Université Gaston Berger
DE SAINT-LOUIS

UFR DE SCIENCES APPLIQUEES
ET TECHNOLOGIE

SECTION INFORMATIQUE

ALGORITHMIQUE & PROGRAMMATION

EN
PREMIERE ANNEE DE DEUG

Moussa LÔ

Fatou KAMARA

Juin 2002

Sommaire

Chapitre 1 : Introduction Générale

1 Notions d'algorithme et de programme

Supposons que l'on veuille automatiser la résolution de l'équation du second degré à solutions réelles

$$ax^2 + bx + c = 0 \qquad (a \neq 0)$$

Rendre automatique la résolution d'un problème revient à confier à cette tâche à un ordinateur.

Sachez, à priori, qu'un ordinateur ne sait rien faire à part ce qu'on lui a appris à faire.

Donc, si nous décidons de lui confier la résolution de cette équation, nous sommes tenus de lui apprendre à le faire.

En d'autres termes, il nous faut lui donner l'ensemble des instructions qu'il doit exécuter pour résoudre ce problème.

C'est cet ensemble d'instructions que nous appellerons **programme**. Et ce programme doit être écrit dans un langage compréhensible par l'ordinateur, par exemple le langage *Pascal*.

Mais, avant l'écriture d'un tel programme, nous devons d'abord élaborer une liste ordonnée des opérations qui, appliquées aux données du problème dans l'ordre prescrit, doivent aboutir à sa résolution.

Cette suite d'opérations s'appelle algorithme et c'est la théorie des algorithmes que l'on nomme **algorithme.**

Définition :

Un *algorithme* est une suite d'actions ordonnées en séquence qui portent sur des objets. Ces objets doivent être définis de manière très précise.

Un *programme* c'est le codage d'un algorithme dans un langage de programmation, et qui peut être traité par une machine donnée.

La construction d'un programme se fait par étape :

problème → énoncé → algorithme → programme.

2 Objets, Données et Programmes

2.1 Notion d'Objet

Le traitement d'un objet concerne la valeur de cet objet.

Si cette valeur ne peut pas être modifiée, nous parlons de constante, sinon nous parlons de variable.

Un objet est parfaitement défini si nous connaissons ses trois caractéristiques, à savoir :

- Son *identificateur* : il est représenté par une suite quelconque de caractères alphanumériques (numériques et alphabétiques sans espace) commençant obligatoirement par une lettre ou un tiret. De préférence, le nom est choisi en rapport avec le contenu de l'objet.

Exemple : DELTA, Premiere_Racine .

- Sa *valeur*: constante ou variable
- Son ***type : n***ous ne pouvons pas appliquer de traitement de traitement à la valeur d'un objet si nous ne connaissons pas son type.

Un **type** est défini par un ensemble de constantes et l'ensemble des opérations que nous pouvons leur appliquer.

Il y a trois grands types d'objet :

- **Booléen**

constantes : vrai, faux
opérations : et logique, ou logique, non logique

- **Numérique :** entier ou réel

constantes : ensemble des entiers relatifs ou ensemble des réels
opérations : toutes les opérations arithmétiques et trigonométriques, notamment l'addition, la soustraction, la multiplication, la division, la puissance, la division entière, la partie entière d'un réel.

- **Caractère :** c'est soit une lettre de l'alphabet latin, soit un code opération ('+', '-', '*', '/', '%', …) ou un code de ponctuation (',', '.', ' :', ' !', …) ou un code spécial ('&', '$', '#', ..).

Un caractère est lié à un code numérique (par exemple, code ASCII) qui le représenté en machine et qui permet d'établir une relation d'ordre entre les caractères.

2.2 Instructions élémentaires

2.2.1 Affectation

L'opération d'affectation consiste à effectuer une valeur à un objet (une variable). Elle s'écrit sous la forme :

<variable> := <valeur>

La valeur affectée peut être :

- une variable de même type
- une constante du type de l'identificateur
- une expression dont l'évaluation produit un résultat du type de l'identificateur.

Exemples :

```
a := 3
b := a
Delta := b*b – 4*a*c
```

2.2.2 Instructions d'entrée et de sortie

Nous disposons d'une instruction de saisie qui permet de récupérer une valeur sur un périphérique d'entrée (le clavier), et d'une instruction d'affichage qui permet l'édition d'une valeur sur un périphérique de sortie (l'écran).

La forme générale de telles instructions est la suivante :

lire (<identificateur>) pour la saisie

ecrire (<valeur>) pour l'affichage

L'identificateur est un objet déclaré. La valeur située derrière l'ordre d'affichage peut être soit : un identificateur, une constante, une expression.

Exemples :

1.

```
lire (b)
ecrire (delta)
ecrire (10)
```

2.

```
ecrire ('donnez votre nom :')
lire (nom)
ecrire ('donnez votre age :')
lire (age)
ecrire (nom, 'vous avez ', age, 'ans')
```

Exercice d'application[1]

Ecrire un algorithme qui lit trois nombres entiers, calcule et affiche leur somme, leur produit et leur moyenne.

```
Programme calcul ;
Variables nbre1, nbre2, nbre3, S, P : entier ;
          M : réel ;
Debut
    ecrire ('donnez trois nombres entiers :') ;
    lire (nbre1, nbre2, nbre3) ;
    S := nbre1 + nbre2 + nbre3 ;
    M := S / 3 ;
    P := nbre1 * nbre2 * nbre3 ;
      ecrire (S, M, P) ;
Fin
```

Le langage utilisé pour écrire cet algorithme s'appelle ***pseudo-code*** (ou langage algorithmique).

Le terme *variable* permet de définir des objets variables ; des objets de nature constante pouvaient être définis avec le terme *constante*.

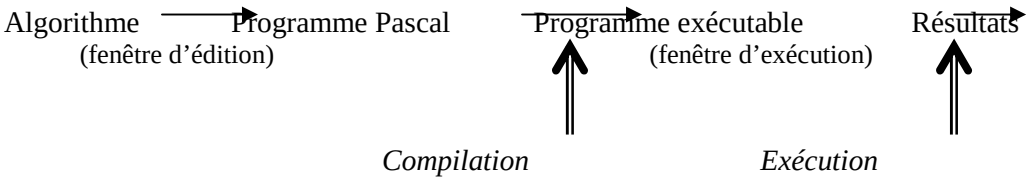
Les termes *debut* et *fin* délimitent les instructions de l'algorithme.

Les variables nbre1, nbre2 et nbre3 sont appelées les *données d'entrée* de l'algorithme alors que S, M et P sont appelées ses *données de sortie*.

3 Démarrer avec Turbo Pascal

Turbo Pascal est un environnement de programmation qui intègre un compilateur du langage Pascal.

Voici les différentes étapes de construction d'exécution d'un programme en Turbo Pascal :



[1] Essayez avant de regarder la solution !

3.1 Structure générale d'un programme Pascal

Voici la structure générale d'un programme Pascal :

Program {nom du programme} **;**
CONST {déclarations des constantes} **;**
TYPE {déclaration des types} **;**
VAR {déclaration des variables} **;**
Begin
{Instructions du programme principal}
End.

Le point-virgule (**;**) permet de séparer deux instructions. Il est à noter que le compilateur du langage pascal ne fait pas la différence entre les minuscules et les majuscules.

3.2 Types de données élémentaires en pascal

- Le type *INTEGER* (nombres entiers) a 5 variantes :

Type	*Domaine*	*Taille*
ShortInt	-128 .. 127	1 octet
Integer	-32 768 .. 32 767	2 octets
Longint	-2147483648 .. 2147483647	4 octets
Byte	0 .. 255	1 octet
Word	0 .. 65 535	2 octets

- Le type *REAL* (nombres réels) a 5 variantes :

Type	*Domaine*	*Taille*
Real	2.9e-39 .. 1.7e38	6 octets
Single	1.5e-45 .. 3.4e38	4 octets
Double	5.0e-324 .. 1.7e308	8 octets
Extended	3.4e-4932.. 1.1e4932	10 octets
Comp	-9.2e18.. 9.2e18	8 octets

Le type *CHAR* (caractères) permet de représenter les caractères ASCII. Les constantes de type caractère s'écrivent entre apostrophe :'A' ,'3' ou '*'.
Le type *BOOLEAN* prend deux valeurs possibles: TRUE et FALSE.
Le type *STRING* permet de représenter les chaînes de caractères ; plus précisément des séquences, ou chaînes de caractères de longueur variable.

3.3 Déclaration des objets (constantes et variables) en Pascal

Les constantes se déclarent comme suit :

Const <identificateur constante> **=** <valeur> **;**

Exemple :

const pi = 3.14 ;

Les variables se déclarent comme suit :

Var <identificateur variable> **:** <type variable> **;**

Exemple :

```
Var     x : integer ;
    Y, z: real;
```

3.4 Les instructions élémentaires en Pascal

L'instruction d'affectation s'écrit sous la forme

<identificateur> **:=** <valeur> **;**

3.5 Les instructions d'entrée / sortie

L'instruction d'affichage s'appelle **write** et s'utilise généralement comme suit :

write (<identificateur>) ;
ou write (<valeur>) ;
ou write ('<message>') ;
ou encore write ('<message>',<identificateur>) ;

Le message est une constante chaîne de caractères non analysée par le compilateur et restituée comme telle à l'écran lors de l'exécution de l'instruction.

L'identificateur est remplacé par la valeur de la variable correspondante au moment de l'exécution de l'instruction.

La valeur peut être une constante ou une expression.

Exemples :

```
write('Bienvenue aux nouveaux bacheliers !') ;
write(10) ;
write(50+25) ;
write('la valeur de delta est',delta) ;
```

Lorsque l'on veut aller à la ligne après un affichage, on utilise **writeln** à la place de write.

L'instruction de saisie s'appelle **read** et s'utilise généralement comme suit :

read (<identificateur>) ;

Elle permet de récupérer une valeur à partir du clavier et de l'affecter à la variable désignée par l'identificateur.

Il est possible de lire plusieurs valeurs en même temps :

read (<identificateur1>, <identificateur2>, …, <identificateurN>) ;

Il est conseillé d'utiliser **readln** à la place de read afin de supprimer le retour chariot (récupéré suite à la validation de l'utilisateur par la touche «Entrée») du tampon.

Exemples :

```
readln(a) ;
readln(b,c) ;
```

3.6 Un premier programme Pascal

On va rédiger le programme correspondant à l'algorithme qui lit trois nombres entiers, calcule et affiche leur somme, leur produit et leur moyenne.

```
Program calcul ; { cette ligne est optionnelle en Turbo Pascal }
    Var nbre1, nbre2, nbre3, S, P : integer ;  M : real ;
    begin
        write ('donnez trois nombres entiers :') ;
        readln (nbre1, nbre2, nbre3) ;
        S := nbre1 + nbre2 + nbre3 ;
        M := S / 3 ;
```

```
        P := nbre1 * nbre2 * nbre3 ;
            writeln (S, M, P) ;
    end.
```

Remarque:
Pascal permet d'inclure des commentaires personnels à un programme ; ceci est très important pour la lisibilité et la compréhension des programmes.
Il suffit alors de mettre le texte concerné entre

{ mettre le commentaire ici **}** ou **(*** mettre le commentaire ici ***)**

4 Exercices proposés

Exercice 1 :

Ecrire les déclarations en Turbo Pascal pour :

a) une variable entière nommée nombre et deux variables réelles nommées rayon et surface .
b) x,y,z : réelles,
   n : entière,
   car, lettre : caractères,
   valable, ok : booléens.

Exercice 2 :

Choisir les identificateurs appropriés et écrire la partie déclaration d'un programme Pascal de facturation qui aurait :

- en donnée d'entrée : un prix hors taxes et une quantité,
- en résultat : un prix TTC

Exercice 3 :

Quels résultats produira ce programme ?

```
Var  val, double : integer ;
Begin
        val := 231 ;
        double := val*2 ;
        Write(val) ;
        Write(double) ;
End.
```

Exercice 4 :

a) écrire un programme qui calcule et écrit le carré de 547.
b) écrire un programme qui calcule et écrit le carré d'un entier x lu au clavier.

Exercice 5 :

Ecrire un programme qui, à partir d'un prix unitaire et d'un nombre d'articles fournis en données, calcule le prix hors taxes et le prix TTC correspondant. Le taux de TVA sera supposé égal à 18,6% et l'affichage devra se présenter ainsi :

```
Prix unitaire       : 45.65
Nombre d'articles       : 17
Prix hors taxe      :776.05
Tva             : 144.35
-----------------------------------
prix TTC            : 920.40
```

Chapitre 2 : Les structures de contrôle

Lors de la rédaction d'un algorithme, il faut être très explicite et indiquer clairement les conditions et l'ordre dans lequels les actions envisagées doivent être effectuées.

Les directives d'un algorithme qui sont utilisées à cette fin sont appelées structure de contrôle. Elles permettent de décrire la structure logique d'un algorithme et facilitent sa traduction en programme.

Lorsqu'elles sont traduites dans un langage de programmation, les structures de contrôle prennent le nom d'instructions de contrôle.

Les structures de contrôle de base sont l'alternative, la répétition, la structure de choix. Elles sont immédiatement disponibles sous forme d'instructions de contrôle dans les langages de programmation comme Pascal.

1 L'alternative

Exemple :

Ecrivons l'algorithme permettant de calculer les valeurs de la fonction f(x) = x pour un réel x donné.

Pour calculer f(x), on doit tester le signe de x et effectuer l'une des deux actions suivantes :
f(x) =x (si x >=0) et f(x) = -x (si x<0)

La structure de contrôle permettant de programmer ceci est l'alternative, dont la forme générale est :

si condition (booléenne) **alors**
  action 1
Sinon
  action 2

Cela nous donne pour notre exemple :

```
si x>=0 alors
    f(x) =x
sinon
    f(x)=-x
```

Remarque : Dans le cas où la 2eme action est vide, on peut omettre le SINON.

En Turbo Pascal, cela se traduit par l'instruction

If condition **Then**
  Instruction(s)
Else
  Instruction(s) ;

Remarques :

- Il ne faut jamais faire précéder un *else* par un *point-virgule*.
- S'il y a plus d'une instruction à exécuter, il faut les encadrer par **begin** et **end** :

```
If condition Then
begin
    Instructions
end
else
```

```
begin
    Instructions
end;
```

Exercice d'application:

Une société paie ses employés chaque semaine. Le salaire étant calculé sur la base d'un taux horaire, écrire un programme qui calcule et affiche le salaire sachant qu'il y a une majoration de 20% si le nombre d'heures est supérieur à 35.

```
Program salaire ;
Var nbre_heures, taux_horaire : integer ;
    salaire_hebdo : real ;
begin
    write ('donnez le nombre d''heures:') ;
    readln (nbre_heures) ;
    write ('donnez le taux horaire') ;
    readln (taux_horaire) ;
        if nbre_heures<= 35 then (* pas de majoration *)
            salaire_hebdo := nbre_heures*taux_horaire
        else      (* majoration de 20%, i.e. 0,2 du salaire hebdomadaire*)
            salaire_hebdo := nbre_heures*taux_horaire* 1.2 ) ;
        writeln ('Le salaire hebdomadaire est de ', salaire_hebdo) ;
end.
```

2 La répétition

Elle permet d'effectuer plusieurs fois consécutives la même opération. Sa forme la plus simple est :

répéter action

Mais un programme simulant une telle opération ne s'arrêterait, en principe jamais. Donc il nous faut pouvoir contrôler une répétition. On dispose pour cela de deux moyens :

- ajouter à cette structure une condition permettant de décider s'il faut poursuivre ou arrêter l'exécution de l'action répétée,
- prescrire le nombre de fois qu'il s'agira d'exécuter l'action.

2.1 La boucle avec compteur

La forme générale est

pour <compteur> **:=** <valeur initiale> **à** <valeur finale> **faire**
    action (à répéter)

Un pas (généralement égal à 1) permet de faire passer automatiquement le compteur d'une valeur initiale à une valeur finale, et à chaque étape (ou itération ou tour de boucle), on exécute l'action spécifiée.

Ce pas, appelé aussi incrément, peut être négatif ; cela sous entend que la valeur initiale est supérieure à celle finale.

Il est à remarquer que cette structure de contrôle s'apparente beaucoup avec la formule mathématique Σ.

Exemple :

Voici un algorithme qui affiche les entiers compris entre 1 et 100.

```
variable i : entier
debut
        pour i :=1 à 100 faire      ecrire(i)
fin
```

En Pascal, la boucle avec compteur se traduit par l'instruction

for <compteur> **:=** <valeur initiale> **to** <valeur finale> **do**
instruction(s)

S'il y a plus d'une instruction à exécuter, il faut les encadrer par **begin** et **end** :

for <compteur> **:=** <valeur initiale> **to** <valeur finale> **do**
begin
instructions
end ;

Lorsque la valeur initiale est supérieure à la valeur finale, le **to** doit être replacé par **downto**.

Exemple :

Voici la traduction de l'algorithme précédent :

```
var i : integer;
begin
        for i :=1 to 100 do
            writeln(i) ;
end.
```

Exercice d'application:

Ecrire un programme qui calcule la somme des n premiers entiers, n donné.

```
var i, n, s : integer;
begin
        readln(n);
        s := 0;
        for i :=1 to n do
            s := s + i;
        writeln(s) ;
end.
```

2.2 La boucle TANT QUE

Elle s'écrit sous la forme

tant que <condition> **faire**
<instruction(s)>

et permet de contrôler la répétition et de garantir son achèvement.

Par exemple, pour afficher dix fois le message 'Bonjour à tous les nouveaux programmeurs !', on peut utiliser la suite d'instructions:

```
Nbfois := 0 ;
Tant que Nbfois < 10 faire
Debut
   Ecrire ('Bonjour à tous les nouveaux programmeurs !')
   Nbfois := Nbfois + 1 ;
Fin
```

Remarques :

1. Dans une boucle **tant que**, la condition contrôlant la répétition est testée avant chaque itération i.e. l'exécution du corps de la boucle.

2. Lorsque le corps de la boucle contient plus d'une instruction (comme dans l'exemple ci-dessus) il faut l'encadrer par les termes *debut* et *fin.*

En Pascal, cela s'écrit :

while <condition> **do**
<instruction>
ou

while <condition> **do**
begin
<instructions>
end

Exercice d'application

Ecrire un programme permettant la saisie de la réponse à la question : « Aimez vous l'algorithmique (o/n) ? ». Le programme doit afficher un message en cas de mauvaise réponse, et renouveler la question jusqu'à ce que la réponse soit correcte.

```
Var rep :char ;
Begin
    Write('Aimez vous l''algorithmique (o/n) ? ') ;
    Readln(rep) ;
    While (rep<>'o') and (rep<>'n') do
    Begin
    Write('Veuillez répondre par oui(o) ou non(n) svp') ;
       Write('Aimez vous l''algorithmique (o/n) ? ') ;
    Readln(rep) ;
    End ;
End.
```

2.3 La boucle REPETER

Elle s'écrit

Repeter
<instruction(s)>
jusqu'à <condition>

Elle est presque identique à la boucle **tant que** à la différence près que la condition de sortie est testée après exécution des instructions.

Dans une boucle **repeter**, on est sûr que le corps de la boucle sera xécuté au moins une fois. Donc, on choisira cette structure de contrôle si l'on veut que le corps de la boucle soit exécuté au moins une fois.

Ici, même si le corps de la boucle contient plus d'une instruction, il n'est pas nécessaire de l'encadrer par *debut* et *fin.*

Par exemple, si l'on devait rédiger l'algorithme de l'exercice précédent en utilisant l'instruction repeter :

```
Variable rep :caractère ;
debut
    repeter
            ecrire('Aimez vous l''algorithmique (o/n) ? ') ;
            lire(rep) ;
    jusqu'à (rep='o') ou (rep='n');
fin.
```

En Pascal, cette structure se traduit par l'instruction :

repeat
<instruction(s)>
until <condition>

Exercice d'application

Ecrire un programme qui permet de saisir une valeur entière positive et de calculer sa racine carrée.

```
Var entier_positif : integer ;
    Racine :real ;
Begin
    repeat
        Write('donner un entier positif :') ;
        Readln(entier_positif) ;
      Until entier_positif > 0;
      Racine := sqrt(entier_positif) ;
      {sqrt est une fonction prédéfinie en Pascal, elle calcule la racine carrée d'un
      entier positif}
       writeln('la racine carrée de ',entier_positif, ' est ', racine) ;
End.
```

Remarque:

On utilise généralement les instructions **while** ou **repeat** lorsque l'on ne connaît pas, à l'avance, le nombre d'itérations.

3 La structure de choix

Considérons le programme qui permet d'afficher littéralement un chiffre saisi au clavier.

On pourrait l'écrire en utilisant l'alternative, on imbriquerait alors des *Si :*

```
Lire (X)
Si X = 1 alors ecrire ('un')
Sinon
Si X = 2 alors ecrire ('deux')
sinon
Si X = 3 alors ecrire ('trois')
sinon
Si X = 4 alors ecrire ('quatre')
sinon
Si X = 5 alors ecrire ('cinq')
sinon
Si X = 6 alors ecrire ('six')
sinon
Si X = 7 alors ecrire ('sept')
sinon
Si X = 8 alors ecrire ('huit')
sinon
Si X = 9 alors ecrire ('neuf')
sinon
Si X= 0 alors ecrire ('zero')
```

La structure de choix dont la forme générale est la suivante est plus adaptée à ce problème.

Suivant <expression> **faire**
Valeur 1 **:** action 1
Valeur 2 **:** action 2
…
Valeur n **:** action n
Sinon : action par defaut
Fin

Son effet est de réaliser l'action correspondant à une *condition i* qui est satisfaite ; si aucune des conditions n'est satisfaite, c'est la condition par défaut qui est réalisée.
condition i signifie ici : *expression = valeur i*
L'expression peut être un identificateur.

En Pascal cette la structure de choix est codée par l'instruction **case :**
case <expression> **of**
Valeur 1 **:** instuction(s) 1;
Valeur 2 **:** instuction(s) 2 ;
…
Valeur n **:** instuction(s) n ;
else instuction(s) par defaut
end

S'il y a plus d'une instruction à exécuter, il faut les encadrer par **begin** et **end.**

Maintenant, reprenons l'exemple précédent en Pascal:

```
Readln(x) ;
case x of
        1 : writeln('un');
        2 : writeln('deux');
        3 : writeln('trois');
        4 : writeln('quatre');
        5 : writeln('cinq');
        6 : writeln('six');
        7 : writeln('sept');
        8 : writeln('huit');
        9 : writeln('neuf');
        0 : writeln('zéro');
        else writeln('ce n''est pas un chiffre !');
end ;
```

4 Exercices proposés

Exercice 1

Un billet d'avion Dakar-Paris-Dakar de la compagnie Air Sanar coûte 375 000F CFA. Cependant, une réduction est accordée sur ce billet pour les voyageurs de moins de 25 ans (20% de réduction) et pour les voyageurs de plus de 65 ans (15% de réduction).
Ecrire un programme Pascal qui calcule le prix du billet pour un voyageur donné.

Exercice 2

a) Que réalise la séquence algorithmique suivante ?

```
A := 2
B := 2
REPETER
   A := A + 2
   B := B + 1
   Afficher A
   Afficher B
JUSQU'A  (A+B >= 10)
```

b) Traduire cet algorithme en Pascal en complétant (déclarations des variables, etc.).

c) Reprendre le programme précédent en utilisant une boucle WHILE.

d) Quelles sont les différences entre ces deux instructions ?
Qu'est-ce qui les différencie de la boucle FOR ?

Exercice 3

a) Ecrire un programme qui effectue la multiplication de deux entiers positifs saisis au clavier, par additions successives. *Par exemple, 6 x 3 = 3 + 3 + 3 + 3 + 3 + 3*

b) Améliorer le programme du a) en choisissant le plus petit élément des deux nombres pour les itérations et en introduisant un contrôle de saisie pour obliger la saisie de nombres strictement positifs.

c)

Exercice 4

Ecrire un programme Pascal qui, à partir de la saisie du prix unitaire d'un produit (PU) et de la quantité commandée (QTCOM), affiche le prix à payer (PAP) en détaillant le port (PORT) et la remise (REM), sachant que:

- le port est gratuit si le prix des produits (TOT) est supérieur à 5 000 F. Dans le cas contraire, le port est de 2% de TOT.
- la remise est de 5% si TOT est compris entre 2 000 et 10 000 F et de 10% au-delà.

Exercice 5

Ecrire un algorithme qui permet de calculer et d'afficher la date du lendemain connaissant la date du jour (par exemple si la date du jour est 31/12/1998, la date du lendemain sera 01/01/1999). Coder cet algorithme en Turbo Pascal.

Exercice 6

Ecrire un programme qui calcule la moyenne de notes lues au clavier. Le nombre de notes n'est pas connu à l'avance et l'utilisateur peut en fournir autant qu'il le désire sans toutefois dépasser 1000 notes. Pour signaler qu'il a terminé, on convient qu'il fournira une note fictive négative. Celle-ci ne devra naturellement pas être prise en compte dans le calcul de la moyenne.

Chapitre 3 : Les tableaux

1 Notion de tableau

Soit un entier n positif.

Supposons qu'on veuille saisir n valeurs réelles afin de calculer leur moyenne, ou de trouver leur minimum, ou de les afficher par ordre croissant.

Pour des valeurs petites de n, on peut déclarer n variables réelles pour résoudre le problème. Mais si n est assez grand, on se rend compte que cela devient impropre, fastidieux, voire impossible.

Il faudrait, dans ce cas, utiliser une variable permettant de représenter les n valeurs. Le type de données de cette variable serait le type tableau.

Définition :

Un tableau est une collection séquentielle d'éléments de même type, où chaque élément peut être identifié par sa position dans la collection. Cette position est appelée indice et doit être de type scalaire.

Déclaration :

Pour déclarer un tableau, il faut donner :

- son nom (identificateur de la variable)
- ses bornes : la borne inférieure correspondant à l'indice minimal et la borne supérieure correspondant à l'indice maximal.
- le type des éléments le composant.

Syntaxe :

type

nom = **tableau[**<indice minimum> **..** <indice maximum>**] de** <type des composants>

ou

Variable

nom : **tableau[**<indice minimum> **..** <indice maximum>**] de** <type des composants>

Exemple :

variable t : tableau[1..10] de réels

en pascal, on a :

type nom = **array[**<indice minimum> **..** <indice maximum>**] of** <type des composants> **;**

ou

Var nom : **array[**<indice minimum> **..** <indice maximum>**] of** <type des composants> **;**

Exemple :

var t : array [1..10] of real ;

Schématiquement, on va représenter la variable t comme suit:

1	2	3	4	5	6	7	8	9	10
8.4	3.5	12	20	10	13.34	50	100	30.1	60.9

Dans la mémoire centrale, les éléments d'un tableau sont stockés de façon linéaire, dans des zones contiguës.
Le tableau ci-dessus est de dimension 1, nous verrons un peu plus loin que l'on peut représenter des tableaux à 2 dimensions, voire même plus.
L'élément n° I sera représenté par l'expression t[I].
Dans notre exemple, t[I] peut être traité comme une variable réelle. On dit que le tableau t est de taille 10.

2 Création d'un tableau

La création d'un tableau consiste au remplissage des cases le composant. Cela peut se faire par saisie, ou par affectation.

Par exemple, pour remplir le tableau t précédent, on peut faire :

t[1] := 8.4 ; t[2] := 3.5 ; … t[10] := 60.9 ;

Si on devait saisir les valeurs, il faudrait écrire :

Pour i := 1 à 10 faire lire(t[i]);

3 Affichage d'un tableau

Afficher un tableau revient à afficher les différents éléments qui le composent. Pour cela, on le parcourt (généralement à l'aide d'une boucle avec compteur) et on affiche les éléments un à un.

Exercice d'application
Ecrire un programme qui permet de créer un tableau d'entiers t1 de taille 20 par saisie, et un tableau t2 de même taille en mettant dans t2[i] le double de t1[i], i ∈ {1, .., 20 }.

```
Type tab = array[1..20] of integer ;
var t1, t2 : tab ;
      i : integer ;

begin
        { saisie de t1 }
        for i :=1 to 20 do
        begin
           write('donner t1[',i,']:'); readln(t1[i]);
        end;
        { création de t2 }
        for i :=1 to 20 do
        t2[i] := 2*t1[i];
        { affichage de t2 }
        for i :=1 to 20 do
        write('t2[',i,']=', t2[i]);
end.
```

4 Traitement d'un tableau

Après avoir créé un tableau, on peut y effectuer plusieurs opérations comme le calcul de la somme ou de la moyenne des éléments, la recherche du plus petit ou du grand élément du tableau, le test d'appartenance d'un objet au tableau, …

Pour la suite, on considère un tableau d'entiers t déclaré comme suit :

Var t : array[1 .. n] of integer ;

4.1 Somme des éléments d'un tableau

On effectue la somme des éléments du tableau t, le résultat est dans la variable S :

```
S := 0 ;
for i :=1 to n do
        S := S + t[i];
writeln('la somme des éléments de t est:', S);
```

4.2 Minimum d'un tableau

On cherche le plus petit élément du tableau t, le résultat est dans la variable min :

```
min := t[1] ;
for i :=2 to 10 do
   if t[i] < min then  min := t[i];
writeln('Le minimum des elements de t est:', min) ;
```

4.3 Test d'appartenance

On cherche si l'entier x appartient à t, le résultat est mis dans la variable booléenne appartient :

```
appartient := false
for i:=1 to n do
        if t[i]=x then     appartient := true;
if appartient then
        write('x appartient à t')
else  write('x n''appartient pas à t');
```

On remarque que l'on peut arrêter les itérations (la recherche) si l'on rencontre l'élément x dans t. Pour cela, il faut utiliser une boucle *Tant Que* ou *Repeat*:

```
i := 1;
while (t[i]<>x) and (i<=n) do
     i:=i+1;
if i>n then
     write('x n''appartient pas à t')
else
     write('x appartient à t');
```

```
i := 0 ;
Repeat
     i:=i+1;
Until (t[i]=x) or (i>n);
if i>n then
     write('x n''appartient pas à t')
else
     write('x appartient à t');
```

Dans les deux cas, si x appartient à t, la valeur de i est l'indice de la case qui le contient.

4.4 Ajout d'un élément

On suppose que le tableau t est « rempli » et qu'il reste une case non occupée à la fin (la $n^{ième}$ case).

Si on veut alors ajouter un entier x à la fin du tableau, il suffira d'écrire

```
t[n] := x ;
```

Mais, si on veut ajouter x dans une case dont l'indice k est différent de n, il faudra décaler les éléments t[k], t[k+1], …, t[n-1] vers la droite pour libérer la case d'indice k :

```
for i := n downto k+1 do
   t[i] := t[i-1];
t[k] := x;
```

4.5 Suppression d'un élément

Pour supprimer l'élément se trouvant à la position k de t, on l'écrase en faisant décaler les éléments placés après lui vers la gauche :

```
for i := k to n do
    t[i] := t[i+1];
```

Il faut noter qu'après une telle opération, l'élément se trouvant à la dernière position (n) n'est plus significatif.

5 Tableaux de caractères

5.1 Notion de chaîne de caractères

Une chaîne de caractères est soit une chaîne vide, soit un caractère suivi d'une chaîne de caractères ; en un mot c'est une collection de caractères.

Exemples : "Bonjour", "L'UGB se situe à Sanar", "A", "2002"

La plupart des langages de programmation, Pascal notamment, disposent d'un type chaîne de caractères et d'un ensemble de fonctions prédéfinies permettant de traiter les chaînes de caractères. Ces fonctions permettent de calculer la longueur d'une chaîne, de concaténer deux chaînes, d'extraire une sous-chaîne, de comparer deux chaînes, etc.

Il faut noter qu'une chaîne de caractères peut être traitée comme un tableau de caractères.

5.2 Le type STRING de Pascal

En Pascal, une variable de type STRING est une séquence de caractères de longueur variable au cours de l'exécution et une taille prédéfinie entre 1 et 255.

Exemple :

```
var s : string ; (* 255 caractères sont alors réservés *)
    s10 : string[10] ; (* seuls 10 caractères sont réservés *)
```

On peut appliquer les opérateurs suivants sur des variables de type STRING :

+, =, <>, <=, >=, <, >.

Les fonctions prédéfinies les plus usuelles sont :

- length(s : STRING) : integer;

retourne la longueur courante de s.

- copy(s : STRING ; p : integer ; l : integer): STRING;

renvoie une chaîne constituée de **l** caractères à partir de la position **p**.

- concat(s1,s2 : STRING): STRING ;

renvoie la chaîne résultant de la concaténation de s1 et de s2.

- pos (ssch : STRING ; ch : STRING): Byte;

renvoie la position du premier caractère de la sous-chaîne **ssch** dans la chaîne **ch** ; si la sous-chaîne ne se trouve pas dans la chaîne, elle renvoie 0.

6 Les tableaux à deux dimensions

Pour traiter les notes obtenues par un étudiant à 10 épreuves on peut utiliser un tableau de 10 réels. Pour traiter les notes obtenues par 5 étudiants, on pourrait utiliser 5 tableaux de 10 réels chacun.

Mais puisqu'on va effectuer très probablement les mêmes traitements sur ces tableaux, il est préférable de les regrouper dans une seule variable qui sera un tableau de 5 lignes et 10 colonnes.

Chaque élément de ce tableau multidimensionnel sera identifié par deux indices : la position indiquant la ligne et la position indiquant la colonne.

Déclaration :

variable t : tableau[1..5, 1..10] de reels ;
En pascal, on aurait
var t : array[1..5, 1..10] of real ;

Pour accéder à l'élément se trouvant sur la ligne i et la colonne j, on utilise le terme **t[i,j]**.

Exercice d'application

Ecrire un programme qui permet de créer (par saisie) et d'afficher un tableau t à deux dimensions d'entiers de taille 3x5.

```
Type TAB_3_5 = array[1..3, 1..5] of integer ;
var t : TAB_3_5 ;
    i,j : integer ;
begin
    (* saisie de t *)
    for i :=1 to 3 do
        for j :=1 to 5 do
        begin
            write('donner t[',i,',',j,']:'); readln(t[i]);
        end;
    (* affichage de t *)
    for i :=1 to 3 do
    begin
        for j :=1 to 5 do
            write(t[i,j],'  ');
        writeln;
    end;
end.
```

Remarque : Les ensembles en Pascal

Le langage Pascal permet la définition de types « ensembles » à l'aide du mot réservé **set**.

L'expression **set of T** définit un type ensemble dont les éléments sont de type **T**. **T** est appelé le type de base de l'ensemble et doit être un type scalaire comportant moins de 256 valeurs ; les bornes doivent s'inscrire entre 0 et 255.

Exemple de définition de types ensembles:

```
TYPE
    EnsCar = set of char ;
    Chiffre = set of 0 .. 9; (* 0 .. 9 est appelé sous-intervalle *)
    JOUR = (dim, lun, mar, mer, jeu, ven, sam) ; (* JOUR est un type énuméré *)
    Jours = set of JOUR ;
```

7 Exercices proposés

Exercice 1 :

On considère un tableau t de n entiers. Ecrire les programmes permettant:
a) de compter le nombre d'éléments nuls de t
b) de chercher la position et la valeur du premier élément non nul de t
c) de remplacer les éléments négatifs par leur valeur absolue et les éléments positifs par leur carré

Exercice 2 :

Ecrire un programme qui permet de saisir des nombres entiers dans un tableau à deux dimensions TAB (10, 20) et de calculer les totaux par ligne et par colonne dans des tableaux TOTLIG(10) et TOTCOL(20).

Exercice 3 :

Un palindrome est un mot ou une phrase qui se peut se lire de la même façon de gauche à droite et de droite à gauche. Par exemple: "ANNA" et "ESOPE RESTE ICI ET SE REPOSE".
Proposer un programme qui permet de dire si un mot ou une phrase est un palindrome.

Exercice 4 :

On considère un tableau t (n,m) à deux dimensions.
a) Ecrire un programme qui calcule la somme et le produit des éléments de t.
b) Ecrire un programme qui affiche le plus petit élément de t et sa position.

Exercice 5 :

Ecrire un programme qui met à zéro les éléments d'un tableau à deux dimensions tels que l'indice de ligne est égal à l'indice de colonne.

Exercice 6 :

Ecrire un programme qui permet de chercher une valeur x dans un tableau à deux dimensions t (m,n). Le programme doit aussi afficher les indices ligne et colonne si x a été trouvé.

Chapitre 4 : Les sous programmes

1 Notions de procédure et fonction

1.1 Exemple 1

Supposons qu'on ait à écrire le programme permettant de calculer la fonction
$f(x) = 2x^5 + 4x^3 + 1$ pour un réel x donné.

S'il n'existe pas d'instruction permettant de coder la calcul de la puissance d'un nombre, l'algorithme va s'écrire comme suit :

```
Programme fonction ;
Variables x, f1, f2, f : réels
            i : entier ;
début
     lire (x)
     f1 := 1
     pour i := 1 à 5 faire
            f1 := f1 * x
     f2 := 1
     pour i := 1 à 3 faire
            f2 := f2 * x
     f := 2*f1 + 4*f2  + 1
     ecrire ('f(',x,') = ', f)
fin
```

On remarque que l'on utilise deux fois la même chose pour calculer les puissances de x (x^5 et x^3) et on pourrait paramétrer cela. Il suffira d'écrire un programme qui calcule la puissance d'un nombre et l'utiliser dans le programme précédent.

```
Programme puissance ;
Variables p, a : réels
            i, b : entier ;
début
     lire (a, b)
     p := 1
     pour i := 1 à b faire
            p := p * a
     ecrire (a, ' puissance ',b ' = ', p)
fin
```

Pour calculer la valeur de **f1**, il suffit d'utiliser ce programme en donnant à **a** et **b** les valeurs de **x** et **5** à partir du premier programme (fonction). Cela signifie que les valeurs de **a** et **b** sont fournies non plus par saisie mais par le programme «utilisateur » c'est-à-dire fonction. Il nous faut donc réécrire le programme puissance sous la forme:

```
Programme puissance (a :réel ; b : entier)
Variables p : réel
          i : entier ;
début
     p := 1
     pour i := 1 à b faire
          p := p * a
     ecrire (a, ' puissance ',b ' = ', p)
fin
```

a et **b** sont les arguments du programme puissance.

Mais la valeur de **p** doit être récupérée dans le programme de calcul de la fonction dans la variable **f1** et ce n'est pas en l'affichant dans le programme puissance que cela va se faire. Il faut que le programme puissance retourne (ou renvoie) la valeur de **p** au programme «utilisateur». Ainsi, à la place de l'instruction d'affichage ecrire (a, ' puissance ',b ' = ', p) on écrira **retourner (p)** ou **puissance := p.**

Ainsi écrit, le programme puissance pourra être utilisé par le premier programme comme suit :

```
Programme fonction ;
Variables x, f1, f2, f : réels
début
     lire (x)
     f1 := puissance (x,5)
     f2 := puissance (x,3)
     f := 2*f1 + 4*f2  + 1
     ecrire ('f(',x,') = ', f)
fin
```

Quelles sont les particularités du programme puissance écrit et utilisé de cette façon ?

(i) Il est écrit comme un programme (structurellement parlant).

(ii) Il peut recevoir ses données d'entrée à partir d'un autre programme.

(iii) Il peut renvoyer ses données de sortie à un autre programme.

(iv) Son exécution est commandée par un autre programme.

Le programme puissance est un **sous programme**, plus précisément une **fonction**.

1.2 Définition

Un ***sous programme*** est rédigé de façon telle que son exécution puisse être commandée par un programme.

L'exécution d'un sous programme est donc déclenchée par un programme. Celui-ci est appelé *programme appelant* (ou *programme principal).* Il fournit des données au sous-programme et récupère les résultats de ce dernier.

En général, on distingue deux types de sous programme : les procédures et les fonctions. La différence est qu'une fonction renvie une valeur (c'est l'exemple du sous programme puissance) alors qu'une procédure ne renvoie pas de valeur.

En Pascal, le terme Programme est remplacé par le terme **function** s'il s'agit d'une fonction et **procedure** s'il s'agit d'une procédure. Généralement, les sous-programmes seront écrits à l'intérieur des programmes qui les utilisent, juste après la déclaration des variables.

Le programme Pascal complet de l'exemple est :

```
Program fonction ;
Var x, f1, f2, f : real ;

(* definition du sous programme puissance *)
function puissance (a :real ; b : integer) : real ;
Var p : real;
      i : integer;
begin
      p := 1 ;
      for i := 1 to b do
            p := p * a ;
       puissance := p ;
end ;

(* programme principal *)
begin
      readln(x) ;
      f1 := puissance (x,5) ;
      f2 := puissance (x,3) ;
      f := 2*f1 + 4*f2  + 1;
      writeln('f(',x,') = ', f) ;
end.
```

1.3 Exemple 2

On veut écrire un programme Pascal qui saisit une heure et affiche «Bonjour» si on est avant 19h et «Bonsoir» sinon en utilisant deux procédures *SalutationsJour* et *SalutationsSoir* qui affichent respectivement «Bonjour» et «Bonsoir ».

```
Program salutations ;
var heure : integer ;

procedure SalutationsJour ;
begin
      writeln('Bonjour');
end ;

procedure SalutationsSoir ;
begin
      writeln('Bonsoir');
end ;

begin
      readln(heure);
      if heure < 19 then
            SalutationsJour
      Else
            SalutationsSoir;
end.
```

1.4 Remarque

Quand un programme A utilise un sous programme B, il se passe successivement :

a) Appel du sous programme B par le programme appelant A ;
b) Suspension de l'exécution du programme A ;
c) Exécution du sous programme B ;
d) Retour au programme A.

2 Echange d'informations entre programme et sous-programme

2.1 Les variables globales et les variables locales

On distinguera les variables du programme principal, appelées *variables globales*, de celles des sous-programmes, appelées *variables locales*.

Les variables globales peuvent être utilisées aussi bien par le programme principal que par les sous programmes alors que les variables locales ne peuvent être utilisées que dans les sous programmes qui les ont définies (leur durée de vie est la durée d'exécution du sous programme).

Par exemple, considérons le programme suivant :

```
Program A ;
var VA : integer ;

Procedure B ;
var VB : integer;
begin
     VB := 2*VA;
     writeln (VB) ;
end;

Procedure C ;
var VC : integer;
begin
     readln (VC) ;
     VA := 3*VC;
end;

BEGIN (* programme principal A *)
     C ; (* appel de la procédure C *)
     B ; (* appel de la procédure B *)
END.
```

Dans ce programme, VA est une variable globale ; elle peut être utilisée aussi bien dans le programme principal (A) que dans les procédures (B et C). VB et VC sont des variables locales aux procédures B et C ; elles ne peuvent pas être utilisées en dehors de ces procédures.

2.2 Les paramètres

2.2.1 Définition

Les paramètres permettent à un programme appelant de transmettre à un sous programme des données lors de l'appel de ce dernier.

Un sous programme est rédigé de façon à pouvoir recevoir des données du programme appelant : cela est possible grâce aux paramètres. On les appellera formels dans leur définition

dans le sous programme et effectifs lors de l'appel du sous programme. A cet instant, les paramètres formels sont remplacés par des paramètres effectifs. Ces derniers sont fournis par le programme appelant.

2.2.2 Exemples

Essayons d'écrire un programme qui calcule et affiche la somme de deux entiers en utilisant :

a) une procédure sans paramètres

```
Program somme_a ;
var x, y, s : integer ;

Procedure som ;
begin
      s := x + y;
end;

BEGIN
      readln(x,y);
      som;  (* appel de la procédure som *)
      writeln (s) ;
END.
```

b) une fonction sans paramètres

```
Program somme_b ;
var x, y : integer ;

function som :integer ;
begin
      som := x + y;
end;

BEGIN
      readln(x,y);
      writeln (som) ; (* appel de la fonction som et affichage direct du résultat*)
END.
```

c) une procédure avec paramètres

```
Program somme_c ;
var x, y, s : integer ;

Procedure som (a,b : integer);
begin
      s := a + b;
end;

BEGIN
      readln(x,y);
      som(x, y);   (* appel de la procédure som *)
      writeln (s) ;
END.
```

d) une fonction avec paramètres

```
Program somme_d ;
var x, y : integer ;

function som (a,b:integer) : integer ;
begin
     som := a + b;
end;

BEGIN
     readln(x,y);
     writeln (som (x,y)) ; (* appel de la fonction som et affichage direct du résultat*)
END.
```

2.2.3 Le passage des paramètres

Rappelons que les variables globales existent pendant tout le temps que dure l'exécution du programme principal alors que les variables locales n'existent que durant l'exécution du sous programme qui les a déclarées. Ce qui fait que les valeurs affectées à ces variables sont perdues après l'exécution du sous programme.

Les paramètres des sous programmes se comportent comme des variables locales.

Exemple 1 : Considérons le programme suivant :

```
Program addition ;
var x, y, s : integer ;

function somme (a,b:integer) : integer ;
begin
     somme := a + b;
end;

BEGIN
     readln(x,y);
     s := som (x,y) ;
     writeln ('la somme de ',x, 'et ',y,' vaut :',s) ;
END.
```

Dans ce programme, l'appel de la fonction *somme* avec les paramètres *x* et *y* donne les valeurs de ces deux variables à *a* et *b* qui sont créés à cet instant. Mais, leur existence ne dure que lors de l'exécution de la fonction.

Ici, on n'a pas besoin de garder *a* et *b* après l'exécution de la fonction *somme* mais il peut arriver qu'on ait besoin des valeurs prises par les paramètres d'un sous programme après son exécution.

Exemple 2 : Intéressons nous maintenant au programme suivant qui doit permettre d'échanger les valeurs contenues dans deux variables:

```
Program echange ;
var x, y : integer ;

procedure permuter (a,b:integer) ;
var c : integer;
```

```
begin
      c := a;
      a := b;
      b := c;
end;

BEGIN
      readln(x,y);
      permuter (x,y) ;
      writeln ('apres echange ',x,' vaut :', x , 'et ',y,' vaut :', y) ;
END.
```

Lors de l'appel de la procédure permuter, les valeurs des paramètres *a* et *b* sont permutées mais celles des variables *x* et *y* ne le sont pas.

Pour cela, il faudrait que lors du passage des paramètres effectifs (*x* et *y*), les paramètres formels (*a* et *b*) reçoivent les variables *x* et *y* et non leurs valeurs. On dira alors que les paramètres sont passés **par variable** ou **par adresse**, et non **par valeur** comme précédemment.

Il faut alors réécrire la procédure permuter comme suit :

```
procedure permuter (var a,b:integer) ;
var c : integer;
begin
      c := a;
      a := b;
      b := c;
end;
```

On a ainsi ajouté le terme **var** dans la déclaration des paramètres formels *a* et *b* de la procédure ; cela permet de signifier que le passage des paramètres se fait par variable. Si le terme **var** est omis à ce niveau devant un paramètre, cela veut dire que son passage se fait par valeur.

Exercice d'application : Réécrire le programme permettant de calculer et d'afficher la somme de deux entiers en utilisant un passage par variable.

```
Program somme ;
var x, y, s : integer ;

procedure som (a,b : integer; var  sp: integer);
begin
      sp := a + b;
end;

BEGIN
      readln(x,y);
      som(x, y,s);        (* appel de la procédure som *)
      writeln (s) ;
END.
```

3. Exercices proposés

Exercice 1:
Ecrire un programme qui permet la saisie, l'inversion et l'affichage des éléments d'un tableau d'entiers. Le programme doit comprendre une procédure par opération.

Exercice 2:
a) Ecrire une procédure permettant de déterminer si un nombre entier est premier. Elle comportera deux arguments: le nombre à examiner et un indicateur booléen précisant si ce nombre est premier ou non.
b) Ecrire la procédure précédente sous forme d'une fonction.

Exercice 3:
On considère les déclarations suivantes:

```
Const       D = 1; F = 10;
Type        Valeur = integer;
            TableauDeValeurs = array[D..F] of  Valeur;
```

1) Ecrire une procédure INVERSER qui inverse l'ordre des éléments d'un tableau T de type TableauDeValeurs.
2) Ecrire une fonction Delta qui calcule la différence entre les deux plus petites valeurs d'un tableau T de type TableauDeValeurs.

Exercice 4:
Ecrire la procédure PASCAL *transfo_en_maj* ***(minus, maj)*** qui reçoit une chaîne de caractères minuscules (par le paramètre *minus*), et la transforme (dans le paramètre *maj*) en chaîne de caractères majuscules. On pourra utiliser la fonction prédéfinie upcase (f*unction upcase(c:char):char;)*, qui reçoit en argument un caractère et retourne la majuscule correspondante.

Donner un exemple d'utilisation de la procédure *transfo_en_maj* à l'aide d'un petit programme PASCAL.

Exercice 5:
Ecrire un programme Pascal qui lit une chaîne de caractères et affiche le nombre de lettres de l'alphabet latin et le nombre de chiffres contenus dans la chaîne.

On peut utiliser les deux fonctions suivantes (*on ne vous demande pas de les écrire*!):

Function ***EstUneLettre*** *(c:char): boolean* qui renvoie *true* si le paramètre reçu c est une lettre de l'alphabet latin et *false* sinon.

Function ***EstUnChiffre*** *(c:char): boolean* qui renvoie *true* si le paramètre reçu c est un chiffre et *false* sinon.
Ecrire une procédure qui supprime tous les espaces d'une chaîne de caractères (de longueur max 50).

Exercice 6:
Ecrire une procédure DIVIS (**N**) qui met dans un ensemble tous les diviseurs naturels de l'entier strictement positif **N**.

Chapitre 5 : Les structures d'enregistrements

1 Notion d'enregistrement

1.1 Introduction

La structure de tableau permet de traiter un nombre fini d'informations de même type auxquelles on peut accéder par un indice. Or souvent il est nécessaire d'organiser de façon hiérarchique un ensemble de données de types différents mais qualifiant un même objet comme par exemple :

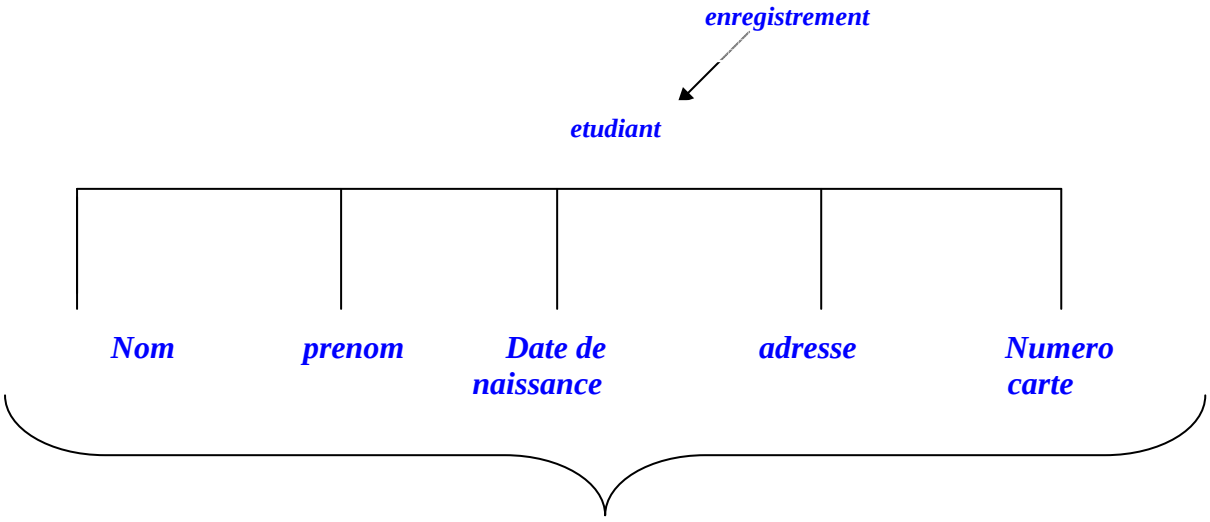
- les dates chronologiques (année, mois, jour),
- les fiches bibliographiques (titre du livre, auteur, date de parution),
- les fiches personnelles (nom, prénom, âge, sexe, taille).

La nature différente de ces éléments (données) conduit le programmeur à utiliser une structure permettant la définition explicite de chacun de ces éléments : structure enregistrement.

Définition:

Un enregistrement est un type structuré formé d'un ou de plusieurs éléments pouvant être de types différents. Chaque enregistrement est désigné par un identificateur (son nom). Chacun des éléments de l'enregistrement occupe un emplacement appelé champ désigné à son tour par un identificateur et caractérisé par un type. Une information représente la valeur du champ.

Exemple :



Champs de l'enregistrement

Etudiant est la variable enregistrement,
Nom, prenom, date de naissance, adresse, numero carte sont les champs de l'enregistrement.

2 Déclaration d'enregistrement

2.1 Définition d'un type d'enregistrement

On procède à la définition d'un type enregistrement puis à la déclaration d'une variable enregistrement de ce type selon la syntaxe suivante :

En pseudo-code :

```
TYPE  id_enr = ENREGISTREMENT
      Chp1 : type1;
      Chp2 :  type2;
      …
      Chpn : typen ;
FIN
Variable idv  : id_enr ;
```

En Pascal:

```
TYPE  id_enr = RECORD
      Chp1 : type1;
      Chp2 : type2;
      …
      Chpn : typen ;
END ;
Var idv  : id_enr ;
```

Où :
id_enr est l'identificateur de type enregistrement,
chp1, chp2, ..., chpn les champs de l'enregistrement,
type1, type2, type3, …, typen les types respectifs des champs chpi (i = 1,..,n),
idv est l'identificateur de la variable enregistrement.

Exemple :

```
TYPE etud = RECORD
      Nom, prenom, ; string[30];
      Date_naissance :string[8] ;
      Adrese : string[50] ;
      Numero_carte : integer;
END;

VAR etudiant : etud ;.
```

Nous avons défini un type enregistrement *etud* et déclaré une variable enregistrement *etudiant* de type *etud*.

Remarque : La définition d'un type enregistrement ne crée pas une variable enregistrement. Pour créer une variable enregistrement, il faut procéder à sa déclaration dans la section VAR.

Exercice d'application :

Définir un type d'enregistrement pour représenter une voiture. On suppose qu'une voiture est caractérisée par sa marque, sa date de fabrication, son numéro d'immatriculation et le nombre de places.

```
TYPE voiture = RECORD
     Marque : string[25] ;
     Date_fabrication :string[8] ;
     Immat :string[8] ;
     Nbreplace : integer ;
END;
```

2.2 Déclaration d'une variable enregistrement sans définition préalable d'un type enregistrement

Syntaxe :

```
VAR  idv   : RECORD
             Chp1 : type1;
             Chp2 : type2;
             …
             Chpn : typen;
     END ;
```

Exemple :

```
VAR  voiture : RECORD
     Marque : string[25] ;
     Date_fabrication :string[8] ;
     Immat :string[8] ;
     Nbreplace : integer ;
END;
```

Cette déclaration crée une variable enregistrement *voiture* composée de quatre champs.

3 Accès aux champs d'un enregistrement

L'accès (ou référence) aux champs d'un enregistrement se fait de deux façons : par nom composé ou par l'instruction WITH.

3.1 Accès par nom composé

L'accès à un champ par cette méthode se fait donc en nommant **la variable enregistrement** puis **le champ** et séparant les deux noms par un point « **.** ».

Syntaxe :

idev.chpi ;

où idev est l'identificateur de la variable enregistrement et chpi est l'identificateur du champ.

Exemple :

Voici les instructions permettant de remplir les champs de l'enregistrement *etudiant* déclaré précédemment (section 2.1) :

```
WRITE('nom :' ) ; READLN(etudiant.nom);
WRITE('prenom :' ) ; READLN(etudiant.prenom);
WRITE('date de naissance:' ) ; READLN(etudiant.date_naissance);
WRITE('adresse :' ) ; READLN(etudiant.adresse);
```

Exercice d'application

Ecrire les instructions permettant d'afficher les champs de l'enregistrement *voiture* déclaré précédemment (section 2.2).

3.2 Accès par l'instruction WITH

Le langage pascal permet, avec l'utilisation de l'instruction WITH, d'éviter la répétition de l'identificateur de la variable enregistrement.

Syntaxe :

WITH idv **DO**
<instruction>

où *idv* est l'identificateur de la variable enregistrement

<instructions> est une instruction simple ou composée (dans ce cas il faut la placer entre un BEGIN et END).

Exemple :

Reprenons l'exemple précédent de la section 3.1

```
WITH  etudiant  DO
BEGIN
    WRITE('nom : ') ; READLN(nom);
    WRITE('prenom : ') ; READLN(.prenom);
    WRITE('date de naissance:' ) ; READLN(date_naissance);
    WRITE('adresse : ') ; READLN(adresse);
END;.
```

Exercice d'application :

Reprendre l'exercice de la section précédente (3.1).

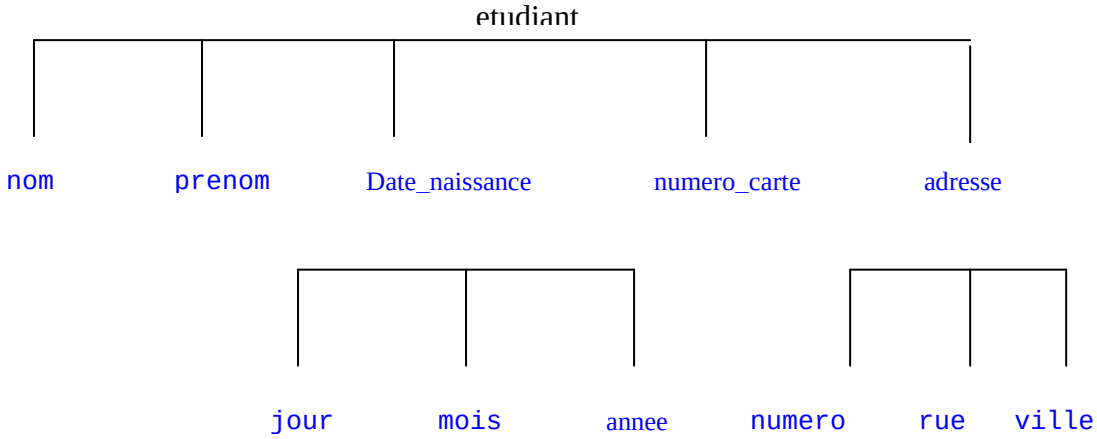
4 Enregistrement d'enregistrements

On parle d'enregistrement d'enregistrements lorsqu'un champ est lui-même composé d'autres champs, autrement dit, le champ est de type enregistrement.

Par exemple, définissons une structure qui permet d'enregistrer les informations relatives à un étudiant en considérant la date de naissance et l'adresse comme des enregistrements.

```
TYPE
    date  =  RECORD
        Jour :integer ;
        Mois : string[12] ;
        Annee : integer ;
    END ;
    adr =  RECORD
        numero_rue :integer ;
        rue : string[50] ;
        ville string[30];
    END ;
```

```
etudiant = RECORD
      Nom, prenom :string[30] ;
      Date_naissance : date ;
      Adresse : adr;
      Numero_carte : integer;
END ;
```



L'accès aux champs de l'enregistrement se fait par niveau en utilisant toujours l'une des méthodes précédentes (par nom composé ou avec l'instruction with).

Ainsi, si on déclare une variable *v_etudiant* de type *etudiant*, on peut accéder au champ jour de la date de naissance comme suit :

```
      WRITE(v_etudiant.date_naissance.jour),
```

ou

```
      WITH v_etudiant DO
            WITH date_naissance DO
                  WRITE(jour)
```

5 Enregistrement avec variantes

Supposons qu'on veuille ajouter à la description d'un étudiant un champ devant représenter le taux de l'allocation de l'étudiant, et qu'on ait envie d'identifier ce champ par *bourse* s'il a une bourse et par *aide* s'il bénéficie d'une aide.

Pascal permet de résoudre ce problème en permettant l'inclusion d'une partie variable dans la définition d'un enregistrement. Cette partie doit être placée après la partie fixe, les champs de la partie variable varient selon la valeur d'un champ appelé sélecteur. A chaque valeur (ou ensemble de valeurs) possible du sélecteur correspond un ou plusieurs champs dont l'ensemble est appelé variante.

Le sélecteur est considéré comme un champ de l'enregistrement.

Syntaxe :

```
      TYPE id_variant = RECORD
            Chp1:   type1;
            Chp2 :  type2;
            ……
            chpk :  typek;
            CASE id__selcteur : typeselecteur OF
```

```
        sel1 : (chpk+1,..typek+1);
        ……
        seln : (chpn,…typen);
    END ;
```

où : id_variant est l'identificateur du type enregistrement.

Chp1,…,chpk : champ de la partie fixe avec leurs types respectifs type1,…typek

Id_selecteur et type sélecteur : identificateur et type de sélecteur de la partie variable

Sel1,..,seln : les valeurs possibles du sélecteur.

Chpk+1,…,chpn : les identificateurs des champs variables et leurs types respectifs typek+1,…,typen.

Exemple :

```
    TYPE etudiant_allocation   =     RECORD
        Nom, prenom :string[32];
        Date_naiss : string[8];
        Adresse:string[60];
        Case allocation : integer of
            1 : (bourse : integer);
            2 : (aide : integer);
    END;
```

Remarques :

1. Une variante peut elle-même contenir une partie finale variable. Par exemple, si on veut, dans le cas d'un étudiant boursier, identifier différemment le champ relatif au taux de la bourse selon que c'est une bourse entière ou une demi-bourse.

```
    TYPE etudiant_allocation   =     RECORD
        Nom, prenom :string[32];
        Date_naiss : string[8];
        Adresse:string[60];
        Case allocation : integer of
            1 : (case b_e : boolean of
                    true : (bourse :integer);
                    false : (demi_bourse : integer) ;) ;
            2 : (aide : integer);
    END ;
```

2. Un seul END suffit pour fermer le RECORD, et le(s) CASE.

6 Opérations permises sur les variables enregistrement

- ***l'affectation*** : il possible d'accéder à tout un enregistrement par cette opération.

Par exemple, l'instruction suivante copie tout l'enregistrement (tous les champs) idv2 dans idv1 où idv1 et idv2 sont des variables enregistrement ayant le même type :

```
    Idv1 : =idv2
```

- ***Constantes d'enregistrement***

Syntaxe :

CONST<idconst> :<id_type>=(<id_chp>:<valeur>{;<id_chp>:<valeur>}) ;

Lorsqu'il s'agit d'une constante de tableau d'enregistrement, il faudra ajouter des parenthèses (une ouvrante et une fermante).

CONST<idconst> :<id_type>=(

(<id_chp>:<valeur>{;<id_chp>:<valeur>}),
(<id_chp>:<valeur>{;<id_chp>:<valeur>}),
…….
…….
) ;

Exemple :

```
TYPE
    Point =     Record
        X; Y;
    End;
    Vecteur     =     Array [0..1] Of Point;
    Mois =      (jan, fev, mar, avr, mai, juin, juil, août, sept, oct, nov, dec) ;
    Date =      record
        Jr:1..31;
        Ms:Mois;
        An :1972..1983 ;
    End ;
    CONST
        Origine : Point=(x :2.0 ; y :2.0)
        Ligne : vecteur =((x :-3.1 ; y :1.5), (x :5.8 ; y :3.0)) ;
        Jour :date=(jr:2 ;ms :dec ;an :1980) ;
```

Remarques :

1. Les opérations de lecture, d'écriture et de comparaison avec les opérateurs relationnels <>, >=, <=, = ne sont pas permises. Elles ne peuvent être effectuées que sur les champs.
2. Les seules expressions d'un type enregistrement sont les variables de ce type.
3. il n'y a pas de fonction (même prédéfini) dont le résultat est de type enregistrement

7 Tableau d'enregistrements

On peut définir des tableaux dont les éléments sont des enregistrements de même type.

Exemple :

Si on veut gérer un parking de voiture, on peut utiliser un tableau composé d'enregistrement de type voiture.

```
CONST    nmax      =      100 ;
TYPE          voitutre    =    RECORD
        Marque : string[25] ;
        Date_fabrication : string[8] ;
        Immat : string[8] ;
        Nbreplace : integer ;
END;

Parking     =     ARRAY [1..nmax] OF voiture;
VAR t :parking;
```

La saisie des éléments de t pourra se faire comme suit :

```
FOR I:=1 TO nmax DO
BEGIN
    READLN(t[i].marque) ; READLN(t[i].date_fabrication) ;
    READLN(t[i].immat) ; READLN(t[i].nbplace) ;
```

```
        END;
```

Le tableau t précédent pouvait être défini comme suit :

```
VAR     t: ARRAY[1..nmax] OF RECORD
                Marque : string[25] ;
                Date_fabrication : string[8] ;
                Immat : string[8] ;
                Nbreplace : integer ;
        END ;
```

Exercice d'application :

Ecrire une procédure permettant d'afficher les éléments de t et une fonction qui renvoie l'indice de l'élément ayant le plus grand nombre de places.

Chapitre 6 : Les Fichiers

1 Généralités

1.1 Introduction

Les types de données étudiés jusqu'ici servaient à décrire des informations. Ces informations étaient toujours perdues quand nous éteignions l'ordinateur. Par exemple si on utilise un tableau d'enregistrements d'étudiants pour gérer la scolarité, il faut à chaque exécution saisir les éléments du tableau.

Dans ce chapitre, nous allons étudier comment remédier à cet inconvénient en créant et en utilisant les fichiers.

Définition : Un fichier est une suite éventuellement vide de composants de même type stockés sur une mémoire secondaire (disquette, disque dur, CD, etc.).

Il existe en permanence, est indépendant de tout traitement et est accessible par un ou plusieurs programmes. Il permet :

- de sauvegarder l'information entre plusieurs exécutions,
- de faire communiquer entre eux plusieurs programmes, qu'ils soient sur une machine ou sur des machines différentes,
- de transmettre des données entre plusieurs machines différentes.

1.2 Les fichiers séquentiels

Un fichier à organisation séquentielle est composé d'articles consécutifs sur la mémoire de masse. Ces articles sont rangés dans l'ordre de leur enregistrement. Un fichier séquentiel est facile à implémenter mais présente quelques inconvénients :

- l'accès est lent pour de grands fichiers,
- l'ajout d'un article n'est possible qu'en fin de fichier.
- Les fichiers en pascal ont une structure séquentielle.

1.2.1 Organisation et Accès

L'organisation définit la manière dont les articles sont disposés sur le support. On distingue trois types d'organisations :

- l'organisation séquentielle qui ne permet que l'accès séquentiel. On accède au fichier dans l'ordre d'enregistrement des articles,
- l'organisation directe qui permet l'accès direct. On accède directement à un élément sans passer par les articles précédents,
- l'organisation séquentielle indexée qui permet l'accès séquentiel et l'accès direct.

1.2.2 Caractérisations générales d'un fichier

Un fichier est caractérisé par :

- le nom,
- l'unité d'information,
- l'organisation,

- le support.

Le nom du fichier permet de l'identifier, on l'appelle nom externe. Il est formé selon les règles du système d'exploitation. Parfois il faut faire précéder ce nom du chemin d'accès.

1.2.3 Traitement des fichiers

- **Association entre une variable fichier et un fichier externe**

Pour pouvoir être utilisée, une variable fichier ou fichier interne (située en mémoire vive) doit impérativement être initialisée en l'associant à un fichier externe.

Toutes les opérations sur le fichier interne affecteront le fichier externe. On parle de fichier disque (le fichier externe) et de fichier logique (le fichier interne).

- **Ouverture d'un fichier**

Avant toute utilisation, il faut avoir le fichier ouvert. L'ouverture du fichier permet au système d'exploitation la création d'une zone de transfert en mémoire, le tampon d'échange.

En général, on distingue trois modes d'ouverture d'un fichier :

- ouverture en lecture seule,
- ouverture en écriture seule,
- ouverture en lecture / écriture.

Selon le mode d'ouverture l'exploitation du tampon sera différente. L'information n'est pas immédiatement déposée sur le disque après l'opération d'écriture ni qu'elle vient du disque après opération de lecture.

- **Fermeture d'un fichier**

En fin d'utilisation, un fichier doit être fermé pour une purge éventuelle du tampon interne. Ce n'est qu'après cette opération que l'on peut être assuré que toutes les opérations d'écriture sont physiquement réalisées sur le fichier disque ou fichier externe.

- **Ecriture et Lecture dans un fichier**

Lorsqu'un fichier est ouvert en écriture, on peut l'agrandir en effectuant une opération d'écriture.

Lorsqu'un fichier est ouvert en lecture ou lecture / écriture, on peut lire, extraire des informations.

2 Les Fichiers en Pascal

En pascal, la structure d'un fichier est séquentielle. Les fichiers sont au nombre de trois :

- Les fichiers typés,
- Les fichiers non typés,
- Les fichiers texte.

Les fichiers typés et non typés sont des fichiers binaires. Le nombre de composants d'un fichier c'est à dire la taille du fichier n'est pas prédeterminée. Il est limité par la taille disponible sur le support. En pascal, chaque fois qu'un composant est écrit ou lu dans un fichier, le pointeur de fichier est automatiquement avancé sur le composant suivant.

2.1 Les fichiers typés

2.1.1 Déclaration

- En pseudo-code

TYPE identificateur _du _type_du fichier = **FICHIER** identif_du type _des_elements ;

VARIABLE identificateur_de _la _variable : identificateur _du _type_du fichier ;
dentificateur_de _la _variable : **FICHIER** identif_du _type_des_elements ;

- En pascal

TYPE identificateur _du _type_du fichier = **FILE OF** identificateur_du type _des_elements ;

VAR identificateur_de _la _variable : identificateur _du _type_du fichier ;
dentificateur_de _la _variable : **FILE OF** identificateur _du _type_des_elements ;;

Un fichier typé en pascal est défini par le mot réservé FILE suivi par le type des composants de ce fichier. Le type des composants d'un fichier peut être quelconque, sauf un type fichier.

2.1.2 Exemples

TYPE

```
fic =file of integer ;
fic1=file of real;
enr=record
      nom,prenom:string[20];
      numero_carte:integer;
end;
fichier_etudiant = file of enr;
```

VAR

```
f1:fic; f2:fic1;
etudiant :fichier_etudiant;
etud : file of fichier_etudiant;.
```

2.1.3 Association entre le fichier interne et le fichier externe

L'association permet de faire le lien logique entre le fichier interne et le fichier externe. Elle est réalisée avec la primitive suivante :

- en pseudo-code
  ASSOCIER(f, nomfic)
- en pascal
  ASSIGN(f, nomfic)

La variable *f* est une variable de type fichier (file of) et *nomfic* est une constante chaîne de caractères ou une variable de type chaîne de caractères. *f* représente le fichier logique et *nomfic* le fichier disque.

Exemple 1

- Le fichier **essai.dat** se trouve dans **le lecteur C** dans le sous répertoire **travail** du répertoire **TP.**

- Le fichier **essai** se trouve dans **la racine du lecteur C.**

en pseudo-code :

```
VARIABLES
     f :FICHIER DE réels
     g:FICHIER D'entiers
debut
     ASSOCIER (F,'c:\tp\travail\essai.dat')
     ASSOCIER(g, 'essai') .
     …
fin.
```

en Pascal :

```
VAR
     f :FILE OF real ;
     g:FILE OF integer;
 begin
     ASSIGN(F, 'c:\tp\travail\essai.dat') ;
     ASSIGN(g, 'essai') ;.
     …
end.
```

Exemple 2

On définit d'abord la constante nomfic, où on met le chemin d'accès du fichier. On appelle la procédure **ASSIGN.**

```
CONST   Nomfic ='c:\tp\travail\donnees.dat' ;
VAR     F : FILE OF … ;
Begin
     ASSIGN(F, nomfic) ;.
     …
end.
```

Exemple 3

On saisit le nom externe du fichier avant de faire l'assignation.

```
VAR nomfic:string;
     F: FILE of …;
begin
     WRITE('Sur quel fichier va-t-on travailler ?') ;
     READLN(nomfic) ;
     ASSIGN(F, nomfic) ;.
     …
End.
```

2.1.4 Ouverture d'un fichier

Avant de pouvoir être exploitée, une variable fichier doit être associée à un fichier externe, puis le fichier doit être ouvert par l'une des méthodes suivantes :

a) ouverture en écriture seule

en pseudo-code

ouvrir(f :fichier)

en pascal

REWRITE (F) ;

où f est une variable fichier.

Un nouveau fichier disque (fichier externe) dont le nom a été préalablement assigné à la variable F est créé et préparé pour le traitement en écriture seule.

Le contenu de tout fichier préexistant avec le même nom est détruit. Un fichier disque créé par REWRITE ne contient aucune information ;

b) ouverture en lecture seule ou en lecture écriture

en pseudo-code

ouvrir(f)

en pascal

RESET(F) ;

Le fichier disque dont le nom est préalablement assigné à la variable *F* est préparé pour le traitement en lecture seule ou écriture. Le pointeur fichier est positionné au début de ce fichier. Le nom du fichier externe correspondant doit exister sur le disque, sinon il se produira une erreur d'entrée/sortie.

2.1.5 Fermeture d'un fichier

L'unique procédure disponible pour fermer un fichier est la suivante :

en pseudo-code

fermer(f :FICHIER)

en pascal

CLOSE(f) ;

La variable interne redevient disponible pour une autre association ou ouverture du même fichier externe dans un autre mode.

Exemple

Les instructions ci-dessus créent un fichier de taille zéro (vide) nommé *essai* et situé sur le lecteur *a* :

```
...
ASSIGN(f, 'a:\essai') ;
REWRITE(f); {creation}
{traitement en écriture seule}
CLOSE(f);
…
```

Les instructions suivantes ouvrent en lecture/écriture le fichier *essai* :

```
...
ASSIGN(f, 'a:\essai') ;
RESET(f);
{traitement en lecture/écriture seule}
CLOSE(f);
…
```

2.1.6 Ecriture dans fichier

Pour écrire dans un fichier typé, il faut l'ouvrir en écriture seule ou en lecture/écriture. L'écriture dans le fichier permet d'agrandir la taille du fichier. Cette opération s'effectue avec la primitive suivante :

en pseudo-code

ecrire(f, v_article)

en pascal

write(f,v_article) ;

où *f* est le fichier logique et *v_article* la variable contenant l'article de même type que les éléments du fichier.

Exemple 1

On veut créer un répertoire téléphonique, un fichier contenant l'ensemble des personnes ainsi que leurs numéros de téléphone.

en pseudo-code

```
TYPE
     personne = ENREGISTREMENT
                Nom :chaîne de 32 caractères
                Prenom, : chaîne de 32 caractères
                Numtel : chaîne de 10 caractères
          FIN DE L'ENREGISTREMENT
   Repertoire_telephonique =    FICHIER DE personne

VARIABLE
      fpers : Repertoire_telephonique
      vpers :personne

      DEBUT
      {on veut écrire l'article Bouba DIOP 961 19 06 dans le fichier.}

           ASSOCIER(fpers,'repertoire')
           OUVRIR(fpers){création, ouverture en écriture seule}
           vpers.nom ←'DIOP'
           vpers.prenom ←'Bouba'
           vpers.numtel ←'961 19 06'
           ECRIRE(fpers,vpers)
           Fermer(fpers)
      FIN
```

en pascal

```
TYPE
     personne = RECORD
                Nom, Prenom:STRING[32] ;
                Numtel : STRING[10];
                END ;
   Repertoire_telephonique= FILE OF personne ;

VAR fpers : Repertoire_telephonique ;
     vpers :personne ;

     BEGIN
          ASSIGN(fpers,'repertoire')
          REWRITE(fpers){création, ouverture en écriture seule}
          vpers.nom  :='DIOP';
          vpers.prenom :='Bouba' ;
          vpers.numtel :='961 19 06;'
          WRITE(fpers,vpers);
```

```
    CLOSE(fpers);
END.
```

Exemple 2

On veut créer un fichier d'étudiants.

```
TYPE    Etud  = RECORD
        Nom, prenom :string[32] ;
        Date_naiss : string[10];
        Num_ins : string[12];
        END ;
VAR     Etudiant : etud;
        Fetud : FILE OF etud;
        i,n :integer ;
BEGIN
        ASSIGN(fetud,'scolaire');
        REWRITE(fetud);
        WRITE('Combien d''etudiants voulez vous saisir ?);
        READLN(n);
        FOR i =1 TO  n DO
            WITH etudiant DO
            BEGIN
                WRITE('Numéro d'inscription :');
                READLN (num_ins);
                WRITE ('Nom:');
                READLN(nom);
                WRITE ('Prénom :');
                READLN (prenom);
                WRITE ('Date de naissance :');
                READLN (date_naiss);
                WRITE(fetud,etudiant);
            END;
        CLOSE(fetud);
END.
```

2.1.7 Lecture dans un fichier

Pour lire dans un fichier typé, il faut l'ouvrir en lecture seule ou en lecture écriture. Cette opération s'effectue avec la primitive suivante :

en pseudo-code

```
lire(f, v_article)
```

en pascal

```
read(f,v_article) ;
```

où f est le fichier logique et v_article la variable contenant l'article de même type que les éléments du fichier.

Exemple:

On veut afficher les personnes du répertoire téléphonique, i.e. lire les enregistrements du fichier.

en pseudo-code

```
DEBUT
    ASSOCIER(fpers,'repertoire')
    OUVRIR(fpers) {lecture, ouverture en écriture seule}
    LIRE(fpers,vpers)
    ECRIRE(vpers.nom)
    ECRIRE (vpers.prenom)
    ECRIRE (vpers.numtel)
    Fermer(fpers)

FIN
```

en pascal

```
BEGIN
        ASSIGN(fpers,'repertoire');
        RESET(fpers) ;
        READ(fpers,vpers);
        WRITE('nom :', Verps.nom);
        WRITE('prenom :',Verps.prenom);
        WRITE('numero de telephone :',Verps.numtel);
        CLOSE(fpers);
END.
```

Exercice d'application :

Afficher le contenu du fichier étudiant.

Remarque

L'opération de lecture ne peut s'exécuter que si le fichier n'est pas vide et, s'il n'est pas vide, il faut que le dispositif de lecture /écriture (le pointeur) ne soit pas à la fin de fichier.

2.2 Fonctions de manipulation des fichiers typés

2.2.1 La fonction fin de fichier

Elle permet de tester si le dispositif de lecture / écriture est à la fin du fichier. Elle retourne vrai si le pointeur de fichier est positionné après la dernière composante d'un fichier. L'opération s'effectue avec la primitive suivante :

EOF(f) ;

Supposons que le fichier disque ***scolaire*** contient plusieurs articles et qu'on ignore le nombre d'articles, si on doit lire et afficher tous les articles du fichier, il faut avant chaque lecture tester si la fin de fichier est atteinte.

```
BEGIN
    ASSIGN(fetud,'scolaire');
    RESET(fetud);
    WHILE NOT EOF(fetud) DO
        WITH etudiant DO
        BEGIN
            READ(fetud,etudiant);
            WRITE('Numéro d'inscription :',num_ins);
            WRITE('Nom:',nom) ;
            WRITE('Prénom :',prenom) ;
            WRITE('Date de naissance :',date_naiss);
        END;
```

```
    CLOSE(fetud)
END.
```

2.2.2 La fonction IORESULT

Elle permet au programmeur de contrôler les éventuelles erreurs d'entrée / sortie (E/S) qui peuvent subvenir lors de l'exécution d'un programme. Par exemple, la tentative d'ouverture en lecture ou en lecture / écriture d'un fichier inexistant ou la tentative d'écriture sur un fichier disque plein provoquent une erreur E/S et une interruption de l'exécution du programme en cours.

Pour éviter ce désagrément, nous devons utiliser les directives de compilation {$I-} et {$I+}. Lorsque le contrôle d'entrées-sorties est désactivé (lorsqu'une série d'instructions est précédée de {$I-}), une erreur E/S ne cause pas l'arrêt du programme, mais suspend toute autre opération E/S jusqu'à ce que la fonction IORESULT soit appelée.

Après chaque opération E/S, nous testons la fonction IORESULT : elle renvoie 0 en cas de succès et un entier différent de 0 sinon.

Exemple

Ecrire un programme qui ouvre en lecture / écriture un fichier de réels dont le nom est saisi au clavier en vérifiant d'abord son existence .

```
VAR
        f : FILE OF REAL;
        nomfic : STRING[11];
BEGIN
        WRITELN('Donner le nom du fichier');
        READLN(nomfic) ;
        ASSIGN(f, nomfic);
        {$I-} {désactive  l'option de détection des erreurs d'E/S}
        RESET(f) ; {l'exécution est arrêtée si f n'existe pas et l'option désactivée}
        {$I+} {réactive l'option de détection des erreurs d'E/S }
        IF IORESULT = 0 THEN
                Write('Fichier inexistent!') ;
        Else
                {traitement du fichier}
        …
        CLOSE(f);
END.
```

2.2.3 La fonction FILEPOS

C'est une fonction entière qui retourne la valeur actuelle du pointeur de fichier. Avec un fichier qui vient d'être ouvert la valeur est zéro.

2.2.4 La fonction FILESIZE

C'est une fonction qui retourne la taille d'un fichier disque exprimée en nombre d'articles. Elle correspond donc à la valeur maximale que l'on peut affecter au pointeur de fichier. Si FILESIZE est égal à 0 le fichier est vide. Si f est un fichier vide alors FILEPOS(f) est égale à FILESIZE(f).

2.2.5 La procédure SEEK

Elle permet de déplacer le pointeur de fichier sur le $n^{ième}$ article du fichier. *n* est une expression entière de type long (longint). Avec cette procédure on peut aller directement à

n'importe quel enregistrement du fichier et il n'a pas besoin de parcourir les enregistrements précédents. La syntaxe est la suivante :

SEEK(f , numero_enregistrement) ;

La position de la première fiche est 0. L'instruction SEEK(f,0) permet de se déplacer au début du fichier. Pour agrandir un fichier on doit déplacer le pointeur juste après le dernier enregistrement et écrire à cet endroit. On utilise l'instruction suivante

SEEK(f, FILESIZE(f)) ;

2.2.6 La procédure TRUNCATE

Elle coupe le fichier à la position courante ; il s'agit de l'unique moyen de réduire la taille d'un fichier.

2.3 Mise à jour d'un fichier

La mise à jour consiste à :

- Créer de nouveaux articles,
- Modifier des articles,
- Supprimer des articles.

Toute opération de consultation ou de mise à jour doit commencer par un travail de recherche, qui consiste à trouver l'élément que l'on veut consulter, modifier, ou supprimer.

La recherche est effectuée sur un champ de l'enregistrement.

Dans l'exemple qui suit, on effectue une recherche dans selon le numéro d'inscription qui est unique. Le programme traite un fichier qui permet de gérer une amicale d'étudiants.

```
program gestion_amical ;
    TYPE
        adherent = record
        num_ins : string[12];
        nom, prenom : string[25] ;
        cod_postal : integer;
        num_tel :string[10];
        end ;
    VAR
        membre : adherent ;
        amical : File of adherent ;
        numero :String[12];
        trouve :boolean;
    BEGIN
        ASSIGN(amical, 'c : \amical\club').
        {$I-}  RESET (amical); {$I+}
        IF IORESULT <> 0 THEN
            WRITELN(' le fichier n'existe pas')
        ELSE
            IF FILESIZE(amical) = 0 THEN
                WITELN('le fichier est vide')
            ELSE
                BEGIN
                    trouve : = FALSE ;
                    WRITELN('taper le numéro à saisir')
                    READLN(numero) ;
                    WHILE NOT EOF(amical) DO
                    BEGIN
```

```
                                    READ(amical, membre);
                                    IF member.num_ins = numero THEN
                                        trouve := TRUE ;
                            END ;
                            IF trouve THEN
                                WRITELN(nom,' ', prenom, ' ', code_postal,' ',
                    num_tel )
                            ELSE
                                WRITELN('le numéro n''exsite pas dans le fichier') ;
                            CLOSE(amical);
                    End;
    END.
```

2.4 Les fichiers texte

2.4.1 Définition

Un fichier texte est un fichier de caractères disposés en ligne de texte. En turbo pascal, on dispose du type prédéfini **TEXT** qui permet de déclarer des fichiers texte.

VAR
identificateur_de _la _variable : **TEXT**;

Exemple
VAR f1,fic, f2:**TEXT**;

L'association entre la variable fichier et le fichier disque se fait de la même manière que pour les fichiers typés.

2.4.2 Ouverture et fermeture d'un fichier texte

Avec les fichiers texte, on dispose de trois modes d'ouverture :

- en écriture seule (la création),
- en écriture seule (ajout d'éléments)
- et en lecture seule.

L'ouverture en lecture seule et l'ouverture en écriture seule (nouveau fichier) s'effectuent de la même manière en appelant respectivement la primitive RESET et la primitive REWRITE.

Pour ouvrir un fichier texte existant en mode ajout, la procédure APPEND doit être utilisée à la place de REWRITE.

```
{$I-}
APPEND(T) ;
IF IORESULT <> 0 THEN
    REWRITE(T);
{$I+}
```

La fermeture se fait avec la primitive CLOSE.

2.4.3 Fonctions de manipulation des fichiers texte

- Fonction **EOF**(var T :TEXT)

Cette fonction permet de tester si le pointeur est positionné sur la marque de fin de fichier. EOF est toujours vrai pour un fichier ouvert avec REWRITE et APPEND.

- Fonction **SEEKEOF**(var T :TEXT) :boolean

Similaire à EOF, mais élimine les espaces, les tabulations, et les lignes vides avant de tester la marque de fin de fichier.

- Fonction **EOLN**(var T /TEXT) : boolean

C'est une fonction booléenne retournant VRAI si la fin de ligne a été atteinte. Si EOF(T) est vrai alors EOLN(T) est également vrai.

- Fonction **SEEKEOLN**(var T :TEXT) :boolean

Similaire à EOLN, mais écarte les espaces, les tabulations, et les lignes vides avant de tester la marque de fin de ligne.

- Les fichiers texte utilisent les primitives **READ**, **READLN**, **WRITE**, **WRITELN** exactement comme pour les entrées sorties au clavier et à l'écran.

La procédure **READ**(var T :TEXT, V1, V2, …, Vn);

Elle permet l'entrée de caractères, chaînes de caractères, et de données numériques.

La procédure **READLN**

Elle identique à READ, excepté que lorsque la dernière variable à été lue, le reste de la ligne est ignoré.

3 Exercice proposé

Une société veut disposer d'un fichier pour gérer les salaires de ses employés. Sur chaque employé, on dispose des informations suivantes: nom (15 caractères), prénom (30 caractères), numéro d'identification (4 chiffres), date d'embauche, statut (marié = 'M' ou célibataire = 'C'), l'année en cours (4 chiffres), le nombre d'heures de travail par mois, la prime (réel), les impôts (réels).

Un employé ayant au moins cinq ans d'ancienneté bénéficie d'une prime de 12 500 F (par mois). Un célibataire paye 30 000 F d'impôts alors qu'un marié n'en paye que 24 000 F. L'heure de travail est rémunérée 1 250 F.

Ecrire un programme permettant, à l'aide d'un fichier séquentiel, la saisie, l'édition, la consultation et le calcul des salaires mensuels des employés de la société.

Chapitre 7 : Les Algorithmes de tri

1 Notion de tri

On désigne par tri l'opération consistant à ordonner un ensemble d'éléments en fonction de clés sur lesquelles est définie une relation d'ordre. Si l'opération de tri est effectuée sur des données stockées en mémoire vive, on parlera de tri interne ; par contre lorsque les données sont enregistrées sur disque, il s'agit de tri externe.

Dans ce chapitre, nous nous intéressons au tri interne.

Il s'agit de trier les *n* éléments d'un ensemble *X*. L'ensemble est trié par une réorganisation dont les opérations de base sont la comparaison et la permutation. Considérons une suite finie $X=(x_1, \ldots, x_n)$ d'éléments appartenant à un ensemble muni d'un ordre. La suite ordonnée $Y=(y_1, \ldots, y_n)$ correspondant à la suite $X=(x_1, \ldots, x_n)$ est une permutation de cette dernière vérifiant :

Pour tout i, $1 \leq i \leq n$ il existe j unique, $1 \leq j \leq n$, tel que $x_i = y_j$

Pour tout i, $1 \leq i \leq n-1$, on a, y_i op y_{i+1}.

Où op est l'un des opérateurs de comparaison ($>$, $<$, $\leq$, $\geq$).

Pour obtenir, à partir d'une suite donnée X, la suite triée Y correspondante, il faut répondre à trois questions :

- Comment peut-on représenter les suites X et Y ?
- Quelle est l'opération élémentaire à utiliser pour effectuer le tri ?
- Avec quel procédé peut-on construire la suite Y ?

La réponse à la première question sera simple : on représentera les deux suites X et Y par un ou plusieurs tableaux, fichiers.

Pour la seconde question, on utilisera la relation d'ordre sur les éléments pour les comparer deux à deux.

La réponse à la troisième question est plus complexe et de très nombreuses solutions peuvent être proposées, qui définissent les différents algorithmes de tris.

2 Quelques algorithmes classiques de tri

Nous allons nous limiter à des données représentées par un tableau. Il existe plusieurs algorithmes de tri dont :

- tri par sélection,
- tri par insertion,
- tri à bulles,
- tri accès indirect

Nous donnerons les algorithmes de tri selon un ordre croissant. *t* représente le tableau à trier, et *n* le nombre d'éléments de *t*.

2.1 Le tri par sélection

2.1.1 Principe :

On parcourt le tableau pour chercher le plus petit élément qui est alors permuté avec le premier élément du tableau.

On fait la même chose avec les n-1 éléments restants du tableau sans le premier élément, puis avec les n-2 éléments, ... et jusqu'à ce qu'il ne reste plus qu'un seul élément (le dernier , lequel est alors le plus grand élément).

2.1.2 Algorithme

```
Variables :
    i, j, aux, imin :entiers
Début
    Pour i allant de 1 à n-1 faire
        Début
            Imin := i
            Pour j allant i-1 à n faire
                Si t[imin]>t[j] alors
                    Imin :=j
            Aux :=t[i]
            T[i] :=t[imin]
            T[imin] :=aux
        Fin(pour i)
Fin
```

2.1.3 Exemple :

Considérons le tableau d'entiers suivant composé de 5 (n=5) éléments.

t

5	1	4	8	2

i ← 1

imin ← i {supposé l'indice du plus petit élément}

j ← i+1 = 2

t[j] = t[2]

t[imin] = t[1]

t[2] > t[1]

imin ne change pas de valeur

j ← j+1 = 3

t[j] = t[3]

t[imin] = t[1]

t[3] < t[1]

imin change de valeur imin ←j(3)

j ← j+1 = 4

t[j] = t[4]

t[imin] = t[3

t[4] > t[3]

imin ne change pas de valeur

j ← j+1 = 5

t[j] = t[5]

t[imin] = t[3]

t[5] < t[3]

imin change de valeur imin ←j(5)

{*on fait la permutation*}
aux ← t[i]
t[i] ← t[imin]
t[imin]← aux
{*après la 1 itération nous obtenons le tableau suivant* }

t

2	1	4	8	5

i ← 2
imin ← i {supposé l'indice du plus petit élément}
j ← i+1 = 3
t[j] = t[3]
t[imin] = t[2]
t[3] < t[2]
imin change de valeur imin ← j(3)

j ← j+1 = 4
t[j] = t[4]
t[imin] = t[3]
t[4] > t[3]
imin ne change pas de valeur

j ← j+1 = 5
t[j] = t[5]
t[imin] = t[3]
t[5] > t[3]
imin ne change pas de valeur

{*on fait la permutation*}
aux ← t[i]
t[i] ← t[imin]
t[imin]← aux
{*après la 2 itération nous obtenons le tableau suivant* }
t

2	4	1	8	5

i ← 3
imin ← i {supposé l'indice du plus petit élément}
j ← j+1 = 4
t[j] = t[4]
t[imin] = t[3]
t[4] < t[3]
imin change de valeur imin ← j(4)

j ← j+1 = 5
t[j] = t[5]
t[imin]= t[4]
t[5] < t[4]
imin change de valeur imin ← j(5)

{*on fait la permutation*}
aux ← t[i]
t[i] ← t[imin]
t[imin]← aux
{*après la 3 itération nous obtenons le tableau suivant* }
t

2	4	5	8	10

i ← 4
imin ← i {supposé l'indice du plus petit élément}
j ← j+1 = 5
t[j] = t[5]
t[imin] = t[4]
t[5] < t[4]
imin ne change pas de valeur

{*on fait la permutation*}
aux ← t[i]
t[i] ← t[imin]
t[imin]← aux
{*après la 4 itération nous obtenons le tableau suivant* }
t

2	4	5	8	10

2.2 Le tri par insertion simple

2.2.1 Principe :

On choisit un élément du tableau, on trie les autres et on insère l'élément initialement choisit à la bonne place en parcourant le tableau. Chaque élément sera inséré à sa bonne place.

2.2.2 Algorithme

```
Variables :
    i, j, x : entiers
Début
    Pour  i allant de 1 à n faire
        Début
            X:=t[i]
            J :=i-1
            Tant que (j>0) et (x<t[j]) faire
                Début
                    T[j+1]:=t[j]
                    J:=j-1
                Fin(tant que)
            T[j+1]:=x
        Fin(pour)
Fin
```

2.2.3 Exemple :

Considérons le tableau d'entiers suivant composé de5 (n=5) éléments.
t

5	1	4	8	2

Début
i ← 2
x ← t[i]
j ← 1
t[j] = 5
x > t[j]
(*on entre pas dans la boucle*)
t[j+1] ← t[2] = x

(*le tableau pour i=2*)

5	1	4	8	2

i ← i+1 = 3
x ← t[i] = 4
j ← i-1 = t[2]
x < t[j]
t[j+1] ←t[j]

5	1	1	8	2

j ← j-1 = 1
x < t[j] = 5
t[j+1] ← t[j]

5	5	1	8	2

la condition n'est plus vérifiée alors t[j+1]= x
{le tableau pour i=3}

4	5	1	8	2

i ← i+1 = 4
x ← t[i] = 8
j ← i-1 = t[3]
x < t[j]
t[j+1] ← t[j]

4	5	1	1	2

j ← j-1 = 2
x > t[j] =
(*on entre pas dans la boucle*)
t[j+1] ← x

(*le tableau pour i=4*)

4	5	8	1	2

i ← i+1 = 5
x ← t[i] = 2
j ← i-1 = t[4]
x < t[j]
t[j+1] ← t[j]

4	5	8	1	10

j ← j-1 = 3
x < t[j]
t[j+1] ← t[j]

4	5	8	8	10

j ← j-1 = 2
x < t[j]
t[j+1] ← t[j]

4	5	5	8	10

j ← j-1 = 1
x < t[j]
t[j+1] ← t[j]

4	4	5	8	10

on sort de la boucle tant que
t[j+1] ← x

{le tableau pour i=5}

2	5	8	1	2

2.3 Le tri à bulles

2.3.1 Principe :

On parcourt le tableau en comparant deux éléments consécutifs, s'ils sont mal placés, on les permute. Cela revient à faire remonter le plus grand élément à chaque parcours, d'où le nom de tri à bulles. Il est aussi appelé tri par permutations ou tri par échanges.

2.3.2 Algorithme (première version)

```
Variables :
    i, dernier :entiers
Début
    dernier:=n-1
    répéter
        Pour i allant 1 à dernier faire
            Si t[i] > t[i+1] alors
                Permuter(t[i], t[i+1])
        dernier:=dernier - 1
    Jusqu'à dernier =1
Fin
```

Exemple :

Considérons le tableau d'entiers suivant composé de n (n=5) éléments.

t

5	1	4	8	2

dernier ← 4
i ← 1
t[i] = 5
t[i+1] = 10
t[i] < t[I+1]
pas de permutation

5	1	4	8	2

i ← 2
t[i] = 10
t[i+1] = 4
t[i] > t[I+1]
on permute

5	4	1	8	2

i ← 3
t[i] = 10
t[i+1] = 8
t[i] > t[I+1]
on permute

5	4	8	1	2

i ← 4
t[i] = 10
t[i+1] = 2
t[i] > t[I+1]

on permute

5	4	8	2	10

dernier ← 3
i ← 1
t[i] = 5
t[i+1] = 4
t[i] > t[I+1]
on permute

4	5	8	2	10

i ← 2
t[i] = 5
t[i+1] = 8
t[i] < t[I+1]
pas de permutation

4	5	8	2	10

i ← 3
t[i] = 8
t[i+1] = 2
t[i] > t[I+1]
on permute

4	5	2	8	10

i ← 4
t[i] = 8
t[i+1]= 10
t[i] < t[I+1]
pas de permutation

4	5	2	8	10

dernier ← 2
i ← 1
t[i] = 4
t[i+1] = 5
t[i] > t[I+1]
pas de permutation

4	5	2	8	10

i ← 2
t[i] = 5
t[i+1] = 2
t[i] < t[I+1]
on permute

4	2	5	8	10

i ← 3
t[i] =5
t[i+1]= 8
t[i] < t[I+1]
pas de permutation

4	2	5	8	10

i ← 4
t[i] = 8
t[i+1]=10
t[i] < t[I+1]

pas de permutation

4	2	5	8	10

dernier ← 1
i ← 1
t[i] =4
t[i+1]= 2
t[i] > t[I+1]
on permute

2	4	5	8	10

i ← 2
t[i] = 4
t[i+1]= 5
t[i] < t[I+1]
pas de permutation

2	4	5	8	10

i ← 3
t[i] = 5
t[i+1] = 8
t[i] < t[I+1]
pas de permutation

2	4	5	8	10

i ← 4
t[i] = 8
t[i+1] = 10
t[i] < t[I+1]
pas de permutation

2	4	5	8	10

Parfois les dernières itérations sont inutiles, elles ne décèlent aucune inversion et n'effectuent donc aucune permutation.

On peut améliorer l'algorithme en arrêtant les itérations si on ne décèle plus d'inversion, alors on obtient l'algorithme suivant.

2.3.3 Algorithme (deuxième version)

```
Variables :
    i, dernier :entiers
    echange :booléen
Début
    dernier:= n-1
    Répéter
        Echange :=faux
        Pour i allant 1 à dernier faire
            Si t[i] > t[i+1] alors
            Début
                Permuter(t[i], t[i+1])
                Echange :=vrai
            Fin(si)
        dernier:=dernier-1
    Jusqu'à echange = faux
Fin
```

2.4 Le tri à accès indirect

2.4.1 Principe

Dans les tris précédents, le classement était réalisé par déplacement effectif des éléments du tableau. Le tri indirect s'opère sur les indices du tableau et non sur ses éléments.

Pour trier les indices, on pourra utiliser l'une des méthodes ci-dessus.

2.4.2 Algorithme

```
Variables :
ti : tableau d'entiers
i, j, aux, min
Début
    Pour i allant de 1 à n faire {on remplit le tableau indice}
        ti[i]:=i
    Pour i allant de 1 à n-1 faire
        Début
            Imin := i
            Pour j allant i-1 à n faire
                Si t[imin]>t[ti[j]] alors
                    Imin :=j
            Aux :=ti[i]
            Ti[i] :=ti[imin]
            Ti[imin] :=aux
        Fin(pour i)
Fin
```

2.4.3 Exemple :

Considérons le tableau d'entiers suivant composé de n (n=5) éléments.

t

5	1	4	8	2

Début

{*le tableau indice *}

ti

1	2	3	4	5

i ← 1
imin ← i {supposé l'indice du plus petit élément}
j ← i+1 = 2
ti[j] = 2
ti[imin] = 1
t[ti[j]] = t[2]
t[2] > t[1]
imin ne change pas de valeur

j ← j+1 = 3
ti[j] = 3
ti[imin = 1
t[ti[j]] = t[3]
t[3] < t[1]
imin change de valeur imin ←ti[j] (3)

j ← j+1 = 4
ti[j] = 4
ti[imin] = 3
t[ti[j]] = t[4]
t[4] > t[3]

imin ne change pas de valeur

j ← j+1 = 5
ti[j] = 5
ti[imin] = 3
t[ti[j]] = t[5]
t[5] < t[3]
imin change de valeur imin ←j(5)

{*on fait la permutation*}
aux ← ti[i]
ti[i] ← ti[imin]
ti[imin] ← aux
{*après la première itération nous obtenons le tableau suivant* }
ti

5	2	3	4	1

i ← 2
imin ← i {supposé l'indice du plus petit élément}
j ← i+1 = 3
ti[j] = 3
ti[imin] = 2
t[ti[j]] < t[3]
imin change de valeur imin ← j(3)

j ← j+1 = 4
ti[j] = 4
ti[imin] = 3
t[ti[j]] > t[3]
imin ne change pas de valeur

j ← j+1 = 5
t[j] = 5
t[imin]= 3
t[5] > t[3]
imin ne change pas de valeur

{*après la 2 itération nous obtenons le tableau suivant* }
ti

5	3	2	4	1

i ← 3
imin ← i {supposé l'indice du plus petit élément}
j ← j+1 = 4
ti[j] = 4
ti[imin] = 2
t[4] < t[2]
imin change de valeur imin ← j(4)
j ← j+1 = 5
ti[j] = 5
ti[imin] = 4
t[5] < t[4]
imin change de valeur imin ← j(5)

{*on fait la permutation*}
aux ← t[i]
t[i] ← t[imin]
t[imin]← aux
{*après la 3 itération nous obtenons le tableau suivant* }

t

5	3	1	4	2

i ← 4
imin ← i {supposé l'indice du plus petit élément}
j ← j+1 = 5
ti[j] = 2
ti[imin] = 4
t[5] < t[4]
imin ne change pas de valeur

ti

5	3	1	4	2

Chapitre 8 : La récursivité

1 Définition et exemples

2.1 Définition

Un algorithme A est dit récursif s'il s'appelle lui-même ou s'il appelle un autre algorithme A' contenant un appel de A.

Un algorithme récursif possède deux propriétés :

(i) Il doit exister des critères pour lesquels les appels cessent,

(ii) Chaque fois que l'algorithme s'appelle (directement ou indirectement), il doit être plus proche de ses critères d'arrêt.

On définit de la même façon une procédure ou une fonction récursive.

1.1 Premier exemple : Calcul du factoriel d'un entier

5! = 5x4x3x2x1=5x4 !
n!= nx(n-1)x…..x1= nx(n-1)! ,n≥1
0! = 1

On peut donc définir le factoriel d'un entier n comme suit :
Factoriel (n) =n x factoriel (n-1) pour n > 0
Et factoriel (0)

On en déduit la fonction récursive suivante :

```
Fonction factoriel (n : entier) : entier ;
Debut
    Si n = 0 alors factoriel : =1
    Sinon factoriel :=n x  factoriel (n-1)
Fin
```

Si on appelle cette fonction avec n=4, on obtient :

Factoriel(4) = 5 x factoriel (4)
4 x factoriel (3)
3 x factoriel (2)
2 x factoriel (1)
1 x factoriel(0)
1
=5x4x3x2x1

1.2 Deuxième exemple : Somme des éléments d'un tableau

On considère un tableau T de n entiers et la somme S(n) de ses éléments
S(n) = T[1] + T[2] +…. +T[n]

Si n = 0 , on a S (n) = 0

Si n ≥ 1 , on a S(n) = s(n-1) + T [n]

Le calcul de S peut se définir de façon récursive grâce à la procédure suivante:

```
Procedure Somme (T : tableau [1..nmax] d'entiers; n : entier; variable S : entier)
Debut
    Si n= 0 alors
    S : =0
    Sinon
    Debut
        Somme (T, n-1, S)
        S := S + T[n]
    Fin
Fin
```

On calcule d'abord la somme des *n-1* éléments à laquelle on ajoute T[n] pour obtenir la somme voulue.

Pour n = 3, on obtient :

Somme (T, 3, S)

Somme (T, 2, S) S := S + T[3] (en attente)

Somme (T, 1, S) S := S + T[2] (en attente)

Somme (T, 0, S) S := S + T[1] (en attente)

S := 0 à ce niveau, on exécute dans l'ordre les instructions en attente :

S := 0 + T[3] = T[3]

S := T[3] + T[2]

S := T[3] + T[2] + T[1]

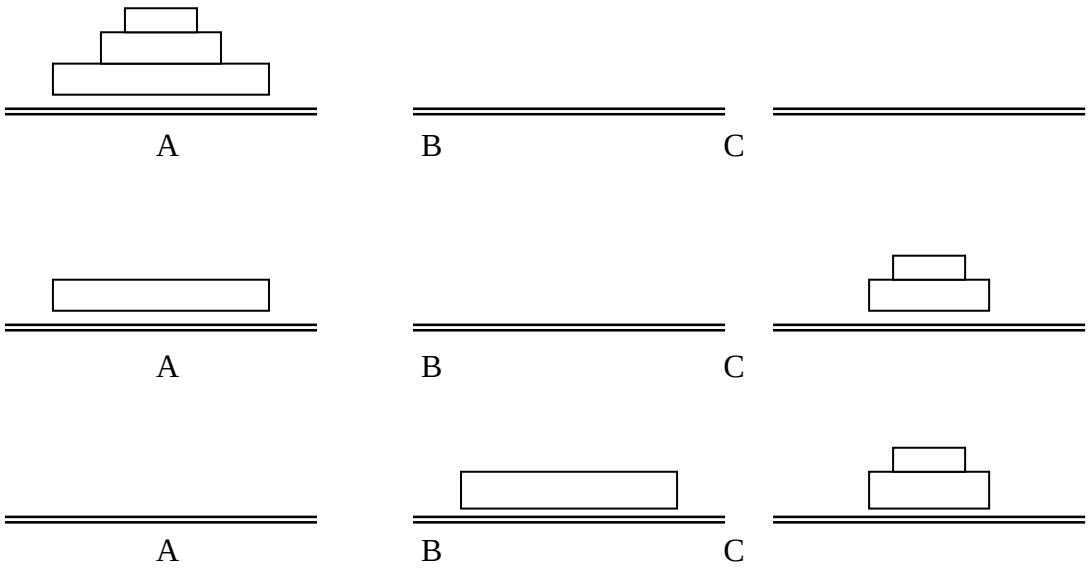
Exercice d'application :

Appliquer l'algorithme pour n = 5.

1.3 Troisième exemple : les tours de Hanoï

Les prêtres d'une certaine secte ont eu à résoudre le problème suivant :

64 disques étant posés les uns sur les autres, par ordre de taille décroissante, sur un plateau A, les transférer sur un plateau B, en utilisant un plateau intermédiaire C. Les seuls déplacements autorisés consistent à prendre un disque au sommet d'une des piles et à le poser sur un disque plus grand ou sur un plateau vide.



On peut résoudre ce problème de façon récursive pour un nombre de disques n donné.

Pour n = 1, c'est trivial, il suffit de déplacer un disque de A à B.

Pour n ≥ 2, on suppose qu'on sait déplacer n-1 disques de A à C.

Comme schématisé dans la figure ci-dessus, on déplace d'abord les n-1 disques du plateau A au plateau C. Ensuite, on déplace le plus grand disque ($n^{ième}$ disque) du plateau A au plateau B. Enfin, il reste à déplacer les n-1 disques du plateau C au plateau B.

Pour cela, on va se servir d'une procédure *deplacer (m, x, y)* qui permet de déplacer un disque m du plateau x au plateau y. En pratique, cette procédure pourrait fournir l'affichage «*déplacer le disque m de x à y*».

La procédure qui résout le problème de Hanoï sera alors :

```
Procedure Hanoi (n : entier ; origine, destination, intermediaire : caracteres)
Debut
    Si n = 1 alors
        Deplacer (1, origine, destination)
    Sinon
    Debut
        Hanoi(n-1, origine, intermediaire, destination)
        Deplacer (n, origine, destination)
        Hanoi(n-1, intermediaire, destination, origine)
    fin
Fin
```

2 Un algorithme de tri récursif : le tri rapide

Le principe de base du tri rapide (ou *quick sort*) est d'opérer le partage en deux de l'ensemble de valeurs à ordonner.

On choisit une valeur de comparaison, généralement la valeur se trouvant à l'indice médian :

$$IM = \text{partie entière } ((debut + fin)/2)$$

où *debut* et *fin* les bornes inférieure et supérieure du tableau traité.

Le partage, énoncé précédemment, a pour but de diviser le tableau en deux parties[2] :

- la première partie contient les valeurs du tableau inférieures à la valeur de comparaison (appelée aussi valeur de partage),
- la deuxième partie contient les valeurs du tableau supérieures à la valeur de comparaison .

Ce partage se fait de la façon suivante : on se déplace vers la droite du tableau tant qu'on reste inférieur à la valeur de comparaison et vers la gauche tant qu'on reste supérieur vers à la valeur de comparaison. On permute les deux éléments mal placés, et on réitère ce processus jusqu'à ce que l'indice de parcours partant de la gauche dépasse celui partant de la droite.

Après le partage, on applique le même processus aux deux parties obtenues, ce qui revient à opérer le tri sur les deux partitions.

Voici la procédure qui réalise ce tri :

[2] On suppose qu'on effectue un tri par ordre croissant.

```
Procedure tri_rapide (variable t : tableau d'entiers ; D, F : entiers)
Variables i, j, vp : entiers;
Debut
    i := D ;
    j := F ;
    vp := t[(D+F) div 2] ;
    repeter
        Tant que t[i] < vp faire i := i+1 ;
        Tant que t[j] > vp faire j := j-1 ;
        Si i <= j alors
        debut
            permuter(t[i], t[j]) ;
            i :=i+1 ;
            j := j-1 ;
        fin
    jusqu'à i > j
    si D < j alors tri_rapide (t, D, j) ;
    si i < F alors tri_rapide (t, i, F) ;
Fin
```

Exemple :

Considérons le tableau ci-dessous, et appliquons-lui l'algorithme de tri rapide :

1	*2*	*3*	*4*	*5*	*6*	*7*
5	**3**	**7**	**1**	**8**	**6**	**4**

1	3	7	**5**	8	6	4
1	3	**4**	5	8	6	**7**
1	3	4	5	**6**	**8**	7
1	3	4	5	6	**7**	**8**

3 Exercices proposés

Exercice 1 : On donne la fonction d'Ackermann définie par:

A(m,n) = A(m-1,A(m,n-1)) pour m et n > 0

A(0,n) = n+1

A(m,0) = A(m-1,1) pour m > 0

Donner un algorithme pour calculer la fonction d'Ackermann d'un couple (m,n) $\in$ N×N.

Exercice 2 : Donner un algorithme récursif qui calcule les n (n entier) premiers nombres de Fibonacci, représentés par la formule :

$F_{n+2} = F_{n+1} + F_n$ avec $F_0 = 0$ et $F_1 = 1$.

Bibliographie

1.