

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT

ExecBase : Le C ANSI	2	Denis Jarril
AMOS : La nouvelle extension musicale	4	François Lionet
ARexx : Travail pratique (III)	6	François Gueugnon
Pascal : Présentations	8	Olivier Garrigues
GFA Basic : Le timer.device	10	Pierre Philippe Launay
Algos : Les listes chaînées	12	Lucien Turski et Bruno Bailluet
Hardware : La voix de Paula	14	Philippe Rivaillon
Transactor : Les Plasmas	16	Thomas Landspurg
ToolBox : Les créateurs d'images	18	Herr Doktor von Glupa
Demos : Le RollerText	20	Jérôme Etienne
Utilitaire : BigCLI	24	Max
Routinettes : GetDir	27	Tonton François Lionet
SubWay : Les Bobs	28	Herr Doktor von Glupa

Le pont de
San Francisco
dans le
brouillard.



Amiga NewsTech est une publication de Commodore Revue SARL 5, rue de la Fidélité, 75010 PARIS.

Ce numéro de l'Amiga NewsTech est exceptionnel à plus d'un titre.

Exceptionnel, parce chaque numéro d'Amiga NewsTech est déjà exceptionnel en lui-même, c'est pas à vous que je vais l'apprendre.

Exceptionnel, parce qu'il marque la première de plusieurs nouvelles rubriques : Pascal, GFA, Algos... Bon, vous connaissez le refrain : vous pouvez toujours nous faire part de vos remarques et suggestions sur ces rubriques. Et soyez sans pitié, si telle ou telle vous semble superflue, dites-le nous franchement, on la verra ! Bien sûr, vous avez également le droit de complimenter.

Exceptionnel, parce qu'il marque l'arrivée de six nouveaux venus dans notre grande équipe rédactionnelle : Bruno Bailluet, Olivier Garrigues, Thomas Landspurg, Pierre Philippe Launay, Philippe Rivailon et Lucien Turski-Marcon. Bienvenue à eux !

Exceptionnel, parce qu'il est le premier numéro d'une nouvelle année qui s'annonce très riche. Vous en saurez plus le moment venu, je ne veux pas gâcher la surprise.

Exceptionnel, parce que grâce au groupe Poltergeist et notamment à son coder Enzo, la parodie d'accompagnement est maintenant parée d'une AntIntro superbe !

Exceptionnel, enfin, parce que bon, j'ai décidé qu'il serait exceptionnel. Non mais, c'est qui le chef, ici ?

Bonne année !

Stéphane Schreiber

OURS

Bienvenue dans l'Ours de l'Amiga NewsTech !

Amiga NewsTech est vendu chaque mois et uniquement sur abonnement par Commodore Revue SARL, 5, rue de la Fidélité, 75010 Paris, 42.46.92.90.

Directeur de la publication : Jean-Yves Primas.

Red'chef : Stéphane Schreiber.

Ont collaboré : Pascal Amiable, Bruno Bailluet, Jérôme Etienne, Loïc Far, Olivier Garrigues, François Guegnon, Denis Jarril, Thomas Landspurg, Pierre Philippe Launay, François Lionet, Max, Philippe Rivailon, Stéphane Schreiber, Lucien Turski-Marcon.

PAO : Stéphane Schreiber.

Support orthographique : Christine (et) Robert.

Répondeur téléphonique : Valérie (modèle agréé).

Piqueuse de clopes : Anne (avec l'aide de Pop).

Ménach' : Mme Maria.

Encouragements : la voisine d'en face qui n'a pas de rideaux à la fenêtre de sa salle de bain...

Special thanks to : Yvounet, Gaëllounette, Elodine, Bibounet, String, Poum, Sined, Idéfix, Roger Zabel.

Flashage : ColorWay, 5, rue de la Fidélité, 75010 Paris, 47.70.87.87 (désolé, j'avais pas la place pour le logo).

Impression : INO Montrouge, 9, rue Auber, 92120 Montrouge, 46.56.27.69 (désolé, j'avais pas de logo).

Abonnements : MCM Ant, 16, Quai Jean-Baptiste Clément, 94140 Afortville, 43.78.92.10.

Ce numéro 29 de l'Amiga NewsTech est dédié à tous ceux qui ont contribué, directement ou indirectement, à mon épanouissement culturel ces cinquante dernières années, ainsi qu'à l'inventeur du treizième mois.

Ce numéro 29 de l'Amiga NewsTech n'est pas dédié à tous ceux qui ne se sont pas abonnés, ni à ceux qui ne connaissent même pas l'ANT.

Ce numéro 29 de l'Amiga NewsTech aurait pu être dédié à tous mes amis, ceux qui j'aime et qui me le rendent bien. Mais il ne l'est pas.

EXECBASE : LE C ANSI

Nous débutons ici une nouvelle série à l'intérieur de la série... Au menu, les spécifications de la norme ANSI pour le langage C, avec comme plat de résistance les fonctions de la bibliothèque standard.

ANSI est l'abréviation de *American National Standards Institute*, ce qui signifie en français et à peu de choses près, Institut National Américain de Standardisation (INAS). Mais bon, ANSI, c'est tout de même plus simple. Le but de ce vénérable établissement est tout simplement et comme son nom l'indique, de fournir au bon peuple américain des standards pour tout ce qu'il est possible de standardiser. Et Dieu sait que rien qu'en informatique, ces gens-là ont bien du boulot...

On ne reviendra pas particulièrement sur le rôle et l'utilité d'un standard, qu'il s'agisse de machines à laver ou de contrôleurs de disques optiques réinscriptibles à triple faisceau laser orthogonalement entrelacé. Les travaux informatiques de l'ANSI sont mondialement connus, ne serait-ce que concernant les codes de contrôles des consoles d'ordinateurs, codes auxquels l'Amiga a d'ailleurs totalement adhéré. Ainsi, un fichier texte ASCII contenant quelques codes ANSI de mise en valeur (gras, italique, etc.) sera visualisé exactement de la même manière sur un Amiga que sur un PC équipé du driver *ANSI.SYS*. Sauf que ce sera plus joli sur l'Amiga, paske bon, quand même, hein, faut pas déconner.

L'un des travaux les plus importants de l'ANSI a surtout été de fournir une standardisation pour presque tout langage informatique évolué, de façon à le rendre indépendant de la machine. C'est ainsi qu'il existe une norme ANSI pour le Basic, pour le Pascal, le Lisp... Et bien entendu le C, qui nous intéresse plus particulièrement aujourd'hui.

K&R vs ANSI

Les créateurs du langage C, messieurs Kernighan et Ritchie, avaient déjà entrepris un effort de standardisation de leur bébé, que l'on a tout naturellement baptisé "norme K&R" et qu'ils ont exposé dans leur livre *"Le Langage C"* (1). Seulement voilà, deux hommes seuls, et malgré toute la bonne volonté dont ils ont pu faire preuve, ne sauraient penser à tout. Aussi les ANSI-mens se sont-ils penchés sur le problème et ont étendu cette première norme, tout en essayant d'en rester le plus près possible (le problème étant bien entendu de rester compatible). Résultat, la norme ANSI, par ailleurs elle aussi décrite dans un excellent livre *"Le Langage C, 2ème édition"* (2) signé... Kernighan et Ritchie.

Par la suite, nous ne parlerons plus de la norme K&R, qui n'est finalement plus qu'un sous-ensemble de la norme ANSI. Pour la petite histoire, on peut signaler que la norme K&R date de 1978, cependant que la norme ANSI date, elle, de 1983, et qu'elle fut définitivement acceptée fin 1988.

PRINCIPES

Aujourd'hui, on trouve de ça et de là divers compilateurs C, dont beaucoup se réclament compatibles ANSI. L'utilisation d'un tel compilateur ANSI sur une machine donnée, garantira que le code source sera directement accepté par un autre compilateur ANSI sur une autre machine, disposant d'un autre système d'exploitation.

Cela suppose donc qu'un certain nombre de fonctions, notamment d'entrées-sorties, soit effectivement disponibles dans les deux environnements. Comble de bonheur, ces fonctions sont toutes regroupées dans une bibliothèque (en anglais, *library*) qui se doit d'être standard. Peu importe donc pour le programmeur C la manière dont ces fonctions ont été effectivement implémentées sur telle ou telle machine, il est sûr de les retrouver quelque soit l'environnement et surtout, il est sûr que le nombre et la signification des paramètres de chacune sont identiques. C'est ça, la standardisation.

LIMITES

Pour être indépendante de la machine, la norme doit faire quelques sacrifices.

Premier exemple, très parlant : l'ouverture d'une fenêtre sur Amiga s'effectue par un appel de la fonction *OpenWindow()* de l'*intuition.library*. Bien. MS-DOS ne disposant pas de système de fenêtrage, un programme Amiga utilisant *OpenWindow()* ne sera pas directement portable en MS-DOS. Quant aux MacIntosh, Archimedes, Atari ST et autre NeXT, même s'ils disposent de fenêtres



graphiques, ils n'ont pas d'intuition.library, donc pas d'OpenWindow(), donc notre programme ne sera pas non plus portable sur ces machines. Ce qui explique sans aucun doute possible que la fonction OpenWindow() ne fasse pas partie de la norme ANSI.

Autre exemple, moins évident : supposons que l'on désire ouvrir un fichier à l'aide de la fonction ANSI appropriée. Assumons pour les besoins de l'exemple que cette fois-ci, la portabilité est totale, et que le source soit compilé sans aucune erreur particulière quelque soit le système. Des problèmes risquent encore de surgir à l'exécution du programme : AmigaDOS autorise des noms de fichiers d'une longueur de 30 caractères maximum, et le point décimal n'est pas considéré comme un séparateur particulier. Cette règle est également vraie sur Mac, mais pas en MS-DOS, où seuls 8 caractères maximum sont autorisés, plus le point, plus 3 autres caractères pour l'extension.

Ce second exemple démontre clairement que toute la standardisation du monde ne saurait suffir à assurer une portabilité parfaite. Le programmeur doit également faire attention à ne pas utiliser, directement ou indirectement, telle ou telle particularité d'un système donné, sous peine de problèmes sérieux. En cas de conflit, comme dans l'exemple précédent, il devra se conformer aux exigences du système le plus contraignant (ici, MS-DOS) ou modifier explicitement chacune des versions de son oeuvre en conséquence - mais alors, à quoi donc sert une norme ?

PROTOTYPES

Encore une petite digression avant de parler des fonctions elles-mêmes. L'un des principaux apports de la norme ANSI pour le langage C est la possibilité de déclarer des *prototypes* pour les fonctions que le programme emploie.

Le raisonnement est simple : avant, la norme K&R indiquait qu'une déclaration de fonction ne spécifiait que le type de la fonction, c'est-à-dire et plus exactement, le type du résultat que cette fonction renvoyait. Par exemple :

```
int Puissance(); /* Déclaration de fonction, façon K&R */
```

Le nombre et le type des paramètres qu'elle exigeait n'était connu du compilateur qu'au moment-même de la compilation de la fonction. De plus, une fonction non-déclarée avant son utilisation était automatiquement considérée comme renvoyant un entier (*int*).

Depuis la norme ANSI, une déclaration de fonction peut également indiquer le nombre et le type de ses paramètres. Ainsi, le compilateur peut vérifier dès l'appel de la fonction que le programmeur n'a pas commis d'erreur. Pour reprendre l'exemple ci-dessus, la déclaration de notre fonction Puissance() devient :

```
int Puissance(int, int); /* La même, façon ANSI */
```

Il est même possible de donner également les noms des paramètres. Cela n'ajoute rien aux vérifications que le compilateur peut entreprendre, mais peut rendre le listing plus clair et plus facilement compréhensible par une tierce personne :

```
int Puissance(int base, int n); /* encore mieux ! */
```

L'ancienne forme est toujours valable, et tout compilateur se doit de l'accepter. Cela dit, les avantages de la seconde forme sont évidents, et je vous encourage plus que vivement à l'utiliser chaque fois que vous le pourrez.

On pourrait continuer à disserter pendant des heures sur toutes les modifications, bonnes et mauvaises, que la norme ANSI a apporté à la norme K&R, mais ce n'est pas vraiment là le but de cet article...

LA BIBLIOTHEQUE

Nous reviendrons plus en détails dans les mois à venir sur les pièges à éviter si l'on veut que son logiciel soit portable. Pour l'heure, attachons-nous à découvrir quelles fonctions l'on trouve dans la bibliothèque standard ANSI, ainsi que leur utilité.

Note : pour le compilateur SAS/Lattice C 5.10, cette bibliothèque porte le nom évocateur de *lc.lib*, ou *lcs.lib* pour le modèle "small" (entiers 16 bits), ou *lcr.lib* pour la version ré-entrante, ou encore *lcm.lib* pour la version avec nombres en virgule flottante... Evidemment, seules les fonctions effectivement utilisées seront placées par le linker dans le programme exécutable final. En Manx/Aztec C, je sais pas, vérifiez avec votre documentation.

La bibliothèque ne fait pas partie du langage C proprement dit. Cependant, tout compilateur C ANSI fournira les déclarations de fonctions, de macros et de types de cette bibliothèque. Elle se base sur plusieurs fichiers d'en-tête (header files), eux aussi standards :

```
<assert.h> <float.h> <math.h> <stdarg.h> <stdlib.h>
<ctype.h> <limits.h> <setjmp.h> <stddef.h> <string.h>
<errno.h> <signal.h> <stdio.h> <time.h>
```

Un ou plusieurs de ces fichiers seront bien sûr inclus au source principal par la directive *#include* du pré-processeur. Ils peuvent être inclus plusieurs fois et dans n'importe quel ordre sans que cela ne provoque d'erreur, mais quoiqu'il arrive, et c'est chose normale en C, l'inclusion doit précéder toute utilisation des éléments qu'ils contiennent.

Leur rôle est donc de définir les prototypes des fonctions de la bibliothèques (*printf()* et consœurs), ainsi que de fournir au programmeur des constantes (la plus célèbre étant *NULL*), des macros (*max()* par exemple) et des types (*size_t* par autre exemple), utiles pour son travail de développement. Ils contiennent également des références aux variables globales de la bibliothèque, telle que *errno* pour ne citer qu'elle, qui contient le code d'erreur éventuel de la dernière fonction appelée. Encore une fois et au risque de me répéter, toutes ces fonctions, constantes et macros sont (censées être) disponibles sur tous les systèmes de France, de Navarre et des environs.

La multiplicité des fichiers d'en-tête s'explique tout simplement par une envie de regrouper par thèmes toutes les déclarations. Ainsi, *<stdio.h>* déclare les fonctions d'entrées-sorties, *<string.h>*, celles destinées à manipuler les chaînes de caractères, et *<time.h>* celles de gestion de l'heure système. C'est simple, pratique et économique en temps de compilation : si on n'a pas besoin de nombres flottants dans un programme, on n'inclut pas *<math.h>* et c'est toujours ça de gagné.

Bon, ben la place qui m'est impartie étant (bien) remplie, je me vois dans l'obligation de paraphraser mon idole de toujours, Lucien Jeunesse : "chers amis bonsoir, et au mois prochain, si vous le voulez bien !".

(1) *Le Langage C*, par B.W. Kernighan et D.M. Ritchie, 1978, Prentice Hall. Edition française par Masson.

(2) *Le Langage C*, 2ème Edition : le C ANSI, par B.W. Kernighan et D.M. Ritchie, 1988, Prentice Hall. Edition française par Masson.

A noter l'excellent livre "Le Langage C, Solutions" par C.L. Tondo et S.E. Gimpel, chez Masson, qui contient toutes les solutions aux exercices proposés dans "Le Langage C", première et seconde éditions.

Par Denis Jarril



AMOS : LA NOUVELLE EXTENSION MUSI

Une fois n'est pas coutume, je ne vais pas vous donner de listing aujourd'hui, mais plutôt vous parler des nouvelles routines musicales disponibles pour AMOS.

Vous trouverez la nouvelle extension musicale sur la disquette de l'Amiga NewsTech de ce mois-ci. Elle sera également mise en téléchargement sur les deux serveurs de la Rédaction, j'ai nommé le 3615 ANT et le 3615 ComRev. J'y ai fourré bien sûr le programme de l'extension, mais aussi le source assembleur ainsi que l'exemple.

INSTALLATION

Bon, vous prenez votre Basic favori (AmigaBasic... heu, non, pardon, AMOS), vous chargez le programme Musique_Install.AMOS : la prochaine fois que vous chargerez l'interpréteur, les nouvelles instructions seront disponibles.

QUOI DE NEUF ?

Plein de choses.

- **Les longs samples** : j'ai totalement modifié la gestion des samples, ils utilisent maintenant les interruptions de fin de son. La conséquence est que l'on peut jouer des samples de plus de 64 Ko, en fait sans aucune limitation de taille (sauf la mémoire bien sûr).

- **Le double-buffer sonore** : de toutes nouvelles instructions permettent de jouer des samples aussi longs que vous le désirez, en utilisant très peu de mémoire. La méthode utilisée est celle du double-buffer. Comme pour les bobs, on va travailler dans deux espaces mémoire distincts : l'un est joué sur le haut-parleur pendant que l'on charge la suite des sons dans l'autre.

Avant de voir comment programmer un tel système, on passe en revue les nouvelles instructions.

SLOAD Fichier TO Adresse, Longueur

SLOAD est mis pour "Sample LOAD". Il s'agit d'un BLOAD amélioré.

- Fichier est le numéro d'un fichier déjà ouvert par OPEN IN.
- Adresse est l'adresse de chargement.
- Longueur est le nombre d'octets à charger. Si vous spécifiez une longueur supérieure à celle restant à charger, AMOS ne produira pas d'erreur et chargera simplement le nombre correct d'octets.

Cette instruction est conçue pour être utilisée conjointement aux fonctions de gestion des fichiers séquentiels =POF(), =LOF(), =EOF(). Vous pourrez l'utiliser pour faire bien plus de choses que simplement charger des samples...

SAM SWAP Canaux TO Adresse_Buffer, Longueur

Cette instruction permet de réaliser l'échange des buffers, et n'a d'action qu'une fois que le buffer actuel est terminé.

- Canaux est un masque de bits représentant les voix affectées par l'instruction (comme pour SAM PLAY par exemple).
- Adresse_Buffer est l'adresse à laquelle doit se poursuivre la lecture du son. La jonction entre les deux buffers est faite sous interruption et est donc totalement inaudible.

- Longueur est le nombre d'octets sonores à lire dans le nouveau buffer.

=SAM SWAPPED(Voix)

Cette fonction permet de se synchroniser avec l'échange des buffers. C'est elle qui indique quand charger une nouvelle partie du sample.

- Voix est un nombre de 0 à 3.

Elle peut retourner trois valeurs différentes :

0 : les circuits n'ont pas terminé la lecture du buffer actuel. Il faut encore attendre.

-1 : ça y est, on peut y aller...

+1 : on a attendu trop longtemps, le son est coupé. Il faut alors réinitialiser totalement la voix.

SAM STOP [Canaux]

Une instruction qui manquait cruellement : il était quasiment impossible d'arrêter un sample, une fois lancé !

- Canaux est un masque de bits, comme pour SAM SWAP.

UN EXEMPLE

... valant mieux qu'un long discours, en voici un petit permettant de jouer un sample de longueur indéfinie, en n'utilisant que 16 Ko de mémoire CHIP ! Voyez le listing en fin d'article.

LES INSTRUCTIONS TRACK

J'ai prévu les routines sonores de l'AMOS pour être aussi générales que possible. En théorie, il est possible de jouer n'importe-quelle musique, en utilisant un programme de conversion.

Cet utilitaire de conversion s'est avéré difficile à réaliser, et finalement assez pénible à utiliser. De plus, je passais mon temps à le modifier, vu qu'un nouveau modèle de SoundTracker sort chaque semaine, au moins.

Bien, pour essayer d'améliorer les choses, j'ai mis dans cette version de l'extension musicale, un player direct de module SoundTracker, celui de StarTracker pour être précis. Ce player est compatible avec la plupart des trackers (sauf MED) au moment où j'écris cet article. Quand vous le lirez il ne le sera certainement plus, mais bon, vous ferez avec !

Alors, il y a de nouvelles et toutes belles instructions.

TRACK LOAD "Nom du module", Banque

Comme son nom l'indique, cette instruction charge un module dans une banque. Elle se charge de réserver la banque en mémoire CHIP, et de charger le fichier. Elle appelle également TRACK BANK : tout est prêt pour jouer la musique.

TRACK PLAY [Banque] [,Pattern]

Démarré la musique !

- Banque est un paramètre facultatif qui précise la banque (ou l'adresse) où prendre la musique. S'il est omis, la banque fixée par TRACK BANK ou TRACK LOAD est utilisée...

- Pattern est facultatif lui aussi et précise le début de la musique. Attention, aucun contrôle n'est fait sur sa validité : vous devez être sûr de vous, sous peine de crasher la machine. Cette option peut être



très utile pour avoir plus d'une musique dans un module ; il suffit alors de choisir le premier pattern pour démarrer une autre musique...

TRACK BANK Banque

Vous pouvez charger plusieurs modules dans plusieurs banques. TRACK BANK sert alors à spécifier la banque par défaut, comme SAM BANK pour les samples...

TRACK STOP

Pour... devinez quoi ? J'en vois un qui suit dans le fond. C'est ça, arrêter une musique en train d'être jouée.

TRACK LOOP ON/OFF

Inhibe ou autorise le bouclage de la musique lorsque celle-ci est terminée. Mode par défaut: TRACK LOOP OFF.

BON A SAVOIR...

Plusieurs petites choses sont à noter :

- les musiques jouées par TRACK PLAY n'autorisent pas les interruptions comme on pouvait le faire avec une musique AMOS normale. Par exemple, vous ne pouvez pas faire de bruitage pendant l'exécution d'une musique Tracker, sans prévoir dans la musique, une voix libre spécifiquement pour cet usage. Pour un jeu complexe, il donc vaut mieux utiliser les banques de musique normales...
- Les vu-mètres fonctionnent parfaitement !
- Je ne vais pas vous proposer d'exemple, tout de même ! Oh 'pi si, un 'achement gros exemple.

```
Track Load "mod.essai",5
Track Play
```

Et voila, en avant la musique !

Bien, j'en ai terminé. Désolé d'avoir un peu bousculé l'ordre (le désordre ?) habituel de cette rubrique, mais je pense que ces nouvelles instructions feront plaisir à plus d'un (dont moi, hé, hé!).

```

'-----
' Exemple de double buffer sonore...
' Par F.Lionet Paru dans l'ANT!
'-----
'
' NB: le double buffer sonore fonctionne parfaitement, et à une
' grande fréquence de reproduction sur un disque dur ou un CDTV.
'
' Sur disquette, il est illusoire d'espérer jouer convenablement
' des samples à plus de 8000 Herz, sans avoir recours à des
' accélérateurs de disquette.
'-----
'
' Reserve les banques. Il faut prévoir DEUX banques, en CHIP
' mémoire bien sur: les circuits doivent y avoir accès...
' Plus la longueur des banques est importante, plus les accès
' disque seront rares...
'
LBANK=1024*8
Reserve As Chip Work 10, LBANK
Reserve As Chip Work 11, LBANK
'
' Définit les voix utilisées
V=%11 : VX=0
'
' Ouvre le fichier.
F$="Samessai.sam" : F=10000
Open In 1,F$
'
' Prend une fois pour toute la longueur du fichier, pour éviter
' les POF et autres EOF, qui bougent la tête de lecture...
```

```

LSAM=Lof(1)
'
' A fond les potards!
Volume 63
'
' On initialise la première partie du sample
Sload 1 To Start(10),LBANK
'
' Le lancement doit être fait par un SAM RAW: c'est lui qui fixe
' la fréquence de reproduction utilisée par la suite...
Sam Raw V,Start(10),LBANK,F
'
' On soustrait la longueur d'un buffer à la longueur du sample.
LSAM=LSAM-LBANK
PSAM=1
'
' Boucle de chargement des samples.
While LSAM>0
'
Repeat
'
' Une bouclette, pour simuler un mauvais accès disque: en
' maintenant la souris enfoncée on désynchronise la boucle.
' Essayez donc pour voir!
While Mouse Key : Wend
'
' On ne bouffe pas le temps des autres taches
Multi Wait
'
' Peut-on maintenant charger?
P=Sam Swapped(VX)
'
Until P
'
' Ok, d'abord, faire le ménage dans la banque. Comme ça,
' lorsque l'on atteind la fin d'un sample, on entend plus
' rien. Je sais, j'aurais pu calculer la longueur restant à
' jouer. Mais bon, mon poil dans la main a pas mal grossi
' ces derniers temps...
Fill Start(10+PSAM) To Start(10+PSAM)+LBANK,0
'
' On charge la suite
Sload 1 To Start(10+PSAM),LBANK
'
' On initialise la suite:
If P=-1
'
' Ok, on peut swapper, on est toujours synchro
Sam Swap V To Start(10+PSAM),LBANK
Else
'
' Arghh! On est désynchronisé, il faut réinitialiser
' le sample
Sam Raw V,Start(10+PSAM),LBANK,F
End If
'
' Ya moins d'octets à charger...
LSAM=LSAM-LBANK
PSAM=1-PSAM
'
' Et on exite en pressant une touche...
If Inkey$<>"": Sam Stop : Exit : End If
'
Wend
'
' Ne pas oublier de fermer le fichier...
Close
```

Par François Lionet 

AREXX : LE DEVERMINAGE

Après avoir réalisé quelques pas guidés en ARExx, il va falloir maintenant écrire soi-même ses programmes. Habituellement, cela ne va pas sans quelques erreurs...

Même le programmeur le plus chevronné sait combien il est difficile et nécessaire de traquer les erreurs diverses nommées "bugs" partout dans le monde informatique et "bogues" chez les irréductibles gaulois ! Heureusement, conscient par expérience de ces difficultés, le père d'ARExx a fourni à l'interpréteur du langage un environnement de déverminage tout à fait efficace.

DEFAULTS

La liste des défauts possibles est aussi immense que l'ingéniosité du programmeur, et il serait vain d'en tenter une revue exhaustive. Cependant, le seul type contre lequel on ne peut rien est le défaut de logique dans la réalisation. On peut alors déterminer les grandes classes de défauts et donner les méthodes pour leur détection et correction, soit par ordre de gravité : la faute d'orthographe, la faute de syntaxe et la faute de système.

Faute d'orthographe

Cette faute apparaît sous deux formes : orthographe des mots réservés par le langage et orthographe d'une variable, non constante.

Dans le premier cas, l'analyseur orthographique du langage peut, mais pas toujours, détecter l'erreur. Dans le deuxième cas, l'analyseur orthographique est impuissant puisqu'il s'agit de noms de variables donnés par l'utilisateur, qui ne lui sont donc pas connus.

Il faut se rappeler que l'analyseur orthographique n'est pas tout puissant et qu'il ne sait détecter au mieux que l'endroit où il ne comprend plus le programme, endroit qui est fréquemment différent de celui où se situe réellement la faute. Il peut donc indiquer une faute qui n'est pas la bonne. Un travail d'interprétation est parfois nécessaire.

Faute de syntaxe

Cette faute traduit la mauvaise écriture d'une instruction : boucle non fermée, mauvais emploi (au sens de l'usage) de ses paramètres, etc. Elle peut être localisée ou répartie si elle est placée à cheval sur un sous programme.

L'analyseur de syntaxe du langage trouve certaines erreurs mais seulement lorsqu'il ne comprend plus, si bien que l'indication de l'endroit et de l'erreur peuvent être erronés, indépendamment. Fréquemment, la détection d'une erreur est la seule chose que l'analyseur peut faire, il faut alors un travail de recherche en aveugle pour effectuer la correction.

Faute de système

Dans cette rubrique, on place toutes les fautes non couvertes par les définitions précédentes. La multitude se refuse à toute description.

MOYENS DISPONIBLES

Plusieurs moyens sont disponibles en ARExx et peuvent être employés concurremment.

Analyseur orthographique et syntaxique

Il est toujours présent et fonctionne à chaque mise en oeuvre d'un programme (la rançon est le ralentissement de l'exécution dudit programme). Un catalogue de 48 messages est donné ; certains sont très explicites, comme le numéro 3 (mémoire insuffisante), d'autres le sont moins, comme le numéro 34 (caractère étranger) qui ne précise

rien et qui peut avoir rapport à un caractère non ASCII, donc invisible à l'écran.

A chaque message est associé une sévérité qui permettra par la suite d'effectuer une recherche sélective en fonction du niveau de sévérité.

Dans tous les cas, l'erreur provoque l'arrêt de l'exécution et l'affichage de la raison supposée. Une analyse est effectuée avant exécution du programme et certaines erreurs, comme le manque d'apostrophes (quotes) sont signalées dès le début ; d'autres ne sont signalées qu'à la découverte.

Ecriture du résultat

Dans la forme la plus simple, le résultat de la détection d'erreurs est affiché dans la fenêtre en cours. Cette disposition mélange le résultat du programme et celui de la détection d'erreurs, ce qui peut être désagréable à divers points de vue. Il peut être préférable d'utiliser une fenêtre séparée. C'est le rôle du petit programme nommé "tco" et distribué avec ARExx. "tco" signifie "Tracing Console Open". Ce programme doit être lancé en premier. Il ouvre une console spécifique dans laquelle on verra défiler les erreurs éventuelles. Pour le quitter, on utilise "tcc" (Tracing Console Close). Ces commandes doivent figurer dans le répertoire C:.

Cette forme de détection d'erreurs implicite, si l'on peut dire, permet de trouver les erreurs grossières mais devient rapidement hors de propos lorsque les défauts sont plus subtils.

La fonction TRACE

Cette instruction qu'on inscrit dans le programme suit son déroulement et en fait un rapport instruction par instruction. Le "tracage" peut être paramétré pour ne s'intéresser qu'à certains éléments du programme.

Il n'est pas non plus nécessaire de tracer tout le programme. Une instruction d'arrêt du tracage permet de contrôler précisément la portion de programme à tracer. La forme est donc :

```
/* Début du programme */
trace ...
/* Partie tracée */
trace off
/* Suite du programme */
```

Si tco a été lancé, le résultat de la trace est écrit dans cette console.

L'instruction TRACE s'écrit :

TRACE symbole | chaîne | VALEUR expression

- *symbole* peut être ? ou !. Le ? commande le mode interactif, dans lequel l'opérateur suit et peut modifier les valeurs en cours dans son programme et ré-exécuter telle ou telle instruction, pendant que le ! commande le mode inhibition. Dans ce mode, les commandes à destination de l'hôte ne sont pas exécutées réellement, mais n'entravent pas le déroulement du programme. C'est particulièrement intéressant lors de l'émission de commandes à caractère destructif (DELETE ou FORMAT par exemple). Attention : ces symboles fonctionnent en bascule qui en change la signification chaque fois qu'ils sont rencontrés : ? mode interactif ? mode normal ? mode interactif...

- *chaîne* peut prendre les valeurs suivantes : ALL, COMMANDS, ERRORS, INTERMEDIATES, LABELS, NORMAL, RESULTS, SCAN, ou seulement la lettre initiale. L'exemple ci-après permettra de voir leurs résultats respectifs.

- *VALEUR* expression permet de sauter des instructions si l'expression spécifiée est négative.

Le programme d'essai que voici n'a pas d'autre intérêt que de permettre l'évaluation des différentes options de trace qui seront écrites à la place de XXX à la ligne 3. Le symbole d'interdiction (!) dans cette même ligne empêche la réalisation réelle de la ligne 4, sans pour autant perturber le fonctionnement du programme.

```
1 /**/
2
```



```

3 trace ! XXX installe le mode trace avec interdiction (!)
4 ADDRESS COMMAND 'dir df1:' commande vers l'extérieur
5 DO i=1 TO 2 boucle
6 SAY i
7 a= test(i) appel à sous-programme
8 SAY a
9 END
10 TRACE OFF fin de traçage
11 EXIT fin du programme
12 test: label de sous programme
13 ARG k extraction de la variable à traiter
14 RETURN(k**k) retour avec calcul

```

Les tableaux ci-dessous résument les résultats de l'exécution de ce programme avec différentes valeurs pour XXX.

• pour XXX égale ALL : trace de toutes les instructions. Cela permet de voir comment se déroule le programme.

```

4 ** address command 'dir df1:';
5 ** do i=1 to 2;
6 ** say i;
7 ** a= test(i);
12 ** test:
12 ** ;
13 ** arg k;
14 ** return(k**k);
8 ** say a;
9 ** end;
5 ** do i=1 to 2;
6 ** say i;
7 ** a= test(i);
12 ** test:
12 ** ;
13 ** arg k;
14 ** return(k**k);
8 ** say a;
9 ** end;
5 ** do i=1 to 2;
10 ** trace off;

```

• pour XXX égale COMMANDS : seules les commandes vers l'extérieur sont tracées.

```

>>> "dir df1:" La commande vers l'extérieur est la lecture
du répertoire de DF1:.

```

• pour XXX égale ERRORS : l'erreur provoquée était ligne 7 : a=test(z), z étant une variable nouvelle sans valeur attribuée.

```

+++ Error 47 in line 14: Arithmetic conversion error
+++ Error 47 in line 7: Arithmetic conversion error

```

La variable étant non initialisée, l'appel du sous-programme s'effectue mal, d'où l'erreur en ligne 7. En ligne 14, l'opération n'a donc pas pu se faire. Les +++ en tête de ligne indiquent une erreur de syntaxe ou de commande.

• pour XXX égale INTERMEDIATES : c'est la forme la plus complète, dans laquelle toutes les valeurs intermédiaires sont tracées. Plusieurs sigles sont utilisés, dont voici la signification :

```

4 ** est une instruction, avec en tête son numéro de ligne
+++ est une erreur de syntaxe ou de commande
>C> nom de tableau complet
>F> résultat d'un appel de fonction
>L> label
>O> résultat d'une opération à deux termes
>P> résultat d'une opération préfixée
>U> variable non initialisée
>V> valeur d'une variable
>>> résultat d'expression
>> place vide.

```

```

4 ** address command 'dir df1:';
>L> "dir df1:" la commande n'est pas effectuée
5 ** do i=1 to 2;
>L> "1" label de départ
>>> "1" résultat de l'expression
>L> "2" label d'arrivée
>V> "1" la variable est i=1

```

```

6 ** say i;
>V> "1" idem
7 ** a= test(i);
>V> "1" appel avec variable=1
12 ** test:
12 ** ;
13 ** arg k;
>>> "1" résultat de l'extraction
14 ** return(k**k);
>V> "1" 1ère variable=1
>V> "1" 2ème variable=1
>O> "1" résultat de l'opération entre variables
>>> "1" résultat de l'expression (..)
>F> "1" résultat de l'appel à fonction
>>> "1" résultat de l'expression
8 ** say a;
>V> "1"
9 ** end;
5 ** do i=1 to 2; on avance d'un cran
>V> "1" la variable est 1
>O> "2" l'incrément de 1 donne pour résultat 2
>>> "2" le résultat de l'opération est 2
6 ** say i;
>V> "2"
7 ** a= test(i); appel au sous-programme
>V> "2" = 2
12 ** test:
12 ** ;
13 ** arg k;
>>> "2" extraction de la valeur transmise 2
14 ** return(k**k);
>V> "2" variable 1 = 2
>V> "2" variable 2 = 2
>O> "4" résultat de K**K=4
>>> "4" résultat de (...)=4
>F> "4" résultat de return=4
>>> "4" renvoi de 4
8 ** say a;
>V> "4"
9 ** end;
5 ** do i=1 to 2;
>V> "2" la variable actuelle est 2
>O> "3" le résultat avec incrément de 1 est 3
>>> "3" le résultat de l'incrément est 3 d'où abandon
10 ** trace off;

```

• pour XXX égale LABELS : ce sont les expressions suivies du signe deux-points (:), comme les sous-programmes ou les points de branchement.

```

12 ** test:
12 ** test:

```

L'appel au sous programme a été fait dans une boucle à deux passes, il y a deux fois appel à ce sous-programme.

• pour XXX égale NORMAL : ce mode ne trace que les erreurs qui sont d'une sévérité supérieure à un niveau pré-établi ; on verra comment après.

• pour XXX égale RESULTS : ce mode n'affiche que les résultats.

```

4 ** address command 'dir df1:';
>>> "dir df1:"
5 ** do i=1 to 2;
>>> "1"
>>> "2"
6 ** say i;
>>> "1"
7 ** a= test(i);
>>> "1"
12 ** test:
12 ** ;
13 ** arg k;
>>> "1"
14 ** return(k**k);
>>> "1"
>>> "1"
8 ** say a;
>>> "1"
9 ** end;

```

(suite page 23)

PASCAL : PRESENTATIONS

Bienvenue dans le fabuleux monde du Pascal ! Ensemble, nous allons découvrir ce langage de programmation créé spécialement pour l'enseignement par le Professeur Wirth.

Et pour commencer, un peu de culture générale sur ce langage (ne vous inquiétez pas, ce ne sera pas long)... Le Pascal est un langage évolué (pour nous) développé comme une aide à la programmation des ordinateurs suivant une approche dite de "programmation structurée". Il a été créé par le professeur Wirth (également père du Modula II, qui descend du Pascal). Il est un des langages les plus utilisés (si ce n'est le plus utilisé) et peut servir aussi bien pour des applications système (par exemple, l'écriture du système Mac), que pour la programmation éducative et personnelle. Pascal existe sur de nombreuses machines (MS-DOS, Mac, UNIX) ce qui permet une certaine portabilité (il existe un standard décrit dans le livre de Wirth et Jensen: "Pascal User Manual and Report").

Le Pascal possède de nombreux avantages, dont ceux de structurer la pensée, de faciliter l'implémentation d'algorithmes, de simplifier la correction d'un programme (si on se borne à respecter quelques règles de présentation et d'écriture). De plus, le langage est rapidement assimilable et permet d'acquérir une méthodologie qui vous sera utile dans d'autres langages plus laxistes.

Et comme pour tout langage, il dépend de son implémentation sur machine. Pour nous, ce sera le très célèbre PCQ de Patrick Quaid. Vous pourrez trouver ce compilateur sur la disquette ANT numéro 22 ou la Fish 339 (reportez-vous à l'excellent article de Herr Doktor du numéro 22 pour l'installation et assurez-vous que votre Startup-Sequence contient la ligne `assign INC: xxx/include`, où XXX est le chemin pour arriver au répertoire "include" du Pascal, ceci afin de le rendre compatible avec les listings des programmes comportant des inclusions).

En résumé, nous avons une disquette nommée PCQ: devant contenir au moins les répertoires et fichiers suivants :

```
c (dir)
A68k
Az (ou tout autre éditeur de votre choix)
BLink
Pascal
include (dir)
  les ".i" du Pascal
pcq.lib
```

Pour compiler, j'utilise le script du Herr Doktor Von Glupa (à l'époque) que je nomme "make".

Quelques notes sur le compilateur PCQ : il n'est pas tout-à-fait standard, mais P. Quaid a importé des facilités (procédures et autres conventions) du Turbo-Pascal de Borland et du Quick Pascal de Microsoft que l'on trouve dans le monde MS-DOS, ainsi que certaines du Modula II. Le source Pascal est traduit en assembleur et produit un fichier assez gros. Si vous avez de la mémoire (1 Mo minimum), utilisez le disk Ram.

Remarques importantes : si vous jonglez avec les minuscules et les majuscules, sachez qu'en Pascal, elles sont équivalentes : `Toto = TOTO = TotO`. Mais les caractères accentués, eux, ne le sont pas et je ne les recommande pas.

Par la suite, nous supposerons que vous connaissez tout ce qui est externe au Pascal, et même la façon de compiler, d'éditer un texte et d'exécuter un programme, car tout ceci dépend du système d'exploitation et ce n'est pas le but de cette initiation.

Pour réussir quelque chose en Pascal, il faut pouvoir écrire un programme. Un programme est un texte, comme celui que vous lisez

en ce moment-même, mais il obéit à des règles bien précises, autant sur le vocabulaire que sur la syntaxe (et c'est là toute la difficulté !).

Pour débiter cette initiation, voici un petit exemple que nous allons étudier (accrochez-vous !).

```
PROGRAM Premier_Exemple;
```

```
BEGIN
  writeln('Vive AMIGA !')
  { Whaou, l'exemple ! }
END.
```

Tapez, sauvez, compilez puis exécutez (non pas au fusil !) ce programme et, ô joie, le texte "Vive AMIGA !" apparaît sur l'écran.

Vous aurez deviné le rôle du programme, ce que l'on nomme sa *spécification* : "afficher un texte chauvin sur la plus belle des bécane". Ainsi, votre prochaine mission, si vous l'acceptez, sera de faire l'inverse : à partir d'une spécification, vous produirez un texte Pascal équivalent.

ETUDE DU PROGRAMME (A LA NICOLAS HULOT)

Séquence description

Le premier mot du texte est *PROGRAM* suivi du nom du programme et d'un point-virgule, puis du mot *BEGIN* puis un grand mot *writeln*("Vive AMIGA !");, un texte entre accolades et enfin, le mot *END* avec un point.

Séquence explications

Nous sommes en présence de ce que nous nommerons un bloc. Il commence au mot *PROGRAM*, qui signifie que tout ce qui suit forme un programme. Le mot *Premier_Exemple* est en fait un identificateur : une suite (ou plutôt une séquence, car le sens a de l'importance) de lettres et du signe de soulignement "_". Le point-virgule termine ce que nous appellerons l'en-tête du bloc (ici, le bloc programme).

Après cet en-tête, nous avons une grande phrase commençant par *BEGIN* et finissant par *END*. Cette partie est la zone (séquence, ou segment) d'instructions ; elle sera exécutée à chaque appel du programme. La seule chose qui semble compter est donc, en fait, le fameux grand mot dont seul ce qui est entre quotes (les français disent apostrophes) est affiché. C'est ce que l'on appelle une constante *chaîne de caractères*.

Le premier sous-mot est en fait l'appel d'une procédure qui porte le nom de *writeln*. Une procédure est le résumé d'une séquence d'instructions. A chaque référence de ce nom (un appel), cette séquence sera exécutée. Dans le cas de notre exemple, la procédure est appelée avec un argument, qui est la chaîne de caractères. La spécification de *writeln* est : "envoi sur la sortie standard les arguments sous forme de chaînes ASCII avec un retour chariot tout à la fin". Cette procédure est prédéfinie, ce qui nous évite de le faire, car il existe une règle importante en Pascal : tout ce qui est utilisé doit être défini plus haut dans le texte. La règle n'est pas complète et nous verrons un peu plus tard comment définir quelque chose (on dira "définir un objet [du programme]").

J'ai volontairement passé sous silence la cinquième ligne (celle avec des accolades) parce que j'ai joué au compilateur : tout ce qui est mis entre les signes "{" et "}" est ignoré (à l'exception des directives de compilation) et le tout est considéré comme un séparateur (comme un blanc ou une tabulation). En Pascal, il toujours bon de commenter un programme afin de le comprendre quelques mois plus tard.

De ce que nous venons de voir, nous déduisons que le plus court programme Pascal qui soit est le suivant :

```
PROGRAM plusPetit;
BEGIN
END.
```




Si vous le compilez, vous verrez que le source produit par PCQ et assemblé par A68K est de 62 lignes et que le code final est de 2048 octets (2 Ko tout juste) en mémoire (2752 sur disque).

Une autre constatation : les retour-chariots équivalent à un simple séparateur comme les commentaires, les espaces, les tabulations. Le seul séparateur d'instructions est le point virgule. Mais une question s'impose "qu'est-ce qu'une instruction ?".

LES INSTRUCTIONS

Une instruction est un ordre Pascal qui sera compilé sous forme de code machine exécutable (en opposition aux données qui sont les objets manipulés par le programme). Elle peut être une affectation, un appel de procédure, ou un contrôle du flux d'exécution. Nous avons déjà vu l'appel de procédure. Un résumé de sa syntaxe générale serait : le nom de la procédure à appeler suivi, si nécessaire, d'une parenthèse ouvrante, d'une séquence d'expressions (calcul au sens large) séparées par des virgules (les arguments) et une parenthèse fermante. Quelques exemples avec la procédure `writeln` (qui n'a pas un nombre d'arguments fixe) :

```
writeln;
writeln('toto');
writeln('toto', 'mon héros');
```

Dans un segment d'instructions, on peut mettre plusieurs instructions à la suite, que l'on sépare par un point-virgule comme par exemple :

```
BEGIN      {Seul le segment d'instruction est donné }
  writeln('on ne change pas de procédure');
  writeln('toujours ce bon writeln');
  writeln {sans argument cette fois on passe simplement à la
  ligne }
END.
```

Remarquez que le dernier appel de fonction n'a pas de point virgule du fait que `END` est considéré comme la fin d'une instruction. En Pascal, le couple `BEGIN...END` (où les trois points représentent une séquence d'instructions) équivaut à une seule instruction. Donc, nous pouvons aussi avoir, en reprenant l'exemple ci-dessus :

```
BEGIN      {Seul le segment d'instruction est donné }
  Begin
    writeln('on ne change pas de procédure');
    writeln('toujours ce bon writeln')
  End; { remarquez le point-virgule indiquant la fin d'une
  instruction }
  writeln; { le point virgule n'est pas autorisé devant un end
  mais beaucoup de compilateurs dont PCQ l'acceptent }
END.
```

Dans cette exemple, cette technique n'est pas très pratique, mais pour certaines formes d'expression où une seule instruction est autorisée, il est très utile de l'étendre de la sorte. Nous ne verons pas tout de suite ces expressions car cela dépasse le cadre de cet article.

LES VARIABLES

Les variables sont des objets qui sont manipulés par le segment d'instructions. Une variable est une place réservée en mémoire dans laquelle on range des valeurs susceptibles de changer dans le temps. Le compilateur utilise une méthode pour coder cette valeur, méthode différente pour chaque type de variable.

En informatique, une variable est souvent représentée par le quadruplet *nom/type/portée/adresse*.

Le nom permet simplement de référencer la variable (comme pour les procédures). Le type indique la méthode de stockage utilisée par le compilateur (qui est le plus souvent transparente au programmeur), ainsi que les opérations possibles sur cette variable. La portée indique en quel(s) endroit(s) du programme la variable est connue et donc référençable. C'est l'un des principes fondamentaux du Pascal, sur lequel nous reviendrons en temps et en heure. Enfin, l'adresse de la variable indique tout simplement où dans la mémoire de l'ordinateur elle est stockée.

Comme précisé plus haut, pour utiliser un objet quelconque, il faut d'abord le déclarer. Donc ...

LES DECLARATIONS

Certains types de variables sont prédefinis en Pascal. Nous les étudierons le moment venu. Pour l'instant, sachez seulement que la forme syntaxique de la déclaration de variable en Pascal est :

`VAR Identificateur : type;`

où *Identificateur* est le nom de la variable, et *type* le nom du type de la variable.

Le mot `VAR` indique au compilateur que ce qui suit est une zone de déclarations de variables (car il peut y en avoir plusieurs à la suite). Un exemple pour fixer les idées.

```
VAR UnEntier : integer ;
  { déclaration de la variable "UNENTIER" du type entier }

VAR UnAutre, PuisUnAutre : integer ;
  { déclaration multiple de variable de même type }
```

Dans un programme Pascal, il faut mettre toute déclaration entre l'entête et le mot `BEGIN`. Cette zone est appelée (devinez ?) zone de déclarations locales. Pourquoi locales ? Parce que les variables ne sont connues que du programme (et pas des autres, na!). Bon, à ce niveau, on peut oublier le local, mais on touche là l'une des difficultés du Pascal (ce que l'on nomme les "effets de bord" et la "localité"). Encore un petit programme, pour clarifier la situation.

```
PROGRAM Essai_4;

var UnPetitNbre : integer ; { déclaration de UNPETIT de type
entier }

BEGIN
  UnPetitNbre := 1 + 1 ;
  { on affecte à la variable UnPetit la valeur 1+1 (1+1 est de
  même type qu'UNPETIT, c-à-d integer) }

  writeln ('le résultat de 1 + 1 est ', UnPetitNbre);

  { L'avantage de certains types prédéfinis est qu'ils peuvent
  être affichés sans difficulté supplémentaire }

END.
```

Bon, on s'arrêtera là pour aujourd'hui, ça fait déjà beaucoup de notions nouvelles à assimiler d'un coup : variables, architecture d'un programme simple (trop simple pour certains, je sais !), type *integer* et procédure `writeln`... Le mois prochain, on reprendra le tout mais avec plus de précision et on avancera un peu plus dans la pratique. En attendant, je vous laisse un petit devoir : trouvez la spécification du programme suivant.

```
Program Pas_Encore_de_Specification;

var age,date, cetteAnnee : integer;

BEGIN
  write ('Quel est-on âge, mon petit ? ');
  readln (age);
  writeln(' Ho ! tu ne fais pas tes ',age,' ans, mon grand !');

  { avouez que je commence fort, non ? }

  write ('Quel est ton année de naissance :');
  readln (date);
  cetteAnnee := age + date;
  { additionne la valeur de la variable age avec celle
  de date et la stocke dans la variable CETTEANNEE }
  { en gros, je complique la mathématique }
  writeln ('On ne serait pas en ',CetteAnnee,' ?');
  { qui dit que l'informatique est très intellectuel ? }
END.
```

Bye et au mois prochain, pour de nouvelles z'aventures Pascalissimes.

Par Olivier Garrigues

GFA : LE TIMER.DEVICE

En quelques années, GFA Basic a acquis une immense renommée, au point d'être possédé par environ un million de personnes. Et c'est un juste retour des choses après ses débuts difficiles. Comme tout nouveau produit, il avait souffert de nombreux bugs de jeunesse.

GFA-Basic est maintenant plus performant que le C dans de nombreux domaines (NDLR : cet avis n'engage que l'auteur !). Et on va le prouver. J'utilise la version 3.52F mais attention, la dernière version disponible en France est la 3.042. (ni 3.1, ni 3.2, ni 3.3, ni même 3.4). Il faut également signaler qu'il existe plusieurs ouvrages, publiés par Micro-Application, qui traitent du GFA-Basic version Amiga.

Un logiciel professionnel se reconnaît à la simplicité de son codage et à la puissance de son exécutable. Pour commencer la série, je voulais frapper un grand coup en proposant un Plasma, géré sur toutes les lignes et non pas une ligne sur deux comme cela se voit trop souvent dans les démos. Micro Application me dit qu'il serait plus professionnel de montrer la gestion d'un device. Mais voilà, c'est compliqué pour des lecteurs ne connaissant que l'aspect Basic du GFA.

Sauf peut-être le timer.device. Et en plus, on s'aperçoit de l'ignoble vérité: GFA est souvent plus rapide que Lattice C, de 1% à 15%.

Le codeur recherche toujours trois choses : la simplicité, la rapidité et la synchronisation. Le délai se calcule fréquemment avec les fonctions *TIMER*, *DELAY* et *PAUSE*. Pour plus de précision, on utilise également souvent le blanc vertical avec le registre *VPOS*. Ou mieux, on ouvre le dispositif approprié. Timer.device permet de gérer les deux timers internes de l'Amiga grâce aux unités *UNIT_VBLANK* et *UNIT_MICROHZ*.

UNIT_VBLANK (valeur 1) sélectionne le timer de synchronisation sur le retour du faisceau d'électrons. C'est celui des longs moments (précision de 0.02 secondes). Avec *UNIT_MICROHZ* (valeur 0), on sélectionne le timer du CIA 8520, ultra précis, mais qui accapare trop le microprocesseur pour être valable sur des durées dépassant la seconde.

Peut-on coder directement le Hardware en GFA? Oui, mais ce n'est pas élégant. La vraie solution passe donc par le timer.device, qui assurera la pérennité de vos applications avec l'évolution de l'Amiga.

Le codage commence par charger instantanément une image de 5 plans de bits. Heureux utilisateurs des disquettes ANT, vous êtes des privilégiés car vous aurez l'exécutable *IMENSITE* et la source *IMENSITE.GFA* (deux fois 75 Ko, dont une bonne partie pour l'image). On crée ensuite une structure de port que l'on étend à 40 octets. Puis l'on ouvre le dispositif périphérique timer.device et l'on nomme les parties intéressantes de la structure *timerequest%*. On est prêt, timer.device peut commencer à s'amuser.

Il charge donc le nombre de secondes écoulées depuis la création de l'univers, le premier janvier 1978, peu de temps avant la naissance des Mammouths préhistoriques. Après ces milliards et ces milliards de secondes, il affiche sa différence. Le programme est fortement inspiré des pages 677 et suivantes de la Nouvelle Bible de l'Amiga.

Rassasié, on referme le tout puis on passe à la deuxième partie qui utilise la fonction *CurrentTime* de la bibliothèque *Intuition*. Une fois compilé, le codage se synchronise au début de la seconde suivante pour éviter les aberrations de mesure, puis il teste la ou les fonctions désirées. Une boucle à vide met 64 cycles en GFA compilé avec 1 plan de bits de 320 pixels sur 256. Ce résultat varie bien sûr selon les autres tâches en cours. Je trouve de 62 à 65 sur un Amiga 500, lecteur externe, 1 Mo. En Lattice C, on aura 70 cycles et plus. De même, pour afficher la lettre A avec *printf("A")*; on met 10535 cycles en Lattice C, contre 10490 en GFA qui déteste pourtant le *PRINT*. En utilisant des fonctions spécifiques GFA, on descend à 1409 cycles, et moins quand on ruse.

Puis on passe à la troisième partie, qui ne dure que le temps que reste enfoncé le bouton de la souris.

```
*****
* TIMER DEVICE *
*
* CODAGE Pierre Philippe Launay *
* GRAPHISME Inconnu Domaine Public *
* SCENARIO Catherine Lardy *
* COPYRIGHT Mai 1991 *
* EXTRAIT DE Annabella 3 exemple 2 *
*
* Résidence les Cottages-83 rue André *
* Theurist- F63000 CLERMONT-FERRAND *
*****

programme ! Une seule instruction.
> PROCEDURE programme ! Grain de sable par grain de sable
met.ecran ! Oui
prEcalcule ! Pour aller plus vite à l' essentiel
copie.cercle ! Dessin et sauvegarde de l' horloge
variable ! Le timer en chiffre
crEation.port ! Ouverture d' un port pour les messages
ouvre.device ! Ouverture du périphérique timer.device
crEation.structure ! Remplissage de la structure timerequest
REPEAT
charge.heure ! Saisie de l' heure en heure Amiga
traduct.heure ! Formatage en heure humaine
affiche.heure ! Visionnage de l' heure
bip.heure ! Radio France Télécom MF 3699
UNTIL MOUSEK ! Attente du clic
fin ! Good night
SYSTEM ! Enlever lors de la session de codage
RETURN

> PROCEDURE crEation.port ! Nouveau port de message
timerport%=CreatePort(0,0) ! Création d'une sortie des réponses
IF timerport%=0 ! Manque de mémoire
SYSTEM ! Fin Dare dare
ENDIF
timerequest%=CreateExtIO(timerport%,40)
IF timerequest%=0 ! Glou glou glou
DeletePort(timerport%) ! Remise à l' état normal
ENDIF
RETURN

> PROCEDURE crEation.structure ! Structure timerequest
ABSOLUTE ln_succ%,timerequest% ! Entrée suivante
ABSOLUTE ln_pred%,ADD(timerequest%,4) ! Entrée précédente
ABSOLUTE ln_type%,ADD(timerequest%,8) ! Indication du type
ABSOLUTE ln_pri%,ADD(timerequest%,9) ! Niveau de priorité
ABSOLUTE ln_name%,ADD(timerequest%,10) ! Nom de la structure
ABSOLUTE mn_replyport%,ADD(timerequest%,14) ! Port
ABSOLUTE mn_length%,ADD(timerequest%,18) ! Longueur du noeud
ABSOLUTE io_device%,ADD(timerequest%,20) ! Périphérique
ABSOLUTE io_unit%,ADD(timerequest%,24) ! Numéro du lecteur
ABSOLUTE io_command%,ADD(timerequest%,28) ! Commandes cmd
ABSOLUTE io_flags%,ADD(timerequest%,30) ! Drapeau
ABSOLUTE io_error%,ADD(timerequest%,31) ! Statut
ABSOLUTE tv_secs%,ADD(timerequest%,32) ! Les secondes
ABSOLUTE tv_micro%,ADD(timerequest%,36) ! Les 1/100 secondes
RETURN

> PROCEDURE delete_extio(extio%) ! Destruction de la structure
EXTIO(ADD(extio%,8))=-1 ! Type
LONG(ADD(extio%,14))=-1 ! Port des Messages
LONG(ADD(extio%,20))=-18 ! Pointeur sur le device
-MFREE(extio%,WORD(ADD(extio%,18)))
RETURN

> PROCEDURE ouvre.device ! Ouverture du timer.device
erreur%=OpenDevice(V:dispositif$,unit_microhz$,timerequest%,unit_vblank%)
IF erreur%<>0 ! Mon Dieu, ça plante
DeletePort(timerport%) ! Dissolution du canal du timer
delete_extio(timerequest%) ! Désintégration du canal
SYSTEM ! Toujours tout refermer
ENDIF
RETURN

> PROCEDURE variable ! Les constantes du système
unit_microhz%=0 ! Unité du périphérique. CIA-Timer
unit_vblank%=1 ! Vertical Blank Timer
REM ! Avec 0 les microsecondes sont inutiles
dispositif$="timer.device"+CHR$(0)
cmd_invalid%=0 ! Commande invalide
cmd_reset%=1 ! Retour du timer à l' état initial
cmd_read%=2 ! Lecture dans le périphérique
cmd_write%=3 ! Ecriture dans le périphérique
cmd_update%=4 ! Traitement des espaces tampons
cmd_clear%=5 ! Effacement de tous les tampons
cmd_stop%=6 ! Mise en place d' une pause
cmd_start%=7 ! Reprendre après la pause
cmd_flush%=8 ! Interruption du travail en cours
cmd_nonstd%=9 ! La suite de la structure
tr_address%&=ADD(cmd_nonstd%,0)
tr_getsystem%&=ADD(cmd_nonstd%,1)
tr_setsystem%&=ADD(cmd_nonstd%,2)
RETURN

> PROCEDURE chiffre ! En écriture romaine
COLOR 3,0
TEXT 80,78,"XXII"
TEXT 86,147,"VI"
TEXT 25,113,"IX"
TEXT 143,113,"III"
RETURN

> PROCEDURE copie.cercle ! Copie d' une portion d' écran
COLOR 3,0
-Disable() ! PCIRCLE et PELLIPSE ont été optimisés
FOR rayon%=33 TO 38 ! Pour le compilateur. Le Basic n' est
-DRAWellipse(rastport%,92,110,60,rayon%) ! Pas assez rapide
NEXT rayon%
-Enable()
chiffre ! Le cadran
GET 20,70,160,150,horloge$
COLOR 1
GRAPHMODE 0 ! OR
TEXT 215,253,"Code PPL & CL" ! Copyright
```




```
COLOR 18
TEXT 8,6,"GFA EST PLUS RAPIDE QUE LE C DE 10 %" ! La preuve
GRAPHMODE 1 ! Normal
GET 0,0,297,7,titre$ ! Plus rapide
RETURN
> PROCEDURE met.ecran ! L'image
DIM c%(31),bp%(4),a$(4) ! Listes c() et bp() et a$()
FOR remplissage=0 TO 4 ! Depuis remplissage=0 jusqu'à 1
a$(remplissage)=SPACE$(10641)
NEXT remplissage
b$=SPACE$(129) ! Fin du bloc FOR NEXT
INLINE bp1%,10641
INLINE bp2%,10641
INLINE bp3%,10641
INLINE bp4%,10641
INLINE bp5%,10641
INLINE cou%,129
SWAP bp%(0),bp1% ! Echange des valeurs de bp1% et bp%(0)
SWAP bp%(1),bp2% ! Echange des valeurs de bp2% et bp%(1)
SWAP bp%(2),bp3% ! Echange des valeurs de bp3% et bp%(2)
SWAP bp%(3),bp4% ! Echange des valeurs de bp4% et bp%(3)
SWAP bp%(4),bp5% ! Echange des valeurs de bp5% et bp%(4)
OPENW 1,0,0,320,256,5,0 ! N° 1 X320 x Y256 x 4Couleurs Standard
ecran1%=SCREEN(1) ! Pointeur sur la structure écran N° 1
FOR couleur=0 TO 31 ! Lecture des couleurs
SETCOLOR couleur%,LONG(ADD(cou%,MUL(couleur%,4)))
NEXT couleur
FOR plan=0 TO 4
MOVE bp%(plan%),V:a$(plan%),10641 ! Copie 10641 octets de
bp%(0) sur a$(0)
NEXT plan
SPUT a$(0) ! Affiche la première image
rastport%=(ADD(WINDOW(0),50)) ! Pour la fonction C DrawEllipse
RETURN
> PROCEDURE precalculer ! Résultats précalculés
DIM cossec%(60),sinsec%(60),cosmin%(60),sinmin%(60)
DIM cosheure%(24),sinheure%(24),secondex%(3),secondey%(3)
DIM gamme%(3)
centrex%=92
centrey%=110
FOR top%=0 TO 59
top2%=SUB(MUL(top%,6),90)
cossec%(top%)=ADD(COSQ(top2%)*60,centrex%)
sinsec%(top%)=ADD(SINQ(top2%)*33,centrey%)
cosmin%(top%)=ADD(COSQ(top2%)*70,centrex%)
sinmin%(top%)=ADD(SINQ(top2%)*35,centrey%)
NEXT top%
cosheure%(0)=centrex%
sinheure%(0)=SUB(centrey%,20)
FOR top%=1 TO 23
top2%=SUB(MUL(top%,30),90)
cosheure%(top%)=ADD(COSQ(top2%)*30,centrex%)
sinheure%(top%)=ADD(SINQ(top2%)*20,centrey%)
NEXT top%
secondex%(1)=centrex%
secondey%(1)=centrey%
secondex%(2)=centrex%
secondey%(2)=centrey%
gamme%(0)=523 ! Do
gamme%(1)=659 ! Mi
gamme%(2)=784 ! Sol
gamme%(3)=1046 ! Do
octave%=1
DEFFILL 2,2,21 ! Tramme des secondes
RETURN
> PROCEDURE son(frequence%) ! La flemme d' écrire
SOUND frequence%,2 ! Son 880 hertz (LA) durant 2 cycles
pause(1,0) ! Délais de 1 seconde 0 bip
RETURN
> PROCEDURE affiche.heure ! La montre qui tient ses promesses
IF OR(temoin%<>secondes%,secondes%=0)
COLOR 2,0 ! L' aiguille des secondes
secondex%(0)=cossec%(secondes%)
secondey%(0)=sinsec%(secondes%)
secondex%(3)=secondex%(0)
secondey%(3)=secondey%(0)
PUT 20,70,horloge$ ! Dessin de l' horloge déjà saisie
chiffre ! Le cadran
COLOR 4,0 ! Les aiguilles heures minutes
LINE centrex%,centrey%,cosmin%(minutes%),sinmin%(minutes%)
LINE centrex%,centrey%,cosheure%(heures%),sinheure%(heures%)
secondex%(2)=cossec%(secondes%)
secondey%(2)=sinsec%(secondes%)
COLOR 2,31
temoin%=secondes%
ENDIF
RETURN
> PROCEDURE bip.heure ! Top top top top
SELECT secondes%
CASE 0 ! Flash au début de chaque minute
PUT 20,70,horloge$ ! Dessin de l'horloge déjà saisie
test.heure ! Changement de seconde
COLOR 3,30 ! Changement de couleur du fond
TEXT 10,6,"Il est très exactement sur Amiga !"
pause(0,2) ! Délais de 0 seconde 2 bips
son(1000) ! Son 1000 hertz durant 2 cycles
son(800) ! Son 800 hertz durant 2 cycles
son(600) ! Son 600 hertz durant 2 cycles
PUT 0,0,titre$ ! Itself
DEFFILL 2,2,SUCC(minutes% MOD 24) ! Tramme de remplissage
CASE 5,10,15,20,25,30,35,40,45,50,55 ! Toutes les 5 secondes
test.heure
~DisplayBeep(0) ! Flashage de tous les écrans
octave%=SUCC(RAND(4))
DEFAULT
test.heure
son(MUL(gamme%(son%),octave%))
son%=SUCC(son%) MOD 4
ENDSELECT
RETURN
```

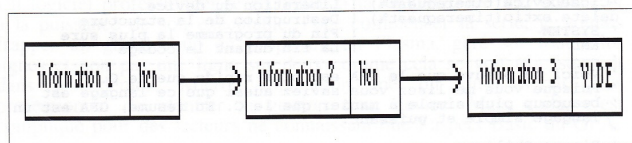
```
> PROCEDURE charge.heure ! Le Load Heure
io_command%=tr_getsysstime% ! La commande
~DoIO(timerrequest%) ! Mise en action du périphérique
Fin au signal de fin donné alors par le timer.device
seconde%=tv_secs% ! C'est fait: on obtient les secondes
tick%=tv_micro% ! Et les microsecondes
RETURN
> PROCEDURE pause(secondes%,tick%) ! L' ultra précision
FOR temps=1 TO ADD(MUL(secondes%,100),tick%)
~Disable() ! Système presque complètement gelé
io_command%=tr_addrrequest% ! Attente du temps imparti
tv_secs%=0
tv_micro%=1
~DoIO(timerrequest%) ! Mise en action. C' est déjà fini.
~Enable() ! Dégèlement de l' Amiga
NEXT temps ! La boucle n' est qu' un petit gag
! d'accélération temporelle
! Pour comparer
> PROCEDURE test.heure
TEXT 56,160,TIME$
TEXT 40,168,DATE$
TEXT 48,176,STR$(seconde%,9)
tick%=STR$(tick%)
tick%=STRING$(SUB(6,LEN(tick%)),"0")+tick%
TEXT 72,184,tick%
RETURN
> PROCEDURE traduct.heure ! En bon français bien de chez nous
seconde indique le nombre de secondes depuis la création
heures%=DIV(secondes%,3600) MOD 24 ! 24 heures de 3600 secondes
minutes%=DIV(secondes%,60) MOD 60 ! 60 minutes de 60 secondes
secondes%=secondes% MOD 60 ! 60 secondes par minute
RETURN
~DeletePort(timerport%) ! Abolition du port du timer
~CloseDevice(timerrequest%) ! Libération du device
delete.extio(timerrequest%) ! Destruction de la structure
SYSTEM ! Fin du programme la plus sûre
END ! La fin durant le codage
Voici la preuve que le GFA est plus rapide que le C.
Puisque vous ne lisez vous saviez aussi que ce langage est
beaucoup plus simple à manier que le C. En résumé, GFA est un
langage simple et puissant.
Pierre Philippe
TEXT 2,75,"15 secondes de noir complet dès le clic"
WHILE MOUSEK
WEND
REPEAT
UNTIL MOUSEK
SPUT a$(0)
GRAPHMODE 0
TEXT 215,253,"Code PPL & CL" ! Copyright
seconde%=0
tick%=0
~CurrentTime(V:seconde%,V:tick%) ! Retourne l' heure locale
COLOR 10
TEXT 10,75,"EXEMPLE 2 AVEC CURRENTTIME"
COLOR 4
TEXT 10,85,"Seconde Système"+STR$(seconde%,10)
TEXT 10,95,"Tick Système "+STR$(tick%,10)
durEe%=15 ! En seconde pour ne pas se quiquiner
COLOR 3,2
TEXT 10,105,"Départ du chronométrage "
COLOR 1,0
TEXT 10,114,"Durée : "+STR$(durEe%,3)+" secondes"
compteur%=0 ! Quelques variables
departsec%=0
departtick%=0
chronosec%=0
chronotick%=0
DISPLAY OFF
REPEAT ! Départ synchro à la seconde suivante
UNTIL departsec%>seconde%
REPEAT
... ! Mettre ici les instructions à tester
INC compteur%
~CurrentTime(V:chronosec%,V:chronotick%)
UNTIL SUB(chronosec%,departsec%)>durEe%
delaix%=SUB(chronosec%,departsec%)
MUL delaix%,1000000 ! Resultat en microseconde
COLOR 3,2
TEXT 10,123,"Fin du chronométrage "
COLOR 1,0
TEXT 10,132,"Boucles "+STR$(compteur%,10)
TEXT 10,140,"Délais "+STR$(delaix%/compteur%-62,10)+" 10^6"
COLOR 4
TEXT 10,152,"En tournant à vide, GFA met 64 cycles"
GRAPHMODE 1 ! 70 et 64 pour un plan de bits sans DISPLAY OFF
TEXT 10,162,"En Lattice C on enlève 70 pour avoir 0"
TEXT 10,170,"En GFA on enlève 64 pour avoir 0"
COLOR 30,4
TEXT 10,179," Attention : on est en multitâche "
TEXT 10,187," Résultats variants de 62 à 66 "
TEXT 10,195," Et selon le nombre de Plan de bits "
PUT 10,200,titre$
DISPLAY ON
Je remercie BatchMan pour sa contribution au Basic GFA.
REPEAT
UNTIL MOUSEK
SWAP a$(0),a$(1)
SPUT a$(0)
TEXT 15,75,"Passez une bonne et agréable journée"
TEXT 199,253,"Pierre Philippe"
WHILE MOUSEK
WEND
RETURN
```

Par Pierre Philippe Launay

ALGOS : LES LISTES CHAÎNÉES

Cette nouvelle rubrique "Algorithmes et structures de Données" aura pour but de vous présenter chaque mois un exemple d'utilisation des structures de données, à travers quoi ? A travers un algorithme, oui.

Pour notre premier article, nous allons aborder les listes doublement et simplement chaînées. Et pour commencer, une petite définition : une liste chaînée est une structure de données dynamiques. Il s'agit d'un noeud composé de deux parties : un morceau information et un autre morceau, appelé lien, qui permet de chaîner les noeuds entre eux. Le schéma ci-dessous résume parfaitement la structure d'une liste chaînée.



STATIQUE, DYNAMIQUE : DIFFÉRENCES

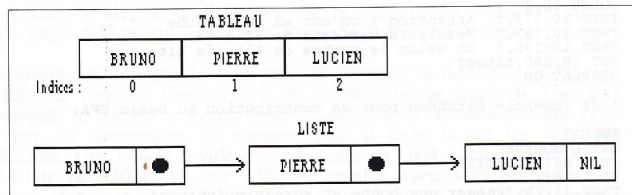
Afin de bien comprendre ce qui sépare ces deux organisations, effectuons un parallèle entre le tableau (statique) et la liste (dynamique).

Le tableau

- **taille prédéfinie** : il faut connaître à l'avance ses besoins en espace de façon bien précise. En effet, la taille du tableau doit être initialisée dans le code source du programme.
- **taille figée** : cette deuxième restriction découle de la première. En effet, une fois le code source compilé, l'on ne peut plus modifier (de façon décente) la taille d'un tableau, et ceci pendant toute la durée d'exécution du programme.

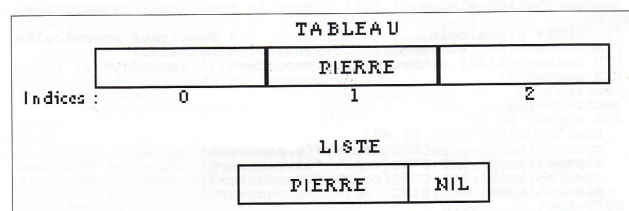
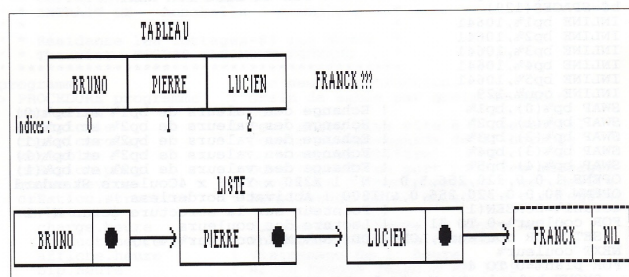
La liste chaînée

- **taille non prédéfinie** : l'on n'est pas obligé de connaître la quantité exacte de données nécessaires à l'application. Le schéma 1 montre très bien que rien n'empêche d'ajouter un autre noeud à la liste.
- **taille modulable** : l'on peut aussi supprimer un élément de la liste. De ce fait, nous voyons que notre liste peut croître ou décroître à volonté. Ceci est d'autant plus intéressant que toutes ces modifications peuvent intervenir pendant l'exécution du programme, en fonction des besoins de l'utilisateur. Exemple : un répertoire téléphonique comporte trois noms. Représentons-les à l'aide de nos deux types de structures (cf. schéma 2).



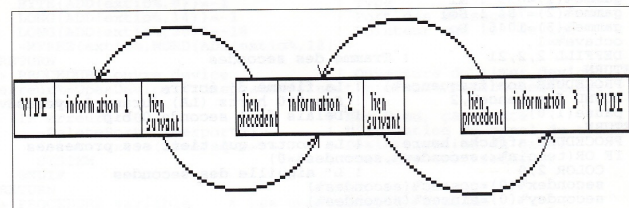
Jusque là, tout est correct. A présent, ajoutons un élément "Franck" (cf. schéma 3). Nous venons de voir à quel point il est facile d'ajouter un élément dans la liste. En revanche, le tableau n'accepte pas le nom supplémentaire, car il n'y a plus de place.

Supprimons maintenant deux éléments, "Bruno" et "Lucien" (cf. schéma 4). Bien que nous ayons supprimé deux prénoms, le tableau conserve sa taille initiale et les "cases" non utilisées occupent toujours de l'espace mémoire. Au contraire, la liste chaînée est réduite à son strict minimum. Les emplacements occupés par les noeuds supprimés ont été restitués au système.



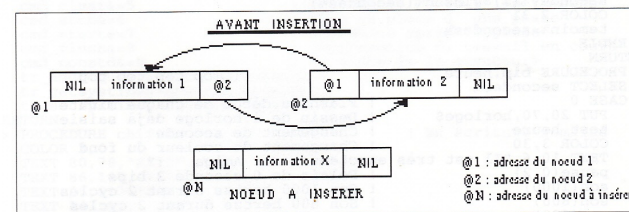
PRIMITIVES

Maintenant que les listes nous sont familières, nous allons étudier les différentes opérations que nous pouvons leur faire subir. Les primitives sont en pseudo-code et elles concernent les listes simplement et doublement chaînées.



La liste double ne diffère de la simple que par la présence d'un lien supplémentaire, qui relie chaque noeud à son prédécesseur, permettant ainsi de parcourir la liste dans les deux sens.

Insertion



- (1) `noeud2_adr <- noeud1.suivant;`
- (2) `noeud1.suivant <- nouveau_noeud;`
- (3) `nouveau_noeud.précédent <- noeud1;`
- (4) `nouveau_noeud.suivant <- noeud2_adr;`
- (5) `noeud2_adr.précédent <- nouveau_noeud;`

Les étapes (3) et (5) ne concernent pas les listes simplement chaînées.

HARDWARE : LA VOIX DE PAULA

Ah, quelle belle chose que d'avoir un ordinateur capable de générer autre chose que de vulgaires petits bips .. L'Amiga est, comme chacun sait, assez riche dans ce domaine puisqu'il dispose de quatre canaux sonores indépendants pour ravir vos oreilles en stéréo.

Et qui doit-on complimenter pour cela ? C'est Paula (et accessoirement Agnus pour les transferts DMA). Mais ses pouvoirs ne s'arrêtent pas à un simple "recrassage" de samples. En effet, cette diva est capable de moduler l'amplitude et la fréquence de sortie d'un ou plusieurs canaux sonores afin de produire les effets voulus. Et si cela ne vous suffit pas, alors vous pouvez toujours faire joujou avec le filtre passe-bas mis à votre disposition.

A QUOI TU JOUES, PAULA ?

Un son étant une vibration, il est défini par deux critères : son amplitude et sa fréquence. Cette dernière définira si un son est grave ou aigu (plus la fréquence est élevée, plus le son est aigu). L'amplitude moyenne définit la puissance d'un son (ou le volume), tandis que les variations de cette même valeur définissent le son lui-même. La table de données transmise à Paula sera donc une suite d'amplitudes instantanées. Cette table sera découpée en octets, ce qui autorise des amplitudes de +127 à -128. Paula aura bien entendu besoin d'autres renseignements sur le son avant de le jouer : son adresse, sa longueur, sa période (correspondant à la fréquence du signal) et son volume (force avec laquelle il sera joué). Mais voyons tout de suite les registres mis en cause.

AUD*LCH \$DFF0+0 Adresse du son (3 bits hauts)
AUD*LCL \$DFF0+2 Adresse du son (16 bits bas)

Ces deux registres peuvent bien entendu être initialisés en même temps de la façon suivante à l'aide d'une seule instruction 68000 **MOVE.L**. Notez que les données doivent se trouver en mémoire CHIP.

AUD*LEN \$DFF0+4 longueur du son (16 bits)

La longueur de l'échantillon est exprimée en mots. On remarque donc que le maximum est de 128 Ko... C'est déjà pas mal, non ?

AUD*PER \$DFF0+6 Période du son (16 bits)

Cela permet de fixer la rapidité de restitution de l'échantillon. Le nombre transmis dans ce registre s'exprime en cycles machines. Pour ceux qui préfèrent parler de fréquence (en octets par seconde), voilà la formule de conversion :

$$\text{PERIODE} = 1/(\text{FREQUENCE})/279,365.10^{-9}$$

Le dernier nombre n'est autre que la durée d'un cycle machine pour le 68000. Sachez de plus que le circuit audio a tout de même des limites du fait de sa synchronisation avec l'écran : sachant qu'il charge un mot de données sonores par ligne écran, on obtient une fréquence théorique maximale de (2 octets * 262.5 lignes/écran * 59,94 écrans/seconde), soit 31469 octets/seconde. Mais tout ceci reste théorique : comme le contrôleur n'a pas que ça à faire, la fréquence maximale est de 28867 octets/seconde, soit encore une période de 124 cycles. Si malgré cela, vous mettiez une valeur inférieure à 124 dans ce registre, Agnus n'aura pas forcément le temps de charger la donnée audio et c'est l'ancienne qui sera utilisée, d'où une déformation du son.

AUD*VOL \$DFF0+8 Volume du son (16 bits)

Juste une note pour ce registre : seuls les 6 bits inférieurs définissent le volume. Le volume maximal est donc de 63 et une valeur de 0 le coupe totalement. Les autres bits n'ont aucune utilité. Enfin, voici quelques notes supplémentaires sur ces adresses :

- elles sont uniquement accessibles en écriture ;
- le signe * représente le numéro du canal (0 à 3) ;
- le signe + signifie égale \$A pour le canal 0, \$B pour le canal 1, \$C pour le canal 2 et \$D pour le canal 3.

DIS PAPA, COMMENT CA MARCHE ?

Maintenant que vous connaissez les registres qu'il faut utiliser, on va pouvoir passer à la pratique. Nous allons débiter par une simple émission de son en mode automatique : Paula s'occupe de tout... ou presque.

Tout d'abord, on initialise de la façon indiquée les registres décrits plus haut, et ceci pour chaque canal à mettre en oeuvre. Pour lancer le(s) son(s), il suffit alors d'autoriser les canaux DMA nécessaires (bits 9 et 0 à 3 du registre **DMACON**). L'émission du son commencera aussitôt et se poursuivra jusqu'à ce qu'on la stoppe : Paula, après avoir lu le nombre de mots indiqué dans **AUD*LEN**, se remplacera automatiquement au début des données (hmmm, les belles digits bouclées).

Pour stopper toute émission sonore, il suffit d'effacer les bits de **DMACON** correspondant aux canaux à bloquer. Notez que si vous remettez en route ces canaux DMA, PAULA recommencera le son du début, à une exception près : si l'intervalle de temps écoulé entre la coupure et la remise en route est inférieur à 2 fois la période, alors la lecture se poursuivra de là où PAULA l'avait laissée (ce phénomène sera expliqué ultérieurement).

Arrivé ici, un rappel des bits important de **DMACON** n'est sans doute pas inopportuniste.

DMACON \$dff096 (écriture)
DMACONR \$dff002 (lecture)

Bit	Nom	Fonction
15	SET/CLR	Voir note
9	DMAEN	Lorsque ce bit est mis, tous les canaux DMA peuvent être utilisés. S'il est à 0, les canaux DMA sont bloqués
3	AUD3EN	En mettant ce bit à 1, on démarre la sortie audio du canal 3 (gauche). En effaçant ce bit, on interrompt la sortie
2	AUD2EN	Idem pour le canal 2 (droite)
1	AUD1EN	Idem pour le canal 1 (droite)
0	AUD0EN	Idem pour le canal 0 (gauche)

Note : lors d'une écriture, si le bit 15 SET/CLR est mis, tous les autres bits à 1 du mot transféré sont mis à 1 dans **DMACON**, ceux à 0 restant inchangés. Si SET/CLR est à 0, tous les bits à 1 de la source seront effacés dans la destination, ceux à 0 restant inchangés.

ACTION !

Voilà un petit exemple pour achever de vous convaincre de la facilité de la chose...

```
START_SON:
    move.l    #DATA_SON,$DFF0A0      ; ADRESSE
    move.w    #0001,$DFF0A4          ; LONGUEUR en mots
    move.w    #5000,$DFF0A6          ; PERIODE
    move.w    #0050,$DFF0A8          ; VOLUME
    move.w    #8201,$DFF096          ; Envoie canal 0
BOUCLE_WAIT:
    btst     #6,$BFE001; bouton gauche
```



```
bne.s BOUCLE_WAIT ; de la souris
move.w #0001,$dff096 ; SILENCE ! (*)
rts
```

```
DATA_SON:
dc.b 120,-120; Signal carré... Rien de plus simple
```

(*) Vous remarquerez que je n'ai pas mis le bit 9 de DMACON à 0. En règle générale, il vaut mieux s'abstenir, car cela interdirait tout travail du DMA. Pour être rigoureux, il faudrait le remettre comme il était avant.

OH, PAULA, TU ME FAIS DE L'EFFET !

On peut distinguer deux sortes de modulations possibles : la modulation d'amplitude (volume) et la modulation de fréquence (période).

La première pourra servir à faire des effets du type vibrato, tremolo, imitation du rapprochement ou de l'éloignement d'un son, etc... La seconde permet entre autres de simuler des accords. De plus, il est possible de combiner les deux modulations. Bref, les possibilités sont énormes. Seule ombre au tableau : pour moduler une voix, on est obligé d'utiliser deux canaux consécutifs.

Le premier (celui de plus basse numérotation) sert pour l'enveloppe et le second (celui de plus forte numérotation) sert pour le son de base. Les registres des canaux sont initialisés comme s'il s'agissait de sons ordinaires (il est toutefois inutile d'initialiser le volume du canal de modulation, puisque celui-ci ne sera pas joué). Par contre, pour le canal de modulation, la table de données n'est plus composée d'octets mais de mots représentant soit un volume (0 à 64) soit une période (avec les mêmes restrictions que celles citées plus haut). Si vous voulez faire une modulation en fréquence et en amplitude à la fois, la table doit contenir alternativement un volume et une période, dans cet ordre. Dans ce dernier cas, il y aura deux lectures au lieu d'une afin que les modulations se fassent simultanément et non l'une après l'autre. Pas besoin donc de toucher à la période pour compenser.

Une fois que vous avez initialisé correctement les registres de deux canaux consécutifs et fait votre table de modulation, il vous suffit de les "lier" pour la modulation à l'aide du registre *ADKCON* (qui sert aussi au pilotage des lecteurs de disquettes, alors soyez prudent).

ADKCON \$dff09E (écriture)
ADKCONR \$dff010 (lecture)

Bit	Nom	Fonction
15	SET/CLR	Même fonctionnement que pour DMACON
7	ATPER3	Si ce bit est mis, la voix 3 ne module la période d'aucune autre voix (cela a pour effet de la couper)
6	ATPER2	Si ce bit est mis, la voix 2 module la période de la voix 3
5	ATPER1	Même chose avec les voix 1 et 2
4	ATPER0	Même chose avec les voix 0 et 1
3	ATVOL3	Si ce bit est mis, la voix 3 ne module le volume d'aucune autre voix (cela a pour effet de la couper)
2	ATVOL2	Si ce bit est mis, la voix 2 module le volume de la voix 3
1	ATVOL1	Même chose avec les voix 1 et 2
0	ATVOL0	Même chose avec les voix 0 et 1

Note : les bits 7 et 3 ont le même effet : ils coupent tous les deux la voix 3, qui est la plus haute et ne peut donc pas en moduler d'autre. Bien qu'aucun son ne sorte, les données continuent à être chargées normalement par Agnus. Ces deux bits identiques n'ont en fait aucune utilité. On peut à la limite s'en servir pour créer un morcellement de la sortie audio dans la voix 3 (un coup rien, un coup de la musique et ceci à la fréquence avec laquelle vous mettez et enlevez ce bit). Mais il existe d'autres méthodes pas plus contraignantes pour faire cela

(modulation d'amplitude avec une table composée de seulement deux valeurs, à savoir 0 et 64).

Une fois que vous avez lié les deux canaux choisis, il ne reste plus qu'à autoriser le transfert DMA sur ces deux canaux avec DMACON (comme pour des sons ordinaires). L'effet sera le suivant : le son de base sera joué sur la voix la plus haute des deux. Il utilisera comme volume ou fréquence (ou les deux) les mots chargés à chaque période du canal de modulation. Notez que ces mots sont chargés à l'adresse du volume où de la fréquence du canal modulé. Si vous arrêtez la modulation, n'oubliez pas de réinitialiser le volume ou la fréquence des canaux modulés et de couper les canaux DMA des canaux modulants.

Une dernière note : rien n'empêche de moduler un canal "à la main" en tripataillant le volume ou la fréquence du canal voulu. Cela évitera d'utiliser deux canaux, mais adieu synchronisation et facilité d'emploi (sans compter que le 68000 sera alors utilisé en permanence pour le son). Enfin, ça peut toujours servir...

ACTION 2, LE RETOUR !

Et voilà l'exemple tant attendu qui prouve une fois encore que ce n'est pas si compliqué que ça en a l'air. La suite de ce passionnant article dans notre prochain numéro.

```
START_SON:
; CANAL DE MODULATION
; MODULATION en fréquence
move.l #DATA_MOD,$dff0A0 ; ADRESSE
; move.l #DATA_MOD2,$dff0A0 ;-> pour mod. amplitude
; move.l #DATA_MOD3,$dff0A0 ;-> pour les deux mod.

move.w #0008,$dff0A4 ; LONGUEUR en mots
; move.w #0016,$dff0A4 ;-> pour les deux mod.

move.w #40000,$dff0A6 ; PERIODE

; SON DE BASE
move.l #DATA_SON,$dff0B0 ; ADRESSE
move.w #0001,$dff0B4 ; LONGUEUR en mots
move.w #5000,$dff0B6 ; PERIODE
move.w #0050,$dff0B8 ; VOLUME

move.w# $8010,$dff09E ; On lie 0 et 1
; move.w #8001,$dff09E ;-> pour mod. amplitude
; move.w #8011,$dff09E ;-> pour les deux mod.

move.w #8203,$dff096 ; Envoit 0 et 1

BOUCLE_WAIT:
btst #6,$BFE001 ; bouton gauche
bne.s BOUCLE_WAIT ; de la souris
move.w #0011,$dff09E ; FIN MODULATION
move.w #0003,$dff096 ; SILENCE !
rts
```

```
DATA_SON:
dc.b 120,-120; Toujours un signal carré...

DATA_MOD:
dc.w 1000,2000,3000,4000,5000,6000,7000,8000
; Ici, il s'agit d'un signal triangulaire

DATA_MOD2:
dc.w 2,8,10,20,30,40,50,60

DATA_MOD3:
dc.w 2,1000,8,2000,10,3000,20,4000
dc.w 30,5000,40,6000,50,7000,60,8000
```

Par Philippe Rivailon

TRANSACTION : LES PLASMAS

*Ces derniers temps, les plasmas
ont déferlé sur vos écrans...
Encore un effet que seul l'Amiga
était capable d'offrir, et qui se
révèle d'une étonnante simplicité.*

Le principe est simple : il suffit de créer une énorme CopperList qui ne contient quasiment que des instructions de changement de couleur. Classiquement, les changements de couleur se font au début de la ligne de raster (ce qui donne les effets de "barres de Copper" bien connus), mais ici, on change la couleur plusieurs fois durant la même ligne (en l'occurrence 53 fois).

Il n'y donc aucune image "réelle" affichée (0 bitplanes), seulement des changements de couleur, mais le résultat est saisissant ! De plus vous pouvez toujours ajouter une image par dessus ou par dessous ces plasmas.

Les couleurs mises dans la CopperList sont prises dans un tableau (tabcol) puis rangées dans la CopperList à l'aide du Blitter pour plus de rapidité. Deux mouvements de sinus sont appliqués lors du transfert, ce qui donne cet effet de vague sur la composante x. Une double superposition de sinus est appliquée aussi sur y, en jouant sur le retard vidéo à chaque ligne. Le mouvement est donc composé de quatre sinus, deux sur X et deux sur Y. Il y a donc huit paramètres qui déterminent le mouvement : la vitesse de déplacement et le pas entre deux sinus consécutifs, et ce pour les quatre sinus, ce qui nous fait bien huit paramètres.

Je vous conseille de modifier ces paramètres et d'observer le résultat. Vous pouvez aussi modifier le dégradé de couleurs pour en mettre un de votre cru.

Il faut noter qu'il y a deux CopperLists, une qui est affichée tandis que l'autre est modifiée. Et à la prochaine trame, on inverse ces deux CopperLists. C'est l'équivalent du "double buffering" pour les CopperLists.

Pour créer la table de sinus (qui est un tableau de 256 octets) il suffit de taper le petit programme GFA suivant :

```
OPEN "o",#1,"datasin"
FOR i=0 TO 255
    val=INT(127*SIN(i*2*PI/256))
    OUT #1,val
NEXT i
```

Il est facilement adaptable pour l'AmigaBasic. Il ne reste plus qu'à assembler le programme Plasma.s et à l'exécuter.

Bon coloriage !

```
; #####
; # PLASMA T.Landsburg 91 #
; #####
; A assembler sous devpac
opt c+,o+,ow-

NB_COL_COP=53 ; Nb de changements de couleurs par ligne
NB_LIG_COP=264 ; Nombre de lignes de la plasma
NB_BYTE_PLASMA=(NB_COL_COP+1)*4
taillecop=20+(NB_COL_COP+2)*4*Nb_LIG_COP; Taille de la copperlist
NB_DEG=15*3 ; Utilise pour la creation de degrade
NB_REP=10 ; idem
CUSTOM=$dff000 ; Registres hardware

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

; DEBUT DU PROGRAMME
start: lea b(pc),a4 ; a4 POINTE TOUJOURS SUR LA BASE!
lea graphname(pc),a1; Ouverture de la graphics
CALLEXECOpenLibrary
move.l d0,graphbase-b(a4)
bsr initcop ; initialise la copper list
bsr init_col ; initialisation des couleurs de
```

```
; la plasma
lea CUSTOM,a5 ; Base des coprocesseurs
move.w intena(a5),save_intena-b(a4)
or.w #$c000,save_intena-b(a4)
move.w #$7fff,intena(a5)
move.w #$0020,dmacon(a5) ; controle DMA
move.w #$8400,dmacon(a5) ; Bltpri
CALLEXECForbid
move.b #$10000111,$bfd100 ; ARRET DU DRIVE (BEURK!)
lea CUSTOM,a5 ; a5 pointe les registres hard

waitbout: ; BOUCLE PRINCIPALE
sync: btst #0,vposr+1(a5)
bne.s sync
sync1: cmp.b #40,vhposr(a5) ; Synchronisation
bne.s sync1
movem.l p_coplist(pc),d0/d1
exg d0,d1
movem.l d0/d1,p_coplist-b(a4)
move.l d0,copllc(a5)
clr.w copjmpl(a5)
bsr move_plasma
btst #6,$bfe001 ; Test le bouton de la souris
bne waitbout
out: move.w save_intena-b(a4),intena(a5) ; interruption
move.l graphbase(pc),a0 ; Restaure l'ancienne copperlist
move.l oldcop-b(a4),$32(a0)
move.l $26(a0),copllc(a5)
CALLEXECPermit
move.l graphbase(pc),a1 ; On referme la librairie
CALLEXECCloseLibrary
move.w #$0400,dmaconr(a5)
move.w #$8020,dmaconr(a5)
moveq #0,d0
rts
save_intena:dc.w 0
waitblit:macro
waitblit:@
    btst #14,dmaconr(a5)
    bne waitblit\@
endm
graphname:dc.b 'graphics.library',0
even
graphbase:dc.l 0
; INITIALISATION DE LA COPPER LIST
initcop:
    lea coplist,a0
    move.w #NB_LIG_COP-1,d7
    move.w #$2a31,d0 ; Premiere ligne de l'ecran
next_lig_plasma:
    move.w d0,(a0)+; le wait
    move.w $ffff,(a0)+
    move.l #$01020000,(a0)+ ; decalage horizontal
    add.w #$0100,d0 ; prochaine ligne
    moveq.w #NB_COL_COP-1,d6; on met le changements
    move.w d7,d1 ; de couleur pour une ligne
next_col_plasma:
    move.l $1800000,(a0)+
    dbf d6,next_col_plasma
    dbf d7,next_lig_plasma
    move.l $01000000,(a0)+; 0 bitplane
    move.l $ffffff,(a0)+; fin de la copperlist

    lea coplist,a0
    lea coplist+taillecop,a1
    move.w #taillecop/4-1,d7
next_recopie:
    move.l (a0)+(a1)+
    dbf d7,next_recopie

    move.l graphbase-b(a4),a0
    move.l $32(a0),oldcop-b(a4)
    rts
; MOVEMENT DES PLASMAS
move_plasma: ; C'est le coeur du programme
    move.l p_coplist(pc),a0; Pointeur sur la copperlist
    lea tabsin(pc),a2 ; Tableau de sinus
    move.w $fff,d5 ; -taille de la table de sinus
; D'abord on va creer les decalages sur y
    move.w p_sinusyl(pc),d1; premiere wave sur y
    add.w speed_sinusyl-b(a4),d1
    move.w d1,p_sinusyl-b(a4)
    move.w p_sinusyl2(pc),d2 ; premiere wave sur y
    add.w speed_sinusyl2-b(a4),d2
    move.w d2,p_sinusyl2-b(a4)
    move.w #NB_LIG_COP-1,d7 ; Nombre de lignes
next_lig_plasma2
    and.w d5,d1
    and.w d5,d2
    move.b (a2,d1.w),d3 ; decalage sinus
    ext.w d3
    move.b (a2,d2.w),d4 ; decalage sinus
    ext.w d4
    add.w d4,d3 ; On ajoute les deux sinus
    asr.w #4,d3 ; decalage y de faible amplitude
    add.w $20,d3
    bset #0,d3
```


ANCIENS NUMEROS DE L'ANT

Tiens, si on profitait du fait que Thomas a écrit un article trop court, pour établir une compilation des sommaires des anciens numéros de l'ANT depuis l'Indépendance ? C'est parti.

Numéro 20 (mars 91)

AMOS	Les extensions
Concours	Les fontes vectorielles
Demos	Les rasters verticaux
ExecBase	Les bibliothèques
Interview	Franck Ostrowski
SubWay	Les écrans ; le console.device
ToolBox	Devpac II vs K-Seka
Utilitaire	Le Screen Blanker

Numéro 21 (avril 91)

AMOS	Les interruptions
CATS	Leçon de savoir-vivre
Concours	Un ciel étoilé
Demos	Bobs & Lissajous
ExecBase	Utiliser amiga.lib ; le 68000 de A à Z
SubWay	Les fenêtres Intuition ; le console.device (II)
ToolBox	DICE
Utilitaire	Le SuperShell

Numéro 22 (mai 91)

AMOS	Les banques de mémoire
ARExx	La voie royale
CATS	Le débogage
Concours	De toutes les couleurs
Debug	Alea Jacta Est
Demos	La 3D
ExecBase	Utiliser amiga.lib (II) ; le 68000 de A à Z (II)
SubWay	Les Gadgets ; les ressources
ToolBox	Le compilateur GFA Basic ; PCQ
Utilitaire	EasyWindow

Numéro 23 (juin 91)

AMOS	Les banques de mémoire (II) ; les DaisyDemOS
ARExx	La voie royale (II)
CATS	Question de style
Concours	Scroll en rouleau
Demos	Rasters ondulants
ExecBase	Melange C/Assembleur : le 68000 de A à Z (III) ; le DOS et ses devices
Hardware	Les CIAs 8520
SubWay	Les Ressources (II) ; liaisons Intuition/Graphics
ToolBox	SAS Lmk ; la Bible de l'Amiga ; 3 calculatrices du DP
Utilitaire	Le traqueur de virus

Numéro 24 (juillet/août 91)

AMOS	La musique
ARExx	La voie royale (III)
CATS	Question de style (II)
Concours	De mal en pl
Demos	La 3D faces pleines
ExecBase	Du C à l'assembleur ; le 68000 de A à Z (fin)
Hardware	Les disquettes
SubWay	La communication inter-tâches ; les menus Intuition
ToolBox	ReqLib & ParNet
Utilitaire	ScreenSaver

Numéro 25 (septembre 91)

AMOS	Trucs en vrac ; les DaisyDemOS
ARExx	La voie royale (IV)
CATS	Le multitâche sur Amiga
Demos	La 3D faces pleines (II)
ExecBase	Programmer ré-entrant ; la mémoire du monde
Hardware	Les disquettes (II)
SubWay	La communication inter-tâches (II) ; le GELS
ToolBox	AMXLisp
Transactor	Gérer les joysticks
Utilitaire	L'espion de la mémoire

Numéro 26 (octobre 91)

AMOS	Trucs en vrac (II) ; les DaisyDemOS
ARExx	Travail pratique
Demos	Le trial-playfield
ExecBase	Déboguer avec MonAm2 ; les interruptions sous Exec
Hardware	Le clavier
SubWay	Les layers ; le GELS (II)
ToolBox	Draco
Transactor	Les rouleaux de lumière
Utilitaire	NewPic

Numéro 27 (novembre 91)

AMOS	Un interpréteur d'expressions
ARExx	Travail pratique (II)
Demos	Scrolling "kick-off"
ExecBase	Les overlays ; les interruptions sous Exec (II) ; la mémoire du monde (II)
Hardware	Les disquettes (complément)
SubWay	Les layers (II)
ToolBox	ADebog ; GAWK
Transactor	Le bon octant
Utilitaire	ExecMaster

Numéro 28 (décembre 91)

AMOS	Les belles courbes ; les DaisyDemOS
ARExx	Travail pratique (III)
Demos	De toutes les couleurs
ExecBase	Les overlays (II)
Hardware	Le clavier (II)
Routinettes	NoEvent
SubWay	Le GELS (III) ; les layers (III)
ToolBox	Flex
Transactor	Sinus-Scroll
Utilitaire	Le transformeur ; ExecMaster (II)

```

move.b d3,1(a0) ; on le met dans la copperlist
add.w step_sinusyl(pc),d1
add.w step_sinusyl2(pc),d2
lea NB_BYTE_PLASMA(a0),a0
dbf d7,next_lig_plasma2
; Maintenant les decalages sur X
waitblit ; attend le blitter
move.w #$09f0,bltcon0(a5)
clr.w bltcon1(a5) ; mode suite
move.l #-1,bltafm(a5) ; Masque a -1
move.w #NB_BYTE_PLASMA-2,bltmod(a5) ; Modulo
clr.w bltamod(a5) ; En source pas de modulo
move.w #((NB_LIG_COP)<<6+1),d6 ; Taille=une colonne

move.l p_coplist(pc),a0
lea tab_col_plasma+128,a1
move.w p_sinusxl(pc),d1 ; premiere wave sur x
add.w speed_sinusxl(pc),d1
move.w d1,p_sinusxl-b(a4)
move.w p_sinusx2(pc),d2 ; deuxieme wave sur x
add.w speed_sinusx2(pc),d2
move.w d2,p_sinusx2-b(a4)
lea 6+4(a0),a0 ; a0 = debut des couleurs
moveq.w #NB_COL_COP-2,d7
move.w step_sinusxl(pc),d0
next_col_plasma2
add.w d0,d1
add.w step_sinusx2(pc),d2
and.w d5,d2
and.w d5,d1
move.b (a2,d1.w),d3 ; decalage sinus
ext.w d3
move.b (a2,d2.w),d4 ; decalage sinus
ext.w d4
add.w d4,d3 ; On ajoute les deux sinus
asr.w #1,d3 ; d3 = offset dans la table
lea (a1,d3.w),a3
waitblit
move.l a3,bltapt(a5) ; Source=table de couleur
move.l a0,bltapt(a5) ; Destination=copperlist
move.w d6,bltsize(a5) ; Taille=une colonne
lea 4(a0),a0
dbf d7,next_col_plasma2
rts

; Variables qui contiennent les donnees pour la plasma
p_sinusxl:dc.w 0
p_sinusx2:dc.w 0
p_sinusyl:dc.w 0
p_sinusyl2:dc.w 0
step_sinusxl:dc.w -3
speed_sinusxl:dc.w -1
step_sinusx2:dc.w -4
speed_sinusx2:dc.w -1
step_sinusyl:dc.w 2
speed_sinusyl:dc.w 1
step_sinusyl2:dc.w -3
speed_sinusyl2:dc.w -5

; Cree le degrade de couleur pour la plasma
init_col:
lea tab_col_plasma,a0 ; Table des couleurs
move.w #$fff,d0
move.w #-$110,d1 ; modifie la composante verte
bsr cree_one_deg ; On cree trois degradés
move.w #$100,d1 ; idem avec la composante rouge
bsr cree_one_deg
move.w #-$1,d1 ; et la composante jaune
bsr cree_one_deg
rts

cree_one_deg:
move.w #NB_DEG/3-1,d7 ; Cree un degrade
12: move.w #NB_REP-1,d6 ; Nombre d'iterations
11: move.w d0,(a0)+ ; Nombre de fois ou l'on garde
dbf d6,11 ; la meme couleur
add.w d1,d0 ; Prochaine couleur
dbf d7,12 ; Tant qu'il y a des couleurs
rts

b:
oldcop: dc.l 0
p_coplist: dc.l coplist
p_coplist_old: dc.l coplist+taillecop
dir: dc.w -1

; Le fichier datasin est cree avec le programme GFA annexe
tabsin:incbin datasin
; Section qui contient les donnees
section plasma,BSS_C
tab_col_plasma:
dcb.w NB_DEG*Nb_REP
coplist:dcb.b taillecop*2

```

Par Thomas Landsburg

TOOLBOX : LES CREATEURS D'IMAGES

Notre ToolBox de ce mois complète fort judicieusement la rubrique SubWay, puisqu'elle présente les meilleurs DP spécialement destinés à créer des programmes sources C à partir d'images IFF.

Le programme illustrant la rubrique SubWay utilise la sortie produite par un tel DP pour inclure directement dans le source, la définition du Bob (NDLR : pour des raisons de mise en page, j'ai modifié un tout petit peu le look de la sortie produite, mais bon, c'est pas si grave que ça). Il en existe plusieurs, dont voici illico presto ceux que nous pensons être les meilleurs. Sans oublier bien entendu le Convertisseur de Denis Genot, promu Utilitaire du mois dernier.

ILBM2IMAGE

Sous ce répertoire de la disquette Fish 190 se cachent trois petits utilitaires, d'un emploi fort aisé, permettant de créer et d'afficher des structures Image, au sens Intuition du terme.

Pour vous en servir, rien de plus simple. Vous dessinez d'abord l'image voulue avec votre programme de dessin favori et vous la sauvegardez au format IFF (le fait que ce soit une page entière ou une simple brosse n'a aucune importance). Vous êtes dès lors prêt à traiter votre fichier IFF suivant trois méthodes différentes.

Imaginons pour commencer que vous désiriez créer une Bitmap correspondante à votre dessin, pour créer par exemple un Bob. Vous utiliserez dans ce cas ILBM2image en utilisant la syntaxe suivante :

```
ILBM2image toto ; où toto est le nom de votre fichier IFF
```

L'utilitaire générera alors un fichier toto.c qui n'est autre que la déclaration du tableau de UWORD correspondant à la Bitmap :

```
UWORD toto[] = {
    0x3e91, 0xa038, 0x1590, 0xe1e0,
    0x33ff, 0xc03c, 0x1f01, 0xe2c0,
    0x0, 0x0, 0x700, 0x0
};
```

Si maintenant vous désirez afficher sous Intuition une structure Image, vous aurez recours au programme ILBM2func, qui non seulement générera la Bitmap associée, mais également la structure Image correspondante et la fonction d'affichage dans une fenêtre dont le nom est Window. La fonction système utilisée pour l'affichage est bien sûr DrawImage().

```
ILBM2func toto ; où toto est le nom de votre fichier IFF
```

Ce qui donnera par exemple :

```
#include <exec/types.h>
#include <intuition/intuition.h>

/*----- bitmap : w = 64, h = 3 ----- words = 12 ----*/
UWORD toto[] = {
    /*----- plane # 0: -----*/
    0x3e91, 0xa038, 0x1590, 0xe1e0,
    0x33ff, 0xc03c, 0x1f01, 0xe2c0,
    0x0, 0x0, 0x700, 0x0
};

struct Image totoimage = {
    0,0,
    64, 3, 1,
    &toto[0],
    0x1f,0x00,
    NULL,
};

totoimagefunc()
{
```

```
extern struct Window *Window;

DrawImage(Window->RPort,&totoimage,100,10);
return(0);
}
```

Que nous manque-t-il ? Les plus perspicaces d'entre vous auront remarqué qu'il nous manque la palette ou table des couleurs associée à notre dessin. Et c'est là qu'intervient le dernier utilitaire, à savoir GetCMap. GetCMap analyse un fichier IFF à la recherche de sa palette de couleurs et génère un fichier C déclarant un tableau de mots (UWORD), soit :

```
GetCMap
```

Le logiciel vous demande alors le nom du fichier ILBM (IFF)

```
ILBM File name ?
```

Saisissez toto. GetCMap demande alors le nom du fichier à générer :

```
C file name ?
```

Par exemple : toto.c

Finalement, le résultat obtenu sera de la forme :

```
/* Image toto
Width: 64
Height: 58
Depth: 4 */

USHORT map[] = {
    0x0000,
    0x0eca,
    0x0f00,
    0x0f16,
    0x0f1c,
    0x0c2e,
    0x072e,
    0x033e,
    0x066e,
    0x089d,
    0x0bbd,
    0x0aac,
    0x099b,
    0x099a,
    0x0889,
    0x0444
};
```

Comme vous pouvez le voir, l'utilisation de ces programmes est enfantine, pour un résultat nécessitant des heures d'un travail fastidieux lorsqu'accompli manuellement.

SPRIGHT V1.2

Spright est un éditeur de sprites, qui permet de sauvegarder vos oeuvres sous formes de fichiers sources. Il s'agit d'un programme shareware que vous pourrez trouver sur la disquette Fish 484.

Spright permet de créer des sprites d'une hauteur variant de 1 à 200 pixels. Bien que la largeur d'un sprite soit limitée à 16 pixels, le logiciel possède un mode double, permettant d'éditer deux sprites côte à côte. Il permet également d'éditer des sprites attachés, pour obtenir par superposition des bitplanes, un sprite unique de 15 couleurs + transparence.

L'interface, très orientée Intuition, comporte deux boîtes : la première au centre de l'écran, permet de dessiner un sprite grossi 4 fois (pour faciliter le dessin). La deuxième boîte, sur le côté, affiche en permanence l'image taille réelle. Les fonctionnalités d'édition sont nombreuses.

- **DUAL** : ce mode utilise deux sprites, placés côte à côte. Des couleurs différentes peuvent être utilisées pour chaque sprite si le gadget ATTACH n'est pas sélectionné.

- **SWITCH** : en mode DUAL, ce gadget permet d'éditer alternativement l'un ou l'autre des deux sprites.

• **ATTACH** : permet d'attacher un sprite à un autre. Ceci permet d'utiliser, par superposition des bitplanes quinze couleurs au lieu de trois (transparence exclue).

• **SAVE/LOAD** : sauvegarde/chargement d'un sprite dans l'éditeur.

• **PRINT** : impression de l'image zoomée.

• **SNAPSHOT/RESTORE** : sauvegarde en mémoire d'un sprite. Ce sprite peut ensuite être rappelé à tout moment. Cette fonction est très utile en cas de fausse manipulation, ou si l'on veut effectuer un essai sans risquer de détruire l'oeuvre naissante. On utilise RESTORE pour récupérer l'image "snapshotée".

• **RGB Slider Gadget** : ces trois gadgets permettent de régler les intensités RVB de chaque couleur.

• **COPY** : permet de copier une zone de la boîte d'édition à une autre place.

• **ROTATE** : autorise la rotation du Sprite créé, soit vers la gauche soit vers la droite, par tranche de 90 degrés.

Plus, bien évidemment, les commandes standard de déplacement dans la page de dessin.

Après avoir dessiné un sprite dans l'éditeur, il faut utiliser la fonction SAVE qui va générer le source en C dont voici un exemple.

```
/* The sprite->height was 16 */
WORD titi_Colors[32] = {
    0x000, 0xffff, 0xf00, 0xff0,
    0x0f0, 0x00f, 0xf00, 0xff0,
    0x0ff, 0x0f0, 0x00f, 0xf0f,
    0xcc0, 0xdd0, 0xee0, 0xfff,
    0x754, 0xfff, 0xf00, 0xff0,
    0x0f0, 0x00f, 0xf00, 0xff0,
    0x0ff, 0xfff, 0xf00, 0xff0,
    0x0f0, 0x00f, 0xf00, 0xff0
};

WORD titi_data[36] = {
    0, 0,
    0x0000, 0x0000,
    0x0000, 0x0000,
    0x0000, 0x7800,
    0x0000, 0x4800,
    0x0000, 0x4800,
    0x0210, 0x7800,
    0x0210, 0x7800,
    0x0310, 0x4800,
    0x029f, 0x480f,
    0x025f, 0x480f,
    0x0236, 0x0006,
    0x0216, 0x0006,
    0x0216, 0x0006,
    0x0006, 0x0006,
    0x0006, 0x0006,
    0x0000, 0x0000,
    0x0000, 0x0000
};
```

Comme l'indique le commentaire, la hauteur du sprite est de 16 pixels. Spright sauvegarde à la fois la table des couleurs et l'image associée, qui je vous le rappelle, ne possède pas du tout la même organisation que celle d'une image Bitmap.

IMAGE EDITOR V2.4

Ce programme, comme son nom l'indique, permet d'éditer directement des images afin de les sauvegarder sous forme de fichier source, en vue d'une inclusion dans un programme quelconque.

Le lancement d'Image Editor s'effectue sous shell en tapant simplement im24 et [Return]. Pour le quitter, tapez q ou cliquez dans le gadget de fermeture. Un requester apparait alors, vous demandant une confirmation, que vous lui donnerez en cliquant sur [O.K].

L'utilisation d'Image Editor est des plus simples puisqu'une fois l'image dessinée, il suffit de sélectionner le gadget Disk pour

sauvegarder votre travail sous la forme d'un source C ou, c'est là sa grande originalité, sous la forme d'un source assembleur.

Au niveau de l'interface utilisateur, l'accès aux différentes fonctions peut se faire, soit grâce à la souris, soit par des raccourcis clavier, indiqués directement sur chaque gadget (ex: d pour le gadget Disk). Il possède une ergonomie correcte.

Au niveau des fonctionnalités, Image Editor, permet de dessiner des images d'une taille maxi de 166*58 en haute résolution ou 83*58 en basse. Le nombre de couleurs possible est de 16 au maximum. Outre les possibilités classiques que nous sommes en droit d'attendre d'un tel programme (tracé, gomme, remplissage de surface, choix des couleurs...), Image Editor permet également de charger une Image IFF ILBM ou simplement la palette de couleurs qui lui est associée.

Petite déception toutefois, il faudra indiquer au pixel près la largeur et la hauteur de l'image que vous allez charger, ainsi que son nombre de Bitplanes. Cette complication inutile est vraiment dommage, d'autant que toutes ces informations sont codées, bien évidemment, dans l'entête du fichier IFF. On se demande où voulait en venir le programmeur...

Un examen rapide du source, montre la qualité et le soin apportés à la programmation. Aucune astuce n'est en effet utilisée et le programme se sort sans plantage de tous les problèmes qu'il peut rencontrer (comme un manque de mémoire par exemple). Je vous conseille de l'étudier, il en vaut la peine.

Ce programme est disponible dans le domaine Public, et plus précisément sur la disquette Fish 355. Il s'agit d'un logiciel Shareware.

ET LES AUTRES...

Il existe bien entendu d'autres programme qui réalisent, avec plus ou moins de bonheur, ce travail. Je citerai, à titre informatif, les programmes suivants :

CITAS Version 2.0 (Fish 475)

Ce programme convertit une image IFF en source assembleur ou C en vue de son pilotage par le Blitter.

IFF2Src (Fish 420)

IFF2src convertit des images IFF en source C ou assembleur, avec génération des bitplanes, masques blitter, tables des couleurs. Il ne fonctionne que sous AmigaDos V 2.0.

BI (Fish 375)

Ce petit programme convertit une brosse IFF ILBM en source C, tout simplement.

IFF2C (Fish 316)

Ce convertisseur d'image IFF en source C possède deux fonctionnalités intéressantes, la génération de commentaires associés à l'image et un calcul automatique des champs PlanePick et PlaneOnOff, nécessaires à l'utilisation de l'image par le GELS.

C'EST FINI

Dans notre prochain numéro, nous reprendrons le cours normal de cette rubrique. Yacc sera bien entendu de la partie, comme promis la dernière fois. D'ici là, passez de bonnes fêtes de fin d'année.

Par Herr Doktor von GlutenStimmellmDorf

DEMOS : SCROLL TEXT EN ROULEAU

Le but de notre programme du mois est de créer l'effet d'un texte s'enroulant autour d'un tube transparent. Un effet qui commence certes à dater, mais qui reste toujours autant apprécié...

Dans le source ci-dessous, j'en ai même fait un double pour des raisons quelques peu inattendues. On peut s'imaginer que j'ai agis ainsi pour me compliquer la tâche par plaisir, ou pour rendre l'effet plus efficace encore, mais en fait il n'en est rien : je n'ai fait cela uniquement que pour gagner du temps-machine. Là, surprise dans l'auditoire... En effet comment puis-je gagner du temps en ajoutant des éléments à l'écran ? Eh bien voilà la substantifique moëlle de cet article : je vais, dans un premier temps, vous expliquer comment je procède et vous en comprendrez les raisons à la fin (suspense insoutenable).

Tout d'abord, je vous conseille de faire chez vous un petit croquis de façon à mieux percevoir la technique utilisée, car elle est relativement dure à expliquer. Je définis une petite fenêtre dans laquelle on fera scroller le texte. Le scroll est assez lent, un pixel par VBL. Cela vient du fait qu'à chaque VBL, j'affiche une colonne de un pixel de large appartenant à un caractère puis je fait scroller toute la fenêtre de un pixel vers le haut et de un pixel vers la gauche. Avant le scroll, on capture la ligne située tout en haut de la fenêtre et on la mémorise quelque part pour pouvoir ensuite la déposer tout en bas, le vide laissé par le scroll horizontal étant comblé par le prochain caractère. Au fil des VBLs, le texte va défiler à 45 degrés dans la fenêtre et une fois arrivé en haut, va repasser en dessous grâce à la ligne supérieure, que l'on fait descendre à chaque fois. Ainsi, avec le temps, on obtient une fenêtre hachurée sur toute sa largeur par du texte orienté à 45 degrés.

La foule haletante s'impatiente, se demandant comment cela peut nous servir à construire un scroll en rouleau (c'est fou, je viens de m'apercevoir que j'ai raté une vocation d'écrivain à suspense...).

Ensuite, je fais une CopperList qui définit un petit écran en couleur avec 2 plans. Un plan sera consacré à la partie rouge du tube, c'est-à-dire "l'avant", l'autre sera consacré à la partie bleue, représentant "l'arrière" du tube. Grâce au Copper, on peut changer à tout moment le pointeur écran de chaque bitplan, on va se servir de cet avantage pour déformer le plan avant de manière à donner l'impression qu'il épouse la forme d'un tube. On fait cela grâce à une belle formule mathématique écrite dans le programme. Pour embellir et renforcer la crédibilité de l'illusion, on change la couleur du texte sur le tube en calculant la distance entre le spectateur et le texte. Ainsi, pour la partie avant, plus le texte est près du centre plus il est clair, et inversement pour la partie arrière.

Le scroll ne s'effectuant que sur un plan et sur une partie d'écran relativement faible, le temps d'exécution est peu important. On obtient un effet pas trop médiocre pour peu de temps machine. Il est loin d'être suffisant pour une démo, mais peut sans honte être placé dans un slide show, par exemple.

La phrase du mois, qui a pour auteur Douglas Adams : "l'infinité est tout bonnement si énorme qu'en comparaison, l'énormité paraît franchement rikiki".

*** SCROLL EN ROULEAU DE JEROME ETIENNE POUR L'ANT ***

```
; reg circuits specialises
dmacon = $096
dmaconr = $002
vposr = $004
vhposr = $006
```

```
; reg color
color00 = $180
```

```
; reg blitter
bltcon0 = $040
bltcon1 = $042
bltcopt = $048
bltcopt1 = $04a
bltbpt = $04c
bltbpt1 = $04e
bltapt = $050
bltapt1 = $052
bltdpt = $054
bltdpt1 = $056
bltmod = $060
bltmod1 = $062
bltamd = $064
bltamd1 = $066
bltsize = $058
bltadat = $070
bltadat1 = $072
bltad1 = $074
bltawm = $044
bltalwm = $046
```

```
; reg copper
copilc = $080 ; adr de la 1' list
copjmpl = $088 ; saut de la liste 1
```

```
; reg bitplane
bplcon0 = $100 ; reg 0 du controleur de bitplane
bplcon1 = $102 ; 1 (valeur du scroll)
bplcon2 = $104 ; 2 (priorite entre sprite & playfield)
bpl1pht = $0e0 ; \pointeur sur le
bpl1pt1 = $0e2 ; >1er bitplan
bpl2pht = $0e4 ; pointeur sur le bitplane 2
bpl1mod = $108 ; valeur modulo pour les plans impairs
bpl2mod = $10a ; valeur modulo pour les plans pairs
diwstrt = $08e ; debut de la fenetre ecran
diwstop = $090 ; fin de la fenetre ecran
ddfstrt = $092 ; debut dma bitplan
ddfstop = $094 ; fin dma bitplan
```

```
; cia-a (use for mouse's bouton)
ciaapra = $bfe001 ; exec library base offset
openlibrary = -30-522 ; nom de la library, version/
al,d0
forbid = -30-102 ; suppression du multitasking
permit = -30-108 ; autorisation du multitasking
allocmem = -30-168 ; nb d'octets, specification /d0,d1
freemem = -30-180 ; pointeur mem,nb d'octets /a1,d0
chip = $2 ; for chip ram (cf:allocmem)
clear = $10000 ; for clear the memory
```

```
; graphicbase
startlist = 38
; graphic library base offset
ownblitter = -30-426
disownblitter = -30-432
```

```
; autres labels
key = $bfec001
custom = $dfff000
execbase = 4
```

```
coplist_size = 4*2000
char_x = 32
char_y = 32
fnt_width = 40
nb_char_per_fnt = 50
char_width = char_x/8
fnt_size = nb_char_per_fnt*char_y*char_width
nb_char_per_line = fnt_width/char_width
```

```
***** def macros
allocmem: macro
move.l #\1,d0 ; demande l'allocation de $1 octets
move.l #\2,d1 ; #\2 type de ram sollicitee
jsr allocmem(a6)
move.l d0,\3 ; \3= adr ds la mem recuperee
endm ; si erreur d'allocation => MEGA BUG !!!!
```

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

```
*****
vsync: macro
.wait_sync@:
move.l vposr(a5),d0
and.l #$1fff,d0
cmp.l #$0c000,d0
bne.s .wait_sync@
endm
```



```

*****
    bsr      main_init
    bsr      init_roller

main_loop:
    vsync

    move.w   #$fff,color00(a5)
    bsr      mobify_roller
    bsr      wait_blt

    clr.w    color00(a5)

    move.b   key,d0          * \
    not      d0              * >capture the key
    ror.b    #1,d0           * >and put is code RAW in d0
    cmp.b    #$45,d0         * tst if esc is pressed
    beq      init_end        * if no loop again
    bra      main_loop

*****
roller_x      = 352
roller_y      = 3*32
plane_width_roller = roller_x/8
roller_plane_size = plane_width_roller*(roller_y+10)
mobify_roller:
    bsr      wait_blt
    clr.w    bltcon1(a5)
    move.w   #0,bltamod(a5)      * modulo in byte
    move.w   #0,bltdmod(a5)      * modulo in byte
    move.w   #-1,bltafwm(a5)
    move.w   #-1,bltalwm(a5)

    move.l   roller_base(pc),a0
    lea      -2(a0),a1
    move.l   a1,bltdpth(a5)
    lea      plane_width_roller(a0),a1
    move.l   a1,bltaph(a5)
    move.w   #$f9f0,bltcon0(a5)  * A = D/ 1 shift
    move.w   #roller_y*64+plane_width_roller/2+2,bltsize(a5)

    move.l   roller_base(pc),a0
    move.l   a0,bltaph(a5)
    lea      plane_width_roller*roller_y(a0),a0
    move.l   a0,bltdpth(a5)
    move.w   #$09f0,bltcon0(a5)  * A = D
    move.w   #64+plane_width_roller/2,bltsize(a5)  * 1 line

    move.w   #$f,color00(a5)

*** MODIFY CUR_TXT & CUR SLIDE
    move.w   cur_shift_roller(pc),d0
    subq.w   #1,d0
    tst.w    d0
    bpl.s    no_change_cur_txt_roller
    moveq     #char_x-1,d0          * & cur_text += 1
    addq.l   #4,cur_txt_roller      * if new char cur_slide = 0
    cmp.l    #text_lenght*4,cur_txt_roller  * >
    bne.s    no_change_cur_txt_roller
    move.l   #30*4,cur_txt_roller
no_change_cur_txt_roller:
    move.w   d0,cur_shift_roller    * to save d0
    lea      coded_text(pc),a0
    add.l    cur_txt_roller,a0
    move.l   (a0),a0
    add.l    #picture+64,a0          * >adr of actual char

    move.l   roller_base(pc),a1
    lea      plane_width_roller(a1),a1
    add.l    #plane_width_roller-1,a1
    lea      plane_width_roller*64(a1),a2
    moveq     #char_y-1,d0
    move.w   cur_shift_roller(pc),d1
loop_each_bit_modify_roller:
    move.l   (a0),d2
    lea      fnt_width(a0),a0
    btst     d1,d2
    bne.s    bit_true_modify_roller
bit_false_modify_roller:
    bclr     #0,(a1)
    bclr     #0,plane_width_roller(a1)
    bclr     #0,(a2)
    lea      plane_width_roller*2(a1),a1
    lea      plane_width_roller(a2),a2
    dbf      d0,loop_each_bit_modify_roller
    bra.s    end_modify_roller
bit_true_modify_roller:
    bset     #0,(a1)
    bset     #0,plane_width_roller(a1)
    bclr     #0,(a2)
    lea      plane_width_roller*2(a1),a1
    lea      plane_width_roller(a2),a2
    dbf      d0,loop_each_bit_modify_roller
end_modify_roller:

    rts
cur_shift_roller:    ds.w    1
cur_txt_roller:      ds.l    1
*****
init_roller:
    ; allocmem for bitplane
    allocmem roller_plane_size,chip+clear,roller_plane_adr

*** INIT SOME VARIABLES
    move.l   roller_plane_adr(pc),roller_base
    add.l    #plane_width_roller,roller_base
    move.w   #char_x,cur_shift_roller
    move.l   #30*4,cur_txt_roller

*** MODIFY ROLLER COPLIST
    move.l   coplist_adr(pc),a0
    lea      delta_roller_coplist(a0),a0
    clr.w    roller_angle
    clr.l    roller_var
init_roller_coplist:
    add.l    #$8888,roller_var
    addq.w   #1,roller_angle      * goto next degr
    moveq     #0,d1
    move.w   roller_angle(pc),d1  * d1= angle
    cmp.w    #180,d1             * >exit if d2=180
    beq      init_roller_end      * because it is finish
    move.l   d1,d0
    bsr      cos
    muls     #30,d0
    lsl.l    #1,d0
    swap     d0
    cmp.w    d4,d0
    beq.s    init_roller_coplist  * \
    move.w   d0,d4               * /
                                   * > next line
    addq.w   #4,a0               * to go over wait of line's start

    moveq     #0,d2
    move.w   roller_var,d2        * to modify bpl1pth
    mulu     #plane_width_roller,d2 * >compute the good adr
    add.l    roller_base(pc),d2
    addq.w   #2,a0
    swap     d2
    move.w   d2,(a0)+
    swap     d2
    addq.l   #2,a0
    move.w   d2,(a0)+

    moveq     #0,d0
    move.w   roller_angle(pc),d0  * to modify color of bpl1pth
    bsr      sin
    mulu     #13,d0
    lsl.l    #1,d0
    swap     d0
    addq.w   #3,d0
    lsl.l    #8,d0
    addq.l   #2,a0
    move.w   d0,(a0)+
    moveq     #0,d2
    move.w   roller_var,d2        * to modify bpl2pth
    neg.l    d2
    add.l    #roller_y,d2         * >compute the good target adr
    mulu     #plane_width_roller,d2
    add.l    roller_base(pc),d2
    addq.w   #2,a0
    swap     d2
    move.w   d2,(a0)+
    swap     d2
    addq.l   #2,a0
    move.w   d2,(a0)+

    moveq     #0,d0
    move.w   roller_angle(pc),d0  * to modify color of bpl2pth
    bsr      sin
    mulu     #13,d0
    lsl.l    #1,d0
    swap     d0
    neg.w    d0
    add.w    #13,d0
    addq.w   #3,d0
    addq.l   #2,a0
    move.w   d0,(a0)+

    bra      init_roller_coplist
init_roller_end:

    rts
roller_base:    ds.l    1
roller_plane_adr: ds.l    1
roller_angle:   ds.w    1
roller_var:     ds.l    1
*****
main_init:
    move.l   (exechbase).w,a6
    lea      custom,a5
    jsr      forbid(a6)

```



```

move.w #03e0,dmacon(a5) ; all dma off except disk
bsr init_plane
bsr init_coplist
bsr compute_sin_tab

bsr compute_ascii_table
bsr code_text

move.l coplist_adr,copllc(a5) ; > run my own coplist
clr.w copjmp1(a5) ;/
move.w #87c0,dmacon(a5) ; dma blitter,copper & bitplane

rts
;*****
init_plane:
rts
;*****
init_coplist:
; allocmem for coplist
; allocmem coplist_size,chip,coplist_adr

move.l coplist_adr(pc),a0

move.w #dmacon,(a0)+ * > dma blitter,copper & bitplane
move.w #87c0,(a0)+ /* bltpri is true
move.w #bpl1mod,(a0)+ * > no modulo
move.w #0,(a0)+ /*
move.w #bpl2mod,(a0)+ * > no modulo
move.w #0,(a0)+ /*
move.w #bplcon0,(a0)+ * > 2 planes and dual playfield
move.w #2600,(a0)+ /*
move.w #bplcon1,(a0)+ * > no shift
move.w #0,(a0)+ /*
move.w #bplcon2,(a0)+ * > no sprite priority
move.w #0,(a0)+ /*
move.w #ddfstrt,(a0)+ * > init ddfstrt
move.w #00031,(a0)+ /*
move.w #ddfstop,(a0)+ * > init ddfstop
move.w #000d9,(a0)+ /*
move.w #diwstrt,(a0)+ * > init diwstrt
move.w #2a74,(a0)+ /*
move.w #diwstop,(a0)+ * > init diwstop
move.w #29d4,(a0)+ /*

delta_roller_coplist = 40

move.w #61-1,d0 * d0= nblin -1
moveq #29,d1 * d1= first line
moveq #0,d2 * d2= target is useless buffer but clear

build_roller_coplist:
move.b d1,(a0)+ * wait y
move.b #03,(a0)+ * wait x
move.w #fffe,(a0)+ * wait mask
move.w #bpl1pth,(a0)+
swap d2
move.w d2,(a0)+
swap d2
move.w #bpl1pth+2,(a0)+
move.w d2,(a0)+

move.w #color00+2*1,(a0)+
move.w #f00,(a0)+

move.w #bpl2pth,(a0)+
swap d2
move.w d2,(a0)+
swap d2
move.w #bpl2pth+2,(a0)+
move.w d2,(a0)+

move.w #color00+2*9,(a0)+
move.w #f,(a0)+

addq.w #1,d1 * 1 ligne traite en +
dbf d0,build_roller_coplist

move.w #bplcon0,(a0)+
move.w #0,(a0)+

move.l #fffffffe,(a0)+ * 1 cop_instruction

rts
;*****
; routine inspire de "graphisme en gfa" de Micro Aplication p 642
; in: d0= angle
; out: d0= cos or sin
cos: add.w #90,d0 ; prinsipec: cos(d0) = sin(d0+90)
sin: cmp.w #360,d0 ; angle sup ou equ a 360
blt.s sin1 ; > if true d0=d0-360
sub.w #360,d0 ;/
bra.s sin ; make test again once
sin1: tst.w d0 ; negatif ?
bpl sin2 ; > if true d0=d0+360
add.w #360,d0 ;/
bra.s sin1

```

```

sin2: lsl.w #1,d0
move.w sin_tab(pc,d0.w),d0 ; d0 = cos ou sin
rts
;*****
; tableau tire de "graphisme en gfa" de Micro Aplication p 648
sin_tab: ; valeur des sin* 32768 de 0 a 90 deg
; valeur des sin* 2^15 de 0 a 90 deg

dc.w $0000,$023b,$0477,$06b2,$08ed ; 0-4 degr
dc.w $0b27,$0d61,$0f99,$11d0,$1406 ; 5-9 degr
dc.w $163a,$186c,$1a9c,$1ccb,$1ef7 ; 10-14 degr
dc.w $2120,$2348,$256c,$278d,$29ac ; 15-19 degr
dc.w $2bc7,$2ddf,$2ff3,$3203,$340f
dc.w $3618,$381c,$3a1c,$3c17,$3e0e
dc.w $4000,$41ec,$43d4,$45b6,$4793
dc.w $496a,$4b3c,$4d08,$4ecd,$508d
dc.w $5246,$53f9,$55a6,$574b,$58ea
dc.w $5a82,$5c13,$5d9c,$5f1f,$609a
dc.w $620d,$6379,$64dd,$6639,$678d
dc.w $68d9,$6aid,$6b59,$6c8c,$6db7
dc.w $6ed9,$6ff3,$7104,$720c,$730b
dc.w $7401,$74ef,$75d3,$761d,$777f
dc.w $7847,$7906,$79bc,$7a68,$7b0a
dc.w $7ba3,$7c32,$7cb8,$7d33,$7da5
dc.w $7e0e,$7e6c,$7ec1,$7f0b,$7f4c ; 80-84 degr
dc.w $7f83,$7fb0,$7fd3,$7fec,$7ffb ; 85-89 degr

tab90: ds.l 3*90/2 ; de 90 a 359 degr caculer par le prg
tab180 = sin_tab+180*2
tab270 = sin_tab+270*2

;*****
compute_sin_tab:
; compute end of sin_table 90-359
; sintab 90-180
lea sin_tab,a0 ; a1 pointe sur le degr 180
lea (90+90+1)*2(a0),a1
move.w #7fff,90*2(a0) ; place degr 90
moveq #90-1,d0 ; 90 deg are copy
copy_90_180: ; principe: sin 0= sin 180
move.w (a0)+,-(a1) ; sin 1= sin 179
dbf d0,copy_90_180 ; sin 2= sin 178 etc...
; sintab 180-270
lea sin_tab,a0 ; a0 pointe sur 0 degr
lea 180*2(a0),a1 ; a1 pointe sur 180 degr
moveq #90-1,d0 ; 90 deg are copy
moveq #0,d2

copy_180_270:
moveq #1,d1 ; \
swap d1 ; /d1=$10000
move.w (a0)+,d2;
sub.l d2,d1 ;
move.w d1,(a1)+ ;d1= $10000- d2
dbf d0,copy_180_270
; sintab 270-359
lea sin_tab+180*2,a0 ; a1 pointe sur le degr 360
lea (90+90)*2(a0),a1
move.w #8000,90*2(a0) ; place degr 270
lea 2(a0),a0
moveq #89-1,d0 ; 89 deg are copy
copy_270_359: ; principe: sin 360= sin 180
move.w (a0)+,-(a1) ; sin 359= sin 181
dbf d0,copy_270_359 ; sin 358= sin 182 etc...
rts

;*****
; reactivation de l'old coplist
; restore multitasking & dmacon
init_end:
bsr wait_blt
move.w #400,dmacon(a5)
lea gfxname(pc),a1
moveq #0,d0
jsr openlibrary(a6)
move.l d0,a4
move.l startlist(a4),copllc(a5); chargement de l'adr de
clr.w copjmp1(a5) ; l'old coplist
move.w #83e0,dmacon(a5) ; activation des canaux dma
jsr permit(a6) ; multi switching autorise

freemem coplist_adr(pc),coplist_size
freemem roller_plane_adr(pc),roller_plane_size

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

;*****
wait_for_blitter:
wait_blt:
btst #14,dmaconr(a5)
bne.s wait_blt
rts
;*****
code_text:
lea ascii_table(pc),a0
lea text(pc),a1 * a1= pointer on text
lea coded_text(pc),a2
move.l #text_lenght-1+20,d1 * +20 for the 20 last spaces

```



```

loop_code_text:
    moveq    #0,d0
    move.b   (a1)+,d0
    sub.w    #32,d0
    lsl.w    #2,d0
    move.l   (a0,d0.w), (a2)+
    dbf      d1,loop_code_text
    rts

***** compute the the adr of each char on with ascii code
compute_ascii_table:
    moveq    #' ',d3
compute_adr_of_each_char:
    lea      char_on_picture(pc),a0
    moveq    #nb_char_on_picture-1,d0
    move.l   d3,d2
search_char_in_ascii_table:
    cmp.b    (a0)+,d2          *\search the char in ascii table
    bneq     d0,search_char_in_ascii_table  *\if it doesn't exist it will
    sub.l    #char_on_picture+1,a0 *\be the last
    move.l   a0,d0
    divu     #nb_char_per_line,d0
    move.w   d0,d1          *\>d1= adr of char
    mulu     #font_width*char_y,d1
    swap     d0
    and.l    #$ffff,d0      *\isolate the remainder of divu in d0
    lsl.w    #2,d0
    add.w    d1,d0          *\d1= adr of line of char
    sub.w    #32,d2
    lsl.l    #2,d2          *\d2=4*d2 because the data is long word
    lea      ascii_table(pc),a0
    move.l   d0,(a0,d2.w)
    addq.w   #1,d3          *\d3= ascii code of next char
    cmp.b    #'Z',d3        *\if d3 sup than 'Z' then it's finish
    ble.s    compute_adr_of_each_char
    rts

;***** datas
; variables
coplist_adr: ds.l 1
gfxbase:     dc.l 1
gfxname:     dc.b "graphics.library",0
***** data for text
ascii_table: ds.l 91-32
nb_char_on_picture= 50
char_on_picture:dc.b "ABCDEFGH I J"
                  dc.b "KLMNOPQRST"
                  dc.b "UVWXYZ:;!'"
                  dc.b "0123456789"
                  dc.b "[]? , - . / > < "

text:  dc.b 30,'0'
       dc.b "THE FONT WAS DRAWN BY KEFRENS !"
       dc.b " THIS SCROLL USE BLITTER AND COPPER ."
text_lenght= *text
          dc.b 20,' '
          even
coded_text: ds.l text_lenght+20 * +20 for the 20 last
spaces
picture:    incbin 'fnt_32.bit'

end

```



Format de l'image de la fonte : 320x256x1

Par Jérôme Etienne



```

5 *- do i=1 to 2;
  >>> "2"
6 *- say i;
  >>> "2"
7 *- a= test(i);
  >>> "2"
12 *- test:
12 *- ;
13 *- arg k;
  >>> "2"
14 *- return(k**k);
  >>> "4"
  >>> "4"
8 *- say a;
  >>> "4"
9 *- end;
5 *- do i=1 to 2;
  >>> "3"
10 *- trace off;

```

* pour XXX égale SCAN

Ce mode est particulièrement intéressant. Il trace toutes les instructions pour tester les erreurs, mais sans les exécuter réellement. Il est donc utile de le mettre en oeuvre avant l'exécution réelle du programme (solution anti-Guru).

```

4 *- address command 'dir df1:';
5 *- do i=1 to 2;
10 *- trace off;
11 *- exit;
12 *- test:
12 *- ;
13 *- arg k;
14 *- return(k**k);

```

EXPLOITATION DES RESULTATS

L'endroit où les résultats sont affichés dépend du choix qui a été effectué au préalable.

Si rien de particulier n'a été décidé, tout se passe sur la console standard.

Si tco a été lancé, seules les opérations de traçage apparaissent dans la fenêtre ou console spécialisée. Cependant, les dimensions de l'écran étant limitées et l'exécution pouvant concerner un grand nombre de lignes, il est possible qu'on ne puisse pas examiner à loisir tout le traçage. Tout au plus peut-on arrêter momentanément le défilement avec le bouton droit de la souris, mais cela ne permet pas d'aller et venir pour traquer le défaut.

Il est possible, moyennant une ligne de programme supplémentaire, de diriger le flux qui apparaît sur la console tco vers le support le plus intéressant : imprimante, fichier en RAM: ou sur disque, etc. A partir d'un fichier, n'importe quel traitement de texte ou même l'éditeur dont on se sert pour écrire le programme permet l'édition du fichier de traçage et par là même, son analyse à loisir.

L'instruction miracle qui peut être placée dans le programme avant l'instruction trace s'écrit :

```
CALL OPEN('STDERR','ZZZ','W')
```

où zzz égale PRT: pour l'imprimante, ou RAM:test pour un fichier test en Ram, ou encore DH0:test/essai pour un fichier essai dans le répertoire test du disque dur DH0:. On notera cependant que STDERR n'a pas de possibilité d'être ouvert vers plusieurs sorties simultanées.

Pour terminer, à l'endroit où l'on veut on insérera l'instruction

```
Call Close('STDERR')
```

ce qui permet d'effectuer un traçage localisé si besoin est.

Dans l'article suivant, nous verrons la fin des possibilités de déverminage telles que les points d'arrêt, les commandes te, signal, le return code (RC), le déverminage interactif, ou encore le transfert en cas d'erreur comme partie du programme.

Par François Gueugnon





L'UTILITAIRE DU MOIS : BigCLI

Si vous avez la nostalgie des anciens ordinateurs dont le Shell était plein écran, BigCLI est fait pour vous...

BigCLI est un petit utilitaire très simple à réaliser, qui permet d'ouvrir chaque nouveau CLI ou Shell dans son propre écran - au sens Intuition du terme - et non dans une fenêtre sur le Workbench. Et quand je dis très simple à réaliser, je le pense vraiment. C'est tellement simple, que je m'étonne que personne ne l'ai fait avant (en tout cas, pas à ma connaissance).

COMMENT CA MARCHE

La démarche que j'ai suivie n'est peut-être pas la plus orthodoxe qui soit. J'en discutais encore récemment avec Frédéric Mazué, qui avait, lui, trouvé le moyen de détourner un Shell sur n'importe quelle fenêtre Intuition déjà ouverte (imaginez le résultat d'un Dir au beau milieu de votre dessin DeluxePaint !). A son humble avis de pourfendeur de bugs devant l'Eternel, ma méthode n'est pas très sûre et peut planter dans certaines conditions, qu'il reste d'ailleurs à définir... Rassurez-vous, malgré tous les essais auxquels je me suis livré, je n'ai encore jamais réussi à provoquer le moindre Guru. BigCLI est donc un programme sûr à 99% (bien que je n'aie pu le tester en WB 2.0).

La méthode en question consiste ni plus ni moins à détourner le vecteur *OpenWindow()* de la bibliothèque Intuition, de telle sorte que chaque fois que le Shell essaie d'ouvrir une nouvelle fenêtre, un écran particulier soit d'abord ouvert, qui accueillera ladite fenêtre. Cela suppose également, bien sûr, de détourner *CloseWindow()* afin de refermer cet écran en même temps que la fenêtre lors du *EndCLI* final.

Reste à savoir comment déterminer si c'est un Shell qui essaie d'ouvrir une nouvelle fenêtre, ou bien une application quelconque. Après maintes tentatives et étant donné la manière dont AmigaDOS gère l'ouverture d'une nouvelle console (à travers *CreateTask()* d'Exec), je me suis finalement décidé pour une petite astuce qui ne paye pas de mine : c'est le titre de la nouvelle fenêtre qui déterminera sa provenance. En d'autres termes, si la fenêtre à ouvrir possède un titre particulier, on assumera qu'il s'agit d'une fenêtre Shell, donc à ouvrir dans un écran propre.

Afin d'éviter tout risque de conflit, ce titre a été choisi avec soin. Il s'agit de BigCLI avec deux soulignés de part et d'autre, les majuscules étant différenciées des minuscules.

Le problème se pose à nouveau lorsque, après un *EndCLI* vindicatif, le Shell veut refermer sa fenêtre : comment savoir s'il faut également fermer l'écran supplémentaire ouvert précédemment ? Là encore, la solution est simple : on utilisera le champ *UserData* de la structure *Window* associée à chaque fenêtre. Normalement, ce champ contient un pointeur sur des données propres au propriétaire de la fenêtre, s'il en a besoin. CON: ne l'utilisant pas, nous y inscrivons simplement une marque de reconnaissance. Dans notre propre routine *CloseWindow()*, ce champ est testé et, si la marque est reconnue, l'écran est fermé en même temps que la fenêtre.

INSTALLATION

Copiez le programme *BigCLI_Inst* dans votre répertoire C: et ajoutez dans votre Startup-Sequence, à l'endroit de votre choix (généralement juste avant la commande *LoadWB*), la ligne

```
BigCLI_Inst
```

(notez qu'il n'est pas utile de spécifier *Run* ou un équivalent, le programme incorporant une version simplifiée de *BackStart.i* de Frédéric Mazué, qui permet de lancer un programme en tâche de fond et de rendre immédiatement la main au CLI appelant). Si l'installation s'est correctement effectuée, un petit message vous en prévient.

Une fois le gestionnaire principal installé, le meilleur moyen d'invoquer BigCLI est encore d'insérer la ligne suivante dans le fichier *Shell-Startup* de votre répertoire S :

```
Alias BigCLI NewShell NewCON:////_BigCLI_
```

Ainsi, en tapant simplement BigCLI, vous obtiendrez un nouveau Shell plein écran. Les aficionados du Workbench pourront également mettre dans la partie *ToolTypes* de l'icône du Shell

```
WINDOW=NewCON:////_BigCLI_
```

Pour supprimer le gestionnaire BigCLI de la mémoire, il suffit de lancer une seconde fois *BigCLI_Inst*, ou de lui envoyer, au moyen de la commande *Break*, un CTRL-F bien placé entre les deux yeux.

Une dernière note pour finir : si toutes les commandes CLI (*Dir*, *Copy*, etc.) s'affichent bel et bien dans notre écran, les programmes qui demandent explicitement une fenêtre Workbench ne sont pas détournés. Par exemple, si vous lancez *More* depuis une fenêtre BigCLI, sa fenêtre s'ouvrira de toute manière sur l'écran du Workbench. Et ça, on ne peut rien y faire.

```
; BigCLI v1.0 - (c) Max pour ANT.
;
opt      C+,OW-
incdir   "Include:"

include  "exec/tasks.i"
include  "exec/ports.i"
include  "exec/memory.i"
include  "exec/execbase.i"
include  "intuition/intuition.i"
include  "libraries/dos.i"
include  "libraries/dosextens.i"

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

; *****
CALLSYS MACRO      ; Petite macro très très pratique
IFNC      '','\2' ; piquée à Loïc Far. Merci Loïc !
movea.l   \2,a6
ENDC
jsr       _LVO\1(a6)
ENDM

EXEC      MACRO
CALLSYS   \1,$4.w
ENDM

DOS       MACRO
CALLSYS   \1,DosBase(a5)
ENDM

INT       MACRO
CALLSYS   \1,IntBase(a5)
ENDM

; *****
args      rs.reset      2      ; Arguments du CLI
MsgPort   rs.b          MP_SIZE ; MsgPort standard
DosBase   rs.l          1      ; DOSBase
IntBase   rs.l          1      ; IntuitionBase
OpenW     rs.l          1      ; Ancien vecteur OpenWindow()
CloseW    rs.l          1      ; Ancien vecteur CloseWindow()
count     rs.w          1      ; Compteur d'écrans
VARSIZE   rs.w          0

NSFLAGS   EQU           CUSTOMSCREEN

NWFLAGS   SET           BACKDROP|BORDERLESS|RMBTRAP|ACTIVATE
NWFLAGS   SET           NWFLAGS|SIMPLE_REFRESH|NOCAREREFRESH

MARQUE    EQU           'MAX!'

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

lea       dosname(pc),a1
moveq     #33,d0
EXEC      OpenLibrary
move.l    d0,DosBase(a5)
beq       NoDos
```



```

    lea    intname(pc),a1
    moveq  #33,d0
    CALLSYS OpenLibrary
    move.l d0,IntBase(a5)
    beq    NoInt

    lea    myname(pc),a1 ; On est déjà présent ?
    CALLSYS FindPort
    tst.l  d0
    beq.s  Install

    movea.l d0,a1 ; Si oui, on signale de quitter.
    movea.l MP_SIGTASK(a1),a1
    move.l  #SIGBREAKF_CTRL_F,d0
    CALLSYS Signal
    bra     Exit

; *****
Install lea    MsgPort(a5),a1
    move.b  #NT_MSGPORT,MP+LN_TYPE(a1)
    move.l  #myname,MP+LN_NAME(a1)
    move.b  #FA_IGNORE,MP_FLAGS(a1)
    move.l  ThisTask(a6),MP_SIGTASK(a1)
    CALLSYS AddPort

; Ici, on recherche les informations sur l'écran du WB
; afin de pouvoir ouvrir un écran identique pour chaque
; nouveau Shell lancé.
    lea    -sc_SIZEOF(sp),sp ; sizeof(struct Screen)
    movea.l sp,a0
    suba.l  a1,a1
    move.l  #sc_SIZEOF,d0
    moveq   #WBENCHSCREEN,d1
    INT     GetScreenData

; Copie les paramètres de l'écran du WB
; dans notre propre structure NewScreen.
    movea.l sp,a0
    lea    ns(pc),a1
    move.w  sc_Width(a0),ns_Width(a1)
    move.w  sc_Height(a0),ns_Height(a1)
    move.w  sc_ViewPort+vp_Modes(a0),d0
    andi.w  #(V_HIRES|V_LACE),d0
    move.w  d0,ns_ViewModes(a1)
    lea    sc_SIZEOF(sp),sp

; Détourne les vecteurs OpenWindow() et CloseWindow().
EXEC     Forbid

    movea.l IntBase(a5),a1
    lea    (_LVOOpenWindow).w,a0
    move.l  #NewOpenWindow,d0
    CALLSYS SetFunction
    move.l  d0,OpenW(a5) ; Sauve l'ancien vecteur

    movea.l IntBase(a5),a1
    lea    (_LVOCloseWindow).w,a0
    move.l  #NewCloseWindow,d0
    CALLSYS SetFunction
    move.l  d0,CloseW(a5) ; Sauve l'ancien vecteur

CALLSYS Permit

; Prévient l'utilisateur qu'on est installé.
    lea    ontxt(pc),a0
    bsr    ShowReq

; Ok, on a fini, il ne reste plus qu'à attendre qu'on
; nous demande de quitter.
WaitEnd move.l  #SIGBREAKF_CTRL_F,d0
EXEC     Wait

; Reste-t-il des écrans BigCLI ouverts ?
    tst.w  count(a5)
    beq.s  Remove

; Si oui, on affiche un Requester maison
    suba.l  a0,a0
    lea    autoreq(pc),a1
    lea    button1(pc),a2
    lea    button2(pc),a3
    moveq  #0,d0
    moveq  #0,d1
    move.l  #320,d2
    moveq  #90,d3
    INT     AutoRequest
    tst.l  d0
    beq.s  WaitEnd

; Si on quitte quand même alors qu'il reste un (des)
; écran(s) ouvert(s), il(s) ne sera(ont) jamais fermé(s) !

; Restaure les vecteurs OpenWindow() et CloseWindow().
Remove EXEC     Forbid
    movea.l IntBase(a5),a1
    lea    (_LVOCloseWindow).w,a0
    move.l  CloseW(a5),d0

```

```

CALLSYS SetFunction

    movea.l IntBase(a5),a1
    lea    (_LVOOpenWindow).w,a0
    move.l  OpenW(a5),d0
    CALLSYS SetFunction

CALLSYS Permit

; Supprime notre MsgPort
    lea    MsgPort(a5),a1
    CALLSYS RemPort

    lea    offtxt(pc),a0
    bsr    ShowReq

; Ferme l'intuition.library
Exit     movea.l IntBase(a5),a1
    EXEC   CloseLibrary

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

; Retour au CLI/Shell
NoDos    moveq  #0,d0
    rts

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

    lea    nwreq(pc),a0
    INT     OpenWindow
    move.l  d0,d2
    beq     NoReq

    lea    request(pc),a0
    move.l  (sp)+,rq_ReqText(a0)
    movea.l d2,a1
    CALLSYS Request

    move.l  #TICKS_PER_SECOND,d1
    DOS     Delay

    lea    request(pc),a0
    movea.l d2,a1
    INT     EndRequest

    movea.l d2,a0
    CALLSYS CloseWindow
    rts

NoReq    addq.l #4,sp
    rts

; *****
NewOpenWindow:
    movem.l d2-d3/a2-a6,-(sp)

    lea    VARS(pc),a5 ; a5 = nos variables
    movea.l a0,a3 ; a3 = NewWindow

    movea.l nw_Title(a0),a0 ; Compare le titre de la
    lea    .title(pc),a1 ; nouvelle fenêtre
    cmpm.b (a0)+,(a1)+ ; à 'BigCLI'.
    bne     .normal ; C'est une fenêtre normale...
    tst.b  -1(a0)
    bne.s   .cmp

; On est obligé d'utiliser une COPIE de la structure
; NewScreen si on veut rester ré-entrant, c'est-à-dire
; pouvoir être appelé par plusieurs programmes à la fois.
    lea    -ns_SIZEOF(sp),sp
    lea    ns(pc),a0
    movea.l sp,a1
    moveq   #ns_SIZEOF-1,d0
    move.b  (a0)+,(a1)+
    dbra   d0,.copy

    movea.l sp,a0 ; Ouvre l'écran
    CALLSYS OpenScreen
    lea    ns_SIZEOF(sp),sp
    move.l  d0,nw_Screen(a3)
    beq     .rate1

; Modifie la structure NewWindow de la fenêtre, de sorte
; qu'elle s'accommode le mieux possible à l'écran.
    movea.l d0,a0
    move.b  sc_BarHeight(a0),d0
    ext.w  d0
    addq.w  #1,d0

    move.w  sc_LeftEdge(a0),nw_LeftEdge(a3)
    move.w  sc_TopEdge(a0),nw_TopEdge(a3)
    add.w  d0,nw_TopEdge(a3)
    move.w  sc_Width(a0),nw_Width(a3)
    move.w  sc_Height(a0),nw_Height(a3)
    sub.w  d0,nw_Height(a3)

```


ROUTINETTES : DEUXIEME !

*Youpie, youpla, voilà la deuxième
Routinette de Tonton François. La
joie m'envahit et me rend tout
fébrile. Quelle excitation !*

En arrivant dans le monde de l'Amiga, j'ai pris AmigaDOS en grippe, comme beaucoup de monde. Il faut dire que la vitesse d'affichage d'un directory sur disquette est certainement la plus mauvaise publicité d'AmigaDOS. Sous prétexte d'une sécurité plus grande des disques (!!! J'ai bouzillé beaucoup plus de disquettes Amiga que PC ou Atari, car elles ont besoin d'être validées !) on est obligé de passer de longues minutes à écouter le lecteur grincer, en voyant d'afficher trois misérables noms de fichiers à la vitesse d'un escargot asmatique.

Si la personne responsable de cet état de fait (le programmeur initial d'AmigaDOS) voyait déferler sur lui le potentiel de haine accumulée par les millions d'Amigaïstes mécontents, il se désintégrerait immédiatement !

Quelle mauvaise publicité, donc, pour AmigaDOS. Mais la lenteur des répertoires n'est qu'une misérable façade, cachant un système finalement très bien conçu, dont on découvre les finesses avec l'usage, et que l'on se prend à aimer et finalement à préférer au MS-DOS !

Qui aime bien châtie bien, il reste tout de même quelques points bizarres dans le DOS. Certaines fonctions, qui semblent pourtant indispensables, sont tout bonnement omises, comme par exemple une fonction pour obtenir le directory courant. Il est tout de même primordial de savoir où l'on est pour pouvoir accéder au disque !

Quelle coïncidence, je vous propose aujourd'hui une routinette ramenant le path complet, depuis le nom du disque jusqu'à l'endroit d'où le programme a été lancé.

LE BEAU PROGRAMME

Pour appeler la routinette d'aujourd'hui, vous devez :

- ouvrir la dos.library et mettre son adresse de base dans *DosBase*.
- charger A0 avec un pathname pointant sur un fichier quelconque du directory dont vous désirez connaître le cheminement. Une chaîne nulle ramène le pathname courant.
- charger A1 avec l'adresse d'un buffer de 128 octets minimum dans lequel sera inscrit le cheminement.
- appeler la routinette.

Vous remarquerez que j'utilise A5 comme pointeur sur la zone de données. Pourquoi ? Pour faire du code relogeable. On peut parfaitement utiliser les instructions "relatif PC". Le problème est qu'avec un 68000, ces instructions ne sont valables que pour l'opérande source. En chargeant A5 avec l'adresse de la zone de données, on peut accéder en lecture et en écriture à cette zone.

Grands avantages de ce système : plus rapide, plus court (on gagne deux octets par instruction !), réentrant (il suffit de changer A5 et on change de zone de données). Evidemment, on perd A5, mais bon, il reste tout de même cinq autres registres...

LA BELLE CONCLUSION

Qui sera aussi très courte : à plus.

```

*-----
* ROUTINETTE numéro 2
* _GetDir: trouve un pathname complet.
*
* Par François Lionet, paru dans l'A.N.T. du 1er Janvier
* 1992
*-----
*----- Definition de variables système
* Appel du dos
DosCall MACRO
    move.l a6, -(sp)

```

```

    move.l DosBase-Dt(a5), a6
    jsr    \1(a6)
    move.l (sp)+, a6
    ENDM

* Offsets des fonctions dos
_LVOExamine    equ    -102
_LVOParentDir  equ    -210
_LVOLock       equ    -84
_LVOUnLock     equ    -90
_LVOWrite      equ    -48
_LVOOutput     equ    -60

* Offset des fonctions EXEC
_LVOOpenLibrary    equ    -552
_LVOCloseLibrary   equ    -414

*----- Demonstration
    lea    Dt(pc), a5 ; Base des donnees

* Ouvre le dos
    move.l $4.w, a6
    lea    DosName-Dt(a5), a1
    moveq  #0, d0
    jsr    _LVOOpenLibrary(a6)
    move.l d0, DosBase-Dt(a5) ; Une erreur est très
    beq.s  Err ; improbable!

* Va demander le chemin
    lea    PathNul-Dt(a5), a0
    lea    Buffer-Dt(a5), a1
    bsr    _GetDir

* Imprime sur le canal standart
    move.b #10, 0(a0, d0.l) ; Retour à la ligne
    addq.l #1, d0
    move.l a0, d2 ; Adresse de la chaîne
    move.l d0, d3 ; Longueur
    DosCall _LVOOutput ; Demande le canal de sortie
    move.l d0, d1 ; sans modifier d2 et d3 !
    DosCall _LVOWrite ; Astuss !

* Ferme le dos
    move.l DosBase-Dt(a5), a1
    jsr    _LVOCloseLibrary(a6)

* No problemo
    Err    moveq  #0, d0
    rts

*----- Routinette
* En entrée A0= Nom d'un fichier, ou pointeur
* sur une chaîne nulle pour avoir le
* directory courant...
* En sortie A1= Espace de 128 octets ou paker le nom
* A0= debut du buffer
* D0= longueur de la chaîne
GetDir:
* Preparation des parametres
    movem.l a1-a3/d1-d3, -(sp)
    move.l a1, a2 ; A2 est un registre préserve
    lea    -250(sp), sp ; Reserve un espace de travail
    move.l sp, a3 ; en A3
* Demande un lock sur le point d'entree
    move.l a0, d1
    moveq  #-2, d2
    DosCall _LVOLock
* On travaille avec la pile
    clr.l -(sp) ; Marque le debut
    move.l d0, -(sp) ; Pousse le lock
    move.l d0, d1 ; Demande le précédent
    DosCall _LVOParentDir
    tst.l d0
    bne.s .Gd1 ; On a atteint le premier?
    ; Non: on reboucle!
    clr.b (a2) ; Un zero à la fin
    moveq  #":", d3 ; Le premier est un disque
    .Gd2: move.l (sp), d1 ; On reprend les locks
    beq.s .Gd5 ; Si terminé...
    move.l a3, d2 ; Poke dans la pile
    DosCall _LVOExamine ; Demande le nom!
    tst.l d0 ; En cas de probleme...
    beq.s .Gd4
    lea    8(a3), a1 ; On recopie le nom!
    .Gd3: move.b (a1)+, (a2)+
    bne.s .Gd3
    move.b d3, -1(a2) ; Met : ou /
    clr.b (a2) ; Marque la fin
    moveq  #"/", d3 ; Le seul ":" est le premier!
    .Gd4: move.l (sp)+, d1 ; Il faut maintenant
    DosCall _LVOUnLock ; effacer les locks!
    bra.s .Gd2 ; Et on boucle!

* Point de sortie!
    .Gd5: lea    250+4(sp), sp ; Efface le buffer et le Clr
    move.l a2, d0 ; Fin de la chaîne
    movem.l (sp)+, a1-a3/d1-d3 ; Restaure les registres
    move.l a1, a0 ; Debut de la chaîne
    sub.l a0, d0 ; Taille de la chaîne
    rts

```

*----- Pitite mignone zone de donnee

```

* La base des acces à la zone de donnee
Dt    DosBase    dc.l    0
      Buffer     ds.b    128
      PathNul   dc.b    0
      DosName    dc.b    "dos.library", 0
              even

```

Par Tonton François Lionet

ROUTINETTES : DEUXIEME !

*Youpie, youpla, voila la deuxième
Routinette de Tonton François. La
joie m'envahit et me rend tout
fébrile. Quelle excitation !*

En arrivant dans le monde de l'Amiga, j'ai pris AmigaDOS en grippe, comme beaucoup de monde. Il faut dire que la vitesse d'affichage d'un directory sur disquette est certainement la plus mauvaise publicité d'AmigaDOS. Sous prétexte d'une sécurité plus grande des disques (?!? J'ai bouzillé beaucoup plus de disquettes Amiga que PC ou Atari, car elles ont besoin d'être validées !) on est obligé de passer de longues minutes à écouter le lecteur grincer, en voyant d'afficher trois misérables noms de fichiers à la vitesse d'un escargot asmatique.

Si la personne responsable de cet état de fait (le programmeur initial d'AmigaDOS) voyait déferler sur lui le potentiel de haine accumulée par les millions d'Amigaïstes mécontents, il se désintègrerait immédiatement !

Quelle mauvaise publicité, donc, pour AmigaDOS. Mais la lenteur des répertoires n'est qu'une misérable façade, cachant un système finalement très bien conçu, dont on découvre les finesses avec l'usage, et que l'on se prend à aimer et finalement à préférer au MS-DOS !

Qui aime bien châtie bien, il reste tout de même quelques points bizarres dans le DOS. Certaines fonctions, qui semblent pourtant indispensables, sont tout bonnement omises, comme par exemple une fonction pour obtenir le directory courant. Il est tout de même primordial de savoir où l'on est pour pouvoir accéder au disque !

Quelle coïncidence, je vous propose aujourd'hui une routinette ramenant le path complet, depuis le nom du disque jusqu'à l'endroit d'où le programme a été lancé.

LE BEAU PROGRAMME

Pour appeler la routinette d'aujourd'hui, vous devez :

- ouvrir la dos.library et mettre son adresse de base dans *DosBase*.
- charger A0 avec un pathname pointant sur un fichier quelconque du directory dont vous désirez connaître le cheminement. Une chaîne nulle ramène le pathname courant.
- charger A1 avec l'adresse d'un buffer de 128 octets minimum dans lequel sera inscrit le cheminement.
- appeler la routinette.

Vous remarquerez que j'utilise A5 comme pointeur sur la zone de données. Pourquoi ? Pour faire du code relogeable. On peut parfaitement utiliser les instructions "relatif PC". Le problème est qu'avec un 68000, ces instructions ne sont valables que pour l'opérande source. En chargeant A5 avec l'adresse de la zone de données, on peut accéder en lecture et en écriture à cette zone.

Grands avantages de ce système : plus rapide, plus court (on gagne deux octets par instruction !), réentrant (il suffit de changer A5 et on change de zone de données). Evidemment, on perd A5, mais bon, il reste tout de même cinq autres registres...

LA BELLE CONCLUSION

Qui sera aussi très courte : à plus.

```
*-----
* ROUTINETTE numéro 2
* _GetDir: trouve un pathname complet.
*
* Par François Lionet, paru dans l'A.N.T. du 1er Janvier
* 1992
*-----
```

```
*----- Definition de variables système
* Appel du dos
DosCall MACRO
    move.l a6, -(sp)
```

```
move.l DosBase-Dt(a5), a6
jsr _l(a6)
move.l (sp)+, a6
ENDM

* Offsets des fonctions dos
_LVOExamine equ -102
_LVOParentDir equ -210
_LVOLock equ -84
_LVOUnLock equ -90
_LVOWrite equ -48
_LVOOutput equ -60

* Offset des fonctions EXEC
_LVOOpenLibrary equ -552
_LVOCloseLibrary equ -414

*----- Demonstration
lea Dt(pc), a5 ; Base des donnees

* Ouvre le dos
move.l $4.w, a6
lea DosName-Dt(a5), a1
moveq #0, d0
jsr _LVOOpenLibrary(a6)
move.l d0, DosBase-Dt(a5) ; Une erreur est très
beq.s Err ; improbable!

* Va demander le chemin
lea PathNul-Dt(a5), a0
lea Buffer-Dt(a5), a1
bsr _GetDir

* Imprime sur le canal standart
move.b #10, 0(a0, d0.l) ; Retour à la ligne
addq.l #1, d0
move.l a0, d2 ; Adresse de la chaîne
move.l d0, d3 ; Longueur
DosCall _LVOOutput ; Demande le canal de sortie
move.l d0, d1 ; sans modifier d2 et d3 !
DosCall _LVOWrite ; Astuss !

* Ferme le dos
move.l DosBase-Dt(a5), a1
jsr _LVOCloseLibrary(a6)

* No problemo
Err moveq #0, d0
rts

*----- Routinette
* En entrée A0= Nom d'un fichier, ou pointeur
sur une chaîne nulle pour avoir le
directory courant...
* A1= Espace de 128 octets ou poker le nom
* En sortie A0= debut du buffer
D0= longueur de la chaîne

GetDir:
* Preparation des parametres
movem.l a1-a3/d1-d3, -(sp)
move.l a1, a2 ; A2 est un registre préserve
lea -260(sp), sp ; Reserve un espace de travail
move.l sp, a3 ; en A3

* Demande un lock sur le point d'entree
move.l a0, d1
moveq #-2, d2
DosCall _LVOLock

* On travaille avec la pile
clr.l -(sp) ; Marque le debut
.Gd1: move.l d0, -(sp) ; Pousse le lock
move.l d0, d1 ; Demande le précédent
DosCall _LVOParentDir
tst.l d0 ; On a atteint le premier?
bne.s .Gd1 ; Non: on reboucle!

* Redescend les LOCKS en demandant leur NOM
clr.b (a2) ; Un zero à la fin
moveq #":", d3 ; Le premier est un disque
.Gd2: move.l (sp), d1 ; On reprend les locks
beq.s .Gd5 ; Si terminé...
move.l a3, d2 ; Poke dans la pile
DosCall _LVOExamine ; Demande le nom!
tst.l d0 ; En cas de probleme...
beq.s .Gd4
lea 8(a3), a1 ; On recopie le nom!
.Gd3: move.b (a1)+, (a2)+
bne.s .Gd3
move.b d3, -1(a2) ; Met : ou /
clr.b (a2) ; Marque la fin
moveq #"/", d3 ; Le seul ":" est le premier!
.Gd4: move.l (sp)+, d1 ; Il faut maintenant
DosCall _LVOUnLock ; effacer les locks!
bra.s .Gd2 ; Et on boucle!

* Point de sortie!
.Gd5: lea 260+4(sp), sp ; Efface le buffer et le Clr
move.l a2, d0 ; Fin de la chaîne
movem.l (sp)+, a1-a3/d1-d3 ; Restaure les registres
move.l a1, a0 ; Debut de la chaîne
sub.l a0, d0 ; Taille de la chaîne
rts
```

*----- Pitite mignone zone de donnee

```
* La base des acces à la zone de donnee
Dt
DosBase dc.l 0
Buffer ds.b 128
PathNul dc.b 0
DosName dc.b "dos.library", 0
even
```

Par Tonton François Lionet

SUBWAY : VOUS PERMETTEZ QUE JE

Contrairement aux VSprites, qui utilisent les sprites hardware de l'Amiga, les Bobs sont gérés exclusivement par le Blitter. D'où le nom de Bob, d'ailleurs, contraction de "Blitter Object".

Cette simple différence de gestion entraîne, comme vous l'allez pouvoir constater, quelques différences dans la programmation de leur affichage... Comme le sprite virtuel, le Bob fait partie intégrante du GEL System. Mais contrairement aux VSprites, qui, de par leurs liens avec les sprites hardware, sont indépendants du playfield, le Bob en fait partie intégrante. On peut donc utiliser un Bob comme une brosse (comme le fait Deluxe Paint, par exemple) ou au contraire, simuler un sprite en sauvegardant la zone du playfield correspondant au rectangle englobant le Bob lors de son affichage, et en la ré-affichant lorsque le Bob s'est déplacé.

Le Bob n'étant plus lié à un sprite hardware mais au playfield, le nombre de couleurs que l'on peut associer à un Bob est le même que celui du playfield, ainsi que le contenu des registres de couleurs. Il vous faudra donc faire attention à ne pas utiliser les mêmes registres pour ces deux éléments sous peine de voir le Bob disparaître dans le playfield. Dans un playfield à 32 couleurs, on pourra par exemple utiliser les 16 premières pour le fond et les 16 autres pour le ou les Bobs.

La structure de données associée à un Bob contient des champs définissant sa structure propre ainsi qu'un pointeur sur une structure VSprite qui contient la définition des attributs communs aux sprites virtuels et au Bob. Ces données communes sont liées en grande partie à la position du Bob. Ainsi, il sera possible de gérer cette position indépendamment du type d'élément manipulé. Il s'agit là d'une excellente initiative, évitant une duplication inutile du code de gestion des positions.

Au niveau de la structure de données des Bobs, nous allons étudier simplement les champs propres à la structure Bob et les champs de la structure VSprite possédant une valeur spécifique. Pour la description du reste des données, je vous renvoie à l'article du mois dernier sur les VSprites.

L'IMAGE DU BOB

Vous vous rappelez sans doute que l'image d'un VSprite (limitée à 16 pixels en largeur) est définie par une série de couples de deux mots de 16 bits, permettant pour chacun de définir une ligne du VSprite :

```
0x0fc3, 0x0000, /* ligne 1 du VSprite */
```

Pour un Bob, l'organisation des données est différente. Comme nous l'avons vu précédemment, un Bob possède autant de couleurs que le playfield auquel il est associé. De plus, un Bob n'est pas limité arbitrairement dans ses dimensions, mais seulement par la taille du playfield auquel il est associé. Ainsi, rien n'empêche de réaliser un Bob de 64 pixels de large sur 100 lignes de haut.

Dans cet exemple, les dimensions ne sont pas choisies innocemment. En effet, la taille en largeur d'un Bob doit être impérativement un multiple de 16 (nombre de mots de 16 bits). Par contre, sa hauteur s'exprime en nombre de lignes sans contraintes.

Pour définir un Bob de 32 pixels de large sur deux lignes de haut, évoluant sur un écran à deux bitplanes, nous déclarerons l'image associée à ce Bob de la manière suivante :

```
/* Biplane 0 */
0x0000, 0xfce0, /* Première ligne */
0xc000, 0x1111, /* Deuxième ligne */

/* Bitplane 1 */
```

```
0x1777, 0xfce0, /* Première ligne */
0x4444, 0x4444, /* Deuxième ligne */
```

LES COULEURS

Le choix du registre de couleur associé à un pixel s'effectue de la même manière que pour les playfields. Le numéro du registre de couleur est induit par chaque bit de chaque biplane du Bob, le playfield de plus haut numéro étant le bit de poids le plus fort.

Par exemple les combinaisons possibles pour un Bob possédant deux bitplanes sont :

```
Bitplane 0 : 0 0 1 1
Bitplane 1 : 0 1 0 1
Couleur : 0 1 2 3
```

A première vue, lorsque l'on utilise un Bob de deux bitplanes sur un playfield de trois bitplanes ou plus, seuls les quatre premiers registres sont disponibles (0 à 3).

Pour palier à ce problème, vous avez la possibilité, grâce aux champs PlanePick et PlaneOff, de définir quels sont les registres qui vont être réellement utilisés. Ils servent donc à établir une table de correspondance entre le numéro de registre théorique (ex : 0, 1, 2, 3) et le numéro réel du registre utilisé (ex : 10, 11, 14, 15).

PlanePick permet d'associer à chaque bitplane du Bob, un bitplane du playfield. Ainsi, si on reprend l'exemple d'un Bob de deux bitplanes sur un playfield de cinq :

```
N° bitplane | 5 4 3 2 1
PlanePick | 0 1 0 1 0
```

entraînera l'écriture des données du bitplane 0 du Bob dans le bitplane 1 du playfield et le bitplane 1 du Bob dans le bitplane 4 du playfield.

Dans ce cas, les registres de couleurs utilisés ne seront plus (0, 1, 2 et 3) mais (0, 2, 8 et 10). PlanePick ne permet toutefois pas toutes les combinaisons possibles, d'où le champ PlaneOnOff.

Ce champ contient une valeur sur 6 bits, comme PlanePick, qui permet par un OU logique entre chaque registre de couleur et sa valeur de modifier les registres choisis pour afficher le Bob.

Ainsi, si l'on positionne PlaneOnOff à 5, on obtiendra la combinaison suivante :

```
Registre 1 | 0 0 0 1 0
PlaneOnOff | 0 0 1 0 1
Reg. Réel | 0 0 1 1 1 -> registre COLOR 7
```

Idem pour les autres registres, ce qui nous donne (0, 7, 13 et 15).

Pour obtenir dans notre exemple précédent (10, 11, 14 et 15), nous aurions du placer les valeurs suivantes dans PlanePick et PlaneOnOff :

```
PlaneOnOff | 0 1 0 1 0
PlanePick | 0 0 1 0 1
```

AVANTAGES ET INCONVENIENTS

Un des principaux avantages d'utiliser des Bobs par rapport à des VSprites, c'est qu'ils peuvent être de la taille de votre choix, aussi bien en largeur qu'en hauteur. Ceci tant que vous avez encore assez de mémoire CHIP pour stocker leurs images. Par contraste, nous avions vu que le VSprite était limité à seize pixels en largeur.

Du point de vue des couleurs, nous sommes également loin de la limitation du VSprite (trois couleurs effectives, plus une transparente). En effet, nous disposons pour le Bob du même nombre de couleurs que pour le playfield auquel il est attaché. Cela peut donc aller jusqu'à 4096 en mode HAM.

Pour les Bobs, vous pouvez définir les priorités d'affichage, à l'aide des pointeurs Before et After qui leur sont associés. Vous avez ainsi la maîtrise totale de l'affichage des objets en cas de recouvrement.

VOUS APPELLE BOB ?

De part le fait que les Bobs font partie intégrante du playfield, leur utilisation est plus lente que celle des VSprites. Il est d'ailleurs vivement conseillé d'utiliser un écran double-buffer. Il ainsi possible de tracer dans un écran pendant que l'on affiche l'autre, rendant ainsi l'animation plus fluide.

Les Bobs utilisent, à taille et profondeur égale, plus de mémoire que les VSprites. En effet, il est presque toujours nécessaire de positionner le flag *SAVEBACK* afin d'avoir une sauvegarde automatique du fond. Dans ce cas, il faut donc la mémoire nécessaire à sauvegarder cette portion en plus de la mémoire liée à l'image du Bob.

PROGRAMMATION

Nous en savons désormais assez pour écrire un programme d'affichage et d'animation de Bobs. Ce programme va afficher des logos "ANT" en 16 couleurs sur un écran possédant lui aussi 16 couleurs. Les précisions nécessaires sont incluse dans le listing que j'ai le plus possible commenté.

Le logo a été réalisé grâce à un utilitaire du domaine public. Il existe de nombreux programmes permettant de vous aider dans la tâche fastidieuse consistant à inclure le source C d'une image dans un programme. Vous trouverez une description de quelques uns de ces logiciels dans la rubrique ToolBox du mois prochain.

Sur ce, je vous souhaite une bonne étude sur l'animation et je vous retrouve dans un mois pour une nouvelle rubrique qui va faire du bruit. Mais je ne vous en dit pas plus.... Surprise.

```

/*-----*/
/* Programme d'affichage et d'animation de BOBs */
/* Exemple de programmation des BOBs */
/* P. AMIABLE 1991 */
/*-----*/

/*-----*/
/* Quelques includes utiles..... */
/*-----*/
#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <stdio.h>
#include <graphics/gels.h>

/*-----*/
/* Les defines maintenant */
/*-----*/
#define INTUITION_REV 0L /* Version d'Intuition */
#define GFX_REV 0L /* Version de Graphics */
#define NBRBOBS 1 /* Nombre de BOBs affichés */
#define LARGEUR 64 /* Largeur en pixel du bob */
#define HAUTEUR 58 /* Hauteur en ligne du bob */
#define PROFONDEUR 4 /* Profondeur en bitplane du bob */

/*-----*/
/* Définition des variables externes */
/*-----*/
/* Declaration des fonctions définies dans le programme */
void reforme();
void init();
void detruit_bobs();
void ouvrefenetre();
void reformegel();
int intgel();
void graphique();
void deplace();
int bidon();
struct Bob *cree_bob();
struct VSprite *cree_vsprites();

struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;

struct NewScreen nouvecran = {
    0, 0, /* LeftEdge, TopEdge */
    320, 256, /* Width, Height */
    4, /* Depth, soit 16 couleurs possibles */
    1, 0, /* DetailPen, BlockPen */
    SPRITES, /* ViewModes valeur de 0 = basse resolution */
    CUSTOMSCREEN, /* Type d'ecran */
    NULL, /* Font -> NULL = fonts par défaut */
    NULL, /* DefaultTitle pour le titre -> Aucun */
    NULL, /* gadgets, aucun */
    NULL }; /* adresse de la CustomBitmap -> aucune ici */

struct NewWindow nouvfenetre =
{
    0, /* LeftEdge position en x de la fenêtre */
    0, /* TopEdge position en y de la fenêtre */
    320, /* width 320 pixels de large */
    256, /* Height 256 pixels de hauts */
    0, /* DetailPen Le texte doit être écrit dans la couleur 0 */
    1, /* BlockPen les blocks doivent être dans la couleur 1 */
    CLOSEWINDOW, /* IDCMPFlags Rien de particulier */
    WINDOWCLOSE|WINDOWSIZING|WINDOWDEPTH|ACTIVATE,
    NULL, /* FirstGadget Pas de gadgets personnels */

```

```

NULL, /* CheckMark Pas de Checkmark particulière */
"ANT a pris des couleurs" /* Titre Titre de la fenêtre */
NULL, /* Screen Attaché l'écran en cours de programme */
NULL, /* BitMap pas de Bitmap personnelle */
320, /* MinWidth La fenêtre ne peut être plus petite */
10,
320,
256,
CUSTOMSCREEN /* Type On accroche la fenêtre au Workbench */
};

struct Window *fenetre = NULL;
struct Screen *ecran = NULL;
struct RastPort *rastport = NULL;
struct ViewPort *viewport = NULL;

SHORT vitesse[] = { 1, 2, -1, -2 }; /* Echelle de vitesse */

SHORT xmov[NBRBOBS], ymov[NBRBOBS]; /* Direction des bobs */

struct Bob *bob[NBRBOBS]; /* On va créer NBRBOBS Bobs donc on se réserve les pointeurs nécessaires */

struct VSprite *vsprite;
short max_cree = 0; /* nombre maxi de Bobs créés */

struct VSprite *tete = NULL;
struct VSprite *queue = NULL;

struct GelsInfo gelinfo;

UWORD chip bobimage[] = { /* données du sprite */

/*----- bitmap : largeur = 64, hauteur = 58 -----*/

/*----- plan 0: -----*/
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x88, 0x80, 0x0, 0x0, 0x120, 0x80, 0x0,
0x20, 0x30, 0x80, 0x0, 0x6a, 0x2c, 0x180, 0x40,
0xcce, 0x3e, 0x187, 0x99aa, 0x1fb, 0x2a, 0x29f, 0xffe0,
0x3ff, 0x3a, 0x39e, 0xebe0, 0x64d, 0x8018, 0x18e, 0x3d80,
0xa34, 0x10, 0x110, 0x3380, 0x20, 0x0, 0x0, 0x0,
0x2000, 0x8, 0x8101, 0x20, 0x2000, 0x8035, 0x513, 0x2020,
0x2c80, 0xa01c, 0x6191, 0x33e0, 0x3700, 0x603c, 0xe79d, 0xffe0,
0x3f0, 0x3fac, 0xf705, 0x6fa0, 0x3900, 0x3f34, 0xd781, 0xc60,
0x2c9a, 0x5800, 0x600, 0x4240, 0x15e4, 0x2000, 0x1100, 0x80,
0x300, 0x8020, 0x4000, 0x0, 0x20, 0x0, 0x1291, 0x6780,
0x804, 0x8000, 0x1399, 0xe7e0, 0x3e91, 0xa038, 0x1590, 0xe1e0,
0x33ff, 0xc03c, 0x1f01, 0xe2c0, 0x3f7f, 0xe01c, 0xf91, 0xe240,
0x3b01, 0xc01c, 0x580, 0x100, 0x3900, 0x4004, 0x700, 0x2020,
0x0, 0x2000, 0x1, 0x21a0, 0x200, 0x0, 0x1, 0x180,
0x0, 0x2008, 0x1, 0xe0e0, 0x801, 0xa01c, 0x81, 0xc0e0,
0x1f00, 0xa03c, 0x280, 0x6020, 0x3e01, 0x603c, 0x381, 0x80a0,
0x3f01, 0xe038, 0x680, 0x0, 0x1701, 0xe03c, 0x601, 0x8020,
0x1a00, 0xc024, 0x281, 0xe020, 0x2900, 0x4000, 0x301, 0xa020,
0x400, 0x2000, 0x1, 0xe020, 0x0, 0x0, 0x301, 0xa020,
0x1f00, 0xe038, 0x100, 0xe000, 0x2701, 0xa03c, 0x580, 0x0,
0x3f80, 0xe03c, 0x0, 0x0, 0x1, 0xe03c, 0x781, 0xc000,
0x1, 0xa03c, 0x781, 0xe000, 0x1, 0xa024, 0x581, 0xe000,
0x0, 0xe03c, 0x0, 0x0, 0x0, 0x0, 0x4, 0xf800,
0x1, 0xa03c, 0x781, 0xe000, 0x0, 0x0, 0x0, 0x0,
0x0, 0x800c, 0x682, 0x4200, 0x0, 0x0, 0x6020, 0x180, 0x0,
0x0, 0xe010, 0x280, 0x0, 0x0, 0x0, 0x2030, 0x700, 0x0,
0x0, 0x6030, 0x7ff, 0xff00, 0x0, 0x0, 0x2030, 0x6ff, 0xf800,
0x0, 0x0, 0x359, 0xc000, 0x0, 0x0, 0x2000, 0x4, 0xf80,
0x0, 0x0, 0x420, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,

/*----- plan 1: -----*/
0x0, 0x30, 0x0, 0x0, 0x0, 0x70, 0x0, 0x0, 0x0,
0x8, 0xf8, 0x80, 0x0, 0x1c, 0x1f, 0x90, 0x0,
0xc, 0xc, 0x98, 0x20, 0x7e, 0x3c, 0x198, 0x60,
0xfe, 0x3e, 0x19f, 0xffe0, 0x1ff, 0x2a, 0x29f, 0xffe0,
0x3ff, 0x3a, 0x39e, 0xebe0, 0x64d, 0x8018, 0x18e, 0x3d80,
0xa34, 0x10, 0x110, 0x3380, 0x20, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x10, 0x98, 0x8040, 0x600, 0x8, 0x201c, 0xe380,
0x1365, 0xa63c, 0x7999, 0xad40, 0x2a1b, 0xd03c, 0x6e99, 0xefe0,
0x3cff, 0x783c, 0xf799, 0xe7e0, 0x3fff, 0xf03c, 0x3f99, 0xe7e0,
0x3fff, 0xe03c, 0xf999, 0xe7e0, 0x3fff, 0xe03c, 0x1f90, 0xe1e0,
0x3fff, 0xe03c, 0xf81, 0xe2c0, 0x3fff, 0xe01c, 0xf91, 0xe240,
0x3b01, 0xc01c, 0x580, 0x100, 0x3900, 0x4004, 0x700, 0x0,
0x3f00, 0x2000, 0x0, 0x0, 0x200, 0x0, 0x0, 0x0,
0x0, 0x0, 0x1, 0xe0c0, 0x0, 0x0, 0x0, 0x2000,
0x0, 0x4, 0x101, 0xe060, 0x2800, 0x0, 0x181, 0xe060,
0x2501, 0x2018, 0x501, 0xe060, 0x1601, 0xa03c, 0x481, 0xe020,
0x3b01, 0xc03c, 0x781, 0xe020, 0x3f01, 0xe03c, 0x781, 0xa020,
0x3f01, 0xe03c, 0x780, 0xe000, 0x3f01, 0xe03c, 0x780, 0x0,
0x3f81, 0xe03c, 0x780, 0x0, 0x1, 0xe03c, 0x780, 0x0,
0x1, 0xa03c, 0x780, 0x0, 0x1, 0xa024, 0x580, 0x0,
0x0, 0x8010, 0x500, 0x0, 0x0, 0x0, 0x400, 0x0,
0x0, 0x0, 0x1, 0xb00, 0x0, 0x0, 0xf, 0xff80,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x2020, 0xae, 0x0, 0x4000, 0x100, 0x0,
0x0, 0x0, 0x3c0, 0x0, 0x0, 0x20, 0x7fa, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x700, 0x0,

/*----- plan 2: -----*/
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x5, 0x1, 0x1400, 0x1b2, 0x27, 0x211, 0xc260,
0x5cb, 0x802f, 0x868f, 0xcc60, 0x1fcf, 0x03ff, 0x879f, 0xffe0,
0x3cf, 0x03ff, 0xc79f, 0xf000, 0x3fc3, 0xe03d, 0xc79f, 0xffe0,
0x3f81, 0xe03d, 0xe79d, 0xffe0, 0x3f80, 0xf03c, 0xe79d, 0xffe0,
0x3f0, 0x7fbc, 0xf79d, 0xe7e0, 0x3f00, 0x3f3c, 0xf79d, 0xe7e0,
0x3f0, 0xf63c, 0xf799, 0xe7e0, 0x3f0, 0x3f3c, 0xf799, 0xe7e0,
0x3fff, 0xf83c, 0xf799, 0xe7e0, 0x3fff, 0xf03c, 0xf399, 0xe7e0,
0x3fff, 0xe03c, 0xf399, 0xe7e0, 0x3fff, 0xe01c, 0xf159, 0xe1e0,
0x3fff, 0xe03c, 0xf181, 0xe2c0, 0x3fff, 0xe01c, 0xf159, 0xe1e0,
0x3b01, 0xc01c, 0x580, 0x100, 0x3900, 0x4004, 0x700, 0x0,
0x3f00, 0x2000, 0x0, 0x0, 0x200, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,

```




```
gelinfo.leftmost = 0;
gelinfo.rightmost = rastport->BitMap->BytesPerRow * 8 - 1;
gelinfo.topmost = 0;
gelinfo.bottommost = rastport->BitMap->Rows - 1;

rastport->GelsInfo = &gelinfo;
InitGels(tete, queue, &gelinfo);

SetCollision(0, bidon, &gelinfo);
WaitTOF(); /* Attente de fin de trame */
}

/*-----*/
/* Cette fonction désalloue la structure gels . */
/*-----*/
void refermegel()
{
    if (gelinfo.collHandler != NULL)
        FreeMem(gelinfo.collHandler, sizeof(struct collTable));
    if (gelinfo.lastColor != NULL)
        FreeMem(gelinfo.lastColor, sizeof(LONG) * 8);
    if (gelinfo.nextLine != NULL)
        FreeMem(gelinfo.nextLine, sizeof(WORD) * 8);
    if (gelinfo.gelHead != NULL)
        FreeMem(gelinfo.gelHead, sizeof(struct VSprite));
    if (gelinfo.gelTail != NULL)
        FreeMem(gelinfo.gelTail, sizeof(struct VSprite));
}

/*-----*/
/* Cette fonction crée l'ensemble des éléments graphiques */
/*-----*/
void graphique()
{
    int erreur;
    int i,j,x,y;

    erreur = initgel(&gelinfo, rastport);
    for(i=0; i<NBBOBS; i++)
    {
        xmouv[i] = vitesse[RangeRand(4)];
        ymouv[i] = vitesse[RangeRand(4)];

        x = 10 + RangeRand(246);
        y = 10 + RangeRand(182);

        bob[i] =
        cree_bob(LARGEUR,HAUTEUR,PROFONDEUR,bobimage,0x3f,0x3f,x,
        y,SAVEBACK|OVERLAY);

        if(bob[i] == 0)
            break; /* plus de mémoire */

        if (i > 0)
        {
            j = i-1;
            Bob[i]->After = bob[j]->Before;
            Bob[i]->After->Before = bob[j];
        }
        AddBob(bob[i], rastport);
        max_cree++;
    }
}

/*-----*/
/* Cette fonction détruit un bob et libère ses ressources */
/*-----*/
void detruit_bobs(vsprite)
struct VSprite *vsprite;
{
    if (vsprite != NULL) {
        if (vsprite->VSBob != NULL) {
            if (vsprite->VSBob->SaveBuffer != NULL) {
                FreeMem(vsprite->VSBob->SaveBuffer,
                sizeof(SHORT)*vsprite->Width*vsprite->Height*vsprite->Depth);
            }
            if (vsprite->VSBob->DBuffer != NULL) {
                if (vsprite->VSBob->DBuffer->BufBuffer != 0) {
                    FreeMem(vsprite->VSBob->DBuffer->BufBuffer,
                    sizeof(SHORT)*vsprite->Width*vsprite->Height*vsprite->Depth);
                }
                FreeMem(vsprite->VSBob->DBuffer, sizeof(struct
                DBufPacket));
                FreeMem( vsprite->VSBob, sizeof(struct Bob));
            }
            if (vsprite->CollMask != NULL) {
                FreeMem(vsprite->CollMask, sizeof(WORD)*vsprite-
                >Height*vsprite->Width);
            }
            if (vsprite->BorderLine != NULL) {
                FreeMem(vsprite->BorderLine, sizeof(WORD)*vsprite-
                >Width);
            }
            FreeMem(vsprite, sizeof(struct VSprite));
        }
    }
}

/*-----*/
/* Cette fonction crée un bob */
/*-----*/
struct Bob
cree_bob(largeur,hauteur,profondeur,image,planepick,planeonoff,x
,y,flags)
{
    SHORT largeur; /* Largeur en pixels du Bob */
    SHORT hauteur; /* hauteur en lignes du Bob */
    SHORT profondeur; /* nombre de bitplanes */
    WORD *image; /* image associée au Bob */
    SHORT planepick, planeonoff;
    SHORT x,y; /* la position du Bob lors de sa création */
    SHORT flags; /* les flags divers associés au Bob */
    {
        struct Bob *b;
        struct VSprite *v;

        SHORT nb_mot;

        nb_mot = (largeur+15)/16;
        if(( v =
        cree_vsprites(hauteur,image,NULL,x,y,nb_mot,profondeur,flags)) ==
        0)
            return(0);

        v->PlanePick = planepick;
        v->PlaneOnOff = planeonoff;

        if(( b = (struct Bob *)AllocMem(sizeof(struct Bob),MEMF_PUBLIC |
        MEMF_CLEAR)) == 0)
            return(0);

        v->VSBob = b;
        b->Flags = 0;

        if((b->SaveBuffer = (WORD
        *)AllocMem(sizeof(SHORT)*nb_mot*hauteur*profondeur,
        MEMF_CHIP|MEMF_CLEAR)) == 0)
        {
            FreeMem(b,sizeof(struct Bob));
            return(0);
        }

        b->ImageShadow = v->CollMask;
        b->Before = NULL;
        b->After = NULL; /* les priorités d'affichages seront
        indiquées plus tard */
        b->BobVSprite = v;
        b->BobComp = NULL; /* il ne s'agit d'une partie d'un objet plus
        complexe */
        b->DBuffer = NULL; /* il n'y a pas de double buffer */

        return(b);
    }

    /*-----*/
    /* Cette fonction crée un VSprite. */
    /*-----*/
    struct VSprite *cree_vsprites(hauteur, image, tablecol, x,
    y,nb_mot,profondeur,flags)
    SHORT hauteur;
    WORD *image;
    WORD *tablecol;
    SHORT x, y;
    SHORT nb_mot,profondeur,flags;
    {
        struct VSprite *vsprite;

        if ((vsprite = (struct VSprite *)AllocMem(sizeof(struct
        VSprite),
        MEMF_PUBLIC | MEMF_CLEAR)) == 0)
            return(0);

        vsprite->Flags = flags; /* C'est un bob */

        vsprite->Y = y;
        vsprite->X = x;

        vsprite->Height = hauteur;
        vsprite->Width = nb_mot;

        vsprite->Depth = profondeur;

        vsprite->MeMask = 1;
        vsprite->HitMask = 1;

        vsprite->ImageData = image; /* associe l'image au vsprite */

        if ((vsprite->BorderLine = (WORD
        *)AllocMem(sizeof(WORD)*nb_mot,
        MEMF_PUBLIC | MEMF_CLEAR)) == 0)
        {
            FreeMem(vsprite, sizeof(struct VSprite));
            return(0);
        }

        if ((vsprite->CollMask = (WORD
        *)AllocMem(sizeof(WORD)*hauteur*nb_mot,MEMF_CHIP | MEMF_CLEAR))
        == 0)
        {
            FreeMem(vsprite, sizeof(struct VSprite));
            FreeMem(vsprite->BorderLine, nb_mot*sizeof(WORD));
            return(0);
        }

        vsprite->SprColors = tablecol;
        vsprite->PlanePick = 0x00;
        vsprite->PlaneOnOff = 0x00;

        InitMasks(vsprite);
        return(vsprite);
    }
}
```

Par Herr Doktor von GlutenStimelImDorf



Transactor

Règlement et Proposition d'Article

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

Abonnement à L'ANT et Anciens Numéros

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

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

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

Bon de commande à découper ou photocopier (pas de recopie manuelle SVP) et à retourner correctement rempli et accompagné de votre règlement à **MCM Europe, 16, Quai Jean-Baptiste Clément, 94140 Alfortville, France.**

OUI, je m'abonne à *Amiga NewsTech*. Je choisis la formule :

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

Je commande les anciens numéros suivants :

- | | | | |
|-------------------------------|---|-------------------------------|---|
| ☐ n°20 | ☐ (avec disquette) | ☐ n°25 | ☐ (avec disquette) |
| ☐ n°21 | ☐ (avec disquette) | ☐ n°26 | ☐ (avec disquette) |
| ☐ n°22 | ☐ (avec disquette) | ☐ n°27 | ☐ (avec disquette) |
| ☐ n°23 | ☐ (avec disquette) | ☐ n°28 | ☐ (avec disquette) |
| ☐ n°24 | ☐ (avec disquette) | | |

au tarif unitaire de 45 F (60 F avec disquette), pour un total de

F.

Nom :
 Prénom :
 Adresse :
 Code Postal :
 Ville :
 Pays :