LE MAGAZINE DES PROGRAMMEURS SUR AMIGA - N°25 SEPT 1991

AMIGA NEWS-TECH

EN VENTE UNIQUEMENT PAR ABONNEMENT

Dessin de Olivia De Berardinis tiré du CD-Rom "Mac" Exotica-Rom édité par Gazelle Technologies et distribué en France par Avancée, converti à l'aide d'ImageLink.

45 F.

Ce numéro 25 de l'Amiga News-Tech a été tiré à 3500 exemplaires (3012 abonnés enregistrés le 09.08.91).
Ce magazine est une publication Commodore Revue Sarl. 5, rue de la Fidélité - 75010 PARIS.

EDITO

Fin de notre grand roman-feuilleton de l'été : l'ANT ne changera pas de formule d'abonnement. Lisez la rubrique Requester pour en savoir plus.

Les Rom Kernal Manuals en français intéressent décidément beaucoup de monde. Laissez-nous le temps de régler les problèmes de copyright avec Commodore, de faire la traduction, quand même, et on pourra vous les proposer.

Le DevKit ANT est aux choux. Nous n'avons pas pu obtenir toutes les autorisations nécessaires, malgré tout la bonne volonté que nous y mettions. Du coup, le concours lecteur disparaît, au profit d'un Transactor revu et corrigé. Le principe est le même, il n'y a simplement plus rien à gagner. Désolé. Les précédents gagnants seront contactés personnellement pour voir comment on peut les dédommager.

L'ANT recherche des collaborateurs... Si vous pensez maîtriser parfaitement un sujet, quel qu'il soit, système ou style démo, et désirez partager vos connaissances avec vos petits camarades abonnés, n'hésitez pas à nous contacter. Le numéro de téléphone est inscrit dans l'Ours, là, dessous.

Stéphane Schreiber

OURS

L'Amiga NewsTech (ANT pour les intimes) est une GPM (Gentille Publication Mensuelle) disponible uniquement par abonnement et éditée par *Commodore Revue SARL, sise au 5, rue de la Fidélité, 75010 Paris*. Notre GNT (Gentil Numéro de Téléphone) est le **(1) 42 469 290.**

Jean-Yves Primas est notre GDP (Gentil Directeur de la Publication). Il est tout brouté depuis le GPMC (Grand Prix de Magny-Cours) et ma foi, sans être paloplobe, ça lui va à ravir. Mesdames, composez le **(1) 42 471 216** pour lui dire deux mots, doux ou pas.

Yves Huitric est notre GRC (Gentil Rédacteur en Chef) et notre GM (Gentil Maquettiste) en même temps. Il est tout barbu depuis des années et ma foi, sans être barbophile, ça lui va très bien.

Stéphane Schreiber est notre GRC-A (Gentil Rod'Chef Adjoint). Il est rondouillet depuis qu'il est tout petit et ma foi, sans être boulouloplule, ça lui donne un charme indéniable.

Pascal Amiable, Patrick Chazé, Jérôme Etienne, Loïc Far, François Gueugnon, François Lionet, et Max sont nos GP (Gentils Pigistes) et ma foi, sans être scribouillardophobe, hein, bon, on va pas citer toutes leurs qualités.

MCM Europe est notre GSV (Gentille Société de VPC) et c'est elle qui faut appeler lorsque vous avez des ennuis avec votre abonnement, au **(1) 43 789 210**. Ou écrivez-lui au *1, Quai JB Clément, 94150 Alfortville*.

ColorWay est notre GF (Gentil Flasheur). Son adresse est la même que la notre : *5, rue de la Fidélité, 75010 Paris*, mais son numéro de téléphone est le **(1) 40 708 787.**

Quant à notre GI (Gentil Imprimeur), il s'agit de NIC, sis à Reims, et dont le numéro de téléphone est le **(16) 26 075 787.**

ExecBase : PROGRAMMER

Sur une machine multitâche telle que l'Amiga, la programmation ré-entrante n'est pas forcément souvent utilisée : un programme peut être chargé plusieurs fois et cohabiter en mémoire sans problème particulier avec une ou plusieurs copies de lui-même.

Ceci est plus particulièrement vrai pour les grosses applications (traitement de texte, MIDI...) et les utilitaires (copieur, blanker, etc.). Il arrive cependant assez souvent que l'on ait besoin d'écrire un utilitaire particulier, assimilable à une commande AmigaDOS standard, par exemple un Copy amélioré ou un Dir plus souple. Dans ce cas, programmer ré-entrant permet par la suite de laisser ce programme résident en mémoire (commande Resident du CLI/Shell) avec tous les avantages que celà implique : fragmentation de la mémoire supprimée (une seule copie du programme étant chargée en mémoire) et accélération notable de l'exécution des fichiers Script (commande Execute) utilisant le programme. Le seul inconvénient de ce procédé réside dans le fait que la mémoire occupée par le programme résident n'est pas libérée lorsque celui-ci a terminé son travail ; AmigaDOS l'y laisse en prévision d'une autre invocation.

Il est également obligatoire, lorsque l'on écrit une bibliothèque Exec, que son code puisse être partagé par plusieurs tâches pouvant y accéder simultanément (c'est d'ailleurs pourquoi tout le système d'exploitation de l'Amiga repose sur le principe des listes et des structures).

PROGRAMMES RESIDENTS

Dans la version actuelle du système d'exploitation (1.3), ni le CLI ni le Workbench n'utilisent les programmes résidents ; ils chargent quoiqu'il arrive une nouvelle copie du programme en mémoire et l'en suppriment lorsqu'il se termine. Le Shell, lui, commence par parcourir la liste des commandes rendues résidentes (dans la Startup-Sequence ou plus tard, manuellement) avant de décider s'il doit charger la commande appelée depuis le disque ou non.

Ainsi, la Startup-Sequence standard telle qu'elle est fournie par Commodore-Amiga contient normalement les lignes

```
Resident c:Resident PURE ADD
Resident c:Mount PURE ADD
```

qui ont pour effet de charger les commandes Resident et Mount en mémoire une bonne fois pour toutes. Plus tard dans la Startup-Sequence, lorsque l'on "mountera" les devices SPEAK:, PIPE: et AUX:, la copie résidente de Mount sera utilisée.

Dans la théorie, n'importe quel programme peut-être rendu résident : il suffit de positionner le bit de protection *p* pour que ça marche. Des problèmes risquent de survenir dès lors que plusieurs Shells essayent d'accéder à la même commande en même temps : on se retrouve alors dans le cas où deux tâches partagent les mêmes adresses, aussi bien pour le code que pour les données. Si l'on n'y prend pas garde lors de la rédaction du programme, celà peut causer de grosses difficultés, voire même mener tout droit au Gourou.

RE-ENTRANCE

Le principe de la ré-entrance est simple : il suffit de faire attention à ce que les variables globales, donc partagées par les différents accesseurs au code, ne soient pas jamais modifiées. Cela est bien entendu valable pour les pointeurs (par exemple sur des zones de mémoire allouées ou sur des fichiers ouverts) et pour les compteurs de boucles. Si l'assembleur permet d'écrire assez facilement des programme ré-entrants (par l'utilisation des registres du processeur et de la pile), c'est beaucoup moins évident en C.

Dans ce langage, le problème se pose au niveau des variables globales : alors que les variables locales à une fonction sont allouées sur la pile (au moyen du couple d'instructions 68000 LINK et UNLK), les variables globales sont allouées statiquement, dans une section de type BSS. Celà peut bien sûr être évité, pour peu que l'on prenne la peine de se souvenir de quelques points importants. Voici donc les règles à suivre pour écrire un programme C ré-entrant.

Tout d'abord, le Startup-Code doit être modifié. C'est lui qui se charge d'ouvrir la dos.library et les canaux d'entrées-sorties standard stdin, stdout et stderr et de décomposer la ligne de commande CLI/Shell en un tableau de pointeurs nommé argv. Avec le Startup-Code standard, AStartup.obj, toutes ces valeurs sont stockées dans les variables globales statiques _DOSBase, _stdin, _stdout, _stderr, _argc et _argv. C'est pourquoi un nouveau Startup-Code a été écrit. Baptisé RStartup.obj, il accomplit les mêmes opérations mais alloue dynamiquement ces variables. Ainsi, à chaque nouvelle invocation du programme résident, elles seront correctement actualisées. Les abonnés à la disquette en trouveront le code source assembleur et le programme objet dans le répertoire approprié. Avec RStartup.obj, les variables globales _DOSBase et _SysBase sont les seules à pouvoir être utilisées sans risque. Pour _stdin, _stdout, _stderr, _argc et _argv, il faudra ruser.

Comme mentionné précédemment, les bibliothèques système sont ré-entrantes. Celà veut (heureusement !) dire que l'on peut continuer à les utiliser dans un programme ré-entrant sans risque de voir apparaître le Gourou. Le problème se pose toutefois au niveau des adresses de base des bibliothèques utilisées (IntuitionBase, GfxBase, etc.). En effet, le linker attend toujours qu'elles soient déposées dans des variables globales afin de pouvoir résoudre les références. La solution consiste donc à effectivement déclarer ces adresses de base deux fois : une première fois locales et une seconde fois globales. La variable locale recevra le résultat d'OpenLibrary() et n'initilisera la variable globale que si l'ouverture a réussi. Les adresses de base étant fixes (une fois la librarie ouverte, elle n'est pas déplacée par Exec), celà fonctionnera très bien.

Revenons-en à _stdin, _stdout, _stderr, _argc et _argv. Ces deux dernières sont transmises à main() par la pile et peuvent être utilisées sans problème. Pour les trois premières, il conviendra par contre d'ouvrir explicitement les canaux d'entrées-sorties à chaque invocation du programme, avec les fonctions Input() et Output() de la dos.library. Ce qui impose

également des restrictions au niveau des fonctions qui utilisent ces pointeurs, à savoir l'incontournable printf(), mais aussi getch(), etc. Il vaut mieux utiliser à la place fprint() avec le résultat d'Output() pour argument et fgetc() avec le résultat d'Input() pour argument.

Evidemment, garder une multitude de variables locales n'est pas forcément très pratique, surtout lorsqu'il s'agit de les passer en paramètres à des fonctions. C'est pourquoi je vous conseille fermement d'utiliser une seule structure, définie par vos soins et allouée dynamiquement, qui contiendra toutes vos variables normalement globales. Il vous suffit alors de conserver un pointeur local à main() sur cette structure, que vous passerez aux fonctions en ayant besoin. Le programme d'exemple en fin de cet article démontre ce procédé, inspiré d'un article de Carloyn Sheppner paru dans le Bus Commodore.

LES REGLES DU JEU

Voici, en résumé, les règles à suivre pour écrire un programme ré-entrant sur l'Amiga :
- ne jamais utiliser les variables globales _stdin, _stdout, _stderr, _errno et _WBenchMsg ;
- pour les entrées-sorties, utiliser des variantes des fonctions printf(), puts(), getchar() etc. qui se servent de Output() et d'Input() plutôt que de _stdout et _stdin, ou bien utiliser fprintf() et fgetc() ;
- ne pas utiliser de variables globales ni statiques. Les seules globales acceptables sont les constantes : chaines de caractères, structures NewWindow, NewScreen, Gadget, Menu etc. qui ne seront pas modifiées par la suite ;
- un programme ré-entrant n'a pas besoin d'être relogeable.

```
/*
* Exemple de programme ré-entrant en C.
* Se contente d'ouvrir une fenêtre et d'attendre
* qu'on la ferme.
*
* Compilation (SAS/Lattice 5.10) :
*  LC -d4 -v -Li -ck -tr -ms -y -N -La -Lc -Ld -Lv Reentrant.c
*
* Linkage :
*       FROM RStartup.obj,Reentrant.o
*         TO Reentrant
*    LIBRARY LIB:amiga.lib,LIB:lc.lib
*  SMALLCODE
*  SMALLDATA
*  [NODEBUG]
*  [VERBOSE]
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <graphics/gfxbase.h>
#include <graphics/text.h>
#include <libraries/dos.h>

/* Voici la structure qui définit nos variables */
struct Variables
{   ULONG stdin;
    ULONG stdout;
    struct IntuitionBase *intbase;
    struct GfxBase *gfxbase;
    struct Window *win;
    struct IntuiMessage *imsg;
};

/* Les constantes peuvent être globales */
/* Ca, c'est pour ouvrir notre fenêtre */
struct NewWindow nw=
{   0, 11, 420, 40,
    0, 1,
    CLOSEWINDOW,
    WINDOWCLOSE|WINDOWDRAG|WINDOWDEPTH|NOCAREREFRESH,
    NULL, NULL,
    "Fermez moi",   /* Pas besoin d'être relogeable */
    NULL, NULL,
    -1, -1, -1, -1,
    WBENCHSCREEN
};

/* petit texte à afficher dans la fenêtre */
char *msg = "Programme réentrant : lancez-le d'un autre Shell !";

/* Deux variables gloables pour le linker */
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;

/* Prototypes des fonctions */
VOID main(int argc, char **argv);
VOID CleanExit(char *str, struct Variables *VARS);

/* Et voilà main() */
void main(int argc, char **argv)
{
struct Variables *VARS;    /* VARS est local à main() */

/* Allocation du buffer des variables */
    VARS = (struct Variables *)AllocMem(sizeof(struct Variables),
MEMF_CLEAR);
    if (!VARS)
        exit(RETURN_FAIL);

/* Recherche des canaux d'entrées-sorties */
    VARS->stdin  = Input();
    VARS->stdout = Output();

/*
* Ouverture d'intuition.library
* C'est notre variable locale VARS->intbase qui contient le pointeur
* sur l'adresse de base de la bibliothèque. Toutefois, pour que le
* linker soit content, il faudra également initialiser la variable
* globale IntuitionBase si l'ouverture réussit
*/
    VARS->intbase = (struct IntuitionBase
*)OpenLibrary("intuition.library", 33L);
    if (!VARS->intbase)
        CleanExit("Pas d'intuition.", VARS);
    else
        IntuitionBase = VARS->intbase;

/* Idem pour la graphics.library */
    VARS->gfxbase = (struct GfxBase *)OpenLibrary("graphics.library",
33L);
    if (!VARS->gfxbase)
        CleanExit("Pas de graphics.", VARS);
    else
        GfxBase = VARS->gfxbase;

/* Ouverture de la fenêtre */
    VARS->win = (struct Window *)OpenWindow(&nw);
    if (!VARS->win)
        CleanExit("Pas de fenêtre !", VARS);

/* Ecriture du message */
    SetAPen(VARS->win->RPort, 1);
    SetBPen(VARS->win->RPort, 0);
    SetDrMd(VARS->win->RPort, JAM2);
    Move(VARS->win->RPort, 5, 24);
    Text(VARS->win->RPort, msg, strlen(msg));

/* Attend la fermeture */
    WaitPort(VARS->win->UserPort);

/* Réponse normale aux messages Intuition */
    while( (VARS->imsg = (struct IntuiMessage
*)GetMsg(VARS->win->UserPort)) )
        ReplyMsg((struct Message *)VARS->imsg);

/* C'est fini, on s'casse */
    CleanExit(NULL, VARS);
}

/*
* CleanExit()
*/
VOID CleanExit(char *str, struct Variables *VARS)
{
UWORD rc;   /* encore une variable locale */

/* Fermeture de la fenêtre */
    if (VARS->win)
        CloseWindow(VARS->win);

/* Fermeture de graphics.library. On utilise la variable locale ç */
    if (VARS->gfxbase)
        CloseLibrary(VARS->gfxbase);

/* Fermeture d'intuition.library */
    if (VARS->intbase)
        CloseLibrary(VARS->intbase);

/* Affiche un texte dans la fenêtre Shell */
    if (str)
    {   fprintf(VARS->stdout, "%s", str);
        rc = RETURN_FAIL;
    }
    else
    {   fprintf(VARS->stdout, "Tout s'est bien passé.");
        rc = RETURN_OK;
    }

/* Ne pas oublier de libérer la mémoire des variables ! */
    FreeMem(VARS, sizeof(struct Variables));
    exit(rc);
}
```

Par Denis Jarril

AMOS : TRUCS EN VRAC

Allez, pousse toi, sâle clébard ! J'vous jure, on ne peut pas la laisser cinq minutes avec un traitement de texte sans qu'elle ne commence à raconter des âneries. Elle vous a encore fourgué une de ses soi-disant extraordinaires démos? Elle fait le coup à chaque fois. Bon, maintenant on va faire de la belle, de la vraie programmation AMOS.

AMOS contient des instructions de compactage d'image accessibles en mode direct, mais pas de programme permettant une exploitation simple de ces instructions. On aimerait (enfin, moi j'aimerais bien) pouvoir découper des parties d'images, réduire la taille des écrans etc... Le petit programme qui suit fait tout ça !

Voici son mode d'emploi. D'abord, choisir et charger l'image, puis fixer les limites avec les boutons gauche et droit de la souris. Pour compacter un écran, appuyer sur S (comme Screen ou SPack), une simple bitmap, appuyer sur P (comme Planes ou Pack). Le programme fabrique alors un petit écran intermédiaire dans lequel il recopie la portion choisie.

```
'--------------------
' Compacteur d'image
'--------------------
'
Curs Off : Fade 1
Screen Open 2,320,8,4,0 : Curs Off : Flash Off
'
Do
    '
    INFO["> Compacteur d'image <",300,1]
    '
    F$=Fsel$("*.Iff","","Entrez le nom de l'image")
    Exit If F$=""
    '
    Load Iff F$,0 : Wait Vbl
    NC=Screen Colour
    RES=Deek(Screen Base+18*4) and $8000
    EX=Screen Width : EY=Screen Height
    Screen Open 1,EX,EY,NC,RES
    Curs Off : Flash Off
    Screen Hide 1
    Screen Copy 0 To 1
    Wait Vbl
    '
    Screen 0
    Screen To Front 2
    '
    SX=0 : SY=0
    Box SX,SY To EX-1,EY-1
    Limit Mouse X Hard(0),Y Hard(0) To X Hard(EX),Y Hard(EX)
    '
    Do
        Do
            Wait Vbl
            X=X Screen(X Mouse) : Y=Y Screen(Y Mouse)
            K=Mouse Key
            K$=Inkey$
            I$="  X:"+Str$(X)+" / Y:"+Str$(Y)+"  "
            INFO[I$,Y,0]
            Exit If K
            Exit If K$<>"",2
        Loop
        '
        Screen Copy 1,SX,SY,EX,SY+1 To 0,SX,SY
        Screen Copy 1,SX,EY-1,EX,EY+1 To 0,SX,EY-1
        Screen Copy 1,SX,SY,SX+1,EY To 0,SX,SY
        Screen Copy 1,EX-1,SY,EX+1,EY To 0,EX-1,SY
        '
        If K=1
            If X<EX and EX-X>=16 : SX=X : End If
            If Y<EY and EY-Y>=16 : SY=Y : End If
        End If
        If K=2
            If X>SX and X-SX>=16 : EX=X : End If
            If Y>SY and Y-SY>=16 : EY=Y : End If
        End If
        '
        Box SX,SY To EX-1,EY-1
    Loop
    K$=Upper$(K$)
    If K$="Q" : Edit : End If
    INFO["Je compacte!",Y,1]
    '
    Screen Open 0,EX-SX,EY-SY,NC,RES : Curs Off : Flash Off
    Get Palette(1)
    Screen Copy 1,SX,SY,EX,EY To 0,0,0 : Get Palette(1)
    Screen Close 1
    Screen To Front 2 : Wait Vbl
    '
    If K$="S"
        Spack 0 To 10,0,0,EX-SX,EY-SY
    End If
    If K$="P"
        Pack 0 To 10,0,0,EX-SX,EY-SY
    End If
    I$="Taille de la banque:"+Str$(Length(10)) : INFO[I$,Y,F]
    S$=Fsel$("*.Abk","","Entrez le nom de la banque","à sauver (numéro 10)")
    If S$<>""
        Save S$,10
    End If
Loop
Edit
'
Procedure INFO[I$,Y,F]
    '
    Screen 2
    If F : Clw : End If
    If Y<150
        Screen Display 2,,275,,
    Else
        Screen Display 2,,50,,
    End If
    Centre I$
    Screen 0
End Proc
```

IMPRIMER AU CHOIX SOUS AMOS ET CLI

Par imprimer, on entend afficher... Le compilateur permet de produire des utilitaires CLI : l'affichage du Workbench n'est pas perturbé par le démarrage du programme, celui-ci est toujours en "AMOS TO BACK". Seulement voilà, il faut pouvoir imprimer dans la fenêtre CLI courante. Je vous propose une procédure gérant les deux possibilités : si AMOS est là, on imprime à l'écran normal, sinon, si le CLI est là, on demande l'output standard (fonction Output() de la dos.library, soit DosCall(-60)) et on imprime la chaîne.

```
Procedure CPRINT[A$]
    If Amos Here
        Print A$
    Else
        H=Doscall(-60)
        If H
            A$=A$+Chr$(10)
            Dreg(1)=H
            Dreg(2)=Varptr(A$)
            Dreg(3)=Len(A$)
            F=Doscall(-48)
        End If
    End If
End Proc
```

Le mois prochain, nous irons beaucoup plus loin dans la programmation système en AMOS, puisque nous ouvrirons un device. Carrément.

UNE PROCEDURE DE COPIE DE FICHIER

La petite procédure qui suit copie le fichier SS en fichier DS. Son grand avantage est de ne pas consommer de mémoire : tout le travail se fait dans les maigres 8K de buffer de chaînes. Elle est par contre totalement inutilisable avec un seul drive !

```
Procedure FCOPY[S$,D$]
    Open In 1,S$
    Open Out 2,D$
    LF=Lof(1)
    Do
        Exit If P>=LF
        L=Min(2048,LF-P)
        A$=Input$(1,L)
        Print #2,A$;
        Add P,L
    Loop
    Close 1
    Close 2
End Proc
```

Daisy : L'AVIS DES BETES

Après avoir été scandaleusement empêchée de parler le mois dernier, moi, Daisy, me suis précipitée sur le clavier ! Je vais enfin pouvoir vous donner le listing d'une de mes meeerveilleuses démOS.

Cette démo est en fait très simple. Toute l'animation est faite par la petite chaîne AMAL, un simple Move. Ce Move dirige chaque bob sur la souris. L'astuce consiste à initialiser les bobs avec un certain décalage dans le temps, décalage qu'ils conserveront pendant le fonctionnement du programme. Les mouvements des bobs seront proches l'un de l'autre, sans pour autant être les mêmes.
La boucle du programme fait tourner la souris pour faire bouger les bobs. Vous pouvez prendre la relève en appuyant sur le bouton gauche : le pointeur apparaît alors. Pour mieux comprendre, vous pouvez également enlever le HIDE ON au début du programme.

Ce programme utilise aussi plusieurs astuces pour avoir beaucoup de bobs au 1/50ième de seconde. D'abord, j'ouvre un affichage dual-playfied avec un écran pour le fond et un écran pour les bobs. Comme l'écran contenant les bobs ne contient qu'eux, il n'est pas nécessaire de sauver le décor. L'instruction Set Bob n,1,, indique à AMOS d'effacer les bobs à l'aide de la couleur zéro. A chaque affichage, on gagne un coup de blitter ! De plus, les bobs ne sont qu'en deux couleurs (1 bitplane), ce qui fait également gagner de la vitesse.

Vous pouvez modifier à loisir les paramètres de décalage et de vitesse au début du programme. Comme dirait monsieur Citroën, ça change tout !

Bon, voila revenir mon maître abhorré - euh pardon, je voulais dire : adoré ! Il faut que je fasse semblant de programmer le compilateur, sinon ce soir, rien à manger. Quelle vie de chien !

```
'------------------
' Daisy Dem'OS II
' Par Daisy Lionet
'------------------
'
SPEED=45
DELAY=7
SMAX=15
BONE=True
Hide On
'
'--------------------------
' Initialisation de l'écran
'--------------------------
SXSC=368 : SYSC=300
Screen Open 0,SXSC,SYSC,2,Lowres
Screen Display 0,112,30,,
Curs Off : Palette 0,0 : Wait Vbl
Screen Open 1,SXSC,SYSC,2,Lowres
Screen Display 1,112,30,,
Curs Off : Palette 0,0 : Wait Vbl
Dual Playfield 1,0
Dual Priority 0,1
Screen 1 : For N=0 To 350 : Print "DAISY!"; : Next
Limit Mouse 0,0 To 500,500
'----------------
' Dessin de l'OS
'----------------
Screen 0
If BONE
    SX=12 : SY=3 : R=4
    BONE[100,100,SX,SY,R]
Else
    R=10
    BALL[100,100,R]
End If
Get Sprite 1,100-SX-R,100-SY-R To 100+SX+R+1,100+SY+R+1
Hot Spot 1,$11
Cls
'---------------
' Fin de l'init
'---------------
Double Buffer
Screen 1 : Fade 1,0,$F50,0,0,0,0,0,0,0,$FF0
Screen 0
'------------------
' AMAL très simple
'------------------
A$=A$+"Loop: Move XS(0,RA)-X,YS(0,RB)-Y,"+Str$(SPEED)+"; Jump Loop"
For N=1 To SMAX
    Bob N,-100,-100,1
    Set Bob N,1,,
    Channel N To Bob N
    Amal N,A$
    Amal On
    Wait 6
Next
'--------------------
' Boucle d'animation
'--------------------
D=1
Do
    NN=Rnd(4)+1
    DX#=Rnd(SXSC/2)+SXSC/4 : DY#=Rnd(SYSC/2)+SYSC/4
    R#=Rnd(SYSC/2)+32
    S#=(2*Pi#)/(Rnd(100)+10)
    D=Rnd(1)
    For N=0 To NN
        If D
            For P#=0.0 To 2*Pi# Step S#
                X=DX#+R#*Sin(P#) : Y=DY#+R#*Cos(P#)
                X Mouse=X Hard(X) : Y Mouse=Y Hard(Y)
                Amreg(0)=X Mouse : Amreg(1)=Y Mouse
                Wait Vbl
                Exit If Mouse Key,2
            Next
        Else
            For P#=2*Pi# To 0.0 Step -S#
                X=DX#+R#*Sin(P#) : Y=DY#+R#*Cos(P#)
                X Mouse=X Hard(X) : Y Mouse=Y Hard(Y)
                Amreg(0)=X Mouse : Amreg(1)=Y Mouse
                Wait Vbl
                Exit If Mouse Key,2
            Next
        End If
    Next
    Exit If Mouse Key=2
    If Mouse Key=1
        Show
        While Mouse Key=1
            Amreg(0)=X Mouse
            Amreg(1)=Y Mouse
            Wait Vbl
        Wend
        Hide
    End If
Loop
Edit
'
Procedure BONE[X,Y,SX,SY,C]
    Bar X-SX,Y-SY To X+SX,Y+SY
    For R=1 To C
        Circle X-SX,Y-SY,R
        Circle X-SX,Y+SY,R
        Circle X+SX,Y-SY,R
        Circle X+SX,Y+SY,R
    Next
End Proc
'
Procedure BALL[X,Y,C]
    For R=1 To C
        Circle X,Y,R
    Next
End Proc
```

par Daisy et FrançoisLIONET

AMOS NEWS

Le compilateur AMOS est enfin sorti ! Nous en avons reçu une version directement de chez Mandarin Software (pardon, je voulais dire : Europress Software), en anglais bien sûr, mais la version française est en préparation chez Ubi Soft. Bonne nouvelle, un update vers la version 1.3 de l'interpréteur est fourni sur les disquettes. Cette version 1.3 comprend quelques fonctions supplémentaires et est pas mal optimisée d'un peu partout.

Quant à AMOS 3D, comme disait je ne sais plus qui, il immine à fond la caisse.

AREXX : LA VOIE ROYALE (IV)

Nous avons analysé, dans le précédent numéro, la première partie de l'instruction PARSE, celle qui correspond à la source de la chaîne de caractères. Il nous reste à analyser la deuxième partie de cette instruction, qui correspond au traitement de la chaîne acquise.

Imaginons que la chaîne acquise soit composée de plusieurs mots séparés par des espaces : "Ceci est la chaîne à traiter abcdefghijklmnopqrstuvwxyz". Nous allons extraire les mots, parties ou groupes de mots, selon un template (mot anglais signifiant "modèle"). Les différentes formes d'analyse concernent uniquement la définition des séparateurs de chaîne.

ANALYSE PAR MARQUE (TOKEN)

Dans cette forme, la partie traitement est composée de noms de variables qu'on utilisera ultérieurement au cours du déroulement du programme. L'écriture formelle de l'analyse par marque est la suivante :

```
PARSE 'source' data1 data2 nom3
```

Les noms des variables sont ceux que l'on souhaite ; ils sont séparés par un espace, qui est le séparateur standard. Plusieurs cas peuvent se présenter :
- le nombre de variables est *exactement égal* au nombre de mots de la chaîne acquise. Dans ce cas, chaque mot de la chaîne acquise est placé dans une variable, dans le même ordre. Ainsi, utilisée avec notre chaine précédente, l'instruction `PARSE 'source' data1 data2 nom3 nom4 toto dimension a15` placera "Ceci" dans *data1*, "est" dans *data2*, "la" dans *nom3*, "chaîne" dans *nom4*, "à" dans *toto*, "traiter" dans *dimension* et "abcdefghijk lmnopqrstuvwxyz" *dans a15*. Les variables utilisables sont celles qui sont légales dans ARexx, tableaux y compris. On pourra les utiliser selon les besoins.
- le nombre de variables est inférieur à celui de mots de la chaîne acquise. L'affectation débute de la même façon (un mot par variable), mais la dernière variable contient tout le reste de la chaîne, de façon à ne pas en "perdre" un morceau au vol. Ainsi, `PARSE 'source' data1 data2 nom3` placera "Ceci" dans *data1*, "est" dans *data2* et "la chaîne à traiter abcdefghijklmnopqrstuvwxyz" dans *nom3*.
- le nombre de variables est plus grand que le nombre de mots de la chaîne acquise. L'affectation se fera alors comme précédemment, un mot par variable, et les variables supplémentaires resteront vides. A ce propos, si une variable n'est pas affectée, ARexx la remplace par son nom. Par exemple, SAY VariableVide écrira littéralement à l'écran la chaine VariableVide. Ce n'est pas un cas d'erreur.
- une variable spéciale, représentée par le point "." peut être utilisée. Sa particularité est de fournir une place dans l'instruction PARSE uniquement pour qu'elle fonctionne correctement, mais ce qui est dans le point n'est pas récupérable. Il s'agit d'une variable jamais affectée, une espèce de filtre. Ainsi, `PARSE 'source' . data1` sépare la chaîne acquise en deux morceaux : le premier mot dans le point, le reste dans *data1*, qui reste la seule partie récupérable. La position du ou des points dans la définition est laissée à la discrétion du programmeur.

ANALYSE PAR POSITION

Cette forme est plutôt utilisée lorsque la chaîne acquise est monolithique, par exemple 'abcdefghijklmnopqrstuvwxyz'. Le séparateur va être défini par un chiffre indiquant la position de coupure.
- analyse par position absolue : `PARSE 'source' 3 data1 5 data2 10 data3` met "cd" dans *data1*, "efghi" dans *data2* et "jklmnopqrstuvwxyz" dans *data3*. La valeur assignée à la variable est la sous-chaîne commençant à la position courante et allant jusqu'à, mais sans l'inclure, la position notée.
- analyse par position relative : dans ce cas, on indique par + ou - le décalage par rapport à la position courante. `PARSE 'source' 3 data1 +9 data2 -6 data3` met "cdefgh" dans *data1*, "ijklmnopqrstuvwxyz" dans *data2* et "cdefghijklmnopqrstuvwxyz" dans *data3*.

Ces deux formes peuvent être mélangées. Cela nécessite une vue claire de ce que l'on veut, car il est rapidement possible de "perdre les pédales" ! Petit truc : effectuez toujours mentalement la différence entre deux délimiteurs ; des valeurs négatives ramènent au début de la chaine source.

ANALYSE PAR FIGURES

Dans ce mode, on décrit le séparateur par une sous-chaîne qui existe dans la chaîne acquise. `PARSE 'source' data1 'g' data2 'qrs' data3` met "abcdef" dans data1, "hijklmnop" dans data2 et "tuvwxyz" dans data3. g et qrs ont été définis comme des séparateurs et sont exclus des sous-chaînes.

ANALYSE MULTIPLE

Il est possible d'effectuer plusieurs analyses sur la même chaîne dans la même instruction. Attention : la chaîne traitée reste toujours la chaîne acquise initiale ; les différents traitements ne l'altèrent pas. Le mot-clé est *WITH*. La notation formelle est la suivante :

```
PARSE 'source' WITH  analyse1, analyse2, ..., analyseN
```

où analyse1 à *analyseN* représentent chacune l'une des formes d'analyse précédemment étudiées. Notons que si une analyse par position est utilisée, il est nécessaire de donner la première position derrière le mot-clé WITH.

Nous voici arrivés à la fin des descriptions élémentaires de l'instruction PARSE. A peu près toutes les combinaisons sont possibles entre 'source' et 'traitement' et à l'intérieur de 'traitement'. La richesse est telle qu'il n'est pas possible de décrire tous les cas. Une expérience est parfois nécessaire.

Nous allons terminer le chapitre des éléments théoriques par les instructions d'entrée/sortie. Après quoi nous pourrons commencer les exercices pratiques.

LECTURE ET ECRITURE D'UN FICHIER

La procédure de lecture ou d'écriture est formellement la même. Le principe en est le suivant : ARexx n'est pas en liaison directe avec le fichier extérieur. Il faut définir une variable logique qui permettra de converser avec l'extérieur. Cela est effectué par la fonction OPEN(). La forme générale est la suivante :

```
x = OPEN('varlogique', 'fichier externe', 'option')
```

- *varlogique* est une variable qui servira à 'interpeller' le fichier externe ;
- *fichier externe* est la définition du fichier extérieur comprenant éventuellement son chemin (path) complet ;
- *option* peut prendre trois valeurs essentielles : A pour accoler (append), R pour lire (read), W pour écrire (write).

On peut ouvrir autant de fichiers que l'on veut. Si l'on veut lire et écrire sur le même, il faut ouvrir une voie lecture et une autre écriture... indépendamment de notions de prudence dans une telle manoeuvre.

Dans la variable x, l'on trouve le résultat de l'opération sous forme booléenne (1 si l'opération a réussi, 0 sinon).

Comme toujours, si varlogique est le nom à utiliser, il doit être placé entre apostrophes (quotes) ; si c'est le contenu de varlogique qui doit être utilisé, c'est une variable sans quotes. Il en va de même pour les autres termes de l'instruction.

Enfin, un fichier ouvert doit être fermé à un moment ou à un autre. S'il est vrai que terminer le programme ferme les fichiers encore ouverts, ce n'est toutefois pas très élégant. "Exit gracefully", suggère le livre des recommandations aux développeurs. Pour ce faire, une instruction simple : `CLOSE(nomlogique).`

L'ouverture des fichiers externes peut se faire à n'importe quel moment du programme en cours, mais toutefois et bien entendu avant leur utilisation !

On peut ouvrir pour tous les cas optionnels prévus:
- les fichiers en :, dfx:, dhx:, prt:, ser:, par:, con:, etc. Nous en verrons différents exemples. Lorsque le fichier requiert une définition spécifique, comme par exemple CON:, celle-ci fait partie de la description, comme dans *CON:100/50/320/100/MaFenêtre*.

Il faut maintenant acquérir ou écrire des informations dans le fichier sélectionné. L'instruction de lecture est `x = READCH('nomlogique', longueur)`, celle d'écriture, `x = WRITECH('nomlogique', longueur).` Dans un cas comme dans l'autre, longueur indique le nombre de caractères à traiter. Pour la lecture, ce qui est réellement lu est placé dans x et peut être plus petit que ce qui est demandé si l'on arrive en fin de fichier. Pour l'écriture, on trouve dans x le nombre de caractères réellement écrits.

Ce sont ces deux instructions qui permettent d'avoir accès à un fichier de nature quelconque, puisque le caractère est l'atome des fichiers en dehors de toute signification relationnelle à l'intérieur de ces fichiers.

Ces deux instructions ont une version simplificatrice lorsque le fichier analysé est organisé en lignes avec un retour chariot en fin de ligne. C'est le cas des fichiers texte. Dans ce cas et uniquement dans ce cas, deux instructions similaires permettent de traiter des lignes plutôt que des caractères, libérant ainsi l'utilisateur du travail de comptage des caractères. Il s'agit de `x = READLN('varlogique')` et `x = WRITELN('varlogique')`, avec les mêmes définitions que pour *READCH()* et *WRITECH()*.

A RETENIR

L'instruction PARSE 'source' 'traitement' permet d'acquérir et de séparer une chaîne source en éléments différents selon plusieurs méthodes mixables ;
- les opérations d'entrées/sorties s'effectuent par l'intermédiaire d'une variable logique ;
- les commandes READLN() et WRITELN() ne peuvent être utilisées que sur des fichiers organisés en lignes (fichier texte), sinon, c'est le plantage assuré !

par François GUEUGNON

ORGANISATION PRATIQUE

Nous décrivons ci-après une organisation permettant de faire fonctionner ARexx aisément. Munissez-vous d'une disquette Workbench standard, de votre original d'ARexx, de vos domaines publics préférés, d'une tasse de café et d'un peu de patience.

Installez une disquette avec le DOS et ajoutez-y les logiciels constituant ARexx dans les directories adéquats. Ajoutez également l'éditeur de textes de votre choix (nous le nommons artificiellement "e"). Bien qu'il ne soit pas indispensable, le programme DMouse (domaine public de Matt Dillon), qui a pour avantage majeur - entre autres - d'activer la fenêtre sous le curseur de la souris sans avoir à cliquer, peut être très utile. Au lieu du Shell standard du Workbench 1.3, utilisez plutôt ConMan (fourni avec la disquette ARexx). Ensuite, éditez le programme suivant, destiné à charger les bibliothèques d'office, et nommez-le rexinit.rexx.

```
/* rexinit */
check=ADDLIB('rexxsupport.library', 0,-30,0)
check=ADDLIB('rexxsmathlib.library', 0,-30,0)
check=ADDLIB('rexxarplib.library', 0,-30,0)
check=ADDLIB('rexxsyslib.library', 0,-30,0)
say SHOW(clip)
say SHOW(files)
say SHOW(libraries)
say SHOW(ports)
exit
```

La Startup-Sequence peut alors s'écrire :

```
........
AddBuffers df0:50                  ;Ceci est pris directement dans
cd c                               ;la Startup-sequence standard
SYS:system/FastMemFirst            ;jusqu'à
BindDrivers                        ;
FF > NIL: -0                       ;
Resident c:execute Pure            ;
Mount NewCon:                      ;
FailAt 11                          ;
execute s:StartUpII                ;ici
cd /                               ;retour au répertoire principal
DMouse                             ;lancement de DMOUSE
RexxMast                           ;lancement de ARexx
RX Rexinit                         ;Fichier ARexx d'initialisation
Conman -c                          ;lancement de la fenêtre CONMAN
e                                  ;lancement de l'éditeur de texte
newcli "con:0/170/640/80/ARexx"    ;description et position de la
                                   ;fenêtre
TCO                                ;Ouverture de la fenêtre TRACE
```

Si tout se passe bien, après un certain remue-ménage du lecteur et différentes indications, l'éditeur devrait être sur l'écran le plus arrière, devant et en bas la fenêtre ConMan nommée ARexx et au milieu devant et en haut la fenêtre TCO nommée elle aussi ARexx. En déplaçant la souris sur les différentes fenêtres on les rend actives sans même cliquer dedans, par la vertu de DMOUSE.

TOOLBOX I : L'INTERPRETEUR Li

Cet interprêteur n'est autre que la version pour Amiga du très célèbre produit du Domaine Public XLisp existant sur de nombreux systèmes. Cette version a été écrite en C par David Betz pour ce qui concerne la partie interprêteur, l'interface Amiga étant dûe à François Rouaix.

Le but de cet article n'est pas de vous faire un cours sur Lisp, mais plutôt de vous inciter à le découvrir par vous-mêmes, au travers de ce produit accessible à toutes les bourses, et par voie de conséquence, d'accéder à un autre style de programmation.
Pour commencer, un rappel sur ce langage assez peu répandu sur Amiga. Il s'agit d'un des plus anciens langages de programmation. Ses premières implémentations ont vu le jour à la fin des années 60, quelques années seulement après Fortran. Lisp a été conçu par John Mc Carthy pour le traitement d'expessions symboliques (par opposition au traitement numérique). Il a été utilisé dès son origine pour écrire des programmes de calcul symbolique différentiel et intégrale, de logique mathématique et dans le cadre de la programmation de jeux. C'est de plus un langage interactif avec un environnement intégré. Ceci a pour conséquence que du cycle classique de développement édition texte -> compilation -> edition de lien -> chargement -> execution des langages compilés habituels, nous passons au cycle réduit lecture d'une expression -> évaluation (cycle propre à l'interprétation).

Ces vertus alliées à la simplicité de la syntaxe de Lisp, offrent la possibilité de programmer immédiatement un problème sans passer par des stades de déclarations de variables ou de types et finalement, le fait que Lisp soit basé sur la récursivité, en fait un excellent langage d'apprentissage et d'enseignement de la programmation.

INITIATION

Comme certaines mauvaises langues le prétendent, Lisp n'est pas l'acronyme de "List of Insipid and Stupid Parenthesis" mais plus simplement de LISt Programming. Ainsi que l'indique ce nom, Lisp traite essentiellement des listes. Une liste est quelque chose qui commence par une parenthèse ouvrante "(" et se termine par une parenthèse fermante ")", par exemple : (A N T 25).

Une liste est composée d'atomes qui peuvent êtres des entiers, des caractères ou des chaînes. Pour manipuler ces listes, on utilise un certain nombre de fonctions parmi lesquelles on peut citer les fonctions de base suivantes : car, cdr et cons qui permettent respectivement d'extraire le premier élement d'une liste, d'obtenir la liste privée de son premier élément, et de construire une nouvelle liste. La notation utilisée pour les appels de fonctions est la notation préfixée, c'est-à-dire la fonction suivie de ses arguments. En d'autres termes, le premier élément d'une liste sera interprêté comme une fonction, sauf indication contraire de votre part. Le symbole ' (apostrophe) permet d'éviter l'évaluation de l'expression qui le suit, qu'il s'agisse d'une liste ou bien d'un atome. Exemple :

```
(car '(A N T 25))
--> A
(cdr '(A N T 25))
--> (N T 25)
(cons 'A '(N T 25))
--> (A N T 25)
```

Il existe un grand nombre d'autres fonctions standard, telle que celle permettant la concaténation de deux listes : `(append '(A N T) '(25))` ou encore celle pemettant la définition d'autres fonctions : `(defun (<listes des paramètres>) <corps de la fonction>)`.

Lisp comprend également tout une batterie de fonctions de contrôle parmi lesquelles l'on retrouve if et cond : `(if <condition> action1 action2)`. Si la condition est vraie, alors on effectuera l'action1, sinon l'action2. Une condition est vraie si son évaluation rend une autre valeur que NIL (symbole du faux en Lisp, par opposition à T (True)).

Comme signalé plus haut, programmer en Lisp, c'est programmer de manière récursive. Pour comprendre cette notion, je vous propose l'exemple typique d'une fonction récursive, la fonction factorielle :

```
(defun factoriel (n)
   (if (<= n 1) 1
          (* n (factoriel (1- n)))))

(factoriel 5)
--> 120
```

AMXLisp

Après cette introduction au concept de Lisp, nous allons nous intéresser de manière plus approfondie à la version dont nous disposons sur la disquette ANT25. Vous verrez sur cette disquette un tiroir nommé AMXLisp, dans lequel se trouve l'ensemble du produit sous forme compactée avec l'utilitaire LHArc (fichier AMXLisp.lzh). LHArc est lui aussi présent sur la disquette, dans le répertoire c.

Pour pouvoir utiliser AMXLisp, je vous invite à suivre la procédure suivante : l'ensemble des fichiers décompactés occupant un peu plus de 700 Ko, formattez une disquette vierge que vous nommerez AMXLisp, puis tapez sous CLI les commandes suivantes :

```
cd AMXLisp:
ANT25:c/LHArc -m -a x ANT25:AMXLisp/AMXLisp.lzh
```

A la suite de quoi vous obtiendrez sur votre disquette AMXLisp, l'ensemble des directories et fichiers suivants :
- *fd/ (fichiers *.fd)*
- *include/ (blibliothèques standard de l'Amiga)*
- *interface_src/ (sources de l'interface Amiga)*
- *lsp/ (sources de configuartion, plus des exemples)*
- *src/ (sources d'AMXLisp)*
- *amxlisp (exécutable, noyau de l'interprêteur)*
- et quelques autres...

Vous pourrez remarquer qu'à l'aide des fichiers includes, vous pourrez sans problème utiliser les bibliothèques internes de l'Amiga et donc faire à peu près autant de chose qu'avec un autre langage.

Il ne vous restera plus qu'à lancer l'interprêteur en tapant sous CLI la commande : amxlisp.

RIEN NE VAUT LA PRATIQUE

A titre d'exemple, et pour illustrer le style de programmation Lisp, je vous propose maintenant de construire un programme capable de résoudre le problème dit "des 8 Reines". Il s'agit de placer huit reines deux à deux non en prise sur un échiquier, c'est-à-dire de telle sorte qu'aucune reine ne soit sur la même ligne horizontale, verticale ou diagonale que l'autre.

Ce problème bien connu peut être résolu de manière simple en utilisant une méthode d'exploration d'arbre des possibilités, ce qui se prête très bien à une programmation récursive.

Vous pouvez taper les fonctions suivantes directemment sous l'interprêteur, mais plus sûrement dans un fichier que vous chargerez ensuite à l'aide de la fonction load (load "nom de fichier") ou bien, si vous possédez la disquette ANT25, chargez le fichier source reines.lsp.

Premièrement, nous allons écrire une fonction qui déterminera si 2 reines sont en prise mutuelle. Aux échecs, une reine peut se déplacer le long de la colonne, de la ligne et des deux diagonales passant par sa position actuelle. On peut donc représenter sa position sur l'échiquier par le numéro de sa ligne et de sa colonne, d'où la fonction :

```
(defun en_prise (lig1 col1 lig2 col2)
   (or (= lig1 lig2)                          ; meme ligne
       (= col1 col2)                          ; meme colonne
       (= (- lig1 col1) (-lig2 col2))         ; diagonale 1
       (= (+ lig1 col1) (+ lig2 col2))))      ; diagonale 2
```

Ensuite, on représente la configuration de plusieurs reines sur l'échiquier en utilisant une liste d'éléments, chacun d'eux étant lui-même une liste à deux

composantes : la ligne et la colonne de la reine sur l'échiquier.

Supposons maintenant que nous voulions disposer une reine. La fonction suivante nous indique si cette position (lig, col) est valide pour la nouvelle reine à placer :

```
(defun en_conflit (lig col echiquier)
    (cond ((null echiquier) NIL)
        ((or (en_prise lig col (caar echiquier) (cadar echiquier))
             (en_conflit lig col (cdr echiquier))))))
```

Aprés ces préliminaires, nous allons pouvoir écrire la fonction qui va parcourir l'échiquier pour nous fournir l'ensemble des solutions possibles. Notons au passage que cette fonction est capable de traiter le problème plus général qui consiste à placer N reines sur un échiquier de dimension NxN.

```
(defun place_reines (n)
    (reine-ligne nil 0 n))

(defun reine-ligne (echiquier lig n)
    (cond ((= lig n) (print (reverse echiquier)))
        (t (reine-colonne echiquier lig 0 n))))

(defun reine-colonne (echiquier lig col n)
    (cond ((= col n))
        (t (cond ((en_conflit lig col echiquier))
                 (t (reine-ligne (cons (list lig col) echiquier)
                                 (+ lig 1) n))
                 )
            (reine-colonne echiquier lig (+ col 1) n))))
```

La fonction place_reine nous donne les solutions sous la forme d'une liste de positions. Il serait plus sympathique de pouvoir les visualiser graphiquement, c'est ce que je vous propose de faire avec les fonctions suivantes :

```
;;; impression d'une solution sous la forme d'un echiquier
(defun print-echiquier (echiquier)
    (print-echiquier-aux echiquier (length echiquier)))

(defun print-echiquier-aux (echiquier dimension)
    (terpri)
    (cond ((null echiquier))
          (t (print-echiquier-sub (cadar echiquier) 0 dimension)
          (print-echiquier-aux (cdr echiquier) dimension))))

(defun print-echiquier-sub (col n dimension)
;;; col = position de R sur la ligne
    (cond ((= n dimension))
        (t (cond ((= col n) (princ 'R))
            (t (princ ".")))
            (princ " ")
            (print-echiquier-sub col (+ n 1) dimension))))
```

Il suffit de modifier la fonction place_reines en y insérant une variable globale qui permet de stocker l'ensemble des solutions :

```
(defun place_reines (n)
    (setq *solutions* ())      ; pour stocker toutes les solutions
    (reine-ligne NIL 0 taille)
    (mapc 'print-echiquier *solutions*) ())   ; impression
```

puis de rajouter dans reine-ligne, après le (print (reverse echiquier)) l'appel à (setq *solutions* (append *solutions* (list (reverse echiquier)))).

Tout celà donnera, pour un échiquier de dimension, 4 :

```
> (place_reines 4)
((0 1) (1 3) (2 0) (3 2))
((0 2) (1 0) (2 3) (3 1))

. R . .
. . . R
R . . .
. . R .

. . R .
R . . .
. . . R
. R . .
NIL
```

Pour ceux qui désireraient tester le problème initial des 8 reines, je vous signale qu'il comporte 92 solutions.

Une dernière remarque sur ce type de problème : il est bien évident que certaines techniques permettraient de ne pas parcourir l'intégralité de l'espace de recherche pour aboutir à une solution. Je citerai à ce propos une technique issue de l'intelligence artificielle : la programmation par contraintes. En effet et comme son nom l'indique, elle permet, en propageant des contraintes liées à un choix (placement d'une reine par exemple), d'enlever de l'espace de recherche un certain nombre de possibilités qu'il ne sera pas utile d'explorer par la suite. J'aurai, j'espère, l'occasion de vous en parler lors d'un prochain article.

AVANT DE VOUS QUITTER

En conclusion, cet interprêteur dont j'ai testé la version BetaTest 2.0, vous est bien sûr proposé sur la disquette ANT accompagnant ce numéro, mais également sur la disquette Fish 181, que vous pourrez vous procurer dans toute bonne association diffusant du domaine public. Globalement, AMXLisp est un bon produit. On pourrait néanmoins regretter que la présente version ne soit qu'une BetaTest et qu'il ne soit pas sorti (à ma connaissance) de version plus récente. Cependant, malgré sa relative ancienneté, ce logiciel a le mérite de fonctionner sur A3000 et sous kickstart 2.0, ce qui n'est pas si courant (NDLR : malheureusement !).

Un dernier point enfin : je n'ai pas abordé dans cet article la possibilité d'employer la couche "objet" d'AMXLisp, jugeant que ce domaine méritait pour le moins que l'on y consacre un certain temps. Ce qui sera fait dans un proche avenir, je l'espère.

```
; Basic Demo

(load "allocate")
(load-c-struct "intuition/intuition" '(newwindow window intuitext))
(load-c-struct "graphics/rastport" '(rastport))
(load-c-struct "exec/ports" '(msgport))
(defamiga 'Wait 'exec)
(defamiga 'OpenWindow 'intuition)
(defamiga 'CloseWindow 'intuition)
(defamiga 'SetWindowTitles 'intuition)
(defamiga 'PrintIText 'intuition)
(defvar wtitle "AMXLisp Demo")
(defvar itxt "Hello Word")

; beuark !
(defmacro str-address (str)
    '(memory-long (+ (address-of ,str) 6)))

; !!??????!!!!
(defmacro iexp (x y)
    '(truncate (expt (float ,x) (float ,y))))
; 2^31 ???
(defun power2 (x)
    (cond ((equal x 31) 2147483648)
        (t (iexp 2 x))))

(defun demo ()
    (let ((nw (newamiga newwindow))
          (txt (newamiga intuitext)))
        (send nw :-> 'LeftEdge 10)
        (send nw :-> 'TopEdge 10)
        (send nw :-> 'Width 400)
        (send nw :-> 'Height 100)
        (send nw :-> 'DetailPen 0)
        (send nw :-> 'BlockPen 1)
        (send nw :-> 'IDCMPFlags #x200)   ; CLOSEWINDOW
        (send nw :-> 'Flags #x100f)   ; ACTIVATE | all system gadgets
        (send nw :-> 'MinWidth 40)
        (send nw :-> 'MinHeight 40)
        (send nw :-> 'Type 1)
        (send txt :-> 'FrontPen 2)
        (send txt :-> 'BackPen 3)
        (send txt :-> 'LeftEdge 20)
        (send txt :-> 'TopEdge  20)
        (send txt :-> 'FrontPen 2)
        (send txt :-> 'IText (str-address itxt))

        (let ((myw (send window :new (callamiga 'OpenWindow intuition nw))))
            (callamiga 'SetWindowTitles intuition myw wtitle 0)
            (callamiga 'PrintIText intuition (send myw :-> 'RPort) txt 0 0)
            (dotimes (i 1000) ())
            (callamiga 'Wait exec (power2
                                    (send (send myw :-> 'UserPort)
                                          :-> 'mp_SigBit)))
            (callamiga 'CloseWindow intuition myw)
            (freeamiga nw) (freeamiga txt))))
```

par Patrick CHAZE

DEMOS : LA 3D FACES PLEINE

Le listing du mois dernier était relativement efficace dans son domaine. Je l'avais agrementé de certaines ruses pour accélérer le temps d'exécution, ce qui est primordial pour une démo digne de ce nom.

Il nécessite donc un nombre important de points à expliquer. Dans le désordre, nous avons : le clipping de polygones (derivé du clipping de droites déjà expliqué), la technique des faces cachées en elle-même (encore des formules mathématiques, mais comment faire autrement en 3D ?), la méthode utilisée pour remplir les polygones visibles et enfin la structure de l'objet en mémoire, afin que vous puissiez faire bouger autre chose qu'un simple cube.

LA TECHNIQUE DES FACES CACHEES

Celle que j'utilise, comme disait Coluche, a des avantages et des inconvénients. Elle permet de déterminer si une face est orientée ou non vers l'observateur. Pour ce faire, on oriente chacune des faces dans un sens constant puis l'on teste si elles sont dirigées vers l'observateur. D'après les résultats, on affiche uniquement celles qui sont bien orientées. Ceci peut s'obtenir grâce à un calcul relativement simple, en utilisant la composante z du produit vectoriel (et c'est reparti pour un cours de maths).
Comme chacun sait (NLDR : ah bon ?), le produit vectoriel est une opération entre deux vecteurs ayant pour résultat un vecteur normal au plan formé par ces deux vecteurs. Ainsi, si l'on effectue cette opération après avoir transformé l'objet spatial dans le plan formé par l'écran, le vecteur résultat sera normal à l'ecran si les points étaient bien orientés, ou dirigé vers le fond s'ils ne l'étaient pas. Le signe de la partie z nous permet de connaître l'orientation du vecteur normal. En effet, si la partie z est positive, la face est visible. Dans le cas contraire, la face est invisible, on ne l'affiche pas et l'on passe à la suivante.

Dans notre cas, les deux vecteurs sont deux arêtes consécutives du polygone représentant la face. Pour accélerer le temps d'exécution et simplifier la création des objets, les vecteurs sont codés sous la forme des trois sommets des deux vecteurs (deux extremités pour chacun, mais il y en a une commune, du fait qu'ils sont consécutifs). On peut noter que ces trois points sont tout bonnement trois points consécutifs du polygone.

Une fois les trois paires de coordonnées (x1,y1), (x2,y2) et (x3,y3) obtenues, on calcule la partie z du produit vetoriel, avec la formule

```
z=(y2-y1)*(x3-x1)-(y3-y1)*(x2-x1)
```

Suivant le signe de z, le vecteur normal à la face sera dirigé soit vers le fond, soit vers l'observateur, ce qui nous permet de choisir si l'on affiche ou pas la face en question.

Les avantages d'une telle technique sont rapidité d'exécution et simplicité de programmation. Mais elle possède également un inconvénient majeur : l'objet doit absolument être convexe (rappel : un objet est dit convexe lorsque toutes ses diagonales sont situées à l'interieur de l'objet). Ceci est une restriction importante, car elle permet de gérer uniquement des objets tel que le cube, le triangle, la sphere, etc. et en aucun cas des objets de la forme d'un hélicoptère, ou autres formes complexes. Dans le cas d'une routine gérant aussi les formes concaves, il faudrait calculer quelle partie de chaque face est visible, puis tracer les parties apparentes. Cette technique est efficace pour l'affichage mais longue au niveau du calcul ; elle est donc utilisable par exemple sur les gros PC connus pour leur rapidité de calcul et la lenteur de leur mémoire graphique, mais difficilement sur un Amiga 68000. On pourrait également utiliser la méthode dite du z buffer, qui consiste à calculer la distance qui sépare chaque face de l'observateur puis à les trier dans l'ordre décroissant afin d'afficher une par une chaque face en commençant par la plus eloignée et finissant par la plus proche. Elle a l'avantage de ne posséder aucune restriction de forme, mais elle est bien plus lente à l'affichage, chaque face de l'objet étant dessinée, qu'elle soit visible ou pas.

L'AFFICHAGE DES POLYGONES

Je passerai sur la description du fonctionnement du mode remplissage du Blitter ; vous pouvez la trouver dans les livres tel que le Hardware Manual.

Grâce à ma technique de calcul très contraignante, il n'y a que deux cas à traiter : soit la face est entièrement visible, soit elle est totalement cachée et nous n'avons pas à l'afficher. Pour faire apparaître l'objet, nous allons donc dans un premier temps tracer toutes les arêtes de chaque polygone dans les bons bitplans. Prenons par exemple une face de couleur 3 : il faut tracer les arêtes sur les bitplans 1 et 2 mais pas sur le 3. Ainsi, après le remplissage, la face ne sera dessinée que sur les bitplans 1 et 2 et sera bien de couleur 3.

Une autre problème se pose. Imaginons en effet le coin d'un polygone qui serait dirigé vers le bas à l'intersection des deux arêtes : il n'y aura plus qu'un seul point sur la ligne horizontale. Le Blitter va donc commencer à remplir, mais ne s'arrêtera pas sur le second côté, car en fait, il était confondu avec le premier, d'où bug hideux et crapuleux. Pour éviter ce problème, il faut dessiner les droites en mode EOR (minterms $4A) et du haut vers le bas. Je ne vous expliquerai pas pourquoi, ayant moi-même trouvé ceci en tatonant. L'utilisaton du mode EOR implique que si l'on dessine deux fois la même droite, nous revenons à l'état initial. Afin d'éviter une opération inutile, j'exécute une routine qui élimine les droites tracées deux fois ; dans le source du mois dernier, elle est mêlée au calcul du produit vectoriel. Son principe est que, sachant que toute les arêtes d'une même face ont la même couleur, à chaque nouvelle face, on met sa couleur dans le registre couleur de chaque ligne (le troisième mot dans le tableau tab_line) composant cette face par un EOR. Ainsi une ligne qui aurait du être dessinée deux fois ne le sera pas du tout.

Après avoir convenablement tracé les lignes, il ne reste plus qu'à remplir les faces pour faire apparaître l'objet. Ceci peut s'effectuer de différentes manières : la première, que je

n'utilise pas, consiste à encadrer l'objet dans un rectangle de la taille minimum et à le remplir avec le Blitter. Elle paraît la plus efficace car la fenêtre est la plus petite possible, mais il n'en est rien. En effet, son temps d'exécution varie énormément en fonction de la taille de l'objet, ce qui peut être très gênant dans une démo. De plus, on passe une grande partie du temps-machine à attendre le Blitter, ce qui est aussi très mauvais.

J'utilise donc une autre méthode, dont le principe est de remplir tout l'écran en permanence. Cela permet d'avoir un temps d'exécution constant et donc d'utiliser de facon optimum le jumelage Blitter/68000. Je vous avais déjà expliqué lors du premier article sur la 3D, que pour effacer plus vite l'écran, je me servais simultanément du 68000 et du Blitter. Ici, on utilise à nouveau cette technique, en l'améliorant toutefois un peu. En effet, non seulement le 68000 efface l'écran en même temps que le Blitter, mais aussi pendant que le celui-ci le remplit ! Ceux qui suivent encore doivent se demander comment je fais pour effacer l'écran tout en le remplissant ? Eh bien c'est une bonne question, et je me remercie de me l'avoir posée. En fait, ce cas ne se produit jamais car j'utilise trois pages pour le flipping : une qui apparaît à l'écran, une que je remplis au Blitter et une que j'efface au 68000 et au Blitter.

LE CLIPPING DE POLYGONES

Resumé des épisodes précédents : la technique utilisée est la dichotomie. Le clipping de droite est le moyen de connaître les coordonnées de la partie visible de la droite. Le clipping de polygone s'inspire fortement de ce dernier, car en fait, il équivaut à un clipping de droite sur chacune des arêtes, plus un léger additif pour le cas où le polygone dépasserait sur la droite de l'écran.

Pour être plus clair, prenons pour exemple un losange régulier dont les quatre bords sortent de chaque côté de l'écran. Les clippings haut et bas se font avec une déconcertante facilité. Il suffit en effet de bien déterminer le registre BLTSIZE de manière à ce que le Blitter commence son action en bas de la partie visible et la termine en haut. Le clipping gauche est lui aussi relativement aisé, car il ne nécessite absolument rien à part le simple clipping de droite qui est commun à chaque cas de clipping. Nous en arrivons au clipping droit qui engendre le supplément de travail evoqué plus haut. Le problème est que si l'extrémité droite du losange (polygone exemple) dépasse du côté droit de l'écran, les points les plus à droite seront théoriquement allumés. De plus, on sait que le Blitter remplit de la droite vers la gauche. Donc, si nous effectuons un unique clipping de droite, les points les plus à droite de l'écran seront vides et le Blitter sera persuadé qu'il ne doit pas remplir cette surface, d'où, encore une fois, un horrible et méchant bug, la surface théoriquement pleine apparaissant en partie vide.

Pour palier à ce problème, il faut dessiner une droite verticale le long du bord droit de l'écran, allant du y de l'extremité droite de la droite clippée jusqu'au y prévu avant le clipping. Affinons l'exemple avec une droite de coordonnées (300,0)-(340,40). Pour clipper convenablement cette arête, il faudra afficher non seulement la droite simplement clippée (300,0)-(320,20) mais en plus une droite verticale (320,20)-(320,40). Grâce à cette droite supplémentaire, le Blitter rencontrera des points allumés à l'extremité droite de l'écran et remplira donc la surface comme prevu.

LA STRUCTURE DES OBJETS

La structure est composée de quatre zones de données :
- coor_e regroupe les coordonnées spatiales des points les unes à la suite des autres, la première donneé décrivant le nombre total de points composant l'objet ;
- coor_p sert uniquement à stocker les coordonnées planes des points après rotation et projection ;
- tab_line definit les lignes par groupes de trois valeurs. Deux sont en fait les numéros des points extrémités, pré-multipliés par 4 pour ne pas avoir à le faire au moment de l'accès à la table des coordonnées planes. La troisième valeur est initialisée à 0 et sert, lors du dessin de la ligne, à savoir dans quel(s) bitplan(s) la tracer, afin de ne pas afficher deux fois la même ligne inutilement. La valeur nb_line doit contenir le nombre de lignes dont l'objet est composé ;
- tab_face détermine les faces, leur orientation et leur couleur. Le premier mot est le nombre de faces de l'objet. Vient ensuite la description de chaque face une par une, ainsi décomposée : trois mots (là encore pré-multipliés par 4) contenant les numéros des trois points d'orientation (qui doivent se suivre dans le sens inverse des aiguilles d'une montre lorsque l'on se trouve devant la face), un mot contenant le nombre d'arêtes qui composent cette face, un mot contenant la couleur puis tous les numéros (pré-multipliés par 6) des lignes du polygone. Pourquoi pré-multipliés par 6 ? Simplement parce que chaque ligne est codée sur trois mots, soit six octets !

CONCLUSION

Je m'insurge contre la vague de conformisme qui s'empare des demo-makers depuis trop longtemps. En effet, il y en a un, en Scandinavie, qui à trouvé un effet original, et le reste de l'Europe l'use jusqu'à la corde en le reproduisant sans vergogne. Et ce n'est encore pas le pire... Il y a aussi la vague de démos purement graphiques qui prend une forte ampleur. Revenons aux sources mes frères ! N'oublions jamais qu'en des temps reculés, le terme "démo" signifiait réellement "démonstration". Le but était de montrer les capacités de la machine...

Prions le Dieu 68000 ! Honorons-le avec des offrandes ne pouvant pas être programmées en Basic ni même en C ! Pour votre pénitence, vous me ferez chacun une démo orginale, demandant un bon niveau technique !

Ainsi finit le sermon du nostalgique. Je reprends le calme qui a fait la réputation du grand prêcheur charismatique que je suis, pour vous annoncer que la prochaine fois, nous verrons comment dépasser allégrement les limites du hardware en mettant plus de huit sprites sur une meme ligne de raster.

Si vous désirez voir traiter ici des effets particuliers, écrivez-moi. Autre chose, je cherche un bon graphiste pour réaliser des jeux sur Amiga et ST. Pas sérieux s'abstenir.

par Jérôme ETIENNE

UTILITAIRE : L'ESPION DE LA

Quand j'étais à l'école, la maîtresse disait toujours au petit surdoué que j'étais déjà, que le meilleur moyen d'apprendre quelque chose est d'observer d'autres personnes le faire. Ce vieil adage m'est depuis resté, et je ne manque jamais une occasion de le mettre en pratique.

Pour ce qui nous concerne ici, nous allons nous fabriquer un outil capable de naviguer dans la mémoire de l'Amiga à la recherche de ses structures si particulières, que nous afficherons en clair - parfois en foncé - de telle sorte que l'utilisateur n'aie plus qu'à cliquer sur l'un des champs de cette structure pour en visualiser le contenu. Ainsi, il (l'utilisateur) sera en mesure de se balader de fenêtre en fenêtre, de menu en menu, de gadget en gadget, de train en train et de port en port et pourra y observer tous les paramètres mis en place par d'autres programmeurs plus chevronnés que lui. En Anglais, on appelle ce type d'outils un "browser", ce qui, si ma mémoire ne me fait pas trop défaut, doit signifier quelque chose comme "baladeur", "scruteur". En aucun cas il ne sera permis de modifier les valeurs trouvées, sinon cela devient un éditeur, ce qui est hors de propos aujourd'hui. Sans parler des conséquences désastreuses que peut avoir la moindre modification de données inconnues.

COMMENT CA MARCHE ?

Le principe est très simple : pour chaque structure connue du programme, on écrit une fonction chargée de l'afficher correctement. Histoire de ne pas trop se surcharger de travail, on lui associe également un bloc d'informations décrivant le nom de chacun des membres de la structure, ainsi que son type. De cette manière, l'affichage à proprement parler de toutes les structures pourra être effectué par une seule et même fonction, ne laissant plus que les réactions de l'utilisateur à gérer. Rien de bien extraordinnaire d'ailleurs, puisque celà consiste à repérer quel champ à été cliqué, et à appeler la routine d'affichage correspondante. Notez à ce sujet qu'une même fonction peut s'appeler elle-même. Par exemple, la fonction prWindow() (qui affiche donc une structure Window) s'appelle elle-même lorsque l'un des champs NextWindow, Parent ou Descendant à été cliqué. En C, celà ne pose aucun problème, il faut juste faire attention à ce que la pile ne déborde point, suite à de trop nombreux appels récursifs.

La version publiée de notre espion ne connaît que les structures IntuitionBase, Screen, Window, Gadget et Menu - la place manquait pour en ajouter d'autres. La version présente sur la disquette accompagnant ce numéro connaît en plus les structures MenuItem, Gadget, IntuiText, Border, Image, RastPort, ViewPort, TextAttr et TextFont.

Si vous désirez ajouter des structures supplémentaires, il vous suffit d'écrire autant de nouvelles fonctions, en vous basant sur le principe de celles déjà en place, qu'il vous faudra tout de même modifier un tout petit peu, afin qu'elles se rendent compte du changement.

Pour clarifier le principe, imaginons que nous voulions ajouter la structure BoolInfo. On commence donc par écrire une fonction prBoolInfo() qui ressemblera à ça :

```
void prBoolInfo(struct BoolInfo *bi)
{
static struct StructureInfo si[] =
{   {   "USHORT ", "Flags",    MOT16, WORDSISE },
    {   "UWORD *", "Mask",     ALIEN, PTRSIZE  },
    {   "ULONG ",  "Reserved", MOT32, LONGSIZE }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)bi);
        choix = GetChoix();
    }
}
```

Le tableau StructureInfo décrit tous les champs de la structure BoolInfo :
- d'abord son type en ASCII ("ULONG", "UWORD *", etc.)
- puis son nom en ASCII aussi ("Flags", "Mask", etc.)
- son type codé, à choisir dans le tableau ci-dessous
- enfin, sa longueur en octets

Les types de champs possibles sont définis dans Spy.h. Ce sont :

```
ALIEN    le champ est d'un type inconnu du programme
STRUC    structure à l'intérieur de la structure
OCTET    valeur sur 1 octet
CARAC    idem, mais affichée en ASCII
MOT16    valeur sur 16 bits
MOT32    valeur sur 32 bits
CHAIN    pointeur sur une chaine de caractères
S_PTR    pointeur
TABLO    tableau de valeurs
```

Une fois ceci fait, il vous faudra modifier la fonction prGadget() afin qu'elle prenne en compte votre nouvelle fonction prBoolInfo(). Il faudra pour celà changer le code du champ SpecialInfo d'ALIEN en S_PTR, et ajouter dans le switch :

```
case GGID_CHOIX + 12:
    if (gg->Flags & BOOLEXTEND)
        prBoolInfo((struct BoolInfo *)gg->SpecialInfo);
    break;
```

Et c'est tout.

Tel quel, ce programme ne présente pas vraiment d'astuces de programmation éclatantes. J'espère toutefois qu'il vous sera utile dans la mise au point de vos propres applications - c'est en tout cas son but.

LISTING 1

```
#makefile pour Spy.c -  1991, Max
EXE=Spy
OBJ=Spy.o SpyGadgets.o
LIB=LIB:amiga.lib LIB:lc.lib
ARG=-d5 -mas -v -cfsuwt -N
LNK=SC SD DEFINE __main=__tinymain NOICONS NODEBUG

Spy:            $(OBJ)
                BLink LIB:c.o $(OBJ) TO $(EXE) LIB $(LIB) $(LNK)

Spy.o:          Spy.c Spy.h
                Lc $(ARG) Spy.c

SpyGadgets.o:   SpyGadgets.c Spy.h
                Lc $(ARG) SpyGadgets.c
```

LISTING2

```
/*
* Spy.h - Fichier d'entête commun à Spy.c et SpyGadgets.c
* /

#define SZ(x) sizeof(struct x)

#define BYTESIZE    sizeof(BYTE)
#define WORDSIZE    sizeof(WORD)
#define LONGSIZE    sizeof(LONG)
#define PTRSIZE     sizeof(APTR)

#define ALIEN   0x0000  /* type inconnnu - non sélectionnable */
#define STRUC   0x0001  /* structure                          */
#define OCTET   0x0002  /* octet                              */
#define CARAC   0x0003  /* caractère alphanumérique           */
#define MOT16   0x0004  /* mot                                */
#define MOT32   0x0005  /* long                               */
#define CHAIN   0x0006  /* chaine de caractères               */
#define S_PTR   0x0007  /* pointeur sur une structure         */
#define TABLO   0x0008  /* tableau de valeurs quelconques     */

#define MAX_BUFFERS     64
#define MAX_LINES  15

#define GGID_CHOIX 0
#define GGID_RETOUR     0x1000
#define GGID_SLIDER     0x1001

#define TOP_LINE    25

#define SI_SIZE     (sizeof(si) / sizeof(struct StructureInfo))
```

```
struct StructureInfo
{   UBYTE *MemberType;
    UBYTE *MemberName;
    WORD   MemberFlag;
    WORD   MemberSize;
};

struct Depart
{   struct IntuitionBase *IntuitionBase;
    struct GfxBase       *GfxBase;
    /* On pourrait continuer avec d'autres libraries */
};
```

LISTING 3

```
/*************************************************************************
* Spy v1.0 - L'Espion des structures en mémoire...                       *
* Par Max pour ANT                                                       *
*                                                                        *
* Ecrit pour le SAS/Lattice C 5.10                                       *
* (voir le makefile ci-joint pour les détails de compilation)            *
*                                                                        *
* NOTE : Pour des raisons de mise en page, la version publiée            *
*        est plus petite que celle sur la disquette...                   *
*************************************************************************/

#include <exec/types.h>
#include <exec/memory.h>
#include <exec/ports.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <intuition/intuitionbase.h>
#include <libraries/dos.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <clib/macros.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>

#include "spy.h"

/***** Variables globales ****************************************/
struct IntuitionBase *IntuitionBase;
struct GfxBase *GfxBase;
struct Window *win;
struct RastPort *rp;

SHORT Overlap, visibleLines,
totalLines, topLine, hidden;

/***** Données globales et externes ******************************/
extern struct NewWindow nw;
extern struct Gadget ChoixGadget[];
extern struct Gadget RetourGadget, SliderGadget;
extern struct PropInfo SliderInfo;

struct IntuiText iText =
{   1, 0, JAM2, 15, (TOP_LINE + 1), NULL, NULL, NULL
};

UBYTE buffers[MAX_BUFFERS][80], buftype[MAX_BUFFERS];

struct Depart Debut;

/***** Prototypes ************************************************/
void main(void);
void cleanexit(LONG code);

void ClearWindow(void);
void MakeBuffers(struct StructureInfo *sinfo,WORD num,UBYTE *mem);
void PrepareGadgets(void);
void SetSlider(void);
void GetSlider(void);
void SetGadgets(void);
SHORT GetChoice(void);

void HexDump(UBYTE *base, WORD nb, WORD size);
void Dump08(UBYTE *base, UBYTE *buf, SHORT nb);
void Dump16(UBYTE *base, UBYTE *buf, SHORT nb);
void Dump32(UBYTE *base, UBYTE *buf, SHORT nb);

void prDepart(struct Depart *deb);
void prIntuitionBase(struct IntuitionBase *ib);
void prScreen(struct Screen *scr);
void prWindow(struct Window *win);
void prGadget(struct Gadget *gg);
void prMenu(struct Menu *mnu);

/***** main() - Initialisation et affiche la structure de départ **/
void main(void)
{
    if (!(IntuitionBase = (struct IntuitionBase *)
        OpenLibrary("intuition.library", 33L)))
    cleanexit(RETURN_FAIL);

    if (!(GfxBase = (struct GfxBase *)
        OpenLibrary("graphics.library", 33L)))
    cleanexit(RETURN_FAIL);

    Debut.IntuitionBase = IntuitionBase;
    Debut.GfxBase       = GfxBase;

    if (!(win = OpenWindow(&nw)))
        cleanexit(RETURN_FAIL);
    rp = win->RPort;

    PrepareGadgets();

    for (;;)    /* FOREVER */
    {   prDepart(&Debut);
    }
}

/***** cleanexit() - Ferme tout et retourne au WB/CLI ************/
void cleanexit(LONG code)
{
    if (win)
        CloseWindow(win);
    if (GfxBase)
        CloseLibrary((struct Library *)GfxBase);
    if (IntuitionBase)
        CloseLibrary((struct Library *)IntuitionBase);
    exit(code);
}

/***** ClearWindow() - Efface la fenêtre et ses gadgets **********/
void ClearWindow(void)
{
    if (SliderGadget.NextGadget)
        RemoveGList(win, SliderGadget.NextGadget, -1L);

    SetAPen(rp, 0);
    RectFill(rp, win->BorderLeft, win->BorderTop,
             win->Width  - win->BorderRight  - 2,
             win->Height - win->BorderBottom - 2);
}

/***** MakeBuffers() - Prépare les textes des gadgets ************/
void MakeBuffers(struct StructureInfo *sinfo, WORD num, UBYTE *mem)
{
LONG offset;
register SHORT i, coul;
UBYTE b[40];

    ClearWindow();

    Overlap = 1;
    visibleLines = MAX_LINES;
    totalLines = topLine = hidden = 0;

    if (num > MAX_BUFFERS) num = MAX_BUFFERS;

    for (i = offset = 0; i < num; i++)
    {   register struct StructureInfo *si = &sinfo[i];
        register WORD type = si->MemberFlag;
        register UBYTE *buf = buffers[i];
        register APTR ptr;

        sprintf(buf,"%-.24s%-.16s",si->MemberType,si->MemberName);

        coul = 2;

        switch(type)
        {   case STRUC:
            case TABLO:
                coul = 1;
            case ALIEN:
                b[0] = ';';
                b[1] = '0';
                break;
            case OCTET: /* écrit un octet en hexa */
                sprintf(b, " = 0x%02lx;", *(UBYTE *)(mem + offset));
                break;
            case CARAC: /* écrit un octet en ASCII */
                sprintf(b, " = '%lc';", *(UBYTE *)(mem + offset));
                break;
            case MOT16: /* écrit un mot en hexa */
                sprintf(b, " = 0x%04lx;", *(UWORD *)(mem + offset));
                break;
            case MOT32: /* écrit un long en hexa */
                sprintf(b, " = 0x%08lx;", *(ULONG *)(mem + offset));
                break;
            case CHAIN: /* écrit une chaine en ASCII */
                if (!(ptr = (APTR)*(ULONG *)(mem + offset)))
                    strcpy(b, " = NULL;");
                else
```

```
            {   coul = 3;
                sprintf(b, " = '%.38s';", ptr);
            }
            break;
        case S_PTR: /* écrit un pointeur */
            if (!(ptr = (APTR)*(ULONG *)(mem + offset)))
                strcpy(b, " = NULL;");
            else
            {   coul = 1;
                sprintf(b, " = 0x%08lx;", ptr);
            }
            break;

    }   /* switch */

    strcat(buf, b);
    buftype[i] = coul;

    if (i < MAX_LINES)
    {   iText.FrontPen = coul;
        iText.IText    = buf;
        PrintIText(rp, &iText, 15, i * 10);
    }
    ++totalLines;
    offset += (LONG)si->MemberSize;
}   /* for */
SetGadgets();
SetSlider();
}

/***** SetGadgets() - Met en place les gadgets de choix **********/
void SetGadgets(void)
{
register SHORT i;

    for (i = 0; (i < MAX_LINES) && (i < totalLines); i++)
    {   if (buftype[topLine + i] == 1)
        {   ChoixGadget[i].GadgetID = topLine + i;
            AddGadget(win, &ChoixGadget[i], -1L);
        }
    }
}

/****** SetSlider() - Règle la taille et la position du slider ***/
void SetSlider(void)
{
register UWORD VertBody, VertPot;

    hidden = MAX(totalLines - visibleLines, 0);

    if (topLine < 0)
        topLine = 0;
    else if (topLine > hidden)
        topLine = hidden;

    if (hidden > 0 && totalLines > Overlap)
        VertBody = (UWORD)(((ULONG)(visibleLines - Overlap)
                    * MAXBODY) / (totalLines - Overlap));
    else
        VertBody = MAXBODY;

    if (hidden > 0)
        VertPot  = (UWORD)(((ULONG)topLine * MAXPOT) / hidden);
    else
        VertPot  = 0;

    NewModifyProp(&SliderGadget, win, NULL, FREEVERT|AUTOKNOB,
                  MAXPOT, VertPot, MAXBODY, VertBody, 1L);
}

/****** GetSlider() - Lit la nouvelle position du slider *********/
void GetSlider(void)
{
register SHORT i;

    topLine = ((((ULONG)hidden * SliderInfo.VertPot) +
              (MAXPOT / 2)) / MAXPOT);

    ClearWindow();
    for (i = 0; (i < MAX_LINES) && (i < totalLines); i++)
    {   iText.FrontPen = buftype[topLine + i];
        iText.IText    = buffers[topLine + i];
        PrintIText(rp, &iText, 15, i * 10);
    }
    SetGadgets();
}

/***** GetChoix() - Gère les gadgets (choix, retour et slider) ***/
SHORT GetChoix(void)
{
struct IntuiMessage *im;
struct Gadget *g;
ULONG class;
USHORT code;

for (;;)    /* FOREVER */
{   WaitPort(win->UserPort);
    while (im = (struct IntuiMessage *)GetMsg(win->UserPort))
    {   class = im->Class;
        code  = im->Code;
        g     = (struct Gadget *)im->IAddress;
        ReplyMsg((struct Message *)im);

        switch(class)
        {   case RAWKEY:
                if (code == CURSORUP)
                    --topLine;
                else if (code == CURSORDOWN)
                    ++topLine;

                SetSlider();
                GetSlider();
                break;

            case GADGETUP:
                if (g->GadgetID == GGID_SLIDER)
                    GetSlider();
                else

                    return((SHORT)g->GadgetID);
                break;

            case CLOSEWINDOW:
                cleanexit(RETURN_OK);
                break;
        }
    }
}
    return(0);  /* Evite un Warning à la compilation */
}

/***** HexDump() - Affiche un dump hexa d'une zone mémoire *******/
void HexDump(UBYTE *base, SHORT nb, SHORT size)
{
UBYTE *mem = base;
SHORT offset;

    ClearWindow();
    totalLines = topLine = hidden = 0;

    while (nb)
    {   if (totalLines < MAX_BUFFERS)
        {   register UBYTE *buf;

            buf = buffers[totalLines];
            *buf = '0';

            buftype[totalLines] = 3;

            switch(size)
            {   case OCTET:
                    offset = MIN(nb, 16);
                    Dump08(mem, buf, offset);
                    mem += (LONG)offset;
                    break;
                case MOT16:
                    offset = MIN(nb, 8);
                    Dump16(mem, buf, offset);
                    mem += (LONG)(offset * 2);
                    break;
                case MOT32:
                    offset = MIN(nb, 4);
                    Dump32(mem, buf, offset);
                    mem += (LONG)(offset * 4);
                    break;
            }
            if (totalLines < MAX_LINES)
            {   iText.FrontPen = 3;
                iText.IText    = buf;
                PrintIText(rp, &iText, 15, totalLines * 10);
            }
            ++totalLines;
            nb -= offset;
        }
        else
            nb = 0; /* pour sortir du while(nb) */
    }
    SetGadgets();
    SetSlider();
    GetChoix(); /* Juste pour attendre 'CLOSE' ou 'Retour' */
}

void Dump08(UBYTE *base, UBYTE *buf, SHORT nb)    /* Dump octets */
{
register SHORT i;
UBYTE b[4];
register UBYTE *m = base;

    for (i = 0; i < nb; i++)
    {   sprintf(b, "%02lx ", *(m++));
```

```
        strcat(buf, b);
    }
}

void Dump16(UBYTE *base, UBYTE *buf, SHORT nb)    /* Dump mots */
{
register SHORT i;
UBYTE b[6];
register UWORD *m = (UWORD *)base;

    for (i = 0; i < nb; i++)
    {   sprintf(b, "%04lx ", *(m++));
        strcat(buf, b);
    }
}

void Dump32(UBYTE *base, UBYTE *buf, SHORT nb)    /* Dump longs */
{
register SHORT i;
UBYTE b[10];
register ULONG *m = (ULONG *)base;

    for (i = 0; i < nb; i++)
    {   sprintf(b, "%08lx ", *(m++));
        strcat(buf, b);
    }
}

/*************************************************************************
* Voici maintenant toutes les routines d'affichages des                  *
* différentes structures du système. Elles sont toutes basées            *
* sur le même squelette... Avec un bon éditeur et quelques coups *
* de copier/coller bien placés, ce ne sera pas trop dur à taper. *
*************************************************************************/

/***** Structure de départ ***************************************/
void prDepart(struct Depart *deb)
{
static struct StructureInfo si[] =
{   {   "struct IntuitionBase *", "IntuitionBase", S_PTR, PTRSIZE },
    {   "struct GfxBase *",       "GfxBase",       ALIEN, PTRSIZE }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)&Debut);
        switch(choix = GetChoix())
        {   case GGID_CHOIX + 0:
                prIntuitionBase(Debut.IntuitionBase);
                break;
        }
    }
}
/***** IntuitionBase (champs publics seulement) ******************/
void prIntuitionBase(struct IntuitionBase *ib)
{
static struct StructureInfo si[] =
{   {   "struct Library ",  "LibNode",      ALIEN, SZ(Library) },
    {   "struct View ",     "ViewLord",     ALIEN, SZ(View),   },
    {   "struct Window *",  "ActiveWindow", S_PTR, PTRSIZE,    },
    {   "struct Screen *",  "ActiveScreen", S_PTR, PTRSIZE,    },
    {   "struct Screen *",  "FirstScreen",  S_PTR, PTRSIZE,    },
    {   "ULONG ",           "Flags",        MOT32, LONGSIZE,   },
    {   "WORD ",            "MouseY",       MOT16, WORDSIZE,   },
    {   "WORD ",            "MouseX",       MOT16, WORDSIZE,   },
    {   "ULONG ",           "Seconds",      MOT32, LONGSIZE,   },
    {   "ULONG ",           "Micros",       MOT32, LONGSIZE,   }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)IntuitionBase);
        switch (choix = GetChoix())
        {   case GGID_CHOIX + 2:
                prWindow(IntuitionBase->ActiveWindow);
                break;
            case GGID_CHOIX + 3:
                prScreen(IntuitionBase->ActiveScreen);
                break;
            case GGID_CHOIX + 4:
                prScreen(IntuitionBase->FirstScreen);
                break;
        }
    }
}
/***** Screen ****************************************************/
void prScreen(struct Screen *scr)
{
static struct StructureInfo si[] =
{   {   "struct Screen *",    "NextScreen",   S_PTR, PTRSIZE     },
    {   "struct Window *",    "FirstWindow",  S_PTR, PTRSIZE     },
    {   "SHORT ",             "LeftEdge",     MOT16, WORDSIZE    },
    {   "SHORT ",             "TopEdge",      MOT16, WORDSIZE    },
    {   "SHORT ",             "Width",        MOT16, WORDSIZE    },
    {   "SHORT ",             "Height",       MOT16, WORDSIZE    },
    {   "SHORT",              "MouseY",       MOT16, WORDSIZE    },
    {   "SHORT",              "MouseX",       MOT16, WORDSIZE    },
    {   "USHORT ",            "Flags",        MOT16, WORDSIZE    },
    {   "UBYTE *",            "Title",        CHAIN, PTRSIZE     },
    {   "UBYTE *",            "DefaultTitle", CHAIN, PTRSIZE     },
    {   "BYTE ",              "BarHeight",    OCTET, BYTESIZE    },
    {   "BYTE ",              "BarVBorder",   OCTET, BYTESIZE    },
    {   "BYTE ",              "BarHBorder",   OCTET, BYTESIZE    },
    {   "BYTE ",              "MenuVBorder",  OCTET, BYTESIZE    },
    {   "BYTE ",              "MenuHBorder",  OCTET, BYTESIZE    },
    {   "BYTE ",              "WBorTop",      OCTET, BYTESIZE    },
    {   "BYTE ",              "WBorLeft",     OCTET, BYTESIZE    },
    {   "BYTE ",              "WBorRight",    OCTET, BYTESIZE    },
    {   "BYTE ",              "WBorBottom",   OCTET, WORDSIZE    },
    {   "struct TextAttr *",  "Font",         ALIEN, PTRSIZE     },
    {   "struct ViewPort ",   "ViewPort",  ALIEN, SZ(ViewPort)   },
    {   "struct RastPort ",   "RastPort",  ALIEN, SZ(RastPort)   },
    {   "struct BitMap ",     "BitMap",    ALIEN, SZ(BitMap)     },
    {   "struct Layer_Info ", "LayerInfo", ALIEN, SZ(Layer_Info) },
    {   "struct Gadget *",    "FirstGadget",  S_PTR, PTRSIZE     },
    {   "UBYTE ",             "DetailPen",    OCTET, BYTESIZE    },
    {   "UBYTE ",             "BlockPen",     OCTET, BYTESIZE    },
    {   "USHORT ",            "SaveColor0",   MOT16, WORDSIZE    },
    {   "struct Layer *",     "BarLayer",     ALIEN, PTRSIZE     },
    {   "UBYTE *",            "ExtData",      ALIEN, PTRSIZE     },
    {   "UBYTE *",            "UserData",     ALIEN, PTRSIZE     }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)scr);
        switch (choix = GetChoix())
        {   case GGID_CHOIX + 0:
                prScreen(scr->NextScreen);
                break;
            case GGID_CHOIX + 1:
                prWindow(scr->FirstWindow);
                break;
            case GGID_CHOIX + 25:
                prGadget(scr->FirstGadget);
                break;
        }
    }
}
/***** Window ****************************************************/
void prWindow(struct Window *win)
{
static struct StructureInfo si[] =
{   {   "struct Window *",    "NextWindow",   S_PTR, PTRSIZE  },
    {   "SHORT ",             "LeftEdge",     MOT16, WORDSIZE },
    {   "SHORT ",             "TopEdge",      MOT16, WORDSIZE },
    {   "SHORT ",             "Width",        MOT16, WORDSIZE },
    {   "SHORT ",             "Height",       MOT16, WORDSIZE },
    {   "SHORT ",             "MouseY",       MOT16, WORDSIZE },
    {   "SHORT ",             "MouseX",       MOT16, WORDSIZE },
    {   "SHORT ",             "MinWidth",     MOT16, WORDSIZE },
    {   "SHORT ",             "MinHeight",    MOT16, WORDSIZE },
    {   "USHORT ",            "MaxWidth",     MOT16, WORDSIZE },
    {   "USHORT ",            "MaxHeight",    MOT16, WORDSIZE },
    {   "ULONG ",             "Flags",        MOT32, LONGSIZE },
    {   "struct Menu *",      "MenuStrip",    S_PTR, PTRSIZE  },
    {   "UBYTE *",            "Title",        CHAIN, PTRSIZE  },
    {   "struct Requester *", "FirstRequest", ALIEN, PTRSIZE  },
    {   "struct Requester *", "DMRequest",    ALIEN, PTRSIZE  },
    {   "SHORT ",             "ReqCount",     MOT16, WORDSIZE },
    {   "struct Screen *",    "WScreen",      S_PTR, PTRSIZE  },
    {   "struct RastPort *",  "RPort",        ALIEN, PTRSIZE  },
    {   "BYTE ",              "BorderLeft",   OCTET, BYTESIZE },
    {   "BYTE ",              "BorderTop",    OCTET, BYTESIZE },
    {   "BYTE ",              "BorderRight",  OCTET, BYTESIZE },
    {   "BYTE ",              "BorderBottom", OCTET, BYTESIZE },
    {   "struct RastPort *",  "BorderRPort",  ALIEN, PTRSIZE  },
    {   "struct Gadget *",    "FirstGadget",  S_PTR, PTRSIZE  },
    {   "struct Window *",    "Parent",       S_PTR, PTRSIZE  },
    {   "struct Window *",    "Descendant",   S_PTR, PTRSIZE  },
    {   "USHORT *",           "Pointer",      ALIEN, PTRSIZE  },
    {   "BYTE ",              "PtrHeight",    OCTET, BYTESIZE },
    {   "BYTE ",              "PtrWidth",     OCTET, BYTESIZE },
    {   "BYTE ",              "XOffset",      OCTET, BYTESIZE },
    {   "BYTE ",              "YOffset",      OCTET, BYTESIZE },
    {   "ULONG ",             "IDCMPFlags",   MOT32, LONGSIZE },
    {   "struct MsgPort *",   "UserPort",     ALIEN, PTRSIZE  },
    {   "struct MsgPort *",   "WindowPort",   ALIEN, PTRSIZE  },
    {   "struct IntuiMessage *", "MessageKey", ALIEN, PTRSIZE },
    {   "UBYTE ",             "DetailPen",    OCTET, BYTESIZE },
    {   "UBYTE ",             "BlockPen",     OCTET, BYTESIZE },
    {   "struct Image *",     "CheckMark",    ALIEN, PTRSIZE  },
    {   "UBYTE *",            "ScreenTitle",  CHAIN, PTRSIZE  },
    {   "SHORT ",             "GZZMouseX",    MOT16, WORDSIZE },
    {   "SHORT ",             "GZZMouseY",    MOT16, WORDSIZE },
    {   "SHORT ",             "GZZWidth",     MOT16, WORDSIZE },
    {   "SHORT ",             "GZZHeight",    MOT16, WORDSIZE },
    {   "UBYTE *",            "ExtData",      ALIEN, PTRSIZE  },
    {   "BYTE *",             "UserData",     ALIEN, PTRSIZE  },
    {   "struct Layer *",     "WLayer",       ALIEN, PTRSIZE  },
    {   "struct TextFont *",  "IFont",        ALIEN, PTRSIZE  }
};
```

```
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)win);
        switch (choix = GetChoix())
        {   case GGID_CHOIX + 0:
                prWindow(win->NextWindow);
                break;
            case GGID_CHOIX + 12:
                prMenu(win->MenuStrip);
                break;
            case GGID_CHOIX + 17:
                prScreen(win->WScreen);
                break;
            case GGID_CHOIX + 24:
                prGadget(win->FirstGadget);
                break;
            case GGID_CHOIX + 25:
                prWindow(win->Parent);
                break;
            case GGID_CHOIX + 26:
                prWindow(win->Descendant);
                break;
        }
    }
}
/***** Gadget ********************************************************/
void prGadget(struct Gadget *gg)
{
static struct StructureInfo si[] =
{   {   "struct Gadget *",     "NextGadget",     S_PTR, PTRSIZE  },
    {   "SHORT ",              "LeftEdge",       MOT16, WORDSIZE },
    {   "SHORT ",              "TopEdge",        MOT16, WORDSIZE },
    {   "SHORT ",              "Width",          MOT16, WORDSIZE },
    {   "SHORT ",              "Height",         MOT16, WORDSIZE },
    {   "USHORT ",             "Flags",          MOT16, WORDSIZE },
    {   "USHORT ",             "Activation",     MOT16, WORDSIZE },
    {   "USHORT ",             "GadgetType",     MOT16, WORDSIZE },
    {   "APTR ",               "GadgetRender",   ALIEN, PTRSIZE  },
    {   "APTR ",               "SelectRender",   ALIEN, PTRSIZE  },
    {   "struct IntuiText *",  "GadgetText",     ALIEN, PTRSIZE  },
    {   "LONG ",               "MutualExclude",  MOT32, LONGSIZE },
    {   "APTR ",               "SpecialInfo",    ALIEN, PTRSIZE  },
    {   "USHORT ",             "GadgetID",       MOT16, WORDSIZE },
    {   "APTR ",               "UserData",       ALIEN, PTRSIZE  }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)gg);
        switch(choix = GetChoix())
        {   case GGID_CHOIX + 0:
                prGadget(gg->NextGadget);
                break;
        }
    }
}

/***** Menu **********************************************************/
void prMenu(struct Menu *mnu)
{
static struct StructureInfo si[] =
{   {   "struct Menu *",       "NextMenu",   S_PTR, PTRSIZE  },
    {   "SHORT ",              "LeftEdge",   MOT16, WORDSIZE },
    {   "SHORT ",              "TopEdge",    MOT16, WORDSIZE },
    {   "SHORT ",              "Width",      MOT16, WORDSIZE },
    {   "SHORT ",              "Height",     MOT16, WORDSIZE },
    {   "USHORT ",             "Flags",      MOT16, WORDSIZE },
    {   "UBYTE *",             "MenuName",   CHAIN, PTRSIZE  },
    {   "struct MenuItem *",   "FirstItem",  ALIEN, PTRSIZE  },
    {   "SHORT ",              "JazzX",      MOT16, WORDSIZE },
    {   "SHORT ",              "JazzY",      MOT16, WORDSIZE },
    {   "SHORT ",              "BeatX",      MOT16, WORDSIZE },
    {   "SHORT ",              "BeatY",      MOT16, WORDSIZE }
};
SHORT choix = 0;
    while (choix != GGID_RETOUR)
    {   MakeBuffers(si, SI_SIZE, (UBYTE *)mnu);
        switch(choix = GetChoix())
        {   case GGID_CHOIX + 0:
                prMenu(mnu->NextMenu);
                break;
        }
    }
}
```

LISTING 4

```
/*
* SpyGadgets.c
* Contient la définition de la fenêtre et des gadgets
* "standard" (ie. 'Retour' et le slider)
*/
#include <exec/types.h>
#include <intuition/intuition.h>

#include "Spy.h"

/*********************************************************************
* Gadgets de choix des structures à visionner                        *
*********************************************************************/
struct Gadget ChoixGadget[MAX_LINES];
/*********************************************************************
* Gadget Slider                                                      *
*********************************************************************/
struct PropInfo SliderInfo =
{   AUTOKNOB|FREEVERT, MAXPOT, MAXPOT, MAXBODY, MAXBODY
};

struct Image SliderImage =
{   0, 0, 8, 154, 0, NULL, 0x00,0x01, NULL
};

struct Gadget SliderGadget =
{   NULL,
    -15, 10, 16, -10,
    GADGIMAGE|GRELRIGHT|GRELHEIGHT,
    RELVERIFY|GADGIMMEDIATE|RIGHTBORDER,
    PROPGADGET,
    (APTR)&SliderImage,
    NULL, NULL, NULL, (APTR)&SliderInfo, GGID_SLIDER, NULL
};
/*********************************************************************
* Gadget "Retour"                                                    *
*********************************************************************/
SHORT RetourVectors[] =
{   0, 0, 101, 0, 101, 11, 0, 11, 0, 0
};

struct Border RetourBorder =
{   -1, -1, 3, 0, JAM1, 5, RetourVectors, NULL
};

struct IntuiText RetourText =
{   1, 0, JAM2, 26, 1, NULL, "Retour", NULL
};

struct Gadget RetourGadget =
{   &SliderGadget,
    15, -15, 100, 10,
    GRELBOTTOM,
    RELVERIFY|BOTTOMBORDER,
    BOOLGADGET,
    (APTR)&RetourBorder,
    NULL,
    &RetourText, NULL, NULL, GGID_RETOUR, NULL
};
/*********************************************************************
* Fenêtre                                                            *
*********************************************************************/
struct NewWindow nw =
{   0, 11, 640, 194,
    -1, -1,
    CLOSEWINDOW|GADGETUP|RAWKEY,
    WINDOWCLOSE|WINDOWDRAG|WINDOWDEPTH|ACTIVATE,
    &RetourGadget,
    NULL,
    "Spy - par Max pour ANT",
    NULL, NULL,
    0, 0, 0, 0,
    WBENCHSCREEN
};
/*********************************************************************
* PrepareGadgets() - Initialise les gadgets de choix                 *
*********************************************************************/
void PrepareGadgets(void)
{
register WORD i;

    for (i = 0; i < MAX_LINES; i++)
    {   register struct Gadget *g = &ChoixGadget[i];

        g->NextGadget    = NULL;
        g->LeftEdge      = 10;
        g->TopEdge       = (i * 10) + TOP_LINE;
        g->Width         = -36;
        g->Height        = 10;
        g->Flags         = GADGHCOMP|GRELWIDTH;
        g->Activation    = RELVERIFY;
        g->GadgetType    = BOOLGADGET;
        g->GadgetRender  = NULL;
        g->SelectRender  = NULL;
        g->MutualExclude = NULL;
        g->SpecialInfo   = NULL;
        g->GadgetID      = GGID_CHOIX;
        g->UserData      = NULL;
    }
}
```

Par Max

SUB.way I : LA COMMUNICATION

Maintenant que nous avons vu les principes de base des messages d'Exec, nous allons mettre tout celà en pratique, en construisant deux programmes distincts comprenant un protocole de communication qui leur sera propre.

Bien entendu, dans un tel cas, les deux programmes doivent connaître ledit protocole. Un message étant avant tout destiné à véhiculer des informations, il faut que l'envoyeur aussi bien que le destinataire sachent de quelle manière ces informations sont organisées. L'exemple le plus frappant est, encore une fois, Intuition : lorsqu'une application reçoit un message en rapport avec sa fenêtre, Intuition remplit d'une manière précise et arrêtée un IntuiMessage, qui décrit le type de l'évènement survenu (menu, souris, clavier...), donne la position de la souris, l'heure système, l'état des touches spéciales du clavier (CTRL, ALT, SHIFT et AMIGA), etc. En fait, lorsque l'on y regarde d'un peu plus près, l'on trouve la définition exacte de l'IntuiMessage dans les fichiers includes standards "intuition/intuition.h" et "intuition/intuition.i" :

```
struct IntuiMessage
{   struct Message ExecMessage;
    ULONG Class;
    USHORT Code;
    USHORT Qualifier;
    APTR IAddress;
    SHORT MouseX, MouseY;
    ULONG Seconds, Micros;
    struct Window *IDCMP Window;
    struct IntuiMessage *SpecialLink;
};
```

Comme on peut le constater, un IntuiMessage commence par un message Exec standard, ceci afin que les fonctions dédiées d'exec.library puissent le gérer sans problème (voir plus loin). Si nous définissons notre propre type de message, il faudra absolument respecter ce schéma. Les autres champs de cette structure ne nous intéressent pas ici et ne seront pas donc décris.

TYPES DE MESSAGES

Un message est lui aussi décrit par une structure C, que l'on trouve dans "exec/ports.h" et "exec/ports.i" :

```
struct Message
{   struct Node mn_Node;
    struct MsgPort *mn_ReplyPort;
    UWORD mn_Length;
};
```

On retrouve en premier lieu une structure Node classique (cf. mois dernier) destinée à lier les messages entre eux (la tête de la liste étant le champ mp_MsgList de la structure MsgPort, vue elle aussi le mois dernier). Suit un pointeur sur le port de messages de l'expéditeur, pour qu'Exec sache qui l'avait envoyé lorsque le destinataire y répondra, et la taille des données suivant cet entête, exprimée sur 16 bits non signés. Ce qui, lorsque l'on compte bien, peut désigner des données allant jusqu'à 65536 octets, soit 64 Ko. Une telle proportion sera bien entendue rarement atteinte, un message étant typiquement composé de pointeurs sur des zones mémoires précises. M'enfin, encore une fois, chacun en fait ce qui lui plaît.

Le type du noeud (champ ln_Type de la structure Node) est NT_MESSAGE pour un message venant tout juste d'être posté et NT_REPLYMSG pour un message auquel le destinataire vient de répondre. Cette distinction permet à Exec de ne pas se mélanger les pinceaux lors du tri des noeuds. A noter qu'un message envoyé est inséré dans la liste des messages du port destinataire avec la fonction Enqueue(), c'est-à-dire de manière automatiquement triée par ordre décroissant des priorités (le message à la plus haute priorité passe en premier). Cette particularité peut parfois permettre des applications intéressantes. Notez enfin qu'Exec se charge tout seul comme un grand de modifier le type du message (NT_MESSAGE à NT_REPLYMSG) lorsque le besoin s'en fait sentir ; celà est totalement transparent pour les applications concernées.

COMMUNIQUONS

La bibliothèque Exec offre quatre fonctions dédiées à la gestion des messages. Ce sont PutMsg() pour l'envoi, GetMsg() pour la réception, ReplyMsg() pour la réponse et WaitPort() pour l'attente. On peut également ajouter Wait(), déjà entrevue le mois dernier : alors que WaitPort() attend qu'un message (en fait, un signal) arrive sur un port particulier, Wait() permet d'attendre sur plusieurs ports à la fois, donc plusieurs messages de plusieurs sources différentes.

Une tâche désirant envoyer un message à une autre utilisera donc PutMsg(), ce qui implique évidemment qu'elle connaisse l'adresse du port de réception. Elle devra par la suite attendre sur son propre port que le message lui soit retourné par le destinataire, à la suite de quoi le message pourra être libéré - s'il avait été alloué avec AllocMem() - ou ré-utilisé le cas échéant. Quoiqu'il arrive, l'expéditeur ne doit pas modifier le contenu du message avant que le destinataire n'y ait effectivement répondu (rappel : les messages sont transmis par adresse, et non par copie dans une mémoire tampon).

DEMONSTRATION

Les deux programmes ci-dessous communiquent entre eux au moyen d'un message défini par nos soins. Le premier, transmet tous les caractères tapés au clavier au second, qui les affiche alors dans sa fenêtre. Le contrôleur teste également si l'un des caractères reçus est ESC (ASCII 27), auquel cas il envoie un message à l'afficheur pour lui signaler qu'il est temps de quitter. Cet exemple n'est pas certes très utile tel quel, mais il démontre bien les mécanismes de la communication intertâches.

Le message utilisé est défini dans le fichier d'entête "Message.h" commun aux deux programmes, par la structure que voici :

```
struct MaxMessage
{   struct Message msg;     /* Message Exec standard */
    ULONG type;     /* type du message */
    ULONG code;     /* code de la touche appuyée */
};
```

Pour le reste, les listings sont suffisamment commentés pour être facilement compréhensibles de tout un chacun.

LISTING 1

```
/*
* Message.h
*
* Fichier entête commun à Controleur.c et Afficheur.c
*
* 1991, par Max pour ANT
*/
#include <exec/types.h>
#include <exec/ports.h>

/* Noms des MsgPorts du Contrôleur et de l'Afficheur */
#define controlName     "Controleur.Port"
#define afficheName     "Afficheur.Port"

/* Petits raccourcis pratiques */
#define controlTask     (controlPort->mp_SigTask)
#define controlMask     (1L<<controlPort->mp_SigBit)
#define afficheMask     (1L<<affichePort->mp_SigBit)
#define afficheTask     (affichePort->mp_SigTask)

/* structure MaxMessage */
struct MaxMessage
{       struct Message msg;     /* Message Exec standard */
```

```
    ULONG type;     /* Type du message (voir ci-dessous) */
    ULONG code;     /* code de la touche appuyée */
};

/* types de MaxMessages possibles */
#define TOUCHE 1L  /* touche appuyée */
#define FIN    2L  /* mettre fin au programme */

#define MAXMSGSIZE (sizeof(struct MaxMessage)-sizeof(struct Message))

/*************************************************************************
```

Dernière minute :

Un détail que j'ai oublié de préciser dans l'article : comment les 2 programmes s'y prennent-ils pour savoir si l'autre est chargé ?

C'est tout bête : chaque programme recherche si le MsgPort de l'autre est déjà créé. Si ce n'est pas le cas, il attend patiemment avec Wait() que l'autre programme se charge et lui signale son arrivée. Si au contraire le MsgPort cherché existe, celà veut dire que l'autre programme est déjà en train d'attendre, auquel cas on lui signale simplement que l'on vient d'arriver.

Plus simplement, le premier arrivé attend l'autre.

```
*************************************************************************/
```

LISTING 2

```
/*
* Controlleur.c
*
* Envoie, par un message particulier, des données à Afficheur.c
*
* 1991, Max pour ANT
*
* SAS/Lattice C 5.10:
* Compilation:
*   lc -cist iv -y Controleur
*
* Linkage:
*   blink FROM LIB:c.o,Controleur.o
*         TO Controleur
*         LIBRARY LIB:amiga.lib,LIB:lc.lib
*         SMALLCODE
*         SMALLDATA
*         NODEBUG
*         DEFINE __main=__tinymain
*/

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

#include <stdio.h>
#include <stdlib.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/console.h>

#include "message.h"

/* Données globales */
struct IOStdReq consoleIO;

struct NewWindow nw =
{   0, 0, 200, 10,
    0, 1,
    RAWKEY,
    WINDOWDRAG|SMART_REFRESH|NOCAREREFRESH|RMBTRAP,
    NULL, NULL, "Controlleur", NULL, NULL,
    0, 0, 0, 0, WBENCHSCREEN
};

struct MaxMessage max =
{   { { NULL, NULL, NT_MESSAGE, 0, NULL }, NULL, MAXMSGSIZE },
    0, 0
};

/* Variables globales */
struct Library *IntuitionBase = NULL;
struct MsgPort *controlPort = NULL, *affichePort = NULL;
struct Window *controlWin = NULL;
struct ConsoleDevice *ConsoleDevice = NULL;
struct MaxMessage *maxmsg = NULL;

/* Prototypes des fonctions */
VOID main(VOID);
VOID cleanexit(LONG code);

/*************************************************************************
* C'est parti !                                                          *
*************************************************************************/
VOID main(VOID)
{
struct MsgPort *winport;
struct IntuiMessage *im;
struct InputEvent ievent = { NULL,IECLASS_RAWKEY,0,0,0,NULL,NULL };
UBYTE buf[4];
BOOL fini = FALSE;

    /* On ne se lance pas deux fois  ! */
    if (FindPort(controlName))
        cleanexit(RETURN_WARN);

    /* Ouvre l'intuition.library */
    if (!(IntuitionBase = OpenLibrary("intuition.library", 33L)))
        cleanexit(RETURN_FAIL);

    /* Ouvre la fenêtre */
    if (!(controlWin = OpenWindow(&nw)))
        cleanexit(RETURN_FAIL);

    /* Prend le pointeur sur le MsgPort de la fenêtre */
    winport = controlWin->UserPort;

    /* Crée un MsgPort pour communiquer avec l'afficheur */
    if (!(controlPort = CreatePort(controlName, NULL)))
        cleanexit(RETURN_FAIL);

    /* Initialise le ReplyPort du message */
    max.msg.mn_ReplyPort = controlPort;

    /* Ouvre le console.device */
    if (OpenDevice("console.device", -1, &consoleIO, 0))
        cleanexit(RETURN_FAIL);

    /* Prend l'adresse de base du console.device */
    /* nécessaire pour RawKeyConvert() */
    ConsoleDevice = (struct ConsoleDevice *)consoleIO.io_Device;

    /* Recherche (et attend au besoin) l'afficheur */
    affichePort = FindPort(afficheName);
    if (!affichePort)
    {   Wait(controlMask);
        affichePort = FindPort(afficheName);
    }
    else
        /* Signale à l'afficheur qu'on est arrivé */
        Signal(afficheTask, afficheMask);

    /* Active notre fenêtre pour recevoir les IDCMP */
    ActivateWindow(controlWin);

    /* Boucle principale */
    while (!fini)
    {   WaitPort(winport);
        while ((im = (struct IntuiMessage *)GetMsg(winport)))
        {   if (im->Class == RAWKEY)
            {   if (!(im->Code & IECODE_UP_PREFIX))
                {   ievent.ie_Code        =im->Code;
                    ievent.ie_Qualifier   =im->Qualifier;
                    ievent.ie_EventAddress=*((APTR*)im->IAddress);
                    if (RawKeyConvert(&ievent, buf, 2, NULL) == 1)
                    {   if (*buf == 27)
                        {   fini = TRUE;
                            max.type = FIN;
                            max.code = '0';
                        }
                        else
                        {   max.type = TOUCHE;
                            max.code = (ULONG)*buf;
                        }
                        /* Envoie le message à l'afficheur... */
                        PutMsg(affichePort,(struct Message *)&max);
```

```
                    /* attend sa réponse... */
                    WaitPort(controlPort);

                    /* et l'enlève de notre MsgPort */
                    GetMsg(controlPort);

                    /* On peut maintenant réutiliser
                       le message comme on veut */
                }
            }
        }
        ReplyMsg((struct Message *)im);
    }
  }
  cleanexit(RETURN_OK);
}

/*********************************************************************
* Referme et libère tout ce qui a été ouvert et alloué et sort    *
*********************************************************************/
VOID cleanexit(LONG code)
{
    if (ConsoleDevice) CloseDevice(&consoleIO);
    if (controlPort)   DeletePort(controlPort);
    if (controlWin)CloseWindow(controlWin);
    if (IntuitionBase) CloseLibrary(IntuitionBase);
    exit(code);
}
```

LISTING 3

```
/*
* Afficheur.c
*
* Affiche les données qui lui sont envoyées par Controleur.c
*
* 1991, Max pour ANT
*
* SAS/Lattice C 5.10
* Compilation :
*   lc -cist -v -yAfficheur
*
* Linkage :
*   blink FROM LIB:c.o,Afficheur.o
*         TO Afficheur
*         LIBRARY LIB:amiga.lib,LIB:lc.lib
*         SMALLCODE
*         SMALLDATA
*         NODEBUG
*         DEFINE _main=__tinymain
*/

#include <exec/types.h>
#include <exec/ports.h>
#include <intuition/intuition.h>
#include <graphics/rastport.h>
#include <libraries/dos.h>

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <proto/exec.h>
#include <proto/intuition.h>
#include <proto/graphics.h>
#include <proto/dos.h>

#include "Message.h"

/* Fenêtre d'affichage */
struct NewWindow nw =
{   0, 11, 640, 30,
    0, 1,
    NULL,   /* aucun IDCMP */
    WINDOWDRAG|SMART_REFRESH|NOCAREREFRESH,
    NULL, NULL, "Afficheur", NULL, NULL,
    0, 0, 0, 0, WBENCHSCREEN
};

/* Variables globales */
struct Library *IntuitionBase = NULL, *GfxBase = NULL;
struct MsgPort *controlPort = NULL, *affichePort = NULL;
struct Window *afficheWin = NULL;
struct MaxMessage *maxmsg = NULL;

/* Prototypes des fonctions */
VOID main(VOID);
VOID cleanexit(LONG code);

/*********************************************************************
* C'est parti !                                                     *
*********************************************************************/
VOID main(VOID)
{
UBYTE *buf = "Touche : -"; /* 10 caractères */
BOOL fini = FALSE;
struct RastPort *rp;

    /* On ne se lance pas deux fois  ! */
    if (FindPort(afficheName))
        cleanexit(RETURN_WARN);

    /* Ouvre l'intuition.library */
    if (!(IntuitionBase = OpenLibrary("intuition.library", 33L)))
        cleanexit(RETURN_FAIL);

    /* Ouvre la graphics.library */
    if (!(GfxBase = OpenLibrary("graphics.library", 33L)))
        cleanexit(RETURN_FAIL);

    /* Ouvre la fenêtre */
    if (!(afficheWin = OpenWindow(&nw)))
        cleanexit(RETURN_FAIL);

    /* Prend le pointeur sur le RastPort de la fenêtre */
    rp = afficheWin->RPort;

    /* Crée un MsgPort pour communiquer avec l'afficheur */
    if (!(affichePort = CreatePort(afficheName, NULL)))
        cleanexit(RETURN_FAIL);

    /* Recherche (et attend au besoin) le contrôleur */
    controlPort = FindPort(controlName);
    if (!controlPort)
    {   Wait(afficheMask); /* attend l'arrivée du contrôleur */
        controlPort = FindPort(controlName);
    }
    else
        /* Signale au contrôleur qu'on est arrivé */
        Signal(controlTask, controlMask);

    /* Boucle principale */
    while (!fini)
    {   WaitPort(affichePort);
        while ((maxmsg = (struct MaxMessage *)GetMsg(affichePort)))
        {   switch(maxmsg->type)
            {   case FIN:
                    fini = TRUE;
                    sprintf(buf, "%-10s", "fini !");
                    break;
                case TOUCHE:
                    buf[9] = (UBYTE)maxmsg->code;
                    break;
                default:
                    break;
            }
            /* Répond au message (le contrôleur attend) */
            ReplyMsg((struct Message *)maxmsg);

            /* Affiche le contenu du buffer */
            Move(rp, 5, 20);
            Text(rp, buf, 10);
        }
    }
    Delay(2 * TICKS_PER_SECOND);   /* petit délai */
    cleanexit(RETURN_OK);      /* avant de partir */
}

/*********************************************************************
* Libère et ferme tout ce qui a été alloué et ouvert               *
*********************************************************************/
VOID cleanexit(LONG code)
{
    if (affichePort)   DeletePort(affichePort);
    if (afficheWin)CloseWindow(afficheWin);
    if (GfxBase)       CloseLibrary(GfxBase);
    if (IntuitionBase) CloseLibrary(IntuitionBase);
    exit(code);
}
```

Par Max

TRANSACTOR : LE RETOUR

Le "Concours Lecteurs" étant officiellement mort, c'est l'occasion pour notre ancienne rubrique "Transactor" de faire son come-back, avec quelques petites modifications minimes.

Un rapide rappel à l'attention de ceux qui ne connaîtraient pas cette rubrique s'impose. Il était une fois un journal canadien en langue anglaise et aujourd'hui disparu, baptisé "Transactor for the Amiga", dont la particularité était de ne proposer que des articles écrits par d'autres programmeurs, souvent connus dans le milieu Amiga, à ses lecteurs. C'est ainsi qu'on y trouvait des articles plus qu'intéressants signés Matthew Dillon, John Toebes et autres Carolyn Sheppner. L'Amiga News-Tech, alors encore partie intégrante de Commodore Revue, eut un jour l'idée d'en faire autant et inaugura une rubrique qu'il baptisa "Transactor", en hommage à son illustre confrère d'outre-Atlantique. La rubrique débuta sur les chapeaux de roues avec la traduction d'un article de John Toebes sur les mille et une façons d'optimiser un programme en assembleur, article par ailleurs très intéressant et fort apprécié des lecteurs. Le but de cet rubrique était de donner à des grands noms de la programmation francophone, la possibilité de s'exprimer sur des sujets qu'ils maîtrisaient parfaitement et de faire partager leurs connaissances aux lecteurs moins savants.
Malheureusement, cette idée ne fit jamais vraiment son chemin, notamment - la honte soit sur nous jusqu'à la cent vingt-cinquième génération - parce que nous ne nous sommes pas donné la peine de rechercher et de contacter les personnes adéquates (je pense en particulier à des gens comme François Rouaix, Jean-Michel Forgeas, Thomas Landspurg ou Nico François). En fait et pour être parfaitement sincère, nous étions sûrs que ces gens viendraient d'eux-même à nous, sans que nous ayons besoin de remuer le petit doigt. Ce qui n'a évidemment pas eu lieu, et la rubrique Transactor a disparu de l'ANT aussi rapidement qu'elle y avait apparu.

LE NOUVEAU TRANSACTOR

Aujourd'hui, tel le phénix, Transactor renaît de ses cendres. Son but est toujours le même - le partage des connaissances - mais la manière a quelque peu changé. Si des grands noms sont toujours attendus et espérés (messieurs, à vos traitements de texte !), le simple lecteur est lui aussi invité à prendre la plume et à nous écrire un article sur quelque domaine qu'il désire, la seule condition étant bien entendu qu'il maîtrise parfaitement son sujet. Celà inclut bien entendu les coders des groupes de démo-makers désireux de montrer leur savoir-faire, des programmeurs de jeux, mais également celui qui s'y connaît en compression de données, en algorythmie générale, en compilateurs, en optimisation de code, en hardware... Tous les sujets sont les bienvenus.
Transactor remplaçant le défunt Concours Lecteurs, il serait injuste de s'en tenir là et de ne rien offrir aux participants. Toute publication dans cette rubrique sera donc rémunérée, au tarif forfaitaire de 1, 500 F HT les deux pages (un peu plus de 1, 000 F TTC). Des articles de plus de deux pages seront publiés en plusieurs fois, avec un maximum de trois mois. C'est évidemment la rédaction de l'ANT qui se réserve le droit de publier ou non tout article proposé. Les conditions générales sont citées dans l'encadré qui devrait se trouver quelque part dans cette page et qu'il vous faudra photocopier et nous envoyer en même temps que vos articles.

ET LE CONCOURS ?

Le Concours Lecteurs n'est pas définitivement mort. Nous ne perdons pas espoir de pouvoir à nouveau vous proposer un jour le DevKit de l'ANT qui, rappelons-le, devait se composer des trois tomes des Rom Kernal Manuals (Addison Wesley Publishing Company, Inc.), de l'assembleur Devpac II (Hisoft Ltd.) et du compilateur Lattice/SAS-C 5.10 (SAS Institute, Inc.). Quitte à modifier quelque peu cette liste.

ET POUR COMMENCER...

Il faut bien un début à tout et ma foi, pour commencer ce Transactor "nouvelle formule", nous avons décidé de donner la parole à un membre de la rédaction, Stéphane Schreiber, qui a beaucoup planché sur la manière d'optimiser la lecture des joysticks dans un programme de jeu. Cet article, qui débute sur la page suivante, vous permettra de vous faire une idée sur ce qu'est Transactor, ce à quoi il ressemble et comment écrire vos propres articles. Une fois de plus, nous comptons sur vous pour alimenter cette rubrique du mieux que vous pourrez.

BIEN LIRE LES JOYSTICKS

Quand on s'attaque à l'écriture d'un jeu, on a toujours besoin d'une routine de lecture de la position des joysticks, par exemple pour décider du mouvement à donner à un sprite et/ou à un scrolling. Bien que celà ne soit pas réellement compliqué sur l'Amiga les routines habituelles ne sont pas forcément les meilleures.

Bien que le hardware de l'Amiga l'autorise à utiliser des joysticks analogiques - ou "proportionnels" - comme l'IBM PC ou l'Apple II, le type le plus courant est celui mis au point par Atari à l'époque de sa console de jeu, la VCS 2600, qui offre simplement quatre switches, un par direction possible (haut, bas, gauche, droite) et qui a été repris par de multiples constructeurs, dont Commodore (on parle d'ailleurs de "standard Atari" pour ce type de joysticks, autant leur laisser au moins ça...). Sur des systèmes plus anciens, comme

CONDITIONS GENERALES

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.
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.
3.) Tout article publié sera rémunéré au tarif forfaitaire de 1, 500 F HT les deux pages ; un article de plus de deux pages sera publié en plusieurs fois, à concurrence de trois épisodes maximum. La paiement a lieu le mois suivant celui de la parution. Les auteurs des articles retenus pour parution seront directement prévenus par téléphone.
4.) Les articles doivent parvenir au format ASCII standard (utilisez un éditeur de textes ou l'option de sauvegarde "text only" ou "ASCII" de votre traitement de texte). Les programmes illustrant les articles, qu'ils soient complets ou de simples routines d'exemple, doivent également être fournis sous forme ASCII, et éventuellement en format "brut" pour les langages en disposant (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'interprêter ses programmes sans difficulté (joindre au besoin une version "run-only" du langage utilisé).
5.) La taille totale du fichier ASCII donne le nombre de signes de votre article ; une page de l'ANT contient en moyenne 7, 000 signes de texte. Une colonne de listing contient 80 lignes de 65 caractères de large. Un programme de 160 lignes occupe donc une page entière. Les programmes en assembleur particulièrement longs peuvent être publiés en trois colonnes de 40 caractères chaque (240 lignes par page).
6.) Toute proposition d'article pour Transactor devra être accompagnée du bon ci-dessous, découpé ou photocopié, mais en aucun cas reproduit à la main.
7.) Toute proposition d'article pour Transactor entraîne l'acceptation par l'auteur que les programmes joints soient placés sur la disquette d'accompagnement de l'ANT. L'auteur doit préciser dans des commentaires en tête de listing s'il place son programme dans le domaine public ou s'il désire conserver son copyright.
8.) Ni l'ANT, ni Commodore Revue SARL, ni les auteurs collaborant à la rubrique Transactor ne sauraient être tenus pour responsables en cas de dommages quelconques survenus à l'ordinateur, suite à une mauvaise utilisation des documents (textes et programmes) publiés dans cette rubrique.
9.) Toute proposition d'article pour Transactor entraîne l'entière acceptation par l'auteur de ce règlement.

le C64 ou l'Amstrad CPC, il suffisait de lire un octet en mémoire ou sur un port pour savoir immédiatement dans quelle(s) direction(s) le joystick était poussé.

Sur l'Amiga, pour des raisons incompréhensibles, c'est un peu plus compliqué ; les bits "haut" et "gauche" occupent deux bits d'un octet, et les "bas" et "droite", deux bits d'un autre octet (l'état du ou des boutons de feu étant quant à lui encore ailleurs). Bien qu'il soit toujours possible de lire ces quatre bits avec une seule instruction (MOVE.W), seul l'état des bits "gauche" et "droite" est directement exploitable ; il faudra d'abord masquer par un EOR l'état des bits "haut" et "bas" avant que de pouvoir prendre leur valeur en compte.

De fait, lorsque que l'on commence à s'intéresser à la chose, la première routine que l'on écrit ressemble souvent à celle publiée dans "la Bible de l'Amiga" :

```
; lecture du joystick - première version
Joy1    move.w    $dff00c,d0          ; JOY1DAT dans d0
        btst      #1,d0      ; test bit 1
        bne       Droite     ; Activé, joystick à droite
        btst      #9,d0      ; test bit 9
        bne       Gauche     ; Activé, joytick à gauche
        move.w    d0,d1      ; copie d0 dans d1
        lsr.w     #1,d1      ; décalage d'un bit à droite
        eor.w     d1,d0      ; masque EOR
        btst      #0,d0      ; test bit 0
        bne       Bas        ; Activé, joystick en bas
        btst      #8,d0      ; test bit 8
        bne       Haut       ; Activé, joystick en haut
Milieu  moveq     #0,d0      ; position centrale
        moveq     #0,d1
        rts
Gauche  moveq     #-1,d0
        moveq     #0,d1
        rts
Droite  moveq     #1,d0
        moveq     #0,d1
        rts
Haut    moveq     #0,d0
        moveq     #-1,d1
        rts
Bas     moveq     #0,d0
        moveq     #1,d1
        rts
```

Cette routine retourne deux valeurs, dans les registres D0 et D1, indiquant respectivement l'état horizontal et l'état vertical du joystick : -1 signifie que le joystick est à gauche (resp. en haut), 0 qu'il est au centre, et 1 qu'il est à droite (resp. en bas). A charge ensuite du programme appelant, de faire ce qu'il convient de ces valeurs.
Ainsi écrite, cette routine fait ce qu'on attend d'elle, mais n'est pas réellement très efficace... En tout cas, vue sa conception, elle serait beaucoup mieux si elle allait piocher dans une table les valeurs correctes : on réduirait ainsi le nombre de tests et de branchements, ce qui ferait gagner autant d'octets et de cycles. On éliminerait également le décalage à droite d'un bit et le masquage par EOR, devenus superflus.

Le problème maintenant est de savoir de quelle manière organiser nos quatre bits pour en faire un index correct pour piocher dans la table ; pour celà, rappelons qu'une lecture de JOY0DAT ou JOY1DAT donne les valeurs suivantes ('G' = gauche, 'H' = haut, 'D'= droite, 'B' = bas, et 'x' indique un bit sans intérêt) :
xxxxxxGHxxxxxxDB

Après plusieurs tests, il apparaît que la manière la plus probante est la suivante :
xxxxxxGHDBxxxxxx

qui s'obtient très facilement par une rotation de deux bits vers la droite de l'octet inférieur. Il ne reste plus qu'à décaler le mot entier de six bits vers la droite, pour obtenir un index valable, de la forme :
xxxxxxxxxxxxGHDB

En étudiant tous les cas avec un débugger, on trouve les valeurs possibles suivantes :

Joy	Valeur	Decimal
-	0000	0
H	0100	4
B	0001	1
G	1100	12
D	0011	3
HG	1000	8
HD	0111	7
BG	1101	13
BD	0010	2

Notre table aura donc l'allure que voici (un 'X' indique une valeur non utilisée, par exemple l'impossible combinaison HBG) :

```
JoyTabX dc.w    0,0,1,1,0,X,X,1,-1,X,X,X,-1,-1,X,X
JoyTabY dc.w    0,1,1,0,-1,X,X,-1,-1,X,X,X,0,1,X,X
```

Cette table peut être un rien optimisée, par exemple en omettant les deux 'X' finaux de chaque ligne (8 octets de gagnés) et en utilisant des valeurs sur des octets plutôt que des mots (50 % de gagnés !). Toutefois, je préfère utiliser des mots, étant donné que dans un jeu, les coordonnées des sprites sont souvent codées sur des mots. Ainsi, dans le programme principal, je n'ai plus qu'à écrire :

```
    bsr Joy
    add.w   d0,sprite_x(a5) ; (je référence TOUJOURS mes varia-
    add.w   d1,sprite_y(a5) ; bles par un registre d'adresse,
    .....   etc.    ; c'est plus court et plus rapide)
```

pour que les coordonnées de mon sprite soient actualisées en fonction de la position du joystick.
Il ne nous reste donc maintenant plus qu'à écrire la version finale de notre routine de lecture des joysticks. Vous remarquerez que je décale le quartet GHDB de cinq bits et non de six, comme annoncé plus haut ; c'est simplement parce j'accède à une table de mots. Les deux instructions

```
    lsr.w   #5,d0
    andi.w  #%11110,d0
```

équivalent en fait aux trois suivantes :

```
    lsr.w   #6,d0
    andi.w  #%1111,d0
    add.w   d0,d0
```

mais sont bien entendues plus courtes et plus rapides. Dans le même ordre d'idées, je lis d'abord la valeur de d1, afin de ne pas modifier d0, qui sert toujours d'index dans l'accès à la table des valeurs en X.
Notez enfin que cette routine n'est prévue que pour le port joystick 1 de l'Amiga (la configuration standard de l'utilisateur voulant que la souris soit branchée dans le port 0 et le joystick dans le port 1). Pour gérer un joystick branché sur le port 0 (par exemple, dans un jeu à deux joueurs), il suffit de remplacer $dff00c (JOY1DAT) par $dff00a (JOY0DAT) ; le reste est strictement identique.

```
; lecture du joystick - seconde version
Joy move.w  $dff00c,d0  ; JOY1DAT dans d0
    ror.b   #2,d0   ; place DB dans les bits 7-6
    lsr.w   #5,d0   ; place GHDB dans les bits 4-1
    andi.w  #%11110,d0  ; masque les bits indésirables
    move.w  JoyTabY(pc,d0.w),d1 ; valeur Y dans d1
    move.w  JoyTabX(pc,d0.w),d0 ; valeur X dans d0
    rts
JoyTabY dc.w    0,1,1,0,-1,0,0,-1,-1,0,0,0,0,1,0,0
JoyTabX dc.w    0,0,1,1,0,0,0,1,-1,0,0,0,-1,-1,0,0
```

CONCLUSION

Mine de rien, on gagne beaucoup à essayer d'améliorer des routines à priori secondaires. Dans ce cas précis, on est passé d'une routine de près de 30 lignes, à une d'à peine 10. La vitesse d'exécution s'en est trouvée considérable multipliée et on s'en sort avec la satisfaction personnelle d'avoir quasiment atteint la perfection. Ce qui est loin d'être désagréable, au niveau de l'égo.

EXECBASE II : LA MEMOIRE DU

L'Amiga étant un système multitâche, la gestion de la mémoire par le système est autrement plus compliquée que celle d'autres systèmes plus banals, comme l'IBM PC ou l'Atari ST. Il lui faut en effet notamment gérer les problèmes de fragmentation qui découlent des multiples allocations successives par plusieurs tâches différentes.

Pour savoir à tout instant quelles sont les zones de mémoire utilisées par des applications et celles disponibles ainsi que leur taille, Exec doit maintenir à jour une liste de pointeurs vers ces zones. Quand une application lui demande de la mémoire - la plupart du temps, via AllocMem(), Exec parcourt ces listes à la recherche de la première zone suffisamment grande pour la contenir. C'est d'ailleurs l'une des premières raisons de la fragmentation : même si plus loin dans la liste, se trouvait une zone de mémoire libre d'exactement la même taille que la requête, Exec allouerait tout de même la première zone suffisament grande. Une fois cette mémoire trouvée dans la liste, Exec l'en retire, jusqu'à ce qu'un appel à FreeMem() l'autorise à l'y replacer. Naturellement, les applications doivent libérer toute la mémoire qu'elles allouent, et seulement celle-ci. En fait, Exec ne rechignera pas à libérer une zone mémoire allouée par quelqu'un d'autre ; simplement, lorsque son légitime propriétaire essaiera à son tour de la libérer, Exec se rendra compte que cette zone est déjà de retour dans la "free list" et assumera que quelque chose ne tourne plus rond dans la machine... Du coup, plutôt que de laisser des applications entrer en conflit, il provoquera un petit gourou (n° #80000009) pour signifier aux petits rigolos concernés qu'on ne la lui fait pas...

Alternativement, Exec ne garde aucune trace de qui alloue quoi. Cela signifie que si une application "oublie" (sans jeu de mots) de rendre sa mémoire, il n'y aura aucun moyen autre que le reset, de la récupérer.

ENCORE UNE LISTE

Comme beaucoup (trop ?) de choses dans l'Amiga, la mémoire est gérée à partir d'une liste, dont la base se trouve dans la structure ExecBase. Son nom est MemList, à ne pas confondre avec la structure MemList, définie dans les includes "exec/memory.h" et"exec/memory.i". Les noms simulaires pourraient facilement vous induire en erreur. Rappel : vous obtenez un pointeur sur la structure ExecBase soit en ouvrant explicitement la bibliothèque exec.library, soit en allant le chercher à l'adresse 4.

La liste de la mémoire libre est organisée comme une "liste de listes". Le champ MemList pointe en fait sur une liste de régions de mémoire, une "région" n'étant finalement rien d'autre qu'une zone de mémoire libre dans laquelle Exec peut aller piocher à sa guise. A l'initialisation du système, une région est d'abord créée pour la Chip-Ram puis pour chaque extension de mémoire connectée (c'est d'ailleurs là l'utilité du programme MergeMem du répertoire System : si deux extensions occupent des adresses contigues, MergeMem regroupe les deux listes en une seule).

Chaque région est décrite par une structure C baptisée MemHeader et définie, elle aussi, dans "exec/memory.h" et "exec/memory.i" :

```
struct MemHeader
{   struct Node mh_Node;
    UWORD mh_Attributes;                                    /*
attributs de la région      */
    struct MemEntry *mh_First;                              /*
première région libre       */
    APTR mh_Lower;     /* adresse la plus basse        */
    APTR mh_Upper;     /* adresse la plus haute + 1    */
    ULONG mh_Free;     /* taille totale de la région */
};
```

Donc, pour obtenir l'adresse de la première région de la liste :

```
struct MemHeader *mh;
mh = (struct MemHeader *)ExecBase->MemList.lh_Head;
```

Normalement, nous avons là pointeur sur la liste de la Chip-Ram. La figure 1 montre celà un peu plus clairement.

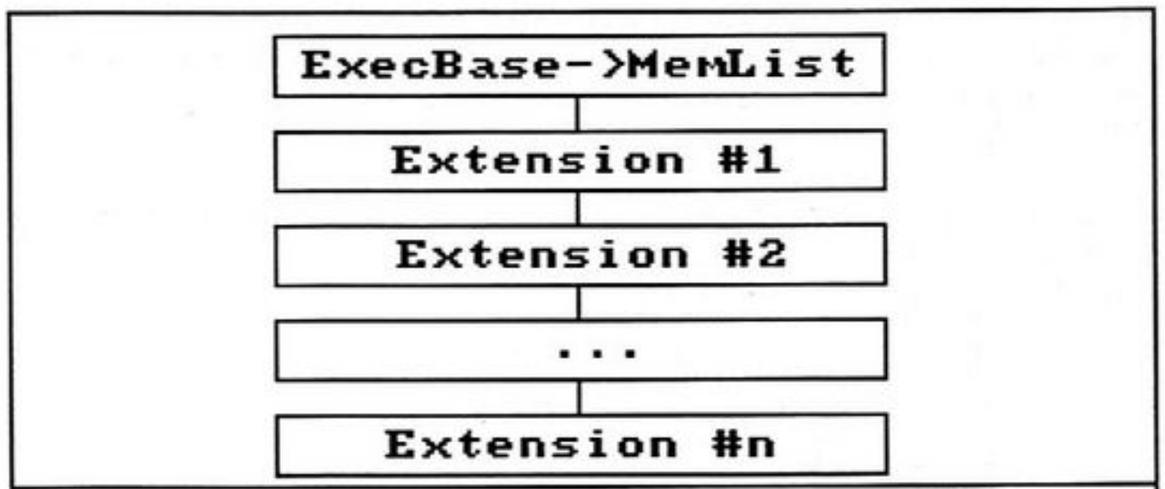

Fig.1: Les listes-mémoire d'Exec

A l'intérieur de cette structure MemHeader, le noeud (node) sert à relier entre elles les différentes régions. Le champ ln_Succ pointe sur la prochaine, le champ ln_Pred, sur la précédente.

La structure MemHeader définit l'adresse la plus basse (mh_Lower) et l'adresse la plus haute (mh_Upper) de la région, ainsi que la taille de mémoire disponible dans cette région (mh_Free). C'est ce champ, mh_Free, qui est décrémenté et incrémenté au fur et à mesure des allocations et des libérations de mémoire par les applications. Le MemHeader contient également un pointeur sur le premier bloc mémoire disponible dans cette région (mh_First).

Un bloc mémoire est appelé en terminologie Amiga, un chunk, et est bien entendu défini par une structure C que l'on trouve, encore une fois, dans "exec/memory.h" et "exec/memory.i" :

```
struct MemChunk
{   struct MemChunk *mc_Next;   /* pointe sur le prochain bloc */
    ULONG mc_Bytes;             /* taille de ce bloc */
};
```

Un champ mc_Next à NULL indique la fin de la liste des blocs. Toute la mémoire située entre la fin de la région et son dernier chunk est libre.

A l'initialisation du système, chaque région ne contiendra qu'un seul bloc. Dans le MemHeader, mh_First et mh_Lower sont égaux, ainsi que mh_Upper et mh_Free.

La plus petite quantité de mémoire allouable dans une région est de huit octets, car c'est exactement la taille d'un MemChunk (quatre octets pour le pointeur mc_Next et quatre autres pour le mot long mc_Bytes). De toute façon, AllocMem() aussi bien que FreeMem() arrondissent et la taille demandée et l'adresse de la mémoire réservée à leur plus grand multiple de 8. La plus grande quantité est évidemment égale à la taille de cette région.

A la première allocation dans une région donnée, la taille du bloc demandé est additionnée à mh_First et soustraite de mh_Free. Dans le MemChunk, mc_Next reste à NULL et la taille demandée est soustraite de mc_Bytes.

Si une deuxième demande d'allocation pour cette même région survient, la taille du bloc demandé sera à nouveau additionnée à mh_First et soustraite de mh_Free et de mc_Bytes. Il n'y aura toujours qu'un chunk dans la liste, simplement sa taille aura diminué.

Le premier cas de fragmentation de la mémoire survient lorsque le premier bloc est libéré, alors que le second reste alloué. Dans ce cas, mh_First sera d'abord copié dans mc_Next puis remis à sa valeur initiale (égale à mh_Lower) et mh_Free sera augmenté de la taille de ce bloc. Cette taille sera également placée dans mc_Bytes. Arrivé à ce stade, nous aurons :

* Pour le MemHeader :
- mh_First pointant sur le MemChunk

- mh_Free égal à la taille de la région moins la taille du second bloc

* Pour le MemChunk :
- mc_Next pointant sur la fin du second bloc
- mc_Bytes égal à la taille du premier bloc

La figure 2 visualise tout cela de manière un peu plus concise.

Exemple de la carte d'extension 512 Ko de l'Amiga 500 (adresses $C00000 à $C7FFFF)

I : la mémoire est ajoutée à la liste des régions (une nouvelle région est créée):

```
MemHeader              MemChunk ($C00000)
mh_First = $C00000 →   mc_Next  = NULL
mh_Lower = $C00000     mc_Bytes = $80000
mh_Upper = $C80000
mh_Free  = $80000
```

II : Allocation d'un bloc de $1000 octets

```
MemHeader              MemChunk ($C00000)
mh_First = $C01000 →   mc_Next  = NULL
mh_Lower = $C00000     mc_Bytes = $7F000
mh_Upper = $C80000
mh_Free  = $7F000
```

III : Allocation d'un second bloc, de $2000 octets

```
MemHeader              MemChunk ($C00000)
mh_First = $C03000 →   mc_Next  = NULL
mh_Lower = $C00000     mc_Bytes = $7D000
mh_Upper = $C80000
mh_Free  = $7D000
```

IV : Libération du 1er bloc (le second reste alloué)

```
MemHeader              MemChunk ($C00000)    MemChunk 2 ($C03000)
mh_First = $C00000 →   mc_Next  = $C03000 →  mc_Next  = NULL
mh_Lower = $C00000     mc_Bytes = $1000      mc_Bytes = $7D000
mh_Upper = $C80000
mh_Free  = $7E000
```

Fig.2: Le processus d'allocation et de libération de mémoire

Quant le premier bloc sera à son tour libéré (en assumant qu'aucune autre allocation n'ait eu lieu dans cette région), le MemHeader et le MemChunk retrouveront leur configuration originale, et le second MemChunk sera tout simplement oublié.

LES FONCTIONS D'EXEC

Exec offre plusieurs paires de routines pour allouer et libérer la mémoire, la plus connue et la plus employée étant bien évidemment la paire AllocMem()/FreeMem(). On précise lors de l'appel à ces fonctions, la taille de mémoire que l'on désire et ses attributs, c'est-à-dire CHIP, FAST et/ou PUBLIC. La mémoire CHIP est celle accessible par les processeurs spécialisés (Denise, Agnus et Paula) et devra être employée pour tout ce qui concerne les images devant être "blittées", le son, les buffers disquette, etc. La mémoire FAST est celle qui n'est pas CHIP ; ces deux attributs sont donc contradictoires et s'annulent l'un l'autre. Enfin, la mémoire PUBLIC est la mémoire destinée à être partagée par plusieurs tâches, ou bien par une tâche donnée et des interruptions... Elle peut aussi bien être CHIP que FAST. Dans la version actuelle d'Exec (jusqu'à la V34 correspondant au Kickstart 1.3), la mémoire PUBLIC n'est pas gérée et cet attribut est tout simplement ignoré, mais il est vivement conseillé de le mettre quand besoin est, pour des raisons de compatibilité avec les futures versions du système (j'avoue à ma grande honte ne pas savoir de quoi il en retourne dans la version V36 - Kickstart 2.0).

Lorsqu'aucun attribut particulier n'est requis, Exec essaie d'abord d'allouer de la mémoire FAST (sous réserve qu'il y en ait de disponible et en quantité suffisante), sinon de la CHIP. Par conséquent, il est quasiment inutile de préciser que l'on désire de la FAST, sauf cas exceptionnel. Si aucune mémoire n'est disponible, un pointeur NULL est renvoyé, laissant à la charge du programme appelant le soin de réagir en conséquence (normalement, sortie "propre" avec avertissement à l'utilisateur).

Notez au passage que si Exec essaie d'abord d'allouer de la mémoire FAST, c'est simplement parce que celle-ci a une priorité plus élevée que la mémoire CHIP. Avec un petit programme adéquat, on peut très bien forcer l'inverse.

Il existe également la paire AllocEntry() et FreeEntry(), qui de manière interne, appele AllocMem() et FreeMem(). Ces fonctions permettent d'allouer et de libérer plusieurs blocs de mémoire en un seul appel, grâce à une structure MemEntry définie, encore une fois, dans "exec/memory.h" et "exec/memory.i" :

```
struct MemEntry
{   union
    { ULONG meu_Reqs; /* attributs demandés    */
      APTR  meu_Adr;  /* adresse de la mémoire */
    } me_Un;
    ULONG me_Length;
};
```

Rappel : en langage C, une union, à l'inverse d'une structure, ne réserve pas de mémoire pour chacun de ses membres, mais seulement pour le plus long ; ainsi, elle peut prendre à tour de rôle le type de chacun de ses membres. Ici, la structure MemEntry occupe donc 8 octets de mémoire, et non 12 comme on pourrait le penser à première vue.

AllocEntry() et FreeEntry() utilisent également une seconde structure, connue sous le nom de MemList (encore une fois, ne confondez pas avec le champ MemList de la structure ExecBase ! Les deux n'ont en commun que le nom !). Cette structure est de taille variable, et est définie ainsi :

```
struct MemList
{   struct Node     ml_Node;
    UWORD           ml_NumEntries;
    struct MemEntry ml_ME[1];
};
```

Le noeud n'est là que pour vous permettre de lier plusieurs MemLists entre elles ; il est totalement ignoré par AllocEntry() et FreeEntry(). Vient ensuite le nombre de structures MemEntry dans cette MemList, puis les MemEntrys elles-mêmes. Cette possibilité n'est que très rarement utilisée sur l'Amiga.

Troisème paire de routines, Allocate() et Deallocate(). Elles agissent de manière similaire à AllocMem() et à FreeMem(), mais permettent de prendre en compte une région de mémoire propre à l'utilisateur. En général, ce sera un gros bloc alloué avec AllocMem(), dans lequel allouera d'autres petits blocs en fonction des besoins. C'est l'idéal, par exemple, pour un interpêteur Basic. L'un des programmes de démonstration utilise ces deux fonctions pour mettre en évidence le principe de l'allocation de mémoire développé dans cet article. Pour information, sachez tout de même qu'AllocMem() et FreeMem() appellent, de manière interne, ces fonctions, qui sont donc la base de toute la gestion de la mémoire par Exec.

Enfin, on peut citer par soucis d'exhaustivité, d'autres routines d'allocations que l'on trouve dans d'autres bibliothèques qu'Exec. Je pense notamment à AllocRemember() et FreeRemember() d'intuition.library, à AllocRaster() et FreeRaster() de la graphics.library... Ces fonctions spécialisées sont destinées à faciliter l'allocations de certaines mémoires particulières (bitplanes, etc.) et appellent toutes, de façon interne, AllocMem() et FreeMem().

TRAVAIL PRATIQUE

La place qui m'était impartie étant bien remplie, je vais devoir remettre au mois prochain les programmes, avec notamment une illustration par l'exemple du processus d'allocation et de libération de la mémoire, et un petit truc pour rendre la mémoire CHIP plus prioritaire que la FAST - un ChipMemFirst en quelque sorte.

Par P. Vautrin

HARDWARE : LES DISQUES

Nous allons aujourd'hui mettre en pratique nos connaissances acquises la dernière fois, en mettant au point une routine de lecture d'une disquette entière piste par piste.

Nous avons déjà effectué les trois-quarts du travail, avec les routines pour déplacer la tête de lecture, sélectionner un drive, etc. passées en revue le mois dernier. Il nous reste tout de même un gros morceau à traiter, à savoir le décodage des données MFM brutes, telles qu'elles sont lues par le DMA disque. Si la réalisation d'une telle routine n'est pas réellement compliquée, elle nécessite toutefois une bonne compréhension du format de codage.

VOUS AVEZ DIT MFM ?

Mais au fait, pourquoi donc coder les données avant de les écrire sur disque ? Simplement pour des raisons de sécurité. Divers codages existent, que nous n'aborderons pas ici (GCR, etc.). Le principe du MFM est d'insérer un bit de parité entre chaque bit de donnée réelle. Ce qui implique logiquement que le buffer MFM doit être deux fois plus grand que celui des données réelles.

Le tordu qui a inventé le codage MFM a décidé de séparer les bits de données en deux zones distinctes. On code d'abord les bits de numéros impairs, puis les bits de numéros pairs, en accord avec le tableau suivant :

```
Donnée      Codage MFM
1           01
0           10 (si précédé d'un 0)
0           00 (si précédé d'un 1)
```

Le décodage est assez simple (en tout cas, beaucoup plus simple que le codage). Reportez-vous à la routine "MFMUncode" du programme pour voir comment l'on s'y prend. A noter que le blitter, paraît-il, pourraît être utilisé pour coder et décoder le MFM... Je n'ai jamais vu l'intérêt d'une telle pratique : le temps-machine gagné serait infime, et de plus, celà oblige à utiliser un buffer de décodage situé en Chip-Ram, alors qu'en utilisant le 68000, ce buffer peut indifféremment se trouver en Fast-Ram.

FORMAT DES SECTEURS SUR DISQUE

En plus du codage, chaque secteur possède un entête permettant, entre autres, de vérifier que la lecture s'est bien passée. Le format de cet entête est le suivant :

```
2 octets à $00 ($AAAA chacun en MFM)
2 octets à $A1 ($4489 chacun en MFM)
1 mot long de description du secteur (8 octets MFM)
16 octets réservés à AmigaDOS (32 octets MFM)
1 mot long de checksum pour l'entête (8 octets MFM)
1 mot long de checksum pour les données (8 octets MFM)
512 octets de données (1024 octets MFM)
```

Les deux octets nuls marquent le début du secteur et ne sont pas lus par le DMA.
Les deux octets suivants servent indiquer au DMA quand il doit commencer à écrire les données lues en mémoire. Cette valeur $4489 est particulière en MFM : elle correspond à la valeur $A1 codée sans prendre en compte le bit numéro 7. De cette manière, on est sûr que cette valeur ne se retrouvera jamais à l'intérieur des données du secteur, et par la même, qu'elle en indique bien le début. Il est évidemment possible, pour des raisons de protection par exemple, d'utiliser une autre valeur que $4489.
Le mot long de description du secteur est formé de quatre octets, qui contiennent, dans l'ordre : $FF (format Amiga), le numéro de la piste, le numéro du secteur, et le nombre de secteurs restant à lire avant d'atteindre la fin de la piste. Le DMA disque lisant une piste entière depuis une position aléatoire, on connait ainsi la position du secteur dans la piste.
Les 16 octets suivants sont réservés à AmigaDOS, qui s'en sert pour décider de la manière dont il va assigner les secteurs aux fichiers.

Les deux checksums permettent de vérifier, pour chaque secteur, que la lecture s'est bien déroulée. Il s'agit d'un ou exclusif (EOR) de chaque mot long de l'entête et des données, respectivement.

Encore une fois, je ne peux que vous conseiller d'étudier de plus près cette routine. Notez simplement qu'elle ne peut lire que des disquettes au format AmigaDOS standard (OFS ou FFS), et foire complètement avec les disquettes spéciales (protection de jeu, etc.).

Voilà. Un dernier point avant de vous laisser étudier ce programme : je vous avais promis de bannir cet inacceptable délai logiciel entre deux impulsions STEP du drive, au profit d'un délai hard utilisant l'un des Timers offerts par les CIAs. Je n'en n'ai malheureusement pas eu le temps. Aussi, c'est avec confusion et humilité que je vous prie de bien vouloir pardonner mes excuses.

```
*****************************************************************
* - lecture d'une disquette placée en DF0: piste par piste      *
*   sans passer par le système.                                 *
* - décodage MFM des données lues                               *
*                                                               *
* 1991 by Loïc Far pour Amiga NewsTech                          *
*****************************************************************

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

        opt o+,ow-

; ************************************
rCIAA   EQUR    A4  ; ce sera plus facile à lire comme ça
rCIAB   EQUR    A3

; ************************************
CIAA    EQU $bfe001 ; adresse du CIA-A
CIAB    EQU $bfd000 ; adresse du CIA-B

; bits utiles du CIAA-PRA
DSKRDY  EQU 5   ; drive prêt
DSKTRK0 EQU 4   ; piste 0 atteinte
DSKPROT EQU 3   ; disquette protégée en écriture
DSKCHNG EQU 2   ; disquette changée

; bits utiles du CIAB-PRB
DSKMTR  EQU 7   ; control moteur
DSKSEL3 EQU 6   ; sélection DF3:
DSKSEL2 EQU 5   ;            DF2:
DSKSEL1 EQU 4   ;            DF1:
DSKSEL0 EQU 3   ;            DF0:
DSKSIDE EQU 2   ; sélection face
DSKDIR  EQU 1   ; direction
DSKSTEP EQU 0   ; déplacement d'une piste

; ************************************
Custom  EQU $dff000

SysCop1 EQU $26
SysCop2 EQU $32

; ************************************
WIDTH   EQU 40      ; Largeur de l'écran (octets)
HEIGHT  EQU 40      ; Hauteur de l'écran (lignes)
BPLSIZE EQU WIDTH*HEIGHT    ; Taille du bitplan

MFMSIZE EQU 512*12+256  ; taille du buffer MFM

; ************************************
        rsreset
olddma  rs.w    1
oldint  rs.w    1
oldcop1 rs.l    1
oldcop2 rs.l    1
VARSIZE rs.w    0

        rsreset
head    rs.w    1       ; tête de lecture (0-1)
track   rs.w    1       ; piste (0-79)
essais  rs.w    1       ; essais en cas d'erreur (max 4)
DVARSIZE rs.w   0

; ************************************
CALL    MACRO
```

CHOUETTES (II)

```
    jsr _LVO1(a6)
    ENDM

; ************************************
    section READTRACK,CODE

Start   movea.l (_SysBase).w,a6
    lea VARS(pc),a5

    lea gfxname(pc),a1 ; Ouvre la graphics.library
    moveq   #0,d0      ; pour y puiser les adresses
    CALL    OpenLibrary; des 2 CopperLists système
    movea.l d0,a1
    move.l  SysCop1(a1),oldcop1(a5)
    move.l  SysCop2(a1),oldcop2(a5)
    CALL    CloseLibrary    ; Il faut la refermer !

    lea Custom,a6   ; c'est
    lea CIAA,rCIAA  ; son
    lea CIAB,rCIAB  ; destain !

    move.w  dmaconr(a6),d0 ; sauve DMACON
    ori.w   #$8200,d0
    move.w  d0,olddma(a5)

    move.w  intenar(a6),d0 ; sauve INTENA
    ori.w   #$c000,d0
    move.w  d0,oldint(a5)

    move.w  #$7fff,d0   ; et les efface tous les deux
    move.w  d0,dmacon(a6)
    move.w  d0,intena(a6)
    move.w  d0,intreq(a6)   ; ainsi que INTREQ

    lea NewCop,a0   ; Construit notre CopperList
    move.l  #Screen,d0 ; (adresse du bitplan)
    move.w  d0,CLPlan-NewCop+6(a0)
    swap    d0
    move.w  d0,CLPlan-NewCop+2(a0)
    move.l  a0,cop1lc(a6)   ; et l'active
    move.w  d0,copjmp1(a6)

    move.w  #$8390,dmacon(a6)
    move.w  #$1002,intena(a6)

    bsr.s   StartDrive ; démarre le moteur de DF0:

    moveq   #0,d1
    move.w  #$0F00,d2   ; couleur des 2 lignes si erreur
ReadAllDisk:
    btst    #6,ciapra(rCIAA)
    beq.s   Exit
    move.w  d1,d0       ; numéro de piste dans d0
    lea TRACK(pc),a0    ; adresse de lecture dans a0
    bsr ReadThisTrack   ; lecture !
    bne.s   Error       ; ça n'a pas marché...
    addq.w  #1,d1
    cmpi.w  #159,d1
    bls.s   ReadAllDisk ; boucle pour toutes les pistes

Exit    moveq   #$000F,d2   ; couleur des 2 lignes si OK
    btst    #6,ciapra(rCIAA)
    beq.s   Exit

Error   bsr.s   StopDrive   ; arrête le moteur de DF0:

    lea NewCop,a0   ; change la couleur des 2 lignes
    move.w  d2,CLCol1-NewCop+6(a0)
    move.w  d2,CLCol2-NewCop+6(a0)

WaitLMB btst    #6,ciapra(rCIAA)
    bne.s   WaitLMB     ; attend Mickey

    move.w  #$7fff,d0   ; remet le système dans
    move.w  d0,dmacon(a6)   ; son état normal
    move.w  d0,intena(a6)
    move.l  oldcop1(a5),cop1lc(a6)
    move.l  oldcop2(a5),cop2lc(a6)
    move.w  d0,copjmp1(a6)
    move.w  olddma(a5),dmacon(a6)
    move.w  oldint(a5),intena(a6)

    moveq   #0,d0       ; Retour au CLI sans code d'erreur
    rts

; ************************************
; Démarre le drive DF0: et place les têtes
; au dessus de la piste 0, face 0
StartDrive:
    bset    #DSKSEL0,ciaprb(rCIAB) ; unselect drive 0
    bclr    #DSKMTR,ciaprb(rCIAB)  ; motor on
    bclr    #DSKSEL0,ciaprb(rCIAB) ; select drive 0

.Wait   btst    #DSKRDY,ciapra(rCIAA)  ; teste DSKRDY
    bne.s   .Wait           ; faut encore attendre

    bset    #DSKDIR,ciaprb(rCIAB)  ; dir = -> piste 0
    bset    #DSKSIDE,ciaprb(rCIAB) ; face = lower (0)
.Seek0  btst    #DSKTRK0,ciapra(rCIAA) ; piste 0 atteinte ?
    beq.s   .Track0         ; oui
    bsr MoveHeads       ; sinon déplace les têtes
    bra.s   .Seek0          ; et boucle
.Track0 rts

; ************************************
; Arrêtre le drive DF0: (la LED s'éteint)
StopDrive:
    bset    #DSKSEL0,ciaprb(rCIAB) ; unselect drive 0
    bset    #DSKMTR,ciaprb(rCIAB)  ; motor off
    bset    #DSKMTR,ciaprb(rCIAB)  ; motor off
    bclr    #DSKSEL0,ciaprb(rCIAB) ; select drive 0
    rts

; ************************************
; Lecture de la piste D0 dans le buffer pointé par A0.
; Cette routine recherche la bonne piste (Seek), la lit
; et décode les données MFM.
ReadThisTrack:
    movem.l a0-a6/d1-d5,-(sp)
    lea DskVars(pc),a5      ; a5 = variables "locales"
    clr.w   essais(a5)      ; nb. essais = 0

    bsr SeekThisTrack       ; recherche la piste à lire

.Retry  bsr PrintStatus     ; affiche les infos
    move.w  #$4000,dsklen(a6)   ; efface DSKLEN
    move.l  #MFMBUF,dskpt(a6)   ; Adresse de lecture
    move.w  #$4489,dsksync(a6) ; Synchro MFM standard
    move.w  #$7f00,adkcon(a6)   ; Efface ADKCON
    move.w  #$9500,adkcon(a6)   ; Valeur correcte dans ADKCON
    move.w  #$8000|MFMSIZE,dsklen(a6) ; Longueur de lecture (mots)
    move.w  #$8000|MFMSIZE,dsklen(a6) ; écrite 2 fois
.Wait   move.w  intreqr(a6),d0      ; Lecture terminée ?
    andi.w  #$2,d0
    beq.s   .Wait           ; Pas encore
    move.w  #$2,intreq(a6)
    move.w  #$4000,dsklen(a6)   ; Efface DSKLEN

    bsr.s   MFMUncode       ; Décodage des données MFM
    beq.s   .ReadOk

.Error  addq.w  #1,essais(a5)       ; si une erreur de lecture
    andi.w  #3,essais(a5)       ; survient, on relit la même
    bne.s   .Retry          ; piste 4 fois au maximum
    moveq   #-1,d0          ; et on abandonne

.ReadOk movem.l (sp)+,a0-a6/d1-d5
    rts

DskVars dcb.b   DVARSIZE

; ************************************
; Cette routine décode les données MFM de MFMBUF (1 piste)
; dans le buffer pointé par A0.
; Retourne 0 si OK, -1 si erreur
MFMUncode:
    lea MFMBUF,a1   ; données MFM
    move.l  #$55555555,d2   ; masque bits impairs
    moveq   #10,d5      ; 11 secteurs à décoder
.GAP    cmpi.w  #$4489,(a1)+    ; cherche le début du secteur
    bne.s   .GAP
    cmpi.w  #$4489,(a1)
    beq.s   .GAP

    move.l  (a1)+,d0
    and.l   d2,d0
    lsl.l   #1,d0
    move.l  (a1)+,d1
    and.l   d2,d1
    or.l    d1,d0       ; d0=format,piste,secteur,nombre

    add.w   d0,d0
    andi.w  #$1E00,d0
    lea 0(a0,d0.w),a2   ; a2=secteur dans le track-buffer

    lea 36(a1),a1   ; saute les infos DOS
    move.l  (a1)+,d0    ; d0=header checksum
    moveq   #9,d3       ; 10 mots longs à vérifier
    lea -48(a1),a3  ; a3 pointe le header
.Check  move.l  (a3)+,d1
    eor.l   d1,d0       ; checksum par EOR
    dbra    d3,.Check
```

```
        and.l   d2,d0
        bne.s   .ReadError          ; foiré !

        addq.l  #4,a1
        move.l  (a1)+,d3  ; d3=data area checksum
        lea     512(a1),a4          ; a1=bits impairs, a4=bits pairs
        moveq   #127,d4             ; 128 mots longs à décoder
.UncodeSector:
        move.l  (a1)+,d0
        eor.l   d0,d3
        and.l   d2,d0
        lsl.l   #1,d0

        move.l  (a4)+,d1
        eor.l   d1,d3
        and.l   d2,d1

        or.l    d1,d0
        move.l  d0,(a2)+
        dbra    d4,.UncodeSector
        and.l   d2,d3
        bne.s   .ReadError          ; erreur !
        dbra    d5,.GAP             ; secteur suivant
        moveq   #0,d0               ; c'est OK
        rts

.ReadError:
        moveq   #-1,d0              ; indique une erreur de lecture
        rts

; ************************************
; Positionne les têtes de lecture/écriture au dessus
; de la piste désignée par D0.
; Ici, les pistes sont numérotées de 0 à 159. La face 0
; contient les pistes paires, la face 1, les pistes impaires
SeekThisTrack:
        moveq   #1,d2
        bset    #DSKSIDE,ciaprb(rCIAB)      ; face 0
        clr.w   head(a5)
        lsr.w   #1,d0
        bcc.s   .LowerSide
        bclr    #DSKSIDE,ciaprb(rCIAB)      ; face 1
        addq.w  #1,head(a5)

.LowerSide:
        move.w  d0,d1
        sub.w   track(a5),d0        ; Dans quelle direction aller ?
        beq.s   .SeekOk             ; Ben.. Nulle part, on y est
déjà !
        bpl.s   .SeekForward        ; Vers le sillon 79 (centre)

.SeekBackward:
        bset    #DSKDIR,ciaprb(rCIAB)       ; Vers le sillon 0
(extérieur)
        neg.w   d2
        bra.s   .SeekIt

.SeekForward:
        bclr    #DSKDIR,ciaprb(rCIAB)

.SeekIt bsr.s   MoveHeads ; Déplace les têtes d'1 piste
        add.w   d2,track(a5)        ; Inc/Dec le compteur de pistes
        cmp.w   track(a5),d1        ; Piste demandée atteinte ?
        bne.s   .SeekIt             ; pas encore...

.SeekOk rts

; ************************************
; Fournit l'impulsion STEP au drive
; suivie du nécessaire délai (soft... argh !)
MoveHeads:
        bset    #DSKSTEP,ciaprb(rCIAB)      ; step = high
        nop
        nop
        nop
        bclr    #DSKSTEP,ciaprb(rCIAB)      ; step = low
        nop
        nop
        nop
        bset    #DSKSTEP,ciaprb(rCIAB)      ; step = high

; Délai logiciel... !!!! ABSOLUMENT INTERDIT !!!!
SoftDelay:
        move.w  #4000,d0
        dbra    d0,*
        rts

; ************************************
; Affiche le numéro de la piste et de la face
; en cours de lecture, ainsi que le nombre
; d'essais
```

```
PrintStatus:
        movem.l a0-a3/a6/d1,-(sp)
        lea     .fmt(pc),a0
        lea     DskVars(pc),a1
        lea     .putch(pc),a2
        lea     .buf(pc),a3
        movea.l (_SysBase).w,a6
        CALL    RawDoFmt  ; Merci Exec...
        moveq   #0,d1
        lea     Screen,a1 ; pointeur sur le bitplan
.loop   lea     Numbers(pc),a0      ; données des chiffres
        move.b  (a3)+,d1
        beq.s   .ret
        subi.b  #" ",d1
        beq.s   .space
        subi.b  #"0"-" ",d1
        lsl.b   #3,d1
        lea     8(a0,d1.w),a0
.space  move.b  (a0)+,HEIGHT*0(a1)  ; Devpac optimisera ça...
        move.b  (a0)+,HEIGHT*1(a1)
        move.b  (a0)+,HEIGHT*2(a1)
        move.b  (a0)+,HEIGHT*3(a1)
        move.b  (a0)+,HEIGHT*4(a1)
        move.b  (a0)+,HEIGHT*5(a1)
        move.b  (a0)+,HEIGHT*6(a1)
        move.b  (a0)+,HEIGHT*7(a1)
        addq.l  #1,a1
        bra.s   .loop
.ret    movem.l (sp)+,a0-a3/a6/d1
        rts

.putch  move.b  d0,(a3)+
        rts

.fmt    dc.b    "%d:%d ;%d<  ",0
.buf    dc.l    0,0,0,0

; ************************************
Numbers DC.B    $00,$00,$00,$00,$00,$00,$00,$00      ; espace
        DC.B    $7C,$C6,$CE,$D6,$E6,$C6,$7C,$00      ; 0
        DC.B    $18,$38,$18,$18,$18,$18,$7E,$00      ; 1
        DC.B    $3C,$66,$06,$3C,$60,$66,$7E,$00      ; 2
        DC.B    $3C,$66,$06,$1C,$06,$66,$3C,$00      ; 3
        DC.B    $1C,$3C,$6C,$CC,$FE,$0C,$1E,$00      ; 4
        DC.B    $7E,$62,$60,$7C,$06,$66,$3C,$00      ; 5
        DC.B    $3C,$66,$60,$7C,$66,$66,$3C,$00      ; 6
        DC.B    $7E,$66,$06,$0C,$18,$18,$18,$00      ; 7
        DC.B    $3C,$66,$66,$3C,$66,$66,$3C,$00      ; 8
        DC.B    $3C,$66,$66,$3E,$06,$66,$3C,$00      ; 9
        DC.B    $06,$0C,$18,$30,$60,$C0,$80,$00      ; /
        DC.B    $0C,$18,$30,$30,$30,$18,$0C,$00      ; (
        DC.B    $30,$18,$0C,$0C,$0C,$18,$30,$00      ; )

; ************************************
VARS    dcb.b   VARSIZE,0
gfxname dc.b    "graphics.library",0
        even

; ************************************
TRACK   dcb.b   11*512

; ************************************
        section CHIP,DATA_C

NewCop  dc.w    diwstrt,$2c81,diwstop,$2cc1  ;    ecran    standard
320x200
        dc.w    ddfstrt,$0038,ddfstop,$00d0
        dc.w    bplcon0,$1200                          ; 1 bitplan
        dc.w    bplcon1,$0000,bplcon2,$0000
        dc.w    bpl1mod,$0000,bpl2mod,$0000
CLPlan  dc.w    bplpt,$0000,bplpt+2,$0000
CLCol1  dc.w    $2a0f,$fffe,color,$00F0
        dc.w    $2b0f,$fffe,color,$0000
CLCol2  dc.w    $530f,$fffe,color,$00F0,bplcon0,$0000
        dc.w    $540f,$fffe,color,$0000
        dc.l    -2

; ************************************
Screen  dcb.b   BPLSIZE
MFMBUF  dcb.w   MFMSIZE

; ************************************
        END
```

Par Loïc Far

SUB.Way II : Bougez avec Intuit

Notre exploration des bibliothèques graphiques de l'Amiga continue, avec ce mois-ci la gestion des sprites hardware en C. Pour illustrer le tout, nous allons réaliser un programme simple d'animation de sprites que vous pourrez à loisir modifier, ce dernier se prêtant à de nombreuses variations.

Ce programme va afficher une fenêtre et un sprite sur l'écran du Workbench, sprite qui se déplacera dans le sens inverse de celui de la souris. Ce sprite sera "accroché" à une fenêtre fixe, qu'il suffira de fermer pour terminer le programme. La démarche est très simple, il suffit :

- d'ouvrir les bibliothèques Intuition et Graphics ;
- de créer un tableau de données correspondant au sprite que l'on veut afficher et déplacer ;
- d'ouvrir une fenêtre initialisée de telle sorte qu'elle puisse nous renvoyer les coordonnées de la souris ;
- d'allouer un sprite hardware et de le déplacer.

Après avoir défini ce qu'il y avait à faire, passons sans plus attendre à la réalisation.

INITIALISATION DU CONTEXTE

Pour l'ouverture des bibliothèques Intuition et Graphics, c'est comme d'habitude ; il vous suffira pour les explications de vous reporter aux articles précédents de cette série. Le seul point important à souligner concerne les "includes". En effet, comme nous allons travailler avec les sprites, il est nécessaire d'inclure le fichier <graphics/sprite.h>, sans lequel nous ne pourrions rien faire.

La création du tableau représentant l'image du sprite n'est pas compliquée. Toutefois et avant d'en parler, il est nécessaire de donner quelques renseignements supplémentaires sur les sprites hardware.

LES SPRITES

Les sprites sont des éléments graphiques à part entière qui présentent la particularité d'être indépendants de l'écran sur lequel ils évoluent. Chaque sprite peut avoir, au maximum, une largeur de 16 points (basse résolution) pour une hauteur maximale égale à celle de l'écran sur lequel il se déplace. Cette restriction à 16 points de largeur est due au fait que les sprites sont toujours affichés en basse résolution. Pour vous en persuader, il vous suffit de regarder attentivement le pointeur de souris de votre Workbench : celui-ci est bien en basse résolution alors que le Workbench est en Hires (640 pixels).

Sur l'Amiga, il est possible d'avoir au maximum 8 sprites, numérotés de 0 à 7. Intuition utilise toujours le sprite numéro zéro pour représenter le pointeur de la souris, celui-ci ne pouvant jamais être inhibé (au contraire des autres). Chaque sprite peut être affiché en 3 couleurs effectives, la quatrième étant transparente. Les sprites sont associés par paire : 0 et 1, 2 et 3, 4 et 5, 6 et 7. A chaque paire de sprites correspondent 3 registres de couleur effectifs. Le tableau ci-dessous résume cet état de fait.

Sprites	Pixel	Registre de couleur
0&1	00	transparent
	01	COLOR 17
	10	COLOR 18
	11	COLOR 19
2&3	00	transparent
	01	COLOR 21
	10	COLOR 22
	11	COLOR 23
4&5	00	transparent
	01	COLOR 25
	10	COLOR 26
	11	COLOR 27
6&7	00	transparent
	01	COLOR 29
	10	COLOR 30
	11	COLOR 31

Comme nous l'avons vu, notre sprite mesure 16 pixels de large. A chacun de ces pixels vont être associés deux bits (4 couleurs par sprite). Pour créer l'image de notre sprite, nous allons donc définir un tableau de données similaire à une mini-Bitmap, en associant à chaque ligne du sprite, deux mots de 16 bits.

On représente cette ligne de pixels de la manière suivante :

```
15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0   (n° des pixels)
 1  1  1  1  0  0  0  0  1  1  1  1  0  0  0  0   (premier mot)
 1  1  1  1  1  1  1  1  0  0  0  0  0  0  0  0   (deuxième mot)
```

En combinant verticalement (du haut vers le bas) les bits des deux mots définissant la ligne de pixels, on retrouve la couleur désirée. Dans notre exemple, les pixels numérotés 0 à 3 seront transparents (image binaire 00), 4 à 7 de la première couleur (01), 8 à 12 de la seconde couleur (10) et 12 à 15, de la troisième couleur (11). Si nous avions réservé le sprite 6, les couleurs 1, 2 et 3 seraient représentées par les registres COLOR 29, 30 et 31.

Comme vous pouvez le remarquer, le définition de l'image d'un sprite, sans être complexe, n'en est pas moins fastidieuse. Vous imaginez sans peine la somme de travail nécessaire à la réalisation d'un sprite animé, puisque pour chaque image de l'animation, il va falloir remplir de cette manière un tableau différent. Pour vous éviter celà, je vous propose d'utiliser un petit logiciel du domaine public qui effectue seul une bonne partie du travail. Il s'agit de GetSprite, qui se trouve sur la disquette Fish 161 que vous pourrez trouver chez votre poissonnier favori. Les heureux abonnés avec disquette le trouveront évidemment dans même le tiroir que le programme de cet article.

CREATION ET GESTION DU SPRITE

Après la théorie, la pratique. Pour créer un sprite dans notre programme, il faut tout d'abord le déclarer sous la forme d'une structure SimpleSprite :

```
struct SimpleSprite
{   UWORD *posctldata; /* données du sprite */
    UWORD height;  /* hauteur en nombre de lignes */
    UWORD x,y;  /* position (X et Y) */
    UWORD num;  /* numéro du sprite hardware */
};
```

Le tableau de données pointé par le champ posctldata se décompose comme suit :

```
struct userdata
{   UWORD posctl[2];  /* position et information de contrôle */
    UWORD sprdata[2*height];     /* image actuelle du sprite */
    UWORD reserved[2];    /* sprite suivant */
};
```

Il est à noter que les deux mots du tableau posctl gérant le contrôle de position du sprite seront initialisés par défaut à zéro et mis à jour automatiquement lors de l'utilisation des fonctions

ion et Graphics

GetSprite() et MoveSprite().

Une fois cette structure initialisée correctement (cf. exemple), il est désormais possible de se réserver un sprite grâce à la fonction GetSprite() dont voici la syntaxe commentée.

```
GetSprite(simplesprite, numero);
```

- simplesprite est un pointeur sur la structure SimpleSprite que nous venons de créer ;
- numero correspond au numéro du sprite hardware que l'on souhaite réserver. Si l'on indique -1, c'est le premier sprite libre qui sera choisi. Par "libre", on entend "non réservé par une autre application". Le sprite 0 n'est donc jamais disponible.

Cette fonction retourne un SHORT correspondant au numéro du sprite alloué, ou -1 si l'allocation à échoué.

Le sprite ainsi créé, nous pouvons maintenant le déplacer. La fonction MoveSprite() se charge de positionner le sprite aux coordonnées spécifiées. Sa syntaxe est la suivante :

```
MoveSprite(viewport, sprite, x, y);
```

- viewport est un pointeur sur le ViewPort de l'écran sur lequel le sprite évolue ;
- sprite est un pointeur sur le sprite à déplacer ;
- x et y réprésentent ses nouvelles coordonnées.

Le programme terminé, il est nécessaire de libérer le sprite pour d'autres applications, à l'aide de la fonction FreeSprite() dont la syntaxe est des plus simples puisqu'il suffit de lui passer le numéro du sprite à libérer :

```
FreeSprite(numero);
```

numero étant de type SHORT.

Enfin, la dernière fonction que nous allons voir s'appelle ChangeSprite(). Bien que non utilisée dans notre, exemple elle pourra vous être utile puisqu'elle permet de modifier l'image du sprite. Sa syntaxe est :

```
ChangeSprite(viewport, sprite, data_sprite);
```

- viewport est un pointeur sur le ViewPort de l'écran associé au sprite ;
- sprite est un pointeur sur le sprite à modifier ;
- data_sprite est un pointeur sur la structure userdata contenant la nouvelle image du sprite.

Voilà pour les sprites. Je ne puis que vous conseiller d'examiner attentivement le programme ci-dessous si le moindre doute subsiste dans votre esprit. Le système d'animation graphique de l'Amiga peut sembler plutôt compliqué à priori, mais avec un peu de pratique, on se rend vite compte à quel point il est puissant. Nous verrons le mois prochain une autre facette de ce système : les Bobs.

```
/* ------------------------------------------------------------------ */
/*                                                                    */
/*  Programme de gestion d'un sprite sous workbench                   */
/*                                                                    */
/*  Exemple de programmation des sprites sous Intuition & Graphics    */
/*          - Sprites                                                 */
/*          - Gestion de la souris                                    */
/*          - Création de structures Image                            */
/*                                  P. AMIABLE 1991                   */
/*                                                                    */
/* ------------------------------------------------------------------ */

/* ------------------------------------------------------------------*/
/*                                                                   */
/*     Quelques includes utiles.....                                 */
/*                                                                   */
/* ------------------------------------------------------------------*/

#include <exec/types.h>
#include <intuition/intuition.h>
#include <stdio.h>
#include <graphics/sprite.h>

/* ------------------------------------------------------------------*/
/*                                                                   */
/*     Les defines maintenants.....                                  */
/*                                                                   */
/* ------------------------------------------------------------------*/

#define INTUITION_REV 0L /* Version d'Intuition (la dernière) */
#define GFX_REV       0L /* Version de Graphics (la dernière) */
#define ROUGE   0xf,0x0,0x0
#define VERT 0x0,0xf,0x0
#define BLEU  0x0,0x0,0xf
#define LARGEUR 640 /* mettre 640 en hi-res et 320 en lo-res */
#define HAUTEUR 256 /* mettre 256 en non entrelacé PAL (200 en NTSC) */
                                        /*  le  double  512  (400)  en
entrelacé */

/* ------------------------------------------------------------------*/
/*                                                                   */
/*    Déclaration des données pour le sprite.                        */
/*                                                                   */
/* ------------------------------------------------------------------*/

UWORD chip sprite[36]=     /* donnée nécessaire à décrire l'image du
sprite */
{
0x0000, 0x0000,

0x0000,0x3000,
0x0000,0x7800,
0x0000,0xcc00,
0x0000,0xcc00,
0x0000,0xcc00,
0x0460,0xfc60,
0x067e,0xfe00,
0x077e,0xcf00,
0x07f8,0xcfe0,
0x06f8,0xcee0,
0x0678,0xce60,
0x0678,0x0660,
0x0678,0x0660,
0x0018,0x0000,
0x0000,0x0000,
0x0000,0x0000,

0x0000, 0x0000
};

struct SimpleSprite pointeur1 =
{
sprite,         /* pointeur sur les données image du sprite. */
16,             /* hauteur, la largeur est fixée à 16 pixels */
0, 0,           /* position en x, y à l'écran. */
-1,             /* numéro du sprite, initialisé par  GetSprite() */
/* on l'initialise à -1 pour le moment */
};

/* ------------------------------------------------------------------*/
/*                                                                   */
/*     Définition des variables externes                             */
/*                                                                   */
/* ------------------------------------------------------------------*/

/* Declararation des fonctions définies dans le programme */

void main();
void referme();
void init();
void cree_sprite();

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

struct Window *fenetre = NULL;
```

```
struct NewWindow nouvfenetre =
{
0,            /* LeftEdge    position en x de la fenêtre */
0,            /* TopEdge     position en y de la fenêtre */
150,           /* Width       150 pixels de large */
15,            /* Height      15 pixels de hauts */
0,             /* DetailPen   Le texte est de la couleur 0 */
1,             /* BlockPen    les blocks sont de la couleur 1 */
CLOSEWINDOW|   /* IDCMPFlags  envoie message si l'utilisateur */
MOUSEMOVE,     /*              a sélectionné le gadget de fermeture ou
*/
/*             si il a déplacé la souris */
SMART_REFRESH| /* Flags       Rafraichissement réalisé par Intuition */
WINDOWCLOSE|   /*              Gadget de fermeture */
WINDOWDEPTH|   /*              Gadget de profondeur */
ACTIVATE|      /*              Fenêtre active lorsqu'elle est ouverte */
REPORTMOUSE,   /*              Suivi de la souris */
NULL,          /* FirstGadget Pas de gadgets personnels */
NULL,          /* CheckMark   Pas de Checkmark particulière */
"SPRITE",      /* Title       Titre de la fenêtre  */
NULL,          /* Screen      Ecran du Workbench */
NULL,          /* BitMap      pas de Bimpap personnelle */
150,           /* MinWidth    Taille minimale : 150x15 */
15,            /* MinHeight   Taille maximale : 150x15 */
150,           /* MaxWidth    reste comme elle est */
15,            /* MaxHeight */
WBENCHSCREEN   /* Type        On accroche la fenêtre au Workbench */
};

struct  RastPort *rastport = NULL;
struct ViewPort *viewport  = NULL;


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

void main()
{

struct IntuiMessage *message;
long GetMsg();
ULONG classemessage;
int fin = 0;
void init(),ouvrefenetre(),referme();
int xmouse,ymouse;
WORD xsprite1,ysprite1;

init();
ouvrefenetre();
cree_sprite();
while(!fin) /* boucle tant que le gadget de fermeture n'est pas actif
*/
{
Wait(1L << fenetre->UserPort->mp_SigBit); /* attente d'un signal */
while(message=(struct  IntuiMessage  *)GetMsg(fenetre->UserPort))  /*  ce
sont des messages */
{
classemessage = message->Class; /* on recupere la classe */
switch(classemessage) {

case CLOSEWINDOW: /* on a fermé la fenêtre */
fin = 1;
break;

case MOUSEMOVE: /* on deplace la souris */
xmouse = message->MouseX;
ymouse = message->MouseY;

/* on déplace le sprite en sens inverse */

xsprite1 = LARGEUR - 32 - xmouse;
ysprite1 = HAUTEUR - 32 - ymouse;

MoveSprite( viewport, &pointeur1, xsprite1, ysprite1 );

WaitTOF(); /* Attente de trame afin de voir le sprite correctement */
break;

default: break;
}
ReplyMsg((struct Message *)message); /* libere le message */
}
}
referme(); /* au revoir */
}
```

```
/* ------------------------------------------------------------------*/
/*     Init() Ouvre la bibliothèque                                  */
/* ------------------------------------------------------------------*/

void init()
{
char *OpenLibrary();

IntuitionBase              =              (struct              IntuitionBase
*)OpenLibrary("intuition.library",INTUITION_REV);
if(!IntuitionBase)
{
exit(FALSE);
}
GfxBase = (struct GfxBase *)OpenLibrary("graphics.library",GFX_REV);
if(!GfxBase)
{
referme();
exit(FALSE);
}
}



/* ------------------------------------------------------------------*/
/*     ouvrefenetre() ouvre la fenetre et les gadgets                */
/* ------------------------------------------------------------------*/

void ouvrefenetre()
{
void referme();

if(!(fenetre=(struct Window*)OpenWindow(&nouvfenetre)))
{
referme();
exit(FALSE);
}

rastport = fenetre->RPort; /* rastport nécessaire pour tracer */
viewport  =  (struct  ViewPort  *)ViewPortAddress(fenetre);  /*  viewport
nécessaire pour la gestion des couleurs */
}



/* ------------------------------------------------------------------*/
/*     Cette fonction referme la fenetre et les bibliothèques        */
/* ------------------------------------------------------------------*/

void referme()
{
if( pointeur1.num != -1 )
FreeSprite( pointeur1.num );

if(fenetre)       CloseWindow(fenetre);
if(IntuitionBase) CloseLibrary(IntuitionBase);
if(GfxBase) CloseLibrary(IntuitionBase);
}


/* ------------------------------------------------------------------*/
/*     Cree_Sprite() crée le deuxième sprite                         */
/* ------------------------------------------------------------------*/

void cree_sprite()

{

/*  Première  étape  on  initialise  les  registres  21,22,23  qui
correspondent à la couleur du sprite */

SetRGB4( viewport, 21, ROUGE ); /* C.F le define plus haut */
SetRGB4( viewport, 22, VERT );
SetRGB4( viewport, 23, BLEU );

if( GetSprite( &pointeur1, 2 ) != 2 )
referme(); /* Le sprite numéro deux est indisponible, on sort. */

}
```

Par Herr Doktor Von Glupa

REQUESTER

La polémique s'étend : pour ou contre l'ANT sans listings, ceux-ci étant disponibles uniquement sur disquette (ce qui entrainerait donc la suppression des formules d'abonnements sans disquette) ? Notre requester de ce mois-ci est presque exclusivement consacré à vos réactions.

Je suis passionné par l'informatique depuis l'âge de 11 ans (j'en ai 27), âge où mon lycée a été équipé de matériel informatique. Par la suite, j'ai eu un Commodore 64 puis un compatible IBM PC. J'ai bradé ce dernier pour acheter une machine plus évoluée : multitâche digne de ce nom (et toc Windows), une interface utilisateur intuitive (!) et une interface développeur style MS-DOS permettant de faire tout ce qu'un programmeur doit faire (et toc le Mac !) : j'ai nommé l'Amiga.
En parallèle, j'ai transformé cette passion en profession et je suis aujourd'hui consultant en informatique. Ceci pour vous dire que les jeux ne m'intéressent que de très loin et que je trouve donc votre revue d'un excellent rapport qualité/prix. En effet, 32 pages d'informations écrites en condensé me redonnent enfin un peu d'espoir quant à la profession de journaliste. Ce qui m'intéresse : des trucs et astuces, des bidouilles, des standards (bien vu CATS), des descriptions de langages (C, assembleur, ARexx, etc.).
Quant à votre proposition de transférer tout ce qui est listing sur disquette, voici ma réponse. Des informations sans exemples sont fatigantes et vite mises de côté, dans la corbeille "pouvant servir un jour". Mais si, vous savez, le carton qui est régulièrement descendu à la cave en contient plein ! Par contre, des listings de 10 km de long comme dans votre numéro de juin sont inutiles. Je ne tape plus de listings depuis le deuxième numéro d'Hebdogiciel. Bien sûr, ils sont une foison de renseignements, mais pas toujours exploitables. Sur disque, 2 solutions : on les lit à l'écran - personnellement, je trouve celà épouvantable ; on n'a jamais une vision globale du programme pour comprendre sa logique. On peut également les imprimer (et encore, à condition d'avoir une imprimante !) mais il est toujours rageant de sortir plusieurs pages de listing pour trouver les 3 lignes réellement intéressantes.
En résumé :
- trop de listings dans l'ANT : je stoppe mon abonnement ;
- des listings sur une disquette à payer 30 F (j'achète les miennes 2,32 F) avec uniquement du bla-bla théorique dans la revue : je stoppe également ;
- une revue sans disquette et sans exemples concrêts : même chose, je stoppe.
Conclusion : votre revue est très bien ainsi, ne changez rien.

Frédéric Ribadeau-Dumas, Thonon-les-Bains

Tout d'abord, merci pour vos compliments. Je tiens tout de même à préciser que nous nous considérons moins comme des journalistes (malgré les titres ronflants de "rédacteur en chef" qui se baladent dans l'Ours) que comme des programmeurs. Le journalisme est un métier particulier, qui demande certainement beaucoup plus de qualités que nous n'avons. C'est là toute la différence entre un "journal" comme SVM ou l'Ordinateur Individuel et un "canard" comme l'ANT. Qui a dit "feuille de choux" ?
Quant à votre réflexion sur le bien-fondé d'un ANT sans listings, elle est plus que pertinente. Bien sûr, des explications sans démonstration se révèlent vite rébarbatives. Bien sûr, nous ne pouvons pas non plus nous permettre de trop gros programmes, qui occuperaient plus de place que les articles. A ce sujet, votre remarque sur le listing "la 3D faces pleines" (ANT numéro 24) est sans doute fondée. Mais même si le sujet ne vous intéresse pas outre-mesure, qu'aurions-nous pu faire d'autre ? Soit nous publiions d'abord le listing puis les explications comme nous avons décidé de le faire, soit les deux en même temps... mais en deux fois, n'offrant ainsi aux lecteurs intéressés qu'une moitié de programme qui ne leur aurait de toute façon servi à rien ! Le problème ne se serait même pas posé si la disquette accompagnait automatiquement le magazine...

Je profite de mon ré-abonnement pour répondre à votre mini-sondage.
1) oui, bien sûr, je suis très intéressé par les Rom Kernal Manuals en français ;
2) comme vous pouvez le constater, j'ai opté pour la solution 11 numéros sans disquette. En effet, depuis que je suis abonné à l'ANT, je n'ai jamais utilisé les sources fournis avec la disquette. Par contre, leur présence au sein d'un article peut être une précieuse source de renseignements, accessibles en un coup d'oeil. Bien sûr, c'est au détriment de la place pour d'autres articles. Une solution serait de pouvoir acheter séparément les sources qui nous intéressent, par exemple en téléchargement. Personnellement, je préférerais conserver la formule actuelle, tout en étant conscient du gâchis de place dû aux listings.

Laurent Bouvot, Marcoussis

Acheter les listings séparément, même en téléchargement sur le 3615 ANT, n'est pas forcément une bonne idée. Celà entrainerait des procédures supplémentaires pour l'abonné, lourdes à gérer pour nous. Or, comme certains d'entre vous ont déjà eu le malheur de s'en apercevoir, la gestion n'est pas vraiment notre fort...
Par contre, il n'est pas impossible que nous mettions en téléchargement les listings parus dans les anciens ANT (du temps où il était encore inclus dans Commodore Revue). Les abonnés à la disquette les trouveraient tout naturellement sur celle-ci. Cela suppose simplement que nous arrivions à remettre la main dessus...

Salut l'ANT ! C'est la première fois que je vous écris depuis que le vous lis (n°14 de Commodore Revue), c'est-à-dire bientôt deux ans. Si j'ai attendu si longtemps, c'est qu'il n'y avait jusqu'alors aucun problème particulier. Mais cette fois-ci, il y en a un réel : vous nous proposez de nous abonner à la disquette pour qu'il y ait plus de place dans l'ANT pour les articles, et ceci en prétextant que vous n'avez pas assez d'argent pour rajouter des pages. Alors je sais que je ne suis pas bon en économie, ce n'est pas mon domaine, mais il y a quelques questions qui trottent dans ma tête.
- Tout d'abord, comment pouvez-vous manquer d'argent en vendant une revue, il faut bien dire pas très épaisse, à un tel prix ? De plus, l'argent n'est certainement pas dépensé dans les frais de port puisque jusqu'à présent, je n'ai jamais reçu mon ANT avant le 23 du mois. Ce mois-ci, je l'ai reçu le 25.
- Ensuite, Amiga Revue que j'achète chaque mois et à laquelle je ne suis pas abonné est exactement trois fois plus épaisse (96 pages contre 32 dans l'ANT) et est vendue au même prix. Mon étonnement est causé par le fait qu'Amiga Revue utilise un matériel et des logiciels autrement plus chers que ceux de l'ANT (Sculpt Animate 4D, Turbo Silver, etc. ainsi que des configurations à base de cartes accélératrices et un bon paquet de RAMs 32 bits) et que malgré ceci, Amiga Revue ne s'est jamais plainte de problèmes financiers. Si ceci est dû à la pub, pourquoi pas dans l'ANT ? On n'est pas à quelques annonceurs près !
- Enfin et pour terminer, je trouve les disquettes de l'ANT franchement chères. Autant qu'un numéro, c'est-à-dire 30 F, ce n'est pas un tarif qui risque de faire concurrence aux associations

de domaine public, qui facturent au maximum 20 F par disque et qui livrent autrement plus rapidement, sans supplément de prix. Je pense en particulier à l'ex-Free Distribution, devenue FDS, mais j'aurais tout aussi bien pu citer PDS Free Line ou la toute dernière, D2P. Ne me dites pas que le prix des disquettes inclue le salaire des programmeurs, je pense que le prix de la revue elle-même s'en charge. Je pense que les disquettes de l'ANT devraient être vendues à leur coût de fabrication, vos bénéfices étant alors apportés par la revue.
Voilà, j'espère que vous ne m'en voudrez pas trop de cette petite mise au point. Pour répondre maintenant à la question de Stéphane sur les cartes passerelles XT et AT, je sais seulement qu'elles se logent à partir du 6ème Mo de RAM, d'où un problème de compatibilité avec les grosses configurations, sauf si l'on a la chance de posséder une carte Combo (only GVP makes it possible!).
Merci enfin à F. Mazué et Max qui sont mes deux programmeurs préférés, mes références. Sans eux, qui sait, peut-être ne serais-je qu'un mauvais programmeur sur Sekaka ? Au fait, juste une dernière question : existe-t-il une version de Devpac pour 68030/68882, en prévision de ma future carte accélératrice ?

Cédric Souchon, Monistrol sur Loire

Celà peut sembler un peu gros à priori, mais c'est pourtant vrai : le magazine ANT est vendu non pas à perte (c'est tout de même interdit par la loi !) mais au plus juste prix. Nos seuls bénéfices se font avec les disquettes. Voici pourquoi.
Primo, nous achetons du papier à un imprimeur. Vue la petite quantité (je dis "petite" par rapport à, par exemple, Amiga Revue, mais celà se chiffre tout de même en plusieures tonnes), il ne peut nous faire de prix réellement concurentiel. Vous avez pu le constater, le papier choisi n'est pas très épais, mais il est beau. C'était le meilleur compromis qualité/prix que nous ayons trouvé. Bref, la tonne de papier n'est pas donnée, de nos jours.
Secundo, ce même imprimeur... imprime le journal. Encore une fois, vu le nombre d'exemplaires imprimés chaque mois (3500 ce mois-ci), il ne peut pas descendre ses prix sans fonctionner à perte (ce qu'il n'a évidemment ni l'intention, ni le droit de faire).
Tertio, des pigistes écrivent les articles que vous lisez. Il ne le font évidemment pas pour la seule gloire. Nous payons pourtant une misère : 500 F la page publiée. Comptez avec moi : 30 x 500 = 15000 F à sortir tous les mois pour eux. Sans compter les charges salariales, qui augmentent ce chiffre de 50%.
Quattro, votre serviteur et un maquettiste, permanents ceux-là, aiment bien recevoir un chèque de salaire à la fin de chaque mois. Les PTT et l'EDF aiment bien également que l'on règle leurs factures. Sans parler du flasheur, qui produit des films à partir de nos fichiers PAO (2000 F environ), ou du routeur, qui gère les abonnements, duplique les disquettes et expédie les journaux.
Vous vous en doutez, celà fait des frais fixes chaque mois, qu'il faut assumer. D'autant plus que l'ANT ne disposant pas de pages de publicité (une page de pub égale un article en moins...), seuls vos abonnements les couvrent tant bien que mal. C'est vrai que l'ANT est vendu au plus juste prix. Ce sont les disquettes qui assurent la majorité de nos bénéfices.
Voilà, vous savez tout. Merci pour les compliments à l'égard de Frédéric Mazué et de Max, ils seront bien entendu transmis. Merci également pour les renseignements sur la carte passerelle. Quant à une version 68030 de Devpac, les rumeurs les plus folles courent en Angleterre sur une nouvelle version de ce must des assembleurs. Nous en saurons plus dans quelques temps.

Salut à toute l'équipe ! Je vous écris pour vous demander de bien vouloir faire paraître cette petite annonce :
Echange toute série de DP, démos, musidisks, etc. Envoyez vos listes de DP à David Gaussinel, 18 rue Fenelon, 24200 Sarlat. Merci d'avance !

Voilà qui est fait !

Bonjour à toute la vaillante équipe. Je vous écris bien sûr pour vous poser deux ou trois questions sur des problèmes qui me gâchent ma vie de programmeur. Voici de quoi il retourne.
Je viens de faire l'acquisition d'un compilateur C du domaine public, le Sozobon C (Fish n°314), des fichiers Includes et de l'amiga.lib, grâce à un ami qui possède la Lattice. Je rencontre avec Zc le même problème qu'avec DICE, fourni sur votre disquette du numéro 21 : le compilateur produit un code source assembleur, qu'il faut passer à a68K ou a Dasm respectivement pour obtenir un code exécutable, ce qui augmente considérablement les temps de développement (surtout sur mon A500 sans disque dur !). Ma question est donc la suivante : existe-t-il dans le domaine public un compilateur C produisant directement du code exécutable, sinon comment modifier Zc (les sources sont fournis) dans ce but ?
Deuxièmement, le programme SuperShell de F. Mazué (ANT 21) fonctionne très bien, mais je me demande s'il est également possible d'ajouter des gadgets dans la barre de titre de la fenêtre du Shell, pour gagner encore plus de temps ?
Enfin, j'ai entendu parler d'un programme du DP (CliMax, je crois) qui permet d'ouvrir des fenêtres Shell sur un Custom Screen plutôt que sur l'écran du Workbench. Inconvénient, il requiert ConMan pour fonctionner. Pourriez-vous me donner un moyen de le satisfaire tout de même ?

François Richet, Lens

Concernant DICE, aucune modification n'est possible, le source du compilateur n'étant pas fourni. Quant à Zc, il faut reconnaître qu'une telle modification n'est pas aisée ; pour être franc, je n'ai aucune idée de la manière dont fonctionne un compilateur C, alors... Ecrivez à Matt Dillon, peut-être que...
SuperShell est facilement modifiable pour greffer aussi quelques gadgets à la fenêtre Shell qu'il utilise. La procédure est exactement la même que pour le menu, il faut simplement utiliser AddGadget() au lieu de SetMenuStrip().
Enfin, CliMax utilise quelques fonctionnalités de ConMan, dont le Con-Handler standard de l'Amiga ne dispose pas. Il ne peut donc pas s'en passer. Mais cherchez bien, d'autres DP réalisent la même chose, sans ConMan. Je pense plus particulièrement à MWB de Matt Dillon (encore lui !), présent sur la disquette Fred Fish 65.

Bon, il est grand temps de mettre fin à cette polémique, Victor : nous avons décidé, après mûre réflexion et des dizaines de lettres, de ne pas modifier les formules d'abonnement. L'ANT reste tel qu'il est pour l'instant. Toutefois, une autre idée est en cours d'évaluation, qui celle-là, devrait satisfaire tout le monde. Nous aurons l'occasion d'y revenir plus tard. Pour l'heure, je voudrais juste signaler à ses fans que Frédéric Mazué à décidé de nous quitter temporairement ; vous avez pu vous en rendre compte depuis le numéro précédent, sa verve inimitable ne viendra plus égayer ces mornes pages pendant quelques temps. Frédéric laisse tomber l'informatique après de longues années de bons et loyaux services, pour se consacrer entièrement à une autre passion qui l'anime depuis déjà longtemps. Nous lui souhaitons de tout coeur une bonne chance.

CATS : LE MULTITACHE SUR AMIGA

L'article qui est la traduction d'un article de Leo L. Schwab, préalablement posté sur le Programmer's Network, rubrique Amiga Conference, du BBS WELL. Leo Schwab est l'auteur de nombreux programmes du domaine public (Fred Fish lui doit beaucoup !) et l'un des programmeurs Amiga les plus prolifiques. On lui doit notamment l'Animation Studio de Walt Disney Software.

John Drapper disait qu'un moyen de devenir célèbre était d'écrire des manuels destinés à aider les programmeurs débutants à mieux comprendre leur système. Ceci est donc ma tentative de démonstration des principes de base du multitâche sur l'Amiga. Ce manuel décrira le multitâche au niveau d'Exec, sans parler du tout du DOS. Personnellement, je préfère éviter le DOS autant que possible pour me concentrer sur Exec. Cela n'engendre aucune difficulté supplémentaire, bien au contraire ; en fait, cela évite pas mal de maux de tête... Ce manuel sera principalement pratique, et assumera que le lecteur a déjà quelques connaissances de bases sur le principe du multitâche. Son but est principalement de vous aider à créer plusieurs tâches concurrentes et complémentaitres.

Alors prenez votre compilateur favori et suivez le guide...

INTRODUCTION

La première chose à faire lorsque l'on s'intéresse au système de l'Amiga est d'oublier tout ce que l'on peut déjà savoir sur d'autres systèmes multitâches. Particulièrement UNIX. Exec sur Amiga n'a rien à voir avec la plupart des autres systèmes multitâches, à l'exception peut-être de XINU (il existe un excellent livre sur XINU, dont beaucoup de concepts peuvent s'appliquer à Exec).

Exec est à mon sens un fourre-tout du multitâche. Vous y trouverez tout ce qu'il vous faut pour faire de dont vous avez besoin, mais pas tout ce que vous voulez. Par exemple, il ne libérera pas toutes les ressources que vous avez ouvertes, et sa manière de gérer les programmes "fous" n'est pas vraiment celle que l'on est en droit d'espérer.

En revange, Exec est petit et compact. Il est écrit en langage-machine extrèmement optimisé. Parce qu'il est petit et rapide et qu'il ne fait rien sans qu'on le lui demande, Exec est un système d'exploitation temps-réel capable de réagir aux interruptions et aux signaux très efficacement.
Donc, en échange de devoir tout faire par soi-même, on a un système multitâche très performant.

POUR COMMENCER

Une tâche, au sens Exec du terme, est divisée en deux parties : du code résident quelque part en mémoire et un bloc de contrôle faisant partie d'une liste chaînée de toutes les tâches gérées par Exec.
Le segment de programme peut être n'importe quoi n'importe où ; un moyen bête d'en créer un est tout simplement de déclarer un fonction C :

```
...
sous_tache()
{
        ...
}
...
```

Le bloc de contrôle de la tâche est une zone de mémoire qui décrit la tâche pour Exec. En particulier, il renseigne sur la position et la taille de votre pile, quels signaux votre tâche a reçus et lesquels elle attend, etc. Chaque tâche possède son propre bloc de contrôle.
Additionnellement, toutes les tâches ont besoin d'une pile. Même si vous n'appelez aucune sous-routine et ne disposez pas de variables locales, une pile est nécessaire à Exec pour sauvegarder les registres du processeurs en cas de changement de contexte (c'est-à-dire d'arrêt de votre tâche au profit d'une autre, N.d.T). La taille minimum recquise pour la pile est de 70 octets (cf. RKM vol. 1, p 1-17, 1-18). Un chiffre rond et sûr est de 1 Ko. Personnellement, je préfère utiliser des piles d'au moins 2 Ko pour mes programmes, juste pour être sûr d'avoir assez d'espace pour tous les appels de sous-routines que je peux effectuer et pour toutes les variables locales. La pile peut être dynamiquement allouée avec AllocMem(). Il est recommandé, mais non nécessaire, de ne pas préciser MEMF_PUBLIC dans ce cas.

DONNER VIE A UNE TACHE

La bibliothèque de support d'Exec contient une fonction très pratique de création de tâche, baptisée (assez justement) CreateTask().
CreateTask() ne réalise que les opérations de base de l'initialisation de la tâche. Elle alloue d'abord de la mémoire pour la pile de la nouvelle tâche et son bloc de contrôle, puis initialise les champs tc_SPUpper, tc_SPLower et tc_SPReg de cette structure, ainsi que tc_Node.ln_Pri, tc_Node.ln_Type et tc_Node.ln_Name, avant d'appeler AddTask() et de vous retourner un pointeur sur le bloc de contrôle nouvellement créé. Si la moindre erreur survient, CreateTask() retourne un pointeur NULL et n'alloue aucune mémoire.
Le format d'appel de CreateTask() est le suivant :

```
struct Task *CreateTask(nom, priorité, initPC, taille_pile)
char *nom;
UBYTE priorité;
APTR initPC;
ULONG taille_pile;
```

"nom" est une chaine de caractère qui définit le nom de votre tâche ; il sera utilisé si une autre tâche désire trouver la vôtre.

"priorité" représente la priorité de votre tâche par rapport aux autres présentes dans le système. Il s'agit d'un octet signé (de -128 à + 127). 0 est la priorité normale.

"initPC" est la valeur initiale du PC. Généralement, ce sera un pointeur sur une fonction, ou quelque chose dans ce genre.

"taille_pile" définit à la fois la taille de votre pile et de votre bloc de contrôle. CreateTask() alloue "taille_pile" octets de mémoire et utilise les sizeof(struct Task) premiers pour le bloc de contrôle, le reste étant effectivement utilisé en tant que pile pour votre tâche. Ceci est important à savoir si vous désirez une pile d'une taille bien particulière.

Le code source de CreateTask() se trouve dans le RKM vol. 2, quelques pages après la page E-78, dans l'appendice F.
Un appel typique à CreateTask() ressemble à ceci :

```
...
struct Task *tsk;
extern void fonction();

        if (!(tsk = CreateTask("tâche", 0, fonction, 2048)))
                printf("Impossible de créer la tâche.");
...
```

Notez bien que CreateTask n'éffectue que l'initialisation de base de la tâche. Si vous désirez effectuer des opérations spéciales, comme le traitement d'exceptions ou des interruptions, CreateTask() ne suffit plus, vous devrez créer votre propre CreateTask(). Le code source du RKM fournit un bon exemple de la manière de s'y prendre.

(à suivre...)