Programmation structurée et algorithmes de base en Pascal

Miage1, IF2, DEUG sciences

Patrice Effi BROU
UFR Mathématiques et informatique

Decembre 2003

Chapitre 1

Sommaire

1 THEORIE DES LANGAGES ET COMPILATION

1.1- Notion de langage

1.1.1- Définition

Un *Langage* peut se définir comme un ensemble de termes et de règles opératoires permettant de communiquer.
Selon le degré d'ambiguïté, on peut classer les langages de la manière suivante :

- les langages **naturels***;*
- les langages **formels** (groupe auxquels appartiennent les langages de programmation).

1.1.2- Spécification d'un langage formel

Un langage formel se définit comme un ensemble constitué de 2 sous-ensembles: le Vocabulaire et la Grammaire:

Le *Vocabulaire* est un ensemble composé de l'alphabet et du lexique.
La *Grammaire* précise le mode de construction des phrases du langage.

1.2- Classification des langages de programmation

Cette classification peut se faire selon le niveau de compréhension du langage par la machine.

1.2.1- Le langage machine

C'est un langage spécifique à chaque machine. Il n'est donc pas portable. Il est généralement écrit en *binaire* (suite de 0 et/ou 1) et donc difficilement maîtrisable par l'homme. Cette manière de programmer avec les 0 et 1 s'appelle la *microprogrammation*.
Le *langage machine* est un langage de bas niveau et qui est directement compréhensible par la machine.

1.2.2- Le langage assembleur

Moins contraignant que le langage machine, le *langage assembleur* utilise les *codes mnémoniques* du processeur pour désigner des instructions.

Exemples :

STA	(i.e store A)	MUL
LDA	(i.e load A)	JMP
ADD		

Le *langage assembleur* n'est pas portable. Ces codes doivent être traduits en binaire pour être exécuter par le processeur (c'est le rôle du décodeur d'instruction).

1.2.3- Les langages évolués

Ce sont des langages indépendants de tous types d'ordinateur et dont les instructions sont proches du langage humain. Un traducteur (compilateur-interpréteur) se charge par la suite de convertir ces instructions pour la machine.

Exemples : Pascal, C, C++, Java, Prolog, Cobol, Basic.

	Pascal	*C*
Afficher à l'écran	writeln	printf
Saisir une information	readln	scanf

Quelques instructions du langage Pascal et du C

On peut classer ces langages. On a les langages :
- *procéduraux* (Pascal, C …)
- *logiques* (Prolog, lisp …)
- *orienté objet* (C++, Java …)

1.3- Notion de compilation

I.3.1- La notion de programme

Conçu pour résoudre les problèmes, l'ordinateur le fait par une suite d'instructions écrite par un programmeur. Cette suite d'instruction est appelée *Programme*. Le programme doit être écrit dans un langage de programmation évolué. Le problème qui se pose est le suivant : l'ordinateur ne comprend que le langage binaire. Il faut donc traduire le programme d'origine en programme objet. Pour cela, on utilise un outil de production de programme appelé *Compilateur* ou *Interpréteur*.

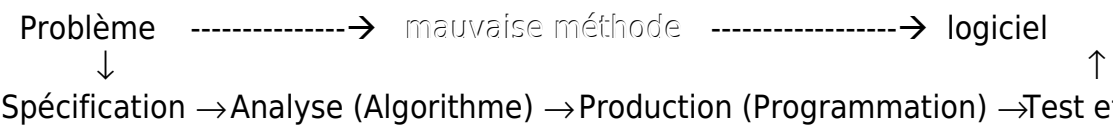
1.3.2- La compilation

La *compilation* est la traduction d'un programme source en un autre programme équivalent en langage machine.

1.3.3- L'interprétation

C'est la traduction d'un programme source en un programme exécutable instructions par instructions. Contrairement à la compilation, le programme se traduit par morceau.
Exemple : Java, Basic …

1.3.4- Le cycle de vie d'un logiciel



Le cycle de vie d'un logiciel

- le *Problème* est censé demander un traitement. Il est donc nécessaire de bien le comprendre.

- le *Logiciel* est censé résoudre un problème donné, c'est-à-dire faire le traitement demandé.

Les différentes étapes de la programmation sont :
- les *spécifications* du programme. Il s'agit dans un énoncé de préciser les hypothèses (les entrées) et les conclusions (les résultats attendus).
- pour passer de la spécification à la programmation, il est nécessaire de passer par une phase d'*Analyse* pour avoir une description ordonnée des calculs devant permettre de résoudre le problème : c'est l'*algorithme*.

- à partir de l'Analyse, on produit à l'aide d'un programme cible, le *programme*.
- avant l'utilisation du Logiciel, on fait des *tests* et des *corrections* pour vérifier son bon fonctionnement et le respect des spécifications.
On peut ensuite passer à l'utilisation.

I.3.5- Les différents modes de programmation

Les modes de programmation les plus courants sont :
- la *programmation séquentielle* ou toutes les actions sont des actions insécables.
- la *programmation structurée* qui introduit les structures de contrôle (*si* « condition » *alors* « action1 » *sinon* « action », répéter).

- la *programmation modulaire* permet de découper un problème donné en sous problèmes, de manière à pouvoir les programmer séparément.
- la *programmation Orientée Objet* permet d'associer à chaque objet crée une liste d'actions possible.

I.3.6- La lisibilité du code

Exemple : programme 1

```
Program cylindre;
Uses crt ;
Var
L, m, h : real ;
Begin
readln(l) ;
readln(h) ;
m := pi*l*l*h ;
writeln(m) ;
readln ;
End.
```

```
Program cylindre ;
Uses crt ;
Var
  rayon, hauteur, volume : real ;
Begin
   writeln('Saisir Rayon : ') ;
   readln(rayon) ;
   writeln('Saisir Hauteur : ') ;
   readln(hauteur) ;
  volume :=pi*rayon*rayon*hauteur ;
  writeln(volume) ;
   readln ;
End.
```

Pour la lisibilité du code, il faut faire une bonne présentation (bonne convivialité, choix des identificateurs pour les données, conversion…).
D'une manière générale, le choix du langage, se fera en fonction du langage informatique du programmeur.

Pour le cours d'Initiation à la Programmation, le langage retenu est le PASCAL. Pourquoi le Pascal ? C'est un langage de programmation fortement structuré en bloc (procédural, simple et exigeant). En plus, Delphi c'est du Pascal associé à une interface Windows.

2 PRESENTATION DU LANGAGE PASCAL

Le langage Pascal à été inventé par NIKLAUS WIRTH en 1968.

2.1 Les éléments du langage

211 Les mots du langage

Mots réservés

And	Else	In	Or	To
Asm	End	Inherited	Packed	Try
Array	Except	Inline	Procedure	Type
Begin	Exports	Interface	Program	Unit
Case	File	Label	Record	Until
Const	Finally	Library	Repeat	Uses
Constructor	For	Mod	Set	Var
Destructor	Function	Nil	Shl	While
Div	Goto*	Not	Shr	With
Do	If	Object	String	Xor
Downto	Implementation	Of	Then	

Liste des mots réservés de Pascal

Chiffres (0,1,..,9)
Les opérateurs (+, -, /, *, <, >, >=, <=...)
{Les commentaires}, (*les commentaires*)
Chaque instruction se termine par un point virgule (;).

212 Constantes et Variables

Les données sur lesquels travaille le pascal peuvent être des variables ou des constantes.

Exemple :

0,**20** est une constante de type réel.

'Au revoir' est une constante de chaînes de caractères.

Les variables contrairement aux constantes ne sont pas connues par leurs valeurs mais par leur nom encore appelé identificateur.

2.1.3 Identificateur

Un identificateur est constitué par une lettre ou le signe souligné suivi de toute combinaison de lettre, de chiffres ou de signes soulignés.

Exemple :

Volume_1 (correct)
_volume1 (correct)
Volume 1 (incorrect)
Pour être valide, chaque identificateur de variable doit être déclaré.

Syntaxe de déclaration d'une variable

Var
< Identificateur> : < type > ;

Exemple :
Var
Rayon, hauteur, volume : Real ;

Syntaxe de déclaration d'une constante

Const
<Identificateur >=valeur ;

Exemple :

```
Age_maximum=55;
Bj='bonjour';
Max_index=1000;
```

2.1.4 L'affectation

La déclaration d'une variable ne lui donne pas de valeur. Elle ne fait que demander au compilateur de réserver un espace mémoire pour ranger la valeur qui sera saisie ou affectée.
L'affectation est une instruction qui consiste à ranger dans l'identificateur de la partie gauche le résultat de l'expression ou la valeur de la partie droite.

Syntaxe de l'affectation

<variable>:=expression;

Exemples:

```
Volume :=rayon*rayon*hauteur*pi ;
i :=1;
trouve:=false ;
```

2.1.5 Les opérations d'entrées/sorties

Elles s'effectuent grâce aux primitives ***writeln*** et ***readln*** ou ***write*** et ***read.***
Elles s'appliquent sur les types de base ***entier, réel, caractère, logique.***

Syntaxe des opérations d'entrée/ sorties

```
Write(45);     affiche 45
Write('bonjour');   affiche bonjour
Write('volume');    affiche  volume
Write('le volume du cylindre est', volume); affiche le volume du cylindre est
```

La primitive Read permet d'initialiser une variable pour la saisie au clavier.

Syntaxe

Read(identificateur1, identificateur2…);

Exemples:

```
Read (rayon);
Read (a, b);
```

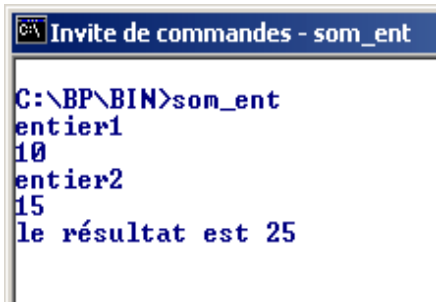
Structure d'un programme simple en PASCAL

Program *<identificateur> ;*
Const
<Déclaration de constantes> ;
var
<Déclaration de variables> ;
Begin
<Instructions>;
End*.*

Exemple: un programme qui calcule la somme de deux entiers

```
Program somme_entier;
Var
     Entier1,entier2,addition:integer;
Begin
     Writeln('entier1');
     Readln (entier1);
     Writeln ('entier2');
     Readln (entier2);
     Addition: =entier1+entier2;
     Writeln('le résultat est ', addition);
     Readln;
End.
```

Exécution du programme



2.2 Les Types de Base

2.2.1 Les entiers (**Integer**)

[-32768 , 32767] → Domaine de valeur
Opérateurs : + ,- ,* , Div,(≡) mod.
Div : division
Mod : reste de la division entière

Quelques Fonctions

Abs = valeur absolue
Sqr = carré (square)
Sqrt = Racine carré

Exemple:
45 div 6 =7
45 mod 6 = 3
abs(nombre) = | nombre |
Sqr (a) = a^2
Sqrt (a) = $\sqrt{a}$

2.2.2 Les booléens (**boolean**)

Le domaine de valeur de type booléen est l'ensemble *true, false* avec *false<true.*
Les opérateurs

`Not` → non
`Or` → ou
`And` → et
`Xor` → ou exclusif

Exemple :

```
Var
R,a,b,c,d,e:Boolean;
begin
R:=a or(b and c)xor(d and e);
```

2.2.3 Les Réels (**Real**)

Domaine de valeur : $2{,}9\ 10^{-39}$ à $1{,}7\ 10^{38}$
Les opérateurs
(-)opposé (opérateur unaire)
+ addition
− soustraction
/ division
* multiplication
} opérateurs Binaires

Quelques fonctions:
Abs, sqr, sqrt, sin, cos, arctan, ln, exp, frac, int, round, trunc

2.2.4 Les caractères et chaînes de caractères (**char**- **string**)

Les caractères servent essentiellement à des opérations de lectures et d'écriture. Le Pascal gère 256 caractères du code ASCII. L'ensemble des caractères est ordonné.
Le mot clé pour la déclaration est ***char***.

Exemple :

```
Var
Rep: char;
```

La fonction *ord('a')* donne la position du caractère *'a'*.

Upcase transforme une minuscule en majuscule.

Exemple :
`Write(Upcase'b')); affiche B`

Le type chaîne utilise le mot clé '***string'***

```
Var
Nom : string[20] ; {taille limitée à 20 caractères}
Commentaire : string ;
```

On peut concaténer deux chaînes de caractères: 'oui'+'non'= 'oui non'
La fonction *length(string)* retourne la longueur du mot.

2.3 – Les structures de contrôle

2.3.1 Les instructions composées

```
Begin
    Instruction1;
    Instruction 2;
End.
```

Une instruction composée est un ensemble d'instruction commençant par '*Begin'* et se terminant par *end.*

2.3.2 Les instructions conditionnelles

Pour faire des choix dans un programme on dispose des instructions conditionnelles

Syntaxe de l'instruction conditionnelle

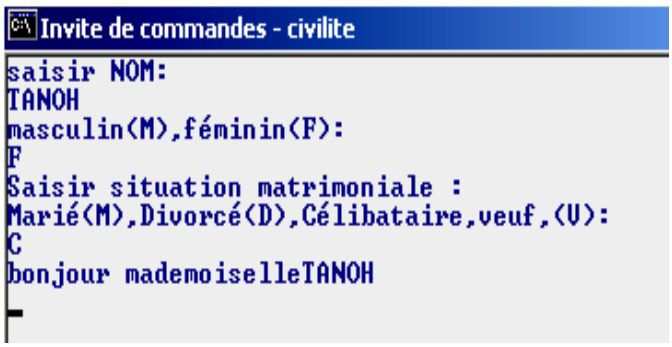
If *(condition)* ***then*** *instruction1*
Else *instruction2;*

Exemple: Un programme qui demande le nom, le sexe et affiche la civilité.

```
Program civilite;
Uses crt;
Var
  Nom: string [20] ;
  etat,sexe : char;
Begin
  Clrscr;
  Writeln('saisir NOM:');
  readln(nom);
  Writeln('masculin(M),féminin(F):');
  Readln(sexe);
  Writeln ('Saisir situation matrimoniale :');
  Writeln('Marié(M),Divorcé(D),Célibataire,veuf,(V):') ;
  Readln(etat);
  If (sexe='M')then writeln('bonjour monsieur',nom)
```

```
  Else
     if (etat='C')then writeln('bonjour mademoiselle',nom)
     Else writeln ('bonjour Madame', nom);
  Readln;
End.
```

Exécution



Remarque

1) Si l'instruction 2 est vide, on peut ne pas écrire *'Else'* on aura alors
   If (condition) then instruction 1;
2) On peut imbriquer des'*if*' et des '*else*'

2.3.3 Instructions de choix multiple

Syntaxe

```
 Case N of
         1:instruction1;
         2: instruction 2;
         3: instruction 3;
         Else
           Instruction 4
End;
```

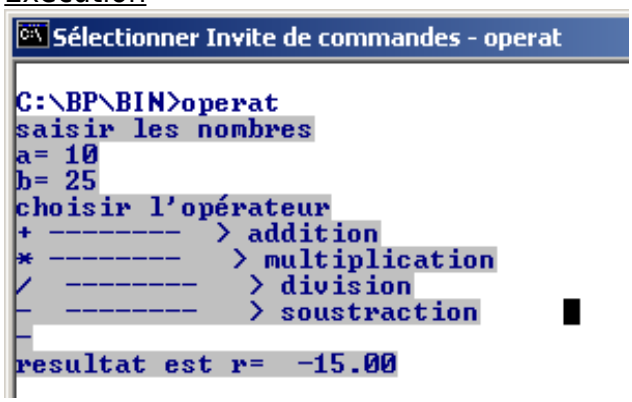
(Ici N est un entier et prend les valeurs 1.. n)

Exemple : Programme de calcul d'opération au choix sur des réels

```
Program calcul;
Uses crt;
Var
 a,b,resultat:real;
operateur:char;
Begin
  Clrscr;
  Writeln('saisir les nombres');
  Write('a= ');readln (a);
  Write('b= ');readln (b);
  Writeln ('choisir l''opérateur');
  Writeln ('+ --------  > addition');
  Writeln ('* --------  > multiplication');
```

```
  Writeln ('/  --------   > division');
  Writeln ('-  --------   > soustraction');
  Readln (operateur);
  Case operateur of
          '+': resultat: =a+b;
          '-': resultat: =a-b;
          '*': resultat: =a*b;
          '/': resultat: =a/b
           else writeln ('opérateur inconnu');
  End;
  Writeln ('resultat est r=  ', resultat: 4:2);
  Readln;
  End.
```

Exécution



L'instruction `Writeln('resultat est r=', resultat: 4:2);` permet de formater l'affichage du nombre réel sur 4 positions avec deux chiffres après la virgule

23.4 Les itératives

Les instructions itératives (les boucles) permettent de répéter une instruction un certain nombre de fois.

- **For**

Syntaxe

For i: = min to max do instruction

Exemple 1: calcul de la somme des 100 premiers entiers non nuls

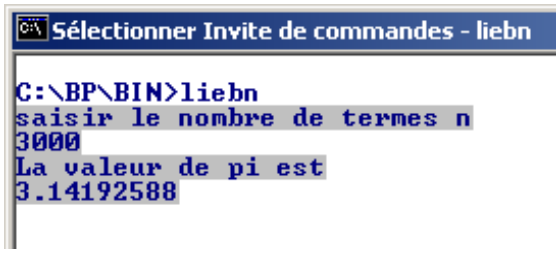
```
Program som_entier;
Uses crt;
Var
Som, n: integer;
Begin
clrscr;
Som: = 0;
For n: = 1 to 100 do
Som : = som+n ;
Writeln ('La somme des 100 premiers entiers non nuls est',som) ;
Readln;
End.
```

Exemple 2 : $\sum (-1)^n * 1/(2n+1)$

Calculer la série ci dessus
Montrer qu'elle tend vers π/4
En déduire la valeur de π

```
Program Liebnitz;
Uses crt;
Var
 i, n :Integer;
Serie, Signe: Real;
Begin
   Clrscr ;
   Writeln('saisir le nombre de termes n');
   Readln (n);
   Signe:= 1;
   Serie:= 1;
   For i:= 1 to n do
     Begin
       Signe := -signe ;
       Serie :=serie+(signe/(2*i+1)) ;
     End ;
   Writeln('La valeur de pi est') ;
   Write (4*serie:10:8);
   Readln;
End.
```

Execution



TAF : afficher les termes par série de 20 avec le message '*Appuyer une touché pour continuer....*'

- **While**

Tant qu'une condition est vraie, exécuter l'instruction. Le test de la condition est faite au départ. Si la condition est fausse l'instruction n'est pas exécutée.

Syntaxe

While (condition) do instruction

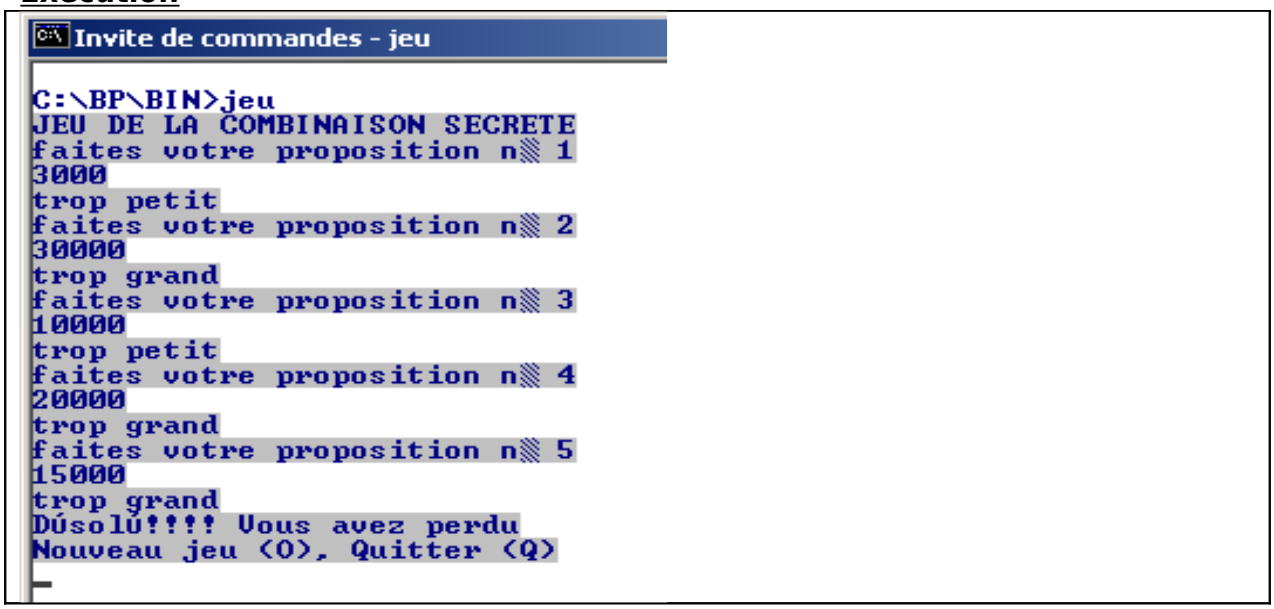
Exemple : le programme du nombre secret à trouver avec une possibilité de cinq essais pour le joueur.

```
Program Jeu;
Uses crt;
Var
  Secret,nombreprop,essai, i :integer ;
  Choix: char;
Begin
  Clrscr;
  Randomize;
```

```
  Secret: =random (32000);
  Repeat
    Clrscr;
    Writeln('JEU DE LA COMBINAISON SECRETE') ;
    nombreprop :=-1 ;
    essai :=5;
    i:=1;
    While ((nombreprop<>secret) and (essai>0)) do
      Begin
        Writeln('faites votre proposition n° ', i ) ;
        Readln(nombre prop) ;
        Essai:= essai-1;
        i:= i+1 ;
        If nombreprop<secret then writeln ('trop petit')
        Else if nombreprop > secret then writeln ('trop grand');
      End;
   If(nombreprop=secret) then writeln('bravo!! Vous avez gagné')
   Else writeln ('Désolé!!!! Vous avez perdu');
    Writeln ('Nouveau jeu (O), Quitter (Q)');
   Readln (choix) ;
   Until(choix='Q') or( choix='q')
End.
```

Les fonctions *random*(x: *integer*) et *Randomize* servent respectivement à générer un nombre de manière aléatoire et à initialiser le générateur de nombre aléatoire

Exécution



Repeat

Syntaxe

```
Repeat
  Instruction_1 ;
  _ _ _ _ _ _ _ _
  Instruction_ n ;
until (condition) ;
```

Cette syntaxe est utilisée dans l'exemple précédent.

3 LES TYPES NON -STANDARDS

3.1 Les types énumérés

Dans un type énuméré il faut donner la liste ordonnée de toutes les valeurs possibles. Toutes variable qui sera crée dans ce type ne pourra prendre que l'une des valeurs citées.

Syntaxe

```
TYPE
  Jour = (lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);
```

Dans la partie déclaration de variable on pourra écrire

```
Var
  Jourarrive : jour ;
```

Remarque

Jourarrive : = 100 ;→erreur de compilation

3.1.1 les opérations possibles

- L'affectation

```
TYPE
Jour =(lundi,mardi,mercredi,jeudi, vendredi, samedi,dimanche);
Var
Jourarrive, jourrepos : jour ;
Begin
jourrepos : = mercredi ;
jourarrive : = jourrepos ;
```

- Les fonctions

PRED(--)⇒précédent

SUCC(--)⇒successeur

ORD(--)⇒indice dans la liste en commençant par 0

Exemple :

ORD(mercredi) est égal à 2

- les comparaisons

Exemple : (Lundi < mercredi)→vrai

- Dans les boucles

Exemple:

```
For jourtravail :=lundi to vendredi do
  Begin
  Writeln('Boulot- DoDo');
  End ;
```

Remarque
Les types prédéfinis ne permettent pas l'utilisation des fonctions d'entrées sorties standard: writeln, readln
Exemple :
Writeln(jour arrive) ;→ERR de compilation

exemple récapitulatif

```
Program listejour ;
Uses crt ;
type
  Jour=(dimanche,lundi,mardi,mercredi,jeudi,vendredi, samedi);
Var
  Joursemaine :jour ;
Begin
  For joursemaine: = dimanche to samedi do
    Writeln(ORD (joursemaine)+1,'e jour de la semaine') ;
Readln;
End.
```

3.2 Le type intervalle

C'est un type particulier défini à parti d'un type énuméré ou d'un type standard, en précisant les bornes.

Syntaxe

TYPE

Jweekend= vendredi.. Dimanche ;
numeromois=1.. 12
Minuscule= 'a'.. 'z'

Toutes les opérations sur les fonctions de types énumérés sont applicables sur les types intervalles.

3.3 Les tableaux (array)

3.3.1 Notion de tableau

Un tableau est une collection de variables de même type.
Exemple :
Les coordonnées d'un vecteur, les relevés de températures journalières

100	-90	121	100	183	94	31	66
1	2	3	4	5	6	7	8 →indices

Syntaxe

```
TYPE
    Vecteur=array[1..8]of real;
 Var
    Vec:vecteur;
```

Autre méthode

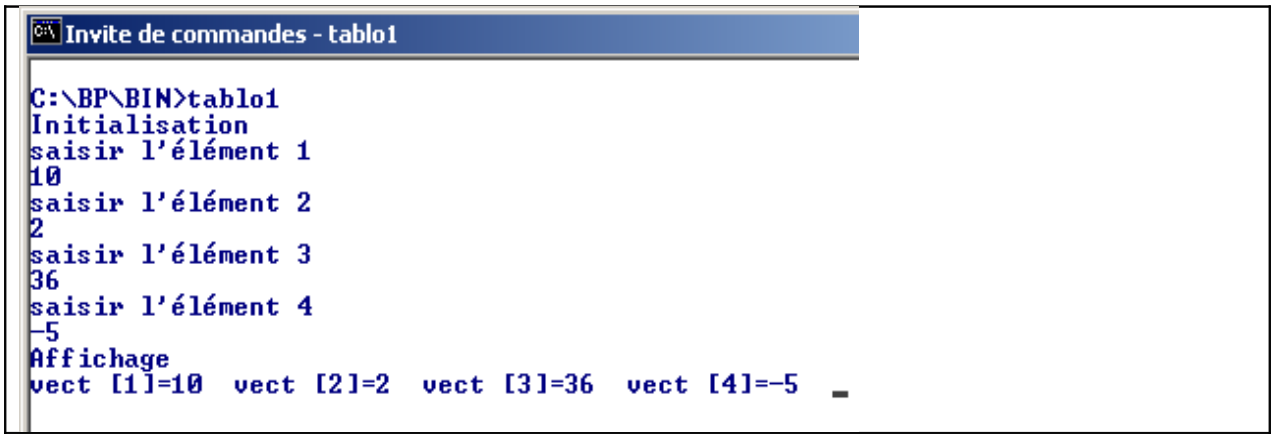
Var

Vect1: array [1.. 8] of real;

vect[1] représente la cellule n°1 du tableau
i := 7
 Vect [i] est égal à vect[7]

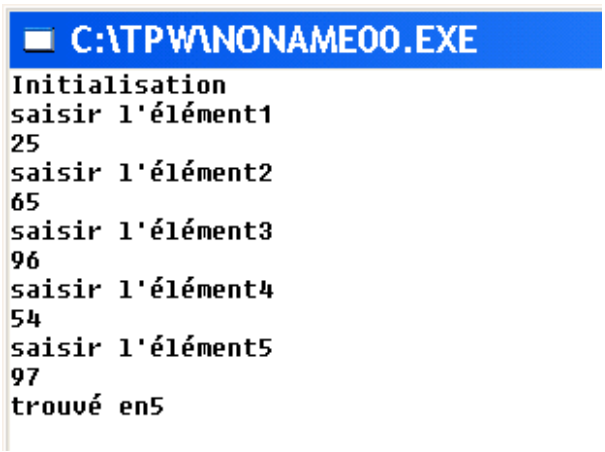
3.3.2 Initialisation et affichage d'un tableau

```
Program init_affiche;
Uses crt;
Const
  Maxtab=5;
TYPE
  Tab= array[1..maxtab]of integer;
Var
  Vect: Tab;
  i:Integer;
Begin
  Writeln('Initialisation');
  For i:=1 to maxtab do
    Begin
     Writeln('saisir l''élément', i) ;
     Readln (vect[i]);
     End;
     Clrscr;
     Writeln('Affichage');
     For i:=1 to maxtab do
       Begin
        Write('vect[',i,']=',vect[i],' ');
    End;
  Readln;
End.
```



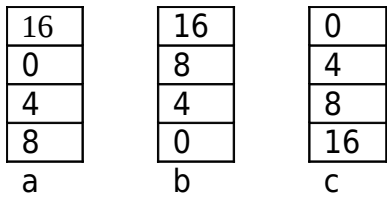
3.3.3- Recherche dans un tableau (La recherche séquentielle)

```
Program recherche1;
  Uses crt ;
  Const
maxtab=5 ;
TYPE
  Tableau= array [1.. maxtab] of real;
Var
  Tab: tableau;
  i: integer ;
Begin
  Writeln('Initialisation');
  For i:=1 to maxtab do
     Begin
       Writeln('saisir l''élément',i) ;
       Readln (Tab[i]);
     End;
  i:=1 ;
  While (Tab[i]<>97) and (i<=maxtab) do i:= i+1;
  If (Tab[i]=97) then Writeln ('trouvé en', i)
  Else
       If (i>maxtab) then Writeln ('pas trouvé');
  readln;
End.
```



Remarque
L'élément recherché est 97
Avec ‘’***while***’’ on arrête de chercher une fois trouvée ce qui n'est pas le cas avec for.
TAF : trouver la version du programme qui utilise ***for***

3.3.4 Les méthodes de tris

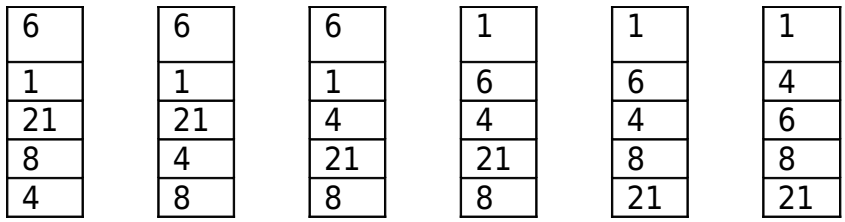


a	b	c
16	16	0
0	8	4
4	4	8
8	0	16

a_ tableau non trié
b_ trié descendant
c_ trié ascendant

Quelques méthodes de tris
Tri par insertion
Tri à bulles
Tri par sélection
…

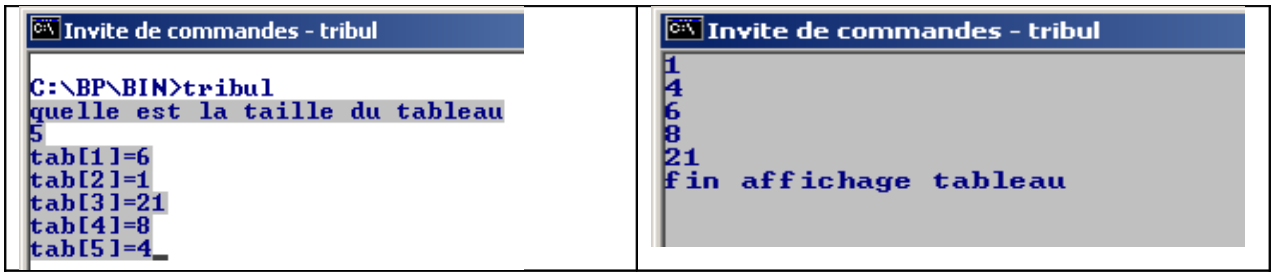
Application : « le tri à bulles »
Cette méthode de tri permet de trier un tableau en faisant remonter les petits nombres au sommet du tableau (comme des bulles).



6	6	6	1	1	1
1	1	1	6	6	4
21	21	4	4	4	6
8	4	21	21	8	8
4	8	8	8	21	21

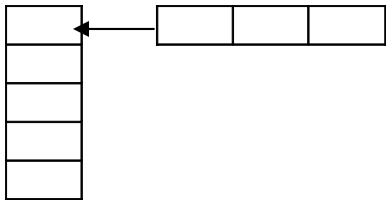
```
Program tri_bulle2;
  Uses crt;
const
  Max_tab=100;
Var
  n,i,j,temp :integer ;
  Tab= array [1.. maxtab] of integer ;
Begin
{ initialisation du tableau  }
  writeln ('Quelle est la taille du tableau ?');
  Readln(n);
  For i:=1 to n do
    Begin
      Writeln('saisir tab[‘,i,’]= ‘);
      Readln (tab[i])
    End;
{tri du tableau}
```

```
  For i:=1 to n-1 do
    For j:= n downto i+1 do
      If tab[j-1] > tab [j] then
        Begin
          temp:= tab[ j-1 ] ;
          Tab[ j-1]:= tab [j];
          Tab[j]:= temp ;
        End ;
{ Affichage du tableau}
  Clrscr;
  For i:=1 to n do
  Writeln (tab[i]);
  writeln('fin affichage tableau') ;
  Readln ;
End.
```



3.2.5- Tableau de tableaux (tableaux de plusieurs dimensions)

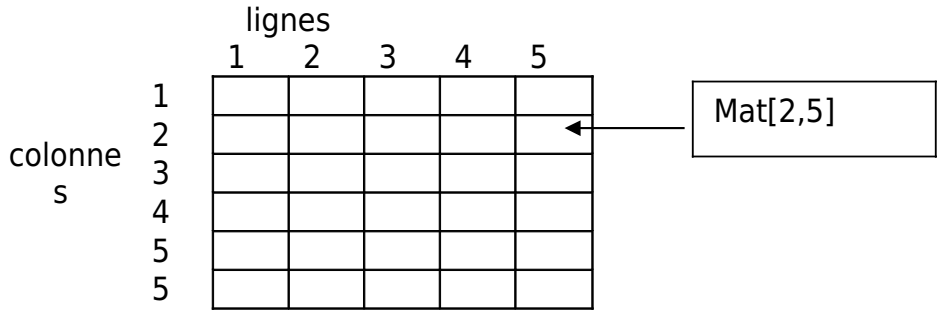
On peut se représenter un tableau de deux dimensions comme un tableau à une dimension dont les cellules sont des tableaux.



La déclaration dans ce cas se fait de la manière suivante:

TYPE
Tab_entier=**array** [1.. n] of integer ;
Tab_de_tab=**array** [1.. p] of Tab_entier ;
VAR
Tab2 : tab_de_tab ; Autre méthode

Un tableau à deux dimmensions peut être representé par une matrice.




```
TYPE
Tab_de_tab=array [1..n, 1..p] of integer;
VAR
Tab2:Tab_de_tab;
Begin
Tab2 [1,3]:=5;
Writeln(Tab2[3,6]);
Readln(Tab2[2,3]);
```

TAF : Ecrire un programme qui calcule le produit de deux matrices.

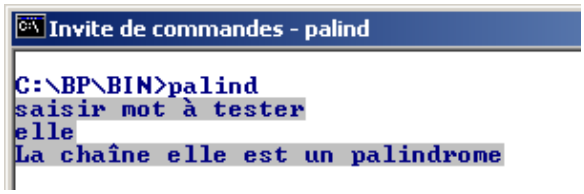
3.5.6- le type chaîne (string)

Exemple: Ecrire un programme qui teste si un mot est un palindrome

```
Program test_palindrome;
  Uses crt ;
```

```
Var
  palindrome: boolean;
  Longueur, i : Integer;
  Mot : string ;
Begin
  Palindrome:=true;
  Writeln('saisir mot à tester') ;
  Readln(mot);
  Longueur:=length(mot) ;
  For i:=1 to (longueur div 2)do
    Palindrome :=(Palindrome)and( mot[i]=mot[longueur-i+1]);
  If (palindrome = true) then writeln('La chaîne', mot,' est un
  palindrome')
  Else writeln('ce mot n''est pas un palindrome') ;
  Readln;
End.
```

Execution



3.4 Les enregistrements

Dans un tableau les données sont de mêmes types. Pour stocker des données de types différents dans une même structure de données, on utilise les enregistrements. Le langage Pascal permet de définir les enregistrements grâce au mot '***record'***

3.4.1 Déclaration d'un enregistrement

Méthode 1

Var
Identidicateur_variable : record
Champ1: type1 ;
Champ2 : type2 ;
…
Champ n: type n ;
End*;*

Methode 2

TYPE
Identificateur_type= record
Champ1: type1 ;
Champ2: type2 ;
…
Champ n type n ;
End ;

Exemple : Déclaration d'un client dont on retient le nom (string [25]), le prénom (string [25]), le téléphone (string [15]) et le solde(real) ;

1ere solution

```
var
 client1 : record ;
      Nom: string[25];
   Prenom: string [50];
Téléphone: string [15];
    Solde: real;
      End;
```

2e solution

```
Type
     Client= record
     Nom: [string 25];
     Prénom:  [string 50];
     Téléphone: [string 15];
     Solde: real;
End ;
var
client1 : client ;
```

3.4.2- Utilisation d'un type enregistrement

Le point (".")permet de référencer les composantes(les champs de) l'enregistrement.

Exemple :

`Client1.nom` représente le champ nom de l'enregistrement client1.

On peut écrire des ordres de lecture et d'écriture sur les champs.

Exemple:

```
Readln (client1.telephone);
Writeln (client1.solde);
```

On peut avoir un tableau d'enregistrement

Exemple :

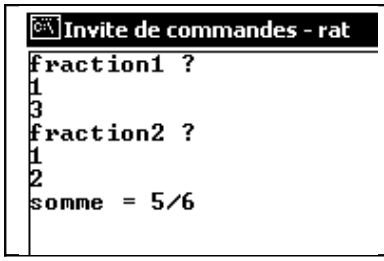
`Clientele:array[1..100] of client;` {C'est un tableau de 100 clients }

Clientèle [i] .nom

Exemple complet: Ecrire un programme en Pascal qui calcul (1/2)+(1/3)= 5/6

```
program fraction;
uses crt;
type rationnel = record
                  num, den : integer
                 end;
var x, y, somme : rationnel;
begin
    clrscr;
    writeln('fraction1 ?');
    readln(x.num, x.den);
    writeln('fraction2 ?');
    readln(y.num, y.den);
    somme.num := x.num * y.den + x.den * y.num ;
    somme.den := x.den * y.den;
    with somme do
       writeln ('somme = ', num, '/', den);
    readln;
end.
```

Exécution



TAF Modifier le programme pour reduire les fractions :
¼+ ¾ = 4/4

3.4.3 - L'instruction ***with***.. ***Do***

Elle permet de mettre en facteur le nom de la variable pour ne travailler qu'avec les champs.

Exemple: Client1.nom

```
whith Client1 do
write(nom);
```

au lieu de `write(client1.nom);`

exemple :

```
program client_liste ;
Type
  Client=record
       Nom, prenom, profession: string [25 ] ;
       Telephone: string [ 15 ] ;
       Solde: real;
  End;
Var
   Client1:client;
Begin
   Writeln('saisie de client 1');
   With client1 do
   Begin
     Write ('Nom');
     Readln(Nom);
     Write('prenom');
     Readln (prenom);
     Write ('profession');
     Readln (Profession);
     Write ('Telephone');
     Readln (Telephone);
     Write('Solde');
     Readln (Solde);
   End;
   Readln;
End.
```

3.5 – Les fichiers

Un fichier un ensemble de données écrit sur support lisible par l'ordinateur (disquette, cd, disque dur,...). Il peut contenir caractères (fichier texte) des textes des programme ou des valeurs (fichiers de données).

3.5.1 Création d'un fichier

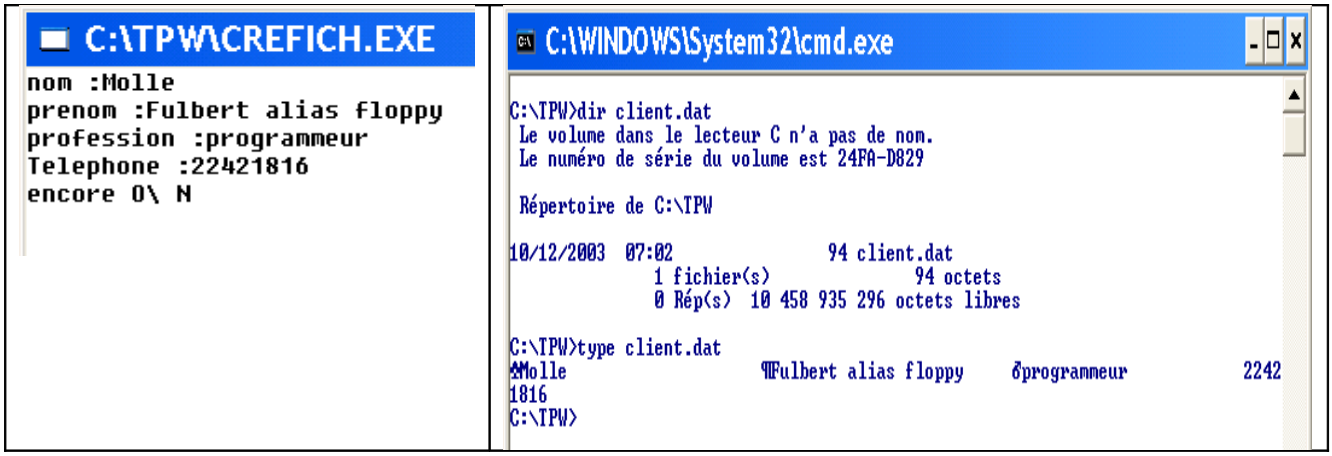
- déclaration d'un fichier logique
- création d'un fichier physique

```
personne=record
 Nom,prenom,profession: string[25 ] ;
Telephone: string[15];
End;
Var
Contact: personne;
Carnet: file of personne;
```

`Carnet: file of personne;` précise que la variable carnet est un fichier dont les enregistrements sont de type personne.
`Contact: personne;` est la variable qui nous permettra d'initialiser les composantes d'un enregistrement avant l'écriture dans le fichier.

```
Program Creation_fichier;
Type
   personne=record
   nom, prenom, profession: string[25 ] ;
   Telephone: string [15];
End;
Var
  Contact: personne;
  Carnet: file of personne;
  Reponse: char;
Begin
  Assign (carnet,'c:\bp\client.dat');
  Rewrite(carnet);
  With contact do
   Begin
   Repeat
   Write ('nom :');
   Readln (nom);
   Write('prenom :');
   Readln (prenom);
   Write ('profession :');
   Readln (Profession);
   Write ('Telephone :');
   Readln (Telephone);
   Write(carnet, contact);
   Write('encore ? O/N');
   Readln(reponse);
```

```
   Until (reponse='n')or(reponse='N');
   End;
Close(carnet);
End.
```



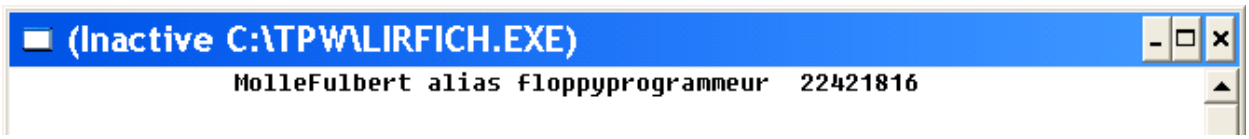
*Les commandes **DIR** et **TYPE** sous DOS SHELL permettenet de vérifier l'existence du fichier sur le disque et son contenu*

`Assign (carnet,'c:\bp\client.dat');`
Demande au compilateur d'établir la correspondance entre `carnet` et `client. Dat` qui se trouve au chemin spécifié.
`Rewrite(carnet);`
Ouvre le fichier en écriture en y effaçant toutes les informations qui y existeraient.
`Write(carnet, contact);`
Ecrit dans le fichier logique carnet donc dans client.dat le contenu de l'enregistrement contact.
`Close(carnet);`
Ferme le et le rend innaccessible

3.5.2 – Affichage du contenu d'un fichier

```
program liste_fichier ;
Type
personne=record
  nom, prenom, profession: string [25 ] ;
Telephone: string [15];
End;
Var
Contact: personne;
Carnet: file of personne;
Reponse: char;
Begin
Assign (carnet, 'c:\bp\client.dat');
Reset (carnet);
While not e of (carnet) do
Begin
Read (carnet, contact);
```

```
With contact do
Begin
Write(nom:20);
Write (prenom:20);
Write (profession: 10);
Write (Telephone: 10);
End;
End ;
Close(carnet);
End.
```



`Reset(carnet);`
Ouvre le fichier carnet en lecture

`eof(carnet)`
désigne la fin du fichier

`Read(carnet,contact);`
Lit l'enregistrement dans le fichier et le place dans la variable contact.

3.5.3 -Ajouter une information dans un fichier

```
program ajout ;
Type
  personne=record
  nom, prenom, profession: string [25 ] ;
Telephone: string [15];
End;
Var
Contact: personne;
Carnet: file of person;
Begin
Assign (carnet, 'c:\bp\client.dat');
Reset (carnet);
Seek (carnet, filesize(carnet));
Whith contact do
begin
Write ('nom');
Readln (nom);
Write('prenom');
Readln (prenom);
Write ('profession');
Readln (Profession);
Write ('Telephone');
Readln (Telephone);
Write(carnet, contact);
```

```
End;
Close(carnet);
End.
```

Le contenu du fichier est modifié

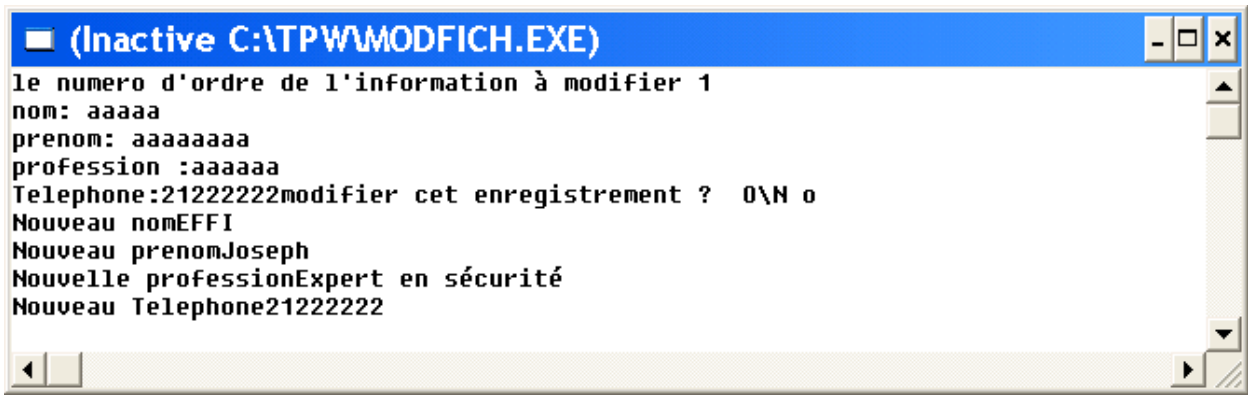
Seek (carnet, file size (carnet));
Permet de se positionner en fin de fichier. En effet la fonction '`filesize(carnet)`' donne la taille du fichier c'est à dire le nombre d'enregistrement.
La numérotation des composantes d'un fichier commence par '0' et non par '1'.

3.5.4 Modification d'un enregistrement

Pour modifier un enregistrement dans un fichier, il faut se positionner à l'emplacement de celui ci avec l'instruction 'seek' et écraser les informations existantes par de nouvelles valeurs saisies.

```
Program modif_file;
  uses crt;
Type
   personne=record
   nom, prenom, profession: string [25 ] ;
  Telephone: string [15];
  end;
Var
  Contact: personne;
  Carnet: file of personne;
  Num: integer;
  Rep: char;
Begin
  Assign (carnet, 'c:\bp\client.dat');
  Reset (carnet);
  Write ('le numero d''ordre de l''information à modifier ') ;
  Readln (num) ;
  Seek (carnet,num);
  read (carnet, contact);
  With contact do
    begin
```

```
    Writeln ('nom: ',nom);
    Writeln('prenom: ',prenom);
    Writeln('profession :',profession);
    Write ('Telephone:', telephone);
    Write ('modifier cet enregistrement ?  O\N ') ;
    Readln (rep);
    if (rep='n') or (rep='N') then writeln('ANNULEE!!!!!')
    else
       begin
       Write ('Nouveau nom');
       Readln (nom);
       Write('Nouveau prenom');
       Readln (prenom);
       Write ('Nouvelle profession');
       Readln (Profession);
       Write ('Nouveau Telephone');
       Readln (Telephone);
       write(carnet, contact);
      end;
   end;
close (carnet);
end.
```



La méthode de modification utilisée exige de connaître la position de l'enregistrement dans le fichier ce qui n' est pas toujours possible quand on a beaucoup d'enregistrement à gérer. La meilleure solution consiste à utiliser une clé informatique.

1ère méthode
incorporer la clé à l'enregistrement
Type
personne=record
  cle=integer;
  nom, prenom, profession: string [25] ;
  Telephone: string [15];
End ;

Pour retrouver le client ayant la clé 1085 on utilisera la recherche séquentielle.

2ème méthode
il faut créer un 2ème fichier qui va contenir un enregistrement liaison qui est composé de *cle et* 'num ou num est la position de l'enregistrement dans le fichier.

Type
Liaison= record
clé=integer;
Num: integer;
end;

3.5.5-Suppression d'un enregistrement

Il est impossible de supprimer un enregistrement . On peut remplacer les valeurs saisies par XXXX...., quand il s'agit de chaînes de caractères ou 000000... , il s'agit de type numérique.
Cf. programme de modification

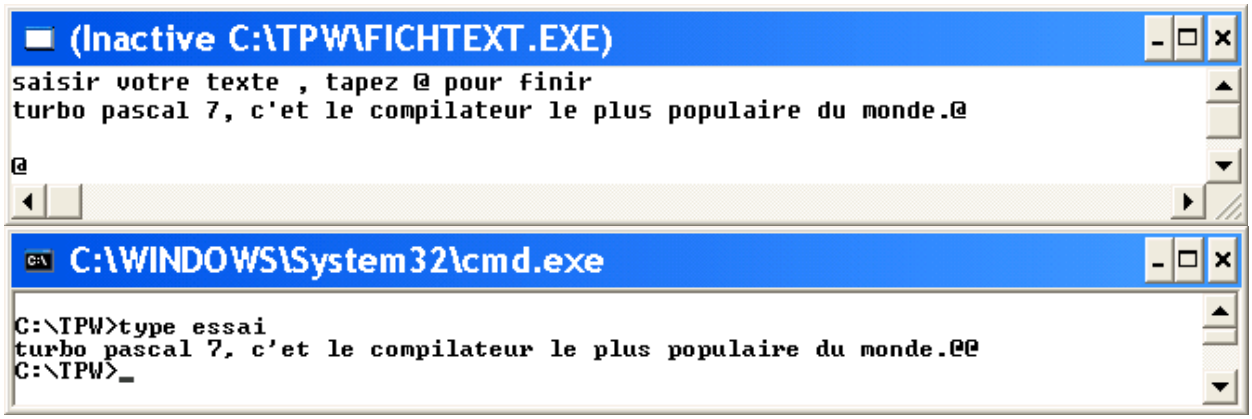
Remarque
Après exécution du programme de suppression , la ligne de l'information supprimée reste occupée et les numéros d'ordre des enregistrements ne changent pas .

3.5. 6-Les fichiers texte

Les fichiers texte portent l'extension '.txt' ce sont les fichiers de caractères.
Avec les fichiers textes on introduit un nouveau type qui est le texte.

```
Program fichier_texte ;
Type
Chaine= string [80] ;
Var
Fic :text ;
Ligne: chaine;
Begin
Assign (fic,'essai') ; { essai.txt}
Rewrite(fic) ;
Writeln('saisir votre texte , tapez @ pour finir') ;
Repeat
Readln(ligne) ;
```

```
Write(fic, ligne);
Until ligne=@;
Close(fic);
End.
```



Le contenu du fichier essai

4 PROCEDURES ET FONCTIONS

Un programme écrit dans un seul bloc est difficile à manipuler dès qu'il dépasse une page d'écran (25 lignes). Une écriture modulaire de ce programme permet de le scinder en plusieurs parties et de regrouper dans un programme principal les instructions en décrivant leur enchaînement.
Le module lui même peut être décomposer en sous module. Les avantages de la programmation modulaire sont :

- Eviter les séquences d'instruction répétitives
- Partager des outils communs

4.1 Notion de procédure

Une procédure est la généralisation de la notion de fonction. Une procédure peut imprimer un message ou réaliser un calcul alors que la fonction fourni obligatoirement un résultat.

```
Program procedure_tab;
Déclaration de variable
Définition de procédures et fonctions
Begin

End.
```

La definition d'une procedure se fait au niveau de la partie déclaration des variables

syntaxe de déclaration

```
procedure < nom de la procedure > (liste de paramètres) ;
type
_ _ _ _ _ _ _ _ _
var
_ _ _ _ _ _ _ _ _
begin
end.
```

Exemple
Procedure PROC1 (z: integer);
Procedure P_Somme(x,y: integer; var z: integer);

4.1.1 Premier exemple

```
Program Procedure_tab ;
Uses crt ;
Const
n = 4;
Type
    Matrice=array[1..n,1..n] of integer;
Var
A:matrice;
procedure lectmat(var t:Matrice);
         Var
            i, j: integer;
            begin
                writeln('Donner les composantes de la matrice');
                 for i :=1 to n do
                        for j := 1 to n do
                          begin
                write('t[', i, j,' ] =');
                readln (t[i,j]);
                                end;
    writeln('Lecture terminée');
end;

procedure affichmat (t: matrice);
         var
            i,j: integer;
         Begin
              Gotoxy(10,10);
Writeln('Affichage');
For I:= 1 to n do
Begin
For j:= 1 to n do
Begin
Write('  ', t[i,j], '  '   );
End;
Writeln;
End;
Writeln('Fin de L''affichage');
End ;

Begin
Writeln('PROGRAMME DE SAISIE DE MATRICE');
Lectmat(A) ;
Clrscr ;affichmat(A) ;
End.
```

```
        Affichage
  1    1    1    1
  2    2    2    2
  3    3    3    3
  4    4    4    4
Fin de L'affichage
```

5.2 Notion de variables globales

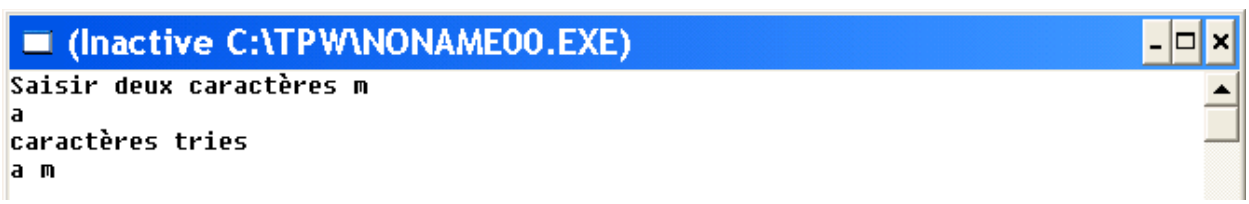
La variable globale est défini dans la partie déclaration ressources données. Elle est manipulée directement par la procédure sans que cette variable ne soit passée en paramètre.
Les paramètres sont définis au même niveau mais sont manipulés par l'intermédiaire d'un appel dans le programmme principal.
Exemple :
Lectmat(A) ;
Lectmat est la procédure et A est le paramètre.

5.3 Variables Locales

Il est tout à fait possible d'effectuer au sain d'une procédure des déclarations de variable de type et de procédure.
Les entités ainsi crées ne sont connu qu'à l'intérieur de la procédure ; on dit qu'elles sont locales à la procédure, ou encore que leur portée est limitée à la procédure.

```
Program exemple3;
uses crt;
 Var
 C1,C2 :char ;
 Procedure tricar ;
  Var
   C:char;
  Begin
   If C1>C2 then
   Begin
   C:= C1;
   C1:= C2;
   C2:= C;
   End;
  End;

Begin
Write('Saisir deux caractères ');
Readln(C1);
readln(C2);
Tricar;
Writeln('caractères tries');
Write(C1,' ',C2);
End.
```

Exemple



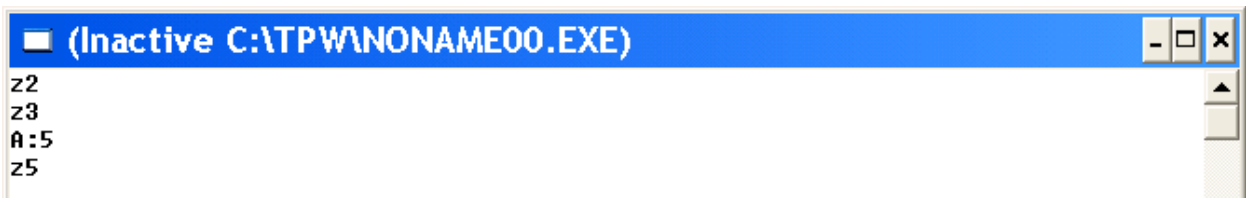
Commentaire

C1 et C2 sont des variables globales, elles sont visibles dans le programme principal, mais aussi dans la procédure `tricar`.
L'instruction « `var c : char ;` » permet de créer une variable locale à la procédure tricar.
L'instruction «write(c) ; » dans le programme principal donnerait un message d'erreur dans la compilation parce que la variable **C** n'est pas visible dans le programme principal.

5.4 Les arguments transmis par valeur

exemple : programme mettant en œuvre la notion d'argument

```
program PROG;
 var
   a : integer ;
   procedure PROC(z: integer);
     var
     y: integer;
     begin
     y:=2*(z-1);
     z:= (z*z)div y;
     writeln('z', z);
     end;
begin
PROC(3);
A:=5;
PROC(A);
Writeln('A:', A);
PROC(A+5);
End.
```



Notons la nouvelle manière de définir la procédure (son entête)
La variable z est un argument qui peut être manipulé au niveau local. Dans la définition de la procédure « z » n'a pas de valeur.
Au cours de l'appel de la procédure propre on peut lui transmettre une valeur

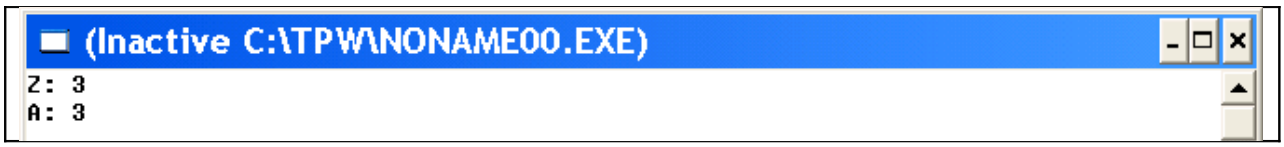
A est un paramètre effectif.
Dans un appel par valeur le paramètre d'appel peut être considéré comme une variable locale. Ainsi, à la fin de PROC(A), la variable conserve sa valeur qui est « 5 ».

5.5 Les arguments transmis par adresse (par variable)

La transmission par valeur permet de fournir les informations à la procédure mais en aucun cas elle ne permet de recueillir un résultat. La transmission par adresse permettra de résoudre ce problème en n'effectuant pas de recopie de la valeur de la variable. Elle travaille directement sur la valeur d'origine elle même et pourra la modifier.

Exemple : Pour effectuer le passage par adresse

```
Program PROG ;
Var
A : integer ;
 Procedure PROC(var Z: integer);
 Var
 Y:integer;
 Begin
  Y:=2*(Z-1);
  Z:=(Z*Z) div Y;
  Writeln('Z: ',z);
End;
Begin
A:=5;
PROC(A);
Writeln('A: ',A);
End.
```

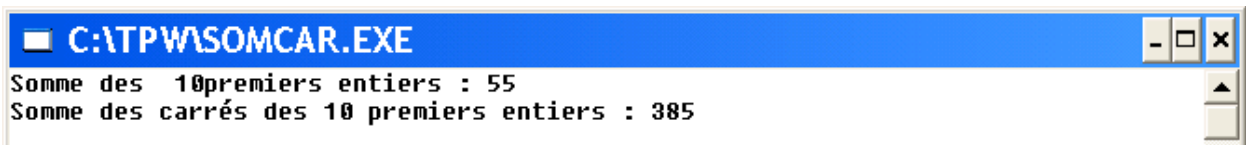


Le fait d'avoir écrit `var Z:integer ;` dans la déclaration de la procédure indique non pas de réserver une nouvelle place en mémoire mais que « Z » pointera sur la même case mémoire que le paramètre réel.
Y est toujours une variable locale.

Exemple

```
Program exemple3 ;
Const
N= n_max=10 ;
Type
Ligne= array[1.. n_max] of integer ;
Var
A,b:integer;
I: integer;
Sa,Sb: integer;
procedure somme(t: ligne  var  som: integer);
var
i: integer;
begin
```

```
som:= 0;
for i:=1 to n_max do
som:= som+ t [i];
end;
begin
for I:= 1 to n_max do
begin
a[I]:= I;
b[I]:= sqr(i);
Somme(a,Sa);
 Somme(b,Sb) ;
Writeln('Somme des  ',n_max,'premiers entiers : ', Sa) ;
Writeln('Somme des  ',n_max,' carrés des premiers entiers : ', Sb) ;
Readln ;
End.
```



Commentaire
Le paramètre somme est passé par adresse et permet de récupérer le résultat à l'appel de la procédure. La meilleure solution est d'utiliser une fonction.

5.6 Les fonctions

La fonction est un cas particulier de procédure. Sa particularité c'est qu'elle retourne un résultat de type élémentaire : integer, real, boolean, char.

```
Function Somme(t:ligne):integer;
Var
I:integer;
Som:integer;
Begin
Som:=0;
For i:=1 to n_max do
  Som:= som+t[i];
Somme:=som;
End;
```

Le mot clé "`function`" permet de déclarer la fonction.
« : `integer` » situé sur la même ligne que « `function` » précise le type élémentaire retourné
« `somme := som` » permet de retourner le contenu de la variable « som ».
On a toujours dans une fonction une instruction de type
`nom_de_fonction:=expression;`

L'appel de la fonction doit se faire dans une expression ou une affectation du type
`Total := somme(a);`
Où total est de type integer
a' : est de type ligne
NB : les règles de passage de paramètre des procédures sont aussi valables pour les fonctions

5.7 La récursivité

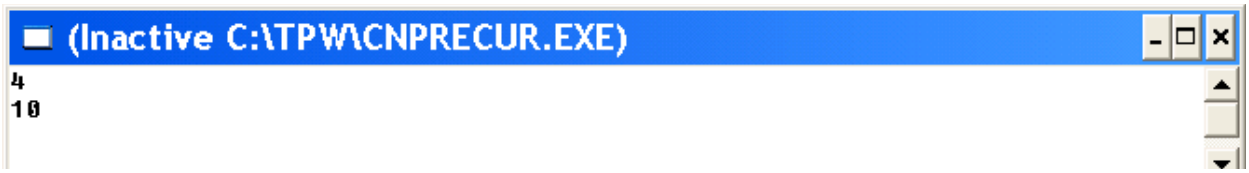
Une procédure ou une fonction se reconnaît elle même. Elle peut donc s'appeler : on dit dans ce cas qu'elle est récursive.

Exemple

Pour calculer Cnp(n,p), il faut calculer Cnp(n-1,p) et Cnp(n-1,p-1) et enfin la somme. Ce calcul n'est pas possible si

$$\begin{cases} n-1 < 0 \\ p-1 < 0 \\ n-1 < p \end{cases}$$

```
Program Combinaison;
uses crt;
var
  x,y:integer;
  Function Cnp(n,p:integer):integer;
  Begin
    If((n=1)or(p=0)or(n<=p)) then Cnp:=1
    Else
    Cnp:=Cnp(n-1, p) +Cnp(n-1,p-1);
  end;
begin
writeln(Cnp(4,3));
x:=5;
y:=2;
writeln(Cnp(x,y));
end.
```



Les inconvénients de la récursivité :

- Les fonctions récursives font beaucoup d'appel de la fonction donc nécessite beaucoup de mémoire ;
- Elles sont à manipuler avec beaucoup de précautions car elles engendrent très souvent des plantages.

Chapitre 5

5 LES UNITES

L'unité en pascal est un concept permettant la modularité des programmes. Le type de fichier obtenu avec une unité n'est pas exécutable mais contient du code exécutable.
Une unité sert en quelque sorte à repartir les éléments de programmation entre divers fichiers.
L'intérêt est :

- Diminuer la taille des fichiers sources .Plusieurs lignes de programmes occupent beaucoup plus d'espace qu'un programme principale.
- Partager les élements réutilisables entre plusieurs programmes .On peut creér "des boites à outils" contenant des programmes qui peuvent être réutulisés dans un nouveau programme.
- Mieux gérer la mémoire:
- créer des unités permet d'avoir un fichier excécutable de taille réduite.Dans ce cas tout le programme n'est pas chargé en memoire.

5.1 Définition de L'unité

Une unité est constituée de 3 parties principales:

- Une partie publique du nom de **INTERFACE**. Celle-ci définit tout ce qui sera accessible une fois l'unité insérer dans le programme. On peut y mettre: des variables globales, des types, des procédures et des fonctions,des constantes. Ces éléments pourront être utilisés dans les programmes faisant appel à l'unité.

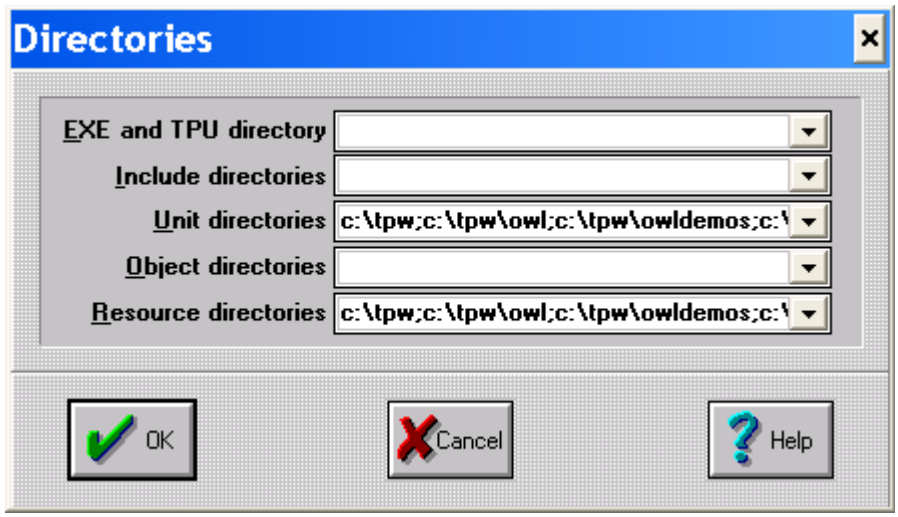
- une partie privée nommée **IMPLEMENTATION**: c'est la partie immergée de l'unité. Elle contient le code des procédures et des fonctions déclarées dans la partie interface mais aussi d'autres procédures.

- Une partie **initialisation**: elle correspond au programme principale d'un programme normal et peut servir à initialiser des variables.

```
Unit <nom_unite> ;
        inclusion nécessaire à l'interface et implémentation
interface
        déclaration de tous les élements et de toutes les procédures et fonctions(en-tête)
implementation
        définition des fonctions et procédures de l'interface
        définition des fonctions et procédures non visibles
begin
      initialisation
end.
```

5.2 Utilisation d'une unité

L'unité écrite avec l'éditeur de Turbo pascal est enregistrée avec l'extension: "
.PAS". Pour être utilisé dans un programme pascal une unité doit être compilée et construite. Un fichier provenant de la compilation de l'unité comportant l'extension ".TPU" existe et est accessible au compilateur dans ce cas.
Le répertoire de recherche est spécifié par: OPTIONS ensuite DIRECTORIES.



Boite de dialogue de turbo pascal for Windows

L'unité doit être déclarée dans le programme voulant l'utiliser au moyen de l'intruction "uses". Ainsi un programme voulant utiliser une unité nommée "**tableau.tpu**", commençera par les lignes:

```
Program   test unité;
Uses tableau;
Begin
…
End.
```

5.3 Exemple d'utilisation d'unité

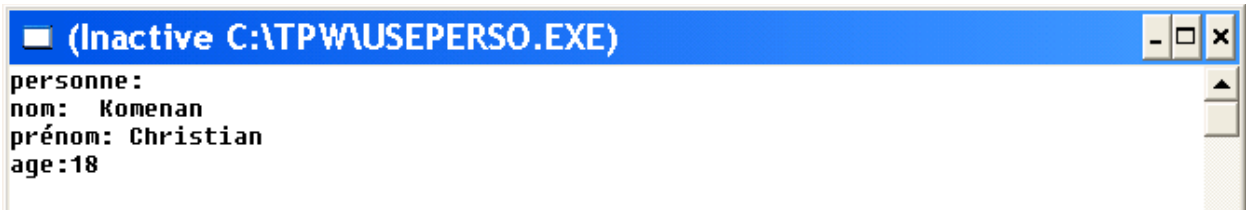
5.3.1 Définition de l'unité

```
Unit personne;
Interface
Type
Tpersonne=record
         Nom,prenom:string;
         Age:integer;
         End;
Procedure affecter ( var V:Tpersonne;n,p:string;a:integer);
Procedure afficher(V:Tpersonne);
Implementation
Procedure affecter  (var V:Tpersonne;n,p:string;a:integer);
```

```
  Begin
  V.nom:=n;
  V.prenom:=p;
  V.age:=a;
End;
Procedure afficher(V:Tpersonne);
  Begin
  Writeln('personne:');
  Writeln('nom: ' , V.nom);
  Writeln('prénom: ' , V.prenom);
  Writeln('age:' , V.age);
End;
End.
```

5.3.2 Utilisation de l'unité

```
Program    testunite_personne;
Uses crt,personne;
Var
Kob:Tpersonne;
Begin
Affecter(Kob,' Komenan','Christian',18);
Afficher(Kob);
End.
```



TAF

Réaliser une unité nommée complexe et son programme de test associer définissant le type nombre complexe permettant de créer,afficher un nombre complexe sous la forme a+ib ,additionner 2 nombre complexe,extraire l'argument,etc.

Unit complexe;
Uses crt;
Interface
Type Tcomplexe=record
Réel,imaginaire:real;
End;
Procédure affiche(Var comp:Tcomplexe;i,r:real);
Procédure addition complexe(var compose:Tcomplexe;z1,z2:Tcomplexe);
Function norme complexe(comp:Tcomplexe):real;
…

Chapitre 6

TP7 dispose de quelques unitées standards dont la connaissance est fort utile :
L'unité **crt**
L'unité **dos**
L'unité **graph**

Consulter la bibliographie, l'aide ou le site *tdprog.free.fr*

6 POINTEURS ET STRUCTURES DE DONNEES COMPLEXES

6.1 Définition

Un pointeur correspond a un couple (adresse mémoire, type).C'est une variable qui contient l'adresse mémoire d'un objet de type donnée. Il permet de manipuler des adresses en mémoire par l'intermédiaire d'objet du type pointeur.

Exemple
Si une variable p contient l'adresse d'une variable q on dit que p pointe sur q.

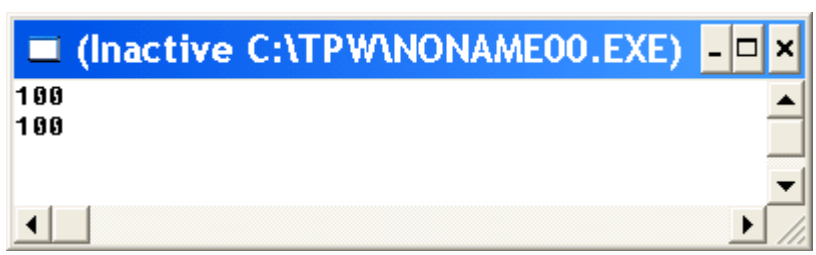
Syntaxe
Soit un type <base>, pour déclarer un pointeur sur un objet de type "base", on procède comme suit:

```
Type
  pbase=^base {base est integer, real, boolean, array...}
var
  pointeur_sur_base:pbase;
```

Exemple

```
 Type
 point_int=^integer;
 var
 pointeur_sur_entier:point_int;
```

```
program exemple;
uses crt;
type
pentier=^integer;
var
 q:integer;
 p:pentier;
```



```
begin
 p:=nil;
 q:=100;
 new(p);
 p^:=q;
 writeln(p^);
 writeln(q);
 dispose(p);
end.
```

Commentaire:

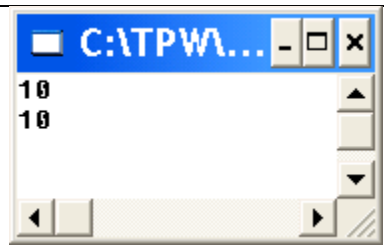
1. Un pointeur peut avoir deux états:
   - **nil** : il ne pointe sur rien,
   - il pointe sur une variable du type précisé lors de la déclaration.

2. Il faut distinguer le pointeur et la variable associée:
   - p est un pointeur sur un entier,
   - p^ est du type entier

3. `new(p)` permet d'associer un pointeur et de créer la variable p^;
La variable p^ est stockee dans une zone particulière de la mémoire géré par le système d'exploitation.

4. pour détruire la variable p^ on utilise la procédure `dispose(p);` celle-ci détruit la variable p^ et affecte la valeur `nil` à p.

```
Program exemple2 ;
uses crt;
type
  pentier=^integer;
var
  p1,p2:pentier;
begin
  p1:=nil;
  p2:=nil;
  new(p1);
  new(p2);
  p1^:=10;
  p2^:=20;
  p2:=p1;
  writeln(p2^);
  writeln(p1^);
  dispose(p1);
  dispose(p2);
end.
```



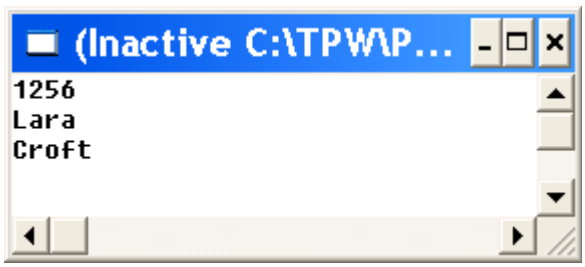
Ce programme affiche 10. Deux pointeurs peuvent pointer sur la même variable. Après l'affectation p2:=p1; les variables p1^ et p2^ sont des alias.

6.2 UTILISATION DES POINTEURS EN ALLOCATION DYNAMIQUE DE LA MEMOIRE

6.2.1 Manipulation des types simples

Un pointeur peut pointer vers n'importe quel type de variable sauf de type fichier (`File Of`, `Text`). Il est possible de combiner les enregistrements, les tableaux et les pointeurs. Cela donne un vaste panel de combinaisons. Essayons-le tableau d'enregistrement.

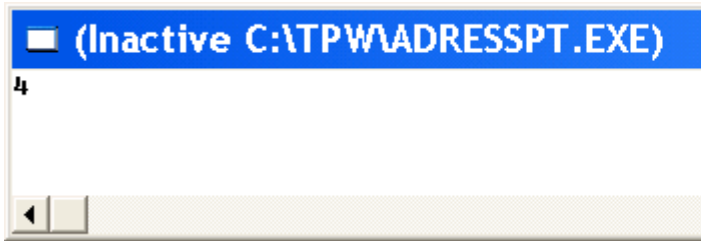
```
Program pointrec ;
  uses crt;
Const
  Max = 10 ;
Type
  Personne = Record
  nom, prenom : String ;
  matricule : Integer ;
  End ;

  Tableau = Array[1..Max] Of Personne ;
  PTableau = ^Tableau ;

Var
 Tab : PTableau ;
 i : Integer ;
Begin
  New(Tab);
  i:=1;
  With Tab^[1] Do
   Begin
   nom := 'Lara' ;
   prenom := 'Croft' ;
   matricule := 1256 ;
  End ;
  WriteLn(Tab^[1].matricule) ;
  WriteLn(Tab^[1].nom) ;
  WriteLn(Tab^[1].prenom) ;
  Dispose(Tab);
End.
```



6.2.2 Utilisation du pointeur dans le passage par adresse

Cette technique est utilisée par le *langage C* qui ne vous offre pas la facilite d'une syntaxe pour ce mode de passage de paramètres comme en pascal.

```
program pointer1;
  uses crt;
type
  pent=^integer;
var
   b:pent;
   procedure ajoute_un( a:pent);
   begin
    a^:=a^+1;
   end;
begin
  new(b);
  b^:=3;
  ajoute_un(b);
  writeln(b^);
  readln;
end.
```



6.3 Application: Les structures de données complexes.

6.2.1 Les liste chainées

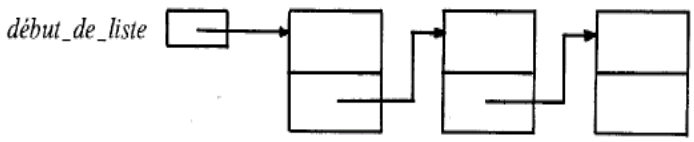


Figure 13.1 Structure de liste contenant trois variables dynamiques

Création et affichage d'une liste chaînée

```
program liste_chainee;
uses crt;
type
pcellule=^cellule;
cellule=record
```

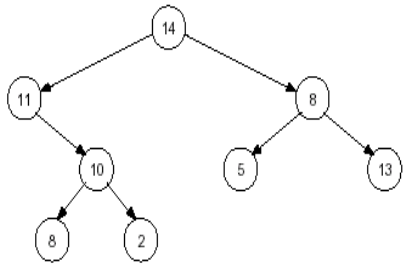
```
    val:integer;
    suivant:pcellule;
        end;
var
  element:pcellule;
  procedure initliste(var l:pcellule);
  var
  i:integer;
  l1:pcellule;
  begin
   l:=nil;
  for i:=1 to 10 do
   begin
    new(l1);
    write('le ',i,'e de la liste :');
    readln(l1^.val);
    l1^.suivant:=l;
    l:=l1;
   end;
  end;

  procedure afficheliste(l:pcellule);
  var
  l1:pcellule;
  begin
    writeln('--- Affichage ---');
    l1:=l;
  while l1<>nil do
   begin
    writeln(l1^.val);
    l1:=l1^.suivant;
   end;
  end;

begin
   initliste(element);
   afficheliste(element);
   readln;
end.
```

```
C:\TPW\POINT1.EXE
le 1e de la liste :3
le 2e de la liste :5
le 3e de la liste :98
le 4e de la liste :45
le 5e de la liste :20
le 6e de la liste :23
le 7e de la liste :65
le 8e de la liste :99
le 9e de la liste :100
le 10e de la liste :37
--- Affichage ---
37
100
99
65
23
20
45
98
5
3
```

6.2.3-Arbres binaires

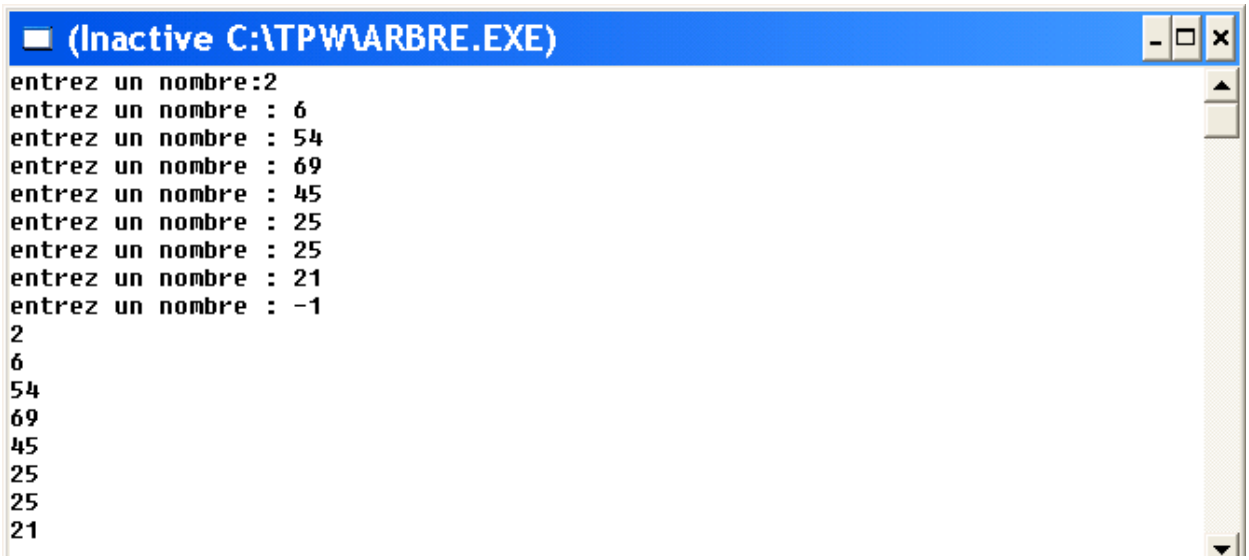
Un programme de création et d'affichage d'un arbre binaire

Un arbre binaire se représente ainsi :




```
program arbresbinaires;
  uses wincrt;
type
    pointeur = ^noeud;
    noeud = record
               valeur : integer;
               gauche : pointeur;
               droit : pointeur;
            end;
var
racine : pointeur;
    n : integer;
procedure construire(var arbre : pointeur; elt : integer);
   begin
      if (arbre=nil) then
         begin
            new(arbre);
```

```
            arbre^.valeur:=elt;
            arbre^.gauche:=nil;
            arbre^.droit:=nil;
          end
       else
            if arbre^.gauche=nil then
               construire(arbre^.gauche, elt)
            else construire(arbre^.droit, elt);
 end;

procedure afficher(arbre : pointeur);
   begin
     if not(arbre=nil) then
       begin
          writeln(arbre^.valeur);
          afficher(arbre^.gauche);
          afficher(arbre^.droit);
       end;
  end;

begin
    racine:=nil;(*Pascal n'initialise pas les pointeurs à nil *)
    write('entrez un nombre:');
    readln(n);
    while (n>=0) do
          begin
              construire(racine, n);
              write('entrez un nombre : ');
              readln(n);
          end;
    afficher(racine);
end.
```



BIBLIOGRAPHIE

Albert Levine - Informatique en terminal, algorithmes et programmation en Turbo Pascal Ellipses

Claude Delannoy Programmer en Turbo Pascal 7.0 Eyrolles – 1996 -352 pages

Alfred Aho, Fohn Hopcroft ,Jeffrey Ullman – Structures de données et Algoritmes - InterEditions -1990 - 423 pages

Philippe Gambrini - Orientation objet, Structures de données et Algoritmes avec ADA 95 - UQUAM -1998 -358 pages

Et de nombreux sites Web

ANNEXES

GUIDE DE DEVELOPPEMENT TURBO PASCAL:

I Etapes de conception d'un programme.

Pour résoudre un problème en informatique il faut écrire un programme.
Les principales étapes de cette conception sont:

- Enoncé formel du problème (pensée globale) qui est généralement vague;
- Enoncé verbal approché qui correspond au cahier de charges (objectifs que devra atteindre le logiciel).
- Recherche des différentes étapes (Analyse descendante).
- Recherche des Algorithmes et méthodes de résolution.
- Programmation qui est la traduction des algorithmes en un langage de programmation. (le **Pascal** dans notre cas)
- Test du Programme.

II Les fautes de programmation

- Fautes de syntaxe: signalées a la compilation ou à l'interprétation des langages.
- Fautes de logique : peuvent conduire à des résultats faux.
- Fautes dans la présentation des données : résultent d'une interactivité médiocre qui conduit à une contradiction entre le programme et les données entrées.

Ces erreurs se traduisent par des messages d'erreurs du type '***Runtime error xxx***' *ou le code xxx désigne le numéro de l'erreur.*

III Règles générales de programmation structurée.

- On ne programme qu'après avoir fait l'analyse;

- Le programme devra être modulaire pour une question de lisibilité, une maintenance aisée.
- Il devra être aussi documenté.

Exemple

```
(**** Programme  de cryptage et décryptages de fichiers ******)
(*                 au format ASCII                            *)
(*Realisé par Mr Yapi Koffi Guillaume                        *)
(*           jk2000ci@yahoo.fr                               *)

program crypto;
var   (*declaration des variables globale*)
.
```

```
.
.
procedure crypter(var f:file of type; var cle:string);
  var   { variables locales à la procédure}
 .
begin
..
..
end;

procedure decrypter(f:file of type ; var cle:string);
var    { variables locales à la procédure}
.
begin
..
..
end;

Begin(* Programme principal.*)
...
...
...
end.
```