

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT

LANGUAGE

Arexx : la voie royale III,
par F.G. (p.2)
Amos connaît la
musique, par F.L. (p.4)

EXECBASE

ExecBase I : du C à
l'assembleur, par D.J.
(p.6)
ExecBase II : le 68000
de A à Z fin, par L.F.
(p.24)

TOOLBOX

ToolBox : reglib et
parnet, par S.S. (p.18)

SUB.way

SUB.way I : la
communication
inter-tâches, par M.
(p.16)
SUB.way II : les menus
sous intuition, par
HDV... (p.26)

DEMO-MAKERS

La 3D faces cachées,
par J.E. (p.10)

CONCOURS

De mal en Pl, par A.C. et
F.M. (p.13)

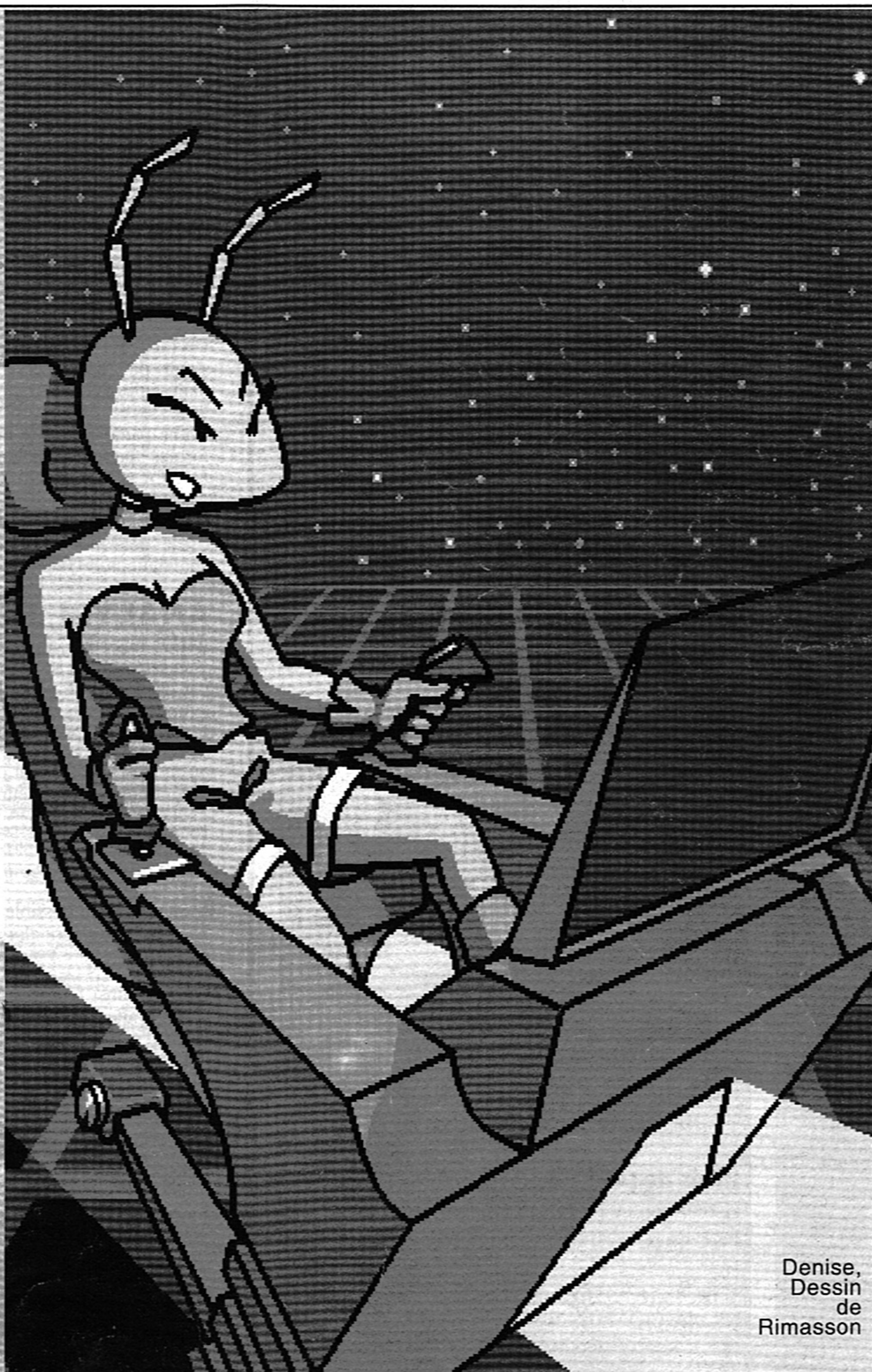
HARDWARE

Les disques chouettes,
par L.F. (p.9)

UTILITAIRE

Sauveur d'écrans IFF,
par M. (p.20)

REQUESTER 30
CATS 32



Denise,
Dessin
de
Rimasson

45 F.

Ce numéro 24 de l'Amiga News-Tech a été tiré à 3000 exemplaires (2895 abonnés enregistrés le 07.06.91).
Ce magazine est une publication Commodore Revue Sarl. 5, rue de la Fidélité - 75010 PARIS.

OURS

L'Amiga NewsTech est une publication de Commodore Revue SARL, société dont nous n'avez pas à connaître le capital, non mais, bande de petits curieux, est-ce que je vous demande le montant de votre compte en banque, moi, hein ? Par contre, son adresse, vous pouvez la connaître : c'est le 5, rue de la Fidélité, 75010 Paris. Le bigophone est le (1) 42 469 290.

Le Môme qui signe les chèques à coups de lance-pierre s'appelle Jean-Yves Primas : c'est le directeur de la publication. Vous pouvez toujours l'appeler au (1) 42 471 216, mais évitez de lui parler football, il est inconsolable depuis la défaite de l'OM en Coupe d'Europe.

Et puisqu'il faut bien un Rédacteur-en-Chef, c'est Yves Huitric qu'on a choisi, alors qu'il était au bord du suicide à cause d'Ayrton Senna qui n'arrête pas de gagner. Pour se changer les idées, il fait l'ANT en PAO sur un Amiga 2000 équipé d'un Flicker Fixer, d'une carte 68030 et de Professional Page 2.0 et espère que les Williams-Renault viendront mettre un peu de piquant dans un championnat jusqu'ici bien triste.

Quand Yves ne fait pas de PAO, il prend un malin plaisir à modifier les chemins de fer que son adjoint, Stéphane Schreiber, passe des heures à figurer avec tendresse et passion, quand il ne va pas s'éclater au squash avec les copains.

Pour écrire les articles, on fait appel à des pigistes qui connaissent bien leur sujet et qui sont tellement passionnés d'Amiga qu'ils oublient de râler quand on les paye en retard. Ce sont : Pascal Amiable, Jérôme Ehenne, Loïc Far, François Gueugnon, Denis Jarril, François Lionet et Max.

Les abonnements sont gérés par la société MCM Europe, que vous pouvez appeler au (1) 43 789 210 si d'aventure vous aviez un quelconque problème de livraison. Si vous préférez faire ça de manière officielle, écrivez au 1, quai Jean-Baptiste Clément, 94140 Alfortville.

Notre gentil flasher se nomme Color Way et habite la même adresse que nous, ce qui, il faut bien l'avouer, nous arrange bien. Son numéro de téléphone est le (1) 40 708 787.

Quant à l'imprimeur, il s'appelle N.J.C., habite Reims, travaille vite et bien et est joignable au (16) 26 075 787.

Bon, on va pas s'étaler, je voulais juste remercier Jean-Loup et Mathieu d'avoir parlé de nous dans un récent numéro de Tilt. Georges de bientôt parler de nous dans un prochain numéro de Micro-News. Yvan, François et Pierre de parler (enfin !) de l'Amiga dans SVM, et Michel de m'avoir complètement ignoré la dernière fois qu'on s'est croisé sur le stand de Joystick Ingrat.

Edito

Numéro 24 : deux ans (déjà !) que l'Amiga NewsTech existe. Que d'évolution en deux années. D'abord un petit cahier au sein de Commodore Revue, puis un véritable "journal dans le journal" pour finalement devenir indépendant... Je sais pas pour vous, mais moi, je suis curieux de voir ce que nous réservent les prochaines années.

Tiens, un mini-sondage : y'a rien à découper, rien à gagner, juste un petit mot à nous glisser dans votre prochain courrier, quand vous aurez le temps.

Une version française des Rom Kernal Manuals, que l'on trouverait facilement, vous intéresserait-elle ? Si oui, soyez nombreux à nous en parler, et on pourra peut-être faire quelque chose.

Voilà. Je laisse la parole à notre service comptabilité, qui a plein de propositions alléchantes à vous communiquer.

Stéphane Schreiber

ABONNEM-ANT

Prix de l'abonnement à l'ANT

- 11 numéros seuls 350,00 F
- 11 numéros + disquette 600,00 F
(Etranger et DOM-TOM : ajouter 85,00 F *)

Anciens numéros

- 1 ancien numéro 40,00 F
- 1 disquette d'ancien numéro 40,00 F
- 1 ancien numéro + disquette 75,00 F

Frais de port : 8,00 F par commande.
(Etranger et DOM-TOM : ajouter 10,00 F ;
frais de port : 10,00 F par commande *)

* Etranger : Du fait des frais bancaires sur les chèques hors-francs, seuls les paiements par mandat international sont acceptés.

LANGAGE : AR

L'instruction PARSE que nous avons entrevue précédemment, mérite une analyse fouillée parce qu'elle est d'une part constamment utilisée et joue un rôle important dans l'acquisition et le traitement des chaînes de caractères (seuls objets connus d'ARexx) et parce que, d'autre part, elle présente des possibilités optionnelles nombreuses et variées.

L'instruction PARSE fournit un mécanisme pour extraire une ou plusieurs sous-chaînes d'une chaîne source et pour les assigner à des variables qui pourront être utilisées par la suite au gré du programmeur. L'instruction PARSE est composée de trois parties distinctes et indépendantes, et peut, de façon formelle, s'écrire :

PARSE source traitement

Le mot réservé PARSE, écrit en majuscules, identifie l'opération. La partie 'source' détermine l'origine de la chaîne de caractères qui va être traitée. La source peut être décrite par des mots réservés différents (ARG, EXTERNAL, NUMERIC, PULL, SOURCE, VAR, et VERSION). Nous allons y revenir. La troisième partie 'traitement' décrit la façon dont la chaîne de caractères acquise depuis la source va être traitée. Il n'y a pas ici de mots réservés mais un formalisme descriptif utilisant des règles que nous précisons plus loin.

Un mot réservé optionnel peut être placé tout juste après le mot PARSE. Il s'agit du mot réservé UPPER, qui transforme la chaîne acquise en majuscules avant le traitement.

Enfin, parce que cette instruction est fréquemment utilisée, il en existe des formes abrégées : PULL pour PARSE UPPER PULL et ARG pour PARSE UPPER ARG. Ces formes sont synonymes, comme l'étaient ECHO et SAY décrites précédemment. Cela simplifie indéniablement l'écriture mais complique la lecture, d'autant plus qu'une fonction ARG() existe aussi. C'est pourquoi nous restons très réservés quant à l'emploi de ces abréviations.

ANALYSE DES OPTIONS SOURCE

ARG : mis pour "argument", c'est l'un des moyens de passer des paramètres à un programme au moment de son invocation depuis la console. Imaginons par exemple que nous disposions d'un programme ARexx qui compte les lettres des mots qu'on lui fournit et dont le nom est CompteLettres. L'invocation à partir de la console CLI/Shell sera :

RX CompteLettres mot

où mot est le mot dont on veut compter les lettres. Les premières lignes du programme CompteLettres seront donc obligatoirement pour l'option ARG :

/* titre */ ; c'est le début d'un programme ARexx

PARSE UPPER ARG 'traitement'

ou, en abrégé :

ARG 'traitement'

Il est tout-à-fait possible de passer ainsi plusieurs paramètres au programme. L'invocation s'écrit alors :

RX CompteLettres mot_1 mot_2... mot_n

On peut ainsi fournir jusqu'à 15 paramètres différents. Il appartiendra à la partie 'traitement' du programme invoqué, de trier les mots. Pour l'instant, il suffit d'écrire les mots avec un espace entre eux et de ne taper la touche retour qu'à la fin. L'espace est un délimiteur, le retour en est un autre. Cette forme est dite implicite, il en existe d'autres que nous indiquerons au passage.

PULL : le fonctionnement de l'option PULL dépend du type de console utilisée. S'il s'agit d'une console de type CLI ou Shell, le fonctionnement est limité. S'il s'agit d'une console de type WShell ou ConMan, le fonctionnement est normal. La différence entre les deux types de consoles est la capacité pour WShell et ConMan de gérer une pile.

- Fonctionnement simplifié avec CLI/Shell

Cette instruction permet d'acquiescer une chaîne de caractères, mais cette fois on peut choisir le moment de l'introduction de la chaîne. L'invocation se fera de façon plus simple : RX CompteLettres. Le programme invoqué se déroule et au moment où la chaîne d'entrée est nécessaire, on trouve une



forme semblable à celle-ci :

```
SAY "Tapez 2 mots séparés par un espace" ; lit l'opérateur  
PARSE UPPER PULL 'traitement' ; ou  
PULL 'traitement' ; forme abrégée.
```

et le programme attend une entrée suivie d'un retour chariot. Là encore, c'est la partie 'traitement' qui triera la chaîne acquise.

- Fonctionnement normal avec console WShell/ConMan

Ces consoles incorporent un raffinement supplémentaire sous forme d'une pile. Une pile est une forme de mémoire atypée dans laquelle on peut mettre à peu près ce que l'on veut. On peut imaginer cette mémoire sous forme d'un tuyau dont une extrémité est dédiée à la sortie par construction. Il existe deux façons d'empiler quelque chose dans ce tuyau. La première consiste à empiler par l'extrémité libre, si bien que par la sortie on verra arriver les éléments mémorisés dans l'ordre de mémorisation (en anglais First IN First Out, premier entré premier sorti). Cette opération d'empilage est réalisée par une commande ARexx *QUEUE*.

La deuxième façon consiste à pousser la pile vers le haut en mémorisant un élément par l'extrémité sortie. Dans ce cas, il s'agit d'une pile Last In First Out, dernier entré premier sorti. La commande ARexx qui réalise cette fonction est *PUSH* (ces deux commandes ne sont efficaces que s'il y a possibilité de pile dans la console).

Pour extraire quelque chose de la pile, il n'y a qu'un chemin : l'extrémité sortie. L'instruction ARexx qui permet d'extraire un élément de la pile est *PULL*. L'élément extrait est ôté de la pile et celle-ci descend d'un cran. Les deux modes de chargement sont simultanément possibles, et la responsabilité de la gestion incombe au programmeur.

Le fonctionnement de l'instruction *PARSE UPPER PULL 'traitement'* devient alors plus clair. Si la console ne gère pas de pile, le programme attend une entrée clavier suivie d'un retour. Si la console gère une pile et que celle-ci contient quelque chose, il y a extraction réelle de la pile.

Deux cas peuvent encore se présenter. Que se passe-t-il à la suite d'un appel par *PULL* si la pile est vide ? La réponse est la même que pour une console sans pile, le programme attend une entrée clavier suivie d'un retour.

Que se passe-t-il si, à la fin du programme, il reste encore quelque chose dans la pile ? La réponse doit être circonstanciée. Cela dépend en effet de qui reprend la main. Mais dans tous les cas, ce qui se présente à la sortie de la pile sera interprété, selon les règles de ce qui prendra la main. Exemple : le programme ARexx se termine et il y a retour à la console. Imaginons qu'il restait la chaîne "dir DH0:" dans la pile. A la fin de l'exécution du programme ARexx, dès le retour à la console, il y aura exécution de l'instruction AmigaDOS "DIR DH0:". Pour généraliser, si ce qui reste dans la pile n'a pas de sens pour celui qui prend la main, il y a dans le meilleur des cas apparition d'un message d'erreur, ou, dans le pire des cas, du Guru.

On peut commencer à envisager l'intérêt d'une console à pile pour échanger des commandes et des paramètres entre des programmes de nature différente.

EXTERNAL : cette option ne fonctionne pas comme décrit dans le manuel. Elle a fait l'objet d'une rectification dans la version 1.15. Cette option attend systématiquement une entrée au clavier, non pas à partir de la console courante (qui gère les deux flux standards stdin et stdout) mais à partir du flux stderr de la console d'erreur, ouverte par appel de la commande TCO (Tracing Console Open), placée dans le répertoire C:. TCO ouvre une fenêtre spéciale dans laquelle s'inscrit la trace de l'exécution du programme en cours et qui peut recevoir des entrées clavier. A partir de cette fenêtre, le fonctionnement de EXTERNAL est identique à celui de PULL, mais l'information ne provient pas de la pile. Ceci est fait pour éviter de perturber la pile qui a le statut de Public et peut donc servir à d'autres programmes. La fermeture de la fenêtre s'effectue par la commande TCC (Tracing Console Close), elle aussi placée dans C:. La fermeture peut toutefois ne pas être immédiate, à cause du double emploi de cette fenêtre.

VAR : mise pour "variable", cette option ne "regarde" plus vers l'extérieur mais fournit à une variable interne du programme, le moyen d'être traitée de la même façon par la partie 'traitement' de l'instruction PARSE. Par exemple, si MaVariable="chaîne de plusieurs groupes de caractères", l'instruction *PARSE VAR MaVariable 'traitement'* effectuera le traitement

décrit sur MaVariable. Là encore et comme partout ailleurs, le mot UPPER peut être placé, s'il y a besoin, juste derrière PARSE.

Il reste maintenant à voir les formes dédiées de l'instruction PARSE. Ces formes sont dites dédiées parce qu'elles fournissent une information identifiée à priori sur l'environnement.

NUMERIC : cette option fournit une information en rapport avec les précisions numériques et les formats des nombres en usage. Elle fournit une chaîne de trois mots :

- *DIGITS*, qui contient le nombre de chiffres de la précision des calculs ;
- *FUZZ*, qui contient le nombre de chiffres à ignorer lors des opérations de comparaison numérique ;
- *FORM*, qui contient soit *SCIENTIFIC* soit *ENGINEERING*. Dans le premier cas, les nombres sont présentés en notation exponentielle, ajustée de façon à ce que la partie non nulle de la mantisse soit comprise entre 0 et 1. Dans le deuxième cas, la forme est aussi exponentielle mais l'exposant varie de trois en trois (par mille) et la mantisse (si non nulle) est comprise entre 0 et 1000. Ces choix peuvent être effectués par ailleurs pour chaque opération réalisée dans le programme, au moyen de l'instruction *NUMERIC* (autre que l'option PARSE ici décrite) et de ses paramètres.

SOURCE : cette option retourne une chaîne qui contient six mots :

- *COMMAND* ou *FONCTION* indique si le programme a été invoqué comme une commande ou une fonction. Une des différences est que la commande retourne un code spécifique nommé RC (pour Return Code), et qui a pour valeur 1 ou 0, selon la réussite d'exécution de la commande (0 pour réussi 1 pour échec), alors que la fonction peut retourner une valeur ou une chaîne ;
- *INDICATEUR BOOLEEN* 0/1 indique que le demandeur attend une chaîne de caractères en retour ou non ;
- *CALLED* indique le nom qui a été utilisé pour invoquer le programme ;
- *RESOLVED* indique le nom qui a été utilisé en fin lors de l'appel réel ;
- *EXT* est l'extension du nom du fichier qui a été utilisée pour la recherche. La valeur par défaut est *REXX*, c'est pourquoi un programme intitulé MonProgramme.REXX peut être invoqué par MonProgramme tout simplement ;
- *HOST* indique l'adresse *REXX* de l'hôte qui a envoyé la commande.

VERSION : cette option retourne une chaîne contenant six mots :

- *AREXX* ;
- *VERSION* contient la version de l'interpréteur courant. Ici par exemple, 1.15.
- *CPU* indique un microprocesseur 68000/10/20/30 selon le cas ;
- *MPU* contient NONE ou 68881, ce qui indique l'éventuelle présence du co-processeur arithmétique ;
- *VIDEO* contient le standard vidéo courant, soit PAL pour les versions européennes ou NTSC pour les autres ;
- *FREQ* contient la fréquence du réseau : 50 Hz ou 60 Hz.

A RETENIR

Selon que l'on utilise une console avec pile ou sans, certaines options de l'instruction *PARSE* fonctionnent différemment.

Les options *SOURCE* et les modes 'traitement' sont indépendants et pourront être combinés.

Dans les options *SOURCE*, il y a des options anonymes qui peuvent être utilisées selon le gré du programmeur et des options dédiées qui fournissent des renseignements sur l'environnement.

Les options *SOURCE* permettent de dialoguer avec l'intérieur du programme (VAR) ou avec l'extérieur de type ARexx (ARG) ou non (PULL)

Nous avons maintenant passé en revue toutes les options *SOURCE* de l'instruction *PARSE*. Nous analyserons prochainement la partie 'traitement'.

par François GUEUGNON

Ca fait maintenant deux mois que je vous en parle, le voici ! Quoi ? Mais le programme permettant de mélanger plusieurs musiques dans une seule banque !

Comme vous avez pu le lire le mois dernier, les banques de musiques sont conçues pour contenir plusieurs musiques exploitant le même jeu d'instruments, ce qui est très avantageux en place mémoire. Hélas, il manquait jusqu'à maintenant un utilitaire permettant de rassembler deux banques en une seule. Le programme que je vous propose fait ce travail : choisissez les noms des deux banques à mélanger, le programme procède à l'exploration des instruments des deux banques à la recherche d'instruments identiques, puis recopie les instruments nécessaires uniquement, suivis des musiques à la suite l'une de l'autre et des patterns, modifie les étiquettes de changement d'instruments et finalement, sauvegarde la nouvelle banque. Son travail terminé, il affiche le nombre d'instruments semblables ainsi que la place mémoire gagnée par rapport à la somme des tailles des deux banques.

Le programme travaille le plus possible sur disquette : il est parfaitement utilisable avec 512 Ko de mémoire. De plus, il est parfaitement possible de fabriquer des banques de plus de deux musiques, en ajoutant simplement les banques l'une après l'autre.

TOP 50

Ce programme permet de mettre en évidence une facilité de l'AMOS passée sous silence jusqu'à maintenant : la possibilité de faire fonctionner une musique sans perturber la précédente...

Exemple et démonstration :

- fusionnez deux des musiques de la disquette AMOS_DATA, disons *Funkey.Abk* et *Music.Abk* ;
- chargez la banque ainsi créée en mode direct ;
- démarrez la première musique avec MUSIC 1 ;
- attendez quelques secondes, puis démarrez la seconde musique avec MUSIC 2 ;
- arrêtez la seconde musique avec MUSIC STOP.

Surprise ! La première musique reprend exactement à l'endroit où elle s'était arrêtée. J'ai conçu ce système pour permettre d'avoir une musique de fond pendant le jeu (Music 1) tout en produisant des bruitages complexes, sous la forme de musiques SoundTracker. Jusqu'à trois musiques peuvent ainsi être démarrées en même temps.

Attention : MUSIC STOP arrête la musique courante, alors que MUSIC OFF arrête toutes les musiques...

ET DAISY ?

La taille du programme ne laisse (heureusement) pas suffisamment de place pour Daisy. Personne se s'en plaindra. D'ailleurs, il faut qu'elle termine le compilateur. Le mois prochain j'essaierai de lui laisser deux ou trois lignes, c'est promis !

```

-----
' MELANGEUR DE MUSIQUES: plusieurs musiques dans une banque...
' (c) 1991 F.Lionet / A.N.T.
-----
Set Buffer 16
Dim I2(256),SUM1(256),SUM2(256)
Global APOKE,SPOKE
Do
Screen Open 0,640,200,2,Hires : Curs Off : Colour 1,$FF4
Centre At(,0)+>>> Mélangeur de musiques <<<
Erase 3
F1$=Fsel$("*.*Abk","", "Nom de la première banque.", "QUIT pour terminer")
If F1$="" : Edit : End If
F2$=Fsel$("*.*Abk","", "Nom de la deuxième banque.", "QUIT pour terminer")
If F2$="" : Edit : End If
Open In 1,F1$
Open In 2,F2$

```

```

' Verification de la validite des banques
A$=Input$(1,20)
If Left$(A$,4)<>"AmBk" : Stop : End If
If Right$(A$,8)<>"Music " : Stop : End If : Rem 3 espaces après Music
A$=Input$(2,20)
If Left$(A$,4)<>"AmBk" : Stop : End If
If Right$(A$,8)<>"Music " : Stop : End If : Rem 3 espaces après Music
' Reservation de la banque de travail
LM1=Lof(1)
LM2=Lof(2)
Reserve As Chip Data 3,LM1+LM2+256
ABANK=Start(3)+20
' Charge les entetes
APOKE=ABANK
HEAD1$=Input$(1,16)
HEAD2$=Input$(2,16)
' Adresses des différentes parties
AINST1=Leek(Varptr(HEAD1$))+20
AINST2=Leek(Varptr(HEAD2$))+20
AMUS1=Leek(Varptr(HEAD1$)+4)+20
AMUS2=Leek(Varptr(HEAD2$)+4)+20
APAT1=Leek(Varptr(HEAD1$)+8)+20
APAT2=Leek(Varptr(HEAD2$)+8)+20
EPAT1=Leek(Varptr(HEAD1$)+12)+20 : If EPAT1=20 : EPAT1=Lof(1) : End If
EPAT2=Leek(Varptr(HEAD2$)+12)+20 : If EPAT2=20 : EPAT2=Lof(2) : End If
' Va faire le travail
Print At(0,10); "Je mélange ";F1$; " avec ";F2$
Print
Print "Nombre total d'instruments:";
Gosub MERGE_SAMPLES
Print NTI;
If NTI<NI1+NI2
Print " - Economie de";NI1+NI2-NTI;" instrument(s)!"
End If
Print
Print "Nombre total de musiques:";
MUMU=SPOKE : Gosub MERGE_MUSICS
Print NMU
Print "Nombre total de patterns:";
Gosub MERGE_PATTERNS : EBANK=APOKE
Print NPAT
APOKE=MUMU : Gosub MERGE_MUSICS
Close
LBANK=EBANK-ABANK+8
E=LM1+LM2-LBANK
If E>0
Print : Print "Economie de";E;" octets!"
End If
Print
Print "Pressez une touche..."
Wait Key
' Sauve la banque...
APOKE=ABANK-20
STROKE["AmBk"]
BDOKE[3] : BDOKE[0] : BDOKE[$80000000+LBANK]
STROKE["Music " ] : Rem 3 espaces après Music
BLOKE[AINST-ABANK]
BLOKE[AMUS-ABANK]
BLOKE[APAT-ABANK]
BLOKE[EBANK-ABANK]
F$=Fsel$("*.*Abk","", "Nom de la banque à sauver...")
If F$<>""
Bsave F$,ABANK-20 To EBANK
End If
Erase 3
Loop
'-----
MERGE_SAMPLES:
' Copie les instruments de la première banque, en stockant les noms
Pof(1)=AINST1 : A$=Input$(1,2) : NI1=Deek(Varptr(A$))
Pof(2)=AINST2 : A$=Input$(2,2) : NI2=Deek(Varptr(A$))
INST1$=Input$(1,NI1*32)
INST2$=Input$(2,NI2*32)
' Pour chaque instrument calcule la somme de controle des 512 1ers samples
For N=0 To NI1-1
A=Varptr(INST1$)+N*32
AI=Leek(A)+AINST1 : LI=Min(512,Deek(A+8)*2)
Pof(1)=AI : A$=Input$(1,LI)
S=0 : SA=Varptr(A$)
For L=0 To LI-1
Add S,Peek(SA+L)
Next
SUM1(N)=S
Next
For N=0 To NI2-1
A=Varptr(INST2$)+N*32
AI=Leek(A)+AINST2 : LI=Min(512,Deek(A+8)*2)
Pof(2)=AI : A$=Input$(2,LI)

```




```

S=0 : SA=Varptr(A$)
For L=0 To LI-1
  Add S, Peek(SA+L)
Next
SUM2(N)=S
Next
' Compte les instruments semblables
NTI=NI1
For N2=0 To NI2-1
  A2=Varptr(INST2$)+N2*32
  L2=Leek(A2+8)
  FLAG=-1
  For N1=0 To NI1-1
    A1=Varptr(INST1$)+N1*32
    L1=Leek(A1+8)
    If L1=L2 and SUM1(N1)=SUM2(N2)
      I2(N2)=N1
      FLAG=0 : Exit
    End If
  Next
  If FLAG
    I2(N2)=NTI : Inc NTI
  End If
Next
' Recopie les instruments
AINST=ABANK+16
APOKE=AINST : BDOKE[NTI]
SVIDE=APOKE+NTI*32
SPOKE=SVIDE+4
For N=0 To NI1-1
  AI=Varptr(INST1$)+N*32
  A1=SPOKE : S1=AINST1+Leek(AI) : L1=Deek(AI+8)*2
  A2=SPOKE+L1 : S2=AINST1+Leek(AI+4) : L2=Deek(AI+10)*2
  If L2<=4
    A2=SVIDE : L2=4
  End If
  BLOKE[A1-AINST] : BLOKE[A2-AINST]
  BDOKE[L1/2] : BDOKE[L2/2]
  For O=0 To 16 Step 4
    BLOKE[Leek(AI+12+O)]
  Next
  SCOPY[1,S1,L1]
  If A2<>SVIDE
    SCOPY[1,S2,L2]
  End If
Next
For N=0 To NI2-1
  If I2(N)>=NI1
    AI=Varptr(INST2$)+N*32
    A1=SPOKE : S1=AINST2+Leek(AI) : L1=Deek(AI+8)*2
    A2=SPOKE+L1 : S2=AINST2+Leek(AI+4) : L2=Deek(AI+10)*2
    If L2<=4
      A2=SVIDE : L2=4
    End If
    BLOKE[A1-AINST] : BLOKE[A2-AINST]
    BDOKE[L1/2] : BDOKE[L2/2]
    For O=0 To 16 Step 4
      BLOKE[Leek(AI+12+O)]
    Next
    SCOPY[2,S1,L1]
    If A2<>SVIDE
      SCOPY[2,S2,L2]
    End If
  End If
Next
INST1$="" : INST2$=""
APOKE=SPOKE
Return
'-----
MERGE_MUSICS:
LMU1=APAT1-AMUS1 : Pof(1)=AMUS1
MU1$=Input$(1,LMU1) : NMU1=Deek(Varptr(MU1$))
LMU2=APAT2-AMUS2 : Pof(2)=AMUS2
MU2$=Input$(2,LMU2) : NMU2=Deek(Varptr(MU2$))
NMU=NMU1+NMU2 : AMUS=APOKE : BDOKE[NMU]
SPOKE=APOKE+NMU*4
' Recopie les musiques de la banque 1
For N=0 To NMU1-1
  BLOKE[SPOKE-AMUS]
  A=Varptr(MU1$) : AM=Leek(A+N*4+2)
  If N<NMU1-1
    FM=Leek(A+N*4+6)
  Else
    FM=LMU1
  End If
  Copy A+AM,A+FM To SPOKE
  Add SPOKE,FM-AM
Next
' Recopie les musiques de la banque 2, en pointant les nouveaux
patterns
For N=0 To NMU2-1
  BLOKE[SPOKE-AMUS]
  A=Varptr(MU2$) : AM=Leek(A+N*4+2)
  If N<NMU2-1
    FM=Leek(A+N*4+6)
  Else

```

```

FM=LMU2
End If
Copy A+AM,A+FM To SPOKE
' Change les numeros de patterns
For P=0 To 3
  AP=SPOKE+Deek(SPOKE+P*2)
  Do
    B=Deek(AP)
    Exit If Bst(15,B)
    Doke AP,B+NPAT1
    Add AP,2
  Loop
Next
Add SPOKE,FM-AM
Next
MU1$="" : MU2$=""
APOKE=SPOKE
Return
'-----
MERGE_PATTERNS:
' Recopie les patterns
Pof(1)=APAT1 : A$=Input$(1,2) : NPAT1=Deek(Varptr(A$))
Pof(2)=APAT2 : A$=Input$(2,2) : NPAT2=Deek(Varptr(A$))
NPAT=NPAT1+NPAT2 : APAT=APOKE : BDOKE[NPAT] : SPOKE=APAT+NPAT*8+2
PAT1$=Input$(1,NPAT1*8) : SPAT1=Pof(1)
PAT2$=Input$(2,NPAT2*8) : SPAT2=Pof(2)
' Recopie betement les patterns de la premiere banque
For N=0 To NPAT1-1
  For P=0 To 3
    BDOKE[Deek(Varptr(PAT1$)+N*8+P*2)+NPAT2*8]
  Next
Next
LPAT1=EPAT1-SPAT1
SCOPY[1,SPAT1,LPAT1]
' Recopie les patterns de la deuxieme de la meme maniere...
For N=0 To NPAT2-1
  For P=0 To 3
    BDOKE[Deek(Varptr(PAT2$)+N*8+P*2)+NPAT1*8+LPAT1]
  Next
Next
LPAT2=EPAT2-SPAT2
SCOPY[2,SPAT2,LPAT2]
' Analyse les patterns de la deuxieme banque pour changer les instruments...
' Etiquette SET_INST=$900
For N=0 To NPAT2-1
  For P=0 To 3
    AP=Deek(APAT+2+NPAT1*8+N*8+P*2)+APAT
    Do
      B=Deek(AP) : Add AP,2
      Exit If B=$8000
      If Bst(15,B)=0
        If Bst(14,B)
          Add AP,2
        End If
      Else
        If (B and $7F00)=$900
          I=B and $FF
          B=$8900+I2(I)
          Doke AP-2,B
        End If
      End If
    Loop
  Next
Next
PAT1$="" : PAT2$=""
APOKE=SPOKE
Return
Procedure STRPOKE[S$]
  For P=1 To Len(S$)
    Poke APOKE,Asc(Mid$(S$,P,1))
    Inc APOKE
  Next
End Proc
Procedure BDOKE[P]
  Doke APOKE,P
  Add APOKE,2
End Proc
Procedure BLOKE[P]
  Loek APOKE,P
  Add APOKE,4
End Proc
Procedure SCOPY[FILE,POS,LONG]
  Pof(FILE)=POS : A=0
  While A<LONG
    L=1024 : If A+L>LONG : L=LONG-A : End If
    A$=Input$(FILE,L)
    Copy Varptr(A$),Varptr(A$)+L To SPOKE+A
    Add A,L
  Wend
  Add SPOKE,LONG
End Proc

```

par François Lionet

*De nombreux lecteurs se - et nous -
posent la même question : ne
disposant pas de compilateur C,
comment profiter des merveilleux
programmes de Frédéric Mazué,
Stéphane Schreiber et autre
Docteur Von Glupa ?*

Quelqu'un d'un tant soit peu mercantile aurait certainement répondu : abonnez-vous à une formule avec disquette... Mais l'ANT, comme à son habitude, va beaucoup plus loin que ça et vous propose carrément de vous prendre par la main, en décortiquant pour vous une (la ?) manière de s'y prendre. Et comme un exemple concret est toujours plus parlant que de longs discours, j'ai choisi d'illustrer mon propos avec un très vieux programme paru dans l'ANT numéro... 6 (Commodore Revue numéro 17) : le "sectorisateur" de la société Loricel.

SECTORISATEUR ?

Un rapide rappel sur le but et l'utilité de ce programme : présenté dans l'ancienne rubrique ViewPort, dans laquelle des éditeurs livraient leurs astuces de programmation, et programmé par Jean-Pierre Vitulli, Sectorise permettait d'écrire sur une suite de secteurs de la disquette placée en DF0:, par le biais du trackdisk.device, un fichier AmigaDOS quelconque (image, musique, programme exécutable...).

PREPARATIFS

De quoi a-t-on besoin pour "transformer" un programme C en son équivalent assembleur ? Tout simplement d'un bon assembleur. J'entends par là "macro-assembleur", c'est-à-dire capable de jongler avec des fichiers includes et, éventuellement, de produire un code linkable plutôt qu'exécutable. Comme vous devriez maintenant en avoir l'habitude, c'est le Devpac II qui a été choisi, mais l'assembleur du Lattice, le DP a68k ou même l'ancêtre Metacomco feront l'affaire. Un peu de documentation ne fera pas de mal non plus et pour une fois, la Bible de l'Amiga (deuxième édition, celle à la couverture bleue) conviendra parfaitement. Mes précédents articles dans l'ANT (utiliser amiga.lib, mélange C/assembleur) ne seront non plus pas de trop. Quelques connaissances, même rudimentaires, du langage C seront également les bienvenues.

PROBLEMES

A quel type de problème se heurte-t-on dans ce cas-là ? Le plus gros concerne les structures du C, intensivement utilisées sur l'Amiga. En effet, alors que le compilateur C sait d'office la taille des champs des structures, l'assembleur ne la connaît absolument pas, et qui plus est, ne se plaindra pas le moins du monde si vous référencez un mot long alors que le champ est de la taille d'un octet. D'où l'utilité de la documentation dont il est question dans le paragraphe précédent.

Un autre problème concerne les paramètres des programmes ; alors que la startup des programmes écrits en C contient une routine qui décompose la ligne de commande du CLI pour remplir le tableau argv et donner sa valeur correcte à argc, rien de tel n'existe en assembleur. De même, le C dispose d'une bibliothèque de fonctions de base (printf, str..., fprintf, etc.) dont l'assembleur ne dispose pas. Une première solution consiste donc à ré-écrire de telles routines en assembleur : on obtient ainsi un code plus compact, répondant parfaitement à nos besoins, mais au prix d'un travail supplémentaire fastidieux. Une seconde solution, par ailleurs celle que j'ai retenue ici, consiste à linker le startup et la bibliothèque C avec le programme assembleur principal. Enfin, une troisième possibilité, celle que j'utilise le plus souvent par fainéantise consiste à se débrouiller comme on

peut pour les obtenir à un moments ou à un autre.

Enfin, dernière difficulté, le choix des variables locales et des variables globales. Là, tout dépend de l'importance de ces variables. Par exemple, un pointeur sur une zone de mémoire allouée devra être conservé, histoire de rendre cette mémoire au système en fin de programme. Par contre, si cette allocation a lieu dans un sous-programme et que la libération correspondante survient à la fin de ce même sous-programme, on aura tout avantage à conserver ce pointer dans un registre An, voire sur la pile dans le pire des cas. Dans le même ordre d'idées, un compteur de boucle for() sera avantageusement placé dans un registre Dn, même s'il a été déclaré global dans le programme C.

POINTEURS ET TABLEAUX

Le C est un langage orienté "pointeurs". Cela signifie qu'il offre tout un tas de facilités pour leur gestion (déclaration, changement de type (casting), arithmétique, etc.). En assembleur, un pointeur est une adresse 32 bits, un point c'est tout. Considérez l'exemple suivant, par ailleurs totalement inutile:

```
/* Cette fonction retourne un pointeur sur le quatrième
élément d'un tableau de mots, suivant le 1er élément égal à -1
*/
UWORD *fonction_quelconque(UWORD *tableau_de_mots)
{
    UWORD *toto; /* toto pointe un tableau de mots (16 bits) */

    toto = tableau_de_mots; /* initialise toto */
    while (*(toto++) != -1); /* saute les mots <> -1 */

    /* toto pointe maintenant le 1er élément APRES celui égal à -1 */
    return(toto + 3); /* saute 3 éléments et fin */
}
```

Transcrire une telle fonction en assembleur ne posera pas de problème particulier, pour peu que l'on se rappelle que toto pointe un tableau de mots de 16 bits, et qu'en tant que tel, le compilateur C l'incrémente de 2 octets lorsqu'il rencontre l'instruction toto++, et de 6 octets lorsqu'il rencontre toto + 3. Le code assembleur de cette même fonction ressemblera donc à quelque chose du genre :

```
fonction_quelconque:
    move.l a0,a1 ; a1 pointe le tableau de mots
    moveq #-1,d0 ; valeur de comparaison
    .loop cmp.w (a1)+,d0 ; compare un MOT et incrémente a1
    bne.s .loop ; boucle tant que différent
    addq.l #3*2,a1 ; saute 3 autres éléments
    move.l a1,d0 ; valeur de retour dans d0
    rts
```

Si la comparaison post-incrémentée se charge pour nous de maintenir a1 aligné sur des mots, l'addition finale devra prendre ce fait en compte.

COMPLEXITE

Comme tout langage évolué qui se respecte, le C permet d'écrire en une seule ligne de programme plusieurs opérations, que le compilateur se chargera de décortiquer en une suite d'instructions assembleur cohérente, suivant la priorité des différents opérateurs utilisés, l'utilisation ou non de parenthèses, etc. Par exemple, la ligne suivante, que l'on rencontre très souvent dans des programmes en C sur l'Amiga :

```
if (GfxBase=(struct GfxBase *)OpenLibrary(GFXNAME,33L)) == NULL)
    CleanExit("Pas de graphics.library");
```

L'ordre des opérations est le suivant :

- initialiser a1 et d0 pour l'appel à OpenLibrary
- appeler OpenLibrary
- sauver le résultat dans la variable GfxBase
- si ce résultat est nul, aller à CleanExit
- sinon, continuer le programme

Ca, c'était un cas facile. Considérez maintenant l'instruction :

Déjà moins évident, non ? La suite des opérations est la suivante :

- prendre le pointeur fl_Volume dans la structure MyLock
- le convertir du BCPL au C
- y puiser le pointeur dl_Name
- le convertir du BCPL au C

TREVE DE BAVARDAGES

```

OPT O+,OW-,L+ ; code linkable et optimisé
; *****
; * Sectorise. *
; * Version assembleur du programme *
; * de JP. Vitulli paru dans CR 17. *
; * Démonstration de "traduction" *
; * C => ASM. Le programme original *
; * est écrit dans les commentaires *
; * commençant par "****" *
; *****
; * Evidemment, c'est pas mal opti- *
; * misable (je me suis contenté de *
; * traduire bêtement le programme) *
; *****

incdir "include:"
include "exec/memory.i"
include "exec/io.i"
include "libraries/dos.i"
include "devices/trackdisk.i"

include "exec/exec_lib.i"
include "libraries/dos_lib.i"

XREF _CreatePort, _DeletePort ; Ce qu'on importe
XREF _CreateStdIO, _DeleteStdIO

XDEF _SysBase, _DOSBase ; Ce qu'on exporte
; *****
BUFSIZE EQU 1000 ; Taille du buffer

DEFSTR MACRO ; cette macro définit une chaîne de
    dc.b .2-.1 ; caractère à la manière BCPL,
    .1 dc.b 1,10 ; en plaçant d'abord sa longueur
    .2 even ; puis la chaîne elle-même. C'est
    ENDM ; très pratique pour Write()
; *****
    rsreset
; D'abord, les mêmes variables que le programme C
args rs.l 2 ; arguments du CLI
tdPort rs.l 1 ; struct MsgPort *tdPort;
tdIO rs.l 1 ; struct IOStdReq *tdReq;
buffer rs.l 1 ; char *buffer;
fichier rs.l 1 ; struct FileHandle *fichier;
numblks rs.w 1 ; nombre de blocs écrits
strtblk rs.w 1 ; bloc de départ (int secteur_depart;)
origine rs.w 1 ; bloc de fin
error rs.w 1 ; int error;

; Puis quelques variables supplémentaires
filenam rs.l 1 ; nom du fichier
filelen rs.l 1 ; sa longueur
stdin rs.l 1 ; nécessaire pour printf()
stdout rs.l 1 ; "" "" ""
bavard rs.b 128 ; buffer pour printf()
VARSIZE rs.w 0

; *****
Start lea VARS(pc),a5 ; a5 pointe les variables

```

7 NUMERO 24 JUIL/AOUT


```

bmi.s      BlocErr
cmpi.w     #1759,d1
bls.s      BlocOk

BlocErr    lea      Err3(pc),a0
           lea      strtblk(a5),a1
           bsr      Printf
           bra      end_prog

***        if ((fichier=Open(argv[1],MODE_OLDFILE))==0)
***        {
***            printf("impossible d'ouvrir le fichier %s",
argv[1]);
***        }
***        end_prog();
BlocOk     move.l   filenam(a5),d1
           move.l   #MODE_OLDFILE,d2
           CALLDOS  Open
           move.l   d0,fichier(a5)
           bne.s    OkFich

           lea      Err4(pc),a0
           lea      filenam(a5),a1
           bsr      Printf
           bra      end_prog

***        tdPort=CreatePort(0,0)
OkFich     pea      $0.w
           pea      $0.w
           jsr      _CreatePort      ; dans amiga.lib
           addq.l   #8,sp
           move.l   d0,tdPort(a5)

***        tdIO=CreateStdIO(tdPort)
           move.l   d0,-(sp)
           jsr      _CreateStdIO     ; dans amiga.lib
           addq.l   #4,sp
           move.l   d0,tdIO(a5)

***        error = OpenDevice(TD_NAME,0,tdIO,0)
***        if (error > 0)
***        {
***            printf("impossible d'ouvrir trackdisk.device");
***            end_prog();
***        }
           lea      tdname(pc),a0
           movea.l   d0,a1
           moveq     #0,d0
           moveq     #0,d1
           CALLEXEC  OpenDevice
           move.w    d0,error(a5)
           bls.s     TDok

           lea      Err5(pc),a0
           bsr      Puts
           bra      end_prog

***        tdIO->io_Command=TD_PROTSTATUS;
***        DoIO(tdIO);
TDok       movea.l   tdIO(a5),a1
           move.w    #TD_PROTSTATUS,IO_COMMAND(a1)
           CALLEXEC  DoIO

***        if (tdIO->io_Actual != 0)
***        {
***            printf("Disque protégé en écriture");
***            end_prog();
***        }
           movea.l   tdIO(a5),a1
           tst.l     IO_ACTUAL(a1)
           beq.s     OkStat

           lea      Err6(pc),a0
           bsr      Puts
           bra      end_prog

***        printf("Placer la disquette destination dans DF0:");
***        printf("puis taper return");
OkStat     lea      Msg1(pc),a0
           bsr      Puts
           lea      Msg2(pc),a0
           bsr      Puts

***        getchar();
           bsr      getchar

***        while(Read(fichier,buffer,512))
WHILE      move.l   fichier(a5),d1
           move.l   buffer(a5),d2
           move.l   #512,d3
           CALLDOS  Read
           tst.l     d0

           beq.s     WEND

***        tdIO->io_Length = 512;
***        tdIO->io_Data = buffer;
***        tdIO->io_Command = CMD_WRITE;
***        tdIO->io_Offset = (long)(secteur_depart++ * 512);
***        DoIO(tdIO);
           movea.l   tdIO(a5),a1
           move.l   #512,IO_LENGTH(a1)
           move.l   buffer(a5),IO_DATA(a1)
           move.w    #CMD_WRITE,IO_COMMAND(a1)
           move.w    strtblk(a5),d0
           ext.l     d0
           addq.l   #1,d0
           moveq     #9,d1
           lsl.l     d1,d0
           CALLEXEC  DoIO

***        if (secteur_depart > 1759)
***        {
***            printf("Dépassement du secteur 1759");
***            printf("Plus de place sur la disquette");
***            break;
***        }
           addq.w    #1,strtblk(a5)
           cmpi.w    #1759,strtblk(a5)
           bls.s     WHILE

           lea      Err7.1(pc),a0
           bsr      Puts
           lea      Err7.2(pc),a0
           bsr      Puts
           bra.s     end_prog

***        printf("Premier secteur : %d", secteur_depart);
***        printf("Dernier secteur : %d", secteur_fin);
WEND       lea      Msg3.1(pc),a0
           lea      origine(a5),a1
           bsr      Printf
           lea      Msg3.2(pc),a0
           lea      strtblk(a5),a1
           bsr      Printf

***        tdIO->io_Command=CMD_UPDATE;
***        DoIO(tdIO);
           movea.l   tdIO(a5),a1
           move.w    #CMD_UPDATE,IO_COMMAND(a1)
           CALLEXEC  DoIO

***        error = tdIO->io_Error;
***        movea.l   tdIO(a5),a1
***        move.b    IO_ERROR(a1),d0
***        ext.w     d0
***        move.w    d0,error(a5)

***        tdIO->io_Length=0;
***        tdIO->io_Command=ETD_MOTOR;
***        DoIO(tdIO);
           clr.l     IO_LENGTH(a1)
           move.w    #ETD_MOTOR,IO_COMMAND(a1)
           CALLEXEC  DoIO

***        if (error > 19)
***        {
***            printf("Erreur trackdisk, code %d", error);
***            cmpi.w    #19,error(a5)
***            bls.s     ErrOk

           lea      Err8(pc),a0
           lea      error(a5),a1
           bsr.s     Printf

***        }
           end_prog();
end_prog:  if (fichier) Close(fichier);
ErrOk     move.l   fichier(a5),d1
           beq.s     .end1
           CALLDOS  Close

***        if (buffer) FreeMem(buffer,BUFSIZE);
.end1     move.l   buffer(a5),d0
           beq.s     .end2
           lea      (BUFSIZE).w,a1
           exg      a1,d0
           CALLEXEC  FreeMem

***        if (tdPort) DeletePort(tdPort);
.end2     move.l   tdPort(a5),d0
           beq.s     .end3
           move.l   d0,-(sp)
           jsr      _DeletePort      ; dans amiga.lib
           .end3

```


HARDWARE : DISKS CHOUETTES

Le Complex Interface Adapter, CIA pour les intimes, n'est pas seulement un super-chronomètre, comme certains auraient pu le penser suite à notre article du mois dernier. C'est également par lui que le système contrôle les quatre lecteurs de disquettes connectables à l'Amiga.

Bien entendu, notre brave petit CIA ne fait pas tout ; il se contente d'assister le DMA disque, en démarrant le(s) moteur(s), en déplaçant la tête de lecture/écriture, en choisissant la face active et en fournissant au système l'état de certaines broches, indispensable au bon fonctionnement du DOS. Ce qui n'est déjà pas si mal. Le DMA disque se charge quant à lui, lorsque le drive est fin prêt, de la lecture ou de l'écriture des données. Il est capable de synchroniser la lecture avec un mot de contrôle, appelé SyncWord, usuellement égal à \$4489, pour garantir l'alignement des données en mémoire. Le type de codage couramment utilisé est le MFM (comme sur IBM-PC ou Atari ST), mais le contrôleur de disquette peut également lire des disquettes codées en GCR (comme sur l'ancêtre Apple II). Il paraît même qu'il serait capable de lire des disquettes 5" 1/4 Commodore 64, mais je n'ai jamais eu l'occasion de vérifier !

ENCORE LES CIAS ?

Ce sont en fait les deux CIAs qui participent à cette tâche ; 4 bits du port A du CIA-A (adresse \$bfe001, faut-il le rappeler ?) servent d'indicateurs, tandis que carrément 8 bits du port B du CIA-B (adresse \$bfd100) contrôlent les drives. Pour simplifier, le CIA-A est utilisé en entrée et le CIA-B en sortie.

Les tableaux donnés le mois dernier ne faisaient que rapidement état de ces bits ; les voici à nouveau, en plus détaillé pour le propos qui nous intéresse.

CIA-A, port A (\$bfe001). Utilisé en entrée.

Bit	Nom	Fonction
5	RDY	Disk Ready. Ce bit passe à 0 quand le moteur du drive sélectionné tourne à sa pleine vitesse.
4	TRACK0	Track Zero Detect. Ce bit passe à 0 quand la tête de lecture/écriture est positionnée sur la piste 0.
3	WPRO	Write Protected. Ce bit passe à 0 si la disquette présente dans le drive est protégée en écriture.
2	CHNG	Disk Change. Ce bit passe à 0 quand on retire la disquette du drive. Il ne re-passe à 1 que lorsque qu'une disquette est à nouveau insérée et qu'une impulsion STEP a été transmise.

CIA-B, port B (\$bfd100). Utilisé en sortie.

Bit	Nom	Fonction
7	MTR	Disk Motor Control. Voir note 1.
6	SEL3	Sélectionne le drive 3 (DF3:) si présent
5	SEL2	Sélectionne le drive 2 (DF2:) si présent
4	SEL1	Sélectionne le drive 1 (DF1:) si présent
3	SEL0	Sélectionne le drive 0 (DF0:) TOUJOURS présent
2	SIDE	Sélectionne la face. Voir note 2.
1	DIR	Sélectionne la direction de déplacement de la tête. Voir note 3.
0	STEP	Envoie une impulsion de déplacement à la tête. Voir note 4.

Tous ces signaux, exceptés SIDE et DIR, sont actifs à l'état bas (0).

Note 1 : ce signal est un peu particulier ; il n'a pas d'action immédiate, mais n'est reçu par le drive que lorsque celui-ci est sélectionné par SEL3-SEL0. De fait, pour démarrer ou arrêter le moteur d'un lecteur donnée, il faut d'abord positionner MTR selon l'opération choisie PUIS sélectionner le drive en question.

Note 2 : SIDE doit être maintenu pendant 100 microsecondes avant une opération d'écriture, et pendant au moins 1,3 milliseconde entre une écriture et un changement de face. 0 sélectionne la face supérieure, 1 la face inférieure.

Note 3 : 0 sélectionne un déplacement vers le centre du disque (la piste 0 est située à l'extérieur du disque). Ce signal doit être mis AVANT l'impulsion

STEP. Deux instructions du micro-processeur sont donc nécessaires.

Note 4 : l'impulsion STEP doit toujours être rapide : inactif, actif, inactif. Les drives Amiga se déplacent d'une piste à l'autre en un maximum de 3 millisecondes. D'autres drives peut être plus rapide, d'autres encore plus lents. Lors d'un changement de direction, un minimum de 18 millisecondes est nécessaire entre les deux impulsions STEP.

ATTENTION : messieurs les demo-makers, soyez sympas, pensez à ceux qui ont un tout petit peu gonflé leur Amiga ! Avec ma 68020 de compétition, les 3/4 des démos en track-loading refusent de tourner sur mon ordinateur, simplement parce que les délais logiciels (décrémenter d'un compteur) s'exécutent plus rapidement ! Vous avez 4 timers à votre disposition dans les CIAs, utilisez-en au moins 1 pour régler les différents délais nécessaires à la mécanique... D'avance, merci.

PETITS EXEMPLES RAPIDES

Pour tous ces exemples, on considère qu'a5 contient l'adresse de base du CIA-B, et a4, celle du CIA-A. Je vous conseille, pour des raisons de lisibilité, d'utiliser les labels définis dans "hardware/cia.i" et de laisser Devpac supprimer pour vous les offsets inutiles (cf. ciapra qui est égal à 0).

```
include "hardware/cia.i"
lea $bfd000,a5 ; a5 = CIA-B
lea $bfe001,a4 ; a4 = CIA-A
```

- démarrage du moteur de DF0:

```
MotorOn:
bset #CIAB_DSKSEL0,ciaprb(a5) ; DF0: off
bclr #CIAB_DSKMOTOR,ciaprb(a5) ; motor on
bclr #CIAB_DSKSEL0,ciaprb(a5) ; DF0: on
rts ; le moteur tourne
```

- arrêt du moteur de DF0:

```
MotorOff:
bset #CIAB_DSKSEL0,ciaprb(a5) ; DF0: off
bset #CIAB_DSKMOTOR,ciaprb(a5) ; motor off
bclr #CIAB_DSKSEL0,ciaprb(a5) ; DF0: on
rts ; le moteur s'arrête
```

- recherche de la piste 0 du drive sélectionné :

```
Track0
.seek bset #CIAB_DSKDIREC,ciaprb(a5) ; DIR=1
btst #CIAB_DSKTRACK0,ciapra(a4)
beq.s .Ok
bsr Step
bsr Delay
bra.s .Seek
.Ok rts
```

- avance la tête de lecture dans la direction indiquée par DIR :

```
Step bclr #CIAB_DSKSTEP,ciaprb(a5)
nop
nop
nop
bset #CIAB_DSKSTEP,ciaprb(a5)
rts
```

- produit un délai suffisamment grand entre deux STEPs. Totalement en désaccord avec ce que j'ai dit plus haut, cette petite routine utilise un délai logiciel, alors qu'il faudrait programmer un Timer pour 3 millisecondes.

```
Delai move.w #3000,d7
dbra d7,* ; décompte et TestDrive
```

- teste si le drive sélectionné est prêt à lire ou à écrire :

```
TestDrive:
btst #CIAB_DSKRDY,ciapra(a4)
bne.s TestDrive
rts
```

La place étant limitée, vous retrouverez toutes ces routines, ainsi que d'autres, dans le programme d'exemple du prochain numéro. Nous en profiterons pour programmer un véritable délai totalement compatible, en utilisant nos connaissances sur les timers du CIA, engrangées le mois dernier. D'ici là, essayez de vous documenter sur le codage MFM, vous verrez, c'est pas triste.

Loïc FAR

DEMOS : LA 3D FACES PLEINES

Après une interruption d'un mois dûe selon la formule consacrée à des problèmes techniques indépendants de notre volonté, voici la suite de notre petit tour au pays de la 3D en assembleur spécial démos.

Comme vous pourrez le constater dans les colonnes et les pages qui suivent, le programme que nous vous proposons est plutôt conséquent (ce sont pas loin de 800 lignes d'assembleur pur et dur que les abonnés sans disquette devront se taper...). Aussi, plutôt que de le passer en deux fois comme n'importe quel journal un tant soit peu civilisé l'aurait fait, nous avons pris le parti de publier dans un premier temps, le listing "nu", et dans un second temps (comprenez "dans le prochain numéro de l'ANT"), l'article et les explications de Jérôme Etienne.

Les plus perspicaces remarqueront que ce programme utilise encore certaines techniques mises en cause par Thomas Landspurg dans la rubrique Requirer du mois dernier : assembleur auto-modifié, effaçage de l'écran par le couple blitter/68000, calcul du point médian d'une droite selon une formule (dichotomie) jugée douteuse... Certes, mais comprenez que l'on ne puisse demander à un programmeur de changer ses habitudes comme ça, du jour au lendemain. De plus, le but de cette rubrique étant avant tout théorique, ceux qui n'aiment pas la mise en pratique peuvent toujours utiliser d'autres méthodes. Enfin, c'est la dernière fois que nous nous permettons des entorses à la philosophie de l'Amiga à laquelle tout le monde devrait adhérer, c'est promis-juré !

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

; autres labels
key      = $bfec01
custom   = $dff000
chip     = $2
clear    = $10000

plane_x  = 352
plane_y  = 285
bpl_depth = 2
bpl_width = plane_x/8
line_width = bpl_width*bpl_depth
modulo    = bpl_width*(bpl_depth-1)
bpl_size  = bpl_width*plane_y
page_size = line_width*plane_y

coplist_size = 40*4

***** macros
allocmem: macro
    move.l #1,d0 ; allocation de $1 octets
    move.l #2,d1 ; type de ram sollicitée
    move.l (_SysBase).w,a6
    jsr _LVOAllocMem(a6)
    move.l d0,\3 ; \3=adr de la mem recupérée
    beq \4 ; si erreur d'allocation => goto \4
endm

*****
freemem: macro
    move.l 1,a1 ; liberer #2 octet a partir
    move.l #2,d0 ; de l'adresse 1
    move.l (_SysBase).w,a6
    jsr _LVOfreeMem(a6) ; memoire a nouveau libre
endm

*****
wait_blt: macro
```

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

***** main prg *****
    bsr main_init

main_loop:
    move.l vposr(a5),d0
    and.l #$1fff,d0
    cmp.l #$f00,d0
    bne.s main_loop

    bsr fill_obj
    bsr compute_sin_cos
    bsr coor_e_to_coor_p
    bsr move_obj
    bsr compute_real_line_color
    bsr clr_obj
    bsr draw_object
    bsr clip_screen

    move.b key,d0 ; teste le clavier
    not d0
    ror.b #1,d0
    cmp.b #$45,d0 ; ESC ?
    beq init_end ;
    bra main_loop ; non => boucle

***** effectue le flipping de page
clip_screen:
    move.l plane_adr(pc),a0
    move.l bpl_log2_adr(pc),plane_adr
    move.l bpl_log_adr(pc),bpl_log2_adr
    move.l a0,bpl_log_adr
    * swap bitplanes dans la copperlist
    move.l coplist_adr(pc),a0
    moveq #bpl_depth-1,d0
    move.l plane_adr(pc),d1
    add.l #line_width,d1
loop_modify_clist:
    addq.l #2,a0
    swap d1
    move.w d1,(a0)+
    swap d1
    addq.l #2,a0
    move.w d1,(a0)+
    add.l #bpl_width,d1
    dbf d0,loop_modify_clist
    rts

***** Gère le clavier pour faire bouger l'objet
move_obj:
    move.b key,d0
    not d0
    ror.b #1,d0
    and.w #$ff,d0
    cmp.w #$3d,d0 ; 7
    bne.s not_key_7
    addq.w #1,inc_ang_a+2
    rts
not_key_7:
    cmp.w #$1d,d0 ; 1
    bne.s not_key_1
    subq.w #1,inc_ang_a+2
    rts
not_key_1:
    cmp.w #$2d,d0 ; 4
    bne.s not_key_4
    clr.w inc_ang_a+2
    rts
not_key_4:
    cmp.w #$3e,d0 ; 8
    bne.s not_key_8
    addq.w #1,inc_ang_b+2
    rts
not_key_8:
    cmp.w #$1e,d0 ; 2
    bne.s not_key_2
    subq.w #1,inc_ang_b+2
    rts
not_key_2:
    cmp.w #$2e,d0 ; 5
    bne.s not_key_5
    clr.w inc_ang_b+2
    rts
not_key_5:
    cmp.w #$3f,d0 ; 9
    bne.s not_key_9
    addq.w #1,inc_ang_c+2
    rts
not_key_9:
    cmp.w #$1f,d0 ; 3
```

```
    bne.s not_key_3
    subq.w #1,inc_ang_c+2
    rts
not_key_3:
    cmp.w #$2f,d0 ; 6
    bne.s not_key_6
    clr.w inc_ang_c+2
    rts
not_key_6:
    cmp.w #$f,d0 ; 0
    bne.s not_key_0
    add.w #10,coor_z_val+2
    rts
not_key_0:
    cmp.w #$3c,d0 ; .
    bne.s not_key_point
    sub.w #10,coor_z_val+2
    rts
not_key_point:
    rts

***** calcul les plans de lignes a tracer
compute_real_line_color:
    lea tab_face(pc),a0
    lea tab_line+4(pc),a1
    lea coor_p(pc),a2
    move.w (a0)+,d7
    subq.w #1,d7
loop_line_color_each_face:
    move.w (a0)+,d0 ;
    movem.w (a2,d0.w),d1/d2 ; > coor du pt1 (x1,y1)
    move.w (a0)+,d0 ;
    movem.w (a2,d0.w),d3/d4 ; > coor du pt2 (x2,y2)
    move.w (a0)+,d0 ;
    movem.w (a2,d0.w),d5/d6 ; > coor du pt3 (x3,y3)
    sub.w d1,d3 ; d3= x2-x1
    sub.w d2,d4 ; d4= y2-y1
    sub.w d1,d5 ; d5= x3-x1
    sub.w d2,d6 ; d6= y3-y1
    muls d3,d6 ; d6= (y3-y1) * (x2-x1)
    muls d4,d5 ; d5= (y2-y1) * (x3-x1)
    sub.l d6,d5 ; d5= (y2-y1)(x3-x1) - (y3-y1)(x2-x1)
    blt.s hidden_face ; si d5<0 then goto
    ; si_face_cachée
    move.w (a0)+,d1
    subq.w #1,d1
    move.w (a0)+,d2
loop_line_color_each_line:
    move.w (a0)+,d3
    eor.w d2,(a1,d3.w)
    dbf d1,loop_line_color_each_line
    dbf d7,loop_line_color_each_face
    rts
hidden_face:
    move.w (a0)+,d0
    add.w d0,d0
    addq.w #2,d0
    lea (a0,d0.w),a0
    dbf d7,loop_line_color_each_face
    rts

***** trace les lignes de l'objet
draw_object:
    lea tab_line(pc),a1
    lea coor_p(pc),a2
    move.w nb_line(pc),count_line
loop_each_face:
loop_each_line:
    move.w (a1)+,d0
    movem.w (a2,d0.w),d0-d1
    move.w (a1)+,d2
    movem.w (a2,d2.w),d2-d3
    tst.w (a1)
    beq.s dont_draw_this_line
    move.w (a1),d7
    clr.w (a1)+
    bsr draw_line
    subq.w #1,count_line
    bne.s loop_each_line
    rts
dont_draw_this_line:
    addq.l #2,a1
    subq.w #1,count_line
    bne.s loop_each_line
    rts

count_face: ds.w 1
count_line: ds.w 1

***** tranforme les coordonnees spaciales
***** en coordonnees planes
coor_e_to_coor_p:
    lea coor_e(pc),a0
    lea coor_p(pc),a1
    move.w (a0)+,d7
```



```

subq.w #1,d7
loop_coor_e_to_coor_p:
    movem.w (a0)+,d0-d2 * d0-d2 = coor e
cos_c_val:
    move.w #0,d4
sin_c_val:
    move.w #0,d5
    move.w d0,d3
    move.w d1,d6
    muls d5,d0
    muls d4,d6
    sub.l d6,d0
    add.l d0,d0
    swap d0
    muls d4,d3
    muls d5,d1
    add.l d3,d1
    add.l d1,d1
    swap d1
cos_b_val:
    move.w #0,d4
sin_b_val:
    move.w #0,d5
    move.w d0,d3
    move.w d2,d6
    muls d5,d0
    muls d4,d6
    sub.l d6,d0
    add.l d0,d0
    swap d0
    muls d3,d4
    muls d5,d2
    add.l d4,d2
    add.l d2,d2
    swap d2
cos_a_val:
    move.w #0,d4
sin_a_val:
    move.w #0,d5
    move.w d1,d3
    move.w d2,d6
    muls d5,d1
    muls d4,d6
    sub.l d6,d1
    add.l d1,d1
    swap d1
    muls d3,d4
    muls d5,d2
    add.l d4,d2
    add.l d2,d2
    swap d2
coor_z_val:
    add.w #3a0,d2
    ext.l d0
    ext.l d1
    moveq #9,d3 * détermine le point de fuite
    asl.l d3,d0
    asl.l d3,d1
    tst.w d2
    beq.s no_zero_division
    divs d2,d0
    divs d2,d1
no_zero_division:
    add.w #plane_x/2,d0
    add.w #plane_y/2,d1
    move.w d0,(a1)+ * > d0-d1= coor p
    move.w d1,(a1)+ */
    dbra d7,loop_coor_e_to_coor_p
    rts

***** trouve les cosinus et sinus
***** compute_sin_cos:
inc_ang_a:
    add.w #4,ang_a+2 * des bons angles
inc_ang_b:
    add.w #4,ang_b+2
inc_ang_c:
    add.w #4,ang_c+2
    lea cos_tab(pc),a0
    move.w #100,d6
    move.w #3fe,d7
ang_c: move.w #0,d3 ; d3= angle * 2
    and.w d7,d3 ;
    move.w (a0,d3.w),d4 ; > d4 = cos(c)
    add.w d6,d3 ;
    and.w d7,d3 ; > d5 = sin(c)
    move.w (a0,d3.w),d5 ;/
    move.w d4,cos_c_val+2 ; put cos in ram
    move.w d5,sin_c_val+2 ; put sin in ram
ang_b: move.w #0,d3 ; d3= angle * 2
    and.w d7,d3 ;
    move.w (a0,d3.w),d4 ; > d4 = cos b
    add.w d6,d3 ;
    and.w d7,d3 ; > d5 = sin b
    move.w (a0,d3.w),d5 ;/

```

```

    move.w d4,cos_b_val+2 ; put cos in ram
    move.w d5,sin_b_val+2 ; put sin in ram
ang_a: move.w #0,d3 ; d3= angle*2
    and.w d7,d3 ;
    move.w (a0,d3.w),d4 ; > d4= cos a
    add.w d6,d3 ;
    and.w d7,d3 ; > d5= sin a
    move.w (a0,d3.w),d5 ;/
    move.w d4,cos_a_val+2 ; put cos in ram
    move.w d5,sin_a_val+2 ; put sin in ram
    rts

***** clipping en Y des lignes
inter_y:
    move.w d0,d4 *
    add.w d2,d4 * > milieu x
    asr.w #1,d4 */
    move.w d1,d5 *
    add.w d3,d5 * > milieu y
    asr.w #1,d5 */
    cmp.w d6,d5
    bne.s inter_y_not_found
    rts
inter_y_not_found:
    blt.s middle_inf_y* if middle_y(d5) is greater
    move.w d4,d2 * > than d6
    move.w d5,d3 * / then modify coord and loop
    bra.s inter_y */
middle_inf_y: *
    move.w d4,d0 * f middle_y(d5) is less
    move.w d5,d1 * / than d6
    bra.s inter_y */ then modify coord and loop

***** clipping en X des lignes
inter_x:
    move.w d0,d4 *
    add.w d2,d4 * > milieu x
    asr.w #1,d4 */
    move.w d1,d5 *
    add.w d3,d5 * > milieu y
    asr.w #1,d5 */
    cmp.w d6,d4
    bne.s inter_x_not_found
    rts
inter_x_not_found:
    blt.s middle_inf_x* if middle_x(d4) is greater
    move.w d4,d2 * > than d6
    move.w d5,d3 * / then modify coord and loop
    bra.s inter_x */
middle_inf_x: *
    move.w d4,d0 * f middle_x(d4) is less
    move.w d5,d1 * / than d6
    bra.s inter_x */ then modify coord and loop

x1: ds.w 2
x2: ds.w 2
save_x: ds.w 1

***** dessine une ligne au blitter de
***** (D0,D1) à (D2-D3)
x_min = 0
x_max = plane_x-1
y_min = 0
y_max = plane_y-2

draw_line:
    cmp.w d3,d1 * clipping de la droite
    beq line_unvisible
    blt.s y1_less_y2
    exg d0,d2
    exg d1,d3
y1_less_y2:
    movem.w d0-d3,x1
    cmp.w #y_max,d3
    ble.s no_inter_y_max
    cmp.w #y_max,d1
    bgt line_unvisible
    move.w #y_max,d6
    bsr inter_y
    movem.w d4-d5,x2
    movem.w x1(pc),d0-d3
no_inter_y_max:
    cmp.w #y_min,d1
    bge.s no_inter_y_min
    cmp.w #y_min,d3
    ble line_unvisible
    moveq #y_min,d6
    bsr inter_y
    movem.w d4-d5,x1
    movem.w x1(pc),d0-d3
no_inter_y_min:
    cmp.w d2,d0
    ble.s x1_less_x2
    exg d0,d2
    exg d1,d3

```

```

x1_less_x2:
    movem.w d0-d3,x1
    cmp.w #x_min,d0
    bge.s no_inter_x_min
    cmp.w #x_min,d2
    ble line_unvisible
    moveq #x_min,d6
    bsr inter_x
    movem.w d4-d5,x1
    movem.w x1(pc),d0-d3
no_inter_x_min:
    cmp.w #x_max,d2
    ble.s no_inter_x_max
    cmp.w #x_max,d0
    bgt line_vert
inter_x_max:
    move.w #x_max,d6
    move.w d3,save_x
    bsr inter_x
    movem.w d4-d5,x2
    move.w d5,d1
    move.w save_x(pc),d3
    bsr line_vert
    movem.w x1(pc),d0-d3
no_inter_x_max:

*** la ligne est visible
line_visible:
    cmp.w d3,d1
    ble.s d3_sup_d1
    exg d0,d2
    exg d1,d3
d3_sup_d1:
    addq.w #1,d1
    sub.w d1,d3
    sub.w d0,d2
    hmi.s xneg
    cmp.w d3,d2
    hmi.s ygtx
    moveq #0011011,d5
    bra.s lineagain
ygtx: exg d2,d3
    moveq #0000111,d5
    bra.s lineagain
xneg: neg.w d2
    cmp.w d3,d2
    hmi.s xnygtx
    moveq #0011111,d5
    bra.s lineagain
xnygtx: exg d2,d3
    moveq #0001111,d5
lineagain:
    add.w d1,d1
    add.w d1,d1
    lea tab_line_adr(pc),a0
    move.l (a0,d1.w),a0
    add.l bpl_log_adr(pc),a0
    ror.l #4,d0
    add.w d0,d0
    add.w d0,a0
    swap d0
    or.w #b5a,d0
    add.w d3,d3
    add.w d3,d3
    add.w d2,d2
    move.w d2,d1
    lsl.w #5,d1
    add.w #42,d1

wait_blt

    moveq #-line_width,d6
    move.w d6,bltcm0(a5)
    move.w d6,bltdm0(a5)
    moveq #-1,d6
    move.l d6,bltafwm(a5)
    move.w d3,bltbm0(a5)
    sub.w d2,d3
    move.w d3,d4
    bpl.s lineover
    or.w #40,d5
lineover:
    sub.w d2,d3
    move.w d3,bltam0(a5)
    moveq #bpl_depth-1,d3
loop_draw_line_each_bpl:
    lsr.w #1,d7
    bcc.s no_line_on_this_bpl
    wait_blt
    move.w d5,bltcon1(a5)
    move.w d0,bltcon0(a5)
    move.w d4,bltapt+2(a5)
    move.w #8000,bltadat(a5)
    move.l a0,bltcpt(a5)
    move.l a0,bltdpt(a5)

```



```

move.w d1,bltsize(a5)
no_line_on_this_bpl:
lea bpl_width(a0),a0
dbf d3,loop_draw_line_each_bpl
line_unvisible:
rts

***** routine speciale
pour tracer des droites verticales
line_vert:
cmp.w d3,d1
beq.s no_line_vert
ble.s d1_inf_d3
exg d1,d3
d1_inf_d3:
sub.w d1,d3
add.w d1,d1
add.w d1,d1
lea tab_line_adr+4(pc),a0
move.l (a0,d1.w),a0
lea bpl_width-2(a0),a0
add.l bpl_log_adr(pc),a0
moveq #line_width-2,d2
cmp.w #3,d7
bne.s line_vert_not_color_3
add.w d3,d3
moveq #bpl_width-2,d2
bra.s draw_line_vert
line_vert_not_color_3:
cmp.w #2,d7
bne.s draw_line_vert
lea bpl_width(a0),a0
draw_line_vert:
lsl.w #6,d3
addq.w #1,d3
wait_blt
moveq #-1,d0
move.l d0,bltawm(a5) * plus rapide que
move.w d2,bltawm(a5) * move.l #-1,bltawm(a5)
move.w d2,bltawm(a5)
clr.w bltcon1(a5)
move.w #35a,bltcon0(a5)
move.w #1,bltadat(a5)
move.l a0,bltapt(a5)
move.l a0,bltdpt(a5)
move.w d3,bltsize(a5)
no_line_vert:
rts

***** efface l'ecran avec
***** le blitter et le 68000
clr_obj:
* premier effacement au 68000
move.l sp,save_sp
movem.l empty_buffer(pc),d0-a6 * met les
* registres
à 0
move.l bpl_log_adr(pc),a7
lea page_size(a7),a7
move.w #32,clr_68000_counter
loop_clr_68000:
rept 10
movem.l d0-a6,-(sp)
endr
subq.w #1,clr_68000_counter
bne.s loop_clr_68000
* effacement au blitter
lea custom,a5
wait_blt
move.l bpl_log_adr(pc),bltdpt(a5)
clr.w bltdmod(a5)
move.l #1000000,bltcon0(a5)
move.w #79*64+bpl_width/2,bltsize(a5)
move.l d0,a5
* second effacement au 68000
move.w #4,clr_68000_counter
loop1_clr_68000:
rept 10
movem.l d0-a6,-(sp)
endr
subq.w #1,clr_68000_counter
bne.s loop1_clr_68000
move.l d0,-(sp)

lea custom,a5
move.l save_sp(pc),sp
rts
save_sp: ds.l 1
clr_68000_counter: ds.l 1
empty_buffer: ds.l 15

***** remplit la totalite des plans
***** en une seule fois
fill_obj:
move.l bpl_log2_adr(pc),a0

```

```

add.l #page_size-2,a0
wait_blt
moveq #-1,d0
move.l d0,bltawm(a5) * plus rapide qu'un
move.l #-1,bltawm(a5)
clr.w bltmod(a5)
clr.w bltdmod(a5)
move.l a0,bltapt(a5)
move.l a0,bltdpt(a5)
move.w #03aa,bltcon0(a5)
move.w #0012,bltcon1(a5)
move.w #plane_y*bpl_depth*64+bpl_width
/2,bltsize(a5)
rts

*****
main_init:
move.l (_SysBase).w,a6
lea custom,a5

allocmem page_size,chip+clear,bpl_log2_adr,fin
allocmem page_size,chip+clear,bpl_log_adr,
freemem_bpl_log2
allocmem page_size,chip+clear,plane_adr,
freemem_bpl_log
allocmem coplist_size,chip,coplist_adr,
freemem_plane
jsr _LVOPforbid(a6)
move.w #03e0,dmacon(a5); all dma off
except disk

* build tab_line_adr
lea tab_line_adr(pc),a0
move.w #plane_y-1,d2
moveq #0,d0
loop_build_tab_line:
move.w d0,d1
mulu #line_width,d1
move.l d1,(a0)+
addq.w #1,d0
dbf d2,loop_build_tab_line

* initialise l'ecran
move.w #(bpl_depth)<<12+$200,bplcon0(a5)
clr.w bplcon1(a5)
clr.w bplcon2(a5)
move.w #modulo,bplmod(a5)
move.w #modulo,bpl2mod(a5)
move.l #1b7136d1,diwstrt(a5)
move.l #3000d8,dffstrt(a5)

* cree la copper list
move.l coplist_adr(pc),a0
moveq #bpl_depth-1,d0
move.l plane_adr(pc),d1
move.w #bplpt,d2
loop_init_clist:
move.w d2,(a0)+
addq.w #2,d2
swap d1
move.w d1,(a0)+
swap d1
move.w d2,(a0)+
addq.w #2,d2
move.w d1,(a0)+
add.l #bpl_width,d1
dbf d0,loop_init_clist
move.l #fffffffe,(a0)+

* init cmap
lea cmap(pc),a0
lea color(a5),a1
moveq #2<<bpl_depth-1,d0
loop_init_cmap:
move.w (a0)+,(a1)+
dbf d0,loop_init_cmap

move.l coplist_adr,copllc(a5)
clr.w copjmp1(a5)

move.w #83c0,dmacon(a5); dma sprite,
copper & bpl
rts

cmap: dc.w 0,$ff,$cc,$aa
tab_line_adr: ds.l plane_y

***** init end
init_end:
move.l (_SysBase).w,a6
lea grname(pc),a1 ; nom de la library ds a1
moveq #0,d0 ; version 0 (the last)
jsr _LVOPenLibrary(a6) ; lib graphique
ouverte
move.l d0,a1 ; adr de graphibase ds a1
move.l 38(a1),copllc(a5); chargement de
; l'adr de l'old

```

```

clr.w copjmp1(a5) ; coplist et lance-
jsr _LVOCloseLibrary(a6) ; lib graphique
; fermée

move.w #83e0,dmacon(a5); canaux dma
; necessaires
jsr _LVOPermit(a6) ; multi tasking autorise

freemem_coplist: freemem coplist_adr(pc),
coplist_size
freemem_plane: freemem plane_adr(pc),
page_size
freemem_bpl_log: freemem bpl_log_adr(pc),
page_size
freemem_bpl_log2: freemem bpl_log2_adr(pc),
page_size

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

grname: dc.b "graphics.library",0
even
plane_adr: ds.l 1
bpl_log_adr: ds.l 1
bpl_log2_adr: ds.l 1
coplist_adr: ds.l 1

*****
cos_tab: include 'cos_tab.s'

nb_point = 8
c = 200

coor_e:
dc.w nb_point* -ceci est la structure concernant
dc.w c,c,c * les coordonnees des points dans
dc.w c,-c,c * l'espace.
dc.w -c,-c,c * le nombre de points doit etre
dc.w -c,c,c * defini par "nb_point" car
dc.w c,c,-c * je m'en sers plus haut
dc.w c,-c,-c * -la 1' valeur de la liste est le
dc.w -c,-c,-c * nombre de points puis la
dc.w -c,c,-c * succession des coordonnees (x,y,z)
coor_p:
ds.l nb_point* ceci est le buffer ou sont
* memorisees les coordonnees apres
* la projection sur l'ecran

nb_line:
dc.w 12 * indique le lignes
tab_line:
dc.w 0,4,0 *-> chaque ligne est representee par
dc.w 4,8,0 * 3 octets. les 2 premiers sont les
dc.w 8,12,0 * numeros des points a relier et le
dc.w 12,0,0 * troisieme est utilise par le
dc.w 16,20,0 * programme pour savoir sur quel
dc.w 20,24,0 * plan il faut la dessiner.
dc.w 24,28,0 * On peut facilement remarquer que
dc.w 28,16,0 * les numeros des points sont tous
dc.w 0,16,0 * multiplies par 4, ceci pour ne pas
dc.w 4,20,0 * avoir a les multi plier au moment
dc.w 8,24,0 * de l'accès à la coordonnée plane
dc.w 12,28,0

tab_face:
dc.w 6 * le nombre de faces de l'objet
dc.w 24,20,4 * = les 3 points d'orientation
dc.w 4 * = le nombre de cotés du polygone
dc.w 1 * = la couleur du polygone
dc.w 54,6,60,30 * = les lignes du polygone
dc.w 0,16,28 */ autre face
dc.w 4 * -on remarque encore que les numeros
dc.w 1 * des points sont multiplies par 4.
dc.w 48,42,66,18 * toujours pour
* les memes raisons.
dc.w 28,24,8 */ autre face
dc.w 4 * -les numeros des lignes sont eux
dc.w 2 * par 6 pour les memes raisons c-a-d
dc.w 60,12,66,36 * pour avoir un accès
* plus rapide
dc.w 20,16,0 */ autre face
dc.w 4
dc.w 2
dc.w 48,24,54,0
dc.w 16,20,24* autre face
dc.w 4
dc.w 3
dc.w 24,30,36,42
dc.w 8,4,0 * autre face
dc.w 4
dc.w 3
dc.w 0,6,12,18

```

par J ETIENNE



CONCOURS : DE MAL EN PI

Notre concours lecteur nous entraîne ce mois-ci au pays des mathématiques. Franck Chatton et Alain Mignon proposent en effet un programme qui sort des sentiers battus : il calcule le nombre PI avec une précision pouvant atteindre 45 000 décimales !

La première tentative de calcul de PI vraiment raisonnable, après les pifométries à base de fraction de l'antiquité (et d'après..., Lambert n'ayant prouvé l'irrationalité de PI qu'en 1761), semble revenir à Leibniz qui utilise le développement en série de l'arctangente : sachant que $\tan(\pi/4)=1$, $\pi/4=1-1/3+1/5-1/7+1/9$. Mais on a trouvé largement mieux depuis. Signalons que Schanks a calculé en 1874 les 707 (ne rigolez pas, il n'avait pas d'ordinateur ! N'empêche qu'elles pourraient quand même être justes...) qui sont gravées dans une salle du palais de la découverte.

UN PEU DE MATHEMATIQUES

Nous n'allons surtout pas utiliser la formule de Leibniz qui est, en matière de convergence, encore plus un veau qu'une Harley, ni même celle d'Euler ($\pi/4=\text{Atan}(1/2)+\text{Atan}(1/3)$) qui se traîne encore passablement, mais celle de Machin : $\pi/4=4\text{Atan}(1/5)-\text{Atan}(1/239)$. Pour calculer l'arctangente, nous allons utiliser son développement en série entière, sachant que $\text{Atan}(x)=\text{SIGMA}(-1)^i/(2i+1)*x^{(2i+1)}$. Enfin pour calculer les arctangentes, nous n'allons pas calculer bêtement la somme telle que je viens de l'écrire, mais nous allons utiliser l'algorithme de Horner, i.e. pour calculer aX^2+bX+c , nous ne calculons pas $a*X*X+b*X+c$, mais $((a)*X+b)*X+c$, ce qui est la manière la plus économique en nombre d'opérations donc en temps, d'évaluer un polynôme.

PETIT HISTORIQUE

Initialement, nous disposions d'un programme de calcul de PI (merci à Laurent Charbonnel de nous avoir fourni le source), qui bien qu'un peu lent (ah, les subtilités de l'euphémisme : plus de 10 heures de calcul pour... 5000 décimales !), ridiculisait déjà notre polytechnicien de prof de maths prétentieux, avec son super logiciel "c'est môââ qui l'ai fait" sur (âmes sensibles s'abstenir) XT en Turbo Pascal et qui pendant ce temps calculait poussivement ses 2000 décimales, et encore... Ce programme laissait une large place à l'optimisation, et la version actuelle n'a plus beaucoup (et même plus du tout) de points communs avec son ancêtre éloigné, hormis bien sûr la fonction.

UN PEU DE PRATIQUE

L'utilisation de la méthode de Horner nous oblige à calculer le rang jusqu'auquel nous devons pousser le développement de la série pour obtenir la précision requise avant de commencer le calcul. Le rang maximal à été calculé en tenant compte des erreurs diverses et vous garantit une précision de 1E-p quand vous demandez p décimales.

"Oh l'autre eh, comment y va les stocker ses décimales, ça passe même pas en double précision !" pensez-vous. Remarque éminemment pertinente au demeurant. Nous allons regrouper les décimales par groupes de 4 dans des mots, eux-mêmes situés dans des zones de mémoire que nous aurons allouées.

Attention, ce n'est pas du DCB. Mais alors pourquoi quatre et pas 5 ? Parce que $65535 < 10^5$. Les heureux possesseurs de 68020 et plus pourront encore améliorer l'algo car les mots longs permettent de stocker 9 décimales, mais le 68000 ne peut malheureusement multiplier et diviser que des mots, d'où la limitation à 45000 décimales de ce programme (il est possible de faire plus, mais il faut déjà environ 9 heures avec ce programme et il serait notablement

ralenti par cette nouvelle modification).

L'arctangente étant impaire et le signe des coefficient alterné, nous allons améliorer la méthode de Horner en n'écrivant pas $aX-bX^3+cX^5+...=X*(a+X*(0+X*(-b+X*(0+X*(c+...))))$ mais $aX-bX^3+cX^5+...=X*(a-X2*(b-X2(c-...)))$, avec $X2=X^2$ (étonnant non ?) calculé une fois pour toutes au début. Ruse subtile : comme nous allons calculer deux arctangentes, pourquoi calculer deux fois les mêmes coefficients (NDLR : Keskidi ?) ? Une partie des coefficients $1/(2i+1)$ étant commune aux deux développements, leur calcul sera mené simultanément à partir d'un certain rang.

Attention, pour les possesseurs d'A500 sans horloge, le chronomètre ne fonctionnera pas, mais il ne provoque cependant pas de plantage.

Enfin, le but initial de notre programme étant la performance pure, nous avons encore gratté un peu de temps de calcul au système à coups de Forbid et de Disable, le tout assaisonné d'une suppression bestiale de DMA d'écran qui piquait quelques cycles au 68000 en haute résolution, avec pour effet secondaire de fournir un blanker pour les distraits et un témoin d'exécution du programme.

```
* Gary Copper / Présente /
* Copyright / GCT 1990 /

* Calcul des décimales de pi /
* PI =16 ATN(1/5) - 4 ATN(1/239) /

*
* opt c+

* Attention: Démarrage à partir du CLI EXCLUSIVEMENT !!!!!!!

incdir "include:"
include "exec/exec_lib.i"
include "libraries/dos_lib.i"

*
* Usage:
*
* PI nb_de_décimales
*
Start move.b #0,-1(a0,d0) ;0 à la fin des paramètres
movem.l d0/a0,-(a7) ;empile a0 & d0

lea DosName,a1 ;On va ouvrir la
moveq #0,d0 ;dos.library
CALLXEXC OpenLibrary
move.l d0,_DOSBase ;Pointeur dosbase dans d0

bne.s DOS ;Si d0=0 il y a erreur
movem.l (a7)+,d0/a0 ;Récupération de a0 & d0
moveq.l #-1,d0 ;Code d'erreur dans d0
bra FIN ;Au revoir

DOS CALLDOS Output ;Recherche la sortie courante
move.l d0,Stand_output ;et la sauvegarde

***** Récupération de la précision
movem.l (a7)+,d0/a0 ;On dépile a0 & d0
subq #1,d0 ;d0=d0-1
beq Erreur1 ;pas de paramètres, donc erreur

moveq #0,d1 ;d1 = 0 (nb de décimales)
moveq #0,d2 ;d2 = 0 (caractère courant)
Loop1 move.b (a0)+,d2 ;Charge 1 caractère dans d2
cmp.b #$20,d2 ;Est-ce un espace ?
beq.s Loop1 ;Oui, on le saute
move.l d1,d3 ;Multiplication par 10 de d1
lsl.l #3,d1 ;= 3 décalages
add.l d3,d1 ;+ 2 additions
sub.b #48,d2 ;Transforme d2 en chiffre
cmp.w #9,d2 ;d2>9 ?
bgt Erreur1 ;Oui, sortie (message 1)
cmp.w #0,d2 ;d2<0
blt Erreur1 ;Oui, Sortie (message 1)
add.l d2,d1 ;d1=d1+d2
Bcl1 cmp.b #0,(a0) ; Fin ?
bne.s Loop1 ; Non, on boucle !

cmp.l #45000,d1 ;d1>Nombre maxi de décimales ?
```



```

bgt Erreur2 ;Cui sortie (Message 2)

***** Réserve mémoire
move.l d1,Nbdec ;Sauve le nb de décimales
addq #8,d1 ;Augmente la précision
lsr.l #1,d1 ;2 chiffres par octet
bclr #0,d1 ;nb pair d'octets demandés
move.l d1,Taille ;Sauve d1
move.l d1,d0 ;d0=d1*3
lsr.l #1,d0
add.l d1,d0
move.l #10001,d1 ;CLEAR+PUBLIC
CALLEXEC AllocMem
move.l d0,Pi ;Sauve l'adresse du nombre Pi
bne.s MemTemp ;Si d0>0 on continue
move.l #14,-(a7) ;Empile le message d'erreur
bra END ;Sort

MemTemp add.l Taille,d0 ;Calcule l'adresse de Temp
move.l d0,Temp ;et la sauve

MemBis add.l Taille,d0 ;Idem pour Bis
move.l d0,Bis

***** Noyau du programme
Calcul move.l #Depart,a0 ;a0 pointe sur l'adresse Depart
bsr Heure ;où on sauve l'heure horloge

move.w #100000000,$dff096 ;Stoppe la DMA bitplane

CALLEXEC Forbid;Stoppe le Multitask
CALLEXEC Disable ;Stoppe les interruptions
bsr Calcul_pi ;Calcul et affichage de PI
CALLEXEC Enable;Remet les interruptions
CALLEXEC Permit;Remet le Multitask

move.w #1000000100000000,$dff096 ;Remet la DMA bitplane

move.l #Arrivee,a0 ;a0 pointe sur Arrivée
bsr Heure ;où on sauve l'heure
bsr Aff_Heure ;On affiche l'heure

***** Restitution de la RAM
Rendmem move.l #0,-(a7) ;Empile 0 (Pas d'erreur)
move.l Pi,a1 ;Adresse de la RAM
move.l Taille,d0 ;Taille de la ram=Taille*3
lsr.l #1,d0
add.l Taille,d0
CALLEXEC FreeMem ;On rend la RAM
bra.s END ;Et on s'en va

***** Erreur1: paramètres incorrects
Erreur1 move.l Stand_output,d1;Sortie standard dans d1
move.l #Msg1,d2 ;Adresse message dans d2
move.l #FMsg1-Msg1,d3 ;Taille message dans d3
CALLDOS Write
move.l #1b,-(a7) ;On empile un code d'erreur
bra.s END ;Et on s'en va

***** Erreur2: trop de décimales
Erreur2 move.l Stand_output,d1;Idem Erreur1
move.l #Msg2,d2
move.l #FMsg2-Msg2,d3
CALLDOS Write
move.l #1b,-(a7)

END move.l (a7)+,d1 ;Dépile le code d'erreur
move.l d1,d0 ;le place dans d0
CALLDOS Exit ;sort en douceur
FIN rts ;Adieu !

***** Zone des données
DosName dc.b 'dos.library',0
even
_DOSBase dc.l 0
Stand_output dc.l 0

Pi dc.l 0
Temp dc.l 0
Bis dc.l 0
Taille dc.l 0
Nbdec dc.l 0

Msg1 dc.b $a,$9b,'3;33;40mUsage ','$9b,'0mPI <Précision>','$a,$a
dc.b $9b,'1;32;40mPI, par ','$9b,'0;33;40mGary copper '
dc.b $9b,'1;32;40md''après un algorithme','$a,'original de '
dc.b $9b,'0;33;40mLaurent Charbonnel','$9b,'1;32;40m'
dc.b '(Hello to you from GCT!),'$a

```

```

dc.b 'modifié et adapté par ','$9b,'0;33;40m'
dc.b 'Franck Chatton & Gary Copper','$9b,'0m','$a,$a,0
even
FMsg1

Msg2 dc.b 'Pas plus de 45 000 décimales (pour le moment ?)'
!','$a,0
FMsg2 even

***** Récupération de l'heure
Heure lea $dc0000,a1 ; Adresse horloge A500
moveq #2,d0 ; 3 paramètres à récupérer
moveq #15,d1 ; Masque
H_bcl move.l (a1)+,d2 ; Récupère 1 paramètre
and.l d1,d2 ; Masque les bits inutiles
add.b #30,d2 ; Transforme en code ASCII
move.b d2,-(a0) ; Range dans la chaîne
move.l (a1)+,d2 ; On passe à la seconde moitié
and.l d1,d2 ; du paramètre
add.b #30,d2 ; que l'on range aussi dans
move.b d2,-(a0) ; la chaîne
subq #1,a0 ; On saute le ':'
dbf d0,H_bcl ; et on boucle
rts ; fini !

***** Affichage de l'heure
Aff_Heure:
move.l Stand_output,d1; Sortie standard dans d1
move.l #H_msg,d2 ; Adresse de la chaîne
move.l #H_msgf-H_msg,d3 ; Taille de la chaîne
CALLDOS Write
rts ; fin !

H_msg dc.b 'Début du calcul: HH:MM:SS'
Depart dc.b $a,'Fin du calcul: HH:MM:SS'
Arrivee dc.b $a
H_msgf even

***** Coeur du programme de calcul
Calcul_pi:
move.l Temp,a0 ; pointeurs Temp, Bis, Pi
move.l Bis,a1 ; et Taille/2 à demeure dans
move.l Pi,a2 ; a0, a1, a2 et d2
move.l Taille,d2 ; d2=Taille/2-1
lsr.l #1,d2
subq #1,d2

bsr.s Arctg ; évalue Arctg 1/5 & Arctg 1/239
bsr Mulpi4 ; Arctg 1/5 * 4
bsr Sub ; Arctg 1/5 - Arctg 1/239 -> PI
bsr Mulpi4 ; PI = 4* PI
bsr AffichePI ; Affichage
rts ; Sortie

***** Calcul des 2 Arctangentes
***** Arctg239
Arctg:
move.w #1000,(a0) ; Temp = 1 (Temp est le nom de
; la variable qui contient 1/2n+1)
move.l Taille,d3 ; On va calculer le rang n jusqu'où

lsr.l #1,d3 ; pousser le développement de l'Arctg
addq #1,d3 ; en rusant pour ne pas calculer de
divu #36223,d3 ; LOG et pouvoir évaluer n avec la
move.l d3,d6 ; formule n = nb dec*7615/36223 sans
mulu #7615,d3 ; dépasser les limites de divu. On
swap d6 ; peut donc utiliser cette routine
mulu #7615,d6 ; pour calculer plus de 45000
divu #36323,d6 ; décimales, la limitation étant
; imputable l'algo utilisé par DIV.
and.l #ffff,d6 ; On efface le reste
add.l d6,d3 ; n est dans d3

move.l d3,d0 ; On calcule maintenant 2*n+1.
lsr.l #1,d0 ; Si 2*n+1 > 65535 alors la
addq #1,d0 ; routine DIV fait n'importe quoi
move.l d0,d1 ; 2n+1 -> d1
move.l a0,a3 ; On divise Temp
bsr Div ; grâce à Div
subq #1,d3 ; n=n-1

***** Arctg5
move.l Taille,d7 ; Même routine que Arctg239
lsr.l #1,d7 ; sauf que n= nb dec*14369/20087
addq #1,d7
divu #20087,d7 ;
move.l d7,d6
mulu #14369,d7
swap d6
mulu #14369,d6
divu #20087,d6

```



```

and.l #ffff,d6 ;On efface le reste
add.l d6,d7 ; n dans d7
move.w #1000,(a2) ; PI = 1
move.l d7,d0
lsl.l #1,d0
addq #1,d0
move.l d0,d1 ;2n+1 dans d1
move.l a2,a3 ;On divise PI
bsr.s Div
subq #1,d7 ;n=n-1

**** Developpement
* Pour gagner du temps, on calcule 1 seule fois 1/2n+1 et on travaille
* sur les 2 arctangentes en même temps.
Dev subq #2,d0 ; d0 = d0 -2 (rappel: d0 = 2n+1)
move.l a2,a3 ; on va diviser PI (càd Arctg 1/5)
moveq #25,d1 ; par 25
bsr.s Div ; c'est fait
bsr.s Div1 ; On calcule 1/2n+1
move.l a2,a4 ; a4 pointe sur PI
bsr Subtemp ; PI = 1/2n+1-PI
cmp.l d3,d7 ; calcul de arctg 239 débuté ?
dble d7,Dev ; non -> Dev
move.l a0,a3 ; Oui, on divise Temp
move.l #57121,d1 ; par 239*239=57121
bsr Div ; Et hop !
move.l a0,a4 ; Temp = 1/2n+1 - Temp
bsr Subtemp ; Et hop !
dbf d7,Dev ; On boucle

* Remarque: une fois que n2=n1, le calcul de Arctg 1/239 commence.
Alors, le
* test cmp ... dbf devient inutile et ralentit le prg. Pour un
* maximum de vitesse, on peut faire s'automodifier le code et le
* supprimer, mais ce n'est pas très élégant et nous ne l'avons
* pas fait, d'autant que le gain est peu élevé ( cmp.l + dbf
* = 20 cycles et 3 NOP = 12 cycles) sauf en rusant beaucoup.

*; Arrivé ici, le calcul est presque terminé. Il ne reste plus qu'à
* diviser PI par 5 et Temp par 239
move.l #239,d1 ; On divise temp
move.l a0,a3 ; Par 239
bsr Div ; Ok!

moveq #5,d1 ; Idem pour PI
move.l a2,a3 ; mais division par 5
bsr.s Div ; Right !
EndArctg:
rts ; Ouf! le plus gros est fait !

***** 1ère routine de division
* Paramètres: a3 pointe sur la variable à diviser
* d1 contient le diviseur
Div move.l d2,d6 ; Taille variable dans d6
moveq #0,d4 ; d4 est utilisé pour le reste
moveq #0,d5 ; et d5 pour le calcul

Divise mulu #10000,d4 ; Reste * 10000
move.w (a3),d5 ; nombre suivant dans d5
add.l d5,d4 ; on ajoute le reste
divu d1,d4 ; et on divise
move.w d4,(a3)+ ; on range le quotient
clr.w d4 ; et on l'efface de d4
swap d4 ; d4 contient maintenant le reste
dbf d6,Divise ; On boucle
rts ; c'est déjà terminé !

***** Seconde routine de division
Div1 move.l d2,d6 ; Taille dans d6
move.l a1,a3 ; On divise Bis

move.l #1000,d4 ; On calcule les 1eres
divu d0,d4 ; décimales de 1/2n+1
move.w d4,(a3)+ ; dans d4 et on les range
clr.w d4 ; puis on garde le reste
swap d4 ; dans d4
Divise2 mulu #10000,d4 ; reste * 10000
divu d0,d4 ; on divise par 2n+1
move.w d4,(a3)+ ; on range le quotient
clr.w d4 ; et on ne garde
swap d4 ; que le reste dans d4
dbf d6,Divise2 ; on boucle
rts ; Au revoir !

***** 1ere routine de soustraction
Sub move.l Taille,d6 ; Taille (en octets) -> d6
move.l a0,a4 ; a4 sur bis
move.l a2,a3 ; a3 sur PI
add.l d6,a3 ; Fin de PI -> a3

```

```

add.l d6,a4 ; Fin de la BIS -> a4
move.l d2,d6 ; Taille (en mots) -> d6
and #fef,CCR ; On efface le bit X de CCR
Moins move.w -(a3),d5 ; On va effectuer la soustraction
move.w -(a4),d4 ; a4-a3, le 68000 gérant le report
subx.w d4,d5 ; de la retenue grâce au bit x !
bpl.s Pas_retenue ; test retenue
Retenue add.w #10000,d5 ; oui on ajoute 10000
Pas_retenue:
move.w d5,(a3) ; on range le résultat
dbf d6,Moins ; et on boucle
rts ; Tchao !

***** Seconde routine de soustraction
Subtemp move.l Taille,d6 ; Taille dans d6
move.l a1,a3 ; Temp dans a3
add.l d6,a3 ; On pointe sur la fin
add.l d6,a4 ; des 2 variables
move.l d2,d6 ; Taille (mots) dans d6
and #fef,CCR ; On efface x
Moins2 move.w -(a3),d5 ; Idem Sub
move.w -(a4),d4
subx.w d4,d5
bpl.s Pas_retenue2
Retenue2:
add.w #10000,d5
Pas_retenue2:
move.w d5,(a4)
dbf d6,Moins2
rts ; Adios !

***** routine de multiplication
Mulpi4 move.l Taille,d6 ; Taille -> d6
move.l a2,a3 ; PI pointé par a3
add.l d6,a3 ; Fin de PI pointée par a3
move.l d2,d6 ; Taille -> d6
moveq #0,d5 ; d5 initialisé
Fois moveq #0,d4 ; d4=0
move.w -(a3),d4 ; mot suivant -> d4
lsl.w #2,d4 ; d4=d4*4
add.w d5,d4 ; d4=d4+retenue
divu #10000,d4 ; on calcule le reste et
move.w d4,d5 ; la retenue qui est placée dans d5
swap d4 ; le reste est lui
move.w d4,(a3) ; rangé dans PI
dbf d6,Fois ; On boucle
rts ; end !

***** Routine d'affichage
* Remarque: on se set de Bis et de Temp comme tampon d'affichage !
AffichePI
move.l Nbdec,d3 ; Nb dec -> d3
addq.l #3,d3 ; d3=d3+3 pour le return de fin & 3.
move.l d2,d7 ; Taille (mots) -d7
move.l _DOSBase,a6 ; a5 pointe sur dos.library
move.l a2,a4 ; a4 sur PI
move.l a0,a3 ; et a3 sur Bis
addq.l #1,a3 ; On laisse la place pour '.'
Aff move.w (a4)+,d6 ; mot à convertir
lea Tabrang,a1 ; table de conversion
moveq #3,d1 ; 4 chiffres
Rang move.w (a1)+,d4 ; valeur suivante de la table
moveq #-30,d5 ; -code ascii de 0 dans d5
Soust sub.w d4,d6 ; tant que d4>0, d4=d4-d6
dbmi d5,Soust ; et d5=d5-1
neg.b d5 ; code ascii du chiffre dans d5
move.b d5,(a3)+ ; on le range dans le tampon
add.w d4,d6 ; et on remplace d6
dbf d1,Rang ; chiffre suivant
dbf d7,Aff ; mot suivant
move.l a0,a3 ; a3 pointe sur Bis
move.b 1(a3),(a3) ; Déplace le 1er chiffre
move.b #'.' ,1(a3) ; Et ajoute '.' à la suite
add.l Nbdec,a3 ; a3 pointe sur la fin de chaîne
move.b #$0a,2(a3) ; Return
move.l Stand_output,d1 ; sortie courante -> d1
move.l a0,d2 ; Adresse chaîne = Bis
jsr _LVOWrite(a6) ; On affiche
rts ; C'est la FIN, sniff !

Tabrang:
dc.w 1000,100,10,1 ; Table de conversion

```

Par F CHATTON et A MIGNON

Il est des cas fréquents, sur une machine multitâche, où plusieurs programmes ont besoin de communiquer entre eux : le domaine public regorge d'exemples d'utilitaires initialisant des tâches de fond, auxquelles le programme principal pourra plus tard transmettre diverses informations suivant ses besoins du moment.

Mais l'exemple le plus évident est encore Intuition et ses fameux flags IDCMP, dont notre docteur national vous entretient dès qu'il en a l'occasion, avec tout le brio qu'on lui connaît : chaque fois qu'Intuition a besoin de signaler à une application un évènement particulier, elle le fait par le biais d'IntuiMessages, qui contiennent toutes les informations nécessaires (type de l'évènement, position de la souris au moment où il est survenu, état du clavier et plus particulièrement des touches spéciales, heure système...).

Ce mécanisme de communication inter-tâches fait partie intégrante d'Exec, à travers ses signaux et ses ports de messages. Le principe est à la fois fort simple et ingénieux : les tâches communiquent entre elles au moyen de ports de messages (MsgPort), qui sont autant de boîtes à lettres grâce auxquelles elles peuvent s'envoyer des petits mots d'amour (Messages). Et comme les tâches sont des gens très polis, elles se font un devoir de répondre à chaque message, juste histoire de signaler à l'expéditeur qu'il a bien été reçu. Le seul rôle d'Exec dans tout ça est de véhiculer les messages d'un port à l'autre. Pour continuer l'analogie ci-dessus, c'est Monsieur P&T en personne.

MORDS-MOI LE NOEUD

L'une des particularités du système d'exploitation de l'Amiga est de ne disposer d'aucune adresse fixe (hormis l'adresse 4, qui contient un pointeur sur la structure ExecBase). Cela signifie plus précisément, qu'un programme ne peut jamais savoir où se trouve en RAM une structure particulière sans d'abord la rechercher explicitement. C'est pourquoi Exec offre un mécanisme de localisation basé sur le principe des listes chaînées : chaque membre de la liste contient un pointeur sur son prédécesseur et sur son successeur. Ainsi, en parcourant la liste, on peut facilement retrouver un membre particulier. Ce principe a un autre avantage, la rapidité : on peut facilement ajouter ou supprimer un membre de la liste simplement en modifiant quelques pointeurs, alors que dans le cas d'une liste séquentielle (les membres sont à la suite les uns des autres), on serait obligé de procéder à un déplacement de bloc mémoire coûteux en temps machine, et donc en performance du système. Dans la terminologie de l'Amiga, un membre d'une liste est appelé un noeud (Node). Exec tout entier repose sur le principe des noeuds et des listes. Le lecteur averti - celui qui en vaut deux - pourra toujours sauter le paragraphe suivant s'il en connaît déjà le principe. Pour le novice, précisons simplement que les listes sont largement utilisées par les ports de messages et qu'il convient donc d'en saisir parfaitement le fonctionnement avant que d'aller plus loin.

ON ENCHAINE

La sagesse populaire dit qu'il faut un début à tout ; une liste chaînée n'échappe pas à cette règle. Comme on y est maintenant habitué avec l'Amiga, le début d'une liste est représenté par une structure C, que l'on trouve dans les includes "exec/lists.h" et "exec/lists.i" :

```
struct List
{
    struct Node *lh_Head;
    struct Node *lh_Tail;
    struct Node *lh_TailPred;
    UBYTE lh_Type;
    UBYTE l_pad;
};
```

lh_Head est un pointeur sur le premier noeud de la liste, **lh_Tail** est toujours NULL, et **lh_TailPred** pointe sur le dernier noeud de la liste. Une liste est dite vide lorsqu'elle ne contient aucun noeud, c'est-à-dire lorsque le champ

lh_TailPred pointe sur une valeur nulle et **lh_Head**, sur la liste elle-même. **lh_Type** contient un code indiquant le type de la liste, et **l_pad** fait en sorte que la structure contienne un nombre pair d'octets.

NOTE : le fichier "exec/lists.h" tel qu'il est fourni avec le compilateur SAS/Lattice C 5.10 fait état d'un champ "l_pad" alors que la logique voudrait que ce soit "lh_pad". Je préfère quant à moi m'en tenir à la version Lattice, puisque c'est elle qui sera utilisée en cas de référence à ce champ dans un programme.

Un noeud est également représenté par une structure, que l'on trouve cette fois-ci dans "exec/nodes.h" et "exec/nodes.i" :

```
struct Node
{
    struct Node *ln_Succ;
    struct Node *ln_Pred;
    UBYTE ln_Type;
    BYTE ln_Pri;
    char *ln_Name;
};
```

ln_Succ pointe sur le prochain noeud de la liste et **ln_Pred**, sur le précédent. Le dernier noeud d'une liste est caractérisé par un champ **ln_Succ** initialisé à 0 (NULL). **ln_Type** indique le type du noeud (évidemment, une liste d'un type donné ne peut contenir que des noeuds du même type) et **ln_Pri** sa priorité au sein de la liste. Enfin, **ln_Name** pointe sur une chaîne de caractères pouvant représenter le nom du noeud, défini au gré du programmeur, et qui pourra être utilisé pour retrouver un noeud particulier par son nom.

Voilà. La dernière chose que l'on puisse dire sur les listes, est que la bibliothèque exec.library fournit toute une rimbambelle de fonctions pour leur gestion : initialisation, insertion, suppression et recherche de noeuds sont au menu du jour.

SIGNAL SANS FLUOR

Le signal est le niveau le plus simple et le plus performant à la fois de la communication inter-tâches. Leur utilisation la plus simple consiste simplement à synchroniser deux tâches entre elles (NDLR : vous en avez un exemple flagrant dans l'utilitaire du mois, le ScreenSaver), mais ils sont également utilisés, justement, dans les ports de messages.

Chaque tâche peut disposer d'un maximum de 32 signaux, qui correspondent à autant de bits du champ **tc_SigWait** de sa structure Task. Sur ces 32 signaux, 16 sont réservés par le système pour des évènements particuliers, par exemple les 4 "break" possibles (CTRL/C, CTRL/D, CTRL/E et CTRL/F). Les 16 autres sont à la disposition de la tâche, qui peut leur assigner les évènements qu'elle désire. Ceci est très important : le signal numéro 13 peut signifier pour une certaine tâche qu'un temps donné vient de s'écouler, alors que pour la tâche voisine, il signifiera peut-être qu'un message vient d'arriver sur son port.

Une tâche alloue un signal avec la fonction **AllocSignal()**. Il faut en effet savoir que chaque fois qu'une tâche crée, directement ou non, un port de message (par exemple, en ouvrant une fenêtre avec des flags IDCMP), un nouveau signal lui est alloué par le système. En d'autres termes, si elle désire se réserver un signal particulier, elle doit le faire de manière à ce que ce signal ne soit pas ré-utilisé à son insus, d'où l'utilité d'**AllocSignal()**. Cette fonction demande en paramètre le numéro du signal à allouer, où -1 pour le prochain libre. En pratique, on transmet toujours -1, laissant ainsi à Exec le soin de choisir le bon signal. En retour, **AllocSignal()** transmet le numéro du signal alloué, ou -1 si aucun n'était disponible. En C, cela donne :

```
BYTE signal;

signal = AllocSignal(-1L);
if (signal == -1)
    printf("Plus de signal libre");
else
    printf("Signal numéro %d alloué", signal);
```

Lorsqu'un signal n'est plus nécessaire, il doit être libéré au moyen de la fonction **FreeSignal()** :

```
FreeSignal(signal);
```




Avant que de pouvoir être utilisé, un signal doit être converti en masque, ceci afin d'accélérer le traitement de la demande par Exec. Ceci se fait de manière très simple, par l'instruction suivante :

```
ULONG signalmask;
signalmask = 1L << signal;
```

Une tâche peut alors attendre qu'un ou plusieurs signaux qu'elle a alloués surviennent (encore une fois, les signaux peuvent lui être alloués indirectement par le système) au moyen de la fonction **Wait()**, qui demande en paramètre le masque du (des) signal (aux) à attendre. Par exemple, imaginons un programme ayant ouvert une fenêtre avec un menu (disposant donc de flags IDCMP) ainsi que l'input.device, et qui peut se faire signaler par un serveur d'interruption qu'il a mis en place, qu'un temps donné vient de s'écouler. En C, cela ressemblera à quelque chose du genre :

```
/* la fenêtre et l'input.device sont déjà ouverts ! */
BYTE irqSignal;
ULONG windowMask, inputMask, irqMask;
ULONG signal_recu;

windowMask = 1L << window->UserPort->mp_SigBit;
inputMask = 1L << inputPort->mp_SigBit;
if ( (irqSignal = AllocSignal(-1L)) == -1 )
{ printf("Plus de signal pour l'interruption");
  exit(-1);
}
else
  irqMask = 1L << irqSignal;

/* attente d'un ou plusieurs des 3 signaux possibles */
signal_recu = Wait(windowMask | inputMask | irqMask);
if (signal_recu & windowMask)
  printf("Une option de menu a été choisie");

if (signal_recu & inputMask)
  printf("Une touche a été appuyée");

if (signal_recu & irqMask)
  printf("Le temps s'est écoulé");
```

Rappelez-vous qu'une tâche donnée peut attendre jusqu'à 16 signaux simultanément.

Enfin, pour envoyer un signal, une tâche doit d'abord savoir à qui elle va l'adresser, et bien sûr, quel signal envoyer (n'oubliez pas que chaque tâche dispose de ses propres signaux). C'est pourquoi deux tâches qui désirent communiquer ne peuvent le faire que si les deux sont parfaitement au courant de ce qui va se passer. Quand la tâche émettrice signale la tâche réceptrice, elle utilise la fonction **Signal()**, qui demande en paramètre l'adresse de la tâche à signaler, et le masque du signal à lui envoyer :

```
Signal(task, signalmask);
```

LES MARINS QUI PISSENT

Dans le port de message, on retrouve le signal en tant que base de la communication. Un message est composé de deux parties distinctes : une utilisée par Exec pour envoyer le message à bon port, l'autre étant le corps même du message. Cette partie est entièrement et librement définissable par le programmeur, qui peut donc en faire ce qu'il veut, dans la limite toutefois de... 64 Ko !

Les messages sont envoyés sur des ports de messages (MsgPort), selon un système de file d'attente : le premier message arrivé sur un port en est le premier extrait (en anglais, first in first out, ou encore FIFO). Il n'y a aucune restriction autre que la quantité de mémoire disponible, quant au nombre de ports dont une tâche peut disposer et au nombre de messages qui peuvent arriver sur chaque port.

Pour des raisons de performance, Exec ne procède pas à la copie des messages dans une mémoire tampon. En clair, cela signifie que si une tâche A envoie un message à une tâche B, les deux tâches vont se partager - provisoirement - l'espace mémoire occupé par ce message. La tâche A doit donc faire attention à ne pas modifier le contenu du message avant que la tâche B en ait terminé avec celui-ci, ce qu'elle indique en renvoyant le même message à la tâche A. Exec offre bien entendu toute une panoplie de

fonctions dédiées à l'envoi, l'attente, la réception et la réponse aux messages.

SALUT LES PORTS

Les ports de messages sont donc en quelque sorte des points de rendez-vous où tous les messages destinés à une tâche particulière arrivent. En fait et pour être plus précis, tous les messages destinés à une tâche particulière et envoyés par une autre tâche non moins particulière. En d'autres termes, si une tâche désire communiquer avec - par exemple - Intuition, le console.device et le trackdisk.device, elle devra posséder 3 ports de messages, un par tâche communicante.

Encore une fois, c'est une structure C qui définit l'aspect d'un port de messages. On trouve sa définition dans les includes "exec/ports.h" et "exec/ports.i" :

```
struct MsgPort
{ struct Node mp_Node;
  UBYTE mp_Flags;
  UBYTE mp_SigBit;
  struct Task *mp_SigTask;
  struct List mp_MsgList;
};
```

Le premier champ de cette structure, **mp_Node**, est un noeud standard, qui peut être utilisé pour donner un nom à ce port (par l'intermédiaire du champ *ln_Name* de la structure Node). Ceci peut se révéler très pratique lorsqu'une tâche désire savoir si une autre tâche est déjà présente en mémoire : grâce à la fonction *FindPort(nom)*, elle peut très aisément s'en rendre compte (NLDR : encore une fois, jetez un oeil sur l'utilitaire ScreenSaver de ce mois-ci).

Le champ **mp_Flags** indique à Exec ce qu'il doit faire lorsqu'un message arrive sur ce port. Avec **PA_SIGNAL**, Exec signale - par la fonction *Signal()* - à la tâche désignée par *mp_SigTask*, l'arrivée du message ; avec **PA_SOFTINT**, une interruption est déclenchée ; enfin, avec **PA_IGNORE**, rien ne se passe, le message est simplement inséré dans la file d'attente.

mp_SigBit contient le numéro du signal alloué à ce port. **mp_SigTask** est un pointeur sur la tâche devant être signalée lorsqu'un message arrive sur ce port (si *mp_Flags == PA_SIGNAL*), et **mp_MsgList** est le début de la liste des messages déjà arrivés sur ce port.

Pour se créer un port de messages, une tâche doit donc initialiser une structure *MsgPort* en remplissant correctement le noeud et en initialisant la liste. Si ce port doit être rendu public, c'est-à-dire si d'autres tâches peuvent y accéder, elle devra de plus appeler la fonction **AddPort()** d'exec.library. Heureusement, la fonction **CreatePort()** contenue dans amiga.lib, simplifie tout ce processus de création. Elle demande pour paramètre le nom du port à créer (ou NULL s'il ne doit pas être rendu public) et sa priorité :

```
monPort = CreatePort("Max.Port", NULL);
```

Quand une tâche n'a plus besoin d'un port, elle doit le supprimer de la liste des ports publics le cas échéant - c'est-à-dire s'il y avait été inséré avec **AddPort()** - au moyen de la fonction **RemPort()**. La fonction d'amiga.lib **DeletePort()** s'occupe de tout ça ; il suffit de lui transmettre un pointeur sur le port à supprimer :

```
DeletePort(monPort);
```

Tous les messages restant dans la file d'attente de ce port devront en avoir été enlevés, ce qui s'obtient en y répondant jusqu'à ce la file soit vide.

Mais je parle, je parle, et je constate que la place qui m'était impartie est bien remplie. Nous verrons donc le moins prochain les messages et leurs différents types (depuis les messages standards d'Exec et Intuition jusqu'aux messages personnels qu'une tâche peut envoyer) et mettrons enfin en pratique tout ce qui vient d'être dit. Asta la vista !

Max adore les messages thaïlandais 

TOOLBOX : REQLIB & PARNET

Deux utilitaires du domaine public pour le prix d'un : c'est ce que vous propose notre Toolbox de ce mois. Et qui plus est, ces deux utilitaires n'ont aucun point commun entre eux !

Le premier, ReqLib, est une bibliothèque Amiga standard, totalement ré-entrante, destinée à faciliter - et à uniformiser - la création de requesters. Elle dispose notamment d'un puissant FileRequester pouvant être utilisé en tant que FontRequester, d'un ColorRequester, et de StringRequesters supportant des opérations de formatage du texte, façon printf(). ReqLib s'utilise exactement comme une bibliothèque système (dos, intuition...) : il faut d'abord l'ouvrir, mémoriser son adresse de base, puis la fermer lorsqu'on n'en a plus besoin. Elle est évidemment utilisable aussi bien depuis l'assembleur que depuis le C (Aztec ou Lattice). En fait, n'importe quel langage, même l'AmigaBasic, convient !

Exemple d'ouverture en C :

```
#include "libraries/req.h"

struct ReqBase *ReqBase = NULL;

main()
{
    /* Ouverture de la req.library */
    ReqBase=(struct ReqBase *)OpenLibrary("req.library",0L);
    if (!ReqBase)
        CleanExit("Pas de req.library !");
    ...
    CloseLibrary(ReqBase);
}
```

Le même, en assembleur :

```
include "libraries/req.i"

; Ouverture de la req.library
lea reqname(pc),a1
moveq #0,d0
CALLEXEC OpenLibrary
move.l d0,_ReqBase
beq NoReq ; ouverture ratée
...
movea.l _ReqBase(pc),a1
CALLEXEC CloseLibrary
```

Partant de là, les fonctions listées ci-dessous (et classées par thème), sont disponibles.

- Affichage d'un requester de type AutoRequest, mais plus évolué :
SimpleRequest() Requester à 1 bouton
TwoGadRequest() Requester à 2 boutons
TextRequest() Requester à 3 boutons
- Affichage d'un requester de chaîne :
GetString Requester de chaîne de caractères
GetLong Requester de nombre 32 bits
- Affichage d'un sélecteur de fichier(s) ou de fonte(s) :
FileRequester()
PurgeFiles()
- Affiche un requester de palette :
ColorRequester()
- Création dynamique de gadgets (pour programmes ré-entrants) :
MakeGadget() Alloue un gadget booléen
MakeString() Alloue un string gadget
MakeProp() Alloue un gadget proportionnel
MakeButton() Alloue un gadget booléen
MakeScrollBar() Alloue un gadget proportionnel
- Gestion des gadgets proportionnels :
SetSize()

ReadLocation()
SetLocation()

- Gestion des gadgets liés à une fenêtre :

LinkGadget()
LinkStringGadget()
LinkPropGadget()

- Routines diverses :

DrawBox Dessine un rectangle d'un coup
GetFontWidthAndHeight Renvoie la taille de la fonte
RealTimeScroll Scrolling suivant un ascenseur
RawKeyToAscii Conversion touches RAW en ASCII
Center Centrage d'une fenêtre
Format Formate de texte à la sprintf()

Il serait vain de décrire ici toutes ces fonctions, une à une. L'archive de distribution contient un exemple d'utilisation de la req.library, ShowOffReq, démontrant une grande partie de ces fonctions (en fait, toutes celles qui ont un résultat visible à l'écran).

Une bibliothèque étant faite pour les programmeurs, divers fichiers incluses sont fournis, tant pour les adeptes du C que pour ceux de l'assembleur. Ce sont req.h, reqbase.h et reqbase.i (étonnant, non ?) qu'il conviendra de placer dans votre répertoire Include/Libraries/. Un fichier reqproto.h est également fourni, pour utilisation avec le Lattice. Concernant l'assembleur, reqbase.i est écrit comme les autres includes standards de Commodore, à savoir en utilisant les macros STRUCTURE, LABEL et autres LONG, et non avec la pseudo-instruction "rs" du Devpac (bien que Devpac comprenne également cette forme, l'assemblage est plus rapide avec "rs"). Par contre, reqbase.i contient également les définitions des Library Vector Offsets (_LVOxxx), alors qu'il eût été plus logique de les mettre à part, dans req_lib.i par exemple (ce pour rester conforme avec la structure des autres includes). Enfin, un fichier nommé macros.i contient devinez quoi ? Quelques macros destinées à vous faciliter la vie.

Deux autres fichiers objet, areqglue.o et lreqglue.o sont plus particulièrement destinés à être liés avec votre code, C ou assembleur, suivant l'utilisation que vous désirez faire de la req.library. Il s'agit de ce que l'on nomme des routines "glue", c'est-à-dire des bouts de routines destinés à faciliter l'interfaçage avec d'autres routines, plus complexes. En l'occurrence, on y trouve les routines récupérant les paramètres depuis la pile dans les registres adéquats puis appelant les fonctions de la bibliothèque (ceci plus particulièrement pour le langage C) et la définition des fonctions SimpleRequest() et TwoGadRequest(), qui ne sont que des simplifications de l'appel à TextRequest, qui elle fait partie intégrante de la bibliothèque (heu... Ca va, vous suivez encore ?). areqglue.o est destiné au compilateur ou à l'assembleur Aztec, tandis que lreqglue.o est destiné au Lattice.

Un petit programme supplémentaire, nommé CustomizeFile et dont le source est fourni, montre comment "patcher" les fonctions de la bibliothèque (avec SetFunction() d'exec.library) pour, par exemple, customiser le FileRequester.

Dernier point important, la documentation req.doc liste toutes les fonctions de la bibliothèque de manière détaillée, avec syntaxe, description des paramètres, effet, bugs connus... exactement à la manière du RKM Includes & AutoDocs.

ReqLib est un programme de Colin Fox (de Pyramid Designs) et de Bruce Dawson (de CygnusSoft Software). Il est librement distribuable, mais n'est pas placé dans le domaine public (c'est-à-dire que les sources ne sont pas fournies, et que vous n'avez pas le droit de modifier le code exécutable). Vous pouvez l'utiliser dans vos propres programmes, même commerciaux, sans payer ne serait-ce que l'ombre d'une royauté aux auteurs. A ma connaissance, de nombreux programmes utilisent déjà la req.library, dont The Art Department de ASDG, Inc. et ImageLink de Active Circuits, Inc.

PARNET

ParNet n'a absolument aucun rapport avec ce qui précède : il s'agit d'un mini-réseau 2 postes fonctionnant avec le port parallèle de l'Amiga (d'où le nom de "ParNet". Etonnant, non ?).

ParNet permet donc à deux Amigas de co-exister de manière tout-à-fait transparente pour l'un comme pour l'autre. Chacun pourra avoir accès aux devices de l'autre comme s'il s'agissait des siens (lecteur de disquette, disque dur, imprimante, etc.).

Devant la complexité d'un tel logiciel, nous avons décidé de traduire la documentation originale. Pour respecter les vœux des auteurs, nous ne pouvons malheureusement pas inclure la nouvelle documentation dans la distribution, aussi la voici ici imprimée.

```

/* *****
* |_o_o|\ Copyright (c) 1988, 1989, 1990 The Software Distillery.
*
* |. o.| || All Rights Reserved
*
* | . | || Written by Doug Walker and John Toebes
*
* | o | || The Software Distillery
*
* | . | || 405 B3 Gooseneck Drive
*
* ===== Cary, NC 27513
*
* PLINK: DWALKER BIX: DJWALKER
*
* BBS:(919)-460-7430 Deep Thought
*
* *****
*/

```

LISEZ LE FICHIER README DANS CETTE DISTRIBUTION POUR AVOIR DES INFORMATIONS IMPORTANTES SUR LE COPYRIGHT. CONTACTEZ LES AUTEURS AUX ADRESSES CI-DESSUS SI VOUS N'AVEZ PAS CE FICHIER.

Traduction Française par Stéphane SCHREIBER

INSTALLATION DU RESEAU:

1. Fabriquez un câble d'après les instructions incluses dans le fichier 'Cable.doc' ci-joint. Connectez les deux Amigas.
2. Copiez parnet.device dans les répertoire DEVS: des DEUX machines.
3. Insérez une entrée pour le réseau dans le fichier DEVS:MOUNTLIST des deux Amigas. Vous pouvez simplement ajouter le fichier 'mountlist' fourni dans cette distribution à la fin de votre DEVS:MOUNTLIST en utilisant un éditeur de textes quelconque. Choisissez la machine qui sera l'UNIT 1 et mettez la ligne 'Unit =' de sa MOUNTLIST à 1.
4. Copiez netpnet-handler dans le répertoire L: des deux machines.
5. Copiez netpnet-serveur dans le répertoire C: des deux machines.

MISE EN ROUTE DU RESEAU:

1. Lancez les deux serveurs NET:. Sur la machine déclarée en UNIT=0 dans la mountlist, démarrez le serveur avec la commande

```
NETPNET-SERVEUR UNIT 1
```

Sur la machine déclarée en UNIT=1 dans la mountlist, tapez

```
NETPNET-SERVER UNIT 0
```

ou bien, UNIT 0 étant le défaut, vous pouvez simplifier

```
NETPNET-SERVER
```

La raison de cette inversion des UNIT est que vous démarrez un serveur NET: sur l'AUTRE machine.

2. Tapez une commande MOUNT sur les deux Amigas:

MOUNT NET:

3. NET: est maintenant prêt. NET: apparaîtra comme vide la première fois que vous en demanderez le DIRECTORY depuis le CLI. Vous pouvez accéder aux devices de l'autre machine en spécifiant NET:device/chemin (par exemple, NET:DH0/C/LIST). De cette manière, NET: prendra en compte

l'autre device. Le device (DH0 dans ce cas) apparaîtra alors dans le DIRECTORY de NET:. Aucune icône de tiroir n'apparaîtra dans la fenêtre NET: du Workbench tant que vous ne l'aurez pas forcé à les lire. La commande

```
CD NET:device
```

suffit amplement à initialiser le device 'device'.

Le 'device' peut être n'importe quel device AmigaDOS ou volume valide sur l'autre machine.

4. Si vous voulez utiliser NET: avec le Workbench, vous devez fournir une icône pour chaque device pris en compte sur l'autre machine. Pour ce faire, copiez n'importe quelle icône de tiroir en NET:device.info après avoir pris en compte le device 'device'.

```

CD NET:DH0 ; Prends DH0 en compte
COPY SYS:C.INFO NET:DH0.INFO ; Lui fabrique une icône

```

Vous n'avez besoin de faire tout ça qu'une fois pour les disques durs et autres devices permanents, mais RAM: perdra son icône chaque fois que vous ré-initialiserez l'Amiga, vous obligeant à recopier l'icône. (NOTE: un fichier nommé 'node.rinfo' apparaîtra dans la racine de l'autre device. NET: y stocke les icônes que vous lui spécifiez. Si vous supprimez ce fichier, vous supprimez toutes les icônes en même temps).

Notez que si vous effectuez un DIRECTORY de NET:, vous verrez toujours les fichiers '.info' des devices de l'autre machine, MEME S'ILS N'EXISTENT PAS. NET: assume que si les icônes sont là, le device également et ne vérifie pas plus avant, sauf nécessaire. Si les '.info' apparaissent avec la commande DIR mais pas sous Workbench, vous devrez copier une icône de tiroir dans NET: comme décrit plus haut.

La commande NETSTAT donne une représentation visuelle du travail du réseau
Lancez la par

```
RUN NETSTAT NET:
```

et fermez la fenêtre pour quitter.

Verions et améliorations:

VERSION 08/90

- Corrections pour AmigaDOS 2.0
- Support Reset amélioré
- Supporte la connection bilatérale
- Supression d'un niveau de copie de donnée pour gagner en vitesse

VERSION 04/90:

Deuxième version du port parallèle

- Plusieurs bogues mineurs corrigées
- ACTION_SET_DATE implémenté
- Le Reset d'une machine est indépendant de l'autre
- Utilisation du nouveau parnet.device de Matt Dillon

VERSION 08/89:

Première version du port parallèle

VERSION 07/18/89:

- Vitesse accrue grâce à la réduction de la taille des paquets
- Augmente la vitesse de chargement des programmes par le réseau et celle des opérations de directory.

- Suppression du code de débogage
- Rend le code exécutable plus compact et rapide.

- Bogue corrigé dans ACTION_CREATE_DIR

VERSION 05/09/89:

Première version, avec DNET

par STEPHANE SCHREIBER

L'utilitaire de ce mois va vous permettre de sauvegarder en tant qu'image IFF, n'importe quel écran Intuition ouvert au moment de son invocation.

Ce type d'utilitaire existe déjà dans le domaine public, le plus célèbre étant - je pense - ScreenX, de Steve Tibbett (également auteur de VirusX). Malheureusement, ScreenX agit selon un principe qui ne me plaît guère : il ouvre une (petite) fenêtre sur le Workbench lorsqu'il est inactif et carrément un Custom Screen lorsqu'il est actif. Un petit panneau de contrôle est alors proposé, qui permet de choisir entre tous les écrans ouverts, celui qui sera sauvegardé, voire imprimé. Bon.

De mon côté, je préfère et de loin le principe de YASS! (pour "Yet Another Screen Saver !"). Les adeptes d'Unix reconnaîtront la référence : il ajoute un handler à la chaîne d'événements gérée par l'input.device, tout comme le CRBlanker de Frédéric Mazué, proposé dans ce même magazine quelques mois plus tôt. Dès lors, à chaque fois qu'une certaine combinaison de touches est pressée, en l'occurrence CTRL/ALT/DEL (encore une référence...), l'écran de front est sauvegardé par le programme principal dans un fichier IFF tout-à-fait standard, visualisable et éditable par n'importe quel soft de dessin capable de comprendre ce format (c'est-à-dire tous).

PRELIMINAIRES

La routine d'installation de l'input-handler est maintenant classique : on ouvre l'input.device et à l'aide de la commande IND_ADDHANDLER, le handler est ajouté à la chaîne. Sa priorité est de 101, ce qui est amplement suffisant pour le placer en premier dans la chaîne (rappel : l'input-handler d'Intuition dispose d'une priorité de 51). Dorénavant, chaque fois qu'un événement survient (souris, joystick, clavier...), le contrôle passe d'abord par notre handler, qui vérifie dans un premier temps que la tâche principale n'était pas déjà en train de sauvegarder un écran, et dans un deuxième temps, que la bonne combinaison de touches a été pressée. Si ces deux conditions sont remplies, il envoie un signal à la tâche principale, qui commence alors son travail. Notez au passage que vous trouverez dans le listing, un équivalent assembleur des fonctions CreatePort(), DeletePort(), CreateExtIO() et DeleteExtIO() de l'amiga.lib (je déteste linker !).

IFF

Le fichier IFF sauvegardé est parfaitement standard ; simplement, les gens de chez Electronic Arts piqueraient sans doute une crise cardiaque s'ils voyaient de quelle manière on s'y prend...

L'écriture d'un fichier IFF est moins aisée que la lecture : il faut d'abord connaître la taille de chaque chunk avant de pouvoir l'inscrire dans le fichier. Or, pour connaître la taille d'un chunk, il faut justement l'avoir déjà écrit... Ce qui oblige à quelques jongleries avec le DOS et la fonction Seek().

Pour éviter cela, on utilise un petit truc : sachant d'une part que l'on n'écrit que les chunks réellement indispensables (BMHD, CAMG, CMAP et BODY) et que d'autre part, le BODY n'est pas compressé, on peut connaître la taille de la FORM avant même de l'avoir écrite : les chunks BMHD et CAMG sont de longueur constante (respectivement 20 et 4 octets), seule les tailles du CMAP et du BODY sont à calculer. Ce calcul ne pose toutefois aucun problème, tous les paramètres étant issus de la structure Screen et du RastPort de l'écran à sauvegarder. A la taille ainsi calculée, on ajoute encore quelques octets nécessaires aux différents identificateurs de chunks. Pour

résumer, la taille de la FORM IFF sera :

- 4 octets pour "ILBM" ("FORM" et sa taille ne sont pas comptés) ;
- 28 octets pour le BMHD ;
- 12 octets pour le CAMG ;
- 8 octets pour "CMAP", sa taille, et le CMAP lui-même ;
- 8 octets pour "BODY", sa taille, et le BODY lui-même ;

soit 60 octets, plus la taille du CMAP, plus la taille du BODY (essayez, vous verrez bien).

Pour ne pas allonger inutilement le programme, le nom du fichier à écrire est inclus directement dans le source : il s'agit de "Ram:ScreenSaver.image". Ce qui implique qu'à chaque sauvegarde, l'ancienne est écrasée. Il ne tient qu'à vous de modifier cela.

Max
se prend pour Le Sauveur

```
; *****
; * Programme : ScreenSaver
; * Version   : 1.1
; * But      : Sauvegarder l'écran de front en image IFF
; * Auteur   : Max pour ANT
; * Date     : 22 Avril 1991
; * Langage  : Assembleur Devpac II v2.14
; *****

incdir "Include:"
include "exec/memory.i"
include "exec/io.i"
include "exec/interrupts.i"
include "intuition/intuition.i"
include "intuition/intuitionbase.i"
include "graphics/gfx.i"
include "libraries/dos.i"
include "devices/input.i"
include "devices/inputevent.i"

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

opt o+,ow- ; pour retirer les offsets inutiles

; *****
; Quelques macros utiles
EXEC MACRO
    CALLEXEC 1
ENDM

INT MACRO
    movea.l IntBase(a5),a6
    jsr _LVO1(a6)
ENDM

DOS MACRO
    movea.l DosBase(a5),a6
    jsr _LVO1(a6)
ENDM

; *****
REV EQU 33

CAMGMSK EQU (~(V_SPRITES|GENLOCK_VIDEO)&$0000FFFF)

CTRLED EQU IEQUALIFIER_CONTROL
ALTED EQU IEQUALIFIER_LALT|IEQUALIFIER_RALT
QUALMSK EQU CTRLED|ALTED
QUAL1 EQU CTRLED|IEQUALIFIER_LALT
QUAL2 EQU CTRLED|IEQUALIFIER_RALT

; *****
; * Définition du BMHD façon include
rsreset
BitMapHeader rs.b 0
bmhd_w rs.w 1
bmhd_h rs.w 1
bmhd_x rs.w 1
```



```

bmhd_y      rs.w 1
bmhd_nPlanes rs.b 1
bmhd_Masking rs.b 1
bmhd_Compression rs.b 1
bmhd_pad1   rs.b 1
bmhd_transparentColor rs.w 1
bmhd_xAspect rs.b 1
bmhd_yAspect rs.b 1
bmhd_pageWidth rs.w 1
bmhd_pageHeight rs.w 1
bmhd_SIZEOF rs.w 0

; * Définition des variables du programme (pointées par a5)
rsreset
args      rs.l 2 ; Arguments du CLI
IntBase rs.l 1 ; Adresse de base d'intuition.library
DosBase rs.l 1 ; Adresse de base d'exec.library
myTask rs.l 1 ; Ma propre tâche
myPort rs.l 1 ; MsgPort
myIO rs.l 1 ; IOStdReq pour l'input.device
busy rs.b 1 ; TRUE si en train de sauver un écran
iffSig rs.b 1 ; n° du signal alloué pour IFF
iffMsk rs.l 1 ; Mask du signal IFF
filehd rs.l 1 ; FileHandle du fichier de sauvegarde
window rs.l 1 ; Fenêtre active au moment de sauver
pointer rs.l 1 ; Sauvegarde du pointeur de souris
ptrsize rs.l 1 ; Taille et offsets du pointer
scrW rs.w 1 ; Largeur de l'écran en octets
scrH rs.w 1 ; Hauteur de l'écran
scrD rs.w 1 ; Profondeur de l'écran
scrC rs.w 1 ; Nombre de couleurs
VARSIZE rs.w 0

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

; ***** Vérifie si ScreenSaver est déjà chargé
lea port_id(pc),a1
EXEC FindPort
tst.l d0
beq.s Install ; pas encore, donc on s'installe

movea.l d0,a0 ; si oui, on lui signale de quitter
movea.l MP_SIGTASK(a0),a1
move.l #SIGBREAKF_CTRL_C,d0
EXEC Signal
moveq #RETURN_OK,d0 ; et c'est tout.
rts

; ***** Installation
Install moveq #RETURN_FAIL,d7

lea intname(pc),a1 ; ouvre l'intuition.library
moveq #REV,d0
EXEC OpenLibrary
move.l d0,IntBase(a5)
beq NoInt

lea dosname(pc),a1 ; ouvre la dos.library
moveq #REV,d0
EXEC OpenLibrary
move.l d0,DosBase(a5)
beq NoDos

DOS Output
move.l d0,d1
move.l #Hello,d2
moveq #HellLen,d3
DOS Write

suba.l a1,a1 ; où suis-je ?
EXEC FindTask
move.l d0,myTask(a5)

lea port_id(pc),a0 ; crée un MsgPort
moveq #0,d0
bsr CreatePort
move.l d0,myPort(a5)
beq NoPort

movea.l d0,a0 ; crée un bloc IO
moveq #IOSTD_SIZE,d0
bsr CreateExtIO
move.l d0,myIO(a5)
beq NoIO

```

```

lea idname(pc),a0 ; ouvre l'input.device
movea.l d0,a1
moveq #0,d0
moveq #0,d1
EXEC OpenDevice
tst.l d0
bne NoDev

moveq #-1,d0 ; alloue un signal
EXEC AllocSignal
move.b d0,iffSig(a5)
bmi NoSig
moveq #1,d1
lsl.l d0,d1
move.l d1,iffMsk(a5) ; et calcule son masque

movea.l myIO(a5),a1 ; ajoute le handler à la chaîne
move.l #myInter,IO_DATA(a1)
move.w #IND_ADDHANDLER,IO_COMMAND(a1)
EXEC DoIO

; ***** Boucle principale
TheLoop move.l iffMsk(a5),d0 ; il suffira d'un signe...
bset #SIGBREAKF_CTRL_C,d0
EXEC Wait
st busy(a5) ; Qu'on ne me dérange pas !
btst #SIGBREAKF_CTRL_C,d0 ; c'est la fin ?
bne Ciao ; oui, on s'barre

; *****
; Routine de sauvegarde de l'écran
ScreenSave:
movea.l IntBase(a5),a3 ; repère l'écran de front
movea.l ib_FirstScreen(a3),a3
lea sc_BitMap(a3),a4

movea.l IntBase(a5),a0 ; change le pointeur actif
movea.l ib_ActiveWindow(a0),a0
move.l a0>window(a5)
move.l wd_Pointer(a0),pointer(a5)
move.l wd_PtrHeight(a0),ptrsize(a5)
lea myPointer,a1 ; Zzzz...
movem.l (a1)+,d0-d3
INT SetPointer

move.l #ss_file,d1 ; ouvre le fichier de sauvegarde
move.l #MODE_NEWFILE,d2
DOS Open
move.l d0,filehd(a5)
bne.s FileOK ; quelque chose a cloché...

movea.l a3,a0 ; on l'indique par un Beep
INT DisplayBeep ; et on restaure le pointeur
bra SaveEnd ; de la fenêtre active

; ***** Le fichier est ouvert. On peut commencer la sauvegarde
FileOK move.w bm_BytesPerRow(a4),scrW(a5) ; Largeur
move.w bm_Rows(a4),scrH(a5) ; Hauteur
move.b bm_Depth(a4),scrD+1(a5) ; Profondeur
moveq #1,d0
move.w scrD(a5),d1
lsl.l d1,d0
move.w d0,scrC(a5) ; Nb. couleurs

; ***** Ecrit le header iff "FORMxxxxILEM" (12 octets)
WriteFORM:
lea theFORM(pc),a0 ; voir article pour le détail du
moveq #60,d0 ; "calcul" de la taille du fichier
move.w scrW(a5),d1
mulu scrH(a5),d1
mulu scrD(a5),d1 ; taille du BODY
add.l d1,d0
move.w scrC(a5),d1
mulu #3,d1 ; taille du CMAP
add.l d1,d0
move.l d0,4(a0) ; Longueur de la FORM

move.l filehd(a5),d1
move.l a0,d2
moveq #12,d3
DOS Write

; ***** Ecrit le chunk BMHD (20 octets + "BMHDxxxx")
WriteBMHD:
lea theBMHD(pc),a0

```



```

move.l #bmhd_SIZEOF,4(a0) ; Longueur du BMHD

move.w scrW(a5),d0 ; Largeur de l'image
lsl.w #3,d0 ; *8 pour nb. pixels
move.w scrH(a5),d1 ; hauteur

move.w d0,8+bmhd_w(a0)
move.w d1,8+bmhd_h(a0)
clr.w 8+bmhd_x(a0)
clr.w 8+bmhd_y(a0)
move.b scrD+1(a5),8+bmhd_nPlanes(a0)
clr.b 8+bmhd_Masking(a0)
clr.b 8+bmhd_Compression(a0) ; Pas de compression !
clr.b 8+bmhd_pad1(a0)
clr.w 8+bmhd_transparentColor(a0)
move.b #$01,8+bmhd_xAspect(a0) ; pas vraiment ça...
move.b #$01,8+bmhd_yAspect(a0)
move.w d0,8+bmhd_pageWidth(a0)
move.w d1,8+bmhd_pageHeight(a0)

move.l filehd(a5),d1
move.l a0,d2
moveq #bmhd_SIZEOF+8,d3
DOS Write

; ***** Ecrit le chunk CMAP ((3*nb. couleurs) octets + "CMAPxxxx")
WriteCMAP:
lea theCMAP(pc),a0
movea.l sc_ViewPort+vp_ColorMap(a3),a1
movea.l cm_ColorTable(a1),a1 ; table des couleurs
move.w scrC(a5),d0
mulu #3,d0
move.l d0,4(a0)

lea 8(a0),a2
move.w scrC(a5),d2
subq.w #1,d2
move.w (a1)+,d0 ; Valeur RGB
move.w d0,d1
lsl.w #4,d1
andi.b #$f0,d1
move.b d1,(a2)+ ; Composante Rouge
move.w d0,d1
andi.b #$f0,d1
move.b d1,(a2)+ ; Composante Verte
lsl.b #4,d0
move.b d0,(a2)+ ; Composante Bleue
dbra d2,CMAPlp

move.l filehd(a5),d1
move.l a0,d2
move.l 4(a0),d3
addq.l #8,d3
DOS Write

; ***** Ecrit le chunk CAMG "CAMG0004xxxx" (12 octets)
WriteCAMG:
lea theCAMG(pc),a0
move.l #00000004,4(a0) ; Longueur du CAMG
move.w sc_ViewPort+vp_Modes(a3),d0
andi.l #CAMGMSK,d0 ; Masque certains bits
move.l d0,8(a0)

move.l filehd(a5),d1
move.l a0,d2
moveq #12,d3
DOS Write

; ***** Ecrit le chunk BODY ((h*l*p) octets + "BODYxxxx")
WriteBODY:
lea theBODY(pc),a0
move.w scrW(a5),d0
mulu scrH(a5),d0
mulu scrD(a5),d0
move.l d0,4(a0)

move.l filehd(a5),d1
move.l a0,d2
moveq #8,d3
DOS Write

moveq #0,d7
move.w scrW(a5),d6
ext.l d6

move.w scrH(a5),d5 ; Ecrit les lignes de l'image

BODY_1 subq.w #1,d5 ; une par une, plan par plan
move.w scrD(a5),d4
subq.w #1,d4
lea km_Planes(a4),a2
BODY_2 move.l filehd(a5),d1
move.l (a2)+,d2
add.l d7,d2
move.l d6,d3
DOS Write
dbra d4,BODY_2
add.l d6,d7
dbra d5,BODY_1

; ***** Ferme le fichier
move.l filehd(a5),d1
DOS Close

; ***** Restaure le pointeur de souris original
SaveEnd movea.l window(a5),a0
move.l pointer(a5),d0
beq.s NoMouse
movea.l d0,a1
moveq #0,d0
moveq #0,d1
moveq #0,d2
moveq #0,d3
move.b ptrsize+0(a5),d0 ; pourquoi ce sont des
move.b ptrsize+1(a5),d1 ; octets, hein, pourquoi ?
move.b ptrsize+2(a5),d2
move.b ptrsize+3(a5),d3 ; pourquoi tant de haine ?
INT SetPointer
bra.s Again

NoMouse INT ClearPointer

Again sf busy(a5) ; on peut à nouveau être signalé
bra TheLoop ; boucle sur Wait()

; *****
; Fin du programme. Libère et ferme tout.
Ciao moveq #RETURN_OK,d7

Exit movea.l myIO(a5),a1 ; enlève le handler de la chaîne
move.l #myInter,IO_DATA(a1)
move.w #IND_REMHANDLER,IO_COMMAND(a1)
EXEC DoIO

moveq #0,d0 ; libère le signal
move.b iffSig(a5),d0
EXEC FreeSignal

NoSig movea.l myIO(a5),a1 ; ferme l'input.device
EXEC CloseDevice

NoDev movea.l myIO(a5),a0 ; détruit le bloc IO
bsr DeleteExtIO

NoIO movea.l myPort(a5),a0 ; détruit le MsgPort
bsr DeletePort

NoPort movea.l DosBase(a5),a1 ; ferme la dos.library
EXEC CloseLibrary

NoDos movea.l IntBase(a5),a1 ; ferme l'intuition.library
EXEC CloseLibrary

NoInt move.l d7,d0 ; code d'erreur dans d0
rts ; et retour au CLI

; *****
; Routines tirées d'Amiga.Lib

; ***** Port=CreatePort(Name,Pri)
CreatePort:
movem.l d2-d7/a2/a6,-(sp)
move.l a0,d7 ; d7=name
move.b d0,d6 ; d6=pri
moveq #-1,d0
CALLEXEC AllocSignal
cmpi.b #-1,d0
bne.s .SigOk
moveq #0,d0
bra.s .CP_Ret
.SigOk move.l d0,d5 ; d5=SigBit
moveq #$22,d0 ; sizeof(struct MsgPort)
move.l #$10001,d1 ; MEMF_PUBLIC|MEMF_CLEAR

```



```

    CALLEEXEC AllocMem
    move.l d0,d4 ; d4=port
    bne.s .PortOk
.NoPort move.l d5,d0
    CALLEEXEC FreeSignal
    moveq #0,d0
    bra.s .CP_Ret
.PortOk move.l d4,a2
    move.l d7,MP+LN_NAME(a2)
    move.b d6,MP+LN_PRI(a2)
    move.b #NT_MSGPORT,MP+LN_TYPE(a2)
    move.b #PA_SIGNAL,MP_FLAGS(a2)
    move.b d5,MP_SIGBIT(a2)
    suba.l a1,a1
    CALLEEXEC FindTask
    move.l d0,MP_SIGTASK(a2)
    tst.l d7
    beq.s .List
    move.l a2,a1
    CALLEEXEC AddPort
    bra.s .CP_Ok
.List lea MP_MSGLIST(a2),a1
    NEWLIST a1
.CP_Ok move.l a2,d0
.CP_Ret move.l (sp)+,d2-d7/a2/a6
    rts

; ***** void DeletePort(port)
DeletePort:
    move.l d2-d7/a2/a6,-(sp)
    move.l a0,a2
    tst.l MP+LN_NAME(a2)
    beq.s .NoName
    move.l a2,a1
    CALLEEXEC RemPort
.NoName moveq #-1,d0
    move.l d0,MP_SIGTASK(a2)
    move.l d0,MP_MSGLIST+LN_HEAD(a2)
    moveq #0,d0
    move.b MP_SIGBIT(a2),d0
    CALLEEXEC FreeSignal
    move.l a2,a1
    moveq #$22,d0
    CALLEEXEC FreeMem
    move.l (sp)+,d2-d7/a2/a6
    rts

; ***** IORequest=CreateExtIO(Port,Size)
CreateExtIO:
    move.l d2-d7/a2/a6,-(sp)
    move.l d0,d6 ; d6=Size
    move.l a0,d7 ; d7=Port
    bne.s .PortOk
.NoPort moveq #0,d0
    bra.s .CE_Ret
.PortOk move.l d6,d0
    move.l #$10001,d1
    CALLEEXEC AllocMem
    tst.l d0
    bne.s .IO_Ok
.NoIO moveq #0,d0
    bra.s .CE_Ret
.IO_Ok move.l d0,a2
    move.b #NT_MESSAGE,IO+MP+LN_TYPE(a2)
    move.w d6,IO+MP+LN_LENGTH(a2)
    move.l d7,IO+MP+LN_REPLYPORT(a2)
    move.l a2,d0
.CE_Ret move.l (sp)+,d2-d7/a2/a6
    rts

; ***** void DeleteExtIO(IORequest)
DeleteExtIO:
    move.l d2-d7/a2/a6,-(sp)
    moveq #-1,d0
    move.b d0,IO+MP+LN_TYPE(a0)
    move.l d0,IO+MP+LN_REPLYPORT(a0)
    move.l d0,IO_DEVICE(a0)
    moveq #0,d0
    move.w IO+MP+LN_LENGTH(a0),d0
    move.l a0,a1
    CALLEEXEC FreeMem
    move.l (sp)+,d2-d7/a2/a6
    rts

; *****
; InputHandler. Vérifie si CTRL/ALT/DEL est pressé

```

```

; et le signale au programme principal le cas échéant.
; Rappel : a1 contient is_Data, soit nos variables
myInputHandler:
    tst.b busy(a1) ; sauf si on est déjà en train
    bne.s .Rien ; de sauver

    cmpi.b #IECLASS_RAWKEY,ie_Class(a0)
    bne.s .Rien ; c'est pas le clavier

    cmpi.w #$46,ie_Code(a0) ; DEL ?
    bne.s .Rien

.Maybe move.w ie_Qualifier(a0),d0 ; touches spéciales
    andi.w #QUALMSK,d0 ; masque le code
    cmpi.w #QUALMSK,d0 ; CTRL/LALT/RALT ?
    beq.s .Signal
    cmpi.w #QUAL1,d0 ; CTRL/LALT ?
    beq.s .Signal
    cmpi.w #QUAL2,d0 ; CTRL/RALT ?
    bne.s .Rien

.Signal move.l a0,-(sp) ; sauve a0
    move.l iffMsk(a1),d0
    movea.l myTask(a1),a1
    EXEC Signal ; envoie le signal
    movea.l (sp)+,a0 ; récupère a0

    movea.l ie_NextEvent(a0),a0 ; saute cet évènement

.Rien move.l a0,d0
    rts

; *****
VARS dcb.b VARSIZE,0 ; espace des variables

theFORM dc.l "FORM",0,"ITEM"
theBMHD dc.l "BMHD",0
    dcb.b bmhd_SIZEEOF
theCMAP dc.l "CMAP",0
    dcb.b (64*3) ; place pour 64 couleurs
theCAMG dc.l "CAMG",0,0
theBODY dc.l "BODY",0

myInter dc.l 0,0 ; is_Node.ln_Succ, is_Node.ln_Pred
    dc.b 0,101 ; is_Node.ln_Type, is_Node.ln_Pri
    dc.l hndl_id ; is_Node.ln_Name
    dc.l VARS ; is_Data
    dc.l myInputHandler ; is_Code

intname dc.b "intuition.library",0
dosname dc.b "dos.library",0
idname dc.b "input.device",0,0

port_id dc.b "ScreenSaver.Port",0,0
hndl_id dc.b "ScreenSaver.InputHandler",0,0
ss_file dc.b "ram:ScreenSaver.image",0

Hello dc.b "ScreenSaver v1.0 installé",0a
    dc.b " 1991 par Max pour l'ANT",0a
HellLen EQU *-Hello
even

; *****
; Définition du pointeur indiquant la sauvegarde
; Doit de trouver en CHIP-RAM !
section PointerDefs,DATA_C

myPointer:
    dc.l 24,16,0,0 ; Height, Width, Xoffset, Yoffset
    dc.w $0000,$0000
    dc.w $0600,$0600,$0F40,$0F40,$3FE0,$3FE0,$7FE0,$7FE0
    dc.w $61F0,$7FF0,$7BF8,$7FF8,$7F78,$FFF8,$61FC,$7FFC
    dc.w $7F0C,$7FFC,$3FDE,$3FFE,$7FBC,$7FFC,$3F0C,$3FFC
    dc.w $1FF8,$1FF8,$07F0,$07F0,$01C0,$01C0,$0700,$0700
    dc.w $0FC0,$0FC0,$0680,$0680,$0000,$0000,$00C0,$00C0
    dc.w $00E0,$00E0,$0040,$0040,$0000,$0000,$0000,$0000
    dc.w $0000,$0000

; *****
END

```

Par MAX

ExecBase II : le 68000 de A à Z

Voici la fin de notre étude des instructions du 68000. Reportez-vous aux deux précédents numéros de l'ANT pour plus de détails sur la manière de lire les tableaux.

Le mois prochain (et non ce mois-ci comme annoncé un peu vite dans notre dernier numéro...), nous entamerons une petite mais courte série sur l'utilité et l'utilisation de certaines instructions obscures du 68000, comme LINK, CHK, RTR ou encore DBPL. Mais ceci est une autre histoire...

OR (ou logique)

Syntaxe OR <ea>, Dn
OR Dn, <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 0 0 0 REGISTRE OP-MODE ADRESSE EFFECTIVE

OP-MODE : Octet Mot Long
000 001 010 (syntaxe 1)
100 101 110 (syntaxe 2)

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

ORI (ou logique immédiat)

Syntaxe ORI #donnée, <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 0 0 0 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long
La donnée est codée dans le mot (TAILLE=00 ou 01) ou dans le mot long (TAILLE=10) suivant le code opératoire.

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

ORI to CCR (ou logique immédiat avec CCR)

Syntaxe ORI #donnée, CCR
Taille Octet

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0

La donnée (8 bits) est codée dans le mot suivant le code opératoire.

Flags N : 1 si le bit 3 de la donnée est 1, inchangé sinon
Z : 1 si le bit 2 de la donnée est 1, inchangé sinon
V : 1 si le bit 1 de la donnée est 1, inchangé sinon
C : 1 si le bit 0 de la donnée est 1, inchangé sinon
X : 1 si le bit 4 de la donnée est 1, inchangé sinon

ORI to SR (ou logique immédiat avec SR)

Instruction privilégiée
Syntaxe ORI #donnée, SR
Taille Mot

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 0 0 0 0 0 0 1 1 1 1 0 0

La donnée (16 bits) est codée dans le mot suivant le code opératoire.

Flags N : 1 si le bit 3 de la donnée est 1, inchangé sinon
Z : 1 si le bit 2 de la donnée est 1, inchangé sinon
V : 1 si le bit 1 de la donnée est 1, inchangé sinon
C : 1 si le bit 0 de la donnée est 1, inchangé sinon
X : 1 si le bit 4 de la donnée est 1, inchangé sinon

PEA (empilement d'une adresse effective)

Syntaxe PEA <ea>
Taille Mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 0 0 0 0 1 ADRESSE EFFECTIVE

Flags Inchangés

RESET (remise à 0 des circuits externes)

Instruction privilégiée
Syntaxe RESET
Taille -
Format \$4E70
Flags Inchangés

ROL, ROR (rotation)

Syntaxe ROLx Dx, Dy
RORx #donnée, Dy
RORx <ea>
X=L (Left, gauche) ou R (Right, droite)

Format Décalage d'un registre :
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 1 0 NOMBRE D TAILLE I 1 1 REGISTRE

Décalage d'une adresse :
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 1 0 0 1 1 D 1 1 ADRESSE EFFECTIVE

NOMBRE : nombre de décalages ou numéro du registre de données le contenant (0-7)
D : 0=décalage à gauche, 1=décalage à droite
TAILLE : 00=octet, 01=mot, 10=mot long
I : 0=NOMBRE contient le nombre de décalages (0-7, 0=8)
1=NOMBRE indique le registre contenant le nombre de décalages (modulo 64)

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : 1 si le dernier bit sorti de l'opérande était 1
X : idem C

ROXL, ROXR (rotation avec X)

Syntaxe ROXLx Dx, Dy
ROXR #donnée, Dy
ROXR <ea>
X=L (Left, gauche) ou R (Right, droite)

Format Décalage d'un registre :
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 1 0 NOMBRE D TAILLE I 1 0 REGISTRE

Décalage d'une adresse :
15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 1 1 0 0 1 0 D 1 1 ADRESSE EFFECTIVE

NOMBRE : nombre de décalages ou numéro du registre de données le contenant (0-7)
D : 0=décalage à droite, 1=décalage à gauche
TAILLE : 00=octet, 01=mot, 10=mot long
I : 0=NOMBRE contient le nombre de décalages (0-7, 0=8)
1=NOMBRE indique le registre contenant le nombre de décalages (modulo 64)

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : 1 si le dernier bit sorti de l'opérande était 1
X : idem C

RTE (retour d'exception)

Syntaxe RTE
Taille -
Format \$4E73
Flags Positionnés d'après le mot dépilé

RTR (retour et restitution de CCR)

Syntaxe RTR
Taille -
Format \$4E77
Flags Positionnés d'après le mot dépilé
Note CCR doit avoir été empilé avant JSR ou BSR

RTS (retour de sous-routine)

Syntaxe RTS
Taille -
Format \$4E75
Flags Inchangés

SBCD (soustraction décimale avec le bit d'extension)

Syntaxe SBCD Dy, Dx
SBCD -(Ay), -(Ax)
Taille Octet

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 0 0 0 Rx 1 0 0 0 0 M Ry

Rx : numéro du registre source (0-7)

Ry : numéro du registre destination (0-7)
M : Mode (0=registres de données, 1=registres d'adresse)

Flags N : indéfini
Z : 0 si résultat non-nul, inchangé sinon
V : indéfini
C : 1 si retenue, 0 sinon
X : idem C

Scc (positionnement conditionnel)

Syntaxe Scc <ea>
Taille Octet

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 1 CONDITION 1 1 ADRESSE EFFECTIVE

Flags Inchangés

STOP (arrêt du processeur)

Syntaxe STOP #donnée
Taille -
Format \$4E72

La donnée (16 bits) est codée dans le mot suivant le code opératoire
Flags Positionnés selon la donnée immédiate.
Note La donnée est chargée dans SR et le processeur passe en état arrêté jusqu'à ce qu'une exception TRACE, une interruption ou un RESET survienne

SUB (soustraction binaire)

Syntaxe SUB <ea>, Dn
SUB Dn, <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 0 0 1 REGISTRE OP-MODE ADRESSE EFFECTIVE

OP-MODE : Octet Mot Long
000 001 010 (syntaxe 1)
100 101 110 (syntaxe 2)

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : 1 si débordement, 0 sinon
C : 1 si retenue, 0 sinon
X : idem C

SUBA (soustraction binaire d'un registre d'adresse)

Syntaxe SUBA <ea>, An
Taille Mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 0 0 1 REGISTRE OP-MODE ADRESSE EFFECTIVE

OP-MODE : 011=mot, 111=mot long

Flags Inchangés

SUBI (soustraction binaire immédiate)

Syntaxe SUBI #donnée, <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 0 0 0 0 1 0 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : 1 si débordement, 0 sinon
C : 1 si retenue, 0 sinon
X : idem C

SUBQ (soustraction binaire immédiate rapide)

Syntaxe SUBQ #donnée, <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 1 DONNEE 0 TAILLE ADRESSE EFFECTIVE

DONNEE : donnée immédiate sur 8 bits
TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : 1 si débordement, 0 sinon
C : 1 si retenue, 0 sinon
X : idem C

SUBX (soustraction avec le bit d'extension)

Syntaxe SUBX Dy, Dx

SUBX -(Ay), -(Ax)
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
1 0 0 1 Rx 1 TAILLE 0 0 M Ry

Rx : registre source
Ry : registre destination
M : 0=registres de données, 1=registres d'adresse

Flags N : 1 si résultat négatif, 0 sinon
Z : 0 si résultat non-nul, inchangé sinon
V : 1 si débordement, 0 sinon
C : 1 si retenue, 0 sinon
X : idem C

SWAP (échange des mots)

Syntaxe SWAP Dn
Taille -

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0 REGISTRE

REGISTRE : numéro du registre de données (0-7)

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

TAS (test et positionnement)

Syntaxe TAS <ea>
Taille Octet

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 ADRESSE EFFECTIVE

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

TRAP (instruction TRAP)

Syntaxe TRAP #numéro
Taille -

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 0 NUMERO

NUMERO : numéro de l'exception TRAP à déclencher (0-15)

Flags Inchangés

TRAPV (instruction TRAP)
Syntaxe TRAPV
Taille -
Format \$4E76
Flags Inchangés

TST (test)

Syntaxe TST <ea>
Taille Octet, mot, mot long

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 TAILLE ADRESSE EFFECTIVE

TAILLE : 00=octet, 01=mot, 10=mot long

Flags N : 1 si résultat négatif, 0 sinon
Z : 1 si résultat nul, 0 sinon
V : toujours 0
C : toujours 0
X : inchangé

UNLK (libération de liens)

Syntaxe UNLK An
Taille -

Format 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
0 1 0 0 1 1 1 0 0 1 0 1 1 1 1 1 REGISTRE

REGISTRE : numéro du registre d'adresse (0-7)

Flags Inchangés

La création de menus déroulants sur Amiga s'effectue, comme bien d'autres choses d'ailleurs, avec l'aide de l'interface logicielle Intuition. Celle-ci permet l'affichage des menus et la gestion des interactions entre l'utilisateur et le programme via ce menu.

La première partie de cet article va donc consister en une description des menus : comment les créer et comment les gérer. Un programme d'exemple constituant la seconde partie mettra en pratique toute cette théorie. Dernier point avant que de commencer : les termes anglais entre parenthèses dissimulés tout au long de cet article, sont les termes "officiels" que vous trouverez dans la documentation développeur de Commodore, ceci afin de permettre à ceux qui la possèdent, de s'y retrouver plus facilement.

DEFINITION ET ACCES

Pour activer un menu Intuition, l'utilisateur doit appuyer sur le bouton droit de la souris. Intuition affiche alors la barre de menus (menu strip) liée à la fenêtre active, en haut de l'écran. Elle est composée d'un certain nombre de mots, baptisés "intitulés" ou "titres" des menus.

Quand l'utilisateur positionne le pointeur de la souris sur un menu, il le sélectionne : le titre du menu apparaît alors en surbrillance et une boîte contenant les différentes options qu'il propose (menu items) apparaît juste en dessous. La largeur de cette boîte est indépendante de celle du titre du menu. Sa hauteur dépend quant à elle du nombre d'options dans le menu et de leur hauteur. Lorsque l'utilisateur sélectionne une option (toujours à l'aide du pointeur de la souris), celle-ci apparaît à son tour en surbrillance. La confirmation de la sélection se fait tout simplement en relâchant le bouton droit de la souris, l'option choisie étant toujours en surbrillance. Une option de menu peut faire apparaître un sous-menu (sub-menu) présentant lui-même d'autres options.

HIERARCHIE

La description ci-dessus, bien qu'un peu fastidieuse, nous permet d'appréhender la hiérarchie au sein des menus. En effet, on voit que l'on peut lier X menus à une fenêtre, chacun pouvant contenir Y options, pouvant elles-mêmes contenir Z sous-options. Chaque entité de même niveau est liée à la suivante par une liste chaînée simple.

INITIALISATION

Lorsque l'on veut gérer des menus, il faut d'abord les penser. On dessine sur papier la structure des menus, comprenant toutes les entités associées en partant des sous-options, puis les options, et en finissant par les menus. A noter que divers utilitaires du domaine public ou du commerce permettent de simplifier cette tâche. Leur principale caractéristique est de produire un code source C ou assembleur que le programmeur n'aura plus qu'à inclure dans son projet. Le plus célèbre est PowerWindows de je-ne-sais-plus-qui-mais-ce-n'est-pas-du-domaine-public.

STRUCTURE D'UN MENU

Comme toujours dans l'Amiga, un menu (comprenez "un titre de menu") est défini par une structure C, que voici :

```
struct Menu
{
    struct Menu *NextMenu;
    SHORT LeftEdge, TopEdge, Width, Height;
    USHORT Flags;
    UBYTE *MenuName;
    struct MenuItem *FirstItem;
};
```

Les différents champs de cette structure sont décrits ci-après.

NextMenu : pointeur sur le menu suivant dans la liste. Lorsqu'il s'agit du dernier menu de la barre, NextMenu doit être initialisé à NULL.

LeftEdge, TopEdge, Width, Height : taille de la boîte de sélection du menu. LeftEdge représente la position de la boîte relativement à celle de l'écran. Width représente sa largeur. TopEdge et Height ne servent pas dans la version actuelle d'Intuition, plus tard peut être (pour le moment on les met à 0 et on en parle plus).

Flags : peuvent être fixés par le programme et par Intuition. Ce sont :
MENUENABLED : indique si le menu est ou non sélectionnable. Si ce flag n'est pas présent lorsque l'on soumet le menu à Intuition, il n'est pas sélectionnable (de même que les options qui l'accompagnent). Il est possible d'autoriser ou non l'accès à un menu en cours de programme à l'aide des fonctions OnMenu() et OffMenu().

MIDRAWN : positionné par Intuition lorsque le menu est affiché.

MenuName : pointeur sur une chaîne de caractères (terminée par un '0') qui représente l'intitulé du menu. Intuition affichera ce texte dans la boîte de sélection du menu.

FirstItem : pointeur sur la première option du menu qui est une structure MenuItem.

STRUCTURE D'UNE OPTION

La structure C associée aux fonctions et sous-fonctions est la même, mais elle ne s'utilise pas de la même manière. Cette structure est la suivante.

```
struct MenuItem
{
    struct MenuItem *NextItem;
    SHORT LeftEdge, TopEdge, Width, Height;
    USHORT Flags;
    APTR ItemFill;
    BYTE Command;
    struct MenuItem *SubItem;
    USHORT NextSelect;
};
```

NextItem : pointeur sur la fonction suivante dans la liste chaînée. Le dernier élément dans la liste doit être initialisé à NULL.

LeftEdge, TopEdge, Width, Height : taille et position de la boîte de sélection de l'option (zone dans laquelle l'option est sélectionnée par le pointeur de souris). LeftEdge et TopEdge représentent la position relative du coin haut-gauche de la boîte par rapport à la position du coin haut-gauche de celle du menu. Width et Height représentent respectivement la largeur et la hauteur de la boîte.

Flags : positionnée par le programme ou par Intuition. Ce sont :

CHECKIT : place un CheckMark devant l'option lorsqu'elle est sélectionnée (interrupteur on/off). La structure NewWindow de la fenêtre à laquelle le menu est associée, contient un champ nommé CheckMark pouvant pointer sur une imagerie personnelle pour le checkmark.

CHECKED : si CHECKIT est positionné, CHECKED rend l'option active (on) dès la mise en place du menu.

ITEMTEXT : mis à 1 si l'option de menu est un texte, à 0 si c'est une image.

COMMSEQ : mis à 1 si l'option peut également être sélectionnée au clavier (Amiga droite + une touche). Le code ASCII de cette touche se trouve alors dans le champ Command (voir ci-dessous).

ITEMENABLED : mis à 1 l'option est sélectionnable, sinon elle apparaît en grisé. Il est possible d'autoriser ou d'interdire l'accès à une option en cours de programme à l'aide des fonctions OnMenu() et OffMenu() que nous décrirons ultérieurement.

NOTE : Si une option devient non sélectionnable, toutes les sous-options qui lui sont associées sont également inaccessibles.

HIGH Flags : indique le mode de surbrillance à appliquer à cette option. Il faut positionner un et un seul des bits ci-dessous.

INTUITION

HIGHCOMP : l'option sélectionnée apparaît en vidéo inversée (cas le plus fréquent).

HIGHBOX : traçage d'une boîte (rectangle) autour de la boîte de sélection de l'option (souvent pour "Quit").

HIGHIMAGE : affichage d'un intitulé complémentaire (texte ou image).

HIGHNONE : aucune surbrillance, il n'y a pas de différence entre l'intitulé sélectionné ou non.

Les flags suivants sont positionnés plus particulièrement par Intuition.

ISDRAWN : mis à 1 quand le sous-menu associé à cette option est affiché.

HIGHITEM : mis à 1 lorsque l'option est actuellement affichée en surbrillance.

MutualExclude : là, de plus amples explications me semblent nécessaires. Vous pouvez choisir d'avoir certaines options, déclarées avec le flag **CHECKIT** qui, sélectionnées, désélectionnent d'autres options. C'est ce que l'on appelle l'exclusion mutuelle. Prenons un exemple porté sur la boisson. Un personnage a la possibilité de choisir d'emporter à boire dans quatre types de contenants : une fiole, une bouteille, un bidon ou un tonneau. Il est raisonnable de penser que le personnage choisissant le tonneau ne pourra pas emporter en plus une bouteille ou un bidon. La fiole, elle, peut toujours se glisser dans une poche. Par contre il pourra très bien prendre le bidon, la bouteille et la fiole en même temps (les camarades de l'ANT y arrivent sans problème). Cela signifie que la possession du tonneau exclut la bouteille et le bidon, de même que la possession du bidon ou de la bouteille (à fortiori des deux) exclut le tonneau.

Le champ **MutualExclusion** est un mot long (type **LONG** sur 32 bits) où chaque bit représente une option dans la liste correspondant à un menu. Le premier dans la liste a le bit 0, le second le bit 1, etc. Ce qui signifie dans un premier temps que seuls 32 options appartenant à un même menu peuvent utiliser ce champ (ce qui n'est déjà pas si mal !). Lorsque le Nième bit du champ **MutualExclude** de l'option X est positionné, l'option N est automatiquement exclue lorsque l'option X est sélectionnée. Dans notre exemple, on va mettre \$fff6 pour tonneau (on exclut tous les autres sauf la fiole et lui). Pour le bidon et la bouteille, on met \$0001 car on exclut le tonneau seulement. Pour la fiole on mettra \$0000 car on n'exclut rien du tout.

ItemFill : pointeur sur l'intitulé de la fonction, en l'occurrence une structure **IntuiText** si le flag **ITEMTEXT** est positionné, sinon c'est une structure **Image**.

SelectFill : si le flag **HIGHIMAGE** est positionné, Intuition substituera à la structure contenue dans **ItemFill** celle contenue dans **SelectFill** lorsque l'option sera sélectionnée. **SelectFill** est donc un pointeur sur une structure **IntuiText** ou **Image**, selon la présence ou non du flag **ITEMTEXT**.

NOTE : on ne peut pas afficher une image à la place d'un texte, et vice-versa.

Command : code ASCII du caractère composant le raccourci-clavier lorsque le flag **COMMSEQ** est positionné. L'utilisateur peut choisir directement l'option en appuyant simultanément sur la touche Amiga droite et sur le caractère contenu dans ce champ. Au niveau programme proprement dit, tout se passe comme si la fonction avait été sélectionnée à la souris.

NOTE : aucune vérification n'est faite pour voir si une même commande est utilisée plusieurs fois dans un menu. De plus, Intuition ne fait pas la différence entre les majuscules et minuscules ; s'il en a besoin, le programme doit tester lui-même si une touche Shift (ou Caps Lock) est enfoncée au moyen du champ **Qualifier** de la structure **IntuiMessage** (cf. plus loin).

SubItem : pointeur sur la première sous-option de cette option. Ce champ est ignoré si l'option courante fait déjà partie d'un sous-menu.

NextSelect : dernier champ de la structure, mais il est loin d'être le moins important. En effet, si l'utilisateur a la possibilité de sélectionner plusieurs options dans un menu en une seule opération (en cliquant avec le bouton de

gauche de la souris les différentes options, tout en maintenant le bouton droit de la souris appuyé), rien dans l'**IntuiMessage** n'indique au programme que c'est bel et bien ce qui s'est passé. Pour garder une trace de cette sélection multiple et donc permettre de traiter toutes les options choisies, Intuition remplit le champ **NextSelect** avec le numéro de l'option suivante. Bon allez, je suis compréhensif et vous livre un petit bout de source qui illustre la manière de s'y prendre.

```
while (selection != MENUNULL)
{
    option = ItemAddress(menu, selection);
    /* traitement de l'option */
    ...
    /* récupération de l'adresse de l'option suivante */
    selection = option->NextSelect;
}
```

Pas trop difficile ?

MISE EN PLACE DU MENU

On a donc décrit la structure de son petit menu, en suivant les explications ci-dessus et en s'imprégnant de l'exemple ci-dessous (le va et vient est terminé, on continue).

Première opération à réaliser, ouvrir une fenêtre à l'aide de la fonction **OpenWindow()**. En effet, pour attacher un menu à une fenêtre, il semble raisonnable d'ouvrir une fenêtre (si, si, j'y jure !). Comme déjà dit plus haut, le champ **CheckMark** de la structure **NewWindow** peut pointer sur une structure **Image** décrivant un checkmark personnel à utiliser pour TOUTES les options du menu ayant les flags **CHECKIT** et **CHECKED** positionnés. Ensuite, on attache le menu à la fenêtre, à l'aide de la fonction **SetMenuStrip()**.

```
SetMenuStrip(fenetre, menu)
```

- fenêtre est un pointeur sur une structure **Window** créée par **OpenWindow()**. Il s'agit de la fenêtre à laquelle on veut attacher le menu.
- menu est un pointeur sur le premier menu de la liste chaînée des menus à afficher.

Bien entendu, il ne se passe rien de particulier lorsque l'utilisateur sélectionne une option du menu. C'est au programme de réagir en conséquence. Il faut donc écrire un petit bout de code pour savoir ce que l'utilisateur a sélectionné et quelle fonction activer. Pour ce faire, on va encore se faire aider par Intuition (brave petite).

A chaque fois que l'utilisateur active le système de gestion de menus en pressant le bouton droit de la souris, Intuition génère un message (sous la forme d'une structure **IntuiMessage**) de classe **MENUPICK**. Vous pouvez donc tester les messages que vous envoie Intuition pour savoir si oui ou non il s'agit de l'activation d'un menu. Voici le résultat en quelques lignes.

```
struct Window *fenetre;
struct MsgPort *port;
struct IntuiMessage *message;
ULONG classe_message;
UWORD code_message, qual_message;
BOOL fini;

/* initialisation diverses */
...

port = fenetre->UserPort;
/* boucle d'attente */
while(!fini)
{
    WaitPort(port);
    message = (struct IntuiMessage *)GetMsg(port);
    while (message)
    {
        classe_message = message->Class;
        code_message = message->Code;
        qual_message = message->Qualifier;
        ReplyMessage(message);
        if (classe_message == MENUPICK)
            /* traitement de la fonction */
            ...
            /* récupération du message suivant */
            message = (struct IntuiMessage *)GetMsg(port);
    }
}
```

Le programme fourni ci-dessous utilise ce principe.

Il nous faut maintenant déterminer quels menus, options, et sous-options ont été sélectionnés. Une fois de plus, Intuition nous aide (que ferait-on sans

elle?). En effet, lorsqu'il envoie une structure IntuiMessage de classe MENUPICK, celle-ci contient également dans son champ Code un WORD correspondant au menu, à l'option et éventuellement à la sous-option sélectionnée. Ce mot long étant assez peu parlant par lui-même, on va déterminer quelle option précisément a été sélectionnée à l'aide des macros :

MENUNUM(Code) qui renvoie le numéro du menu sélectionné ;
ITEMMENU(Code) qui renvoie le numéro de la fonction sélectionnée ;
SUBNUM(Code) qui renvoie le numéro de la sous-fonction ;

Les numéros commencent toujours à zéro et non à un.

Si l'on active la fonction 2 du menu 4 on a :

```
MENUNUM(Code) = 3
ITEMNUM(Code) = 1
SUBNUM(Code) = NOSUB
```

Si l'on active la sous-fonction 3 de la fonction 4 du menu 1 on a :

```
MENUNUM(Code) = 0
ITEMNUM(Code) = 3
SUBNUM(Code) = 2
```

Comme on peut le remarquer, lorsqu'aucune sous-option n'est sélectionnée, SUBNUM() renvoie NOSUB. De même, si aucune option n'est sélectionnée, ITEMNUM() renvoie NOITEM. Enfin, si aucun menu n'est sélectionné (on clique le bouton droit et on relâche en dehors de la zone des menus), MENUNUM() vaut NOMENU.

On notera qu'Intuition envoie systématiquement un message MENUPICK lorsque l'utilisateur a enfoncé le bouton droit de la souris, et ce même si aucune sélection n'a été effectuée. D'où l'utilité de vérifier que le champ Code soit différent de MENUNULL. Le programme d'exemple fournit une fonction testmenu() qui utilise ce principe.

Il ne nous reste plus qu'à étudier quelques fonctions qui permettent de travailler dynamiquement sur les menus et nous aurons fait le tour de cet outil si pratique.

FONCTIONS UTILES

Nous avons entrevu la possibilité qui nous était donnée d'interdire ou d'autoriser l'accès à un menu, une option ou une sous-option en plein milieu de l'exécution du programme.

L'interdiction est réalisée à l'aide de la fonction OffMenu()
OffMenu(fenetre, numero)

- fenetre est un pointeur sur la fenêtre à laquelle est lié le menu.
- numero correspond au numéro de la fonction ou du menu à interdire (sous une forme équivalente au champ Code de la structure IntuiMessage de type MENUPICK que renvoie Intuition quand une fonction a été sélectionnée).

On le calcule avec la formule $\text{numero} = \text{numero_sousfonction} * 2048 + \text{numero_fonction} * 32 + \text{numero_menu}$. C'est tout simple et en plus ça fonctionne très bien.

Pour autoriser un menu à nouveau on utilise OnMenu().
OnMenu(fenetre, numero)

Enfin la dernière fonction
ItemAddress(premier_menu, numero)
calcule l'adresse d'une option ou sous-option à partir de l'adresse du premier menu et du numéro associé au menu à la fonction ou la sous-fonction à autoriser.

Enfin, lorsque l'on n'a plus besoin du menu, on l'élimine de la fenêtre grâce à la fonction

ClearMenuStrip(fenetre)
fenetre étant le pointeur sur la fenêtre à laquelle est lié le menu.

CONCLUSION

Il ne vous reste plus qu'à étudier le programme fourni comme exemple et qui se trouve bien entendu sur la disquette d'accompagnement de ce numéro. Il a été compilé sans problème avec le Lattice V 5.10 et l'Aztec V 5.0d

Le mois prochain, la gestion des sprites avec comme exemple quelques variations sur la gestion du pointeur de souris.

```

/*****
Les Menus sous Intuition programme réalisé par Herr Doktor Von
Glutenstimmelindorf (1991)
*****/
On commence par déclarer les includes
*****/
#include <stdio.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <exec/types.h>

#define INTUITION_REV 0L /* dernière révision d'intuition */

/*****
déclaration des variables externes
*****/
struct Window *OpenWindow();
struct Window *fenetre1 = NULL;
struct Window *fenetre2 = NULL;
struct IntuitionBase *IntuitionBase = NULL;

struct IntuiText itext1 = {
    3,1,COMPLEMENT, /* couleurs et mode d'écriture */
    0,0, /* position */
    NULL, /* fonte */
    "Close", /* pointeur vers le texte */
    NULL /* structure IntuiText suivante */
};

struct MenuItem menuitem2 = {
    NULL, /* structure MenuItem suivante */
    0,9,80,8 /* position et taille */
    ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP, /* flags */
    0, /* exclusion mutuelle */
    (APTR)&itext1, /* IntuiText */
    HIGHCOMP, /* intitulé en surbrillance */
    'c', /* séquence de touches */
    NULL, /* sous-menu */
    MENUNULL /* NextSelect */
};

struct IntuiText itext2 = {
    3,1,COMPLEMENT,
    0,0,
    NULL,
    "Open",
    NULL
};

struct MenuItem menuitem1 = {
    &menuitem2,
    0,0,80,8,
    ITEMTEXT+COMMSEQ+ITEMENABLED+HIGHCOMP,
    0,
    (APTR)&itext2,
    HIGHCOMP,
    'o',
    NULL,
    MENUNULL
};

struct Menu menu1 = {
    NULL, /* structure Menu suivante */
    0,0, /* position */
    111,0, /* largeur et hauteur */
    MENUENABLED, /* flags du menu */
    "Open Window", /* nom du menu */
    &menuitem1 /* première option du menu */
};

struct NewWindow nouvfenetre1 = {
    50,25,300,100,
    0,1,
    MENUPICK+CLOSEWINDOW+MENUVERIFY,
    WINDOWSIZE+WINDOWDRAG+WINDOWDEPTH+WINDOWCLOSE,
    NULL,
    NULL, /* Image pour le CHECKMARK, NULL = défaut */
    "Une fenetre avec menu",
    NULL,NULL,
    50,50,640,200,
    WBENCHSCREEN
};

struct NewWindow nouvfenetre2 = {
    350, 25, 150, 100,
    0, 1,
    NULL, NULL, NULL, NULL,
    "Fenetre",
    NULL, NULL,
    5,5, 640,200,
    WBENCHSCREEN
};

```



```

/*****
Programme principal (ne fait pas grand chose)
*****/
main()
{
    struct IntuiMessage *message;
    long GetMsg();
    ULONG classe_message;
    void init(), testmenu();

    init();
    fenetre1 = OpenWindow(&nouvfenetre1); /* on ouvre la fenetre */
    if (fenetre1 == NULL)
    { printf("la fenetre est bloquée");
      CloseLibrary(IntuitionBase);
      exit(FALSE);
    }

    SetMenuStrip(fenetre1, &menu1); /* On soumet les menus */

    while(1)
    { message = (struct IntuiMessage *)GetMsg(fenetre1->UserPort);
      if (message)
      { classe_message = message->Class;
        ReplyMsg(message);
        if (classe_message == CLOSEWINDOW)
        /* gadget fermeture de la fenetre actif */
        { if (fenetre2 != NULL) /* si la fenetre existe */
          CloseWindow(fenetre2);

          ClearMenuStrip(fenetre1);
          CloseWindow(fenetre1);
          CloseLibrary(IntuitionBase);
          exit(TRUE);
        }
        if (classe_message == MENUPICK)
        /* bouton droit enfonce */
        testmenu(message);
      }
    }

/*****
Fonction gerant les differents menus.
*****/
void testmenu(message)
{ struct IntuiMessage *message;
  USHORT num_suisant, numero, num_ssfc, num_fct, num_menu;

  if ((num_suisant=message->Code) != MENUNULL) /* aucune selection */
  { if (ITEMNUM(num_suisant) == 0) /* fonction Open */
    { num_ssfc = NOSUB;
      num_fct = 0;
      num_menu = 0;
      numero = num_ssfc*2048 + num_fct*32 + num_menu;
      fenetre2 = OpenWindow(&nouvfenetre2);
      OffMenu(fenetre1, numero); /* interdire Open */
    }

    if (ITEMNUM(num_suisant) == 1) /* fonction close */
    { num_ssfc = NOSUB;
      num_fct = 0;
      num_menu = 0;
      numero = num_ssfc*2048 + num_fct*32 + num_menu;

      if (fenetre2 != NULL) /* si la fenetre existe */
      { CloseWindow(fenetre2);
        fenetre2 = NULL;
      }
      OnMenu(fenetre1, numero); /* autoriser Open */
    }
  }

/*****
Fonction ouvrant la librairie intuition
*****/
void init()
{ char *OpenLibrary();

  IntuitionBase = (struct IntuitionBase *)
    OpenLibrary("intuition.library", INTUITION_REV);
  if (IntuitionBase == NULL)
  { printf("mais ou est donc passe intuition ");
    exit(FALSE);
  }
}

```

```

addq.l    #4, sp

***
if (tdIO)
{
    CloseDevice(tdIO);
    DeleteStdIO(tdIO);
}
.end3
move.l    tdIO(a5), d0
beq.s     .end4
movea.l   d0, a1
tst.l     IO_DEVICE(a1)
beq.s     .end3.1
CALLEXEC  CloseDevice
.end3.1
move.l    tdIO(a5), -(sp)
jsr       DeleteStdIO      ; dans amiga.lib
addq.l    #4, sp

***
exit();
.end4
movea.l   DOSBase(pc), a1
CALLEXEC  CloseLibrary

moveq     #0, d0
rts

; ****
; Ici, on se fabrique deux routines différentes :
; la première, Printf, utilise RawDoFmt() lorsqu'il y a
; des nombres à sortir. La seconde, Puts, imprime directement
; la chaîne de caractères telle que passée en paramètre.
Printf    moveq     #0, d7      ; RawDoFmt() n'utilise pas d7
          lea       Puts(pc), a2 ; alors nous, on s'en sert comme
          lea       bavard(a5), a3 ; compteur de caractères
          clr.b     (a3)+
          CALLEXEC  RawDoFmt
          movea.l   a3, a0
          move.b    d7, -(a0) ; et on tombe dans Puts

Puts      moveq     #0, d3
          move.b    (a0)+, d3
          move.l    a0, d2
          move.l    stdout(a5), d1
          CALLDOS   Write
          rts

getchar   move.l    stdin(a5), d1
          lea       bavard(a5), a0
          move.l    a0, d2
          moveq     #1, d3
          CALLDOS   Read
          rts

; ****
VARS      ds.b      VARSIZE

DOSBase:
          ds.l      1

dosname   dc.b      "dos.library", 0
          even
tdname    dc.b      "trackdisk.device", 0
          even

Err1      DEFSTR    <"Impossible d'allouer le buffer">
Err2      DEFSTR    <"Usage: Sectorise fichier secteur">
Err5      DEFSTR    <"Impossible d'ouvrir le trackdisk.device">
Err6      DEFSTR    <"Disquette protégée en écriture">
Err7.1    DEFSTR    <"Dépassement du secteur 1759">
Err7.2    DEFSTR    <"Plus de place sur la disquette">
Err8      dc.b      "Erreur trackdisk, code %d", 10, 0
Err3      dc.b      "Secteur de départ %d invalide", 10, 0
Err4      dc.b      "Impossible d'ouvrir le fichier %s", 10, 0

Msg1      DEFSTR    <"Placer la disquette dans DF0:">
Msg2      DEFSTR    <"Puis taper [RETURN]">
Msg3.1    dc.b      "Premier secteur : %d", 10, 0
Msg3.2    dc.b      "Dernier secteur : %d", 10, 0

; ****
END

```


Faut-il supprimer la disquette de l'ANT ? La question était posée dans l'édito de l'ANT numéro 23. A l'augmentation du nombre d'articles s'oppose celle du prix de l'abonnement. Quelques lecteurs concernés nous ont d'ores et déjà communiqué leur sentiment à ce sujet.

Cher journal, suite à l'interrogation par vous formulée dans votre numéro 23, je prends la plume pour vous donner mon avis sur la modification de la formule d'abonnement, que vous soumettez à la sagacité de vos humbles lecteurs...

J'ai personnellement choisi la formule "1 an sans disquette" pour disposer d'un magazine orienté programmation sur Amiga chaque mois et ne pas avoir à payer un surcoût inutile pour des disquettes qui, de toute façon, ne me serviraient pas, étant d'avantage intéressé par les techniques de programmation que les exemples d'application.

A priori, votre proposition équivaut, pour ceux qui ne désirent pas se procurer les disquettes, à transformer l'ANT en une revue de théorie technique et là, les débutants - dont je ne suis pas - devront sans doute déclarer forfait. Des articles sur la programmation sans programmes sont très complexes à exploiter pour quiconque n'est pas technicien. L'alternative consiste à accepter de payer également pour les disquettes : vous modifiez alors la nature du magazine en multipliant (par combien au juste ?) son prix.

Pour ma part, comme je ne tiens pas à acquérir des disquettes qui risquent d'être reformatées rapidement, pas plus que de recevoir un magazine au contenu nébuleux (je veux dire, sans les listings), il est vraisemblable que je ne renouvellerai pas mon abonnement si vous mettiez en place ce nouveau mode de fonctionnement.

C'était là mon opinion personnelle sur le sujet. Mais je laisserai à présent la parole aux autres abonnés qui ne manqueront pas d'exprimer la leur.

L. Depeux, Asnières/Seine

Cher ANT, je vous écris suite à la lecture de l'édito du numéro 23, que je viens à peine de recevoir. Si j'ai bien tout compris - car, comme d'autres avant moi, j'ai tout lu Freud... - il serait question de ne plus proposer que des abonnements avec disquette, gagnant ainsi la place utilisée dans le journal par les différents listings.

Cette idée me séduit : je suis pour l'instant abonné à la formule "5 numéros plus disquette" et je dois reconnaître que celle-ci fait un peu double emploi avec la revue. A part m'éviter un peu de fatigue et quelques inévitables fautes de frappe... Dans ma situation, le bénéfice serait évident : pour le même prix (enfin, j'espère...), je double mon volume d'articles intéressants.

Par contre, moi qui pensais changer d'abonnement pour passer à un an sans disquette, je vais sans doute réfreiner mes ardeurs... Le prix encore élevé ne m'autorise de tout façon pas l'année complète avec la disquette, donc... Je pense ne pas être le seul dans ce cas-là, n'est-ce pas ?

François Jacquet, Nantes

Voilà donc deux premiers avis clairs et nets. Il n'est évidemment pas question de les discuter ici. Simplement, nous attendons un nombre incalculable de lettres avant de prendre notre décision. N'hésitez donc pas à nous écrire, la balle est dans votre camp.

Chers amis, si je vous écris aujourd'hui, c'est pour faire un peu le point sur l'ANT après quatre numéros reçus. Dans un mois, mon

abonnement prendra fin et je me demande si je vais le renouveler. Non pas que l'ANT soit totalement inintéressant, bien au contraire, certains articles sont absolument passionnants (je pense notamment à l'Utilitaire de F. Mazué, qui nous en apprend toujours plus sur le système tout en ayant une utilité pratique). Je pense simplement que le rapport qualité-prix n'est pas forcément à la hauteur de ce qu'on pouvait espérer (le prix est très élevé !).

En effet, si quelques articles me semblent tout droit sortis des RKM (Subway I et II, CATS), je comprends parfaitement que tout le monde ne possède pas ces ouvrages, qui de plus sont en anglais. Le manque de documentation en français pour l'Amiga fait sans doute de l'ANT un magazine indispensable, mais ce n'est pas une raison pour tomber dans le piège de la francisation des documentations étrangères déjà existantes ! Vous disposez d'assez de personnes compétentes pour vous permettre de faire preuve d'un peu plus d'originalité...

Mais le plus gros défaut que je puisse trouver à l'ANT est sans aucun doute la piètre qualité de la réalisation. Encore une fois, au prix du numéro, on est en droit d'attendre un magazine dont la maquette soit nickel et qui soit exempt de fautes d'orthographe... Or le numéro 23 est un véritable cauchemard : colonnes apparentes, nombreuses coquilles... Même votre concurrent Amiga-News, pourtant réputé - lui aussi - pour la toute relative qualité de sa mise en page était plus attrayant !

Jean-Pierre Dumest, Paris

La rubrique CATS seule est directement extraite et traduite des Rom Kernal Manuals. Subway, dont le but est d'initier le programmeur débutant aux joies de l'Amiga, s'en inspire sans doute fortement, et c'est inévitable, mais vous exagérez un peu en qualifiant cette rubrique de "francisation des documentations étrangères existantes". Prenons le cas de la série sur Intuition de notre Docteur national, et plus particulièrement le numéro de ce mois-ci, concernant les menus : comment voudriez-vous qu'il puisse décrire chaque structure, chaque flag et chaque fonction sans ressembler à ce qui a déjà été imprimé ailleurs ? Bien sûr, vous pourrez toujours rétorquer, non sans un fond de pertinence, qu'il est toujours mieux et en tout cas plus instructif de mettre en pratique, plutôt que d'étaler inlassablement une mare de connaissances dans laquelle le lecteur ne pourra au bout du compte que se noyer. Soit, mais nous préférons quant à nous penser que c'est au lecteur de faire la part des choses et surtout le tri de ce qui l'intéresse. Ensuite, quand il aura compris les bases indispensables, il pourra, s'il en a envie, aller plus loin et mettre en pratique lui-même, en s'aidant au besoin des programmes publiés dans les autres rubriques ou dans les autres revues. Notre rôle n'est pas de vous prendre par la main mais de vous mettre sur le bon chemin.

Quant aux "imperfections" que vous mentionnez, elles existent, c'est vrai. Des coquilles arrivent à passer, malgré la vigilance du correcteur. La maquette a parfois des problèmes, c'est vrai aussi. Nous ne pouvons que nous en excuser et faire de notre mieux pour que cela ne se reproduise plus. C'est ce qu'on fait.

Cher ANT, n'ayant pas de Minitel le numéro de téléphone où appeler pour me faire aider, je vous pose ces quelques questions :

- le Pascal (n°22) ne comprend pas l'instruction ScreenPtr ("Unknown ID"). Ceci fait planter des exemples de la disquette (Bézier, 3D...). Or, j'ai bien tous les fichiers que vous mentionnez dans l'article. Si vous pouviez m'expliquer ?

- ma disquette qui contient le compilateur C de Matt Dillon (n°21) est pleine alors que les Includes n'y sont pas (je cherche toujours l'amiga.lib). Et si je fais une erreur, ou bien... ?

Pour terminer, une remarque sur la disquette de l'ANT : le Menu se bloque à la fin. Résultat : bootage obligé. Enfin, vous vous améliorez en numéro en numéro. J'attends ma demande de renouvellement avec impatience.



Emmanuel Melero, Saint Herblain

PCQ comprend parfaitement le type (et non l'instruction) ScreenPtr, pour peu que vous incluiez également lors de la compilation le fichier Screen.I qui se trouve dans votre répertoire PCQ:Include. Pour ne je ne sais quelle raison, l'auteur, Patrick Quaid, a oublié de le faire dans ses exemples. Ajoutez donc la ligne :

```
{ $I "Include/Screen.i" }
```

à vous sources pour que cela fonctionne.

Quant à DICE, tous les fichiers ne sont pas nécessaire sur la disquette : les sources des fonctions de la bibliothèque C peuvent être conservés ailleurs que la disquette du compilateur. Les seuls répertoires nécessaires sur la disquette DICE: sont c, bin, dlib et include. Ils occupent environ 400 Ko, soit moins de la moitié de la disquette.

Par l'intermédiaire du journal Tilt, Infogrames a diffusé une démo du jeu Mystical. J'ai constaté que la Startup-Sequence commence par un petit programme appelé "reset" (cf. listing ci-joint, obtenu bien sûr avec Devpac II...) qui effectue un NoFastMem résistant au Reset clavier ! Pour remettre en service l'extension RAM, il faut éteindre et rallumer l'Amiga !

Comme je trouvais cela pas très pratique, ni bénéfique pour les circuits de l'Amiga, j'ai potassé un peu la routine Reset du système (dans La Bible... Hé oui !). J'en ai conclu qu'en "faussant" ChkBase, on oblige à un reset plus "profond" qui reprend en compte l'extension RAM. J'ai donc modifié un petit poil la routine Clean_Reset de Frédéric Mazué, donnée dans l'ANT n°17 :

```
Clean_Reset:
    clr.l ChkBase(a6) ; a6 = ExecBase après Supervisor()
    lea.l $2,a0
    RESET
    jmp (a0)
```

Et ça marche ! Cela intéressera sûrement les lecteurs ayant eu le problème.

Maintenant, une question : parlerez-vous dans l'ANT du débogueur ROM-Wack inclus dans la ROM KickStart, de ses commandes et de la manière de s'en servir à partir d'un terminal ?

En tout cas, bravo et merci pour l'ANT.

G. Boulay, Paucourt

```
; Programme "RESET" de la démo de Mystical
; Débraye l'extension RAM: d'une manière qui
; résiste au reset CTRL+A+A
MOVEA.L 4,A6
MOVE.L $4E(A6),D0 ; $4E = MaxExtMem
BEQ fin
MOVE.B #3,$BFE201
MOVE.B #2,$BFE001
LEA $DFF000,A4
MOVE.W #$7FFF,D0
MOVE.W D0,$9A(A4)
MOVE.W D0,$9C(A4)
MOVE.W D0,$96(A4)
MOVE.W #$200,$100(A4)
MOVE.W #0,$110(A4)
MOVE.W #$444,$180(A4)
LEA $400,A6
SUBA.W #$FD8A,A6
LEA $C00000,A0
LEA $DC0000,A1
MOVEA.L #0,A4 ; Pas d'extention !
MOVE.L A4,D0
MOVEA.L #$40000,A7
MOVE.L $FFFFFF,D6
JMP $FC0208 ; Reset en ROM (beurk !!)
fin: RTS
```

A la lumière de ce listing, on peut dire que les programmeurs de chez Infogrames ne se cassent pas vraiment la tête ! Non seulement cette pratique est indigne de l'Amiga (et en plus y'a un saut en ROM ! La honte sur eux !), mais ils ne se soucient même pas de savoir si l'extension est de type FAST ou CHIP, auquel cas ils pourraient l'exploiter ! Merci donc pour ton reset à toi, beaucoup plus propre.

Quant à ROM-Wack, il fonctionne avec un terminal série configuré à 9600 bauds, 8 bits de données, pas de parité, 1 bit de stop. Sans trop connaître la question, je pense que le Minitel (au moins le M2) pourrait parfaitement convenir. S'il s'avère que c'est le cas, nous en parlerons certainement dans une prochaine rubrique ExecBase.

Bonjour à tous ! Le but de cette lettre n'est pas seulement de poser des questions, auxquelles je suis sûr que vous répondriez de toute façon, mais plutôt de vous soumettre quelques suggestions d'articles pour les numéros à venir.

- comment se fabriquer une librairie personnelle, contenant toutes mes routines les plus fréquemment utilisées, que mes programmes n'auraient plus qu'à ouvrir quand ils en auront besoin ? Cela réduirait leur taille de façon conséquente et me permettrait d'améliorer certaines routines sans avoir à ré-assembler chaque programme qui les utilise. La documentation que l'on peut trouver dans la Bible de l'Amiga est loin d'être suffisante... De plus, vues les similitudes entre les deux, cet article pourrait déboucher sur un autre traitant de la création d'un device particulier.

- en parlant des devices, je possède une imprimante DMP 2160 d'Amstrad, dont j'aimerais exploiter toutes les possibilités avec mon Amiga, ce que le driver d'imprimante EpsonFx ne me permet pas. Un article traitant donc de la création d'un driver particulier me semblerait fort utile. Dans le même genre d'idées, n'y a-t-il pas quelque chose à faire avec les keymaps ?

- pourquoi ne parlez-vous plus de l'AmigaBasic ni du GfABasic dans l'ANT ? Ces deux langages sont pourtant souvent utilisés, surtout le GfA. D'autant plus que son compilateur est sorti et que vous en avez vous même fait l'éloge !

Bien sûr, ces quelques suggestions sont en rapport direct avec les problèmes que je rencontre à la maison. Toutefois, je suis certain que je ne suis pas le seul et que d'autres lecteurs seraient ravis de trouver ce genre d'articles dans l'ANT.

Jérôme Apietto, Savigny/Orge

Une série d'articles sur les bibliothèques ("library" est le terme anglais) est justement en préparation. Le premier épisode devrait être - j'espère - publié dans le numéro 26 ou 27. Il n'était pas prévu d'y inclure les devices, mais si cela intéresse suffisamment de lecteurs, nous y viendrons.

Un driver d'imprimante ? L'idée n'est pas mauvaise, mais le problème est d'en choisir un qui puisse être utile à la majorité des lecteurs. Beaucoup semblent avoir conservé leur DMP en passant à l'Amiga, alors pourquoi pas ?

L'AmigaBasic n'est plus réellement utilisé, sauf peut-être par quelques rares masos, et encore... Quant au GfA, s'il s'avère que la sortie du compilateur a relancé l'intérêt de ce langage, nous y reviendrons sûrement. Encore une fois, il n'est pas possible de contenter toute le monde, et nous devons nous conformer à l'avis de la majorité. Or il ne me semble pas que la majorité des lecteurs de l'ANT soit attirée par ce langage. Aujourd'hui, C et assembleur sont les maîtres absolus, et le Basic n'a pas beaucoup d'avenir sur notre machine.



CATS : QUESTION DE STYLE (II)

Voici la fin des recommandations du Commodore-Amiga Technical Support, sur la manière de réaliser des programmes ayant le "look-and-feel" Amiga.

Utilisez les deux touches Amiga comme des touches Shift supplémentaires. La touche Amiga de gauche est utilisée pour les raccourcis clavier et par Intuition, celle de droite, pour sélectionner des options de menus sans toucher à la souris. Le tableau-ci dessous liste quelques recommandations pour l'utilisation de la touche Amiga de gauche :

N	Place l'écran du WorkBench devant tous les autres (géré automatiquement par Intuition)
:	Place l'écran du WorkBench derrière tous les autres (géré par Intuition)
B	Répond "oui" à un AutoRequester
V	Répond "non" à un AutoRequester
I	sélectionne une petite partie du document, à droite du curseur (ex : le mot suivant)
O	sélectionne une partie du document, à droite du curseur (ex : la prochaine ligne)
P	sélectionne une grande partie du document, à droite du curseur (ex : le prochain paragraphe)
J	sélectionne une petite partie du document, à gauche du curseur (ex : le mot précédent)
K	sélectionne une partie du document, à gauche du curseur (ex : la ligne précédente)
L	sélectionne une grande partie du document, à gauche du curseur (ex : le paragraphe précédent)

Le tableau suivant liste les raccourcis standard à utiliser avec la touche Amiga droite. Les programmes devraient proposer autant de raccourcis que possible : l'utilisateur confirmé apprend vite à se servir du clavier, tandis que le novice continue à utiliser la souris et les menus.

X	Couper (cut)
C	Copier (copy)
V	Coller (paste)
I	Italiques (italics)
B	Gras (bold)
U	Souligné (underline)
P	Normal (plain)
Z	Undo
O	Ouvrir (open)
S	Sauver (save)
Q	Quitter (quit)

LA TOUCHE HELP

Prenez en compte la touche Help. Si votre application dispose d'une aide en ligne intégrée, la touche Help est un bon moyen de l'obtenir. Imaginez la tête de l'utilisateur perdu, qui appuie sur Help alors que rien ne se passe...

LE CURSEUR

Les touches fléchées sont un moyen pratique de contrôler les mouvements du curseur. Voici nos recommandations quant à leur utilisation :

Flèche et...	Déplacement
(rien)	l'unité la plus petite (un caractère ou une ligne)
ALT	l'unité immédiatement supérieure (un mot ou une page)
SHIFT	la plus grande unité (fin de ligne, début de document)
CTRL	Spécial ou sans effet

LA SOURIS

Intuition utilise le bouton gauche de la souris pour la sélection et le bouton droit, pour le transfert d'informations. Nous vous encourageons à suivre ce modèle, pour aider l'utilisateur à comprendre le fonctionnement de chaque application, et à s'en souvenir. Quand l'utilisateur appuie sur le bouton gauche, Intuition examine l'état du système et la position du pointeur. Ces informations sont utilisées pour décider si l'utilisateur essaye de sélectionner un objet, une opération ou une option. Par exemple, l'utilisateur positionne le pointeur sur un gadget et appuie sur le bouton gauche pour le sélectionner. S'il déplace la souris tout en maintenant le bouton appuyé, cela signifie sans doute qu'il désire déplacer ce gadget.

Intuition utilise le bouton droit pour les opérations de menu. Une pression sur le bouton droit affiche normalement la barre de menu. Si l'utilisateur déplace la souris tout en maintenant le bouton droit appuyé, il est sans doute en train de se promener dans le menu, à la recherche d'une option particulière. Un double clic du bouton droit peut faire apparaître un Requester spécial, que l'on peut considérer comme une extension aux menus. Ce Requester étant destiné à recevoir des informations, il est correct de l'assigner au bouton droit.

Si vous décidez de gérer vous-même les événements en provenance de la souris, conformez-vous le plus possible à ce modèle.

ECRAN ET FENETRE

Permettez toujours à l'utilisateur d'avoir accès à la barre de l'écran (ce peut être aussi simple que d'ouvrir votre fenêtre un pixel en dessous du haut de l'écran). Evitez d'ouvrir des fenêtres plein-écran qu'on ne peut déplacer ni changer de taille, surtout si vous utilisez l'écran du Workbench.

De même, si votre programme ne dépend pas particulièrement de la taille de sa fenêtre, fixez ses limites MaxWidth et MaxHeight à -1 : l'écran utilisé peut être plus grand que le 640 x 256 standard.

Quand vous ouvrez un CustomScreen, redirigez-y les Requester d'AmigaDOS (champ pr_WindowPtr de la structure Process), sinon Intuition amènera l'écran du Workbench devant tous les autres. Vous devez restaurer le pointeur avant de fermer la fenêtre.

DIVERS

Un programme devrait toujours appeler OpenWorkbench() avant de quitter, même s'il n'a pas appelé CloseWorkbench() auparavant : si un autre programme a fermé le Workbench et que la fin du votre a libéré suffisamment de mémoire pour lui permettre de se rouvrir, il est préférable qu'il le fasse. Le Workbench devrait être ouvert le plus souvent possible, cela aide l'utilisateur à garder un environnement consistant. Intuition vérifie toujours, lorsqu'un programme ferme un écran, que ce n'était pas le dernier, auquel cas celui du Workbench doit être ré-ouvert.

Autorisez tant que fait se peut, l'utilisateur à configurer les paramètres de votre programme. Par exemple, si vous ouvrez un CustomScreen, laissez-lui la possibilité d'en changer les couleurs. Si votre logiciel produit du son, permettez-lui d'en modifier le ton et le volume. Ne rendez cependant pas la indispensable, et laissez toujours une possibilité de revenir aux réglages par défaut.

COULEURS

Intuition utilise les couleurs suivantes pour le pointeur de la souris (entre parenthèses, le numéro du registre de couleur hardware correspondant) :

0	transparente
1 (17)	intensité moyenne
2 (18)	basse intensité
3 (19)	haute intensité

Entourez votre pointeur personnel d'un filet de couleur 1 ou 3.

Le pointeur de souris étant toujours le sprite 0, vous pouvez en changer les couleurs en appelant SetRGB4() avec le ViewPort de n'importe quel écran, même s'il n'a que deux couleurs. Par exemple :

```
struct Screen *myScreen;
```

```
SetRGV4(&myScreen->ViewPort, 17, R17, V17, B17);  
SetRGV4(&myScreen->ViewPort, 18, R18, V18, B18);  
SetRGV4(&myScreen->ViewPort, 19, R19, V19, B19);
```

ICONES

Vos icônes devraient être dessinées avec une source lumineuse venant du coin haut/gauche de l'écran, et être visibles avec 1 ou 2 plans de bits. Dessinez de belles icônes, qui reflètent l'utilité de votre programme, mais surtout qui ne soient pas trop grandes : cela prend de la place à l'écran, en mémoire et sur disquette. Ayez une approche minimaliste quand vous dessinez vos icônes.

(c) 1987, 1991 Commodore Amiga Technical Support