§1. Instructions

pc -g -o ex5 ex5.pas	ligne de compilation normale
source ~gennart/.pascal	ligne de compilation avec #include "Graphics.h". Une seule fois par session
pc -I$GINC -g -o ex5 ex5.pas -l$GLIB	ligne de compilation avec #include "Graphics.h"
pc -g -o ex5 ex5.pas Module.p	ligne de compilation avec un module "Module.p" créé par l'utilisateur
(*commentaire*);	commentaire
{commentaire};	commentaire
program nom;	début: nom du programme
#include "Graphics.h"	utilisation du module graphique. *#include* doit être placée en début de ligne
#include "Xmodule.h"	utilisation d'un module créé par l'utilisateur
const	déclaration de constante
pi = 3.1415;	attribue la valeur 3.1415 à la constante (≠ variable) pi
type	déclaration de types de variable
vecteurT = **array** [0..4] **of** real;	définition d'un type de variable, ce qui donne dans *var: x, y, z, t: vecteurT;*
chambreT = **record**	type dont la description demande plusieurs paramètres. Ex: chaque variable de type
c1, c2, c3: integer;	*chambreT* a 3 sous-variables *c1, c2, c3: integer* et une sous-variable *nom:*
nom: **varying**[32] **of** char;	*varying[32] of char*. etc.
end;	
carac = (bleu, rond, métal, carré);	type énuméré. Les variables de type *carac* auront 4 valeurs possibles
chambrePT = ^chambreT;	*chambrePT* est un pointeur ^ de *chambreT*. ^: avant le nom dans décl. de: *type / var*
var	déclaration de variables
numero: integer;	définition du type de variable: INTEGER: nombre entier
numero: real;	définition du type de variable: REAL: nombre réel
i, j, k, u, bo: integer;	définit le type de plusieurs variables à la fois
carac: char;	définition du type de variable: CHAR: 1 caractère par variable
nom: **varying**[32] **of** char;	définition du type de variable: mot/phrase/espaces avec max. 32 caractères
plusgrand: boolean;	variable booléenne: 2 valeurs possibles: TRUE et FALSE
x: **array** [1..200] **of** real;	déclare un tableau de 200 valeurs distinctes pour la variable x (x peut être de type: *integer, real, char*, ou prédifini sous type: *array [1..200] of vecteurT;*)
x, y, z: vecteurT;	*vecteurT* est un type de variable personnalisée, défini sous TYPE
M: **array**[1..4, 1..2] **of** real;	définition d'une matrice M: ligne = 1..4, colonne = 1..2
nombre: double;	variable réelle avec 2 fois plus de chiffres après la virgule
nombre: longint;	variable entière avec beaucoup plus de capacité
fichier: text;	si on veut lire un fichier du HD, il sera mémorisé dans la variable *fichier*
fichier: file of integer;	fichier contenant des entiers, peut remplacer integer par un autre type ⇒ binaire
maison: **record**	*record* dans *var*, s'utilise comme *begin, end;* Ex: *maison.chambre.c1 := 4*; c.f. p.39
chambre: chambreT;	
cuisine: cuisineT;	
end;	
pièce: **set of** carac;	pièce définie par l'ensemble de caractéristiques définies dans le type *carac*
pièces: **array**[0..3] **of set of** carac;	ensemble de 4 pièces de type *set of...*
piècesBleues: **set of** 0..3;	au lieu de faire un ensemble de pièces on fait un ensemble de caractéristiques (ici il
piècesRondes: **set of** 0..3;	y a 4 pièces). Ex: ∀pièces bleues: on entre leurs valeurs dans le *set piècesBleues*
piècesMétal: **set of** 0..3;	
piècesCarrées: **set of** 0..3;	
chambres: **array**[1..4] **of** chambrePT;	déclare un tableau de 4 pointeurs contenant chacun un bloc de type *chambreT*
procedure VecScalarProduct(**var** result: real); **external**;	liste de procédures **external**: se trouvent dans un fichier (interface) *xxxx.h* Fichier (module) *xxxx.pas* contient la programmation des procédures mentionnées dans *xxxx.h*. Il faut aussi rajouter après begin: *#include "xxxxx.h";*
procedure InitializeVector(**var** vector: VectorT);	**var** permet de recopier les changements de *vector* dans la variable correspondante
function ACot(x: real; n: integer): real;	permet d'utiliser la fonction comme *sin(x)* par exemple. *: real* donne le type du résultat de la fonction ACot
begin	début du programme principal

```
writeln ('le texte');          avec retour à la ligne
write ('le texte');            sans retour à la ligne
writeln ('bonjour O''Hara');   apostrophe dans texte: '
writeln (numero);              affiche la valeur de la variable numero
writeln (numero:12:3);         l'affichage utilise 3 chiffres après la virgule et 12 avant
numero := 5;                   attribue la valeur 5 à la variable numero
readln (un_nombre);            attend que l'utilisateur tape un nombre pour l'attribuer à la variable
readln (i, j);                 attend que l'utilisateur tape 2 nombres pour les enregistrer dans i, j
readln;                        attend que l'utilisateur presse RETURN
VarEntiere := 10 div 3;        10 div 3 = 3 (integer)
x := 10 / 3;                   10 / 3 = 3.33333333 (real)
réel := x mod 4;               le reste de la division de x par 4
entier := trunc(x);            valeur entière inférieure de x
entier := round(0.74 * x);     l'entier le plus proche de 0.74 * x
sin(x), cos(x) arctan(x)       fonctions disponibles en Pascal
sqr(x), sqrt(x), abs(x), ln(x) fonctions disponibles en Pascal. sqr: carré, sqrt: racine carrée, abs: valeur absolue
x := wallclock;                nombre de secondes écoulées depuis 1970
v := random(x);                variable aléatoire entre 0 et 1. x: valeur real ignorée par la routine
x := seed (wallclock);         séquences de nombres différentes. x: integer
for i := 1 to 10 do            ouvre une boucle dans laquelle i prendra les valeurs successives 1 à 10
    begin                      début de la boucle dans for (nécessaire si il y a plusieurs instructions)
    end;                       fin de la boucle for
for i := val1 downto val2 do   la valeur de la variable val1 est décrémentée de 1 à chaque nouvelle boucle jusqu'à
                               atteindre la valeur de la variable val2. i: integer

if i < 0 then                  ouvre une condition écrite entre le if et le then. Si la cond. est fausse → else. La
                               condition peut aussi contenir: AND, NOT, OR (expressions booléennes)
    begin                      ouvre la condition vraie (nécessaire si il y a plusieurs instructions)
    end                        fin de la condition vraie
else                           pour la condition fausse (pas obligatoire)
repeat                         répète les opérations entre le repeat et le until
until (M > 6) AND (M < 10);    jusqu'à ce que la(les) condition(s) mentionnées par until soient remplies
while i <= N do                répète une boucle jusqu'à ce que la condition après le while soit vraie. La condition
                               doit comporter au moins une variable qui est modifiée à l'intérieur de la boucle
                               sinon la boucle s'exécutera indéfiniment.
    begin                      début de la boucle dans while (nécessaire si il y a plusieurs instructions)
    end;                       fin de la boucle while
carac := 'm';                  met la lettre 'm' dans la variable carac (carac: char;)
writeln (succ ('a'));          succ: affiche la lettre qui vient après 'a'
writeln (pred('z'));           pred: affiche la lettre qui vient avant 'z'
writeln (ord(carac));          ord: affiche la valeur en nombre d'une lettre (contenu de carac)
writeln (chr(nb));             chr: affiche la lettre correspondant à un nombre (contenu de nb)
x := length (nom);             length: donne la longueur d'un nom/phrase
carac := nom[4];               nom[4]: la 4-ème lettre de la variable 'nom'
nom[3] := '*';                 nom[3]: remplace par '*' la 3-ème lettre de la variable 'nom'
x := x + 'efgh';               ajoute 'efgh' à la variable x: varying[32] of char;
plusgrand := false;            donne la valeur FALSE à la variable booléenne plusgrand
plusgrand := (x > 15);         c'est la condition TRUE de plusgrand
if plusgrand then              équivalent à if (x > 15) then , équivalent à la condition plusgrand := TRUE
and, or, not                   opérations logiques: et, ou, non. Exemple: while not fini do (fini: booléenne)
```

Graphics.h:

```
FillRectangle (top, left, bottom, right);      rectangle plein. Coin supérieur gauche (top, left)
DrawRectangle (top, left, bottom, right);          contour d'un rectangle. Coin inférieur droit (bas, right)
FillOval (top, left, bottom, right);           ovale plein. Point (0, 0) se trouve dans le coin supérieur gauche
DrawOval (top, left, bottom, right);               contour d'un ovale. Point (400, 400) se trouve dans le coin inférieur droit
DrawLine (fromX, fromY, toX, toY);             trait
PenSize (pixels);                              épaisseur du trait en pixels
SetColor (color);                              niveaux de gris, de 0 (noir) à 8 (blanc)
SetWindowSize (sizeX, sizeY);                      taille initiale de la fenêtre graphique. Doit être appelé avant toutes les autres.  C'est une nouvelle
fenêtre
SuspendRefresh;                                interromp le rafraîchissement de l'écran (permet de dessiner les primitives)
ResumeRefresh;                                 relance le rafraîchissement de l'écran
```

Delay (millisec);	sespend l'exécution du prg durant un certain nb de millisec (animation)
GetTime (var millisec);	donne un temps en millisec. En appelant 2 fois *gettime*:durée portion code
Flush;	après un *writeln* dans une routine graphique: être sur que le msg est affiché

Functions.h:

ASIN(numero :real) :real;	Arcsinus
ACOS(numero :real) :real;	Arccosinus
ATAN(numero :real) :real;	Arctangente
TAN(numero :real) :real;	Tangente
COT(numero :real) :real;	Cotangente
ACOT(numero :real) :real;	Arccotangente
ASINH(numero :real) :real;	Arcsinus hyperbolique
ACOSH(numero :real) :real;	Arccosinus hyperbolique
ATANH(numero :real) :real;	Arctangente hyperbolique
SINH(numero :real) :real;	Sinus hyperbolique
COSH(numero :real) :real;	Cosinus hyperbolique
TANH(numero :real) :real;	Tangente hyperbolique
COTH(numero :real) :real;	Cotangente hyperbolique
ACOTH(numero :real) :real;	Arccotangente hyperbolique
SIGN(numero :real) :real;	Signature
FACT(numero :integer) :real;	Factorielle
COMB(n, p:integer) :integer;	Combinatoire
LOG(numero :real) :real;	Logarithme base 10
LN(numero :real) :real;	Logarithme naturel
LOGXN(numero, base :real) :real;	Logarithme base x
ROOT(numero :real) :real;	Racine carrée
INV(numero :real) :real;	Inverse
PUISSANCE(numero,pot :real): real;	Puissance x

Graphs.h

SetGraphSize(top, left, bottom, right: real);	Bornes du graphique en coordonnées graphiques (doit tjr être utilisé avant DrawGraph...)
SetScreenSize(top, left, bottom, right: integer);	Taille du graphiqe à l'intérieur de la fenêtre (doit tjs être utilisé avant DrawGraph...)
DrawGraphLine(fromGX, fromGY: real; toGX, toGY: real);	Graphe continu. toGX, toGY sont (x + h) et f(x + h)
DrawGraphPoint(hereX, hereY: real; radius: integer);	Graphe discontinu: points x, f(x). Radius: rayon du point
Draw3DLine(fromX, fromY, fromZ: real; toX, toY, toZ: real);	Affiche une ligne 3D en 2D (écran)
Draw3DPoint(hereX, hereY, hereZ: real; radius: integer);	Affiche un point 3D en 2D. Radius: rayon du point
Draw3DLine2(fromX, fromY, fromZ: real; toX, toY, toZ: real; DistObsX : real);	Affiche une ligne 3D en 2D (écran). DistObsX: distance de l'objet à l'observateur: perspective
Draw3DPoint2(hereX, hereY, hereZ: real; radius: integer; DistObsX : real);	Affiche un point 3D en 2D. Radius: rayon du point. DistObsX: bien entre 1 et 10
CartesianToSpherical(x, y, z: real; **var** a, b: real);	

Matrices.h

Contient des types prédéfinis: Vector3T et Matrix33T pour le calcul en 3D (pas besoin de déclarer ces types dans le programme principal pour les utiliser).

InitializeMatrix(var m: array[a..b; c..d] of real);	Attribue des valeurs aléatoires à la matrice
InitializeVector(var v: array[a..b: integer] of real);	Attribue des valeurs aléatoires au vecteur
VectScalarMult (v: array[a..b] of real; s: real; **var** r: array[a..b] of real);	Scalaire * vecteur. Résultat enregistré dans le 3-eme paramètre
MatScalarMult (m: array[a..b; c..d] of real ; s : real; **var** r: array[a..b; c..d] of real);	Scalaire * matrice = matrice
VectVectSum (v1: array[a..b] of real ; v2 : array[a..b] of real ; **var** r : array[a..b] of real);	Vecteur + vecteur = vecteur
MatMatSum (m : array[a..b; c..d] of real ; n : array[a..b; c..d] of real ; **var** r : array[a..b; c..d] of real);	Matrice + matrice = matrice
VectVectMinus (v1 : array[a..b] of real ; v2 : array[a..b] of real ; **var** r : array[a..b] of real);	Vecteur - vecteur = vecteur
MatMatMinus (m : array[a..b; c..d] of real ; n : array[a..b; c..d] of real ; **var** r : array[a..b; c..d] of real);	Matrice - matrice = matrice
ProduitVectoriel (v1: array[a..b] of real ; v2 : array[a..b] of real ; **var** r : array[a..b] of real);	Produit vectoriel de 2 vecteurs3T = vecteur3T
ScalarProduct (v1 : array[a..b] of real ; v2 : array[a..b] of real ; **var** r : real);	Produit scalaire de 2 vecteurs = réel
ProduitMixte (v1 : array[a..b] of real ; v2 : array[a..b] of real ; v3 : array[a..b] of real ; **var** r : real);	Produit mixte de 3 vecteurs3T: vecteur1 * (vecteur2 × vecteur3) = scalaire
WriteVector (**var** v : array[a..b] of real);	Affiche les valeurs d'un vecteur
WriteMatrix (**var** m : array[a..b; c..d] of real);	Affiche les valeurs d'une matrice
MatVectMult (v : array[a..b] of real ; m : array[a..b; c..d] of real ; **var** r : array[a..b] of real);	Vecteur * matrice = vecteur
MatMatMult (m : array[a..b; c..d] of real ; n : array[a..b; c..d] of real ; **var** r : array[a..b; c..d] of real);	Matrice * Matrice = matrice
InitXRotationMatrix (**var** m : Matrix33T ; angle : real);	

```
                                    Crée une matrice de rotation autour de l'axe X selon un angle donné
InitYRotationMatrix (var m : Matrix33T ; angle : real );
                                    Crée une matrice de rotation autour de l'axe Y selon un angle donné
InitZRotationMatrix (var m : Matrix33T ; angle : real );
                                    Crée une matrice de rotation autour de l'axe Z selon un angle donné
InitXYZRotationMatrix(var m : Matrix33T ; x, y, z: real );
                                    Crée une matrice de rotation autour des axes X, Y, Z selon les angles x, y, z donnés
Rotate(v1: Vector3T; x, y, z: real; var r: Vector3T);
                                    Fait directement la rotation d'un vecteur par 3 angles autour de X, Y, Z. Pas bon si bcp rot: lent
SolveSystem( s : array[a..b; c..d] of real ; var r : array[a..b] of real );
                                    résoud: A * X = B. s: matrice contenant dans la dernière colonne les égalités B des équations

InitializeVector(var vector: VectorT); VAR permet de recopier les changements de vector dans la variable correspondante
case choix of                       équivalent à plusieurs if les uns après les autres
   5: begin                         début de la boucle dans case de condition: varaible choix est égale à 5
   end;                             fin de la boucle dans case de condition: varaible choix est égale à 5
   '+', 'p', 'd': begin             début boucle dans case de condition: variable choix contient un + ou p ou d
   end;                             fin boucle dans case de condition: variable choix contient un + ou p ou d
otherwise                           dans les autres cas (pas obligatoire)
x[20] := 15.0                       met la valeur 15.0 à la place de la valeur num. 20 du tableau de la variable x
M[i,j] := random(x);                élément i(ligne), j(colonne) d'une matrice M, où i, j sont des variables
pcexit(1);                          quitte le programme;
reset(fichier, nom_du_fichier);     ouvre un fichier du HD et le stoke dans la variable fichier de type: file of...
rewrite(fichier, nom_du_fichier);   écrit dans le fichier nom_du_fichier du HD, stoké dans la variable fichier
open(fichier, nom_du_fichier);      ouvre et écrit dans le fichier du HD, par l'intermédiaire de la var fichier
write(fichier, bloc);               enregistre le contenu de la variable bloc dans le fichier: file of... du HD
read(fichier, Bloc[1], Bloc[2]);    lit fichier (ligne i) ⇒ valeur n° 1 dans Bloc[1], valeur n° 2 dans Bloc[2], etc.
while not eof(fichier) do           aussi longtemps que l'on a pas lu tout le fichier (eof = end of file) la boucle
   begin                            s'exécute (lire une ligne de fichier, copie ds chaîne, affiche chaîne)
      readln(fichier, chaîne);
      writeln(chaîne);
   end;
close(fichier);                     lorsque on a fini d'utiliser fichier, on ferme l'accès ⇒ enregistre les modifications
chambre.nom := 'cuisine';           'cuisine' ds sous-variable nom de variable chambre de type défini par record
with maison[2] do                   maison est un array of... type défini par record. Ici, nom := 'cuisine' revient à
   begin                            écrire maison [2].nom := 'cuisine'
      nom := 'cuisine';
   end;
pièce := [bleu, métal, carré];      variable de type set of... Bits n° 1, 2, 4 initialisés à 1
[], +, -, *, <=, =, <>              opérations avec set of... []: ensemble vide, +: union, -: différence, *: intersection,
                                  <=: contenu dans, =: égal, <>: non égal. Sens ensembliste
if ([bleu, métal] <= pièces[i]) then  exemple: cherche les pièces bleues et métalliques
piècesBleues := [2, 3];             la var est de type set of 0..3. Les pièces 2 et 3 sont bleues
piècesRouges := [1..3] - piècesBleues; les autres pièces sont rouges (en l'occurence la pièce 2)
piècesCherchées := piècesRouges + (piècesBleues + piècesCarrées);
                                    on cherche les pièces rouges ∪ (bleues ∩ carrées)
for i := 1 to 3 do                  opérateur in (≡ ∈): sens ensembliste (= si pièce n° i ∈ ensemble des pièces rouges)
   if i in piècesRouges then
      writeln(i:3);
new(chambres[1]);                   création de place mémoire pour l'adresse du pointeur chambres[1]
chambres[1]^.nom := 'cuisine';      ^ modifie la valeur du pointeur (= contenu de la variable)
chambres[1].nom := chambres[2].nom;   manipulation des adresses du pointeur chambres[1]: l'adresse correspondant à
                                    l'emplacement du contenu de chambres[2] est copié sur l'adresse de chambres[1]
chambres[3] := nil;                 le pointeur ne pointe plus sur aucune zone mémoire pour tout le bloc chambres[3]
writeln(chambres[1]^.nom);        cela affiche: cuisine (donc la valeur puisque il y a ^)
writeln(chambres[1].nom);           cela affiche un nombre quelconque (p.ex: 00080C030) qui est l'adresse du contenu
chambres[1]^.c1 := chambres[3];     valeur de chambres[3] est pointée par le pointeur c1 sur l'adresse de chambres[3]
writeln(chambres[1]^.c1.nom);       affiche le nom de chambres[3]: idem writeln(chambres[3]^.nom);
dispose(chambres[1]);               libération de la place mémoire utilisée par le pointeur chambres[1]
end.                                Termine le programme principal
```

M-G goto line.

XEmacs Reference Card

(for version 19)

Starting Emacs

To enter XEmacs, just type its name: `xemacs`

To read in a file to edit, see Files, below.

Leaving Emacs

suspend Emacs (or iconify it under X)	`C-z`
exit Emacs permanently	`C-x C-c`

Files

read a file into Emacs	`C-x C-f`
save a file back to disk	`C-x C-s`
save **all** files	`C-x s`
insert contents of another file into this buffer	`C-x i`
replace this file with the file you really want	`C-x C-v`
write buffer to a specified file	`C-x C-w`

Getting Help

The Help system is simple. Type `C-h` and follow the directions. If you are a first-time user, type `C-h t` for a **tutorial**.

remove Help window	`C-x 1`
scroll Help window	`ESC C-v`
apropos: show commands matching a string	`C-h a`
show the function a key runs	`C-h c`
describe a function	`C-h f`
get mode-specific information	`C-h m`

Error Recovery

abort partially typed or executing command	`C-g`
recover a file lost by a system crash	`M-x recover-file`
undo an unwanted change	`C-x u` or `C-_`
restore a buffer to its original contents	`M-x revert-buffer`
redraw garbaged screen	`C-l`

Incremental Search

search forward	`C-s`
search backward	`C-r`
regular expression search	`C-M-s`
reverse regular expression search	`C-M-r`
select previous search string	`M-p`
select next later search string	`M-n`
exit incremental search	`RET`
undo effect of last character	`DEL`
abort current search	`C-g`

Use `C-s` or `C-r` again to repeat the search in either direction. If Emacs is still searching, `C-g` cancels only the part not done.

© 1995 Free Software Foundation, Inc. Permissions on back. v2.0 XEmacs

Motion

entity to move over	backward	forward
character	`C-b`	`C-f`
word	`M-b`	`M-f`
line	`C-p`	`C-n`
go to line beginning (or end)	`C-a`	`C-e`
sentence	`M-a`	`M-e`
paragraph	`M-{`	`M-}`
page	`C-x [`	`C-x ]`
sexp	`C-M-b`	`C-M-f`
function	`C-M-a`	`C-M-e`
go to buffer beginning (or end)	`M-<`	`M->`

scroll to next screen	`C-v`
scroll to previous screen	`M-v`
scroll left	`C-x <`
scroll right	`C-x >`
scroll current line to center of screen	`C-u C-l`

Killing and Deleting

entity to kill	backward	forward
character (delete, not kill)	`DEL`	`C-d`
word	`M-DEL`	`M-d`
line (to end of)	`M-0 C-k`	`C-k`
sentence	`C-x DEL`	`M-k`
sexp	`M-- C-M-k`	`C-M-k`

kill **region**	`C-w`
copy region to kill ring	`M-w`
kill through next occurrence of *char*	`M-z` *char*
yank back last thing killed	`C-y`
replace last yank with previous kill	`M-y`

Marking

set mark here	`C-@` or `C-SPC`
exchange point and mark	`C-x C-x`
set mark *arg* **words** away	`M-@`
mark **paragraph**	`M-h`
mark **page**	`C-x C-p`
mark **sexp**	`C-M-@`
mark **function**	`C-M-h`
mark entire **buffer**	`C-x h`

Query Replace

interactively replace a text string	`M-%`
using regular expressions	`M-x query-replace-regexp`

Valid responses in query-replace mode are

replace this one, go on to next	`SPC`
replace this one, don't move	`,`
skip to next without replacing	`DEL`
replace all remaining matches	`!`
back up to the previous match	`^`
exit query-replace	`ESC`
enter recursive edit (`C-M-c` to exit)	`C-r`

Multiple Windows

delete all other windows	`C-x 1`
delete this window	`C-x 0`
split window in two vertically	`C-x 2`
split window in two horizontally	`C-x 3`
scroll other window	`C-M-v`
switch cursor to another window	`C-x o`
shrink window shorter	`M-x shrink-window`
grow window taller	`C-x ^`
shrink window narrower	`C-x {`
grow window wider	`C-x }`
select buffer in other window	`C-x 4 b`
display buffer in other window	`C-x 4 C-o`
find file in other window	`C-x 4 f`
find file read-only in other window	`C-x 4 r`
run Dired in other window	`C-x 4 d`
find tag in other window	`C-x 4 .`

Formatting

indent current **line** (mode-dependent)	`TAB`
indent **region** (mode-dependent)	`C-M-\`
indent **sexp** (mode-dependent)	`C-M-q`
indent region rigidly *arg* columns	`C-x TAB`
insert newline after point	`C-o`
move rest of line vertically down	`C-M-o`
delete blank lines around point	`C-x C-o`
join line with previous (with arg, next)	`M-^`
delete all white space around point	`M-\`
put exactly one space at point	`M-SPC`
fill paragraph	`M-q`
set fill column	`C-x f`
set prefix each line starts with	`C-x .`

Case Change

uppercase word	`M-u`
lowercase word	`M-l`
capitalize word	`M-c`
uppercase region	`C-x C-u`
lowercase region	`C-x C-l`
capitalize region	`M-x capitalize-region`

The Minibuffer

The following keys are defined in the minibuffer.

complete as much as possible	`TAB`
complete up to one word	`SPC`
complete and execute	`RET`
show possible completions	`?`
fetch previous minibuffer input	`M-p`
fetch next later minibuffer input	`M-n`
regexp search backward through history	`M-r`
regexp search forward through history	`M-s`
abort command	`C-g`

Type `C-x ESC ESC` to edit and repeat the last command that used the minibuffer. The following keys are then defined.

previous minibuffer command	`M-p`
next minibuffer command	`M-n`

XEmacs Reference Card

Buffers

select another buffer	C-x b
list all buffers	C-x C-b
kill a buffer	C-x k

Transposing

transpose **characters**	C-t
transpose **words**	M-t
transpose **lines**	C-x C-t
transpose **sexps**	C-M-t

Spelling Check

check spelling of current word	M-$
check spelling of all words in region	M-x ispell-region
check spelling of entire buffer	M-x ispell-buffer

Tags

find a tag (a definition)	M-.
find next occurrence of tag	C-u M-.
specify a new tags file	M-x visit-tags-table
regexp search on all files in tags table	M-x tags-search
run query-replace on all the files	M-x tags-query-replace
continue last tags search or query-replace	M-,

Shells

execute a shell command	M-!
run a shell command on the region	M-|
filter region through a shell command	C-u M-|
start a shell in window *shell*	M-x shell

Rectangles

copy rectangle to register	C-x r r
kill rectangle	C-x r k
yank rectangle	C-x r y
open rectangle, shifting text right	C-x r o
blank out rectangle	M-x clear-rectangle
prefix each line with a string	M-x string-rectangle

Abbrevs

add global abbrev	C-x a g
add mode-local abbrev	C-x a l
add global expansion for this abbrev	C-x a i g
add mode-local expansion for this abbrev	C-x a i l
explicitly expand abbrev	C-x a e
expand previous word dynamically	M-/

Regular Expressions

any single character except a newline	. (dot)
zero or more repeats	*
one or more repeats	+
zero or one repeat	?
any character in the set	[...]
any character not in the set	[^ ...]
beginning of line	^
end of line	$
quote a special character *c*	\c
alternative ("or")	|
grouping	\(... \)
*n*th group	\n
beginning of buffer	\`
end of buffer	\'
word break	\b
not beginning or end of word	\B
beginning of word	\<
end of word	\>
any word-syntax character	\w
any non-word-syntax character	\W
character with syntax *c*	\sc
character with syntax not *c*	\Sc

Registers

save region in register	C-x r s
insert register contents into buffer	C-x r i
save value of point in register	C-x r SPC
jump to point saved in register	C-x r j

Info

enter the Info documentation reader	C-h i

Moving within a node:

scroll forward	SPC
scroll reverse	DEL
beginning of node	. (dot)

Moving between nodes:

next node	n
previous node	p
move **up**	u
select menu item by name	m
select *n*th menu item by number (1–5)	*n*
follow cross reference (return with l)	f
return to last node you saw	l
return to directory node	d
go to any node by name	g

Other:

run Info **tutorial**	h
list Info commands	?
quit Info	q
search nodes for regexp	s

Keyboard Macros

start defining a keyboard macro	C-x (
end keyboard macro definition	C-x)
execute last-defined keyboard macro	C-x e
append to last keyboard macro	C-u C-x (
name last keyboard macro	M-x name-last-kbd-macro
insert Lisp definition in buffer	M-x insert-kbd-macro

Commands Dealing with Emacs Lisp

eval **sexp** before point	C-x C-e
eval current **defun**	C-M-x
eval **region**	M-x eval-region
eval entire **buffer**	M-x eval-current-buffer
read and eval minibuffer	M-ESC
re-execute last minibuffer command	C-x ESC ESC
read and eval Emacs Lisp file	M-x load-file
load from standard system directory	M-x load-library

Simple Customization

Here are some examples of binding global keys in Emacs Lisp.

```
(global-set-key [(control c) g] 'goto-line)
(global-set-key [(control x) (control k)] 'kill-region)
(global-set-key [(meta #)] 'query-replace-regexp)
```

An example of setting a variable in Emacs Lisp:

```
(setq backup-by-copying-when-linked t)
```

Writing Commands

```
(defun command-name (args)
  "documentation"
  (interactive "template")
  body)
```

An example:

```
(defun this-line-to-top-of-window (line)
  "Reposition line point is on to top of window.
With ARG, put point on line ARG.
Negative counts from bottom."
  (interactive "P")
  (recenter (if (null line)
                0
              (prefix-numeric-value line))))
```

The argument to `interactive` is a string specifying how to get the arguments when the function is called interactively. Type `C-h f interactive` for more information.

Copyright © 1995 Free Software Foundation, Inc.
designed by Stephen Gildea, February 1995 v2.0 XEmacs
for GNU Emacs version 19 on Unix systems
Updated for XEmacs in February 1995 by Ben Wing

Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.

For copies of the GNU Emacs manual, write to the Free Software Foundation, Inc., 675 Massachusetts Ave, Cambridge MA 02139.

Mémento Unix

1 Commande de traitement des fichiers, et noms de fichiers

ls /home/gennart : list files in /home/gennart
ls : list files in current directory
ls .. : list files in parent directory
ls -a : faire la liste de tous les fichiers, même ceux qui commencent par un '.'. -a est appelé une option.
ls -l : afficher toutes les informations à propos des fichiers du répertoire courant (taille, propriétaire, date de création, autorisatorion d'accès, ...)
ls -al : combinaison des deux options précédentes. Attention, ceci ne marche pas pour toutes les commandes. Si le programme indique une erreur sur les options, vous devez séparer les options, comme à la ligne suivante
ls -a -l : combinaison des deux options précédentes.
ls pascal : list files in pascal subdirectory
cp f1.p f2.p : copy f1.p to f2.p
rm f2.p : remove f2.p, effacer f2.p.
pwd : print working directory, afficher le nom du répertoire courant.
cd : change directory to home dir
mkdir pascal : make directory pascal
cd pascal : change to pascal directory
rmdir pascal : remove directory pascal (seulement si le répertoire est complètement vide)
ln -s exo1.p other.p : soft link. exo1.p est un vrai fichier, et other.p est un lien vers ce fichiers. Cela permet par exemple de faire apparaître le même fichier dans deux répertoires.

2 Commandes avancées de traitement des fichiers

head exo37.p : afficher les dix premières lignes du fichier exo37.p
tail exo37.p : afficher les dix dernières lignes du fichier exo37.p
grep -n 'if' exo*.p :trouver dans tous les fichiers dont le nom commence par exo et terminent par '.p' la chaîne de caractère 'if', et afficher le nom du fichier, le numéro de la ligne, suivi de la ligne complète. Voir dans la section 'expression régulières' la signification de *.
find /home/gennart -name *.p -print : trouver dans toute la hiérarchie des répertoires commencant pat /home/gennart les fichier qui se terminent par .p. Voir dans la section 'expression régulières' la signification de *.
find /home/gennart -name *~ -exec grep 'bonjour' {} \; : trouver dans toute la hiérarchie des répertoires commençant par /home/gennart les fichiers qui se terminent par '~'. et leur appliquer la commande grep 'bonjour'. Dans cette ligne de commande '{}' veut dire 'le fichier que vous avez trouvé en utilisant find., et '\;' indique la fin de la commande à effectuer sur le fichier. N'importe quelle commande UNIX peut suivre l'option -exec. Evitez d'employer la commande rm avec find, vous risquez de faire disparaître tous vos fichiers.

3 Imprimante

lp exo1.p : imprimer le fichier exo1.p sur l'imprimante par défaut
lpstat -a : donner l'état de toutes les imprimantes connectées aux machines sur lesquelles vous travaillez
lp -d lp23 exo1.p : imprimer le fichier exo1.p sur l'imprimante lp23. Attention, ceci fonctionne sur la plupart des systèmes UNIX, mais pas dans les salles CO.

4 Jobs

ps -edf : list all process and their ID
ps -edf | grep gennart : list all processes and select from that the processes which user gennart launched.
kill -9 2356 : kill process with ID 2356
rlogin cosun12 : se loguer sur la machine du voisin
who : qui est logué sur cette machine
C-c : interrupt current command
C-z : suspend current command (utile quand on oublie le '&' à la fin d'une commande (par exemple, emacs & ou netscape &).
bg : resume current command in background (emacs & est equivalent à emacs ; C-z ; bg).

```
jobs :                  list jobs in the background
```

5 Autres commandes

```
man ls :                toutes les informations à propos de la commande ls
man -k directory :      toutes les commandes relatives au répertoires
xemacs & : l            aunch xemacs in the background
echo $PATH :            display the content of shellvariable PATH. La variable PATH contient la
                        liste des répertoires où on peut trouver des commandes
printenv :               list all shell variables
source ~gennart/.pascal : adapt shell variables to use the graphics package
pc -g -o first first.p : compile the file first.p into an executable called first
netscape & :            lancer netscape dans le background
pc -I$GINC -g -o graph graph.p -l$(GLIB) : compile the file graph.p using the graphics
                        routine in an executable called graph
rlogin cosun37 :        se loguer à distance sur la machine cosun37. C'est très pratique
                        lorsque votre machine est plantée. Par example, vous plantez cosun35
                        (vous perdez la main, vous ne pouvez plus rentrer de commandes). Vous
                        vous loguez normalement sur cosun37, puis de là, vous vous loguez
                        à distance sur cosun35, en utilisant rlogin. Vous pouvez alors
                        détruire vos jobs sur cosun35 (par exemple, emacs, netscape, ou
                        mailtool) en utilisant les commandes ps -edf et kill, et vous permettre
                        de reprendre la main sur cosun35.
```

6 Droits d'accès aux fichiers

```
ls -l ascii* :
                        -rwxr-xr-x  1 gennart     11224 Oct 30 08:34 ascii*
                        -rw-r--r--  1 gennart       318 Oct 30 08:34 ascii.p
```

Les drois d'accès sont les 10 lettres qui commencent la description d'un fichier quand ls est utilisé avec l'option -l. La première lettre idnique s'il s'agit d'un répertoire, d'un lien vers un autre fichier, ou d'un fichier. Les neufs autres lettres sont groupées en 3 groupes de 3 : read-write-execute pour l'utilisateur, read-write-execute pour son groupe, et read-write-execute pour les autres. L'utilisateur, c'est vous. Vous êtes groupés par année d'étude (par exemple, Meca 1ère, Physique 1ère). En particulier, le fichier 'ascii' est read-write-execute pour l'utilisateur, read-execute pour le groupe, et read-write pour les autres. Le fichier 'ascii.p' est read-write pour l'utilisateur, read pour l'utilisateur, et read pour les autres.

```
ls -l /home :
                        drwxr-xr-x  26 gennart  assist      3584 nov   8 13:51 gennart
                        drwxr-xr-x   7 ggarciam elec01       512 nov   7 18:53 ggarciam
                        drwxr-xr-x   8 skessler meca01       512 nov   8 13:27 skessler
```

gennart, ggarciam et skessler sont des répertoires (la première lettre est un d), qui sont accessibles en lecture et en écriture par l'utilisateur, en lecture par le groupe et les autres. Attention tous les répertoires doivent etre exécutables (avoir la permission x), si vous voulez pouvoir y accéder.

```
chmod +r exo1.p :       ajouter l'accès en lecture pour l'utilisateur, pour le fichier exo1.p
chmod +w exo1.p :       ajouter l'accès en écriture pour l'utilisateur, pour le fichier exo1.p
chmod +x exo1 :         ajouter l'autorisation d'exécuter pour l'utilisateur, pour le fichier
                        exo1
chmod -r exo1.p :       retirer l'accès en lecture pour l'utilisateur, pour le fichier exo1.p
chmod -w exo1.p :       retirer l'accès en écriture pour l'utilisateur, pour le fichier exo1.p
chmod -x exo1 :         retirer l'autorisation d'exécuter pour l'utilisateur, pour le fichier
                        exo1
chmod o+u *.p :         donner aux autres ('o', for others) les mêmes accès que l'utilisateur,
                        pour les fichiers pascal.
chmod g+u *.p :         donner au groupe ('g') es mêmes accès que l'utilisateur, pour les
                        fichiers pascal.
```

7 Expressions régulières

Les expressions régulières permettent de nommer plusieurs fichiers avec une seule chaîne de caractères. En général, dans une expression régulière, '*' veut dire une liste contenant n'importe quels caractères ; [abcd] veut dire les caractères 'a', 'b', 'c' ou 'd' ; [f-x] veut dire n'importe quelle lettre entre f et x.

```
rm f*.p :               effacer tous les fichiers dont le nom commence par f et termine par
                        '.p'. Attention, cette commande est dangereuse. En particulier rm *
                        efface tous vos fichiers d'un coup.
rm f[234].p             effacer les fichiers qui s'appelent f2.p, f3.p, ou f4.p.
rm f2.p f3.p f4.p :     effacer les fichiers qui s'appelent f2.p, f3.p, et f4.p.
rm f[2-7].p :           effacer les fichiers qui s'appelent f2.p, f3.p, f4.p, f5.p, f6.p ou
                        f7.p.
rm f\*.p :              effacer le fichier qui s'appelle f*.p. Le '\' devant le '*' empêche que
                        * ne soit interprété comme une expression régulière.
```

Cours pascal : 6 novembre :

1. Exercice 18
2. Conseils de style (constantes, types, indentation)
3. Routines graphiques

1. Exercice 18 : avoir tjs un prg. complet : → évite des erreurs. Ex: ouvre une parenthèse, la fermer immédiatement. Tout ce que l'on ouvre doit être refermé automatiquement

a. program Mot;
var
begin
end
} début du prg. = prg. complet

```
program Mot,
  var
     phrase: varying [128] ofcha;
     compteur : integer := 0;
     lon : integer;
     i : integer
```

```
begin
  writeln ('Phrase ?');
  readln ( Phrase) ;
  compteur := 0;
  lon := length (phrase);
  for i = 1 to lon do
      begin
        if (phrase[i] = ' ') then
          begin
            compteur := compteur + 1;
          end;
      end;
  compteur := compteur + 1 ;
  writeln ('nombre de mots =',
           compteur);
end.
```

indiquer à l'utilisateur sa contribution

initialiser les variables
compte le nb. de mots de la phrase
boucle for : pour i de 1 à lon
} ferme automatiquement le begin avec le end correspondant

Rem : Xemacs a des menus avec des fonctions prédéfinies. ex : for → met tous les : for..i, and do, etc.

2. Conseils de style : c.f. à la fin du cours

- améliorer : - lisibilité
  - facilité à modifier les prg.
- indentation : (aller à la ligne) (tabulations) (le premier caractère : espace, les autres RETURN seront indentés.
- choix des identificateurs : noms de variables parlants
- utilisation des constantes : - en majuscules pour les reconnaître. ex : PI = 3,1415 (pas de :)
  - remplacer les grandeurs de tableau par des constantes → si on veut changer la taille, il est plus pratique de le changer dans la déclaration de constantes.
- déclaration de types :

```
const
  size = 5
type
  vectorT = array [1...size] of real;
var
  vector, liste, tableau : vectorT
```

3. Routines graphiques

- fillrectangle ■
- drawrectangle ▭
- Filloval ●
- Drawoval ○
- Drawline ／
- pensize
- setcolor : 0 → noir, 8 → blanc

```
program ... ;
# include " Graphics.h"
```
: prg. qui fait appel aux routines graphiques

(1re caractère de la ligne)

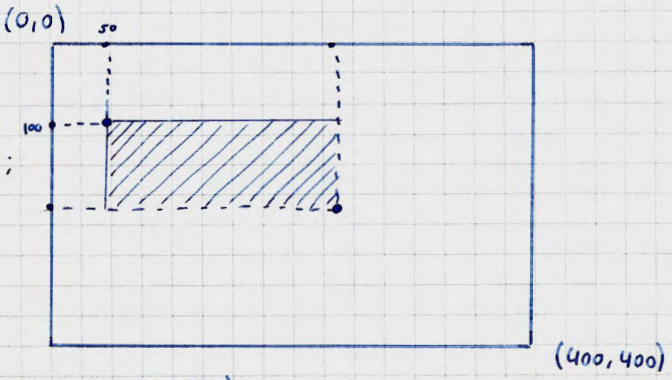
```
begin
  fillrectangle ( top, left, bottom, right : integer) ;
```

Nouvelle ligne de compilation : pc -I$GINC -g -o grp graphics-first.p -l$GLIB

(I majuscule ; l minuscule ; librairie)

fillrectangle :

top left bottom right
(100, 50 200 250);




```
writeln ('Appuyez sur ENTER');
flush ;   (pour être certain que le message soit affiché)
readln ;
```

1) Conversion majuscule / minuscule :

A	65	$\xleftrightarrow{32}$	a	97
B	66	$\xleftrightarrow{32}$	b	98

ord 'a' = 97
chr (97) = 'a'

conversion majuscule → minuscule

```
program MM;
  var
    PetitA, GrandA : integer
    diff : integer
    GrandC, PetitC : char;

begin
  PetitA := ord ('a');
  GrandA := ord ('A');
  diff := PetitA - GrandA;

  writeln ('Entrez un caractère maj : ' );
  readln (GrandC)
  PetitC := chr ( ord (GrandC) + diff);
  writeln ('lettre minuscule = ', PetitC');
```

2) Exercice 19 : première phrase : teste

```
program ex19;
    type
      StringT = varying [32] of char;
    var
      Strings : array [1..10] of stringT;  i:= integer;  DansOrdre : boolean := true;
    begin
      writeln ('Donnez 10 mots');
      for i:=1 to 10 do
        begin
          readln (strings [i]);
        end;
i:=2;  while ( i <= 10)  and (DansOrdre) do
        begin
          if (Strings [i-1] > Strings [i]) then
            begin
              DansOrdre := false;
            end;
        end;
      if (DansOrdre) then
        writeln ('dans l'ordre');
      else
        writeln ('Désordre');
```

2 défauts : (rouge) - déclarer un tableau qui contient 2 chaines de caractères (besoin que de 2 noms alors que Σ tableau est dans la mémoire).
- s'arrête après avoir tout comparé, et non dû qu'il y a une erreur.

Optimisation n°2 :

```
program ex19b; (20)
  type
    StringT = ...;
  var
    S1, S2 : StringT;
    DansOrdre : boolean := true;
  begin
    while (DansOrdre) do        writeln ('entrez des mots');
                                readln (S1);
      begin
        readln (S2);
        if ( S1 > S2 ) then
          begin
            DansOrdre := false;
          end;
                                S1 := S2;
                              end;
```

Exercice 22 : créer un prg. qui demande à l'utilisateur de deviner un mot de 9 lettres

```
program EX22 ;
  var
    Motcaché : varying [32] of char;
    Mottrouvé : varying [32] of char;
    compteur : integer;
    lettre : char;
    i : integer;

  begin
    Motcaché := 'programme';
    Mottrouvé := '*********'
    compteur := 0
    while (motcaché <> mottrouvé ) do
      begin
        writeln ( 'Donnez un caractère');
        readln (lettre);
        for i=1 to length (motcaché ) do
          begin
            if (MotCaché [i] = lettre) then
              begin
                Mottrouvé [i] := lettre;
              end;
          end;

        writeln (Mottrouvé);
        compteur := compteur + 1;
      end;
    writeln ('compteur =', compteur);
  end.
```

Cours du 20 novembre 1996

i) Commentaires sur le Pascal
   a) signification de ' ; '
   b) boucle infinie
   c) affectation (:=) et test (=)
   d) procédures

ii) Procédures

---

i) a) ' ; ' sert à séparer les instructions

```
begin
  writeln ('hello') ;
  writeln ('bonjour') ;
end.
```

```
if (x>5) then X:= 5 ;
(else) x:=x+1 ;
```

↑ l'instruction if est terminée

↳ le compilateur ne comprend pas car l'instruction if est finie

(il ∃ l'instruction vide en Pascal)

```
for i:=1 to 5 do ;
  begin
    writeln (i);
  end;
```

→ l'instruction for est terminée

} le prg. va écrire '5' ... au lieu d'écrire 1, 2, 3, 4, 5

i) b) boucle infinie :

```
i:=1
while (i <=5) do
    begin
      writeln (i);
    end;
```

} boucle infinie

← i:=i+1

avec while il faut : - initialiser la valeur de i
- changer la val. de i du boucle

```
repeat
   ⋮
until (   );
```

} pas obligé de mettre un begin et end.

while : - doit incrémenter les indices
for : - incrémente automatiquement
repeat : - tout l'intérieur de la boucle est passé en revue

i) c) affectation (:=) et test (=)

```
if (x=7) then          } ceci est un test
  begin
    x:=-7;             } ceci est une affectation
  end;
```

i) d) initialisation d'une chaîne de caractères

```
var
  Motcaché : varying [32] of char;
  Mottaouvé : varying [32] of char;
begin
  Motcaché := 'programme';
  {Mottrouvé := '* * * * * * * * *'}
```

faux :

```
for i = 1 to length (motcaché) do
    begin
      Motcaché [i] = '*';
    end;
writeln (Motcaché);
```

juste :

```
Mottrouvé := '';
for i=1 to length (Motcaché) do
    begin
      Mottrouvé := Mottrouvé + '*';
    end;
```

Programmation 27.11.96

1) Mementos: UNIX, PASCAL, EMACS dans : ~gennart / polycopiés / memento Unix.ps
Emacs.ps
Pascal.ps

2) Procédures : Display_vector (name : StringT , vector : VectorT) ;

paramètres formels

mêmes types

Display Vector ('v', V) ; appel de procédure avec paramètres effectifs.
Initialize Vector (V);

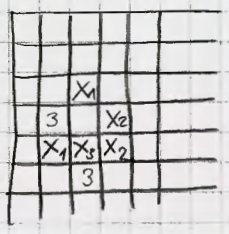
- Initialize Vector (var vector : Vector T);

↳ permet de recopier le contenu de vector dans la variable V.

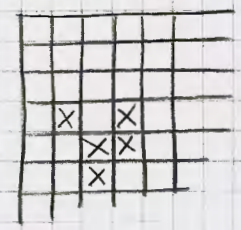
3) type énumérés : SL5-1 ex: boolean : type énuméré.

4) Le jeu de la vie : SL5-5 à SL5-8

état précédent



état suivant



CelluleT = (vivante, morte);

```
program vie ;
  Const
     SIZE = 10;
  type
     CelluleT = (morte, vivante) ;
  var TableauT = array [1...SIZE , 1 SIZE] of celluleT;
     Précédant, Suivant : TableauT ;
     i : integer;
  {liste de procédures}


begin

   EffacerTableau (Précédent);
   Glisseur (Précédent);
   for i := 1 to 10 do
     begin
        CalculerTableau (Suivant, Précédent);
        Afficher Tableau (Suivant);
        Précédent := Suivant ;
        i := i + 1

     end;

end.
```

```
procedure EffacerTableau ( var t: TableauT);
  var
    i,j : integer;

  begin
    for i:=1 to SIZE do
      begin
        for j:=1 to SIZE do
          begin
            t[i,j] := morte;
          end;
      end;
  end;

procedure Glisseur ( var t : TableauT);
  begin
    t[2,3] := vivante;
    t[3;4] := vivante;
    t[4;2] := vivante;
    t[4;3] := vivante;
    t[4;4] := vivante;

procedure CalculerTableau (var S:TableauT; p : TableauT);
  begin var i,j : integer;
    for i:=1 to SIZE do
      begin
        for j:=1 to SIZE do
          begin
            CalculerCellule (S, p, i, j);
          end;
      end;
  end;
```

tableau suivant, tableau précédent

↳ on peut mettre une procédure à l'intérieur d'une autre.

procédure CalculerCellule

pour chaque cellule : compter la Σ des voisins
→ faire des tests pour ne pas sortir du tableau

Programmation I : Examen Intermédiaire

Physiciens

Comment s'y prendre

Au début de l'examen, tapez la commande ~gennart/StartExam dans une fenêtre terminal. Il y a 4 exercices pour lesquels vous devez écrire un programme PASCAL. Remettez les fichiers sources et les exécutables dans votre répertoire ~/lsp-rendu. ***Mettez votre nom en commentaire dans chacun de vos programmes.*** A la fin de l'examen, tapez la commande ~gennart/FinishExam, et demandez à un assitant s'il peut retrouver vos fichiers.

1 Manipulation des chaînes de caractères

On demande d'écrire un programme qui demande à l'utilisateur une phrase qui contient des caractères majuscules et minuscules, copie seulement les caractères majuscules dans une nouvelle chaîne de caractères, et affiche cette dernière. (**eip1.p, 10 points**)

2 Variables booléennes

On demande d'écrire un programme qui initialize une matrice M (tableau à deux dimensions, `array[1..N,1..N] of real`) de taille NxN, et vérifie que la matrice est antisymétrique (M(i,j) = -M(j, i)). Vérifiez que votre programme fonctionne en initialisant la matrice M avec les valeurs M(i,j) := i-j. Revérifiez avec les valeurs M(i,j) := 3*i-j. (**eip2.p, 10 points**)

$$\underset{\text{antisymétrique}}{M(i,j) = i-j} \Rightarrow \begin{bmatrix} 0 & -1 & -2 & -3 \\ 1 & 0 & -1 & -2 \\ 2 & 1 & 0 & -1 \\ 3 & 2 & 1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 2 & 1 & 0 & -1 \\ 5 & 4 & 3 & 2 \\ 8 & 7 & 6 & 5 \\ 11 & 10 & 9 & 8 \end{bmatrix} \Leftarrow \underset{\text{non antisymétrique}}{3i-j = M(i,j)}$$

3 Calcul approché de la fonction arctangeante

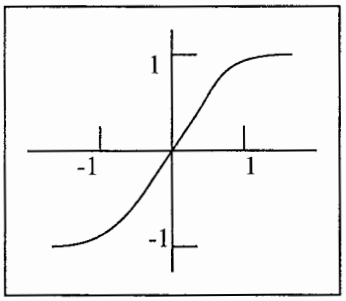
On demande d'écrire un programme qui calcule la valeur approchée de la fonction atan (x), en utilisant la formule :

$$\text{atan}(x) \approx x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \ldots = \sum_{k=0}^{N} (-1)^k \cdot \frac{x^{2k+1}}{2k+1}$$

Faites le programme pour une valeur N = 3, et vérifiez que pour des valeurs petites de x (entre -1 et 1), la formule fonctionne ({x = 0.5, arctan(x) = 0.46365, approximation = 0.46347}, {x = 0.7, arctan (x) = 0.61073, approximation = 0.60752}, {x = 1.0, arctan (x) = 0.78540, approximation = 0.72381}). (**eip3.p, 8 points**). Faites l'exercice pour une valeur quelconque de N (**eip3.p, 10 points**).

4 Routines graphiques

On demande de tracer dans la fenêtre graphique la courbe arctan (x) pour des valeurs de x entre -2.0 et 2.0. Arrangez-vous pour que la courbe soit centrée dans la fenêtre graphique (**eip4.p, 10 points**).



1 Examen Intermédiaire : commentaires sur les erreurs les plus fréquentes

Corrigés. Vous trouverez un corrigé dans ~ysaison/eim pour les mécaniciens et ~ysaison/eip pour les physiciens.

Manipulation des chaînes de caractères. Pour remplir une chaîne de caractères avec un nombre choisi de '*', il faut employer l'opérateur de concaténation '+'. Utiliser l'affectation s[i] := '*' ne fonctionne pas. Cela est dû au fait que (1) chaque chaîne de caractères retient sa longueur utilisée, (2) et que seule l'affection d'une chaîne complète ou l'opérateur de concaténation modifie cette longueur.

```
program NeMarchePas;
  var
    s : varying[32] of char ;
    i : integer ;
  begin
    s := '' ;
    for i := 1 to 10 do
      begin
        s[i] := '*' ;
      end ;
    writeln (s) ;
  end.
(* Dans ce programme, la longueur
 * utilisée de la chaîne reste
 * nulle
 *)
```

```
program Fonctionne;
  var
    s : varying[32] of char ;
    i : integer ;
  begin
    s := '' ;
    for i := 1 to 10 do
      begin
        s := s + '*' ;
      end ;
    writeln (s) ;
  end.
(* Dans ce programme, la longueur
 * utilisée de la chaîne est modifiée
 * grâce à l'opérateur '+'
 *)
```

Le programme de gauche ne fonctionne pas, car, pour le programme, la longueur utilisée de la chaîne reste 0 tout au long du programme. Lorsque l'instruction writeln est appelée, la longueur utilisée de la chaîne étant 0, rien n'est affiché à l'écran, bien que la mémoir contienne effectivement les '*'. Le programme de droite fonctionne parce qu'à chaque itération de la boucle for, la longueur utilisée de la chaîne de caractères est augmentée d'une unité, grâce à l'utilisation de l'opérateur '+'.

Dans le même ordre d'idée, je ne recommande pas l'utilisation du type `string` dans l'environnement Pascal sur les stations de travail Sun. Les problèmes aves `string` sont nombreux, en particulier lorsqu'on veut comparer deux chaînes de caractères. Sa longueur est toujours 80, quelque soit son contenu, ce qui pose aussi des problèmes, en particulier lors de l'affichage et des comparaisons.Il vaut mieux utiliser la desclaration s : **varying**[32] **of** char ;. Le 32 dans la déclaration est la longueur maximale réservée dans la mémoire pour la chaîne de caractères.

```
program NeMarchePas;
  var
    s : string ;
    i : integer ;
  begin
    s := 'AbAbA' ;
    for i := 1 to length (s) do
      begin
        if ('A' <= s[i]) and
           (s[i] <= 'Z') then
          begin
            s[i] := '' ;
          end ;
      end ;
    writeln (s) ; (* bb *)
    writeln (s = 'bb') ; (* FALSE !?!! *)
  end.
```

```
program Fonctionne ;
  var
    s, t: varying[32] of char ;
    i : integer ;
  begin
    s := 'AbAbA' ; t := '' ;
    for i := 1 to length (s) do
      begin
        if ('a' <= s[i]) and
           (s[i] <= 'z') then
          begin
            t := t + s[i] ;
          end ;
      end ;
    writeln (t) ; (* bb *)
    writeln (t = 'bb') ; (* true *)
  end.
```

Les deux programmes ci-dessus ont pour but de retirer les majuscules d'une chaîne de caractères. Dans le programme de gauche, la comparaison entre la variable s et la chaîne de caractères 'bb' donne un résultat `false` bien que les deux chaînes s'affichent à l'écran de la même façon. Ceci est dû au fait que la longueur utilisée de la chaîne de caractères s est 5, tandis que la longueur utilisée de la chaîne de caractères 'bb' est 2. Au moment de l'affichage, la chaîne de caractères s contient '❀b❀b❀'. ❀ n'est pas un caractère affichable, et writeln l'ignore donc. La bonne façon de procéder est dans le programme de droite. Il faut déclarer une chaîne de caractères t, où l'on ajoute les caractères minuscules en utilisant l'opérateur de concaténation '+'.

Variables booléennes. Dans l'exercice 2 de l'examen pour les physiciens, on demandait d'écrire un programme qui vérifie qu'une matrice M est antisymétrique (M(i,j) = -M(j, i)). Cela a donné lieu à de nombreuses erreurs de logique.

```
program NeMarchePas;
  const
    SIZE = 4 ;
  var
    m : array[1..N, 1..N] of real ;
    i, j: integer ;
    antisym : boolean ;
  begin
    antisym := false ;
    for i := 1 to SIZE do
      for j:= 1 to SIZE do
        if (m(i,j) = -m(j,i)) then
          antisym := true ;
end.
(* Pour que antisym soit vrai il suffit
 * que le terme m[SIZE,SIZE] = 0
 *)
```

```
program NeMarchePasNonPlus ;
  const
    SIZE = 4 ;
  var
    m : array[1..N, 1..N] of real ;
    i, j: integer ;
    antisym : boolean ;
  begin
    antisym := false ;
    for i := 1 to SIZE do
      for j:= 1 to SIZE do
        if (m(i,j) = -m(j,i)) then
          antisym := true
        else
          antisym := false ;
end. (* ce programme-ci ne teste que la
        denière valeur du tableau *)
```

Les deux programmes ci-dessus ne fonctionnent pas. Par exemple, dans le programme de gauche, il suffit que `m[SIZE,SIZE] = 0` pour que la variable booléenne `antisym` soit `true`. Le second programme teste seulement le terme `m[SIZE,SIZE]` de la matrice. La bonne façon de faire est montrée ci-dessous.

```
program Correct ;
  const
    SIZE = 4 ;
  var
    m : array[1..N, 1..N] of real ;
    i, j: integer ;
    antisym : boolean ;
  begin
    antisym := true ;
    for i := 1 to SIZE do          (* Pour que antisym soit vrai il faut  *)
      for j:= 1 to SIZE do         (* que pour toutes les valeurs de i et j, *)
        if (m(i,j) <> -m(j,i)) then(* m[i,j] = -m[j,i] *)
          antisym := false ;
end.
```

2 Le reste du cours

Modules (2 semaines) : affichage de fonctions (transparents 4-1 à 4-8) et calcul matriciel (transparents 3-26 à 3-32). Compilation séparée. Application au problème de la rotation en 3 dimensions (p. 49 à 52). But : créer un module Graphics.p contenant les fonctions et un module Matrices.p et les vérifier sur le problème de la rotation 3-D.

Structures les données (2 semaines) : problème du treillis (p. 52 à 55, transparents 7-1 à 7-32). Introduction à l'utilisation des fichiers. Ignorer le bas de la page 55 (à partir de 'Résumé des fichiers fournis'). La routine de résolution de systèmes linéaires est donnée dans les fichiers ~gennart/exercices-corriges/BGMatrices.h et BGMatrices.p. Le programme BGTest.p montre l'utilisation de la routine bgSolveSystem. Pour utiliser le programme BGTest, copiez lesfichiers nécessaires dans un de vos répertoires, et compilez en utilisant la ligne de compilation `pc -g -o BGTest BGTest.p BGMatrices.p`. La matrice du sytème pour un pont à 5 noeuds est :

$$\begin{bmatrix} -0.555 & -1.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ -0.832 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.555 & 0.000 & -0.707 & -1.000 & 0.000 & 0.000 & 0.000 \\ 0.832 & 0.000 & 0.707 & 0.000 & 0.000 & 0.000 & 0.000 \\ 0.000 & 1.000 & 0.707 & 0.000 & -0.707 & 0.000 & -1.000 \\ 0.000 & 0.000 & -0.707 & 0.000 & -0.707 & 0.000 & 0.000 \\ 0.000 & 0.000 & 0.000 & 1.000 & 0.707 & -0.555 & 0.000 \end{bmatrix} \cdot \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \\ f_7 \end{bmatrix} = \begin{bmatrix} 0.000 \\ 1000.0 \\ 0.000 \\ 0.000 \\ 0.000 \\ -2000.0 \\ 0.000 \end{bmatrix}$$

Pointeurs (2 semaines) : listes chaînées (p. 42 à 46, transparents 8-1 à 8-20).

Examen final : dernière semaine. Utilisation des modules, (2 sem) des structures de données (2 sem.) et des pointeurs, (2 sem.)

Informatique 11.12.96 SL3.28 - 29

Vectorsprogram.pas

```
program vecteurs;
#include "VectorsInterfaces.h"
var
  v: array [1..5] of real;
  w: array [1..3] of real;
begin
  : Initialize vector
  : Display vector
end;
```

VectorsInterfaces.h (header)

interface

```
procedure InitializeVector
    (var vector: array [...]..);
    external;
procedure DisplayVector (vector:
    array [bi,.. bs: integer]
    of real; external;
```

VectorsModule.pas

module

```
module VectorsModule.pas
procedure Initialize vector
        (var vector ...);
    var
      i: integer;
    begin
      for i:= ...
      :
    end;
procedure Display vector (...)
var
  ..
begin
  :
end;
```

Ligne de compilation: `pc -g -O vct VectorsProgram.pas VectorsModule.pas`

- vct : exécutable
- VectorsModule.pas : tous les modules utilisés par le programme.

Faire des modules pour : { – vecteurs ; – matrices c.f. SL3-32 }

procedure MatVecMult : $\underset{bsy=3}{\overset{biy=1}{Y}} = \underset{bsmL=3,\ bsmc=3}{\overset{biL=1,\ bimc=3}{M}} \cdot \underset{bsx=3}{\overset{bix=1}{V}} \Rightarrow Y = M \cdot V \Rightarrow Y_i = \sum_{j=1}^{N} M_{ij} \cdot X_j$

```
procedure MatVecMult (X: array [bix ... bsx: integer] of real;
                      m: array [bimL ... bsmL: integer, bimc ... bsmc: integer] of real;
                      var y: array [biy ... bsy: integer] of real);

  var
    i,j: integer;
    mL,mc: integer;
  begin
    mL := bimL;
    mc := bimc;
    for i:= biy to bsy do
      begin
        y[i]:=0;
        for j:=bix to bsx do
          begin
            y[i] := y[i] + m[mL,mc] * x[j];
            mc := mc+1;
          end;
        mL := mL + 1;
      end;
```

Informatique 18.12.96

source ~gennart/. plural

```
program sinus;
# include "Graphics.h"
# include " Graphs.h"
const
  pi = 3.1415

var
  x0, y0, x1, y1 : real
  xi0, yi0, xi1, yi1 : integer;
begin
   SetScreenSize (100, 100, 300, 300)
   SetGraphSize (1, -Pi, -1, Pi)

   x0 := -Pi
   y0 := sin(x0)
   while (x0 ≤ Pi) do
      begin
        x1 := x0 + 0.1 ;
        y1 := sin (x1) ;
        Conversion (x0, y0, xi0, yi0)
        Conversion ( x1, y1, xi1, yi1)
        DrawLine (xi0, yi0, xi1, yi1)
        x0 := x1
        y0 := y1
      end;
```

Graphs.h

```
SetScreenSize (t,l,b,r : integer); external;
SetGraphSize (t,l,b,r : real ); external;
Conversion (x,y : real ; var xi, yi : integer ); external;
```

Graphs.p

```
module Graphs
   var
     TopG, leftG, bottomG, rightG : real;
     TopS, leftS, bottomS, rightS : integer;

procedure SetScreensize (t,l,b,r : integer);
   begin
     TopS := t , leftS := l , bottomS := b, rightS := r ;
   end;

procedure SetGraphsize (t,l,b,r : real);
   begin
     TopG := t , leftG := l ; bottomG := b , rightG := r;
   end;

procedure Conversion (x,y : real ; var xi, yi : integer);
     var
       scaleX , scaleY : real ;

     begin
       scaleX := (RightS - leftS) / (RightG - leftG);
       scaleY := (BottomS - TopS) / (TopG - BottomG);
       xi := round ( (x - leftG) * scaleX) + leftS;
       yi := round ( (topG - y) * scaleY) + topS;
     end;
```

Rotation 3D : c.f. p 49

a. rotation parallèle au plan:

3-D : X, Y, Z : real
2-D : x, y : real ⇒ conversion : Xi, Yi : integer.

si : X=0 , x = Y
y = Z

si : Y=0 et Z=0, alors $x = -\frac{X}{2}$, $Y = -\frac{X}{2}$ } en général : $x = Y - \frac{X}{2}$, $y = Z - \frac{X}{2}$

Projection en perspective :

```
x := Y / (ViewPoint - X)
y := Z / (ViewPoint - x)

const
  ViewPoint = 5.0
```

Rotation :

```
program rotation3D

Const
  pi = 3.1415

Type
  Vecteur3T = array [1..3] of real;
  Matrice33T = array [1..3, 1..3] of real;

Var
  RotationX : Matrice33T;
  V, : Vecteur3T;

procedure InitializeXRotation (var m : Matrice33T; theta : real)
  begin
    m[1,1] := 1; ....
    m[2,2] := cos(theta); ....
  end;
```

Programmation 10.01.96

Encore à traiter : – Structures (record)
– Entées-sorties (fichiers externes)
– pointeurs

} examen : comporte des entées-sorties.

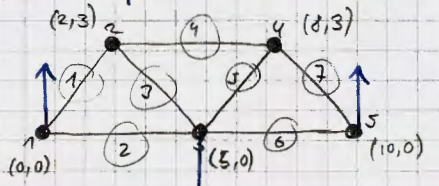
- 8 janvier : rot. 3D + pont + structures
- 15 janvier : pont : entées sorties, pointeurs
- 22 janvier : pour examen final : { – routine de tri de structures
- 29 janvier : pointeurs, listes chain. { – routines de transformation 3D → 2D
- 5 février : examen.

Problème du pont : SL7-2

méthode de construct. d'un treillis :

n noeuds	n barres
2	1
3	3
4	5
5	7
n noeuds	2 * n barres – 3

donner un indice pour chaque noeud / barre :



} pas besoin d'être symmétrique

noeuds :	position X	position Y	Force X	Force Y
1	0	0	0	1000
2	2	3	0	0
3	5	0	0	–2000
4	8	3	0	0
5	10	0		1000

barres :	origine	extémité	Force interne
1	1	2	?
2	1	3	?
3	2	3	?
4	2	4	?
5			?
6			?
7			?

But : afficher le treillis (pour cette semaine)

```
program treillis;
  const
    Nnodes = 5;
    Nbarres = 2 * Nnodes - 3;
  type
    NodeT = record
              index : integer;          (* indice : n° du noeud *)
              posX, posY : real;
              ForceX, ForceY : real;
            end;
    BarT  = record
              index : integer;
              FromNode, ToNode : integer;
              InternalForce : real;
            end;
  var
    Nodes : array[1..Nnodes] of NodeT;
    Bars  : array[1..Nbarres] of BarT;
    i : integer;
    x : real;
```

```
begin
procedure InitializeBridge;
  begin
    Nodes[1].Index := 1;
    Nodes[1].PositionX := 0.0;
          "        Y := 0.0;
    Nodes[1].ForceX := 0.0;
          .ForceY := 1000.0;
          :etc.

  end;

procedure DisplayBridge;
  var
    fromX, fromY, toX, toY : real;

  begin
    for i:=1 to Nodes do
      begin
        :
        FromNode := Bars[i].FromNode;   ToNode := Bars[i].ToNode;
        :
        DrawGraphLine (fromX, fromY, toX, toY);
      end;
  end;
```

```
Bars[1].index := 1;
Bars[1].FromNode := 1;
    [1].ToNode := 2;
     :etc.
```

```
from_X := Nodes[FromNode].PositionX;
from_Y := Nodes[FromNode].PositionY;
toX := Nodes[ToNode].positionX;
toY := Nodes[ToNode].positionY;
```

Programmation 15.01.97

Plan : 1. Entrées / sorties
- i – Ecran
- ii – fichier
- iii – initialisation du pont

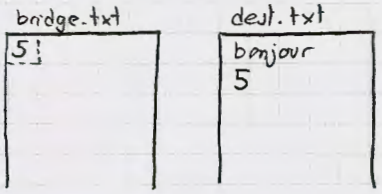
2. réalisation du problème du pont
- i – initialisation d'un système d'équations linéaires
- ii – résolution du système (gennart / exercices-corrigés / BGMatrices.h)

—

1. i) utiliser readln, writeln, write, déconseillé : read.

ii) dans un fichier : – lire : – écrire :

```
program IO;
var
  Fichier : text;
  x : integer;
  r : real;        destination : text;
begin
→ reset (fichier, 'bridge.txt');
→ readln (fichier, x)
→ rewrite (destination, 'dest.txt');
→ writeln (destination, 'bonjour');
  writeln (destination, x);
```



iii) initialisation du pont.
créer un fichier bridge.txt : SL7-13

remarque : si on ouvre avec rewrite un fichier, ce fichier sera totalement effacé.

```
procedure InitializeBridge;
  var
    BridgeFile : text;

  begin
    reset (BridgeFile, 'bridge.txt');
    readln (nnodes);
    if (nnodes <> NNODES) then writeln ('type mismatch');
    for i := 1 to nnodes do
      begin
        readln (BridgeFile, Nodes[i].PositionX, Nodes[i].PositionY, Nodes[i].ForceX
                                                , Nodes[i].ForceY);
        Nodes[i].Index := i;
      end;
  end;
```

→ rajouter une deuxième boucle pour lire les barres : SL7-14

2. ΣF_i à chaque noeud est nulle. Somme des forces : $X_1 = 0$, $Y_1 = 0$, $X_2 = 0$, $Y_2 = 0$

Cela s'écrit : SL7-19 : 5 Noeuds x,y, 10 équations, 7 inconnues, on peut ignorer les 3 dernières équations : 7×7.

Donné : routine solve system fournie : seule difficulté restante : initialiser le système correctement.

```
var
  system : array [1.. NBARS-2 , 1.. NBARS+1-1] of real
  Internalforces : array [1.. NBARS] of real;

begin
  InitializeSystem;
  SolveSystem (System; InternalForces);

end;
```

Difficulté : construire la matrice SL7-20 :

procedure Initialize system:

```
begin   4
    for i:=1 to NBARS do
        begin
          fromNode := Bars[i].FromNode ;      2
          ToNode:= Bars[i].ToNode      ;      4
          Lfx :=2*fromNode -1;        3
          Lfy := 2* fromNode  ;       4
          Ltx := 2 * toNode-1 ;       7
          Lty :=  2* toNode   ;       8
        end;
end;
```

```
if Lfx <= NBARS then
    System[lfx, i] := .....
```

Lfx : line from X
Lfy : line to y

Programmation 22.01.96

1. Rotation 3D } routines graphiques
2. Problème du treillis } routines de calcul matriciel
3. Tri de structures } c.f. polycopié } EXAMEN => le prg. ci-dessous aussi.
4. Pointeurs

programme principal

```
program Names;

const
  MAXSIZE = 10'000;
type
  StringT = varying [32] of char;
  PersonT = record
              Lastname, Firstname : StringT;   2×32 = 64 octets
              Day, Month, Year : integer;      3×4 octets = 12 octets   } 76 octets × 1000 personnes = 760 kb
            end;

var
  persons : array[1..MAXSIZE] of PersonT;
  Npersons : integer;
begin

  readPersons ('noms.txt');
  SortPersons;
  DisplayPersons;

end.
```

```
procedure readPersons (n : StringT);
  var
    f : text;

  begin
    Npersons := 0;
    reset (f,n);
    while not eof (f) do
      begin
        NPersons := Npersons + 1;
        readln (f, Persons[Npersons].LastName);
        readln (f, Persons[Npersons].Firstname);
              ... etc ...
      end;
  end;
```

Algorithme de tri pour sortPersons

7 5 3 1
5 7 3 1
5 3 7 1
5 3 1 (7) ← on est certain que ce nb. est le + grd.

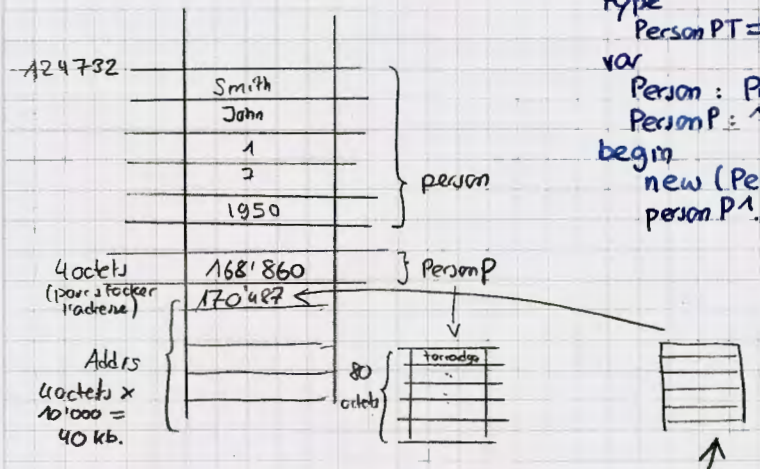
on recommence:
3 5 1
3 1 (5)
1 (3)

```
procedure SortPersons
  var
    i,j : integer;
    temp : PersonT;
  begin
    for i := Npersons - 1 downto 1 do
      begin
        for j := 1 to i do
          begin
            if (Persons[j].LastName > Persons[j+1].LastName) do
              begin
                temp := Persons[j];
                Persons[j] := Persons[j+1];
                Persons[j+1] := temp;
              end;
          end;
      end;
  end;
```

} prend du temps à recopier chaque fois dans la mémoire. Avec les pointeurs on ne recopie que l'adresse.

4. Pointeurs

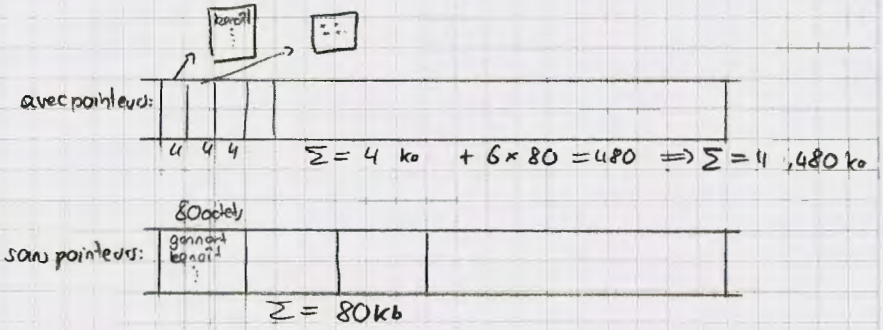
⇒ on a déclaré un tableau de 10'000 structures et on en a utilisé que 6. ⇒ on va éviter de gaspiller de la mémoire et déclarer un pointeur.



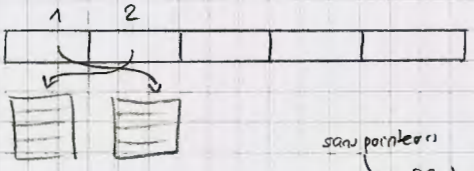

```
type
  PersonPT = ^PersonT;
var
  Person : PersonT;        AddrsP : array [1..MAXSIZE] of PersonPT;
  PersonP : ^PersonT;
begin
  new (PersonP);      ⇒ à présent peut utiliser la variable.
  personP^.LastName := 'torradga';
```

```
procedure ReadPersons (n : StringT);
var
  f : text

begin
  reset (f,n);
  while not eof (f) do
    begin
      Npersons := NPersons + 1;
      new (Addrs [Npersons]^.LastName;)   new: rajoute un espace mémoire : structure pour 1 personne
        ...etc...
    end;
end;
```

```
procedure  Names;
const
  MAX.SIZE = 10'000;

type
  StringT = ... ;
  PersonT = ... ;
  PersonPT = ^PersonT;

var
  NPeaons : integer;
  AddrsP : array [1.. MAXSIZE] of PersonPT;

begin
    :
  Sort Persons;
    :
end.
```




```
procedure  SortPersons;
var
  i,j : integer;
  tmp : PersonPT;
begin
    :  idem
    :
    if (Addrs [j]^.Lastname > Addrs [j+1]^.Lastname) then
      begin
        tmp := Addrs [j];
        Addrs [j] := Addrs [j+1];
        Addrs [j+1] := tmp;
      end;
end;
```



on échange que les addresses ⇒ on recopie moins de bytes ⇒ plus rapide

sans pointeurs → 80 bytes · 3 = 240
avec pointeurs → 4 bytes · 3 = 12
} facteur 20

Programmation 29.01.96

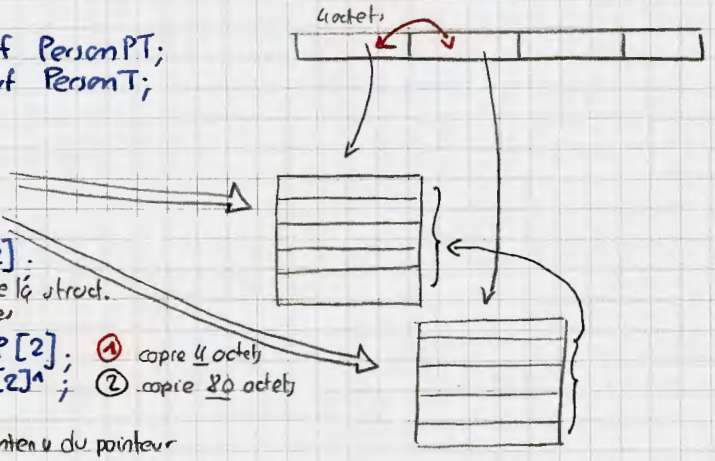
Plan: – pointeurs : listes avec double lien
= programme ne marche pas (writeln)
– programme marche-t-il ?
– Examen : entrées/sorties, traitement, affichage écran. Evaluation

```
program noms;
 const
    MAXSIZE = 10'000;

 type
    StringT = varying [32] of char;
    PersonPT = ^ PersonT
    PersonT = record
                 Firstname, Lastname : StringT;
                 Day, month, year : integer;
              end;

 var
    PersonsP : array [1..4] of PersonPT;
    Persons : array [1..4] of PersonT;

 begin
    new (PersonsP [1]);
    new (PersonsP [2]);
    Persons [1] := Persons [2];
       ↳ sans pointeurs : copie toute la struct.
                          => 80 bytes
    PersonsP [1] := PersonsP [2];   ① copie 4 octets
    PersonsP[1]^ := PersonsP [2]^;  ② copie 80 octets
```

copie 80 octets

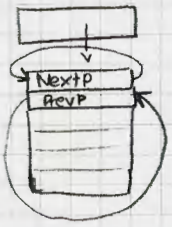
copie uniquement le pointeur (= l'adresse)

copie le contenu du pointeur

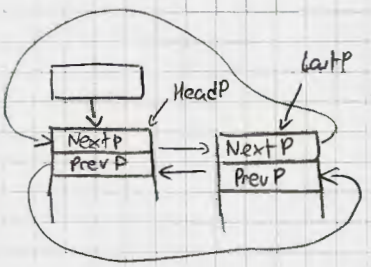


Listes : on travaille avec 1 seul pointeur :

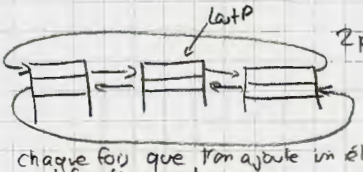
```
type
   PersonPT = PersonT^;
   PersonT = record
                Firstname, Lastname : StringT;
                Day, month, year : integer;
                NextP, PrevP : PersonPT;

var
  PersonListP : PersonPT

begin
  new (PersonListP);
  PersonList^.NextP := PersonListP;
  PersonList^.Prev := PersonListP;
```



"0 personne" : tête de liste : initialise un élément vide

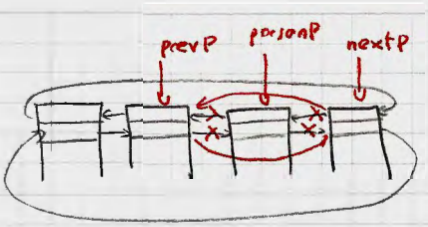



```
procedure AppendPerson (var PersonListP, PersonP : PersonPT);
var
  headP, lastP : PersonPT;
begin
  headP := PersonListP;
  lastP := PersonListP^.PrevP;
  PersonP^.PrevP := lastP;
  PersonP^.NextP := headP;
  lastP^.NextP := PersonP
  headP^.prevP := personP;
end;
```

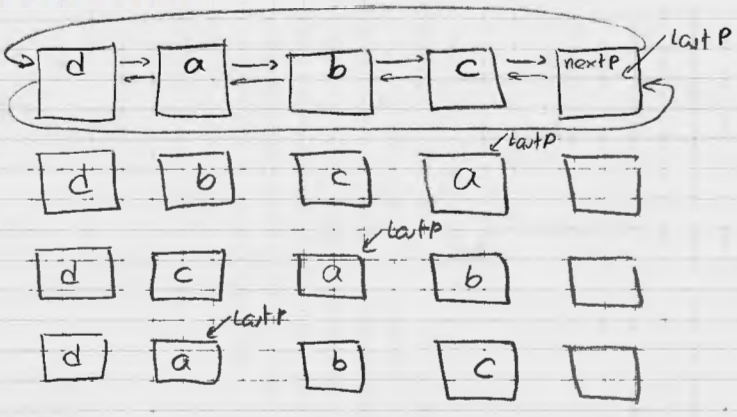


chaque fois que l'on ajoute un élément, on modifie 4 pointeurs

```
procedure RemovePerson (PersonP: PersonPT);
var
   nextP, prevP : PersonPT;

begin
   nextP := PersonP^.NextP;
   prevP := PersonP^.PrevP;
   prevP^.NextP  := nextP;
   nextP^.prevP  := prevP;
   personP^.NextP := nil;
   personp^.PrevP := nil;
end;
```



Utilisation des listes :



on parcourt tous les éléments jusqu'à lastP, on trouve le ⊕ petit et on le met en fin de liste.

```
procedure sortpersons (PersonListP : PersonPT);
var
   personP, minP, last : PersonPT;
begin
   while (lastP^.prev <> PersonlistP) do
      begin
      personP :=  PersonListP^.NextP;
      minP    :=  PersonP;
      while (personP  <>  lastP) do
         begin
            if ( PersonP^.Lastname  <  minP^.Lastname ) then
               begin
                  minP:= personP;
               end;
            PersonP := PersonP^.Next;
         end.
      removeperson (minP);
      AppendPerson (personlistP, minP);
      if  (lastP  =  PersonListP) then
         lastP:= minP;
end.
```

lastP := PersonlistP

Préparation examen

- utiliser les pointeurs
- savoir utiliser SolveSystem

examen :
- lecture fichiers
- 3D : projection, conversion
- mettre en forme une matrice pour SolveSystem
- pointeurs

1 Tri sur structures

On propose de lire dans un fichier un carnet d'adresses (simplifié), et de trier dans l'ordre alphabétique les personnes dans le carnet. Une personne est représentée dans le fichier d'adresses par 5 champs : son nom de famille, son prénom, et sa date de naissance (jour, mois, année).

1.1 Fichier de noms (~exercices-corriges/noms.txt)

```
gennart
benoit
28
6
1962

emmel
patrick
14
10
1971

figueiredo
oscar
21
7
1971

courtois
olivier
21
10
1972

messerli
vincent
30
10
1969

tarraga
joaquin
12
9
1971
```

1.2 Programme principal sans pointeurs (~exercices-corrigés/AddressBook1.p)

```
program AddressBook ;
  const MAXSIZE = 1000 ;
  type
    StringT = varying[32] of char ;
    PersonT = record
        LastName : StringT ;
        FirstName : StringT ;
        BirthDay : integer ;
        BirthMonth : integer ;
        BirthYear : integer ;
      end ;
  var
    Addr : array[1..MAXSIZE] of PersonT ;
    NAddr : integer ;
  procedure ReadAddresses (fn : StringT) ;
    var af : text ;
    begin
      NAddr := 0 ; reset (af, fn) ;
      while (not eof (af)) do
        begin
          NAddr := NAddr + 1 ;
          readln (af, Addr[NAddr].LastName) ;
          readln (af, Addr[NAddr].FirstName) ;
          readln (af, Addr[NAddr].BirthDay) ;
          readln (af, Addr[NAddr].BirthMonth) ;
          readln (af, Addr[NAddr].BirthYear) ;
          readln (af) ;
        end ;
    end ;
  procedure DisplayAddresses ;
    var i : integer ;
    begin
      for i := 1 to NAddr do
        begin
          writeln
            ( Addr[i].LastName:16
            , Addr[i].FirstName:16, ' '
            , Addr[i].BirthDay:2, '-'
            , Addr[i].BirthMonth:2, '-'
            , Addr[i].BirthYear:4
            ) ;
        end ;
    end ;
```

```
procedure SortAddresses ;
  var  tmp : PersonT ;
       i, j : integer ;
  begin
    for i := 1 to NAddr do
      begin
        writeln ('Before step ', i:1) ;
        DisplayAddresses ;
        for j := 1 to NAddr-i do
          begin
            if ( Addr[j].LastName
               > Addr[j+1].LastName
               ) then
              begin
                tmp := Addr[j] ;
                Addr[j] := Addr[j+1] ;
                Addr[j+1] := tmp ;
              end ;
          end ;
      end ;
  end ;
begin
  ReadAddresses ('ab.txt') ;
  SortAddresses ;
  writeln ('Final') ;
  DisplayAddresses ;
end.
```

1.3 Programme principal avec tableau de pointeurs (~exercices-corrigés/AddressBook2.p)

```
program AddressBook ;
  type
    StringT = varying[32] of char ;
    PersonPT = ^PersonT ;
    PersonT = record
        LastName : StringT ;
        FirstName : StringT ;
        BirthDay : integer ;
        BirthMonth : integer ;
        BirthYear : integer ;
      end ;
  var
    Addr : array[1..1000] of PersonPT ;
    NAddr : integer ;
  procedure ReadAddress ( var af : text) ;
    var
      addressP : PersonPT ;
    begin
      new (addressP) ;
      readln (af, addressP^.LastName) ;
      readln (af, addressP^.FirstName) ;
      readln (af, addressP^.BirthDay) ;
      readln (af, addressP^.BirthMonth) ;
      readln (af, addressP^.BirthYear) ;
      readln (af) ;
      NAddr := NAddr + 1 ;
      Addr[NAddr] := addressP ;
    end ;
  procedure ReadAddresses (fn : StringT) ;
    var
      af : text ;
      i : integer ;
    begin
      for i := 1 to 1000 do
        begin
          Addr[i] := nil ;
        end ;
      NAddr := 0 ;
      reset (af, fn) ;
      while not eof (af) do
        begin
          ReadAddress (af) ;
        end ;
      close (af) ;
    end ;
  procedure DisplayAddresses ;
    var
      personP : PersonPT ;
      i : integer ;
```

initialise le pointeur à non-disponible

```
  begin
    for i := 1 to NAddr do
      begin
        personP := Addr[i] ;
        writeln
          ( personP^.LastName:16
          , personP^.FirstName:16, '     '
          , personP^.BirthDay:2, '-'
          , personP^.BirthMonth:2, '-'
          , personP^.BirthYear:4
          ) ;
      end ;
  end ;
procedure SortAddresses ;
  var
    tmpP : PersonPT ;
    i, j : integer ;
  begin
    for i := 1 to NAddr do
      begin
        writeln ('Before step ', i:1) ;
        DisplayAddresses ;
        for j := 1 to NAddr-i do
          begin
            if ( Addr[j]^.LastName
               > Addr[j+1]^.LastName
               ) then
              begin
                tmpP := Addr[j] ;
                Addr[j] := Addr[j+1] ;
                Addr[j+1] := tmpP ;
              end ;
          end ;
      end ;
  end ;
procedure ClearAddresses ;
  var
    i : integer ;
  begin
    for i := 1 to NAddr do
      begin
        dispose (Addr[i]) ;
      end ;
  end ;
begin
  ReadAddresses ('ab.txt') ;
  SortAddresses ;
  writeln ('Final') ;
  DisplayAddresses ;
  ClearAddresses ;
end.
```

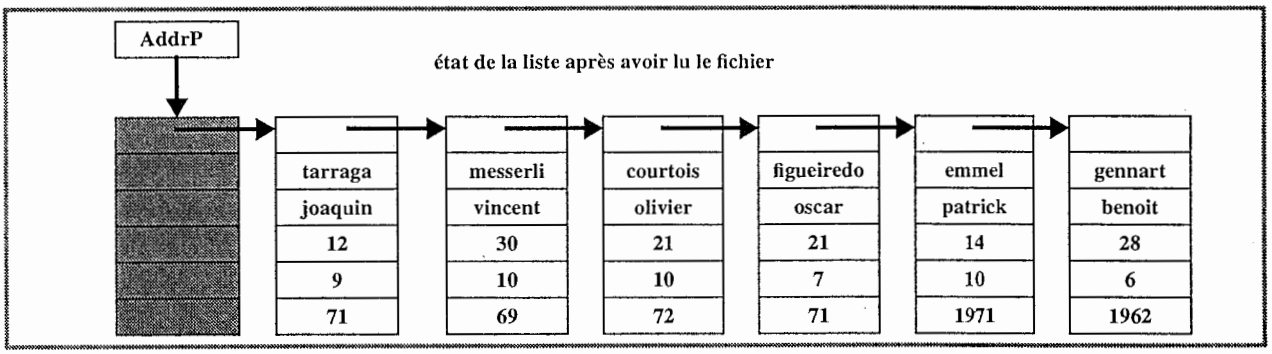
2 Routine de projection d'un point 3-D graphe vers un point 2-D écran

```
procedure ProjectToScreen (x, y, z : real ; var xi, yi : integer ) ;
  var
    xg, yg : real ;
  begin
    (* Projection from 3-D to 2-D, in graph coordinates *)
    Projection (x, y, z, xg, yg) ;
    (* Conversion from 2-D graph coordinates to 2-D screen coordinates *)
    Conversion (xg, yg, xi, yi) ;
  end ;
```

3 Tri sur structure avec listes

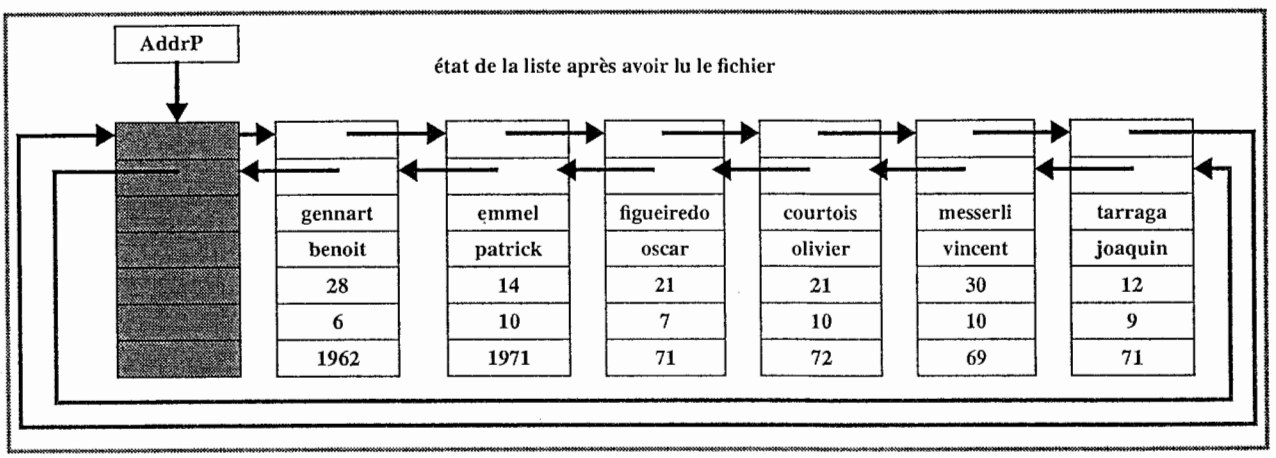
3.1 Liste à simple lien (~exercices-corrigés/AddressBook3.p)

```
program AddressBook ;
  type
    StringT = varying[32] of char ;
    PersonPT = ^PersonT ;
    PersonT = record
        NextP : PersonPT ;
        LastName : StringT ;
        FirstName : StringT ;
        BirthDay : integer ;
        BirthMonth : integer ;
        BirthYear : integer ;
      end ;
  var
    AddrP : PersonPT ;
  procedure ReadAddress ( var af : text) ;
    var pP : PersonPT ;
    begin
      new (pP) ;
      readln (af, pP^.LastName) ;
      readln (af, pP^.FirstName) ;
      readln (af, pP^.BirthDay) ;
      readln (af, pP^.BirthMonth) ;
      readln (af, pP^.BirthYear) ;
      readln (af) ;
      pP^.NextP := AddrP^.NextP ;
      AddrP^.NextP := pP ;
    end ;
  procedure ReadAddresses (fn : StringT) ;
    var af : text ;
    begin
      new (AddrP) ;
      AddrP^.NextP := nil ;
      reset (af, fn) ;
      while not eof (af) do
        begin
          ReadAddress (af) ;
        end ;
      close (af) ;
    end ;
  procedure DisplayAddresses ;
    var pP : PersonPT ;
    begin
      pP := AddrP^.NextP ;
      while (pP <> nil) do
        begin
          writeln ( pP^.LastName:16
                  , pP^.FirstName:16, ' '
                  , pP^.BirthDay:2, '-'
                  , pP^.BirthMonth:2, '-'
                  , pP^.BirthYear:4
                  ) ;
          flush ;
          pP := pP^.NextP ;
        end ;
    end ;

procedure SortAddresses ;
  var fromP : PersonPT ; i : integer ;
      personP, prevP : PersonPT ;
      maxP, maxPrevP : PersonPT ;
  begin
    fromP := AddrP ; i := 0 ;
    while (fromP^.NextP <> nil) do
      begin
        writeln ('Step ', i:1) ;
        DisplayAddresses ;
        maxPrevP := fromP ;
        maxP := fromP^.NextP ;
        prevP := maxPrevP ;
        personP := maxP ;
        while (personP <> nil) do
          begin
            if ( personP^.LastName
               > maxP^.LastName
               ) then
              begin
                maxP := personP ;
                maxPrevP := prevP ;
              end ;
            prevP := personP ;
            personP := personP^.NextP ;
          end ;
        if (fromP = AddrP) then
          begin
            fromP := maxP ;
          end ;
        (* remove maxP from the list *)
        maxPrevP^.NextP := maxP^.NextP ;
        (* put maxP at the beginning *)
        maxP^.NextP := AddrP^.NextP ;
        AddrP^.NextP := maxP ;
        i := i + 1 ;
      end ;
  end ;
begin
   ReadAddresses ('ab.txt') ;
   SortAddresses ;
   writeln ('Final') ;
   DisplayAddresses ;
end.
```



3.2 Liste à double lien (~exercices-corrigés/AddressBook4.p)

```
program AddressBook ;
  type
    StringT = varying[32] of char ;
    PersonPT = ^PersonT ;
    PersonT = record
      PrevP : PersonPT ;
      NextP : PersonPT ;
      LastName : StringT ;
      FirstName : StringT ;
      BirthDay : integer ;
      BirthMonth : integer ;
      BirthYear : integer ;
    end ;
  var (* The address book *)
    AddrP : PersonPT ;
  procedure InitializeAddresses ;
    begin
      new (AddrP) ;
      AddrP^.NextP := AddrP ;
      AddrP^.PrevP := AddrP ;
    end ;
  procedure AppendPerson
    ( var bookP, personP : PersonPT) ;
    var prevP : PersonPT ;
    begin
      prevP := bookP^.PrevP ;
      personP^.NextP := bookP ;
      prevP^.NextP := personP ;
      personP^.PrevP := prevP ;
      bookP^.PrevP := personP ;
    end ;
  procedure RemovePerson
    ( var personP : PersonPT ) ;
    var prevP, nextP : PersonPT ;
    begin
      prevP := personP^.PrevP ;
      nextP := personP^.NextP ;
      prevP^.NextP := nextP ;
      nextP^.PrevP := prevP ;
    end ;
  procedure ReadAddress ( var af : text ) ;
    var pP : PersonPT ;
    begin
      new (pP) ;
      readln (af, pP^.LastName) ;
      readln (af, pP^.FirstName) ;
      readln (af, pP^.BirthDay) ;
      readln (af, pP^.BirthMonth) ;
      readln (af, pP^.BirthYear) ;
      readln (af) ;
      AppendPerson (AddrP, pP) ;
    end ;
  procedure ReadAddresses (fn : StringT) ;
    var af : text ;
    begin
      reset (af, fn) ;
      while not eof (af) do
        begin
          ReadAddress (af) ;
        end ;
      close (af) ;
    end ;
  procedure DisplayAddresses ;
    var pP : PersonPT ;
    begin
      pP := AddrP^.NextP ;
      while (pP <> AddrP) do
        begin
          writeln ( pP^.LastName:16,
            pP^.FirstName:16, '     ',
            pP^.BirthDay:2, '-',
            pP^.BirthMonth:2, '-',
            pP^.BirthYear:4) ;flush ;
          pP := pP^.NextP ;
        end ;
    end ;
  procedure SortAddresses ;
    var minP : PersonPT ; i : integer ;
      personP : PersonPT ;
      firstP, lastP : PersonPT ;
    begin
      i := 1 ;
      firstP := AddrP^.NextP ;
      lastP := AddrP ;
      while (lastP^.PrevP <> AddrP) do
        begin
          writeln ('Before step ', i:1) ;
          DisplayAddresses ;
          personP := AddrP^.NextP ;
          minP := personP ;
          while (personP <> lastP) do
            begin
              if ( personP^.LastName
                 < minP^.LastName
                 ) then
                begin
                  minP := personP ;
                end ;
                personP := personP^.NextP ;
            end ;
          RemovePerson (minP) ;
          AppendPerson (AddrP, minP) ;
          if (lastP = AddrP) then
            begin
              lastP := minP ;
            end ;
          i := i + 1 ;
        end ;
    end ;
  begin
    InitializeAddresses ;
    ReadAddresses ('ab.txt') ;
    SortAddresses ;
    writeln ('Final') ;
    DisplayAddresses ;
  end.
```



Programmation I : Examen Final

Physiciens

Comment s'y prendre

Au début de l'examen, tapez la commande ~gennart/StartExam dans une fenêtre terminal. Il y a 4 exercices pour lesquels vous devez écrire un programme PASCAL. Remettez les fichiers sources et les exécutables dans votre répertoire ~/lsp-rendu. ***Mettez votre nom en commentaire dans chacun de vos programmes***. À la fin de l'examen, tapez la commande ~gennart/FinishExam, et demandez à un assistant s'il peut retrouver vos fichiers.

L'examen comporte est sur 60 points et comporte 3 exercices. Faire les 3 exercices rapporte 18 points par exercice pour un total de 54 points. Pour obtenir les 6 derniers points, vous avez trois possibilités : (1) combiner les 3 exercices (lire dans un fichier une liste de triangles, les trier, et les afficher en utilisant la routine FillTriangle ; (2) utiliser les pointeurs dans au moins un des exercices proposés ; (3) faire tourner la figure de l'exercice 3 autour d'un axe quelconque.

L'enregistrement Triangle

```
type TriangleT = record
  x0, y0, z0 : real ;
  x1, y1, z1 : real ;
  x2, y2, z2 : real ;
end ;
```

L'enregistrement suivant modélise une triangle dans l'espace (trois sommets du triangle, chacun avec trois coordonnées). Vous devez l'utiliser dans les programmes suivants.

1 Entrées-sorties

Prenez dans le répertoire ~gennart/ExamenFinalPhysique tous les fichiers *.txt. Ils contiennent des lignes de 9 chiffres représentant un triangle dans l'espace. Ecrivez un programme qui lit le contenu d'un des fichiers et stocke son contenu dans un ensemble de variables de type TriangleT (ensemble veut dire : tableau ou liste). Pour vérifier que cela fonctionne, afficher en utilisant writeln le contenu des variables TriangleT à l'écran, et comparez vos résultats avec l'exécution du programme ~gennart/ExamenFinalPhysique/efp1 (**efp1.p**, 18 points)

2 Trier une liste de triangle

On demande d'écrire une programme qui initialise un ensemble de 20 triangles avec des valeurs aléatoires, et les trie ensuite avec le critère suivant : moyenne (Triangle[i].x0, Triangle[i].x1, Triangle[i].x2) < moyenne (Triangle[i+1].x0, Triangle[i+1].x1, Triangle[i+1].x2), pour toutes les valeurs de i. Affichez les valeurs des 20 triangles avant et après le tri. Comparez avec le programme ~gennart/ExamenFinalPhysique/efp2 dont le générateur de nombres aléatoires est initialié avec x := seed (1000))(**efp2.p**, 18 points).

3 Afficher une liste de triangles

On demande d'écrire une programme qui initialise un ensemble de 20 triangles avec les valeurs suivantes :

```
Triangle[i].x0:=cos(i*Pi/10); Triangle[i].y0:=sin(i*Pi/10); Triangle[i].z0:=-1.0;
Triangle[i].x1:=cos((i+1)*Pi/10); Triangle[i].y1:=sin((i+1)*Pi/10); Triangle[i].z1:=-1.0;
Triangle[i].x2:=cos((2*i+1)*Pi/20); Triangle[i].y2:=sin((2*i+1)*Pi/20); Triangle[i].z2:=1.0;
```

En utilisant une routine de projection 3D vers 2D et la routine de conversion de coordonnées de graphe en coordonnées d'écran, transformez le triangle en 3-dimensions en un triangle plan en coordonnées d'écran. Utilisez la routine `FillTriangle (x0, y0, x1, y1, x2, y2)` que j'ai ajoutée dans Graphics.h pour afficher les 20 triangles.

Pour que cela fonctionne, il faut donner à chaque triangle une couleur. Pour cela, on initialise une table de couleurs en appelant au début du programme la routine `DefaultColorMap;` (nouvelle routine dans Graphics.h). Déclarez aussi dans votre programme une routine `function FindDefaultColor (var t: TriangleT):integer;external;`. Vous pouvez maintenant trouver la couleur d'un triangle en écrivant l'instruction `color:=FindDefaultColor(t);`. La figure que vous obtiendrez doit ressembler à la figure que vous obtenez en exécutant le programme ~gennart/ExamenFinalPhysique/efp3. (**efp3.p**, 18 points).

Editeur de texte : XEMACS

PASCAL

1 L'environnement de programmation

Cette section est essentielle pour apprendre le vocabulaire nécessaire à la programmation. Les mots qui apparaissent pour la première fois sont écrits en italique. Ils sont expliqués dans la suite du texte. Elle vous décrit votre ordinateur et ses logiciels, dans l'ordre dans lequel vous les utiliserez au cours d'une session. Une session est tout ce que vous faites du moment ou vous asseyez devant l'ordinateur jusqu'au moment où vous le quittez.

1.1 Le matériel

Autrement dit *hardware*. L'ordinateur à votre disposition est une *station de travail* Sparc 4, produite par Sun MicroSystems. Elle dispose d'un écran couleur 20 pouces (50cm de diagonale), d'un clavier et d'une souris, ainsi bien entendu que d'une unité de traitement (la boite en dessous de l'écran). L'ordinateur est toujours allumé. Vous ne devez pas l'éteindre, il se met tout seul en veille. Pour l'activer, il suffit de bouger la souris ou d'enfoncer une touche du clavier. Après une dizaine de secondes, l'écran aura chauffé, et vous affichera une fenêtre horizontale vous demandant votre nom. Nous y reviendrons plus tard (section 1.3).

L'unité de traitement comporte un microprocesseur Sparc, 64MOctets de mémoire, et un disque de 500MOctets. Il y a 90 stations à votre disposition. Les stations sont connectées à deux serveurs et 4 imprimantes. Les serveurs contiennent toutes les données de tous les utilisateurs. Vous pouvez donc vous connecter sur n'importe quelle station : il vous semblera que vos données (par exemple vos programmes Pascal, ou des informations que vous aurez été chercher sur le réseau Internet) y sont. Vous ne devez pas sauver vos données sur une disquette, vous les retrouvez d'une session à l'autre.

1.2 Le logiciel

Autrement dit *software*. Dans la section précédente, nous avons décrit le *matériel* informatique que vous allez utiliser. Cette section décrit les *logiciels*. Le matériel est ce que vous pouvez toucher et qui a du poids. Le logiciel est le reste.

Pour ce cours, il y a trois outils logiciels qui sont essentiels : le *système d'exploitation*, l'*éditeur*, et le *compilateur*. D'autres sont accessoires, comme l'outil de traitement du courrier, ou Netscape, l'outil de navigation sur le réseau Internet.

1.3 Le système d'exploitation

C'est le logiciel qui tourne en permanence sur votre ordinateur. C'est lui qui vous accueille lorsque vous vous mettez à la console. Il vous permet de vous *loguer (log in)*, c'est-à-dire, de donner votre nom (suivi de la touche RETURN, celle à droite de votre clavier qui porte le signe ↵) et mot de passe (suivi de la touche RETURN) qui permet à l'ordinateur de vous placer dans un *environnement* à vous. Un environnement représente toutes les informations propres à un utilisateur. Il permet aussi à l'ordinateur de vous empêcher de détruire les environnements des autres utilisateurs, et les informations nécessaires au bon fonctionnement de l'ordinateur.

Un petit commentaire à propos du mot de passe. Il n'est bien entendu pas affiché à l'écran lorsque vous le tapez. Autre commentaire : si vous faites une erreur de frappe, utilisez la touche 'supprimer' (à droite, entre le clavier alphabétique et le clavier numérique). Enfin, majuscules et minuscules sont des lettres différentes ; faites attention à ne pas confondre le O (o majuscule) et le 0 (zéro), le l (lettre minuscule l) et le 1 (chiffre un).

Une fois logué, l'ordinateur vous ouvre un écran qui comporte un *panneau de commande* au bas de l'écran. Ignorez pour le moment la fenêtre Terminal qui apparaît aussi, nous y reviendrons plus tard. Ce panneau de commande permet de lancer de nombreuses *applications*, que nous allons détailler. Les applications rendent différents services à l'utilisateur, des plus simples (vous donner l'heure ou vous prévenir quand du courrier arrive) aux plus compliqués (édition de texte, compilation, navigation sur le réseau).

Le panneau de commande

Passons en revue le panneau de commande (Figure 1). Le panneau de commande comporte une série d'*icônes* et de *boutons*. Tant les icônes que les boutons sont actifs : on peut lancer une application en double-cliquant avec le bouton gauche de la souris. A l'extrême gauche, l'horloge et le calendrier. Ensuite, l'outil de gestion des *fichiers* (*file* en Anglais). Tout ce dont l'ordinateur se souvient entre deux sessions est mémorisé dans des fichiers. Les applications sont contenues dans des fichiers, les programmes pascal que vous écrirez sont contenus dans des fichiers, le courrier que vous recevez est un fichier. Un fichier contient des informations qui subsistent non seulement entre deux sessions, mais aussi quand la station de travail et les serveurs sont éteints.

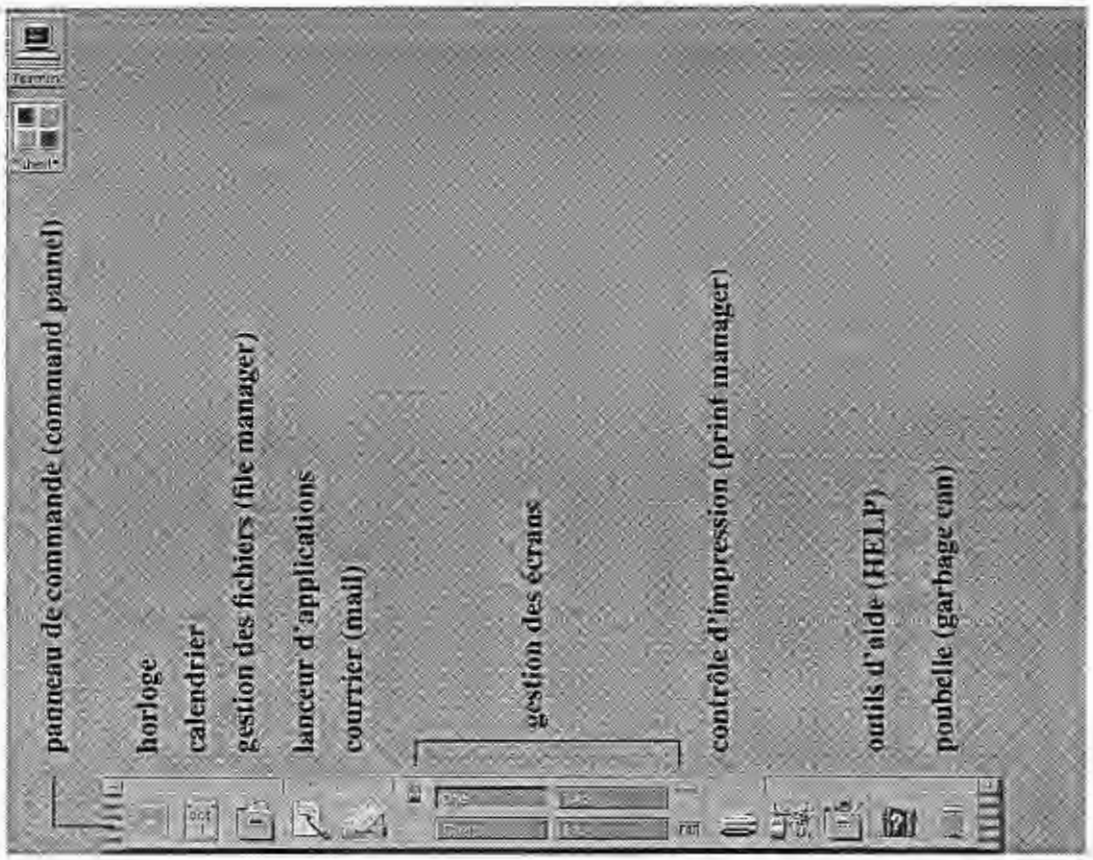


FIGURE 1. L'écran de votre station après le login

Les fenêtres

En double-cliquant avec le bouton gauche de la souris sur l'icône de gestion de fichiers, vous ouvrirez une *fenêtre* de gestion de fichiers (Figure 2). Intéressons nous d'abord à cette fenêtre elle même, car elle autorise toute une série de commandes communes à la plupart des fenêtres que vous ouvrirez par la suite.

Les 4 coins permettent d'en changer la taille : en pressant le bouton gauche de la souris sur le coin choisi, et en traînant la souris avec le bouton enfoncé dans la direction de votre choix, vous changez la taille de la fenêtre). La barre de titre de la fenêtre permet de déplacer la fenêtre (en pressant le bouton gauche de la souris sur la barre de titre, et en traînant la souris avec le bouton enfoncé dans la direction de votre choix, vous déplacez la fenêtre). L'ascenseur vertical à la droite de la fenêtre vous permet de faire défiler son contenu, si toutefois le contenu de la fenêtre est plus grand que la fenêtre. Dans le cas de la Figure 2, mon répertoire contient 89 fichiers, et l'ascenseur est nécessaire. Il est probable que votre répertoire soit vide. La fenêtre n'aura donc pas d'ascenseur.

Les deux petits boutons à droite de la barre de titre vous permettent d'iconifier la fenêtre (l'icône se mettra au sommet gauche de votre écran), et de faire occuper tout l'écran à (maximiser) la fenêtre. Pour rouvrir la fenêtre iconifiée, double-cliquez sur l'icône. Pour minimiser la fenêtre (la faire revenir à sa taille normale), cliquez à nouveau sur le bouton de maximisation/minimisation.

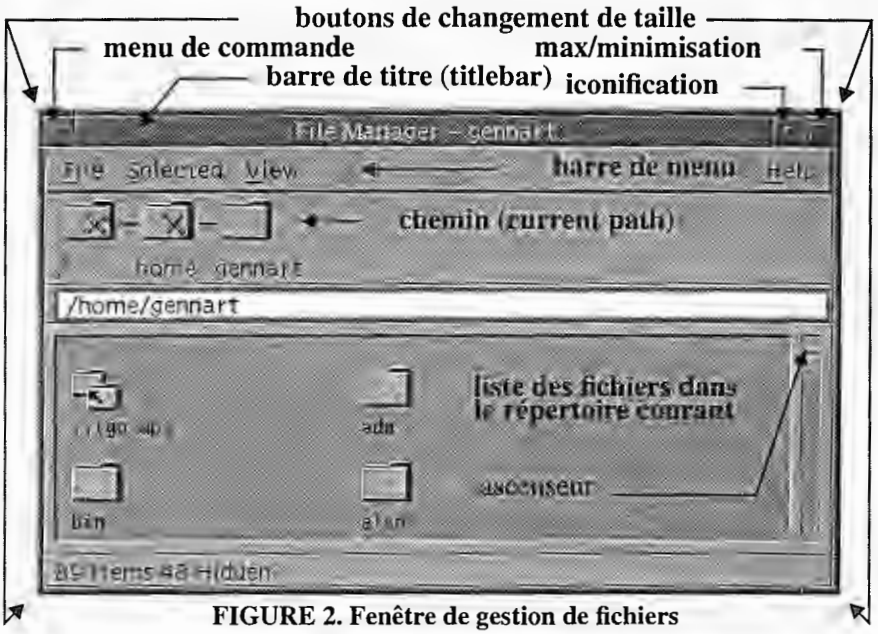


FIGURE 2. Fenêtre de gestion de fichiers

Le bouton à gauche de la barre de titre permet de sélectionner un menu de commande de la fenêtre (Figure 3). Ce menu comporte une liste de commandes que l'on peut effectuer sur une fenêtre. Chaque ligne du menu comporte le nom de la commande et optionnellement son *raccourci clavier*. Par exemple pour la commande 'move', le raccourci clavier est Alt-F7, c'est-à-dire enfoncer la touche Alt, et en maintenant la touche Alt enfoncée, presser la touche F7.

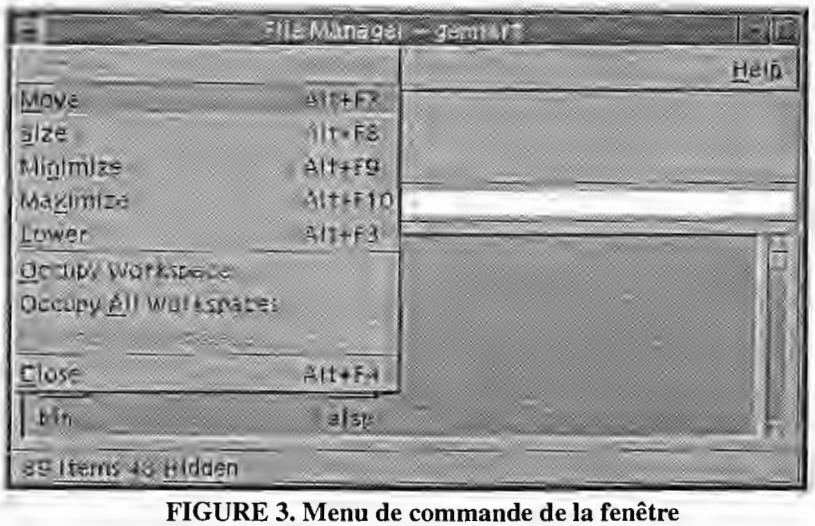


FIGURE 3. Menu de commande de la fenêtre

Vous connaissez maintenant 3 façons de bouger une fenêtre : la barre de titre, le menu et le raccourci clavier. Les menus sont très utiles pour apprendre les possibilités de votre ordinateur. Les commandes directes à la souris ou les raccourcis clavier sont cependant beaucoup plus rapide une fois qu'on a de l'habitude.

Une chose à savoir est que vous avez en fait 4 écrans complets à votre disposition. C'est la signification des boutons one-two-three-four dans le panneau de commande. Pour le moment vous êtes dans l'écran un. Si vous

cliquez sur le bouton Two, vous passerez à l'écran deux, etc. Chacun des écrans est appelé un *workspace*. Si vous souhaitez faire passer une fenêtre d'un écran à l'autre, sélectionner la commande Occupy Workspace dans le menu de fenêtre. Vous pouvez alors choisir dans quel écran vous voulez que votre fenêtre apparaisse.

Si vous souhaitez en savoir davantage sur l'utilisation de votre écran, vous pouvez consulter l'outil d'aide (Figure 4). Dans ce menu, sélectionnez la commande 'desktop introduction', qui vous donnera des informations complètes sur les possibilités du gestionnaire d'écran Sun (Figure 5).

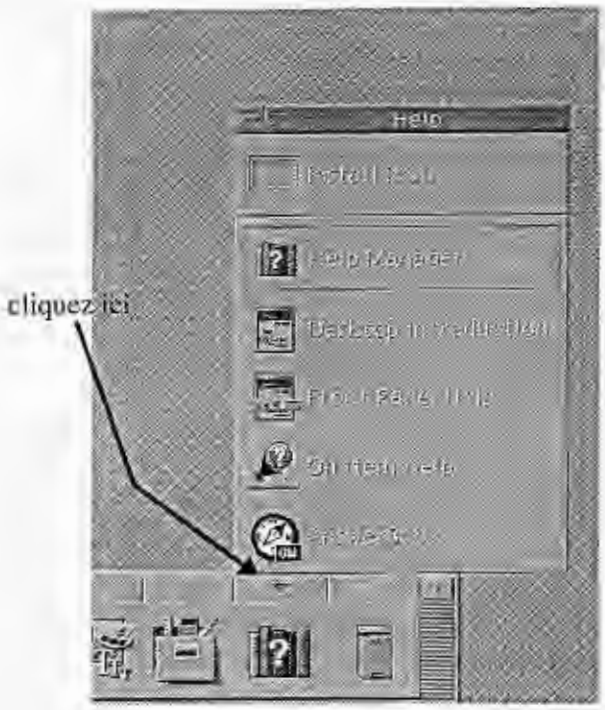


FIGURE 4. Menu d'aide, dans le panneau de commande

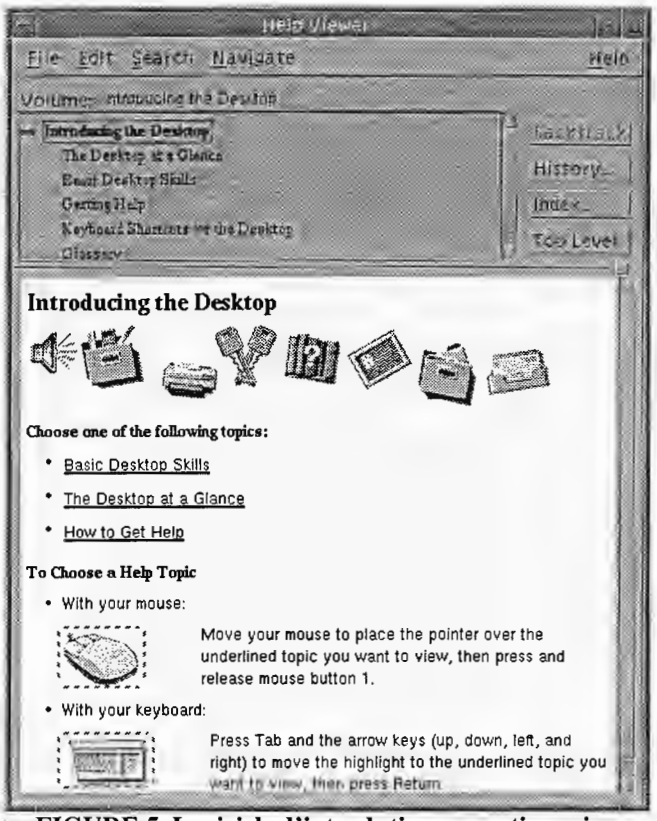


FIGURE 5. Logiciels d'introdution au gestionnaire d'écran

Gestion des fichiers

Les fichiers sont organisés hiérarchiquement en répertoires (*folder* or *directory*). Le gestionnaire de fichier s'ouvre normalement sur votre répertoire qui s'appelle '/home/votre_nom'. Le premier '/' (*slash* en Anglais) représente la racine de tous les répertoires, les autres '/' représentent les séparations entre les différents niveaux de la hiérarchie de répertoires. Un nom de répertoire d'appelle un chemin (*current path* en anglais). Il n'y a pas encore grand chose dans votre répertoire, mais en double-cliquant sur la première icône du chemin, vous pouvez remonter jusqu'à la racine de tous les fichiers. Double-cliquez ensuite sur le répertoire 'usr'. A l'intérieur du répertoire usr, double-cliquez sur le répertoire openwin. A l'intérieur du répertoire openwin, sélectionnez le répertoire demo. Cherchez dans le répertoire l'application xeyes (Figure 6).

Xeyes n'est pas un répertoire (son icône est différente). Il s'agit d'une application qui dessine une paire d'yeux sur l'écran, et les yeux suivent le curseur (le mouvement de la souris). Pour lancer l'application, double-cliquez sur le fichier xeyes. Une fenêtre permettant de spécifier les options et les arguments apparaîtra (Figure 7). Pour référence les arguments et les options permettent de modifier le comportement de l'application que vous allez lancer. Vous pouvez ignorer cette fenêtre et enfoncer le bouton OK. La fenêtre d'arguments disparaît, et deux fenêtres apparaissent. L'une appelée Run récupère tous les messages textes envoyés par l'application, et l'autre est la fenêtre d'application elle-même (Figure 8).

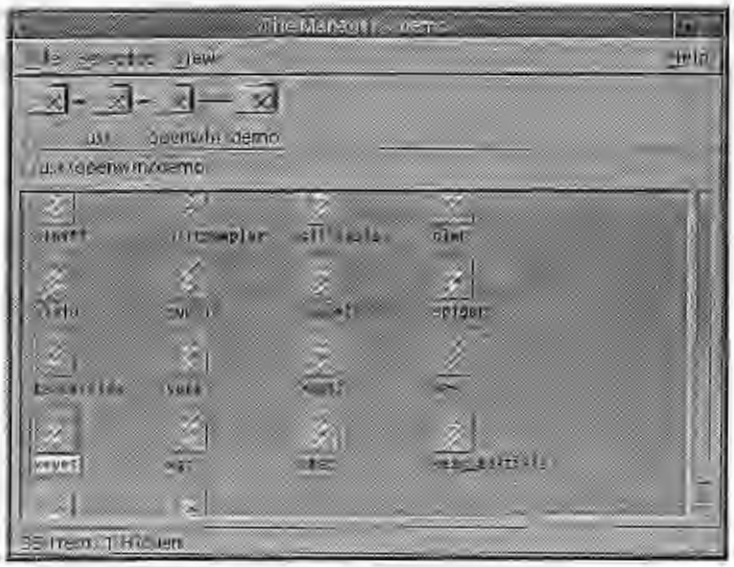


FIGURE 6. Le répertoire /usr/openwin/demo

Un exemple d'option est -geometry 400x400+100+500. Ceci vous permet de lancer un xeyes avec une taille de 400x400 pixel (l'écran fait 1280 par 960), aux coordonnées (100, 500). Les coordonnées (0,0) d'écran sont au coin supérieur gauche. En bougeant de 100 à droite et de 500 vers le bas, on arrive au milieu gauche de l'écran. Le coin supérieur gauche de l'application xeyes aura pour coordonnées (100, 500).

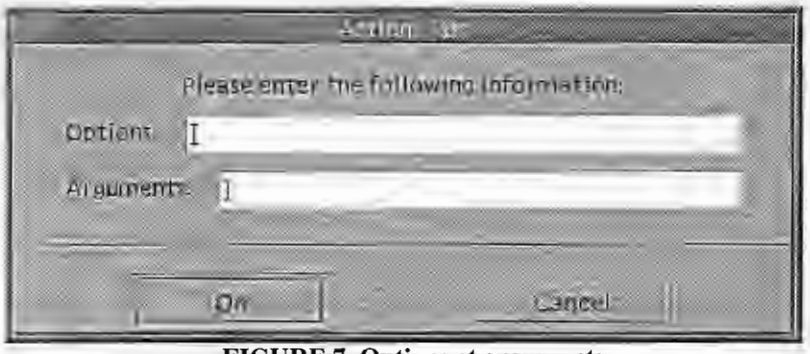


FIGURE 7. Options et arguments

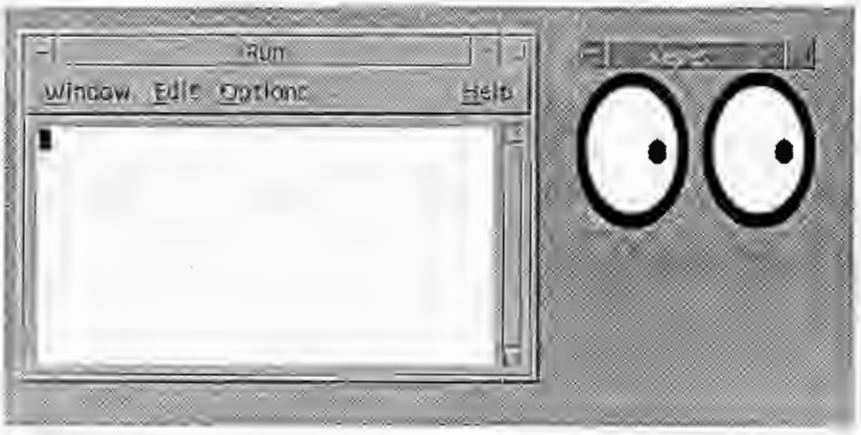


FIGURE 8. L'application xeyes

Pour quitter l'application xeyes, sélectionnez dans le menu de commande des fenêtres 'run' et 'xeyes' la commande close.

Pour visiter mon répertoire (/home/gennart), c'est un peu plus compliqué. Si vous allez dans /home vous verrez simplement votre répertoire. Les répertoires des autres utilisateurs sont cachés. Sélectionnez le menu File/ GoTo, et indiquez le nom de répertoire /home/gennart. Vous vous retrouverez dans mon répertoire. Double-cliquez par exemple sur le fichier saturn.im8.

Nous allons maintenant copier l'application xeyes dans votre répertoire (/home/votre_nom). Cette opération est particulièrement utile. Elle vous permettra (entre autres) de remettre vos examens. Retournez d'abord à l'aide du gestionnaire de fichier dans le répertoire /usr/openwin/demo, et ouvrez une deuxième fenêtre de gestion de fichiers. Pour cela, deux solutions : sélectionnez dans la barre de menu de la fenêtre de gestion de fichier le menu 'View/Open New View' (Figure 9) ; ou double-cliquez sur l'icône 'gestion de fichier' dans le panneau de commande au bas de votre écran.

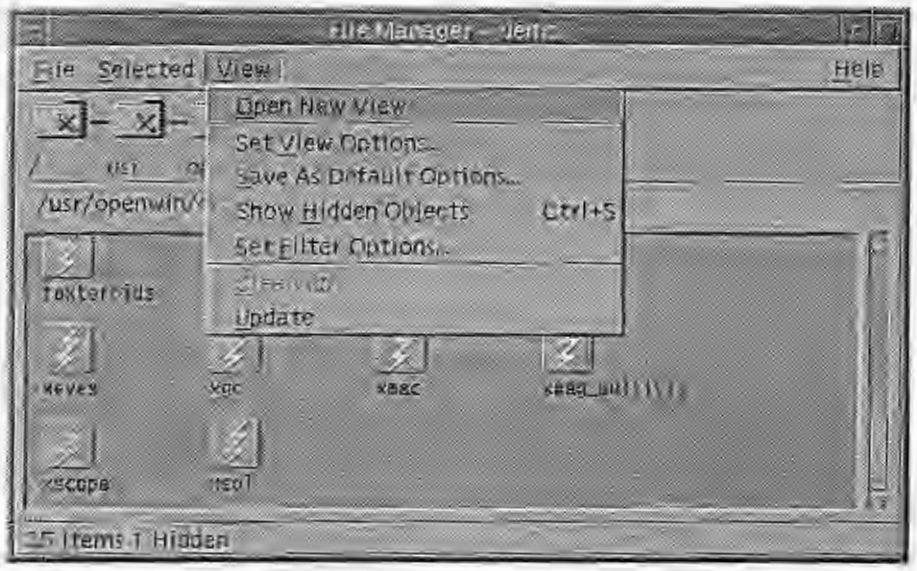


FIGURE 9. Menu d'ouverture d'une nouvelle fenêtre de gestion de fichiers

Arrangez-vous pour que la première fenêtre de gestion de fichiers soit dans le répertoire ou se trouve xeyes (/ usr/openwin/demo) et l'autre dans votre répertoire de base (/home/votre_nom). Vous devriez vous trouver dans la situation de la Figure 10. Pour copier le fichier xeyes, enfoncez la touche Control (et maintenez la enfoncée), cliquez sur le fichier, maintenez le bouton enfoncé, et traînez l'icône jusque dans votre répertoire.

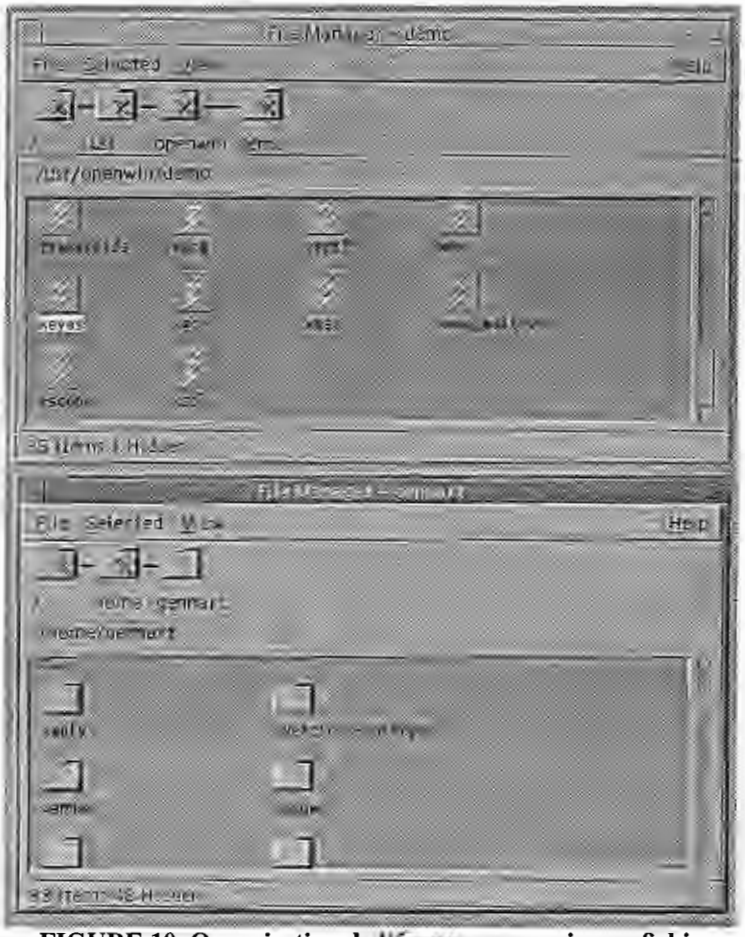


FIGURE 10. Organisation de l'écran pour copier un fichier

Si vous n'enfoncez pas la touche Control, vous essayez alors non seulement de copier le fichier dans votre répertoire local, mais aussi de l'effacer du répertoire original. Cette opération de copie/effacement s'appelle un transfert. Le fichier xeyes étant dans un répertoire qui ne vous appartient pas, vous obtiendrez le message d'erreur qui apparaît à la Figure 11.

FIGURE 11. Message d'erreur lors du transfert d'un fichier

Lanceur d'applications

Pour sélectionner le menu du lanceur d'application, cliquez sur la flèche au-dessus de l'icône 'lanceur d'application' (Figure 12). Ce menu est une liste de 4 applications : un éditeur de texte, un terminal, un éditeur d'icônes, et un visualisateur d'image. Sélectionnez la commande terminal.

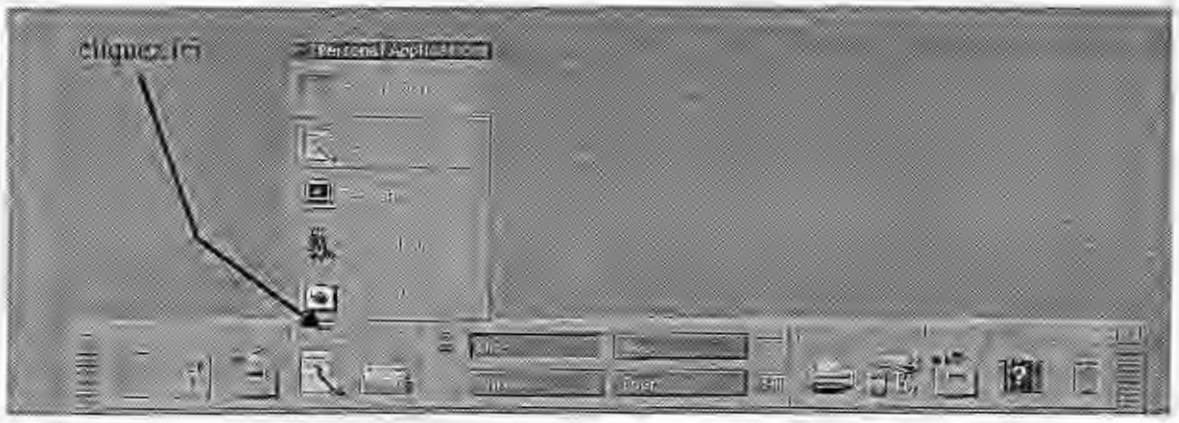


FIGURE 12. Menu du lanceur d'application

Un terminal est une fenêtre qui permet d'exécuter des commandes tapées à la main (Figure 13). La liste des commandes est très longue, et la plupart peuvent être effectuées à l'aide du gestionnaire de fichier. Cependant, pour le programmeur expérimenté, le clavier est beaucoup plus rapide que la souris et l'interface graphique. Je vous explique donc les commandes terminal les plus courantes.

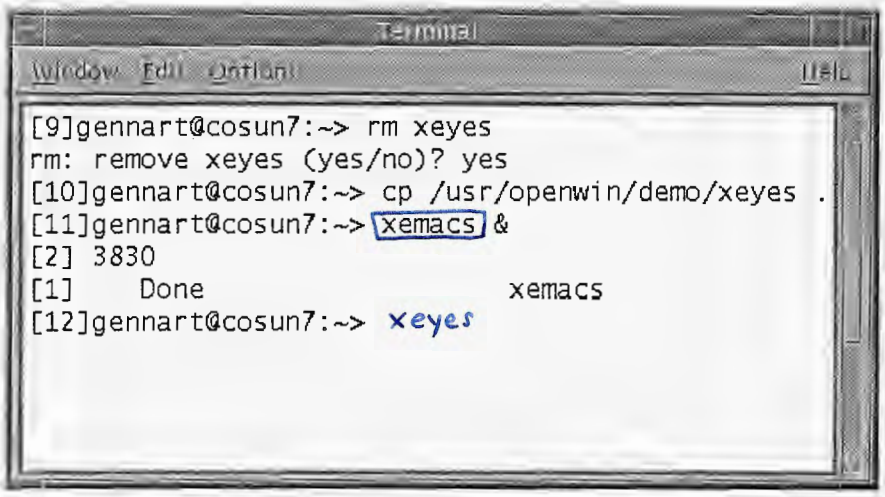


FIGURE 13. La fenêtre terminal

La fenêtre 'Terminal' possède, comme la fenêtre du gestionnaire de fichier une barre de contrôle supérieure, un bouton d'iconification à l'extrême gauche de la barre de contrôle, un ascenseur sur la gauche de la fenêtre. Les commandes pour bouger la fenêtre, changer sa taille sont bien entendu identiques.

Vous pouvez changer la taille des caractères en sélectionnant le menu Options/FontSize (Figure 14).

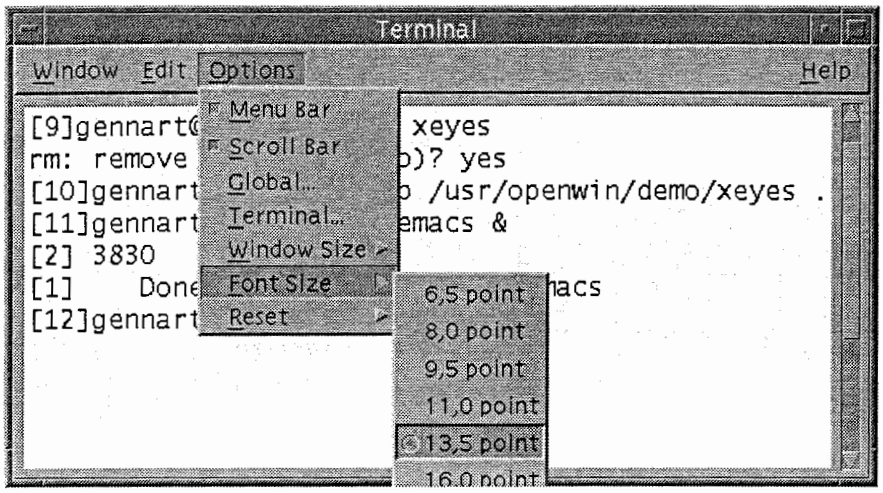


FIGURE 14. Changer la taille des caractères de la fenêtre terminal

Pour exécuter une commande, taper la commande, suivi de la touche Return. Dans les exemples suivant, les caractères gras sont ceux que l'utilisateur tape. Les autres sont produits par votre station de travail. Voici une liste brève des commandes disponibles :

- pwd: Print working directory. Afficher à l'écran le nom du répertoire courant.

```
cosun12% pwd
/home/gennart
cosun12%
```

  Par exemple, la commande ci-dessus vous indique que lorsque j'ai exécuté cette commande, j'étais dans le répertoire home, sous-répertoire gennart.

- mkdir : make directory. Créer un nouveau sous répertoire

```
cosun12% mkdir pascal
cosun12%
```

- ls : list file. Afficher à l'écran la liste de tous les fichiers disponibles dans le répertoire courant.

```
cosun12% ls
pascal
cosun12%
```

- ls /usr/openwin/demo : Afficher à l'écran la liste de tous les fichiers disponibles dans le répertoire /usr/openwin/demo.

```
cosun12% ls /usr/openwin/demo
add             fpadd           muncher         realxfishdb     xgc
dpsexec         fpls            olbiff          spider          xmac
fish_props      fprm            olitsampler     texteroids      xmag_multivis
fontadd         ico             olittable       worm            xscope
fontadmin       ico2            olmh            xepsf           xsol
fontls          kcms            plaid           xev
fontrm          maze            puzzle          xeyes
cosun12%
```

  La fenêtre Terminal complète automatiquement les noms non ambigus de répertoire ou de fichier. Par exemple, si vous tapez 'ls /usr/ope' suivi de la touche TAB, la fenêtre terminal vous donnera automatiquement le reste du nom. Vous verrez donc apparaître le nom du répertoire complété : 'ls /usr/openwin/'. La fenêtre terminal se souvient des commandes que vous avez utilisées précedemment. Vous pouvez les rappeler en utilisant la touche 'flèche vers le haut', suivi de la touche Return.

- cd (change directory) : changer le répertoire courant .

```
cosun12% cd ..
/home
cosun12% pwd
/home
cosun12% cd gennart/pascal
/home/gennart/pascal
cosun12% cd /home/gennart
/home/gennart
cosun12%
```

La commande `cd ..` vous fait passer dans le répertoire parent. La commande `cd gennart/pascal` vous fait passer dans le sous-répertoire gennart, sous-sous-répertoire pascal. La commande `cd /home/gennart` vous fait passer au répertoire /home, sous-répertoire gennart.

La notation /home/gennart/pascal décrit un chemin (path) dans la hiérarchie des répertoires. C'est un chemin *absolu*. On part de la racine (le premier / (slash)), on prend le répertoire home, le sous-répertoire gennart, et le sous-sous-répertoire pascal.

La notation gennart/pascal décrit un chemin relatif dans la hiérarchie des répertoires. Cela veut dire, à partir du répertoire courant, prendre le sous-répertoire gennart, sous-sous-répertoire pascal. Cela ne veut dire quelque chose que si l'on est dans un répertoire ou il y a effectivement un sous-répertoire gennart *qui* contient un sous-sous répertoire pascal.

- rm (remove file) : effacer un fichier

```
cosun12% rm xeyes
rm: remove xeyes (yes/no) ? yes
cosun12%
```

rm permet d'effacer les fichiers devenus inutiles. rm vous demande de confirmer pour chaque fichier que vous voulez l'effacer. Vous pouvez aussi effacer un fichier en traînant un fichier du gestionnaire de fichiers jusqu'à la poubelle qui se trouve à l'extrême droite du panneau de commande.

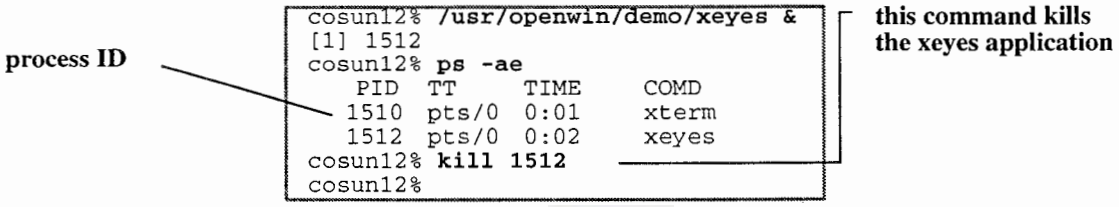
- Lancer une application :

```
cosun12% /usr/openwin/demo/xeyes &
cosun12% xterm &
cosun12%
```

Pour lancer une application, on utilise simplement le nom de l'application, suivi de &. Le & sert à *rendre la main* au terminal. Si vous oubliez le &, le terminal refuse toutes les commandes jusqu'à ce que l'application que vous avez lancée soit terminée.

Si vous êtes coincé (la fenêtre ne réagit plus ni à la souris ni au clavier), mettez le curseur d'écran au dessus de la fenêtre 'Terminal', cliquez, essayez de taper C-q (touche CONTROL enfoncée, touche q), ou bien C-c (, interruption, touche CONTROL enfoncée, touche c). Cela devrait vous remettre dans une situation o ù vous pouvez travailler.

Pour faire la liste de toutes les applications en cours, tapez la commande ps. Cela vous donne par exemple :



Une dernière commande qui a son utilité est la commande passwd, qui permet de changer son mot de passe. Vous devrez préciser l'ancien mot de passe et deux fois le nouveau.

1.4 L'éditeur

L'éditeur permet de créer de nouveaux fichiers, et modifier le contenu de fichiers existants. L'éditeur que nous allons employer est XEMACS. C'est un éditeur assez universel (utilisé par beaucoup de monde, aux Etats-Unis, au Japon, en Europe, et ailleurs), mis à disposition gratuitement par le MIT (Massachusetts Institute of Technology). Pour lancer xemacs, deux possibilités : (1) utiliser la commande

```
cosun12% xemacs &
```

dans une fenêtre terminal, ou (2) sélectionner la commande xemacs dans le lanceur d'applications. Cette deuxième possibilité peut ne pas être disponible sur votre machine.

N'oubliez pas le & (e commercial, ampersand). Un petit commentaire général à propos d'EMACS. Les possibilités d'EMACS sont énormes. On ne peut espérer tout connaître, même en quelques mois de travail intensif. Cependant, il y a moyen de faire les opérations habituelles de façon simple.

Le système d'exploitation va vous ouvrir une fenêtre qui contient l'éditeur. Cette fenêtre a un comportement assez différent des fenêtres 'terminal' et 'gestion de fichier' que vous avez utilisées jusqu'à présent. Elle permet de créer un texte, de modifier un texte, et de le sauver dans un fichier (Figure 16). En plus de la barre de contrôle de la fenêtre, elle comporte une barre de menus, une barre d'icône, une barre inférieure de contrôle qui vous donne des indications sur le fichier que vous êtes en train d'éditer, et un mini-buffer qui vous permet de spécifier les paramètres des commandes xemacs qui en ont besoin.

Ceci peut avoir l'air compliqué, mais vous utiliserez essentiellement la barre de boutons.

Première opération, ouvrir un fichier. Cliquez sur le bouton Open (à l'extrême gauche de la barre de boutons), et tapez le nom du fichier que vous souhaitez créer : dans notre cas 'ulysse.txt' (Figure 15).

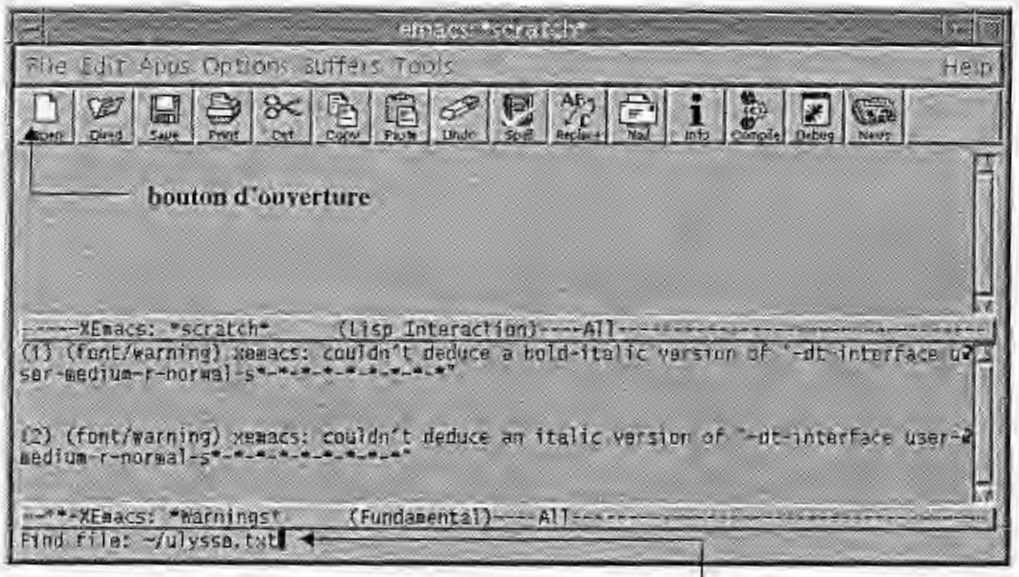


FIGURE 15. Ouvrir un fichier

Un fichier est caractérisé par un nom et un contenu. Le nom d'un fichier comporte typiquement deux parties, séparées par un point : un identificateur décrivant le contenu du fichier, et une *extension* indiquant le type de fichier. Par exemple, le nom du fichier `FirstProgram.p` suggère qu'il contient votre premier programme, et qu'il est écrit en Pascal (extension .p). Pour ulysse.txt, le nom suggère que le texte parlera d'Ulysse, et qu'il s'agit de texte. Le contenu d'un fichier est généralement du texte, mais il peut aussi être de l'image, ou même une application.

1. ouvrir un fichier : demande un nom de fichier

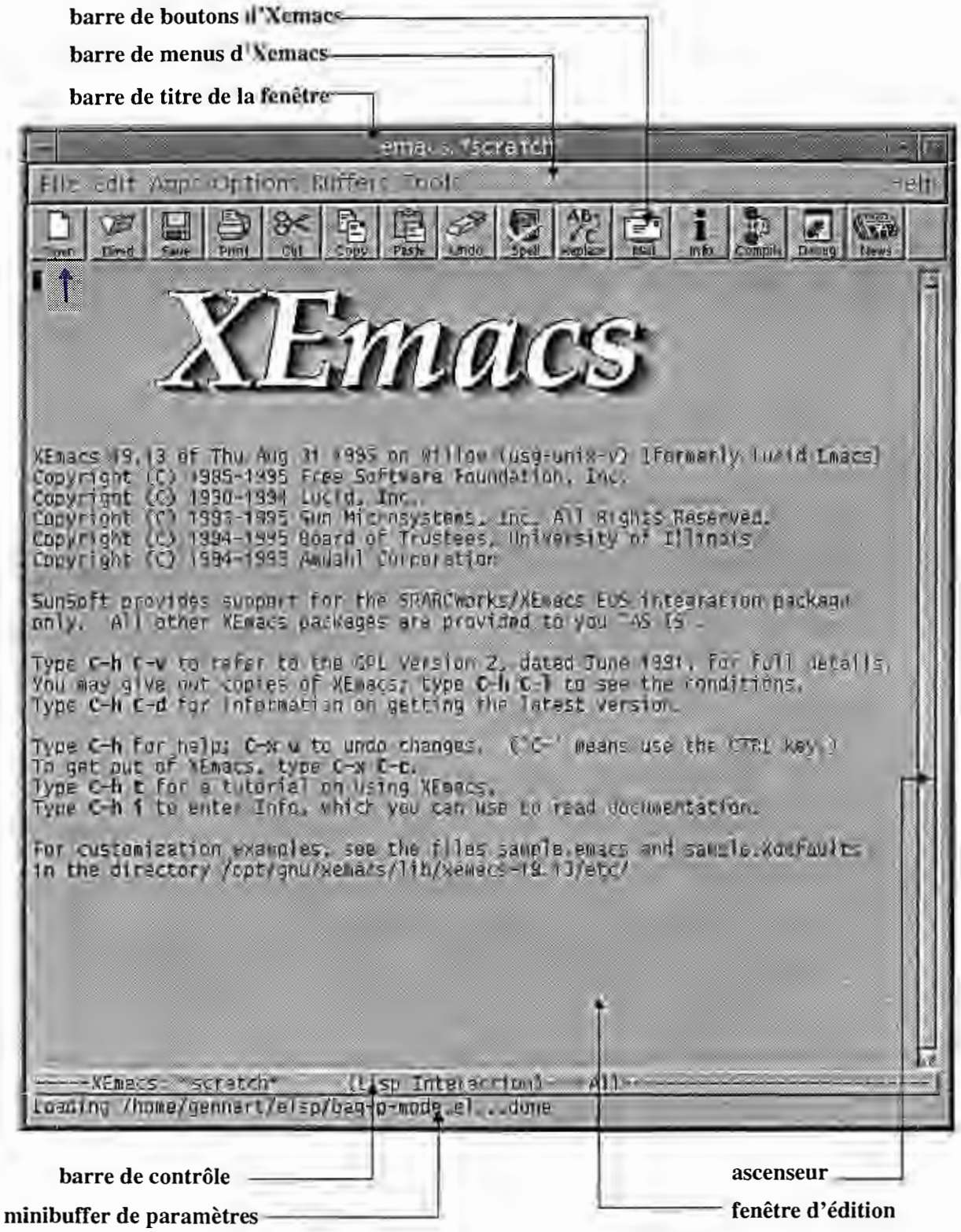


FIGURE 16. La fenêtre Xemacs

Si, après avoir ouvert le fichier ulysse.txt, la fenêtre xemacs est divisé en deux parties (comme cela m'est arrivé), sélectionnez la commande 'Un-split (keep this)' dans le menu File (Figure 17).

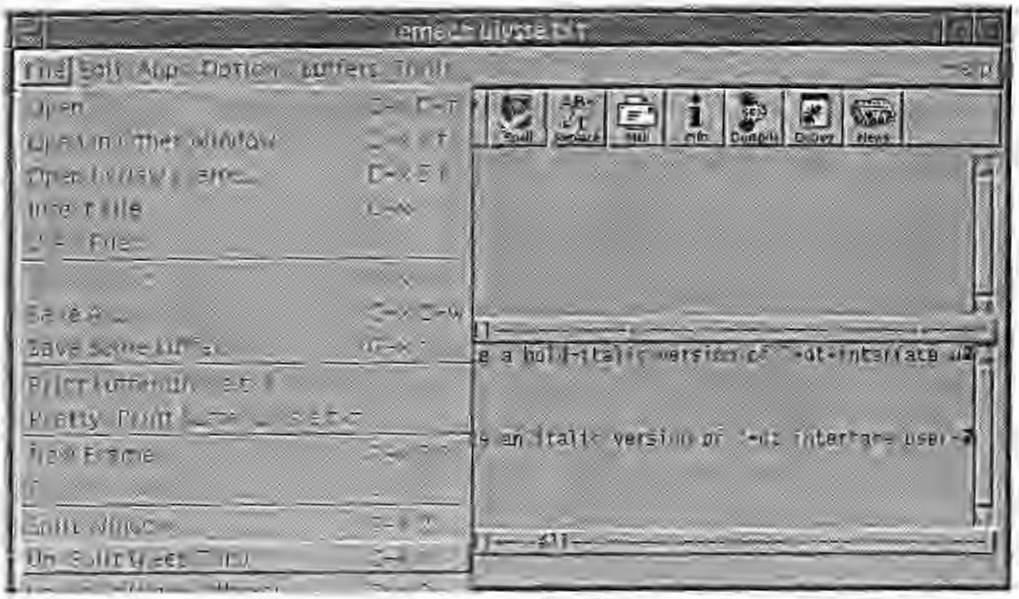


FIGURE 17. Comment se débarasser de la deuxième fenêtre

Tapez dans la fenêtre principale

```
Heureux qui comme Ulysse
a fait un beau voyage
```

Pour passer à la ligne, utilisez la touche RETURN. Vérifiez l'utilisation des flèches de contrôle, en vous baladant à travers votre texte. Vous devriez maintenant vous trouvez dans la situation de la Figure 18.

FIGURE 18. Edition du fichier ulysse.txt

Pour sauver le fichier, utilisez le bouton Save. Pour vérifier que le fichier a bien été sauvé, mettez vous dans une fenêtre terminal, tapez la commande 'ls', et vérifiez que le fichier ulysse.txt apparaît bien dans la liste des fichiers. Alternativement, vous pouvez utiliser le gestionnaire de fichier pour vérifier que le fichier a bien été sauvé. Si vous quittez maintenant xemacs, votre fichier continuera d'exister.

Ouvrez maintenant un deuxième fichier, first.p, qui contiendra votre premier programme en Pascal. Tapez dans ce fichier le texte suivant :

```
program first ;
  begin
    writeln ('hello, world') ;
  end.
```

Sauvez maintenant le fichier, et vérifiez qu'il est effectivement bien sauvé.

Pour ouvrir et sauver un fichier, il existe deux alternative aux boutons : les menus, et les raccourcis clavier. Pour ouvrir un fichier, utilisez le menu Edit/Find, et pour le sauver, utilisez le menu Edit/Save. Les menus indiquent aussi les raccourcis clavier. Pour Edit/Find, le raccourci clavier est C-x C-f. Ca se lit Controle-X, Controle-F, et pour faire cela, il faut enfoncer la touche Controle, la maintenir enfoncée, presser la touche x ; enfoncer la touche Controle, la maintenir enfoncée et presser la touche f.

Vous avez écrit votre premier programme Pascal, dans ce que l'on appelle un fichier *source*. Votre ordinateur ne comprend pas directement le Pascal. Vous devez transformer le texte en langage Pascal par un programme en *langage machine*, appelé *exécutable*. Le compilateur est une application qui se charge de transformer automatiquement un programme source en un programme exécutable. Pour lancer le compilateur, cliquez sur le bouton 'Compile' (3ème à partir de la droite, Figure 19).

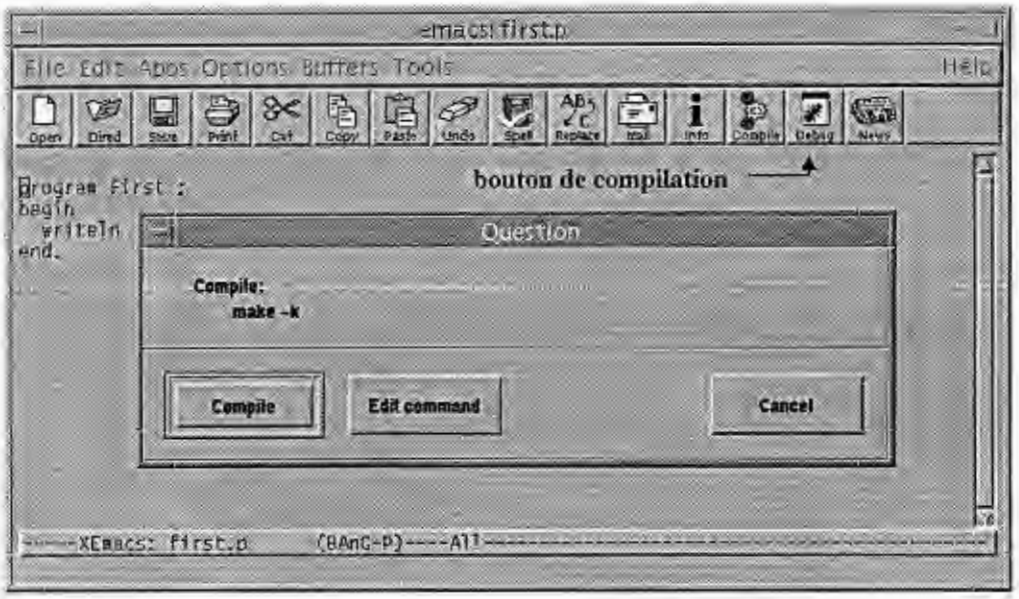


FIGURE 19. fenêtre de commande de compilation

xemacs ouvre une fenêtre temporaire, demandant quelle commande de compilation il faut utiliser. Cliquez sur le bouton 'Edit command', et entrez la commande 'pc -g -o first first.p'. Dans cette ligne de commande, pc est le nom du compilateur (pc veut dire Pascal Compiler). '-g' est une option de compilation qui permet de corriger les programmes qui ne se comportent pas comme on s'y attend (expliquée plus tard, en cours de semestre. '-o first' est une deuxième option de compilation spécifiant le nom de l'exécutable produite par le compilateur (dans ce cas-ci, le nom est 'first'), et 'first.p' est le nom du fichier source. La commande dit donc : compiler le fichier source first.p et produire l'exécutable 'first'. Le nom du fichier source est bien entendu le nom d'un

fichier que vous avez écrit. Le nom de l'exécutable est totalement arbitraire. Vous auriez pu l'appeler xxx ou arthur. Après la compilation, la fenêtre xemacs aura l'allure de la Figure 20.

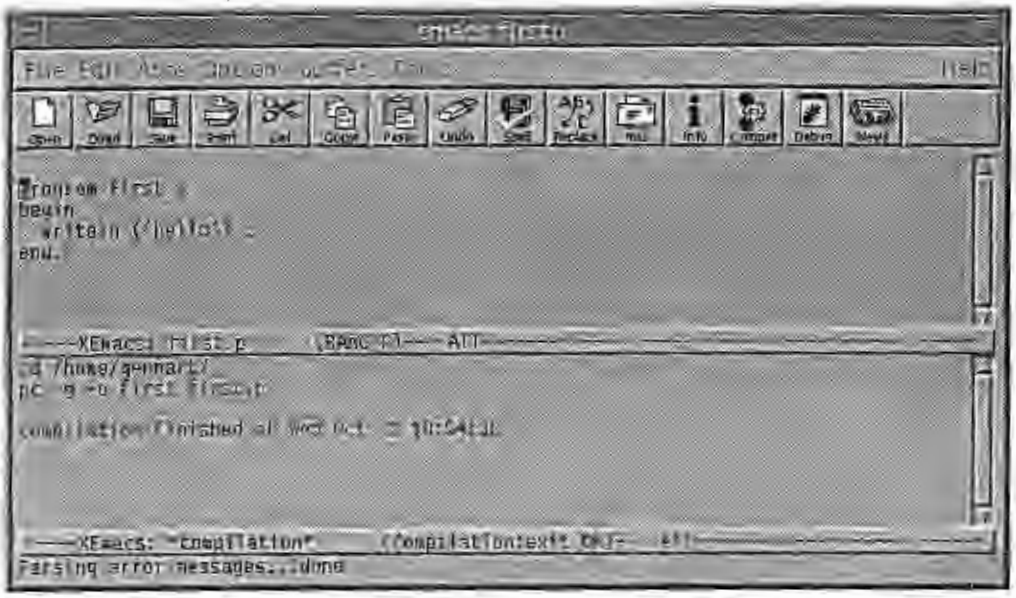


FIGURE 20. La fenêtre Xemacs après la compilation

Pour lancer l'exécutable deux solutions : double-cliquer sur l'icône qui sera apparue dans le gestionnaire de fichier ; ou taper le nom de l'exécutable (first) suivi de RETURN dans une fenêtre terminal. Si vous employez la première solution, le système d'exploitation ouvrira une femêtre d'exécution dans laquelle apparaîtra le texte 'hello, world'. Si vous utilisez la deuxième solution, le texte 'hello, world' apparaîtra en dessous de la commande que vous avez tapé.

Maintenant que vous avez plusieurs fichiers d'ouvert, vous pouvez les retrouver plus rapidement à l'aide du menu Buffers (Figure 21).

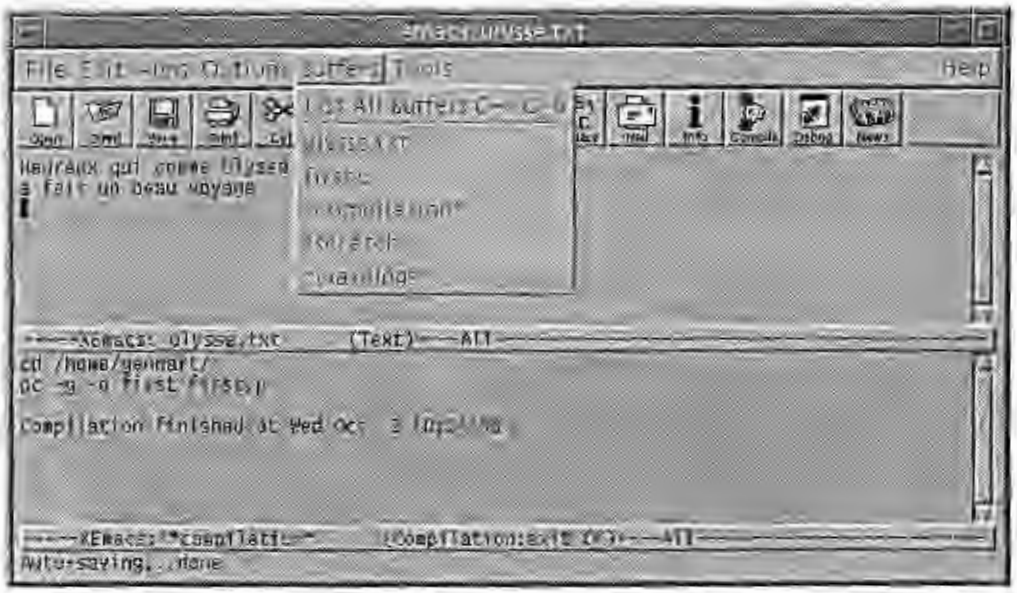


FIGURE 21. Le menu Buffers

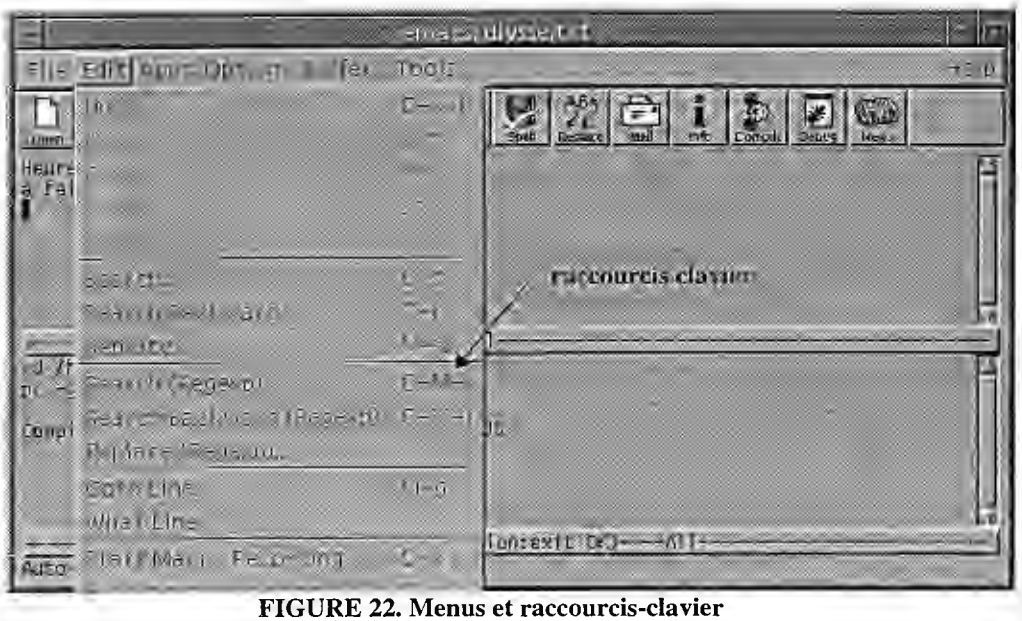


FIGURE 22. Menus et raccourcis-clavier

Une organisation efficace de votre écran pour éditer compiler et exécuter un programme est celle de la Figure 23.

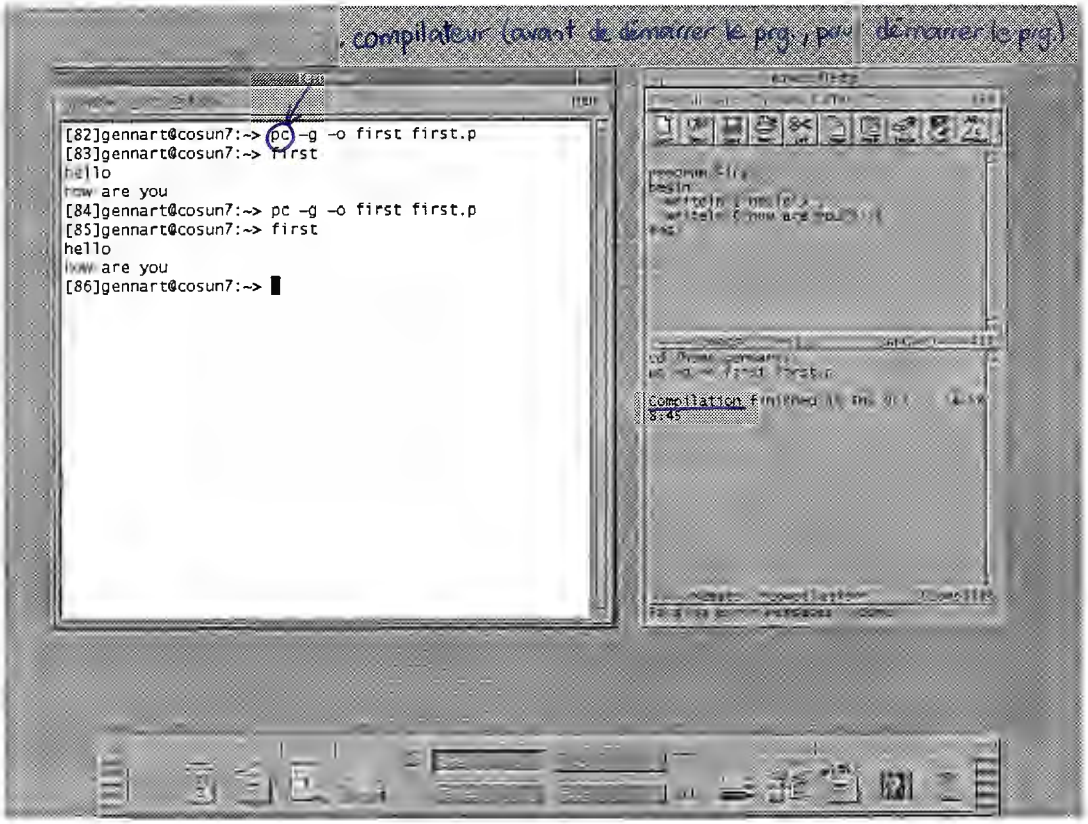


FIGURE 23. L'environnement complet d'édition, de compilation et d'exécution d'un programme pascal

Fichier et Buffer

Les fichiers sont quelques choses de permanent qui survivent lorsque vous quittez EMACS, et même lorsque vous quittez votre environnement (log out). Les buffers sont la version interne à EMACS de vos fichiers. Un

fichier posséde un chemin complet (/home/gennart/pascal/ulysse.txt). Un buffer possède le même nom que le fichier, sans l'information de répertoire (ulysse.txt).

Raccourci clavier

A côté de certaines commandes sur le menu, il y a entre parenthèses le raccourci clavier. Autant les menus sont utiles pour apprendre les commandes d'un logiciels, autant le processus de sélection de commande par menu est lent. Le raccourci clavier permet d'accélérer le travail une fois que l'on connaît mieux les commandes. Par exemple, le raccourci clavier `(C-x C-s)` se lit Control-x, Control-s. Cela veut dire : enfoncer la touche CONTROL, et en la laissant enfoncée, enfoncer la touche x ; relâcher les deux touches ; enfoncer la touche CONTROL, et en la laissant enfoncée, enfoncer la touche s ; relâcher les deux touches. Cela aura le même effet que de sélectionner par menu la commande File/Save , ou de presser le bouton Save.

Une autre raccourci clavier important est (M-f). Cela se lit Meta-f. La touche META a est la touche juste à coté de la barre d'espace, marquée d'un losange noir. M-f veut dire : enfoncer la toucher META, et en la laissant enfoncée, enfoncer la touche f. Cela fait avancer le curseur d'un mot complet (commande forward-word).

Un bon exemple d'utilisation de la touche meta est la fonction query-replace (M-%), menu Edit/Replace, qui permet de retrouver une chaîne de caractères et la remplacer par une autre chaîne de caractères. M-% veut dire : enfoncer la touche META, en la laissant enfoncée, enfoncer la touche SHIFT, et en laissant les deux touches enfoncées, appuyez sur la touche %.

Par exemple, dans le texte Ulysse.txt, tapez M-% eu RETURN ue RETURN. Le curseur va se placer au début de la première instance de la chaîne de caractères "eu". Si vous tapez la touche ESPACE, eu sera remplacé par ue. Si vous tapez la touche Suppr., EMACS déplacera simplement le curseur jusqu'à la prochaine instance de la chaîne de caractères "eu". Et ce jusqu'à ce que toutes les instances de la chaîne de caractères "eu" ait été traversées.

Undo

Somme toute, la modification effectuée à la section précédente n'était pas fameuse, et vous souhaitez "revenir en arrière". Rien de plus facile. Sélectionnez la commande Edit/Undo (raccourci clavier C-x u, ou bouton Undo). Xemacs Undo est très puissant, vous pouvez défaire un grand nombre de commandes précédentes.

Vous ne savez plus ou vous en êtes avec Xemacs

Essayez d'abord C-g, qui interrompt la commande en cours. Cela devrait vous rendre la main. Si rien ne va plus, vous pouvez toujours sélectionner la commande exit dans la barre de contrôle de la fenêtre.

Couper Coller

Pour sélectionner une partie du texte à l'aide de la souris, placez le curseur écran sur le premier caractère du texte que vous souhaitez sélectionner, enfoncez le bouton gauche de la souris, et traînez le curseur (en gardant le bouton de la souris enfoncé) jusqu'à la fin du texte à sélectionner. Relâchez le bouton de la souris. Vous avez sélectionné une partie de texte. Vous pouvez maintenant la mémoriser (bouton Cut ou menu Edit/Cut), bouger le curseur, et coller le texte à un autre endroit (bouton Paste, ou menu Edit/Paste). Si vous souhaitez simplement copier le texte sélectionner à un autre endroit (sans effacer le texte original), utilisez le menu Edit/Copy au lieu de Edit/Cut.

Recherche et recherche incrémentale

Pour rechercher un mot dans un texte, placez le curseur Xemacs au début du texte, et utilisez le menu Edit/Search. Indiquez le mot que vous voulez trouver dans la fenêtre d'interaction, suivi de la touche RETURN. Pour la recherche incrémentale, utilisez le raccourci clavier (C-s). Au fur et à mesure que vous indiquerez les lettres du mot que vous cherchez, le curseur se déplacera à la première instance du mot indiqué.

Quitter EMACS

Pour quitter Xemacs, utilisez la commande File/Exit Xemacs. Xemacs vérifiera que vous avez bien sauvé tous vos fichiers.

La barre de boutons

FIGURE 24. La barre de boutons Xemacs

Nous avons déjà passé en revue les boutons Open, Save, essentiels pour le traitement des fichiers ; les boutons Cut, Copy and Paste pour l'édition de texte, et le bouton Compile pour la compilation. Le bouton Print permet d'imprimer un fichier. Le bouton Undo permet d'annuler une ou plusieurs commandes précédentes. Le bouton Spell permet de vérifier l'orthographe des textes écrits en Anglais. Le bouton Replace a le même effet que la commande menu Edit/Replace ou le raccourci clavier M-%. Le bouton Mail vous permet de lire et d'éditer votre courrier à l'intérieur de xemacs. C'est une alternative à l'outil de lecture de courrier fourni par le panneau de commande (voir section 1.8). Le bouton Info vous donne accès aux informations sur toutes les possibilités e Xemacs et des logiciels fourni par la 'Free Software Foundation', l'organisme qui distribue gratuitement Emacs, divers compilateurs et une grande série de logiciels d'excellente qualités.

Fonctions avancées

- Appel explicite de fonction. Toutes les fonctions de Xemacs sont invocables explicitement. Par exemple, pour avancer d'un caractère dans le texte, vous pouvez utiliser la touche flèche-à-droite, ou bien taper `M-x forward-char`, suivi de la touche RETURN. Pour une commande comme query-replace, il y a trois façons de l'appeler : menu Edit/Replace, raccourci clavier (M-%), et appel explicite (M-x query-replace).
- La liste des fonctions disponibles et leurs raccourcis clavier. Tous les raccourcis clavier disponibles peuvent etre obtenus en utilisant la commande M-x describe-bindings.
- Complétion automatique des noms de fichiers. Lorsque vous utilisez la commande File>Open, vous ne devez pas taper toutes les lettres du fichier que vous souhaitez trouver. En tapant simplement Ul TAB, EMACS complétera automatiquement les lettres manquantes (Ulysse.txt)
- Lorsqu'on a une commande assez longue à taper (comme M-x describe-bindings), on peut le faire de la façon suivante : M-x des TAB bin TAB. Xemacs complétera automatiquement les caractères manquant, à condition bien entendu que les caractères que vous ayez donné ne soit pas ambigus. Si les caractères que vous avez donnés forment le début de plusieurs fonctions de Xemacs, Xemacs vous en fera la liste.
- Introduction à EMACS. EMACS possède une introduction disponible "on-line", que l'on peut obtenir en utilisant la commande `M-x help-with-tutorial`.
- Fenêtres multiples : Pour diviser votre fenêtre Xemacs en deux sous fenêtres horizontales, utilisez (C-x 2). En deux fenêtres verticales, utilisez (C-x 3). Pour passer d'une fenêtre à l'autre, utilisez la souris, et cliquez à l'endroit ou vous voulez placerle curseur. Pour éliminer une fenêtre, placez votre curseur dans cette fenêtre, et utilisez (C-x 0).

1.5 Le compilateur

Plutot que de lancer le compilateur depuis Xemacs, on peut le lancer depuis la fenêtre terminal. Dans une fenêtre terminal, tapez maintenant la commande :

```
cosun12% pc -g -o first first.p
```

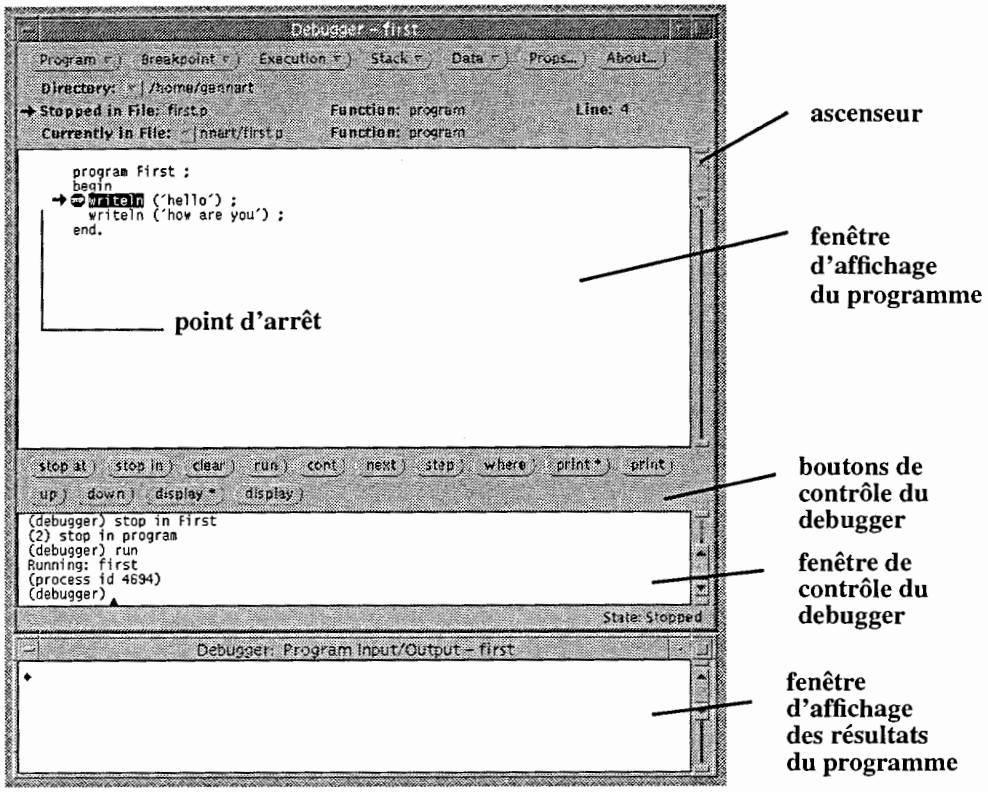
Pour lancer l'application first, tapez dans la fenêtre xterm la commande :

```
cosun12% first
Hello, world
cosun12%
```

Et vous obtiendrez le résultat indiqué. Si par hasard cela ne fonctionnait pas, essayez la commande `./first`.

1.6 Le debugger

Lorsque les programmes deviennent un peu plus compliqué, ils contiennent souvent des fautes, qui sont difficiles à déterminer. Pour vous aider à les découvrir, il existe un outil très pratique, le debugger. Pour le lancer, dans une fenêtre xterm, lancez la commande debugger first &. Cela veut dire, lancer le debugger sur l'application `first`. Le résultat est la fenêtre suivante :



Pour exécuter un programme en utilisant le debugger, les étapes à suivre sont :

- double-cliquez sur le symbole First dans la fenêtre d'affichage du programme. Le symbole est maintenant affiché en blanc sur fond noir. Cliquez sur le bouton `stop in`. Un panneau stop, appelé un point d'arrêt, apparaîtra à la première ligne exécutable du programme. Vous pouvez obtenir exactement le même effet en tapant dans la fenêtre de contrôle du debugger la commande `stop in First`. Pour la commande stop in, vous devez spécifier le nom d'une routine. Pour la commande stop at, vous devez spécifier un numéro de ligne de votre programme.
- cliquez en suite sur le bouton run, ce qui aura pour effet de commencer l'exécution du programme. L'indicateur de ligne doit maintenant être actif (l'indicateur de ligne est maintenant plein, alors qu'auparavant, il ne s'agissait que d'un contour.
- cliquez ensuite sur step, et observez que l'indicateur de ligne se déplace. Vous pouvez ainsi suivre pas à pas l'exécution de votre programme.

1.7 Netscape

Netscape est une application qui permet de "surfer" l'Internet, ou le WWW (World Wide Web). Netscape implémente le concept d'hypertexte au niveau mondial (Figure 20). Pour lancer netscape, taper netscape & dans une fenêtre Terminal.

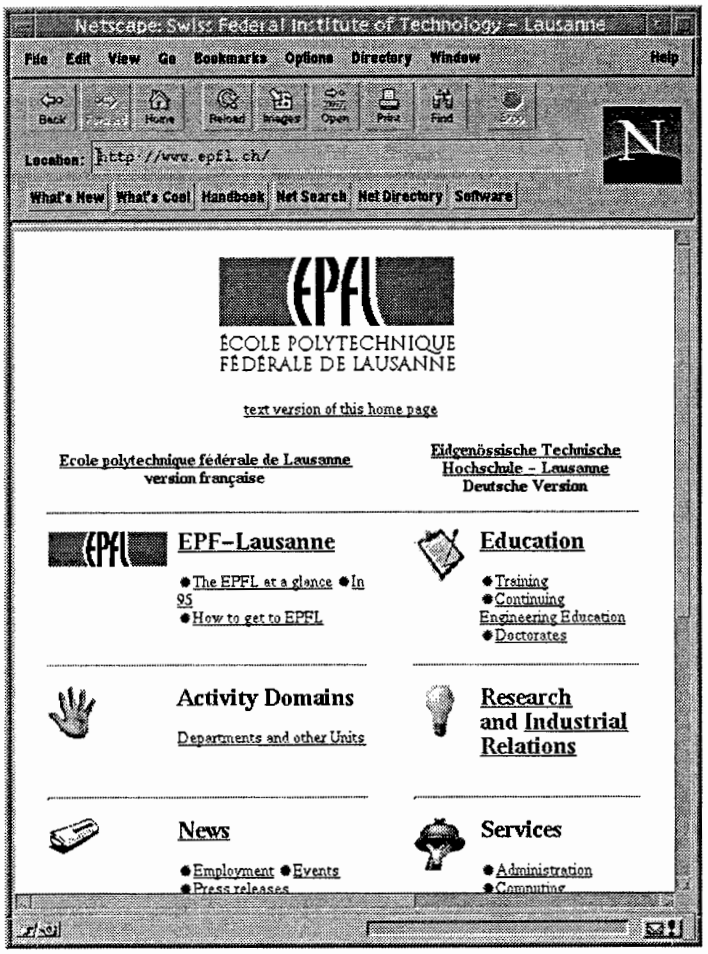


FIGURE 25. Fenêtre NetScape

L'organisation habituelle d'un texte est linéaire. Un hypertexte est un texte actif. Les parties soulignées (dites actives) dans le texte permettent d'accéder à d'autres pages WWW. Ce mécanisme simple est très puissant. Pourvu qu'on ait les bons liens, on peut trouver des informations aussi bien en Californie qu'au Japon. On peut accéder à des informations structurées en arbre, et faire référence à d'autres pages.

Pour référence, je donne l'adresse de quelques sites : l'EPFL , **http://www.epfl.ch/** ; Edicom, ou vous trouverez entre autres choses les programmes de cinéma, **http://www.edicom.ch/** ; Lycos, où vous pouvez faire des recherches Internet par mot-clé, **http://www.lycos.com/** (Figure 26) ; et Altavista, où vous pouvez aussi faire des recherches par mot-clé, **http://altavista.digital.com/**.

Pour visiter un site, taper dans la barre 'Location :' le nom du site que vous voulez visiter, suivi de Return. Pour interrompre un recherche, cliquez sur l'icone NetScape.

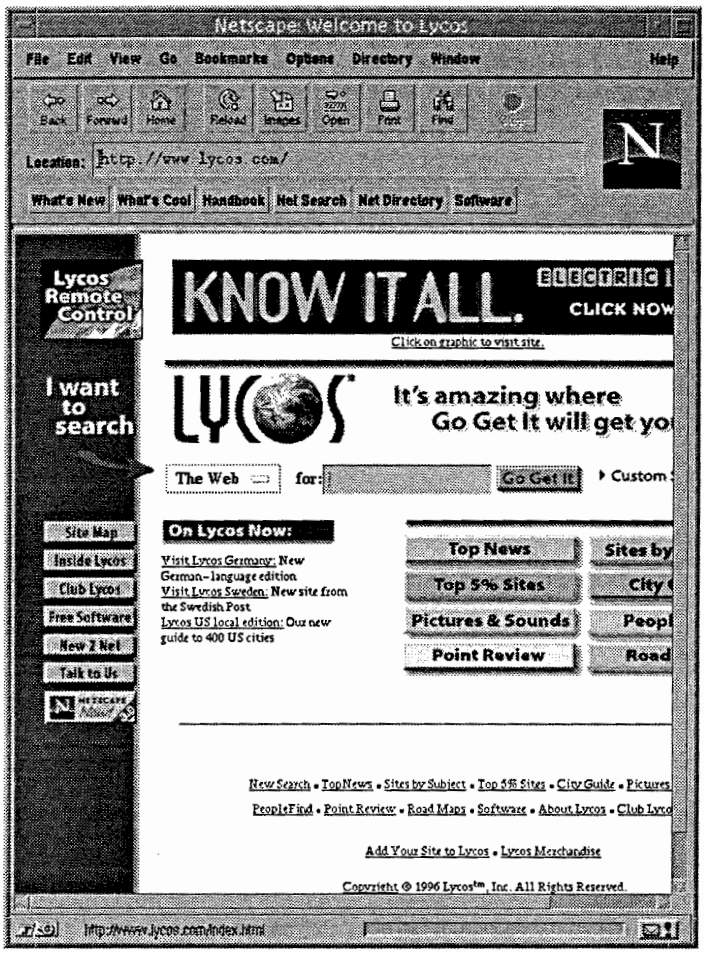


FIGURE 26. Le site Lycos de recherche par mot-clé.

1.8 Mailtool

En plus de netscape, un autre outil de communication très pratique est le courrier électronique. Vous avez tous une adresse électronique. Mon adresse est par exemple gennart@dico.epfl.ch. Avec cette adresse, je suis accessible depuis partout dans le monde. Votre adresse est votre nom de compte, suivi de @ (a commercial, at), suivi de dico.epfl.ch.

Pour utiliser le logiciel de traitement du courrier électronique, double-cliquez sur l'icône courrier (la 5ème à partir de la gauche sur le panneau de commande. Vous verrez apparaitre la fenêtre suivante (Figure 27). La partie du dessus contient la liste des messages, celle du dessous le contenu du message sélectionné dans la liste.

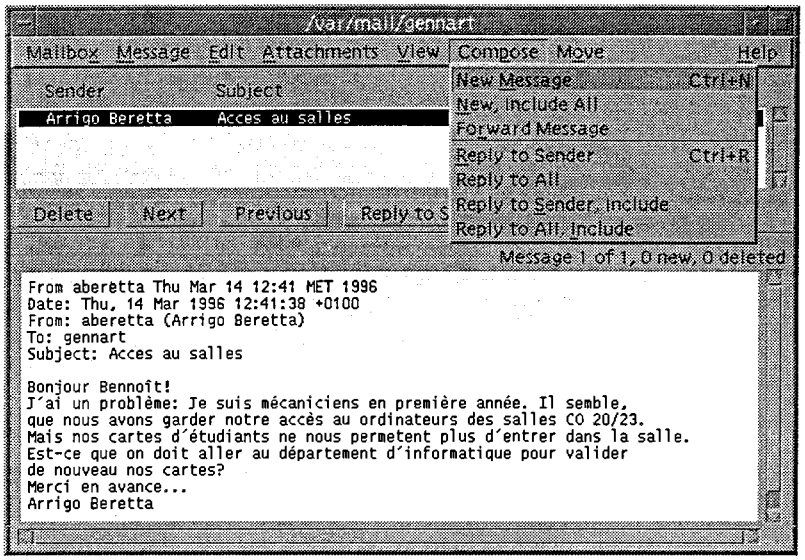
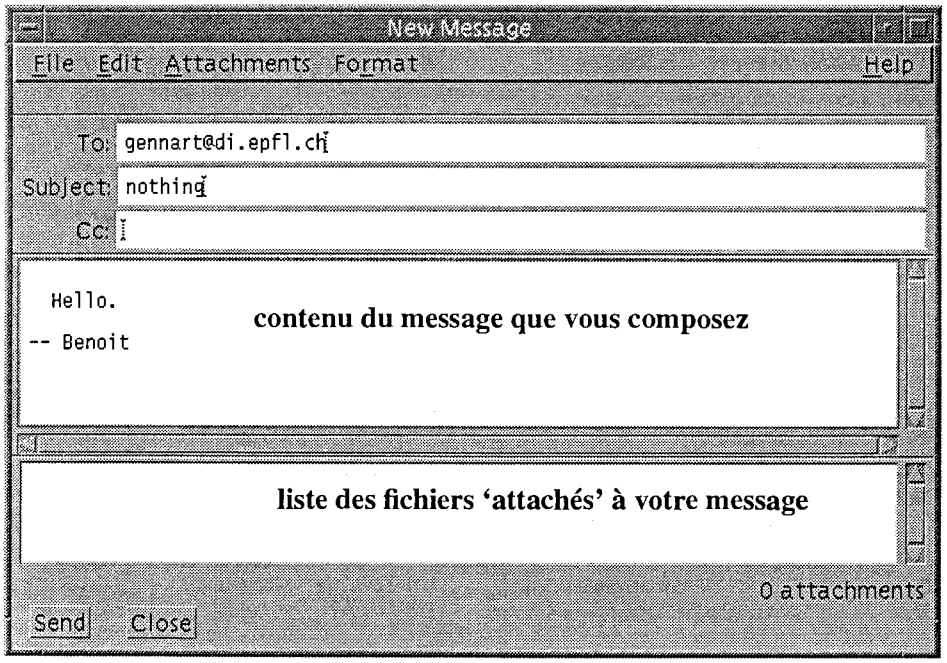


FIGURE 27. Outil de traitement du courrier électronique

Pour envoyer un message, sélectionnez la commande NewMessage dans le menu compose (Figure 27). Une nouvelle fenêtre d'édition de messages apparaîtra. Indiquez l'adresse du destinataire dans la barre To:, le sujet dans la boite Subject:, et le contenu du message dans la fenêtre texte (celle ou j'ai marqué 'hello'). Cliquez sur le bouton send pour envoyer. Vous pouvez aussi *attacher* des fichiers à votre courrier en utilisant le menu Attachements.



2 Une première session à votre station de travail : resumé

Pour vérifier que vous avez acquis la section 1, vérifiez que vous passez ces 10 étapes sans difficultés.

1. **login**. Lorsque vous arrivez à votre station de travail, la seule chose que vous pouvez faire, c'est vous loguer, c'est-à-dire, entrer votre nom, suivi de la touche RETURN et votre mot de passe, suivi de la touche RETURN. Vous aurez reçu une feuille avec le nom de votre compte, et votre mot de passe.
2. **Lancez une application 'file manager'** en double-cliquant sur le 3ème bouton à partir de la gauche dams le panneau de commande. Déplacez-vous dans la hiérarchie de fichiers.
3. **Lancez une fenêtre Terminal** à l'aide du lanceur d'application (4ème bouton à partir de la gauche sur le panneau de commande). Elle contient un prompt, c'est-à-dire un le nom de votre machine suivi de % (par exemple cosun23%).
4. Essayez au moyen de la souris de changer la taille de la fenêtre et de la déplacer. Vous pouvez aussi l'iconifier en cliquant le bouton avec un petit point, à droite dans la barre supérieure de la fenêtre. Pour réouvrir la fenêtre, double-cliquez sur l'icône. En cliquant le bouton de menu (bouton à gauche de la barre de titre), un menu de fenêtre apparaît. Il permet de changer la taille de la fenêtre, de l'iconifier, etc. Si vous perdez le contrôle, essayez de taper C-q (touche CONTROL enfoncée, touche q), ou bien C-c (touche CONTROL enfoncée, touche c). Cela devrait vous remettre dans une situation ou vous pouvez travailler.
5. **lancer une autre application**. Dans la fenêtre Terminal, essayez par exemple de lancer la commande `/usr/openwin/demo/xeyes &`. Cela ouvrira une paire d'yeux, qui suivent le curseur.
6. **lancer l'éditeur**. Tapez `xemacs &` dans la fenêtre Terminal. Une fenêtre d'édition s'ouvre. Ouvrez un fichier bonjour.p en utilisant le bouton Open. Vous pouvez maintenant y écrire du texte. Pour déplacer votre curseur, vous pouvez utiliser la souris et cliquer, ou utiliser les touches flèches.
7. **Ecrire un programme simple** : tapez le programme suivant, et sauvez le en utilisant le bouton Save.

```
program bonjour;
begin
   writeln('Bonjour Hal');
   writeln('Belle journée');
end.
```

Programme 1

8. **compiler le programme**. Dans la fenêtre Terminal, tapez la commande `pc -g -o hello bonjour.p`. Ceci produira à partir de votre programme une nouvelle application exécutable appelée hello.
9. **exécuter le programme**.Vous pouvez lancer votre programme en tapant la commande `hello` dans la fenêtre Terminal. Si ca ne marche pas, essayez `./hello`. Le résultat à l'écran devrait être :

```
cosun12% ./hello
Bonjour Hal
Belle journée
cosun12%
```

10. **terminer votre session de travail**. Pour quitter Xemacs, sélectionner le menu File/Exit Xemacs. Sélectionner ensuite dans le panneau de commande la commande Exit. N'éteignez ni l'écran ni la station.

graphics.h : contient une liste de procédures
graphics.pas : contient la programmation des procédures mentionnées dans graphics.h

1 Module graphique

Le but de cette section est de vous apprendre deux choses : (1) l'utilisation d'une librairie de routines ; (2) le contenu d'une librairie d'affichage à l'écran. Considérons le programme 1. Après l'instruction **program**, il contient une instruction #include, suivie d'un nom de fichier entre guillemets, "Graphics.h". Attention, il est essentiel que l'instruction #include soit placée au tout début d'une ligne.

```
(* file : graphics-first.p *)
program GraphicsFirst ;
#include "Graphics.h"
  begin
    FillRectangle (100, 50, 200, 250) ;
    writeln('Hit return to terminate the program');
    flush ;
    readln ;
  end.
```

Programme 1

Le fichier Graphics.h est ce qu'on appelle un fichier d'interface. Il contient une liste d'interface de procédures (programme 2). Un interface de procédure commence par le mot-clé **procedure**. Suivent le nom de la procédure, qui suggère ce que la procédure fait, ses paramètres *formels*, et enfin le mot clé **external**.

```
procedure FillRectangle (top, left, bottom, right : integer) ; external ;
procedure DrawRectangle (top, left, bottom, right : integer) ; external ;
procedure FillOval (top, left, bottom, right : integer) ; external ;
procedure DrawOval (top, left, bottom, right : integer) ; external ;
procedure DrawLine (fromX, fromY, toX, toY : integer) ; external ;
procedure PenSize (pixels : integer) ; external ;
procedure SetColor (color : integer) ; external ;
procedure SuspendRefresh ; external ;
procedure ResumeRefresh ; external ;
procedure SetWindowSize (sizeX, sizeY : integer) ; external ;

procedure Delay (millisec : integer ) ; external ;
procedure GetTime (var millisec : integer) ; external ;
```

Programme 2

Par exemple, la procédure FillRectangle a quatre paramètres de type entier, appelés top, left, bottom, et right. Elle dessine dans une fenêtre graphique un rectangle plein dont le coin supérieur gauche a pour coordonnée (left, top), et le coin inférieur droit (right, bottom). Le programme 1 contient un appel à la procédure FillRectangle. Un appel de procédure est un nom de procédure suivi de paramètres effectifs. Il faut que le nombre et le type de paramètres effectifs correspondent au nombre et au type de chacun des paramètres formels. En l'occurrence, il faut spécifier 4 valeurs entières entre parenthèses dans l'appel de la procédure FillRectangle.

Une procédure est en fait un emballage pour une série d'instruction (parfois très longue) qui effectue la commande suggérée par le nom de la procédure.

Les autres procédures contenue dans le fichier Graphics.h sont DrawRectangle (contour d'un rectangle) ; FillOval (ovale plein) ; DrawOval (contour d'un ovale) ; DrawLine (trait) ; PenSize (épaisseur du trait) ; SetColor (niveau de gris, de 0 (noir) à 8 (blanc)). La routine SetWindowSize donne la taille initiale de la fenêtre et doit être appelée avant toutes les autres.

Les routines SuspendRefresh et ResumeRefresh sont des routines d'optimisation de la performance. Lorsqu'on a beaucoup de primitives graphiques à dessiner, il vaut mieux d'abord interrompre le rafraîchissement d'écran (SuspendRefresh), dessiner toutes les primitives (cercles, droites, rectangle), et relancer le rafraîchissement d'écran qui affichera alors toutes les primitives d'un coup.

Le fichier Graphics.h contient deux autres procédures, Delay et GetTime. La routine Delay permet de suspendre l'exécution d'un programme pendant un certain nombre de millisecondes. Cela est utile pour simuler le mouvement (afficher un point, attendre 50ms, l'effacer, et afficher le point à un autre endroit). La routine Get-

Time donne un temps en milliseconde, calculé à partir d'une certaine date dans le passé. En appelant deux fois la routine GetTime, on peut déterminer la durée d'une portion de code.

Pour compiler le programme 1, on utilise les commandes suivantes.

une fois par session
compilation
recompilation
info sur les variables d'environnement

```
cosun12% source ~gennart/.pascal
cosun12% pc -I$GINC -g -o graphics-first.p -l$GLIB
cosun12%
cosun12% pc -I$GINC -g -o graphics-first.p -l$GLIB
cosun12% pc -I$GINC -g -o graphics-first.p -l$GLIB
cosun12% echo $GINC
/home/gennart/pascal/benoit/window
cosun12% echo $GLIB
Graphics -lXm -lXt -lX11 -lgen -lposix4
```

La première commande doit être exécutée une seule fois, tout au début d'une session de travail. Elle initialise un certain nombre de variables d'environnement, dont on peut examiner le contenu ($GINC (Graphics INClude) et $GLIB (Graphics LIBraries)). Les programmes qui utilisent les routines graphiques ont une commande de compilation un peu plus compliquée. On doit ajouter l'option -I$GINC au début de la commande et -l$GLIB à la fin de la commande.

Le point (0,0) d'une fenêtre graphique se trouve par convention dans le coin supérieur gauche. La coordonnée x croît vers la droite. La coordonnée y croît vers le bas (figure 1). La figure 1 montre aussi la fenêtre graphique résultant de l'exécution du programme 1.

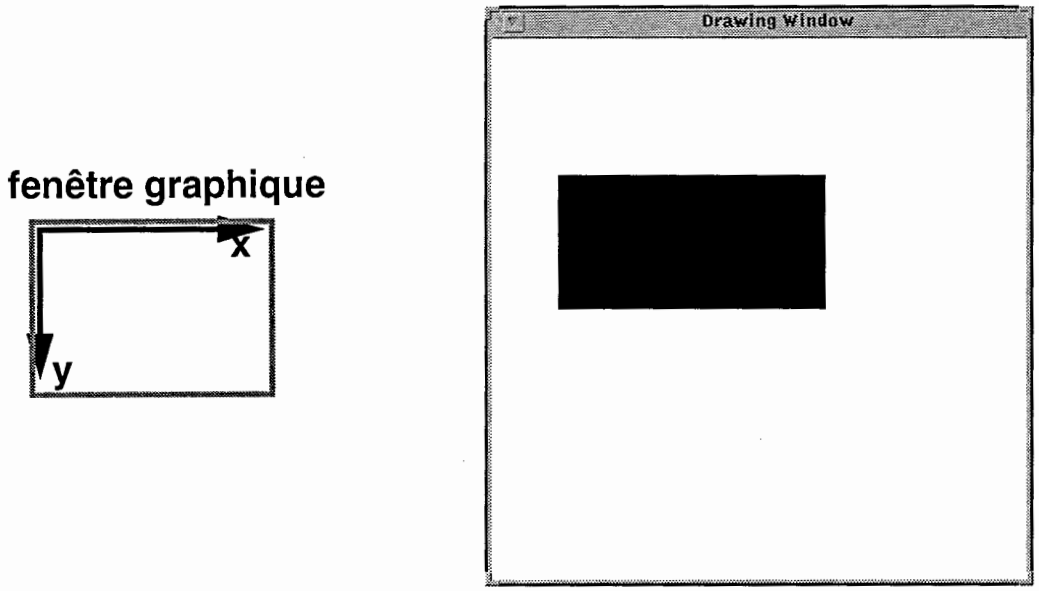


FIGURE 1. Fenêtre graphique

complément : c.f. annexe cours SL4-1 à SL4-4

2 Module de calcul matriciel

Cette section montre, sur base d'un exemple de calcul matriciel, comment structurer un programme, c'est-à-dire, comment le diviser en procédures.

Considérons le programme 3 qui initialise une matrice et l'affiche.

```
(* file : matrice1.p *)
program MatrixOne ;
  const
    n = 5;
  var
    i, j : integer ;
    m1 : array[1..n,1..n] of real ;
    x : real ;
begin
  (* initialiser la matrice *)
  for i := 1 to n do
    begin
      for j := 1 to n do
        begin
          m1[i,j] := random (x) ;
        end ;
    end ;
  (* afficher la matrice *)
  writeln ('matrix = ') ;
  for i := 1 to n do
    begin
      for j := 1 to n do
        begin
          write (' ', m1[i,j]:5:3) ;
        end ;
      writeln ;
    end ;
end.
```

Programme 3

```
(* file : matrice2.p *)
program MatrixTwo ;
  const
    n1 = 5; n2 = 3 ;
  var
    i, j : integer ;
    m1 : array[1..n1,1..n1] of real ;
    m2 : array[1..n2,1..n2] of real ;
    x : real ;
begin
  (* initialiser la matrice m1 *)
  for i := 1 to n1 do
    begin
      for j := 1 to n1 do
        begin
          m1[i,j] := random (x) ;
        end ;
    end ;
  (* initialiser la matrice m2 *)
  for i := 1 to n2 do
    begin
      for j := 1 to n2 do
        begin
          m2[i,j] := random (x) ;
        end ;
    end ;

  (* afficher la matrice m1 *)
  writeln ('m1 = ') ;
  for i := 1 to n1 do
    begin
      for j := 1 to n1 do
        begin
          write (' ', m1[i,j]:5:3) ;
        end ;
      writeln ;
    end ;
  (* afficher la matrice m2 *)
  writeln ('m2 = ') ;
  for i := 1 to n2 do
    begin
      for j := 1 to n2 do
        begin
          write (' ', m2[i,j]:5:3) ;
        end ;
      writeln ;
    end ;
end.
```

Programme 4

Remarquez les commentaires qui indiquent ce que chacune des parties de code fait.

On souhaite maintenant initialiser et afficher deux matrices. Un couper-coller judicieux permet de faire cela sans effort (programme 4). On remarque cependant une chose : deux parties de codes se ressemblent très fort. Le bout de code pour afficher la matrice m1 est virtuellement identique au bout de code pour afficher la matrice m2. Les seules différences sont dans le message ('m1 = ' ou 'm2 = ') et le nom de la matrice ('m1[i,j]' ou 'm2[i,j]'). On peut prendre avantage de cela pour donner un nom et des paramètres au bout de code incriminée. Le nom de la routine indiquera ce que la partie de code fait, et les paramètres donnent un nom aux parties changeantes du bout de code. C'est ce qui apparaît dans le programme 5.

Interface et corps de procédure. L'interface d'une procédure comporte son nom et ses paramètres. Le nom de la procédure doit refléter ce qu'elle fait, par exemple, dans notre cas, `WriteMatrix`. Les paramètres possèdent un nom et un type. On les appellent les paramètres *formels*, car ils décrivent la forme de l'appel de fonction.

```
const
  n = 5 ;
type
  MatrixT = array[1..n,1..n] of real ;

procedure WriteMatrix ( message : string ; matrix : MatrixT ) interface
          nom           paramètres formels

  var
     i, j : integer ;
   begin
     writeln (message) ;
     for i := 1 to n do
       begin
         for j := 1 to n do
           begin
             write (' ', matrix[i,j]:5:3) ;
           end ;
        writeln ;
       end ;
   end ;
```

corps

Programme 5

Le corps de la routine décrit ce que fait la fonction. Il peut y avoir des déclarations de type, des déclarations de variable, des instructions (entre **begin** ... **end**;) et même des déclarations de procédure.

Association des paramètres par position. Tout ce qu'on peut faire avec une routine, c'est l'appeler. Appeler une routine, c'est écrire son nom, et indiquer un nom de variable pour chacun des paramètres formels. Les noms de variables que l'on donne lors d'un appel de fonction sont appelés les paramètres ***effectifs***. Le programme 6 montre une exemple complet de programme faisant recours aux procédures. Il y a deux procédures : InitializeMatrix and WriteMatrix, et 4 appels de procédure (les 4 dernières lignes du programme). Par exemple, l'instruction `WriteMatrix ('m1 =', m1) ;` veut dire : appeler la procédure WriteMatrix, en utilisant comme message la chaîne de caractère ('m1 = '), et comme matrice la variable m1.

```
(* file : matrice3.p *)
program MatrixThree ;
  const
    n = 5 ;
  type
    MatrixT = array[1..n,1..n] of real ;
  var
    m1 : MatrixT ;
    m2 : MatrixT ;
    x : real ;

  procedure InitializeMatrix (var m : MatrixT);
    var
      i, j : integer ;
    begin
      for i := 1 to n do
        begin
          for j := 1 to n do
            begin
              m[i,j] := random (x) ;
            end ;
        end ;
    end ;
```

```
  procedure WriteMatrix
      ( s : string ;m : MatrixT);
    var
      i, j : integer ;
    begin
      writeln (s) ;
      for i := 1 to n do
        begin
          for j := 1 to n do
            begin
              write (' ', m[i,j]:5:3) ;
            end ;
         writeln ;
        end ;
    end ;

begin
  InitializeMatrix (m1) ;
  InitializeMatrix (m2) ;
  WriteMatrix ('m1 =', m1) ;
  WriteMatrix ('m2 = ', m2) ;
end.
```

Programme 6

Passage des paramètres par valeur ou par référence. De façon générale, les paramètres sont passés par *valeur* en PASCAL. Cela veut dire qu'avant de rentrer dans une procédure, une nouvelle copie de la variable

qui sert de paramètre effectif est créée. La copie contient exactement la même valeur que l'original. L'avantage est que si vous changez à l'intérieur de la procédure la valeur d'un paramètre, vous changez seulement la valeur de la copie, et pas la valeur de la variable originale. Cela évite beaucoup d'erreur. L'inconvénient du passage de paramètres par valeur est le manque de performance. Créer des copies de variables, et tout particulièrement des copies de tableau prend du temps.

Que fait-on lorsque précisément on souhaite changer la variable originale ? On ajoute devant la déclaration de paramètre formel le mot-clé var. L'effet est de passer la variable par référence, c'est-à-dire, passer l'adresse de la variable, plutôt que de créer une copie complète. Considérons par exemple la procédure InitializeMatrix. Ce qui est souhaité dans le programme MatrixThree est d'affecter des valeurs aléatoires à chacun des éléments des tableaux m1 et m2. Si l'on a comme interface de procédure `InitializeMatrix ( m : MatrixT)`, le corps de la procédure initialisera la copie interne de la matrice, et cela n'aura aucun effet sur les variables m1 et m2. En ajoutant le mot-clé var dans l'interface de la procédure (`InitializeMatrix (var m : MatrixT)`), cela permet de modifier le contenu de m1 et m2.

En résumé, si vous souhaitez que le contenu d'un paramètre reste modifié après la fin d'un appel de procédure, utilisez le mot clé var. Pour tous les autres paramètres, utilisez la déclaration habituelle des paramètres, c'est plus sur.

Un dernier commentaire. Le passage de paramètres par référence est nettement plus efficace (on ne passe qu'une référence, plutôt qu'une copie de la variable). C'est pourquoi on a tendance à passer les variables de grande taille (tableaux) par référence, même si leur contenu n'est pas modifié.

Passage en paramètres de tableau de taille quelconque. Le programme 7 généralise encore le programme 6, en permettant le passage de tableau de taille quelconque. Pour obtenir ce résultat, on utilise les tableaux homologues. La déclaration de tableaux homologues n'est admise que dans les interfaces de procédure. Par exemple, la déclaration de paramètre `v : array[lb..ub:integer] of real ;` déclare comme paramètre un vecteur dont la taille ne sera connue qu'à l'appel de la fonction. Le programme 7 montre comment déclarer un tableau homologue à deux dimensions. Au moment de l'appel, le tableau homologue et ses bornes prennent les valeurs correspondantes du paramètre actuel. Le programme 7 vous permet donc d'initialiser et d'afficher avec la même routine des tableaux de taille différentes.

```
(* file : matrice4.p *)
program MatrixFour ;
  const
    n1 = 5; n2 = 3 ;
  var
    m1 : array[1..n1,1..n1] of real ;
    m2 : array[1..n2,1..n2] of real ;
    x : real ;

  procedure InitializeMatrix
    ( var m : array[ lbm1..ubm1:integer
                   ; lbm2..ubm2:integer
                   ] of real
    ) ;
    var
      i1, i2: integer ;
    begin
      for i1 := lbm1 to ubm1 do
        begin
          for i2 := lbm2 to ubm2 do
            begin
              m[i1,i2] := random (x) ;
            end ;
        end ;
    end ;
```

```
  procedure WriteMatrix
    ( m : array [ lbm1..ubm1:integer
                ; lbm2..ubm2:integer
                ] of real
    ) ;
    var
      i1, i2: integer ;
    begin
      for i1 := lbm1 to ubm1 do
        begin
          for i2:= lbm2 to ubm2 do
            begin
              write (' ', m[i1,i2]:8:3) ;
            end ;
          writeln ;
        end ;
    end ;

begin
  InitializeMatrix (m1) ;
  InitializeMatrix (m2) ;
  writeln ('m1 =') ;
  WriteMatrix (m1) ;
  writeln ('m2 =') ;
  WriteMatrix (m2) ;
end.
```

Programme 7

Compilation séparée. Que se passe-t-il si l'on a plusieurs programmes qui utilisent les routines d'initialisation et d'affichage de matrice. On peut bien entendu faire du copier-coller à chaque fois. Cela pose cependant un problème. Que faire lorsque vous détectez une erreur dans une de vos routines de calcul matriciel ? Vous devrez aller les changer dans tous les programmes où vous les avez copiées. Ceci est la source de beaucoup d'erreur.

```
(* file : matrice5.p *)
program MatrixFive ;
  const
    n1 = 5; n2 = 3 ;
  procedure InitializeMatrix
    ( var m : array[ lbm1..ubm1:integer
                   ; lbm2..ubm2:integer
                   ] of real
    ) ; external ;
  procedure WriteMatrix
    ( m : array [ lbm1..ubm1:integer
                ; lbm2..ubm2:integer
                ] of real
    ) ; external ;
var
    m1 : array[1..n1,1..n1] of real ;
    m2 : array[1..n2,1..n2] of real ;
begin
  InitializeMatrix (m1) ;
  InitializeMatrix (m2) ;
  writeln ('m1 =') ;
  WriteMatrix (m1) ;
  writeln ('m2 =') ;
  WriteMatrix (m2) ;
end.
```

```
(* file : mat.p *)
module Matrices ;
  procedure InitializeMatrix
    ( var m : array[ lbm1..ubm1:integer
                   ; lbm2..ubm2:integer
                   ] of real
    ) ;
    var
      i, j : integer ;
      x : real ;
    begin
      x := 2.0 ;
      for i := lbm1 to ubm1 do
        begin
          for j := lbm2 to ubm2 do
            begin
              m[i,j] := random (x) ;
            end ;
        end ;
    end ;

  procedure WriteMatrix
    ( m : array [ lbm1..ubm1:integer
                ; lbm2..ubm2:integer
                ] of real
    ) ;
    var
      i, j : integer ;
    begin
      for i := lbm1 to ubm1 do
        begin
          for j := lbm2 to ubm2 do
            begin
              write (' ', m[i,j]:5:3) ;
            end ;
          writeln ;
        end ;
    end ;
```

Programme 8

Pour remédier à cela, on préfère extraire les routines susceptibles d'être utilisées dans plusieurs programmes, et les regrouper dans un fichier séparé. Si l'on compare avec le programme 7, dans le programme 8, les routines InitializeMatrix et WriteMatrix ont été extraites du fichier matrice5.p (à gauche) et copiées dans le fichier mat.p (à droite). Le fichier mat.p est appelé un module, et commence par le mot clé **module**, suivi du nom du module. Un module ne contient que des déclarations de type et de sous-routine. Il ne contient pas de programme à exécuter, à l'opposé de tous les programmes que nous avons vu jusqu'à présent.

On laisse cependant dans le fichier matrices5.p l'***interface*** des procédures, suivis du mot clé **external**. Une déclaration de procédure suivi du mot clé external veut dire : "Il existe une procédure (par exemple InitializeMatrix) avec un paramètres (m) qui initialize un tableau bidimensionnel". Cette déclaration permet aussi de vérifier que les types des paramètres formels et actuels correspondent. Pour compiler ce programme, on utilise la commande :

```
cosun12% pc -g -o m5 matrice5.p mat.p
cosun12%
```

Fichier d'interface. Il reste encore une source d'erreur. Supposons que nous souhaitions changer l'interface d'une des routines contenues dans le fichier matrices.p. A nouveau, nous devrions changer tous les programmes qui font appel à ces routines, et qui donc contiennent les déclarations d'interface de ces routines.

```
(* file : mat.h *)
procedure InitializeMatrix
  ( var m : array[ lbm1..ubm1:integer
                 ; lbm2..ubm2:integer
                 ] of real
  ) ; external ;
procedure WriteMatrix
  ( m : array [ lbm1..ubm1:integer
              ; lbm2..ubm2:integer
              ] of real
  ) ; external ;
```

```
(* file : matrices.p *)
module Matrices ;
  procedure InitializeMatrix
    ( var m : array[ lbm1..ubm1:integer
                   ; lbm2..ubm2:integer
                   ] of real
    ) ;
    var
      i, j : integer ;
      x : real ;
    begin
      x := 2.0 ;
      for i := lbm1 to ubm1 do
        begin
          for j := lbm2 to ubm2 do
            begin
              m[i,j] := random (x) ;
            end ;
        end ;
    end ;

  procedure WriteMatrix
    ( m : array [ lbm1..ubm1:integer
                ; lbm2..ubm2:integer
                ] of real
    ) ;
    var
      i, j : integer ;
    begin
      for i := lbm1 to ubm1 do
        begin
          for j := lbm2 to ubm2 do
            begin
              write (' ', m[i,j]:8:3) ;
            end ;
          writeln ;
        end ;
    end ;
```

```
(* file : matrice6.p *)
program MatrixSix ;
#include "mat.h"
  const
    n1 = 5; n2 = 3 ;
  var
    m1 : array[1..n1,1..n1] of real ;
    m2 : array[1..n2,1..n2] of real ;
begin
  InitializeMatrix (m1) ;
  InitializeMatrix (m2) ;
  writeln ('m1 =') ;
  WriteMatrix (m1) ;
  writeln ('m2 =') ;
  WriteMatrix (m2) ;
end.
```

Programme 9

Pour arranger cela, on rassemble dans ce qu'on appelle un ***fichier d'interface*** les interfaces des routines contenues dans le module. Comparé au programme 8, le programme 9 comporte 3 fichiers. On rassemble les interfaces de sous-routines et les déclarations dans le fichier d'interface mat.h. L'extension .h veut dire "header file". Et l'on met dans le fichier matrice6.p à la place des déclarations d'interface la commande `#include "mat.h"`. La commande #include effectue de la substitution textuelle. Tout se passe comme si le texte du fichier matrices.h se trouvait copié à la place de l'instruction `#include "mat.h"`.

3 Compilation, édition de liens et Makefile

Si l'on observe la commande de compilation du programme 8 (répétée ci-dessous), on constate que l'on recompile à chaque fois les deux fichiers matrice5.p et matrices.p. Supposons que le fichier matrices.p soit grand, et ne change plus beaucoup (il contient beaucoup de routines de calcul matriciel, qui fonctionnent correctement). Supposons que le fichier matrice5.p soit relativement court, et qu'il change souvent (c'est le programme que l'on essaie d'écrire, et il contient des fautes). Chaque fois qu'on lance la commande de

compilation, on recompile les deux fichiers. C'est du temps perdu, vu que le fichier matrices.p ne change pas d'une fois à l'autre.

```
cosun12% pc -g -o m5 matrice5.p matrices.p
cosun12%
```

Compilation et édition de lien. Pour comprendre ce qui suit, il faut savoir que ce que nous avons appelé compilation jusqu'à présent comporte en fait deux étapes : la compilation proprement dite, et l'édition de liens. La compilation proprement dite transforme un programme PASCAL en langage machine, sans se soucier des appels de procédure. L'éditeur de liens se charge seulement des appels de procédure, c'est-à-dire, il vérifie que toutes les procédures qui sont appelées ont bien été déclarée, et lie les appels de procédure à leur déclaration.

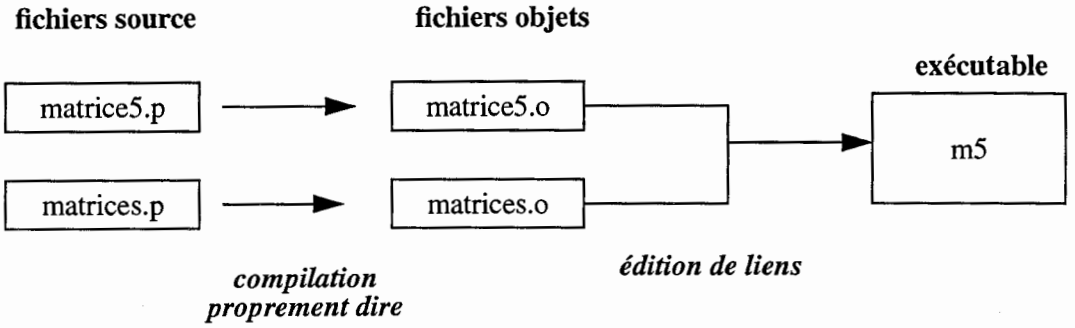


FIGURE 2. Les étapes de compilation

La compilation proprement dite transforme un fichier source en un fichier objet. L'édition de liens transforme une liste de fichiers objet en un fichier exécutable (figure 2). Par défaut, le compilateur effectue à la fois la compilation proprement dite et l'édition de liens. L'option de compilation -c demande au compilateur de ne pas effectuer l'édition de lien. Lorsque vous utilisez l'option de compilation -c, il est important que le fichier de sortie ait l'extension .o. Pour le compilateur, un fichier qui a l'extension .o représente un fichier sur lequel il ne faut faire que l'édition de liens. ; un fichier qui a l'extension .p représente un fichier sur lequel il faut faire à la fois la compilation proprement dite et l'édition de liens.

```
                    option de compilation -c qui demande au compilateur de
                    ne pas effectuer l'édition de lien
cosun12% pc -c -g -o matrices.o matrices.p
cosun12%          commande de compilation proprement dite
cosun12%          du fichier matrices.p en un fichier object
cosun12%
cosun12% pc -c -g -o matrice5.o matrice5.p
cosun12%          commande de compilation proprement dite
cosun12%          du fichier matrice5.p en un fichier object
cosun12%
cosun12% pc -g -o m5 matrice5.o matrices.o
cosun12%          commande d'édition de lien entre les fichiers
                  matrice5.o et matrices.o en un fichier exécutable
```

FIGURE 3. Les commandes de compilation

La figure 3 décrit les trois commandes qui permettent de faire la compilation des deux fichiers source matrice5.p et matrices.p en fichiers objets, et la commande d'édition de lien. La première commande effectue la compilation proprement dite du fichier matrices.p en un fichier objet matrices.o. Le compilateur n'effectue pas l'édition de liens, de par l'option de compilation -c. La deuxième commande transforme le fichier source matrice5.p en un fichier objet matrice5.o. La troisième commande effectue l'édition de lien sur les fichiers objet matrice5.o et matrices.o et produit un fichier exécutable appelé m5.

Pour raccourcir cela, on peut utiliser l'une des commandes suivantes :

```
cosun12% pc -c -g -o matrices.o matrices.p
cosun12% pc -g -o m5 matrice5.p matrices.o
cosun12%
cosun12%
cosun12% pc -g -o m5 matrice5.p matrices.o
cosun12% pc -g -o m6 matrice6.p matrices.o
cosun12%
```

FIGURE 4. Les commandes de compilation (2)

La première commande transforme le fichier source matrices.p en un fichier objet matrices.o. La deuxième commande effectue la compilation proprement dite du fichier matrice5.p, le transforme en un fichier objet, et effectue directement l'édition de lien entre ce fichier objet et le fichier objet matrices.o, pour produire un fichier exécutable appelé m5. Si vous modifiez le fichier matrice5.p, on peut recompiler en utilisant la troisième commande de la figure 4. Pour compiler un nouveau programme matrice6.p qui fait appel aux routines de calcul matriciel, on peut employer la quatrième commande de la figure 4.

Makefile. Lorsqu'un programme contient beaucoup de fichiers, il est parfois assez difficile de se rappeler quel fichier il faut recompiler. Il existe un utilitaire très pratique, appelé make, qui utilise un ***fichier de dépendances*** (quel fichiers source faut-il pour obtenir une exécutable donnée, Makefile) pour recompiler seulement les fichiers qui ont été modifiés depuis la dernière compilation. L'utilisateur écrit un fichier Makefile (figure 5). Attention : le fichier de dépendances doit s'appeler "Makefile", et pas autre chose.

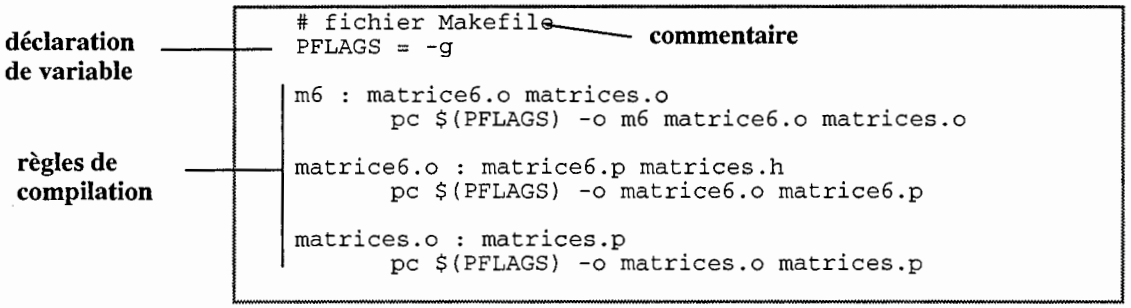


FIGURE 5. Le fichier Makefile

Dans un Makefile, les lignes en commentaires commencent par le caractère # (dièse, pound). Les variables, par convention comportant exclusivement des caractères majuscules et _, sont définies en utilisant la syntaxe `NOM_DE_VARIABLE = value`. La deuxième ligne du fichier Makefile définit une variable PFLAGS (option de compilation PASCAL), qui prend la valeur -g. Dans ce cas, l'intérêt de définir une variable est de pouvoir changer les options de compilation PASCAL d'un seul coup. Tant que l'on n'est pas sur du fonctionnement de son programme, on conserve l'option -g pour pouvoir utiliser le debugger. Une fois que le programme fonctionne, on remplace l'option -g par l'option -O (optimisation), pour améliorer la performance du programme.

Après les déclarations de variables viennent les règles de compilation. Les règles de compilation comportent 3 parties : (1) le nom du fichier que l'on souhaite créer (le fichier ***cible***), suivi de ':' ; (2) la liste des fichiers dont le fichier cible dépend (les ***dépendances***) ; (3) les commandes qui permettent de reconstituer le fichier cible à partir de ses dépendances. ***ATTENTION*** : une ligne de commande dans un fichier Makefile doit commencer par une tabulation (pas des espaces).

Par exemple, la première règle dit que le fichier exécutable m6 dépend des fichiers objets matrice6.o et matrices.o. Pour reconstituer le fichier exécutable à partir des fichiers matrice6.o et matrices.o, on utilise la commande `pc $(PFLAGS) -o m6 matrice6.o matrices.o`. Dans cette commande, la notation $(PFLAGS) veut simplement dire : le contenu de la variable PFLAGS, en l'occurrence "-g". La deuxième règle dit que le fichier matrice6.o dépend des fichiers matrice6.p et matrices.h. Pour reconstituer le fichier matrices.o, on utilise la commande `pc $(PFLAGS) -o matrice6.o matrice6.p`. On remarque que les commandes utilisées sont exactement les commandes de la figure 3.

Pour utiliser le fichier de dépendances, on utilise la commande :

```
cosun12% make m6
cosun12%
```

L'avantage de cette manipulation est qu'en utilisant la commande m6, on ne recompilera que les fichiers qui ont été modifié depuis la dernière compilation. Pour obtenir ce résultat, la commande make vérifie la date de création des différents fichiers auxquels il est fait référence dans le fichier Makefile.

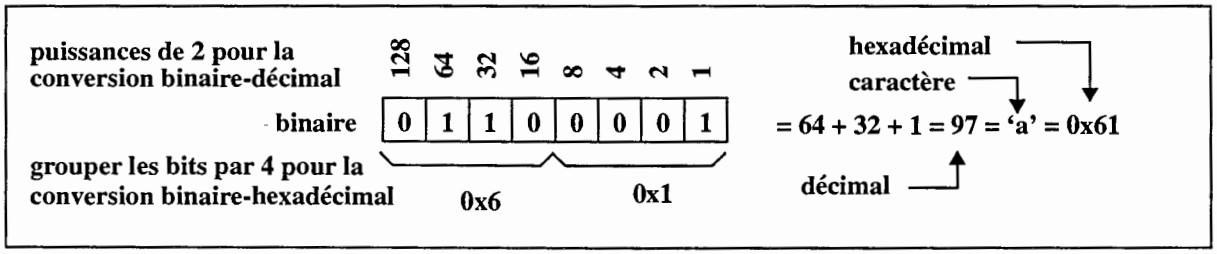
```
# fichier Makefile
PFLAGS = -g

.p.o :
        pc $(PFLAGS) -c -o $@ $<

m6 : matrice6.o matrices.o
        pc $(PFLAGS) -o m6 matrice6.o matrices.o

matrice6.o : matrice6.p matrices.h
matrices.o : matrices.p
```

FIGURE 6. Le fichier Makefile utilsant les rêgles de compilation

On peut raccourcir le fichier Makefile, en utilisant ce que l'on appelle des règles de compilation ***génériques.*** Le fichier de dépendances de la figure 6 a exactement le même effet que le fichier de dépendances de la figure 5. Les deux premières lignes sont identiques dans les figures 5 et 6. Ensuite vient une règle de compilation générique. Elle décrit comment transformer un fichier pascal en un fichier objet (règle `.p.o :`). Pour transformer un fichier avec extension .p en un fichier avec extension .o (par exemple, matrices.p en matrices.o), on utilise la commande `pc $(PFLAGS) -c -o $@ $<` . Les variables $@ et $< réfèrent à la sortie et l'entrée de la règle. Si l'on donne en entrée à la règle un fichier matrices.p, la sortie sera un fichier matrices.o, et les variables $@ et $< vaudront "matrices.o" et "matrices.p" respectivement. La règle de compilation m6 est identique dans les figures 5 et 6. Les deux dernières règles sont raccourcies : elles ne comportent que les dépendances. Pour les règles raccourcies, l'utilitaire cherche une règle générique qu'il peut utiliser. Pour la règle matrice6.o, l'utilitaire make trouve la règle qui permet de transformer un fichier .p en un fichier .o. Il détermine qu'il existe bien un fichier matrice6.p à utiliser en entrée, et exécute la commande décrite dans la règle générique .p.o.

4 Notation hexadecimale

Toutes les données dans un ordinateur sont stockées sous format binaire, c'est-à-dire des suites de bits ayant les valeurs 0 et de 1. Un caractère comporte 8 bits, un entier 32 bits, et un nombre en virgule flottante 64 bits. La notation hexadecimale groupe les bits par paquet de 4, avec des valeurs comprises entre 0 et 15. Ces valeurs sont représentées par les caractères 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a (= 10), b (= 11), c (= 12), d (= 13), e (= 14), f (= 15). Pour un caractère de 8 bits, les représentations binaires, décimales et hexadécimales sont par exemple :



La conversion binaire - décimal se fait en prenant la somme des puissances de 2 pour toutes les bits dont la valeur est 1. La conversion entier-caractère est purement une question de convention : la table standard ASCII fournit les conversions. La conversion binaire hexadécimale se fait en groupant les bits par 4.

Règlement d'utilisation des équipements informatiques

Ce règlement s'applique aux équipements suivants :

- Réseau de stations DEC Alpha dans IN 200 et leur serveur
- Réseau de stations SPARC dans IN3 et leur serveur
- Réseau de stations SPARC dans CO 020 et CO 023 et leur serveur
- Service de courrier électronique AppleMAIL sur Macintosh
- Serveur de fichiers ARTEMIS sur Macintosh.

1. Les étudiants ne doivent utiliser les installations informatiques que pour les travaux relatifs aux études à l'EPFL, conformément aux "Directives sur l'utilisation des moyens informatiques mis à disposition des étudiants pour leur travail" (projet du 7.6.1996).
2. En règle générale, l'autorisation d'accès sera délivrée avec un "username" identique au nom de famille (pour les étudiants, il sera précédé de la première lettre du prénom) et envoyée à domicile pour les étudiants. L'accès sera supprimé un mois après la date présumée du diplôme. Des dérogations peuvent être accordées sur demande.
3. Les étudiants qui redoublent ou qui changent de section doivent avertir le ou les responsables systèmes pour modifier leur situation, sous peine de voir leur compte bloqué ou détruit avant la fin effective de leurs études.
   Les étudiants qui finissent leurs études et qui restent comme collaborateurs à l'EPFL doivent communiquer leur nouveau lieu de travail (changement de groupe).
4. Les collaborateurs qui quittent l'EPFL et les étudiants qui quittent l'EPFL en cours d'étude (ou qui sont en congé dans une autre institution) sont tenus d' annoncer leur départ afin de régulariser leur situation (sauvegarde des comptes en particulier).
5. Chaque étudiant dispose, pour chaque année académique, d'une tranche gratuite de pages sur les imprimantes du département. Une extension gratuite peut être obtenue sur demande signée par le responsable du projet. Le dépassement du quota autorisé sera facturé à l'étudiant qui s'engage à payer ses factures.
6. De façon générale, les utilisateurs doivent préserver le bon fonctionnement des équipements mis à disposition. Ils veilleront à ne pas obstruer les orifices de ventilation.
   Dans les salles d'ordinateurs, il est interdit:
   - de fumer, de manger et de boire
   - de déposer des objets volumineux sur les tables et sur les ordinateurs
   - de déplacer les ordinateurs ou les claviers à cause du risque de détérioration des câbles
7. L'utilisation de logiciels de tout genre, leur copie, la copie de manuels ou d'autres documentations sans autorisation préalable de l'auteur est illicite selon la loi Suisse sur le droit d'auteur du 1er juillet 1993.

   Les utilisateurs ne doivent pas posséder, afficher ou diffuser de documents informatiques ou logiciels en infraction avec le Code Pénal Suisse notamment l'infraction contre les moeurs (en particulier l'art. 197).

   Toute opération de "piratage" est illicite et représente un dommage à la propriété. Chaque utilisateur engage sa propre responsabilité juridique.

   Les comptes et les mots de passe sont personnels. Il est interdit de les communiquer à d'autres personnes.
8. La documentation mise à disposition dans les salles d'ordinateurs ne doit en aucun cas sortir de celles-ci. Les utilisateurs s'engagent à la remettre en place avant de quitter la salle.
9. Toutes les demandes concernant l'utilisation ou les modifications des comptes sont à adresser aux responsables systèmes:

   - DECstation Alpha M. Roland Wuillemin
     Tél. interne: 2586 + Rech.
     Bureau INJ 037 (rez)
     E-Mail: roland.wuillemin@di.epfl.ch
   - SPARCStation (IN3) M. Stéphane Ecuyer
     Tél. interne: 2795 + Rech.
     Bureau INJ 041 (rez)
     E-Mail: stephane.ecuyer@di.epfl.ch
   - SPARCStation (CO020/023) M. Laurent Desimone
     Tél. interne: 2637 + Rech.
     Bureau INJ 039 (rez)
     E-Mail: laurent.desimone@di.epfl.ch
   - Mac/AppleMail M. Sourythep Samoutphonh
     Tél. interne: 4228 + Rech.
     Bureau INN 021 (rez)
     E-Mail: sourythep.samoutphonh@di.epfl.ch

   Cette demande est à déposer au secrétariat du département (bâtiment INM, 1er étage). Les étudiants présenteront leur carte d'étudiant valable. Les étudiants n'appartenant pas à la section d'informatique ou à la section systèmes de communication doivent joindre une justification signée par le professeur du DI responsable du cours ou du projet.
10. En cas de non respect de ce règlement, le compte de l'étudiant sera bloqué. Le contrevenant s'expose également à la prise de mesures disciplinaires. Ces mesures dépendront de l'importance de l'infraction. Au nombre des sanctions pouvant être prises figurent notamment le blâme et l'exclusion de l'Ecole. Ces sanctions internes sont bien entendu sans rapport aucun avec d'éventuelles sanctions pénales qui pourraient être intentées en raison de délits.
    (Cf directives mentionnées sous le point 1.)

14.10.1996/SIDI/RW/cb