans une variable spéciale FS (Field Separator). Pour changer de séparateur il vous suffit d'affecter à FS autre chose que le blanc ' '. Ainsi, l'instruction GAWk

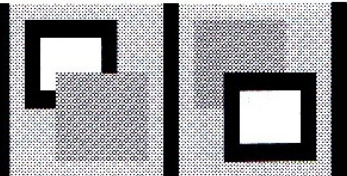
FS = '.'

définit le point comme séparateur.

Un séparateur peut également être une chaîne de caractères avec ou sans pattern(s) de sélection. Chaque champ de la ligne est stocké à la lecture dans une variable. Pour accéder à un champ particulier, il faut donc lire la variable correspondante. Ce sont les variables \$1 pour le premier champ, \$2 pour le deuxième et ainsi de suite jusqu'à \$99, qui est le maximum autorisé. A noter que pour accéder à l'ensemble des champs on utilisera la variable \$0. Par exemple, l'affectation de '2' au troisième champ d'une ligne deviendra :

\$3 = '2'

Parlons maintenant plus en détails de la syntaxe du langage. Comme je vous l'ai dit, l'élément de base du langage est la séquence pattern/actions. Il est toutefois possible de se passer de l'un de ces deux éléments : s'il n'y a pas de pattern, on effectue l'action sur toutes les lignes du fichier, et s'il n'y a pas d'action, GAWk se contente d'écrire la ligne complète sur la sortie standard.



Une action au sens de GAWk est une suite d'instructions régies par les mêmes règles syntaxiques que le C. Voici un détail des principaux ordres de GAWk.

```
if (expression) instruction [instruction]
while (expression) instruction [instruction]
for (expression1; expression2; expression3)
break
print [expression]
printf [format] [expression]
next
delete Array[expression] (détruit un tableau d'éléments)
exit[expression] (sortie immédiate du programme)
```

LES VARIABLES

Pour être complet sur la description de GAWk, il nous faut voir maintenant les variables et la manière dont elles sont gérées.

Les variables de GAWk sont purements dynamiques. C'est-à-dire qu'elles n'existent qu'à partir de leur première assignation. Elles ne sont pas non plus typées dans le sens où elle prennent le type de ce qu'on leur affecte. Les valeurs qu'elles contiennent peuvent être réelles, entières ou de type chaîne de caractères. GAWk supporte les tableaux unidimensionnels seulement, mais il est très simple de simuler plusieurs dimensions par un habile décalage d'indice. GAWk possède également bon nombre de variables réservées qu'il initialise et assigne dynamiquement de manière transparente pour l'utilisateur. Passons en revue quelques une de ces variables réservées du langage qui, vous allez pouvoir en juger, sont très utiles.

- FS : caractère indiquant la séparation de deux champs, par défaut il s'agit du caractère <espace>.
- NF : nombre de champs dans la ligne courante (abréviation anglaise de Number of Fields).
- NR : numéro de la ligne courante (abréviation anglaise de Number of Records).
- FNR : numéro de la ligne courante dans le fichier courant.

PETITS EXEMPLES SIMPLES

Pour terminer cette étude du langage GAWk, nous allons voir quelques exemples des fonctionnalités et de la puissance de traitement du langage.

Exemple 1 : ce premier exemple va permettre la visualisation de toutes les lignes d'un fichier nommé toto, dont la taille est supérieure à 72 caractères. Cet exemple est des plus simples et ne nécessite même pas d'écrire un programme complet ; on passera directement la commande sous la forme :

```
GAWk 'length > 72' toto
```

La fonction *length*, lorsqu'elle ne possède pas de paramètres, renvoie la longueur de la ligne d'entrée.

Exemple 2 : cette fois on complique un peu. Nous allons visualiser toute les lignes du fichier toto comprises entre deux mots-clefs 'start' et 'stop'. La commande à envoyer à GAWk pour faire le traitement est donc la suivante :

```
GAWk '/start/, /stop/' toto
```

L'indication d'une chaîne de caractères se fait en la limitant par deux slashes.

Exemple 3 : nous allons maintenant réaliser notre premier vrai programme, que nous appellerons somme.awk. Il traite un fichier toto en additionnant tous les nombres correspondant à la deuxième colonne de chaque ligne du fichier.

Voici le fichier d'exemple.

121	12	14.5
126	14	18.5
12	14	125

le programme à réaliser est donc le suivant :

```
{
    somme += $2
}
END
{
    print " somme : ", somme;
    print " moyenne : ", somme/NR;
}
```

La première action ajoute la valeur du second champ de chaque ligne à la variable somme. Le mot clé *END* situé avant la deuxième action, indique qu'elle ne devra être exécutée qu'après que toutes les lignes de tous les fichiers d'entrée aient été traitées par la première. Enfin, *NR* est la variable réservée contenant le nombre total de lignes lues.

Exemple 4 : pour notre dernier exemple, nous allons inverser les colonnes du fichier toto, c'est-à-dire inverser les champs de chaque ligne.

Le programme est très simple, le voici :

```
awk '{print $3, $2, $1 }' titi
```

Voilà pour les exemples, il ne nous reste plus qu'à savoir où trouver le logiciel.

OU TROUVER GAWk ?

GAWk est disponible, vous vous en doutez bien, dans le domaine public. La version que j'ai testée se trouve sur la disquette Fish numéro 406. Outre le programme proprement dit, vous trouverez également une documentation (en anglais) dans le plus pur style "man" d'Unix, ainsi qu'un certain nombre de fichiers textes expliquant le travail des auteurs. Pour ceux qui voudraient en savoir plus sur Awk, je vous conseille la "bible" de ce langage : *"the Awk Programming Language"*, écrit par les trois auteurs d'Awk eux-mêmes et édité par Bell Telephone Laboratories, Incorporated.

En conclusion, je vous recommande fortement d'étudier ce petit langage de traitement de fichiers ASCII qu'est GAWk. Le mois prochain, nous continuerons notre exploration du monde Unix avec Flex, la version Amiga du célèbre pré-processeur C, très utile pour générer (entre autres choses) des analyseurs syntaxiques, j'ai nommé Lex. A bientôt donc, si vous le voulez bien !

Par Herr Doktor Von GlutenStimmellImDorf

DEMOS: SCROLLING "KICK-OFF"

Le mois dernier, je vous avais promis le scrolling le plus rapide possible. Voici qui est prometteur, mais beaucoup d'entre vous se sont peut-être dit que ce n'était là qu'un moyen malhonnête d'attirer l'attention du lecteur...

Je viens de me rendre compte qu'il est possible que certains croient que je parle de la technique très répandue dans le domaine des démos, qui consiste à créer un gros bitplan et à faire bouger l'écran dessus... Cette pratique est non seulement très simple à programmer, mais aussi très rapide en temps d'exécution (en fait, plus rapide que la technique dite Kick-Off), mais les contraintes mémoire qu'elle entraîne la rendent inutilisable dans un jeu digne de ce nom.

Le gros avantage du scrolling que je vais vous décrire, provient du fait qu'il permet de ne gérer que les nouveaux blocs de 16 x 16 pixels et donc, si l'on n'affiche que le strict nécessaire, on ne peut pas faire moins (à l'exception de ne rien faire, et pour cela je ne connais que la technique du gros bitplan).

Après cette introduction, je vais vous expliquer plus précisément comment programmer un tel scrolling et quelles sont ses limites.

LE PRINCIPE

Tout d'abord, le scrolling horizontal. L'écran doit être défini à la taille voulue, en ajoutant un modulo de 2 octets permettant d'utiliser le scrolling hard (registre BPLCON1). Pour procéder au scrolling, on calcule la direction en comparant la position actuelle (variables ori_x, ori_y) à l'ancienne (variables old_ori_x, old_ori_y). Si le scrolling va vers la droite, on affiche une nouvelle colonne de blocs à droite de l'écran et s'il va vers la gauche, on l'affiche devinez où ? Hé bien oui, à gauche, bien joué, vous reviendrez en deuxième semaine.

Pour voir le scrolling bouger, on incrémente les pointeurs bitplans pour la droite et on les décrémente pour la gauche, mais uniquement quand cela est nécessaire, c'est-à-dire tous les 16 pixels. Le spectateur a ainsi l'impression que l'écran tout entier bouge dans la direction voulue. Comme vous l'avez sûrement deviné, on peut aller très loin dans la mémoire comme cela et c'est en effet le seul défaut de cette technique : il n'est pas infini en horizontal. Par exemple, pour un scrolling sur 80 écrans de large, il faut prévoir 80 lignes supplémentaires en bas de la page. Ce défaut n'est pas vraiment excessif, compte tenu des avantages. De plus, avec le double-buffering qui est nécessaire dans la majorité des jeux ou autres applications, nous ne sommes pas forcés d'avoir deux fois ce supplément. En effet, quand une page va descendre, l'autre aussi, donc si on espace un peu les 2 pages pour ne pas qu'elles se chevauchent au moment des changements de direction, le supplément de ligne n'est que de 80 pour les deux pages.

Passons maintenant au scrolling vertical, qui est lui plus efficace car infini. Il est en fait la partie forte de la technique grâce à la manière originale dont il est géré. Il a besoin pour pouvoir s'exprimer pleinement, d'une hauteur de playfield égale à celle de l'écran plus 16 pixels. Tout se fait grâce au Copper : quand le scroll descend, on fait descendre le pointeur des bitplans sur la page. Après avoir descendu 16 pixels, la page n'est plus assez grande pour descendre encore de cette manière. Quand le scrolling est dans cette position, on remarque que l'on ne se sert pas des 16 premiers pixels de la page, donc la ruse consiste à afficher la nouvelle ligne de blocs à cet endroit. Pour donner l'illusion au spectateur que le scrolling continue normalement, on interrompt au moment opportun le balayage-écran avec le Copper pour changer les pointeurs qui montrent la zone contenant les nouveaux blocs.

Dans cette posture, on peut encore descendre de 16 pixels. Arrivé à cette nouvelle limite, on affiche encore une nouvelle ligne en bas de l'écran, donc en fait sur le haut de la page, entre les lignes 16 et 32, et ainsi de suite pour le reste de la descente.

Quand le premier pointeur arrive en bas de la page, on le fait remonter tout en haut et on peut continuer comme cela très longtemps. Pour faire monter le scroll, le procédé est identique, à l'exception du pointeur bitplan qui monte et de la nouvelle ligne de blocs qui est affichée en haut de l'écran.

On détecte dans un premier temps la direction de la même manière que dans le scrolling horizontal pour savoir si on affiche une ligne de blocs en haut ou en bas de l'écran.

REALISATION

Ceux qui ont tout suivi se sont très probablement aperçus que le scrolling vertical empêchait l'utilisation du scrolling horizontal tel qu'il a été décrit plus haut. Pour avoir un scrolling multidirectionnel, il faut obtenir une parfaite compatibilité entre les deux. Pour ce faire, on va modifier le scrolling horizontal et plus précisément la routine qui affiche les nouvelles colonnes : on fait en sorte que le premier bloc de la colonne s'affiche à l'endroit que montre le premier pointeur bitplan. On affiche chaque bloc en dessous de l'autre, en testant à chaque fois si l'on arrive en bas de la page ; si oui, on affiche les prochains blocs en haut de la page. De cette façon, on peut avoir un scrolling tout-à-fait intéressant, car il permet un déplacement de 16 pixels en X et en Y simultanément, le tout en cinquante rasterlines pour un écran de 320 x 256 en 4 bitplans.

Dans le cas de l'utilisation du double-buffering, le pas maximum descend à 8 pixels en X et en Y.

Voilà, c'est fini ; le mois prochain vous aurez très probablement droit à quelque chose plus proche de la démo que du jeu, comme par exemple une routine de plasma ou un ciel étoilé multidirectionnel.

La phrase du mois : *"si vous n'affrontez pas vos terreurs en face, elles vous grimpent dans le dos"*.

**** LISTING DE JEROME ETIENNE POUR L'ANT
** sur le scroll KICK OFF**

```
OPT C-

incdir      'include:'
include     'exec/exec_lib.i'
include     'hardware/custom.i'

custom      =      $dff000
execbase    =      4

BPL_X       =      320
BPL_Y       =      192
BPL_DEPTH   =      4
MODULO      =      2
BPL_WIDTH   =      BPL_X/8+MODULO
BPL_SIZE    =      BPL_WIDTH*BPL_Y
SCR_SIZE    =      BPL_SIZE*BPL_DEPTH
MAP_WIDTH   =      35
MAP_HEIGHT  =      35
BLK_WIDTH   =      16
BLK_HEIGHT  =      16
SCROLL_SPEED =      8
vsync:      macro
.wait_vsync@:
    move.l    vposr(a5),d0
    and.l    #$1ff00,d0
    cmp.l    #$11000,d0
    bne.s    .wait_vsync@
endm

wait_blt:   macro
```



```

        btst    #14,dmaconr(a5)
.loop_wait_blt@:
        btst    #14,dmaconr(a5)
        bne.s   .loop_wait_blt@
        endm

MOD: MACRO    \1,\2      * \1 chiffre a modular \2 base du
modulo
.MODULO_GO_ON@:
        cmp.w   \2,\1
        blt.s   .MODULO_IS_FINISH@
        sub.w   \2,\1
        bra.s   .MODULO_GO_ON@
.MODULO_IS_FINISH@:
        ENDM

***** programme principal *****
move.l (execbase).w,a6
lea     custom,a5
CALLEXEC Forbid
move.w #03e0,dmacon(a5) * all dma off except disk
move.w #(BPL_DEPTH<<12)+$200,bplcon0(a5)
clr.w  bplcon1(a5)
clr.w  bplcon2(a5)
move.w #BPL_WIDTH*(BPL_DEPTH-1),bpl1mod(a5)
move.w #BPL_WIDTH*(BPL_DEPTH-1),bpl2mod(a5)
move.w #$2981,diwstrt(a5) * \
move.w #$e9c0,diwstop(a5) * >ecran 320*192
move.w #$0030,ddfstrt(a5) * >avec un mot en plus
move.w #$00d0,ddfstop(a5) * / pour le shift
bsr     build_coplist
move.l coplist_adr,copl1c(a5) * >run my coplist
clr.w  copjmp1(a5)           * /
move.w #$83c0,dmacon(a5) * dma blit,cop & bpl
move.w #$f00,color+15*2(a5)
move.w #$0f0,color+1*2(a5)
move.w #$00f,color+6*2(a5)

bsr     INIT_FIRST_SCR
main_loop:
vsync
move.w #-1,color(a5)
bsr     INIT_SCR_PT_ADR
bsr     SCROLL
bsr     build_coplist
bsr     CHANGE_DIRECTION_WITH_KEYBOARD
tst.w  ORI_X
bge.s  .ORI_X_NE_SORT_PAS_A_GAUCHE
clr.w  ORI_X
.ORI_X_NE_SORT_PAS_A_GAUCHE
cmp.w  #MAP_WIDTH*16-BPL_X-1,ORI_X
ble.s  .ORI_X_NE_SORT_PAS_A_DROITE
move.w #MAP_WIDTH*16-BPL_X-1,ORI_X
.ORI_X_NE_SORT_PAS_A_DROITE
tst.w  ORI_Y
bge.s  .ORI_Y_NE_SORT_PAS_EN_HAUT
clr.w  ORI_Y
.ORI_Y_NE_SORT_PAS_EN_HAUT
cmp.w  #MAP_HEIGHT*16-BPL_Y-1,ORI_Y
ble.s  .ORI_Y_NE_SORT_PAS_EN_BAS
move.w #MAP_HEIGHT*16-BPL_Y-1,ORI_Y
.ORI_Y_NE_SORT_PAS_EN_BAS
clr.w  color(a5)
move.b $bfec01,d0*\
not     d0      * >capture the key wich is pressed
ror.b   #1,d0   * / and put is code RAW in d0
cmp.b   #$45,d0 *\
beq     init_end * / sort si on press sur esc
btst    #6,$bfe001
bne     main_loop

***** init end *****
* reactivation de l'old coplist
init_end:
wait_blt
lea     GfxName(pc),a1      * nom de la library ds a1
moveq   #0,d0               * version 0 (the last)
CALLEXEC OpenLibrary        * lib graphique ouverte
move.l  d0,a4               * adr de graphicbase ds a4
move.l  38(a4),cop11c(a5) * chargement de l'adr de
clr.w   copjmp1(a5)         * l'old coplist et lancement
move.w  #$83e0,dmacon(a5) * activation des canaux dma
CALLEXEC Permit             * multi switching autorise

```

```

fin: moveq   #0,d0      * flag d'erreur desactive
rts

GfxName: dc.b          "graphics.library",0
even

CHANGE_DIRECTION_WITH_KEYBOARD:
move.b   $bfec01,d0 *\
not       d0      * >capture the key pressed
ror.b     #1,d0   * / and put is code RAW in d0
ext.w     d0
lea       TAB_DELTA_X_Y_ACOORDING_TO_KEYBOARD,a0
.TRY_ANOTHER_KEY:
tst.w     (a0)
blt.s     .TST_KEY_FINISH
cmp.w     (a0),d0
bne.s     .NOT_THIS_KEY
move.w    2(a0),d0
add.w     d0,ORI_X
move.w    4(a0),d0
add.w     d0,ORI_Y
bra.s     .TST_KEY_FINISH
.NOT_THIS_KEY:
addq.l    #6,a0
bra.s     .TRY_ANOTHER_KEY
.TST_KEY_FINISH:
rts
s         SET      SCROLL_SPEED
TAB_DELTA_X_Y_ACOORDING_TO_KEYBOARD:
dc.w      $3f,s,-s,$3e,0,-s,$3d,-s,-s,$2d,-
s,0,$2f,s,0,$1d,-s,s
dc.w      $1e,0,s,$1f,s,s
dc.w      -1

INIT_SCR_PT_ADR:      move.l LOG_SCR_ADR,a0
move.w     ORI_X,d0
and.w      #$fff0,d0
lsr.w      #3,d0
lea        (a0,d0.w),a0
move.l     a0,SCR_PT_ADR
rts

SCROLL:      move.w  ORI_Y,d0
move.w      #BPL_Y+BLK_HEIGHT,d1
MOD          d0,d1
move.w      d0,START_FIRST_PART
neg.w       d0
add.w       #BPL_Y+BLK_HEIGHT,d0
cmp.w       #BPL_Y,d0
ble.s       .FIRST_PART_SIZE_NOT_BIGGER_THAN_SCREEN
move.w      #BPL_Y,d0
.FIRST_PART_SIZE_NOT_BIGGER_THAN_SCREEN
move.w      d0,FIRST_PART_SIZE
move.w      ORI_Y,d0
add.w       #BPL_Y,d0
move.w      #BPL_Y+BLK_HEIGHT,d1
MOD          d0,d1
move.w      d0,END_SECOND_PART
lea         MAP,a0
move.w      ORI_X,d0
lsr.w       #4,d0
lea         (a0,d0.w),a0
move.w      ORI_Y,d0
lsr.w       #4,d0
mulu        #MAP_WIDTH,d0
add.l       d0,a0
move.l      a0,MAP_PT
WAIT_BLT
move.w      #$09f0,BLTCON0(a5)
clr.w       BLTCON1(a5)
move.w      #-1,BLTAFWM(a5)
move.w      #-1,BLTALWM(a5)
move.w      #BPL_WIDTH-2,BLTDMOD(a5)
clr.w       BLTAMOD(a5)
* maintenant on affiche les blk pour le scroll en x
move.w      ORI_X,d0
cmp.w       OLD_ORI_X,d0
ble.s       .DONT_SCROLL_TOWARD_RIGHT
move.w      OLD_ORI_X,d0
add.w       #BPL_X-1,d0
and.w       #$fff0,d0
move.w      ORI_X,d1
add.w       #BPL_X-1,d1
and.w       #$fff0,d1
cmp.w       d0,d1

```



```

beq.s .DONT_SCROLL_TOWARD_RIGHT
move.l SCR_PT_ADR,a0
move.w START_FIRST_PART,d0
and.w #$fff0,d0
mulu #BPL_WIDTH*BPL_DEPTH,d0
add.l d0,a0
lea BPL_WIDTH-MODULO(a0),a0
move.l MAP_PT,a1
add.l #BPL_X/16,a1
bsr AFF_A_COLUMN_OF_BLK
.DONT_SCROLL_TOWARD_RIGHT:
move.w ORI_X,d0
cmp.w OLD_ORI_X,d0
bge.s .DONT_SCROLL_TOWARD_LEFT
move.w OLD_ORI_X,d0
and.w #$fff0,d0
move.w ORI_X,d1
and.w #$fff0,d1
cmp.w d0,d1
beq.s .DONT_SCROLL_TOWARD_LEFT
move.l SCR_PT_ADR,a0
move.w START_FIRST_PART,d0
and.w #$fff0,d0
mulu #BPL_WIDTH*BPL_DEPTH,d0
add.l d0,a0
move.l MAP_PT,a1
bsr AFF_A_COLUMN_OF_BLK
.DONT_SCROLL_TOWARD_LEFT:
* maintenant on affiche les blk pour le scroll en y
move.w ORI_Y,d0
cmp.w OLD_ORI_Y,d0
ble.s .DONT_SCROLL_TOWARD_DOWN
move.w OLD_ORI_Y,d0
add.w #BPL_Y-1,d0
and.w #$fff0,d0
move.w ORI_Y,d1
add.w #BPL_Y-1,d1
and.w #$fff0,d1
cmp.w d0,d1
beq.s .DONT_SCROLL_TOWARD_DOWN
move.l SCR_PT_ADR,a0
move.w END_SECOND_PART,d0
and.w #$fff0,d0
mulu #BPL_WIDTH*BPL_DEPTH,d0
add.l d0,a0
move.l MAP_PT,a1
add.l #(BPL_Y/16)*MAP_WIDTH,a1
bsr AFF_A_LINE_OF_BLK
.DONT_SCROLL_TOWARD_DOWN:
move.w ORI_Y,d0
cmp.w OLD_ORI_Y,d0
bge.s .DONT_SCROLL_TOWARD_UP
move.w OLD_ORI_Y,d0
and.w #$fff0,d0
move.w ORI_Y,d1
and.w #$fff0,d1
cmp.w d0,d1
beq.s .DONT_SCROLL_TOWARD_UP
move.l SCR_PT_ADR,a0
move.w START_FIRST_PART,d0
and.w #$fff0,d0
mulu #BPL_WIDTH*BPL_DEPTH,d0
add.l d0,a0
move.l MAP_PT,a1
bsr AFF_A_LINE_OF_BLK
.DONT_SCROLL_TOWARD_UP:
move.w ORI_X,OLD_ORI_X
move.w ORI_Y,OLD_ORI_Y
rts

ORI_X:      ds.w 1      *\ coor du coin en haut a
gauche
ORI_Y:      ds.w 1      */
OLD_ORI_X:  ds.w 1
OLD_ORI_Y:  ds.w 1
START_FIRST_PART: ds.w 1
END_SECOND_PART: ds.w 1
FIRST_PART_SIZE: ds.w 1
MAP_PT:     ds.l1

AFF_A_COLUMN_OF_BLK: * IN: a0= adr ou on va poser les
blk
* a1= adr de la map ou on va chercher quel blk

```

```

afficher
* OUT: rien
lea TAB_ADR_BLK_GFX,a2
moveq #0,d2
move.w #BPL_Y/16-1+1,d0
.LOOP_AFF_EACH_BLK:
move.b (a1),d1
lea MAP_WIDTH(a1),a1
ext.w d1
lsl.w #2,d1
move.l (a2,d1.w),d1
WAIT_BLT
move.l a0,BLTDPT(a5)
move.l d1,BLTAPT(a5)
move.w #BLK_HEIGHT*BPL_DEPTH*64+BLK_WIDTH/16,BLTSIZE(a5)
lea BLK_HEIGHT*BPL_WIDTH*BPL_DEPTH(a0),a0
tst.w d2
bne.s .SECOND_PART_ALREADY_BEGIN
move.l SCR_PT_ADR,a3
add.l #(BPL_Y+BLK_HEIGHT-1)*BPL_WIDTH*BPL_DEPTH,a3
cmp.l a3,a0
ble.s .FIRST_PART_NOT_TOTALLY_PRINT
sub.l #(BPL_Y+BLK_HEIGHT)*BPL_WIDTH*BPL_DEPTH,a0
moveq #-1,d2
.SECOND_PART_ALREADY_BEGIN
.FIRST_PART_NOT_TOTALLY_PRINT:
dbf d0,.LOOP_AFF_EACH_BLK
rts
END_AFF_A_COLUMN_OF_BLK

AFF_A_LINE_OF_BLK:
*****
* IN: a0= adr ou on va poser les blk
* a1= adr de la map ou on va chercher quel blk
afficher
* OUT: rien
lea TAB_ADR_BLK_GFX,a2
moveq #BPL_WIDTH/2-1,d0
.LOOP_AFF_EACH_BLK:
move.b (a1)+,d1
ext.w d1
lsl.w #2,d1
move.l (a2,d1.w),d1
WAIT_BLT
move.l a0,BLTDPT(a5)
move.l d1,BLTAPT(a5)
move.w #BLK_HEIGHT*BPL_DEPTH*64+BLK_WIDTH/16,BLTSIZE(a5)
addq.l #BLK_WIDTH/8,a0
dbf d0,.LOOP_AFF_EACH_BLK
rts
END_AFF_A_LINE_OF_BLK

INIT_FIRST_SCR:
*****
clr.w ORI_X
clr.w ORI_Y
clr.w OLD_ORI_X
clr.w OLD_ORI_Y
* on init les registres blitter qui resteront ainsi
durant toute la routine
WAIT_BLT
move.w #$09f0,BLTCON0(a5)
clr.w BLTCON1(a5)
move.w #-1,BLTAFWM(a5)
move.w #-1,BLTALWM(a5)
move.w #BPL_WIDTH-2,BLTDMOD(a5)
clr.w BLTAMOD(a5)

* maintenant on affiche le premier ecran
move.l LOG_SCR_ADR,a0
move.l PHY_SCR_ADR,a3
lea MAP,a1
lea TAB_ADR_BLK_GFX,a2
moveq #BPL_Y/16+1-1,d2
.LOOP_AFF_EACH_LINE:
move.w #BPL_WIDTH/2-1,d0
.LOOP_AFF_EACH_BLK:
move.b (a1)+,d1
ext.w d1
lsl.w #2,d1
move.l (a2,d1.w),d1
WAIT_BLT
move.l a0,BLTDPT(a5)

```


[illegible]

dc.b	0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0
dc.b	0,0,1,1,1,1,1,0,0,1,0,0,1,0,0,1,0,0
dc.b	0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0
dc.b	0,0,1,0,0,0,1,0,0,1,0,0,0,1,0,1,0,0
dc.b	0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,1,0,0,0,1,0,0,1,0,0,0,0,1,1,0,0
dc.b	0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0
dc.b	0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,1,0,0
dc.b	0,0,1,0,0,0,0,0,0,0,0,1,1,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,1,0
dc.b	0,0,0,0,1,0,0,0,1,1,0,1,0,0,0,1,0
dc.b	0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0
dc.b	0,0,0,1,0,1,0,1,0,0,0,1,1,0,1,1,0
dc.b	0,0,1,1,1,0,0,1,0,0,1,0,1,0,0,0,1,0
dc.b	1,1,0,1,0,1,0,0,1,0,0,1,0,1,0,1,0
dc.b	0,0,1,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0
dc.b	0,0,0,1,1,1,0,0,0,1,0,1,0,0,0,1,0
dc.b	0,0,1,0,0,0,0,0,1,1,0,0,1,1,1,0,1,1
dc.b	1,0,0,1,0,1,0,1,1,0,0,1,0,0,0,1,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0
dc.b	1,1,1,0,1,1,1,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0
dc.b	0,1,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,1,1,0
dc.b	0,1,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0,1,0
dc.b	0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0
dc.b	1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,1,1,1,1,0,0,0,0,0,0,1,0,0,0,1,0
dc.b	0,0,0,0,1,0,0,0,1,1,0,1,0,0,0,1,0
dc.b	0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0
dc.b	0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0
dc.b	0,0,1,0,0,0,0,1,0,0,1,0,1,0,0,0,1,0
dc.b	0,0,0,1,0,0,0,1,0,0,0,1,1,0,0,0,1,0
dc.b	1,0,0,1,0,1,1,1,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,1,0,0,1,0,1,0
dc.b	0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0
dc.b	1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0
dc.b	0,1,0,0,0,0,0,0,0,0,1,0,0,0,0,0,1,1,0
dc.b	0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,1,1,0,0,1,0,1,0
dc.b	1,1,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dc.b	0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1

```

                sectionGFX,DATA_C
BLK0_GFX:      REPT      16
                dc.w      -1,-1,-1,-1
                ENDR
BLK1_GFX:      REPT      16
                dc.w      -1,0,0,0
                ENDR
BLK2_GFX:      REPT      16
                dc.w      0,-1,-1,0
                ENDR

```

```

                sectionZONE_CHIP,BSS_C
                ds.1      1
ecran1:         ds.1      (scr_size*2)/4
ecran2:         ds.1      (scr_size*2)/4
coplist:        ds.1      1000*4 * taille inconnue donc on
prevoit gros
                end

```

Par Jérôme Etienne

Pour illustrer nos nouvelles connaissances sur la gestion de la mémoire par Exec, je vous propose deux petits programmes C pour le prix d'un seul.

Le premier commence par réserver un buffer de 16 Ko et lui attribue un MemHeader particulier. Ainsi, on peut user de ce buffer exactement comme s'il s'agissait de la mémoire principale de l'ordinateur, à ceci près que seules les fonctions Allocate() et Deallocate() sont utilisables. Ce programme est interactif, c'est-à-dire que c'est vous qui décidez des opérations à suivre (allocation ou libération d'un bloc quelconque). Le programme affiche à chaque fois l'état du MemHeader, les différents MemChunks qui peuvent résulter de la fragmentation du buffer, ainsi que l'adresse et la taille des différents blocs alloués.

L'utilisation en est très simple, bien qu'entièrement au clavier. Vous entrez une lettre de commande suivi d'un nombre x. La lettre peut être A, ce qui demande d'allouer x octets dans le buffer, ou bien L, ce qui demande de libérer le bloc numéro x, ou enfin Q, ce qui met fin au programme. Faites attention à ne bien entrer que des nombres décimaux, aucune vérification particulière n'étant faite à ce niveau.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <exec/types.h>
#include <exec/memory.h>

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

#define IsDigit(c) ((c) >= '0' && (c) <= '9')

#define BLOCSIZE 0x4000

UBYTE help[] = "\fAn=Allouer n octets, "
               "Lx=Libérer bloc n°x, "
               "Q=Quitter\n";

void main(void);
void mem_demo(struct MemHeader *mh);

void main(void)
{
    struct MemHeader *mh;
    struct MemChunk *mc;

    if (mh = (struct MemHeader *)AllocMem(sizeof(*mh),
MEMF_CLEAR))
    { if (mc = (struct MemChunk *)AllocMem(BLOCSIZE, 0))
        { mh->mh_Node.ln_Type = NT_MEMORY;
          mh->mh_Node.ln_Name = "MemDemo.chunck";
          mh->mh_First = mc;
          mh->mh_Lower = (APTR)mc;
          mh->mh_Upper = (APTR)((ULONG)mc + BLOCSIZE);
          mh->mh_Free = BLOCSIZE;

          mc->mc_Next = NULL;
          mc->mc_Bytes = BLOCSIZE;

          mem_demo(mh);

          FreeMem((APTR)mc, BLOCSIZE);
        }
        FreeMem((APTR)mh, sizeof(*mh));
    }
    exit(0);
}

void mem_demo(struct MemHeader *mh)
```

```
{
    UBYTE cmd[128], c, *ptr, *adr;
    ULONG blocs[32], tailles[32], taille;
    SHORT bloc, nb_blocs = 0, i;
    BOOL fini = FALSE;
    struct MemChunk *mc;

    for (i = 0; i < 32; i++)
        blocs[i] = tailles[i] = -1L;

    while (!fini)
    { puts(help);

        printf("MemHeader :");
        printf(" Lower=%08lx, Upper=%08lx, "
Free=%08lx\n\n",
            mh->mh_Lower, mh->mh_Upper, mh->mh_Free);

        for (i = 0, mc = mh->mh_First; mc; mc = mc->mc_Next)
            printf("MemChunk %2d, Next=%08lx, Bytes=%08lx\n",
                i++, mc->mc_Next, mc->mc_Bytes);

        printf("\n");
        for (i = 0; i < nb_blocs; i++)
            printf("Bloc %2d, adresse: %08lx, taille: "
%08lx\n",
                i, blocs[i], tailles[i]);

        printf("\n[An|Lx|Q] > ");
        gets(ptr = cmd);

        while (c = *ptr++)
        { switch (c | 0x20)
            { case 'q': /* Quitter */
                fini = TRUE;
                break;

            case 'a': /* Allouer */
                c = *ptr++;
                while (!IsDigit(c)) c = *ptr++;
                taille = c - '0';

                c = *ptr++;
                while (IsDigit(c))
                { taille = (taille * 10) + (c - '0');
                  c = *ptr++;
                }
                *ptr = '\0';

                if (nb_blocs < 32)
                { if (taille)
                    { adr = Allocate(mh, taille);
                      if (adr)
                      { printf("Allocation de %ld octets\n",
taille);
                        blocs[nb_blocs] = (ULONG)adr;
                        tailles[nb_blocs] = taille;
                        nb_blocs++;
                      }
                      else puts("L'allocation a échoué !");
                    }
                    else puts("Taille incorrecte.");
                }
                else puts("Nombre maximum de bloc atteint.");
                break;

            case 'l': /* Libérer */
            case 'f':
                c = *ptr++;
                while (!IsDigit(c)) c = *ptr++;
                bloc = c - '0';

                c = *ptr++;
                while (IsDigit(c))
                { bloc = (bloc * 10) + (c - '0');
                  c = *ptr++;
                }
                *ptr = '\0';

                if (bloc < nb_blocs)
                { if (blocs[bloc] != -1L)
```




```

        { printf("Libération du bloc n°%ld\n\n",
bloc);
        Deallocate(mh, (APTR)blocs[bloc],
tailles[bloc]);
        for (i = bloc; i < 32; i++)
        { blocs[i] = blocs[i+1];
          tailles[i] = tailles[i+1];
        }
        --nb_blocs;
    }
    else puts("Bloc non réservé.");
}
else puts("Numéro de bloc invalide.");
break;
}
}
}
}

```

ChipMemFirst

Ce tout petit programme est un petit truc pour vous permettre de modifier la priorité d'allocation de la mémoire Chip. Il se contente de changer la priorité de la liste CHIP dans la structure ExecBase. Ainsi, et sous réserve qu'il y en ait suffisamment, ce sera en priorité de la mémoire Chip qu'AllocMem() et consœurs alloueront. Cela peut être utile par exemple, pour charger une ancienne démo qui refuse de tourner correctement avec une extension de mémoire.

Par défaut, la nouvelle priorité est de 10, ce qui devrait être amplement suffisant dans la plupart des cas (Exec l'initialise à -10, la Fast Ram étant à 0). Mais il peut arriver que certaines extensions, notamment 32 bits sur les cartes 68030, aient des priorités supérieures. C'est pourquoi il est possible de préciser en paramètre sur la ligne de commande, la nouvelle priorité à appliquer à la mémoire Chip.

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <exec/types.h>
#include <exec/memory.h>
#include <exec/execbase.h>

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

extern struct ExecBase *SysBase;

void main(int argc, char **argv)
{
    int newpri = 10;
    struct MemHeader *mh;

    if (argc > 1)
        newpri = atoi(argv[1]);

    Disable();
    if (mh = FindName(&SysBase->MemList, "Chip Memory"))
    { Remove((struct Node *)mh);
      mh->mh_Node.ln_Pri = newpri;
      Enqueue(&SysBase->MemList, (struct Node *)mh);
    }
    Enable();
}

```

Par Philippe Vautrin

; *****

Affiche_interr:

```

suba.l a0,a0
bsr SetMenu
lea ititre(pc),a0
bsr PTitre
clr.w lignes(a5)
movea.l $4.w,a6
lea IntVects(a6),a6
move.l #intypes,d3
move.w #1110001111111000,d1
moveq #15,d2
AffInts btst d2,d1
        beq.s .serv
.handl1 lea lg_Data(a4),a1
        move.l a6,(a1)
        move.l IV_CODE(a6),4(a1)
        beq.s .nxtint
        move.l IV_NODE(a6),d0
        beq.s .nxtint
        movea.l d0,a0
        move.b LN_PRI(a0),d0
        ext.w d0
        move.w d0,8(a1)
        move.w #'H',10(a1)
        move.l d3,12(a1)
        move.l LN_NAME(a0),16(a1)
        bne.s .ok1
        move.l #Defaut,16(a1)
.ok1 lea ifmt(pc),a0
      bsr DoFmt
      bra.s .nxtint
.serv move.l IV_DATA(a6),a2
      lea AffServ(pc),a3
      bsr AffList
.nxtint addq.l #8,d3
      lea IV_SIZE(a6),a6
      dbra d2,AffInts
      rts

```

```

AffServ move.l a2,(a1)
        move.l IV_CODE(a6),4(a1)
        move.b LN_PRI(a2),d0
        ext.w d0
        move.w d0,8(a1)
        move.w #'S',10(a1)
        move.l d3,12(a1)
        move.l LN_NAME(a2),16(a1)
        bne.s .ok2
        move.l #Defaut,16(a1)
.ok2 lea ifmt(pc),a0
      rts

```

```

ititre dc.b .len-*
        dc.b " N° Node Code Pri Type No"
.len dc.b "m"
even
ifmt dc.b "%3d %08lx %08lx %3d %c %-7.7s %s",0
even
intypes dc.b "TBE DSKBLK SOFTINT PORTS "
        dc.b "COPER VERTB BLIT AUD0 "
        dc.b "AUD1 AUD2 AUD3 RBF "
        dc.b "DSKSYNC EXTER INTEN NIM "
        even

```

; *****

Affiche_Quitter:

```

st fini(a5)
rts

```

; *****

```

VARS dcb.b VARSIZE
dosname dc.b "dos.library",0
intname dc.b "intuition.library",0
gfxname dc.b "graphics.library",0,0
Defaut dc.b "-----",0,0

```

; *****

```

include "em_windows.s"
include "em_menus.s"

```

; *****

```

section BUFFER,BSS

```

```

txtbuff ds.b lg_SIZE*100 ;ça doit être assez sinon boum !
END

```

Par Max

Une petite lettre plus que sympathique est parvenue à la rédaction d'Amiga Revue, celle d'un jeune roumain qui, se sentant isolé, recherche des contacts de programmation. Séquence émotion.

Cher Monsieur, je suis un jeune roumain passionné d'informatique. Très heureux possesseur d'un ordinateur Amiga 1000, j'avais commencé d'étudier le langage A-BASIC de cet ordinateur. Seulement avec cet A-BASIC, et sans avoir accès à d'autres programmes, je me sens un peu désorienté. Mais le problème est que je n'ai pas la possibilité de communiquer et d'échanger des idées avec d'autres possesseurs d'ordinateur. Pour ça je veux, bien sûr si possible, entrer en contact avec d'autres possesseurs d'Amiga qui pourraient m'initier à la technique de programmation en Assembleur et en C, et dans la création graphique. Mon adresse est

Càlin Butiu
Str Giurgiu NR 15 AP 35
1900 Timisoara
Roumanie

Avec mes amitiés, merci.

Cette lettre nous est parvenue telle quelle, c'est-à-dire dans un français presque parfait (à quelques fôtes d'orthographe près). Nous espérons très sincèrement que vous serez nombreux à répondre à Càlin et qu'il pourra rapidement progresser sur l'Amiga.

Zig Herr Doktor... Shaitan et moi-même, fervents lecteurs d'Amiga NewsTech, avons un problème en C : comment générer un nombre aléatoire sans rentrer dans une prise de tête mathématique, simplement en utilisant les fonctions contenues dans notre Lattice 5.0... Notre mémoire se souvient de ces temps reculés, à l'heure de nos premiers balbutiements informatiques, où une simple commande A=RND(1) suffisait pour obtenir, d'une manière réellement aléatoire, une valeur entre 0 et 0,99. Mais c'était l'âge de la micro-magnon et du Basic. Aujourd'hui, c'est C et les micro-sapiens que nous sommes appelent à l'aide : qu'est-ce donc que ces fonctions rand(), drand()... qui renvoient inlassablement la même suite de nombres ?

Cyrille Giquello (MAD), Issy les Moulineaux

Désolé si ce n'est pas le Doktor qui répond, mais il est en train de se la couler douce sur un bateau, quelque part au large des côtes françaises, là où y'a du soleil même en septembre... On prend ses vacances quand on peut.

D'abord, cette bonne vieille instruction Basic RND() ne renvoyait pas un nombre si aléatoire que ça, mais plutôt pioché "au harsard" dans une table... Et encore, l'expression "au hasard" est tout-à-fait subjective, puisqu'une suite de calculs autant mathématiques que précis donnait l'indice du nombre dans la table...

D'ailleurs, les fonctions rand() et drand() agissent exactement de la même manière, à ceci près qu'il ne s'agit plus de piocher dans une table, mais de calculer directement le nouveau nombre aléatoire. Je ne rentrerai pas dans les détails de la formule, qui peut varier d'un compilateur C à un autre, mais l'avantage d'une telle méthode est simplement de permettre à un programmeur, pour des raisons de débogage, de re-crée la même suite de "nombres aléatoires" autant de fois qu'il le désire.

Ces deux fonctions agissent de la même manière : une valeur de départ, baptisée "graine" (seed en anglais), sert de base à la génération de la suite de nombres. Si cette graine ne change pas entre plusieurs exécutions du programme, la suite non plus.

Il est fort heureusement possible de modifier cette graine, en appelant la fonction srand(new_seed), ce qui correspond en gros à ce bon vieux RANDOMIZE du Basic. Si la nouvelle graine est une constante, la suite "aléatoire" ne le sera bien sûr pas

vraiment... Mais il est possible de lui donner une valeur (presque) quelconque avec le petit bout de code suivant :

```
USHORT seed = *(USHORT *)0xdff006;
srand(seed);
```

rand() renvoie un entier de type USHORT, drand() un réel de type DOUBLE, etc. Chaque bibliothèque du Lattice possède sa propre fonction rand(), en fonction du type des données manipulées. Consultez votre documentation.

Un dernier point sur les nombres aléatoires avec le Lattice : Frédéric Mazué avait mis en avant et corrigé un bug des fonctions drand48() et erand48()... Ca se passait dans l'ANT numéro 22 du mois de mai 1991.

Bonjour à tous. Si vous êtes en manque d'imagination pour créer de nouvelles rubriques dans l'ANT, je me fais un plaisir de vous soumettre quelques idées :

- *Infographie : 2D, dessin vectoriel, 3D, Ray Tracing...*
- *Programmation : création d'un langage (interpréteur, compilateur, linker)...*
- *Algorythmie (votre confrère ST Magazine disposait d'une excellence série sur ce sujet).*
- *Mathématiques scientifiques et financières.*
- *etc, etc.*

Vous le voyez, ce sont là des sujets généraux, plutôt que bien arrêtés sur le système d'exploitation (Intuition, Exec). Il y a matière à beaucoup d'articles, qui seront sans aucun doute appréciés de nombreux lecteurs.

Laurent Cadéac, Rennes

Oui, ce sont là de bonnes idées, mais nous pensons qu'avant d'aborder des sujets aussi généraux, il faut d'abord savoir programmer la machine elle-même ! Cela dit, nous sommes conscients du fait qu'on ne vas pas pouvoir vous parler éternellement des fenêtres, des Bobs ou des interruptions... C'est pourquoi nous préparons d'ores et déjà de nouvelles rubriques, parmi lesquelles la télématique, la compression de données, des algorithmes divers et variés, les micro-processeurs 68020 et 68030... Comme disait le prophète, "chaque chose en son temps et les vaches seront bien gardées".

Bonjour à toute la rédaction. Cette lettre s'adresse plus spécialement à votre spécialiste de l'assembleur, Max : j'ai besoin d'un petit conseil sur la meilleure façon de trier des données en assembleur. Je sais que tu as déjà publié un article sur le sujet dans un ancien numéro de l'ANT, mais cela ne concernait que le "tri à bulles", qui est loin d'être assez rapide dans mon cas. Pourrais-tu expliquer d'autres méthodes plus rapides, par exemple le Quick-Sort ou bien le Shell-Sort ? Je ne les connais que de nom mais on m'a certifié que c'étaient les méthodes les plus rapides.

Stéphane Rouget, Paris

Excellente idée, je me mets au travail immédiatement. J'espère pouvoir vous proposer de telles routines d'ici deux ou trois mois. Désolé ne pas pouvoir en parler dans ces pages, mais cela prendrait vraiment trop de place.

Max

Salut à toute l'équipe de l'ANT. Je vous écris pour vous signaler que je ne connais absolument rien au C et qu'il y en a beaucoup trop dans votre journal (qui est pourtant sensass !). J'espérais trouver plus de programmes en assembleur et notamment des exemples de programmation de démos ou des effets bien particuliers. Ca ne m'empêche pas de renouveler mon abonnement, mais si vous pouviez faire un petit effort...

TLP

Messieurs, j'apprécie beaucoup votre revue, bien que les trois-quarts soient incompréhensibles pour moi... Le langage C et l'assembleur

me sont parfaitement inconnus et je ne suis que plutôt médiocre en Basic GFA. J'espérais, à travers votre revue, pouvoir progresser dans ce langage et, même si je dois reconnaître apprendre beaucoup sur le système d'exploitation de l'Amiga, je suis plutôt déçu. Avez-vous l'intention prochaine de réparer cette injustice ?

Alain Souillard, Blois

Bravo pour votre magazine, qui est de loin ce qui se fait de mieux en la matière. On pourra regretter une PAO timide, quelques fautes d'orthographe ou de français qui arrivent encore à se glisser de ci et de là (beaucoup moins qu'au début en tout cas !), mais le contenu est tout-à-fait ce que j'en attendais. Programmeur professionnel en Turbo C et Turbo Pascal (sur un IBM AT 386...), je voulais seulement vous dire que l'Amiga NewsTech soutient largement la comparaison avec des revues similaires sur PC mais vendues en kiosque, à un prix équivalent et parfois plus cher. Continuez sur cette voie et un jour, qui sait, peut-être pourrons-nous trouver Amiga NewsTech dans les rayons de notre libraire habituel. C'est en tout cas tout le mal que je vous souhaite. Au fait, une dernière chose : c'est très gentil à vous et fort intelligent de fournir des langages du domaine public tel que le Pascal ou le Fortran... Mais si vous n'en parlez pas dans le magazine, la disquette ira vite se faire oublier au fond d'un tiroir...

Eric Rochant, Epinay

Ces trois lettres ne sont qu'un tout petit échantillon représentatif du courrier que nous recevons... J'aimerais une fois de plus rappeler qu'il est difficile de contenter tout le monde, même avec la meilleure volonté.

Deux rubriques sont entièrement consacrées à la programmation en mode "démon", c'est-à-dire sans utiliser le système d'exploitation. Ce sont DemoMakers et Hardware. Pour cette dernière, Loïc Far préfère aborder en premier les sujets qui ne l'ont jamais (ou très peu) été ailleurs, comme vous avez certainement pu vous en rendre compte : CIAs, drives, clavier... Copper et Blitter suivront, mais en leur temps. De plus, Transactor peut vous permettre de vous exprimer sur un sujet particulier de votre choix ; c'est votre rubrique, à vous de la remplir !

Le Basic (qu'il soit Amiga- ou GFA-) ne fait effectivement pas partie des langages dont on parle régulièrement dans l'ANT. Il y a plusieurs raisons à cela : d'abord, il faut reconnaître que ce n'est pas une franche réussite (l'Amiga-Basic est lent et lourd, le GFA-Basic est bogué jusqu'à la moëlle). Ensuite, les passionnés préfèrent l'assembleur et les pros, le C. Reste donc l'AMOS, auquel une rubrique est consacrée, signée (est-il réellement besoin de le rappeler ?) par l'auteur du langage lui-même, j'ai nommé François Lionet.

Enfin, pour expliquer l'orientation réellement "système" de l'ANT, il faut vous avouer que nous avons toujours considéré que l'Amiga est un ordinateur par trop extraordinaire pour le confiner au rôle de "console de jeu" ou de "machine à démos", aussi étonnantes soient-elles. Nous soutenons donc la programmation propre, multitâche, bref, "Amiga friendly". Ce qui ne nous empêchera pas de développer à la première occasion l'autre côté de la chose.

Je voudrais juste signaler à Loïc Far que son article "Les Disque Chouettes" sur les lecteur de disquettes était peut-être compréhensible par quelqu'un ayant déjà quelques notions sur le sujet, mais pour celui qui espérait pouvoir programmer un trackload avec... Le format MFM est abordé avec beaucoup trop de rapidité, sans parler réellement du procédé de décodage (et il n'est même pas fait mention du codage). Serait-il donc possible que monsieur Far y apporte quelque complément ?

Un lecteur au hasard parmi beaucoup d'autres

Hé, hé, hé... Je n'ai passé cette lettre que pour embêter l'ami Loïc, qui a d'ores et déjà réalisé le complément demandé. Tournez vite la page et voyez par vous-mêmes.

AMOS (suite de la page 5)

```

Procedure _TOPERAND
If SYNT : Pop Proc : End If
CHAR
If CHAR$="("
Inc NPAR : _TEVALUE[NPAR]
Else
P=Instr(FUN$, "{"+CHAR$)
If P
Repeat
For N=1 To 10
B$=Mid$(FUN$,P+N+1,1)
Exit If B$="}" , 2
Exit If B$<>Mid$(EV$,PEV+N-1,1)
Next
P=Instr(FUN$, "{"+CHAR$,P+1)
Until P=0
If P
E=P+N+2 : PILE$(PPILE)="_"+Mid$(FUN$,E,4) : Inc PPILE
PEV=PEV+N-1
NP=Val(Mid$(FUN$,E+4,1))
If NP
OPAR=NPAR
For N=1 To NP
TEVALUE[NPAR] : PILE$(PPILE)="_END" : Inc PPILE
If N<NP
CHAR:If CHAR$<>"," : SYNT=-1:Pop Proc:End If
End If
Next
If NPAR<>OPAR-1 : SYNT=-1 : Pop Proc : End If
NPAR=OPAR
End If
Else
SYNT=-1 : Pop Proc
End If
Else
If (CHAR$="-") or (CHAR$>="0" and (CHAR$<="9"))
P=PEV-1
Repeat
Inc P : A$=Mid$(EV$,P,1)
If Upper$(A$)="E"
Inc P : A$=Mid$(EV$,P,1)
If A$="+" or (A$="-")
Inc P : A$=Mid$(EV$,P,1)
End If
End If
Until (A$<>" ") and (A$<>".") and ((A$<"0") or (A$>"9"))
PILE$(PPILE)="_CONST" : PILE$(PPILE)=Val(Mid$(EV$,PEV-1,P-PEV+1))
Inc PPILE : PEV=P
Else
SYNT=-1
End If
End If
End If
End Proc
'
Procedure CHAR
If PEV>LEV
CHAR$="\ " : PEV=10000
Else
Repeat : CHAR$=Mid$(EV$,PEV,1) : Inc PEV : Until CHAR$<>" "
End If
End Proc
'
' PROCEDURES DE CALCUL
Procedure EVALUE
PPILE=-1 : EVALIT
End Proc[PP$(PP)]
'
Procedure EVALIT
Repeat
Inc PPILE : Gosub PILE$(PPILE)
Until E
Pop Proc
_END: E=1 : Return
_CONST: Inc PP : PP$(PP)=PILE$(PPILE) : Return
_PLUS: Dec PP : PP$(PP)=PP$(PP)+PP$(PP+1) : Return
_MINS: Dec PP : PP$(PP)=PP$(PP)-PP$(PP+1) : Return
_DIVD: Dec PP : PP$(PP)=PP$(PP)/PP$(PP+1) : Return
_TIME: Dec PP : PP$(PP)=PP$(PP)*PP$(PP+1) : Return
_POWR: Dec PP : PP$(PP)=PP$(PP)^PP$(PP+1) : Return
_VARS: EVALIT : EVALIT
Dec PP : A=PP$(PP) : B=PP$(PP+1) : PP$(PP)=VAR$(A,B)
Return
_SINU: EVALIT : PP$(PP)=Sin(PP$(PP)) : Return
_COSI: EVALIT : PP$(PP)=Cos(PP$(PP)) : Return
_TANG: EVALIT : PP$(PP)=Tan(PP$(PP)) : Return
_PIPi: Inc PP : PP$(PP)=Pi# : Return
_SQRR: EVALIT : PP$(PP)=Sqr(PP$(PP)) : Return
_LOGG: EVALIT : PP$(PP)=Log(PP$(PP)) : Return
_LNNN: EVALIT : PP$(PP)=Ln(PP$(PP)) : Return
_ABSS: EVALIT : PP$(PP)=Abs(PP$(PP)) : Return
_EXPP: EVALIT : PP$(PP)=Exp(PP$(PP)) : Return
_INTT: EVALIT : PP$(PP)=Int(PP$(PP)) : Return
_RNDD: EVALIT : PP$(PP)=Rnd(PP$(PP)) : Return
End Proc

```

Par Ok et Cancel

Par François Lionet

Suite à quelques courriers plutôt critiques, j'ai décidé unilatéralement de reporter au mois prochain la fin de mon article sur le clavier, pour revenir aux lecteurs de disquettes.

Quelques abonnés ont en effet été déçus de ne pas trouver dans cet article la routine promise de délai hard (à grands coups de CIA) permettant de temporiser le déplacement de la tête de lecture de piste à piste. D'autres, plus mesquins, auraient souhaité trouver plus d'informations d'une part sur le codage MFM, d'autre part sur l'utilisation du Blitter pour le décodage des données.

A tous ceux-là, je tiens d'abord à adresser mes excuses les plus plates, à défaut des plus sincères, et à rappeler ce que j'avais alors dit dans : décodage le format MFM au Blitter relève à mon avis de l'hérésie. Cela revient à se limiter volontairement à charger des données en Chip-Ram, alors qu'en utilisant le 68000, on peut utiliser un buffer de piste se trouvant n'importe où en mémoire. De plus, du fait du délai nécessaire au moteur pas à pas pour déplacer la tête de lecture/écriture, le temps gagné serait tout de suite perdu à attendre que celle-ci soit effectivement positionnée sur la piste suivante. Sans oublier que le Blitter peut toujours être utilisé pour autre chose pendant ce temps-là, par exemple pour une animation graphique pendant le chargement. Enfin, il faut bien reconnaître ce qui est, le décodage MFM est beaucoup plus facile à réaliser au 68000 !

Cela dit, voici un tout petit bout de programme assembleur qui initialise le timer A du CIA-B et compte exactement tous les 3 millisecondes, soit le temps minimum recommandé par les ingénieurs de chez Commodore-Amiga. Cet exemple ne fonctionne pas tel quel, et il vous faudra l'insérer correctement dans le programme du numéro 25.

```
; Interdiction de l'interruption CIA-B
move.w #$2000,$dff09a ; INTENA, bit 13
; Initialisation du timer A, mode one-shot, 3 ms
move.l #$bfe001,a0 ; Adresse du CIA-B
move.b $e00(a0),d0 ; Lit le registre CRA
andi.b #$c0,d0 ; Masque les bits inutilisés
ori.b #$8,d0 ; Mode one-shot
move.b d0,$e00(a0) ; Ecrit le registre
; Programme le délai : 3 ms
move.b #$28,$400(a0) ; 3ms = 2128 en PAL
move.b #$21,$500(a0) ; (2148 en NTSC)
; Attend l'expiration du délai
wait btst.b #0,$d00(a0) ; Test le registre ICR
beq.s wait
; On pourrait ici faire autre chose !
; ...
; Reprogramme le timer et boucle
bset.b #0,$e00(a0) ; Le timer redémarre
bra.s wait
```

Voilà, c'était vraiment pas grand chose, et je m'en veux de ne pas l'avoir tout de suite fait dans le programme concerné.

MFM

Vous les aviez demandées, voici quelques informations complémentaires sur le format MFM et sur la manière dont l'Amiga enregistre ses pistes et secteurs sur la disquette.

D'abord, le format MFM lui-même. Rappelons que ce format est destiné à éviter toute erreur de synchronisation lors de la lecture et de l'écriture des données, en insérant un bit de syncho entre chaque bit de donnée. Qui plus est, on code d'abord les bits impairs puis ensuite les bits pairs, dans deux buffers consécutifs en mémoire.

Donc, pour décoder un bloc de données, il suffit d'utiliser la routine suivante :

```
; d0 = nombre de mots longs à décoder - 1
; a0 pointe les bits impairs
; a1 pointe les bits pairs
; a2 pointe le buffer de décodage
Decode move.l (a0)+,d1
move.l (a1)+,d2
andi.l #$55555555,d1
andi.l #$55555555,d2
lsl.l #1,d1
or.l d2,d1
move.l d1,(a2)+
dbra d0,Decode
rts
```

Le codage est plus beaucoup plus délicat et ne sera pas abordé maintenant. Si vraiment vous désirez savoir comment procéder, écrivez-moi des tonnes de lettres, j'adore me faire injurier. J'avoue humblement ne m'être jamais attaqué à ce côté du problème mais qui sait, avec un peu de bonne volonté, on arrive à tout.

Un mot de synchronisation particulier est placé devant chaque secteur, qui permet au DMA d'en reconnaître exactement le début. Ce mot est une valeur MFM bidon, c'est-à-dire qui ne correspond à aucun codage possible (de cette manière, on est sûr qu'il s'agit bien d'une marque de syncho et non d'une donnée du secteur). AmigaDOS utilise le mot \$4489, mais n'importe quel autre mot "bidon" ferait tout aussi bien l'affaire.

LES PISTES

Une piste contient 11 secteurs constitués d'une zone d'informations diverses de 32 octets suivis d'une zone de données de 512 octets. La piste elle-même est précédée d'un GAP d'environ 700 octets. Je dis environ parce que sa taille peut varier suivant la position de la piste (près du centre ou au contraire de l'extérieur) sur la disquette. Le GAP lui-même ne sert strictement à rien pour le DOS, mais permet d'éviter, à l'écriture de la piste, que les dernières données écrites n'écrasent les premières (n'oubliez pas qu'une piste est circulaire !). Si on additionne tout ceci, on obtient une longueur de $700 + (2 * (11 * (32 + 512))) = 12668$ octets MFM par piste.

Voici résumé dans un tableau le format exact d'un secteur codé.

Offset	Valeur	Signification
0	\$aaaa	Premier mot du secteur
2	\$aaaa	Second mot du secteur
4	\$4489	Premier mot de syncho
6	\$4489	Second mot de syncho
8	xxxxxx	ID secteur (bits impairs)
12	xxxxxx	ID secteur (bits pairs)
16	xxxxxx	Données AmigaDOS (bits impairs)
32	xxxxxx	Données AmigaDOS (bits pairs)
48	xxxxxx	Checksum de l'entête (bits impairs)
52	xxxxxx	Checksum de l'entête (bits pairs)
56	xxxxxx	Checksum des données (bits impairs)
60	xxxxxx	Checksum des données (bits pairs)
64	xxxxxx	Données (bits impairs)
576	xxxxxx	Données (bits pairs)

Comme on le voit, si l'on veut décoder tout ça, il faut procéder en quatre étapes : d'abord l'ID secteur, ensuite le checksum de l'entête, puis le checksum des données, et enfin les données elles-mêmes. La routine MFMUnicode du programme de l'ANT 25 gère tout cela et vérifiait en même temps la validité des deux checksums.

Voilà, je pense que le point a été fait et que le goût de trop peu du précédent article aura bel et bien disparu. Sur ce, je vous donne rendez-vous au mois prochain pour la suite de l'étude du clavier. Asta la vista !

Par Loïc Far