Djelloul BOUCHIHA est né à Mecheria, Wilaya de Naâma, Algérie, là où il a eu son Bac en 1997. Il a suivi ses études de graduation et de post-graduation à l'université de Sidi Bel Abbes, Algérie, là où il a eu son ingéniorat en informatique en 2002, ensuite le diplôme de Magister, de Doctorat en Sciences et d'Habilitation Universitaire, respectivement en 2005, 2011 et 2013. Sur le plan professionnel, il a travaillé comme enseignant vacataire à l'université de Sidi Bel Abbes assurant le module Génie Logiciel en 2004-2005. Entre 2005 et 2012, il était enseignant permanent en qualité de Maître de conférences à l'université de Saida, Algérie, assurant comme modules, entre autres, Algorithmique et structures de données, Système Expert, Web sémantique, Programmation orientée objet, etc. Depuis 2012, il est enseignant au Centre Universitaire –SALHI Ahmed- de Naâma, là où il enseigne comme modules : Algorithmique et structures de données et Programmation mobile. Il a eu son grade Professeur en Janvier 2018. Sur le plan scientifique, il est à la tête de l'équipe de recherche DOS (Développement Orienté Service), rattachée au laboratoire EEDIS (Evotionary Engineering & Distributed Information Systems) de l'université de Sidi Bel Abbes. Il travaille en particulier sur la réingénierie des systèmes d'information et sur l'ingénierie ontologique.

الإيداع القانوني السداسي الثاني

9789947650752

طباعة
مكتبة الرشاد للطباعة والنشر - الجزائر
شارع السكة الحديدية/ سيدي بلعباس/ الجزائر
الهاتف والفاكس: 048684843
المحمول 0773394265
المحمول 0561570602

Prof. Djelloul BOUCHIHA Initiation à l'Algorithmique et à la Programmation en Pascal Editions Errachad

C.U.N

Université Djilali Liabès Sidi Bel-Abbès

Initiation à l'Algorithmique et à la Programmation en Pascal

Prof. Djelloul BOUCHIHA
CENTRE UNIVERSITAIRE -SALHI AHMED- DE NAAMA
EVOLUTIONARY ENGINEERING AND DISTRIBUTED INFORMATION SYSTEMS LABORATORY (EEDIS) UDL-SBA

Editions Errachad

Initiation à l'Algorithmique et à la Programmation en Pascal

Prof. Djelloul BOUCHIHA
CENTRE UNIVERSITAIRE -SALHI AHMED- DE NAAMA
EVOLUTIONARY ENGINEERING AND DISTRIBUTED INFORMATION SYSTEMS LABORATORY (EEDIS) UDL-SBA
djelloul.bouchiha@univ-sba.dz bouchiha.dj@gmail.com

Editions Errachad

بسم الله الرحمن الرحيم

الطبعة الأولى
2019

طباعة
مكتبة الرشاد للطباعة والنشر - الجزائر

شارع السكة الحديدية/ سيدي بلعباس/ الجزائر
الهاتف والفاكس: 048684843
المحمول 0773394265
المحمول 0561570602

ISBN 978-9947-65-075-2

Edition Rached
Sidi Bel Abbes
Algerie

Email: ahmedd.2009@yahoo.fr

Table des matières

Avant-propos

Ce livre constitue un support de cours pour différents enseignements d'algorithmique et de programmation en langage Pascal donnés aux étudiants universitaires ayant une base en mathématiques, notamment ceux appartenant aux filières classées "Sciences et Technologies" et "Sciences Exactes". Il s'agit d'un premier volume introduisant les notions de base de l'algorithmique et les structures de données statiques, et initiant à la programmation en Pascal. Il comporte donc des cours simples, avec des exercices corrigés. Le prochain volume sera consacré à des structures de données dynamiques, dites aussi récursives, et à des notions avancées d'algorithmique.
L'objectif de ce cours est d'apprendre aux étudiants comment résoudre un problème par un programme, commençant par l'analyse de ce problème, déterminer la méthode la plus efficace pour résoudre ce problème, exprimer cette méthode en langage algorithmique, et enfin traduire l'algorithme en langage Pascal.
Nous allons maintenant répondre directement, et d'une façon très simple, à des questions qui viennent spontanément dans la tête de quelqu'un qui lit le titre de ce livre, à savoir "Initiation à l'algorithmique et à la programmation en Pascal" :

Que signifie le terme Initiation ?

Selon Larousse, c'est l'action de donner les premiers rudiments d'une discipline.

Qu'est que c'est que l'algorithmique ?

Un algorithme est une suite d'opérations exécutées dans un ordre pour résoudre un problème ou accomplir une tâche. En tant que science on parle d'algorithmique.
Les opérations d'un algorithme peuvent être écrites en langage naturel, et on parle ainsi de la méthode de représentation d'un algorithme par énumération des étapes. Mais pour un programmeur, un algorithme doit être écrit par un langage algorithmique.

Pour faciliter sa compréhension, un algorithme peut être représenté graphiquement par un organigramme.
Un algorithme ne peut s'exécuter sur machine que s'il est traduit en un langage de programmation, et on obtient un programme au lieu d'un algorithme.
Dans ce cours, nous avons choisi Pascal pour écrire nos programmes.

Qu'est ce que c'est que Pascal ?

Un langage de programmation créé en 1969 à l'école polytechnique de ZURICH par N. WIRTH. Il a été conçu pour permettre d'enseigner la programmation comme une science.

Pour quoi Pascal ?

Pascal a été choisi parce que :

- ☞ c'est un langage conçu initialement pour des objectifs pédagogiques.
- ☞ facile à apprendre.
- ☞ proche au langage algorithmique adopté dans ce cours.

Le programmeur n'a pas donc à se soucier des contraintes imposées par le langage et des spécificités techniques de ce langage. En utilisant Pascal, le programmeur aura l'impression de traduire un algorithme du français à l'anglais.

D'où vient le nom Pascal ?

Il a été donné en l'honneur de Blaise Pascal, scientifique, polémiste et philosophe français (1623-1662) qui inventa la première machine à calculer, le calcul des probabilités, parmi tant de travaux qui marquèrent l'évolution des idées.
Dans ce qui suit, nous allons présenter en détail ce qui vient d'être présenté au-dessus, et nous invitons les lecteurs de cet ouvrage de nous envoyer leurs commentaires, suggestions et corrections.

Chapitre 1 : Introduction

1. Résolution d'un problème par ordinateur

L'ordinateur est une machine capable d'exécuter automatiquement une suite d'opérations. Pour résoudre un problème à l'aide d'un ordinateur, il faut :

1. Analyser ce problème : définir ce que j'ai comme données et ce que j'ai besoin comme résultat.
2. Déterminer la méthode de résolution : déterminer la suite des opérations à effectuer pour résoudre le problème posé. Plusieurs méthodes peuvent être trouvées pour résoudre un seul problème, il faut choisir la plus efficace.
3. Formuler l'algorithme définitif : représenter la méthode de résolution par un algorithme écrit en langage algorithmique, dit aussi un langage de description d'algorithme (LDA).
4. Traduire l'algorithme en un langage de programmation adapté.

2. Notion d'algorithme

Le mot algorithme provient du nom du célèbre mathématicien arabe : Muhammad ibn Musa Al Kwarizmi (780-850). Son nom donna au moyen-âge le nom "algorisme" qui devint algorithme avec Lady Ada Lovelace (1792-1871).
Un algorithme est une suite d'opérations élémentaires exécutées dans un ordre donné pour résoudre un problème ou accomplir une tâche. En tant que science, on parle de l'algorithmique.

3. Exemple

Soit le problème de calcul de la somme de deux nombres. Ce problème peut être résolu de la manière suivante:

A. Analyse

Pour le calcul de la somme:

- Nous avons besoin en entrée de deux valeurs : valeur1 et valeur2.

- En sortie, nous aurons la somme de ces deux valeurs.

B. Solution

Le calcul de la somme consiste à:

1. Avoir les deux valeurs (lire valeur1 et valeur2).
2. Additionner les deux valeurs.
3. Afficher le résultat (Ecrire la somme).

Cette forme de représentation d'un algorithme est dite "Enumération des étapes".

C. Ecriture en LDA

En informatique, on utilise un langage algorithmique pour écrire un algorithme ; et l'algorithme au-dessus devient.

```
Algorithme Somme ;
Variables
 valeur1, valeur2, som : entier ;
Début
 Lire (valeur1, valeur2) ;
 som ← valeur1 + valeur2 ;
 Ecrire (som) ;
Fin.
```

Comme illustré dans l'exemple précédent:

- Un algorithme commence par le mot Algorithme suivit de son nom et un point virgule. Généralement le nom de l'algorithme indique sa fonction.
- Le mot Variables précède la liste des variables manipulées dans l'algorithme et leurs types. Les variables du même type sont séparées par des virgules. Deux déclarations différentes sont séparées par un point virgule.
- Les opérations de l'algorithme sont prises entre les mots Début et Fin indiquant le début et la fin de l'algorithme. Ces opérations sont séparées par des points virgules.
- Le mot Lire permet la lecture à partir du clavier. Le mot Ecrire permet l'affichage à l'écran.
- Le symbole ← correspond à l'opération d'affection. Le symbole + est utilisé pour indiquer l'addition.
- Un algorithme se termine par un point.

D'autres mots et symboles utilisés dans notre langage algorithmique seront découverts dans le reste de ce cours.
Remarque : Le langage algorithmique varie d'un document à un autre. Par exemple, le symbole d'affectation représenté dans notre cours par ← peut être représenté dans un autre document par ::-.
Un algorithme doit être :

- Lisible : clair et facile à comprendre.
- De haut niveau : indépendant du langage de programmation et du système d'exploitation.
- Précis : non ambigu.
- Concis : réduire le nombre d'opérations ainsi que l'espace occupé en mémoire.
- Structuré : organisé.

D. Ecriture en langage de programmation

Pour être exécuté sur un ordinateur, un algorithme doit être traduit en un langage compréhensible par la machine, i.e. un langage de programmation, tel que Pascal, C, Java, etc.
Les algorithmes de ce cours sont traduits en langage Pascal. Le programme correspondant à l'algorithme précèdent est le suivant.

```
program Somme ;
var  valeur1, valeur2, som : integer ;
begin
  read(valeur1, valeur2) ;
  som := valeur1 + valeur2 ;
  write(som) ;
end.
```

Il est possible d'améliorer le programme précédent de la manière suivante:

```
program Somme ;
var  valeur1, valeur2, som : integer ;
begin
  write('Entrez la première valeur : ') ;
  readln(valeur1) ;
  write('Entrez la deuxième valeur : ') ;
  readln(valeur2) ;
  som := valeur1 + valeur2 ;
  write('Somme = ', som) ;
end.
```

Remarques et définitions :

- Dans un algorithme, on parle d'opérations. Dans un programme, on dit instructions. Un programme est donc une suite d'instructions.
- Langage machine ou code machine ou code binaire : instructions de base d'un ordinateur correspondant aux opérations élémentaires de son CPU, et pouvant être directement exécutées sans traduction. Ces instructions sont codées en binaire.
- Compilateur : c'est un logiciel qui transforme un programme écrit en un langage de programmation en un programme dont les instructions sont écrites en langage machine, et donc directement exécutables par l'ordinateur. Le fichier exécutable est parfois appelé fichier binaire car les instructions du langage machine sont codées directement par des nombres binaires. Le fichier contenant le programme d'entrée est appelé fichier source.
- Le langage le plus facile à convertir en code machine est l'assembleur. L'assembleur ou langage d'assemblage, est un langage de programmation dont les opérations correspondent au langage machine (c'est-à-dire aux instructions élémentaires du CPU) mais sont écrites en abréviations textuelles plutôt qu'en code binaire.
  Par exemple, un processeur de la famille x86 reconnaît une instruction du type
  10110000 01100001
  En langage assembleur, cette instruction est représentée par un équivalent plus facile à comprendre pour le programmeur
  movb $0x61,%al
  (10110000 = movb %al
  01100001 = $0x61)
  Ce qui signifie : « écrire le nombre 97 (la valeur est donnée en hexadécimal : $61_{16} = 97_{10}$) dans le registre AL ».

Chapitre 2 : Les algorithmes séquentiels simples

1. Parties d'un algorithme

Un algorithme contient deux parties:

- La partie données (déclaration) : contient les variables et les constantes.
- La partie traitement (code, opérations, corps de l'algorithme) : elle correspond au processus de calcul.

2. Données

Les données sont les objets manipulés dans l'algorithme. Dans un algorithme, toute donnée utilisée doit être déclarée. Les données peuvent être des variables ou des constantes.

Les variables : Une variable correspond à un objet dont la valeur peut varier au cours de déroulement de l'algorithme. Une variable est caractérisée par :

- Le nom (identificateur) qui doit être explicite, i.e. indique le rôle de la variable dans l'algorithme.
- Le type : indique les valeurs qui peuvent être prises par une variable.
- La valeur : indique la grandeur prise par la variable à un moment donné.

Sur le plan technique, une variable correspond à une case mémoire avec : le non de la variable est l'adresse de la case mémoire ; le type indique la taille (le nombre d'octets) de la case ; la valeur représente le contenu de la case.

Les constantes : Une constante est un cas particulier de la variable. Il s'agit d'une variable dont la valeur est inchangeable dans l'algorithme tout entier.

Exemple :	**En Pascal :**
Algorithme calcul_surface ;	program calcul_surface ;
Constantes	const
PI=3.14 ;	PI=3.14 ;
Variables	var
rayon, surface : réel ;	rayon, surface : real ;
…	…

3. Types

Le type correspond au genre ou la nature de la variable que l'on souhaite utiliser. Il indique donc les valeurs qui peuvent être prises par cette variable.
La déclaration d'une variable consiste à l'associer à un type. Chaque type donné peut être manipulé par un ensemble d'opérations.
Il existe des types simples et des types structurés.
Les types simples : sont des types dont les valeurs sont primitives, élémentaires non décomposables. A leur tour, les types simples peuvent être classés en deux catégories :

1. Types numériques :
   - Entier : par exemple 12, -55. Les opérateurs de manipulation des entiers sont :
     - Les opérateurs arithmétiques classiques : +, -, *.
     - La division entière, notée ÷, avec n ÷ p donne la partie entière du quotient de la division de n par p.
     - Le modulo, noté MOD ou %, avec n MOD p donne le reste de la division de n par p.
     - Les opérateurs de comparaison classiques : <, >, =, etc.
   - Réel : par exemple 12.5, -2.09. Les opérateurs de manipulation des réels sont :
     - Les opérations arithmétiques classiques : +, -, *, /.
     - Les opérateurs de comparaison classiques : <, >, =, etc.
2. Types Symboliques :
   - Booléen : les deux valeurs VRAI et FAUX. Les opérateurs de manipulation sont NON, ET, OU.

- Caractère : comporte les données alphanumériques, symboliques, ponctuation, etc., contenant un seul caractère, par exemple 'a', '? ', '3', ' ; '. Notons qu'un caractère se met entre deux guillemets simples. Les opérations de manipulation des caractères sont les opérateurs de comparaison : >, <, =, etc. Le résultat de la comparaison dépend du code ASCII des caractères comparés.

Les types structurées : c'est tout type dont la définition fait référence à d'autre type ; c.-à-d., basé sur les types simples. Ces types sont aussi dits types complexes ou composés. Parmi ces types on cite : le type tableau, chaîne de caractères, enregistrement et ensemble qui seront vus ultérieurement.

4. Opérations de base

La partie traitement d'un algorithme implique un ensemble d'opérations qui peuvent être :

- Opérations de base ou encore élémentaires qui permettent une exécution séquentielle d'un algorithme.
- Structures de contrôle qui permettent le saut dans un algorithme.

Les opérations de base sont l'affectation et les opérations d'entrée/sortie.

4.1. L'affectation

L'opération la plus importante en algorithmique est l'affectation (assignation) qui se note ← (:=en Pascal), et qui consiste à attribuer ou affecter à une variable une valeur appartenant à son domaine de définition (type). La valeur affectée est souvent le résultat de calcul d'une expression arithmétique ou une expression logique.

Exemple :

```
Algorithme calculs ;
…
Début
 X ← 4 ;
 Y ← X * 2 ;
 Z ← Y ;
```

En Pascal :

```
program calculs ;
…
begin
 X := 4 ;
 Y := X * 2 ;
 Z := Y ;
```

Z ← Z – 6 ;	Z := Z – 6 ;
H ← (10>5) ET (2<3) ;	H := (10>5) and (2<3) ;
…	…

L'exemple est lu comme suit : la variable X reçoit la valeur 4. La variable Y reçoit la valeur de X multipliée par 2. La variable Z reçoit la valeur de Y. La variable Z reçoit la valeur courante (actuelle) de Z moins 6. Enfin la variable H reçoit la valeur VRAI.

4.2. Les entrées/sorties

Les échanges d'informations entre l'utilisateur et la machine sont appelés opérations d'entrée-sortie. Les opérations d'entrée-sortie sont :

- Lire () : qui récupère la valeur tapée au clavier et l'affecte à l'espace mémoire désigné par la variable entre parenthèses. En Pascal c'est read();
- Ecrire () : qui récupère la valeur située à l'espace mémoire désigné par la variable entre parenthèses, et affiche cette valeur à l'écran. En Pascal c'est write();

Remarque : Le langage Pascal permet de lire et écrire une variable de type entier, réel, caractère ou chaîne de caractères. Une variable booléenne peut être seulement affichée.

Une chaîne de caractères : est une suite de plusieurs caractères, permettant de représenter des mots ou des phrases. Une chaîne de caractères doit être mise entre deux guillemets simples pour la distinguer d'un identificateur de variable. Par exemple, Ecrire('bonjour'); en Pascal write('bonjour'); permet d'afficher le mot bonjour à l'écran.

5. Construction d'un algorithme simple

La construction d'un algorithme consiste à lui donner un nom, identifier les variables et les constantes et écrire le corps de l'algorithme.

Le corps de l'algorithme est constitué d'une séquence d'opérations mises entre Début et Fin et séparées par des points virgules. Le corps de l'algorithme peut contenir des opérations de lecture, écriture, affection, etc. Pour éclaircir l'algorithme, son corps peut contenir des commentaires mis entre (*…*) ou { }.

Exemple :
L'algorithme de calcul de la surface d'un cercle représenté par énumération des étapes est le suivant :

1. Saisir le rayon du cercle (lire le rayon).
2. Calculer la surface par l'expression : $\Pi*(rayon)^2$.
3. Afficher le résultat (écrire la surface).

En utilisant un langage algorithmique on obtient :

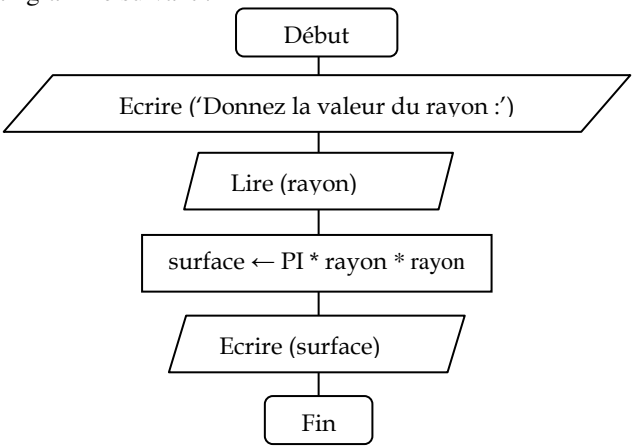
```
Algorithme Calcul_Surface ;
(* algorithme de calcul de la surface d'un cercle *)
Constantes
  PI = 3.14159 ;
Variables
  rayon, surface : réel ;
Début
  Ecrire ('Donnez la valeur du rayon :') ;
  Lire (rayon) ;
  surface ← PI * rayon * rayon;
  Ecrire (surface) ;
Fin.
```

6. Représentation d'un algorithme par un organigramme

Un algorithme peut être représenté aussi par un organigramme facilitant sa compréhension. Un organigramme est une représentation graphique de la solution d'un problème. Pour cela, on utilise les symboles géométriques suivants:

Début/Fin	Début ou fin de l'algorithme.
Lire/Ecrire	Les opérations d'entrée/sortie.
X ← Y + 1	Un traitement (une affectation par exemple).
Oui Condition Non	Un test d'une condition pour une décision ou une sélection.
→	La liaison entre les différents points et indiquent aussi l'ordonnancement des opérations.

L'algorithme de calcul de la surface peut être représenté par l'organigramme suivant :



7. Traduction en langage Pascal

7.1. Exemple

La traduction de l'algorithme de calcul de la surface en un programme Pascal sera comme suit :

```
program Calcul_Surface ;  (* Calcul de la surface d'un cercle *)
const
  pi = 3.14159 ;
var
  rayon, surface : real ;
begin
  writeln('Donnez la valeur du rayon') ;
  readln(rayon) ;
  surface := pi * sqr(rayon) ; (* sqr c'est le carré *)
  writeln(surface) ;
end.
```

7.2. Règles de base

Dans ce qui suit, nous citons quelques règles qui doivent être respectées lors de l'écriture d'un programme Pascal :

1. Un programme Pascal se compose de trois parties : (1) Un en-tête commençant par le mot program ; (2) une section déclarative introduite par le mot var ; (3) une section instructions ou corps du programme, délimitée par les mots begin et end. Le programme se termine par un point.
2. Un identificateur en Pascal (nom du programme, identificateurs de variables ou de constantes, …) ne doit pas dépasser 127 caractères. Il ne doit pas contenir de caractères accentués, ni d'espace, ni des caractères tels que %, ?, *, ., - , etc. Il doit exclusivement être composé des 26 lettres de l'alphabet, des 10 chiffres et du caractère de soulignement. Un chiffre ne peut pas être placé au début d'un identificateur. On note aussi qu'il ne faut pas dupliquer les identificateurs dans un programme.
3. Les constantes sont définies par le mot clé const.
4. Les points virgules sont obligatoires pour séparer les instructions.
5. Les opérations de lecture et d'écriture se traduisent respectivement par read et write (ou readln et writeln) suivies d'une liste de variables ou d'expressions placées entre parenthèses, et séparées par des virgules. L'ajout de ln après write et read force le passage à la ligne lors de l'affichage suivant à l'écran.
6. L'assignation se représente par :=.
7. Les opérateurs arithmétiques sont : +, -, *, /. Pascal utilise deux opérateurs supplémentaires : DIV fournissant la partie entière du quotient de deux nombres entiers, et MOD fournissant le reste de la division de deux nombres entiers.
8. Dans une affectation, le type de l'expression doit correspondre au type de la variable de destination. Cette règle admet les exceptions suivantes : une variable réelle peut recevoir une valeur entière ; une variable chaîne de caractères peut recevoir la valeur d'une variable caractère.
9. Les mots program, var, begin, end, div, mod, integer, read, write, etc. sont des mots réservés ou mots clés qui ne peuvent être utilisés comme identificateurs par le programmeur.

10. Le langage Pascal ne différencie pas entre majuscule et minuscule.
11. Un programme Pascal contient éventuellement des commentaires mises entre (* *) ou { }.

7.3. Différents formats d'affichage

Pour personnaliser le format d'édition, l'instruction write peut être utilisée de différentes manières :

- write(*valeur_entière*) affiche la valeur entière.
- write(*valeur_entière*:n) affiche la valeur entière sur n positions. Par exemple, write(X:5), avec (X=123), affiche ^^123 (^ symbole d'espacement).
- write(*valeur_réelle*) affiche le nombre en notation scientifique à virgule flottante (^x.xxxxxxxxxE±xx). Par exemple, si la variable réelle X contient la valeur 123.4567 alors write(X) affiche ^1.2345670000E+02. on lit 1.234567 * 10^2.
- write(*valeur_réelle*: n1:n2) affiche le nombre sur n1 positions avec n2 décimales (avec ajustement). Par exemple, si la variable réelle X contient la valeur 123.4567 alors write(X:8:2) affiche ^^123.46.
- write(*chaîne*:n) affiche la chaîne de caractères sur n positions. Par exemple, si la variable X de type chaîne de caractères contient la valeur 'AZERTY' alors write(X) affiche AZERTY, write(X:8) affiche ^^AZERTY, et write(X:3) affiche AZERTY.

7.4. Manipulation des nombres

Pascal définit 5 types d'entier :

Type	Intervalle	Occupation en mémoire
SHORTINT	de -128 à +127	1 octet
BYTE	de 0 à 255	1 octet
INTEGER	de -32768 à +32767	2 octets
WORD	de 0 à 65535	2 octets
LONGINT	de -2147483648 à 2147483647	4 octets

Pascal définit 4 types de réels :

Type	Valeurs autorisées	Occupation en mémoire
SINGLE	$1.5 * 10^{-45}$ .. $3.4 * 10^{38}$	4 octets
REAL	$2.9 * 10^{-39}$ .. $1.7 * 10^{38}$	6 octets
DOUBLE	$5.0 * 10^{-324}$ .. $1.7*10^{308}$	8 octets
EXTENDED	$3.4 * 10^{-4932}$ .. $1.1 * 10^{4932}$	10 octets

Remarque : Un nombre réel appartenant aux valeurs autorisées peut prendre le signe positif (+) ou négatif (-).
Les fonctions mathématiques du langage Pascal sont détaillées dans le tableau suivant :

Notation math.	Fonction Pascal	Type de x	Type du résultat	Signification
|x|	ABS(x)	Entier ou réel	Type de x	Valeur absolue de x
x^2	SQR(x)	Entier ou réel	Type de x	Carré de x
$\sqrt{x}$	SQRT(x)	Entier ou réel	Réel	Racine carrée de x
sin(x)	SIN(x)	Entier ou réel	Réel	sin de x (x en radian)
cos(x)	COS(x)	Entier ou réel	Réel	cos de x (x en radian)
arctg(x)	**ARCTAN(x)**	Entier ou réel	Réel	Angle (en radian) dont la tangente vaut x
e^x	EXP(x)	Réel	Réel	Exponentielle de x
ln(x)	LN(x)	Réel	Réel	Logarithme népérien de x
[x]	TRUNC(x)	Réel	Entier	Partie entière de x
[x]	INT(x)	Réel	Réel	Partie entière de x
arrondi de x	**ROUND(x)**	Réel	Entier	Entier le plus proche de x
décimal de x	FRAC(x)	Réel	Réel	Partie décimale de x

7.5. Manipulation des caractères

Un caractère est déclaré par le mot clé CHAR. Une variable caractère (char) occupe 1 octet en mémoire. Chaque caractère

possède un code ASCII (American Standard Code for Information Interchange). A titre d'exemple, les lettres majuscules de 'A' à 'Z' sont codées dans l'ordre par les codes 65 à 90.
La table ASCII est un tableau de 256 caractères, numérotés de 0 à 255, où les 23 premiers sont des caractères de contrôle, associés à des fonctions de base (Suppr, End, Inser, Enter, Esc, Tab, Shift...), et tous les autres sont directement affichables (lettres, ponctuations, symboles, caractères graphiques, chiffres).
Les fonctions avec lesquelles on peut manipuler des caractères sont détaillées dans le tableau suivant :

Fonction Pascal	Type du résultat	Signification
CHR(x)	caractère	Caractère correspondant au code ASCII spécifié entre parenthèses. On peut mettre #x
ORD(c)	entier	Le code ASCII du caractère spécifié entre parenthèses
SUCC(c)	caractère	Le successeur du caractère spécifié entre parenthèses
PRED(c)	caractère	Le prédécesseur du caractère spécifié entre parenthèses

Remarques :
- Chaque caractère doit être mis entre deux guillemets simples pour le distinguer d'un identificateur.
- Il est possible de comparer les caractères suivant l'ordre du code ASCII. Ainsi, ('A'>'B') retourne False.

7.6. Manipulation des booléens

Les variables booléennes, déclarées en Pascal par le mot clé BOOLEAN, peuvent prendre soit la valeur TRUE (VRAI), soit la valeur FALSE (FAUX). Une variable booléenne (boolean) occupe 1 octet en mémoire.
Sur les booléens, on peut effectuer les opérations suivantes : AND, OR et NOT, ainsi que les opérateurs de comparaison : <, >, <>, >=, <=, =.
Il est à noter qu'en Pascal, les priorités données aux opérateurs, de la plus élevée à la plus basse sont :

NOT
* / DIV MOD AND

+ - OR
= <> < <= > >=
Lorsqu'une expression contient plusieurs opérateurs de même priorité, les opérations sont effectuées de gauche à droite. Pour modifier cet ordre, il suffit d'introduire des parenthèses.

8. Exercices corrigés

8.1. Exercices

Exercice 1 :
Ecrire un algorithme permettant de saisir trois nombres, d'en effectuer la somme, le produit et la moyenne, puis les afficher. Traduire l'algorithme en Pascal.

Exercice 2 :
Ecrire un algorithme permettant de saisir deux nombres, de les permuter puis les afficher. Traduire l'algorithme en Pascal.

Exercice 3 :
Ecrire un algorithme qui calcule le périmètre et la surface d'un rectangle. Traduire l'algorithme en Pascal.

8.2. Corrigés

Solution 1 :

```
Algorithme calculs ;
Variables
  somme, produit, moyenne, nb1, nb2, nb3 : réel ;
Début
  Ecrire('Entrez vos trois nombres') ;
  Lire(nb1, nb2, nb3) ;
  somme ← nb1+ nb2 + nb3 ;
  produit ← nb1 * nb2 * nb3 ;
  moyenne ← somme / 3 ;
  Ecrire('La somme de ces trois nombres est : ', somme) ;
  Ecrire('Le produit de ces trois nombres est : ', produit) ;
  Ecrire('La moyenne de ces trois nombres est : ', moyenne) ;
Fin.
```

On peut ne pas utiliser les variables intermédiaires *somme*, *produit* et *moyenne*, et on met directement :

```
  Ecrire('La somme de ces trois nombres est : ', nb1 + nb2 + nb3) ;
  Ecrire('Le produit de ces trois nombres est : ', nb1 * nb2 * nb3) ;
  Ecrire('La moyenne de ces trois nombres est : ', (nb1 + nb2 + nb3) / 3) ;
```

C'est une question de style : dans le premier cas, on favorise la lisibilité de l'algorithme ; dans le deuxième, on favorise l'économie de l'espace mémoire.

Le programme Pascal :

```
program calculs ;
var
  somme, produit, moyenne, nb1, nb2, nb3 : real ;
begin
  writeln ('Entrez vos trois nombres : ') ;
  readln (nb1, nb2, nb3) ;
  somme := nb1 + nb2 + nb3 ;
  produit := nb1 * nb2 * nb3 ;
  moyenne := somme / 3 ;
  writeln ('La somme de ces trois nombres est : ', somme) ;
  writeln ('Le produit de ces trois nombres est : ', produit) ;
  writeln ('La moyenne de ces trois nombres est : ', moyenne) ;
end.
```

Solution 2 :

Permutation en utilisant une variable intermédiaire :

```
Algorithme Permuter1 ;
Variables  nb1, nb2, nb3 : réel ;
Début
  Ecrire('Entrez deux nombres') ;
  Lire(nb1, nb2) ;
  nb3 ← nb1 ;
  nb1 ← nb2 ;
  nb2 ← nb3 ;
  Ecrire('Voici les deux nombres permutés') ;
  Ecrire('nb1 = ', nb1) ;
  Ecrire('nb2 = ', nb2) ;
Fin.
```

Le programme Pascal :

```
program permuter1 ;
var  nb1, nb2, nb3 : real ;
```

```
begin
  writeln ('Entrez deux nombres : ');
  readln (nb1, nb2) ;
  nb3 := nb1 ;
  nb1 := nb2 ;
  nb2 := nb3 ;
  writeln ('Voici les deux nombres permutés : ') ;
  writeln ('nb1 = ', nb1) ;
  writeln ('nb2 = ', nb2) ;
end.
```

Permutation sans variable intermédiaire :

```
Algorithme Permuter2 ;
Variables
  nb1, nb2 : réel ;
Début
  Ecrire('Entrez deux nombres') ;
  Lire(nb1, nb2) ;
  nb1 ← nb1 + nb2 ;
  nb2 ← nb1 – nb2 ;
  nb1 ← nb1 – nb2 ;
  Ecrire('Voici les deux nombres permutés') ;
  Ecrire('nb1 = ', nb1) ;
  Ecrire('nb2 = ', nb2) ;
Fin.
```

Le programme Pascal :

```
program permuter1 ;
var
  nb1, nb2 : real ;
begin
  writeln ('Entrez deux nombres : ');
  readln (nb1, nb2) ;
  nb1 := nb1 + nb2 ;
  nb2 := nb1 - nb2 ;
  nb1 := nb1 - nb2 ;
  writeln ('Voici les deux nombres permutés : ') ;
  writeln ('nb1 = ', nb1) ;
  writeln ('nb2 = ', nb2) ;
end.
```

Solution 3 :

```
Algorithme rectangle ;
Variables
  longueur, largeur, périmètre, surface : réel ;
Début
  Ecrire('Entrez la longueur et la largeur du rectangle') ;
  Lire (longueur, largeur) ;
  Périmètre ← 2 * (longueur + largeur) ;
  surface ←  longueur * largeur ;
  Ecrire('Le périmètre du rectangle est : ', périmètre) ;
  Ecrire('La surface du rectangle est : ', surface) ;
Fin.
```

Le programme Pascal :

```
program rectangle ;
var
  longueur, largeur, perimetre, surface : real ;
begin
  writeln ('Entrez la longueur et la largeur du rectangle : ') ;
  readln (longueur, largeur) ;
  perimetre := 2 * (longueur + largeur) ;
  surface := longueur * largeur ;
  writeln('Le périmètre du rectangle est : ', perimetre) ;
  writeln('La surface du rectangle est : ', surface) ;
end.
```

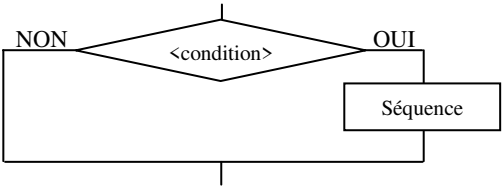
Chapitre 3 : Les structures conditionnelles

1. Introduction

Les algorithmes vus précédemment sont exécutés séquentiellement. Les ruptures des séquences peuvent être effectuées par des structures de contrôle classées en deux catégories : les structures conditionnelles et les boucles. Les structures conditionnelles (simples, composées et multiples) sont aussi appelées structures alternatives, structures de choix ou les tests.

2. Structure conditionnelle simple

Format général : Si <condition> Alors <Séquence>.
Cette opération est lue comme suit : Si la condition est vérifiée (VRAI) alors la séquence d'opérations s'exécute.
La condition est une expression logique qui retourne la valeur VRAI ou FAUX. L'expression peut être simple (condition simple) ou composée (plusieurs conditions composées avec des opérateurs logiques ET, OU et NON).
La séquence peut contenir une ou un ensemble d'opérations. Si la séquence contient plusieurs opérations alors elles sont séparées par des points virgules, et mises entre début et fin. Si la séquence contient une seule opération alors les mots début et fin ne sont pas obligatoires.
La structure conditionnelle simple peut être représentée dans un organigramme comme suit :



Voyons les exemples suivants :

– Si (X > 10) Alors Ecrire(X) ; On affiche la valeur de X si elle est supérieure à 10.

- Si (X > 10) Alors début Ecrire(X) ; Y ← X ; fin ; On affiche la valeur de X et on affecte la valeur de X à Y, si X est supérieure à 10.
- Si (X > 10) ET (X < 15) Alors Ecrire(X) ; On affiche la valeur de X si elle est prise entre 10 et 15.
- Si (X > 10) Alors Si (X < 15) Alors Ecrire(X) ; Cet exemple est équivalent à l'exemple précédent, sauf que cette fois-ci on utilise des tests imbriqués.
- Si (X < 10) Alors Si (X > 15) Alors Ecrire(x) ; Dans cet exemple, la valeur de X n'est jamais affichée car il n'y a aucun cas qui satisfait la condition.

En Pascal, la structure conditionnelle simple s'écrit sous la forme: if <condition> then <Séquence> ;

Exemple : if (X > 10) then write(X);

Si la séquence contient plus d'une instruction alors le begin et end sont obligatoires pour délimiter la suite d'instructions.

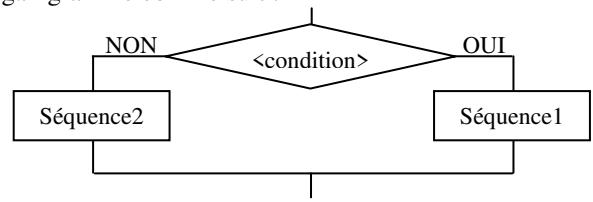
3. Structure conditionnelle composée

Format général : Si <condition> Alors <Séquence1> Sinon <Séquence2>.

Cette opération est lue comme suit : Si la condition est vérifiée (VRAI) alors les opérations de la séquence1 sont exécutées. Dans le cas contraire, ce sont les opérations de la séquence2 qui vont être exécutées.

Les mots début et fin sont utilisés pour délimiter une séquence de plusieurs opérations.

La structure conditionnelle composée peut être représentée dans un organigramme comme suit :



Soit l'exemple suivant : Si (X > 10) Alors Ecrire(X) Sinon Ecrire('valeur non acceptée') ; Dans cet exemple, la valeur de X

est affichée si elle est supérieure à 10, sinon on affiche un message d'erreur.
En Pascal, on aura : if (x > 10) then writeln(x) else writeln ('valeur non acceptée') ;
Si nous avions une suite d'instructions dans une séquence alors on aurait dû utiliser les mots begin et end.

Remarques :

- En Pascal, le else n'est jamais précédé par un point virgule (;).
- Le else se rapporte toujours au if…then le plus proche. Pour casser ce rapport, il est possible d'utiliser le begin et end.
  Par exemple, dans le cas de :

```
if (x > 10) then
  begin if (x < 20) then writeln(x) ; end
else writeln ('valeur non acceptée') ;
```

  le else suit le premier if et non pas le deuxième.

4. Structure conditionnelle multiple

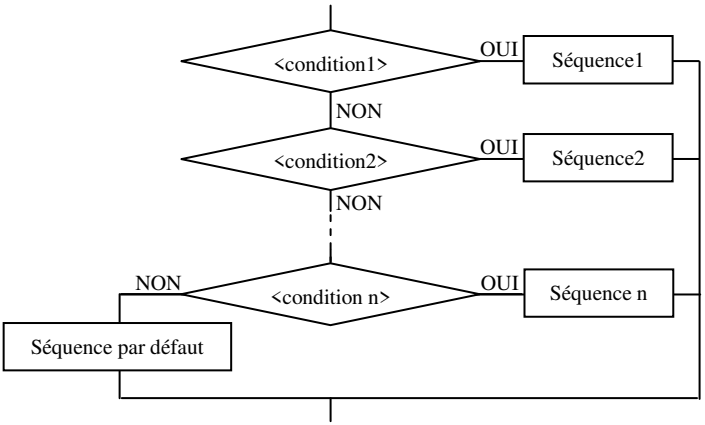
La structure conditionnelle multiple, appelée aussi l'alternative classifiée ou le choix multiple, permet de comparer un objet (variable ou expression) à toute une série de valeurs, et d'exécuter une séquence d'opérations parmi plusieurs, en fonction de la valeur effective de l'objet. Une séquence par défaut peut être prévue dans le cas où l'objet n'est égal à aucune des valeurs énumérées.
Chaque séquence est étiquetée par une valeur. Pour que cette séquence soit choisie, il faut que sa valeur soit équivalente à l'expression. La structure conditionnelle multiple se présente comme suit :

```
Cas <variable ou expression> de
 Valeur 1 : <Séquence1>
 Valeur 2 : <Séquence2>
     …
 Valeur n : <Séquence n>
 Sinon <Séquence par défaut>
fin ;
```

Ceci est équivalent à :

Si (variable ou expression = Valeur1) alors <Séquence1>
Sinon Si (variable ou expression = Valeur2) alors <Séquence2>
…
Sinon Si (variable ou expression = Valeur n) alors <Séquence n>
Sinon <Séquence par défaut>

La structure conditionnelle multiple peut être représentée dans un organigramme comme suit :



Dans l'exemple suivant, la valeur de X est affichée en lettre, si elle est égale à 1 ou 2, sinon afficher un message d'erreur :

```
Cas (X) de
 1 : Ecrire('Un') ;
 2 : Ecrire('Deux')
 Sinon Ecrire('Valeur sup à deux ou inf à un') ;
fin ;
```

En Pascal ça s'écrit :

```
case (X) of
 1 : writeln('Un');
 2 : writeln('Deux')
 else writeln('Valeur sup à deux ou inf à un') ;
end;
```

Remarques :

- Dans l'instruction de choix multiple, l'ordre de présentation ne change rien.
- L'expression et les valeurs à choisir doivent être de même type.
- Si une séquence est composée d'un ensemble d'instructions, elles doivent être prises entre begin et end.

5. Le branchement

Il est possible d'effectuer un saut direct vers une opération en utilisant une opération de branchement de la forme aller à <étiquette>. Les étiquettes sont des adresses qui doivent être déclarées dans la partie déclaration de l'algorithme.

Exemple :

```
Algorithme afficher_nbr_positif ;
 Etiquettes 10 ;
 Variables
  I : entier ;
Début
 Lire (I) ;
 Si (I<0) Alors aller à 10 ;
 Ecrire (I) ;
 10 : Ecrire('Merci') ;
Fin.
```

En Pascal ça s'écrit :

```
program afficher_nbr_positif ;
 label 10 ;
 var   I : integer ;
begin
 readln (I);
 if (I<0) then goto 10 ;
 writeln (I) ;
 10 : writeln('Merci') ;
end.
```

On note qu'il est déconseillé d'utiliser l'instruction de branchement, et cela pour réduire la complexité des programmes en termes de temps.

6. Exercices corrigés

6.1. Exercices

Exercice 1 :
Ecrire un algorithme qui affiche la valeur absolue d'un nombre saisi au clavier. Traduire l'algorithme en Pascal.

Exercice 2 :
Ecrire l'algorithme qui compare deux nombres réels. Traduire l'algorithme en Pascal.

Exercice 3 :
Ecrire l'algorithme permettant à partir d'un menu affiché, d'effectuer la somme ou le produit ou la moyenne de trois nombres. Nous appelons menu, l'association d'un numéro séquentiel aux différents choix proposés. Traduire l'algorithme en Pascal.

Exercice 4 :
Ecrire l'algorithme qui détermine le plus grand des trois nombres lus à partir du clavier. Traduire l'algorithme en Pascal.

Exercice 5 :
Ecrire l'algorithme permettant de lire le prix unitaire (*pu*) d'un produit et la quantité commandée (*qtcom*), ensuite de calculer le prix de livraison (*pl*), le taux de la réduction (*tr*) et en fin calculer le prix à payer (*pap*) sachant que :

- La livraison est gratuite, si le prix des produits (*tot*) est supérieur à 500 DA. Dans le cas contraire, le prix de la livraison est de 2% du *tot*.
- La réduction est de 5%, si *tot* est compris entre 200 et 1000 dinars et de 10% au-delà.

Traduire l'algorithme en Pascal.

6.2. Corrigés

Solution 1 :

```
Algorithme valeur_absolue ;
Variables
  X, Val_abs : réel ;
Début
```

```
 Ecrire('Entrez un nombre') ;  Lire(X) ;
 Val_abs ← X ;
 Si (Val_abs < 0) Alors Val_abs ← -Val_abs ;
 Ecrire('La valeur absolue de ', X, ' est ', Val_abs) ;
Fin.
```

Le programme Pascal :

```
program valeur_absolue ;
 var x, val_abs : real ;
begin
 writeln('Entrez un nombre : ') ;  readln(x) ;
 val_abs := x ;
 if (val_abs < 0) then val_abs := - val_abs;
 writeln ('La valeur absolue de ', x, ' est : ', val_abs) ;
end.
```

Solution 2 :

```
Algorithme comparaison ;
Variables
 X, Y : réel ;
Début
 Ecrire('Entrez deux nombres X et Y) ;  Lire (X, Y) ;
 Si (X = Y) Alors Ecrire(X, ' = ', Y)
   Sinon Si (X > Y) Alors Ecrire(X, ' est supérieur à ', Y)
            Sinon Ecrire(Y, ' est supérieur à ', X) ;
Fin.
```

Le programme Pascal :

```
program comparaison ;
 var x, y : real ;
begin
 writeln (' Entrez deux nombres x et y : ');  readln (x, y) ;
 if (x = y) then writeln(x, ' = ', y)
   else  if (x > y) then writeln(x, ' est supérieur à ', y)
           else writeln(y, ' est supérieur à ', x ) ;
end.
```

Solution 3 :

```
Algorithme menu ;
Variables
 nb1, nb2, nb3 : réel ;
```

```
  Choix : entier ;
Début
  Ecrire('Entrez trois nombres ') ;
  Lire(nb1, nb2, nb3) ;
  { Affichage du menu et saisie du choix } ;
  Ecrire('1-pour la multiplication') ;
  Ecrire('2-pour la somme') ;
  Ecrire('3-pour la moyenne ') ;
  Ecrire('votre choix : ') ;
  Lire (choix) ;
  Cas (choix) de
    1 : Ecrire('Le produit des trois nombres est : ', nb1 * nb2 * nb3) ;
    2 : Ecrire('La somme des trois nombres est : ', nb1 + nb2 + nb3) ;
   3 : Ecrire('La moyenne des trois nombres est : ', (nb1 + nb2 + nb3)/3)
    Sinon Ecrire('saisie de choix incorrecte ') ;
  fin ;
Fin.
```

Le programme Pascal :

```
program menu ;
var nb1, nb2, nb3 : real ; choix : integer ;
begin
  writeln(' Entrez trois nombres : ');
  readln (nb1, nb2, nb3);
  { Affichage du menu et saisie du choix }
  writeln(' 1. pour la multiplication ') ;
  writeln(' 2. pour la somme ') ;
  writeln(' 3. pour la moyenne ') ;
  writeln(' Votre choix :  ') ;
  readln (choix) ;
  case choix of
    1 : writeln (' Le produit des trois nombres est : ', nb1 * nb2 * nb3) ;
    2 : writeln (' La somme des trois nombres est : ', nb1 + nb2 + nb3) ;
    3 : writeln (' La moyenne des trois nombres est : ', (nb1 + nb2 + nb3)/3)
     else writeln (' Saisie de choix incorrecte ' ) ;
  end ;
end.
```

Solution 4 :

```
Algorithme plus_grand  ;
Variables
```

```
  x, y, z, pg : réel ;
Début
  Ecrire (‘Entrez trois nombres :’) ;
  Lire(x, y, z) ;
  Si ( x>= y) et ( x>= z) Alors pg ← x
    Sinon Si (y >= z) Alors pg ← y
              Sinon pg ← z ;
  Ecrire (‘Le plus grand des trois nombres est : ’, pg) ;
Fin.
```

Le programme Pascal :

```
program plus_grand ;
var
  x, y, z, pg : real ;
begin
  writeln ('Entrez trois nombres : ') ;
  readln(x, y, z) ;
  if ( x>=y ) AND (x>=z) then pg := x
    else if (y>=z) then pg := y
            else pg :=  z ;
  writeln ('Le plus grand des trois nombres est : ', pg) ;
end.
```

Solution 5 :

```
Algorithme facture ;
Variables
  Pu, qtcom, tr, pl, tot, pap : réel ;
Début
  Ecrire (‘Entrez le prix unitaire et la quantité commandée’)
  Lire (pu, qtcom) ;
  {calcul du total net}
  tot ← pu * qtcom ;
  {calcul du prix de livraison}
  Si (tot>500) Alors pl ← 0
             Sinon pl ← tot * 0,02 ;
  {calcul de la réduction}
  Si (tot>1000) Alors tr ← tot * 0,10
    Sinon Si (tot>=200) Alors tr ← tot * 0,05
                Sinon tr ← 0 ;
  {calcul du prix à payer}
```

```
  pap ← tot +pl – tr ;
  {édition de la facture}
  Ecrire('Le prix des produits est : ', tot) ;
  Ecrire('Le prix de la livraison : ', pl) ;
  Ecrire('La réduction est de : ', tr) ;
  Ecrire('Le prix à payer : ', pap) ;
Fin.
```

Le programme Pascal :

```
program facture ;
var
  Pu, qtcom, tr, pl, tot, pap : real ;
begin
  writeln('Entrez le prix unitaire et la quantité commandée') ;
  readln(pu, qtcom) ;
  {calcul du total net}
  tot := pu * qtcom ;
  {calcul du prix de livraison}
  if (tot>500) then pl := 0
            else pl := tot * 0.02;
  {calcul de la réduction}
  if (tot>1000) then tr := tot * 0.10
            else if (tot>=200) then tr := tot * 0.05
                      else tr := 0 ;
  {calcul du prix à payer}
  pap := tot + pl - tr ;
  {édition de la facture}
  writeln('Le prix des produits est : ', tot) ;
  writeln('Le prix de la livraison : ', pl) ;
  writeln('La réduction est de : ', tr) ;
  writeln('Le prix à payer : ', pap) ;
end.
```

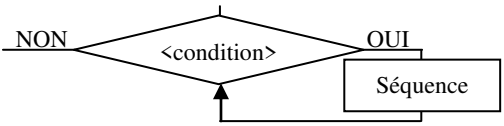
Chapitre 4 : Les boucles

1. Introduction

Lorsqu'on veut répéter une opération ou un bloc d'opérations plusieurs fois dans un algorithme, il est possible d'utiliser des structures répétitives (itératives) appelées les boucles. Il existe trois types de boucles : 'Tant que', 'Répéter' et 'Pour'.

2. La boucle Tant que

Format général : Tant que <Condition> Faire <Séquence>.
Cette structure permet la répétition d'une séquence d'opérations tant que la condition est satisfaite (= VRAI). Quand la condition devient fausse la boucle est terminée.
La condition est une expression logique. Il faut alors que cette expression puisse changer de valeur (avoir la valeur FAUX) pour sortir de la boucle et éviter le cas de la "*boucle infinie*".
La séquence est une ou plusieurs opérations. Pour plusieurs opérations, le début et fin sont obligatoires. Dans le cas d'une seule opération le début et fin sont optionnels.
La boucle Tant que est représentée dans un organigramme comme suit :



Voyons l'exemple suivant qui permet d'afficher les valeurs de 1 à 10 :

```
i ← 1 ;
Tant que (i <= 10) Faire début
        Ecrire (i) ;
        i ← i + 1 ;
    fin ;
```

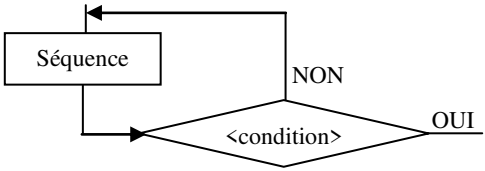
En Pascal, la boucle Tant que s'exprime comme suit :

```
i := 1 ;
```

```
while (i <= 10) do begin
            writeln (i);
            i := i + 1 ;
        end ;
```

3. La boucle Répéter

Format général : Répéter <Séquence> Jusqu'à <Condition>.
Cette structure permet la répétition d'une ou plusieurs opérations jusqu'à ce qu'une condition soit vérifiée (= VRAI). Pour éviter la boucle infinie, il faut que la condition puisse changer de valeur (avoir la valeur VRAI).
La séquence peut contenir une ou plusieurs opérations. Dans les deux cas, début et fin sont facultatifs.
La boucle Répéter peut être représentée dans un organigramme comme suit :



Reprenons l'exemple permettant d'afficher les valeurs de 1 à 10 :

```
i ← 1;
Répéter
  Ecrire (i);
  i ← i + 1;
Jusqu'à (i > 10) ;
```

En Pascal, la boucle Répéter s'exprime comme suit :

```
i := 1 ;
repeat
  write (i);
  i := i + 1 ;
until (i > 10)
```

Remarque : Pour la boucle Tant que et Répéter, on utilise souvent une variable, dite indice, initialisée à une valeur initiale avant d'exécuter la boucle. A l'intérieur de la boucle, l'indice sera incrémenté (en ajoutant une valeur à la valeur courante), ou

décrémenté (en diminuant la valeur courante) pour atteindre la condition d'arrêt de la boucle.
Le tableau suivant récapitule les différences entre les deux boucles Tant que et Répéter:

Boucle	Condition d'exécution	Condition pour quitter	Nombre d'exécutions	Début et fin pour la séquence
Tant que	Condition = VRAI	Condition = FAUX	n'est pas connu d'avance, mais elle peut ne jamais être exécutée si la condition = FAUX dès le départ.	Obligatoires si la séquence contient plusieurs opérations, et facultatifs si la séquence contient une seule opération.
Répéter	Condition = FAUX	Condition = VRAI	n'est pas connu d'avance, mais elle est exécutée au moins une fois.	Facultatifs dans les deux cas.

4. La boucle Pour

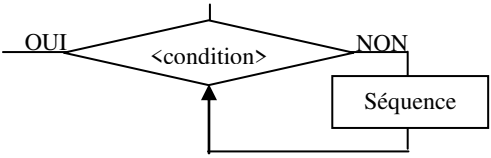
Format général :
Pour <Compteur> ← <Valeur initiale> à <Valeur finale> pas de <Incrément> Faire <Séquence>. Cette structure permet de répéter une séquence d'opérations pour toutes les valeurs d'une variable dite variable de contrôle (Compteur) à partir d'une valeur initiale à une valeur finale.
Le nombre de répétition de la séquence d'instructions est connu d'avance. Après chaque exécution du corps de la boucle, la variable de contrôle est augmentée d'une unité (Incrément). La valeur par défaut de l'incrément est égale à 1 (dans le cas où on ne la précise pas).
La variable de contrôle est de type entier. La valeur initiale et la valeur finale sont des constantes ou des variables de type entier. Ces deux valeurs permettent de calculer le nombre de répétition de la séquence :
nbr = valeur finale - valeur initial + 1 (dans le cas où le pas est égal à 1).

Remarques :

- Dans une boucle Pour, le compteur (variable de contrôle) peut être utilisé, mais il ne doit jamais être modifié à l'intérieur de la boucle.

- Par convention, lorsque l'incrément est égal à 1, il n'est pas indiqué.
- La première valeur, la dernière valeur et l'incrément peuvent être des expressions numériques.

La boucle Pour peut être représentée dans un organigramme comme suit :



Gardons le même exemple permettant d'afficher les valeurs de 1 à 10 :

Pour i ← 1 à 10 Faire Ecrire (i) ;

En Pascal, la boucle pour s'exprime comme suit : for i := 1 to 10 do write(i) ;

L'instruction : for i := 10 downto 1 do write(i) ; affiche les valeurs de 10 à 1.

Remarques :

- En Pascal, la notion de pas de progression n'existe pas explicitement. Le pas est défini implicitement par les expressions to (pour un pas de + 1) et down to (pour un pas de -1).
- L'instruction for est redondante car elle peut être exprimée par les boucles while ou repeat, mais il est recommandé d'utiliser la boucle for chaque fois que c'est possible.
- Pascal permet d'utiliser les formes suivantes :

```
for car := 'a' to 'z' do … ;
for car := 'z' downto 'a' do … ;
for bl := false to true do … ;
for bl := true downto false do … ;
```

  avec (car : char et bl : boolean).
- Le compteur ne doit pas être modifié à l'intérieur de la boucle for, mais la modification de la valeur initiale et la valeur finale n'a aucun effet sur le nombre de répétition de la séquence d'instructions.
- Pour la boucle for i := x to y do … ; avec x>y

ou la boucle for i := x downto y do … ; avec x<y,
la séquence n'est jamais exécutée.

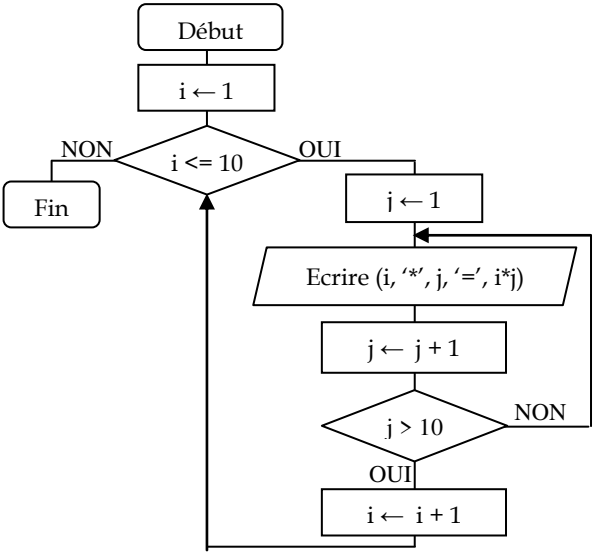
– Pour la boucle for i := x to x do … ;
ou la boucle for i := x downto x do … ;
la séquence est exécutée une seule fois.
– Après avoir quitté la boucle for i := x to n do … ; l'indice i prend la valeur n.

5. Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Une boucle Tant que peut contenir une autre boucle Tant que, une autre boucle Répéter, ou une autre boucle Pour, et vice versa. De plus, une boucle peut contenir une autre boucle, qui elle-même peut contenir une autre boucle et ainsi de suite.
L'algorithme suivant permet d'afficher les tables de multiplication de 1 jusqu'à 10.

```
Algorithme boucles_imbriquées ;
Variables
        i, j : entier ;
Début
 i ← 1 ;
 Tant que (i<=10) Faire début
  j ← 1 ;
  Répéter
    Ecrire (i, '*', j, '=', i*j) ;
    j ← j + 1 ;
  Jusqu'à (j > 10) ;
  i ← i + 1 ;
 Fin ;
Fin.
```

L'organigramme correspondant à l'algorithme précèdent est le suivant :



6. Exercices corrigés

6.1. Exercices

Exercice 1 :
Ecrire des algorithmes permettant de calculer la somme de la suite des nombres 1, 2, 3, …, n (n est un entier lu à partir du clavier) en utilisant les boucles : *Tant que, Répéter* et *Pour*. Traduire ces algorithmes en langage Pascal.

Exercice 2 :
Ecrire un algorithme qui calcule la factorielle d'un nombre entier n strictement positif sachant que :

n! = 1*2*3*…*n, et que

0! = 1, par convention (n! se lit : n factorielle).

La factorielle d'un nombre négatif n'existe pas. Il est donc nécessaire que l'algorithme vérifie si le nombre donné n'est pas négatif.
Traduire l'algorithme en Pascal.

Exercice 3 :
Ecrire un algorithme permettant de saisir une série de nombres entiers positifs, puis propose indéfiniment (en boucle) à l'utilisateur, par l'intermédiaire d'une sorte de menu à choix multiple, d'afficher la valeur minimale, la valeur maximale, la somme ou la moyenne des nombres entrés, ou encore de quitter le programme.
Traduire l'algorithme en Pascal.

6.2. Corrigés

Solution 1 :

Les algorithmes en utilisant les trois boucles sont les suivants:

```
Algorithme somme_suite1 ;
Variables   i, n, somme : entier ;
Début
  Ecrire ('Donnez un nombre') ;
  Lire (n) ;
  Si (n > 0) Alors début
    somme ← 0 ;
    i ← 1 ;
    Tant que (i <= n) Faire début
      somme ← somme + i ;
      i ← i + 1 ;
    fin ;
    Ecrire ('La somme de la suite = ', somme) ;
  fin
    Sinon Ecrire ('Le nombre doit être positif') ;
Fin.

Algorithme somme_suite2 ;
Variables   i, n, somme : entier ;
Début
  Ecrire ('Donnez un nombre') ;
  Lire (n) ;
  Si (n > 0) Alors début
    somme ← 0 ;
    i ← 1 ;
    Répéter
      somme ← somme + i ;
```

```
    i ← i + 1 ;
  jusqu'à (i > n) ;
   Ecrire ('La somme de la suite = ', somme) ;
 Fin
   Sinon Ecrire ('Le nombre doit être positif') ;
Fin.

Algorithme somme_suite3 ;
Variables   i, n, somme : entier ;
Début
 Ecrire ('Donnez un nombre') ;
 Lire (n) ;
 Si (n > 0) Alors début
   somme ← 0 ;
   Pour i ← 1 à n Faire somme ← somme + i ;
   Ecrire ('La somme de la suite = ', somme) ;
 fin
   Sinon Ecrire ('Le nombre doit être positif') ;
Fin.
```

Les programmes Pascal en utilisant les trois boucles sont les suivants :

```
program somme_suite1 ;
var
 i, n, somme : integer ;
begin
 writeln('Donnez un nombre') ;
 readln(n) ;
 if (n > 0) then begin
   somme := 0;
   i := 1 ;
   while (i <= n) do begin
    somme := somme + i ;
    i := i + 1 ;
   end ;
   writeln('La somme de la suite = ', somme) ;
 end
   else writeln ('Le nombre doit être positif') ;
end.
```

```
program somme_suite2;
var   i, n, somme : integer ;
begin
  writeln('Donnez un nombre') ;
  readln(n) ;
  if (n > 0) then begin
    somme := 0;
    i := 1 ;
    repeat
      somme := somme + i ;
      i := i + 1 ;
    until (i > n) ;
    writeln ('La somme de la suite = ', somme) ;
  end
    else writeln ('Le nombre doit être positif') ;
end.
```

```
program somme_suite3;
var   i, n, somme : integer ;
begin
  writeln('Donnez un nombre') ;
  readln(n) ;
  if (n > 0) then begin
    somme := 0;
    for i := 1 to n do somme := somme + i ;
    writeln ('La somme de la suite = ', somme) ;
  end
    else write ('Le nombre doit être positif') ;
end.
```

Solution 2 :

```
Algorithme factorielle_n ;
Variables
  i, n, factorielle : entier ;
Début
  Ecrire (‘Entez un nombre’) ;
  Lire (n) ;
  Si (n < 0) Alors Ecrire (‘La factorielle d’’un nombre négatif n’’existe pas’)
    Sinon début
      factorielle ← 1 ;
```

```
    i ← 1 ;
    Tant que (i <= n) Faire début
      factorielle ← factorielle * i ;
      i ← i + 1
    fin ;
    Ecrire ('La factorielle de ', n, ' = ', factorielle) ;
  fin ;
Fin.
```

Le programme Pascal :

```
program factorielle_n ;
var
 i, n, factorielle : integer ;
begin
 writeln('Enter un nombre') ;
 readln(n) ;
 if (n < 0) then writeln('La factorielle d''un nombre négatif n''existe pas')
  else begin
    factorielle := 1 ;
    i := 1 ;
    while (i<=n) do begin
      factorielle := factorielle * i ;
      i := i + 1 ;
    end ;
    writeln('La factorielle de ',n, ' = ', factorielle) ;
  end ;
end.
```

Solution 3 :

```
Algorithme calculs_menu ;
Variables
 i, choix : entier ;
 s, n, moyenne, minimum, maximum : réel ;
Début
 Ecrire ('Entrez successivement des nombres positifs') ;
 Ecrire ('Entrez un nombre négatif pour finir') ;
 Lire (n) ;
 Si (n >= 0) Alors début
  s ← n ;
```

```
   minimum ← n ;   maximum ← n ;
   i ← 1 ;
   Répéter
    Lire (n);
    Si (n >= 0) Alors début
     Si (n < minimum) Alors minimum ← n ;
     Si (n > maximum) Alors maximum ← n ;
     s ← s + n ;
     i ← i + 1 ;
    fin ;
   Jusqu'à (n < 0) ;
   moyenne ← s/i ;
   Répéter
    Ecrire ('Choisissez entre : ') ;
    Ecrire ('1 : minimum') ;
    Ecrire ('2 : maximum') ;
    Ecrire ('3 : somme') ;
    Ecrire ('4 : moyenne') ;
    Ecrire ('0 : stop') ;
    Ecrire ('Entrez votre choix') ;
    Lire (choix) ;
    Cas choix de
     1 : Ecrire ('Le minimum est : ', minimum) ;
     2 : Ecrire ('Le maximum est : ', maximum) ;
     3 : Ecrire ('La somme est : ', s) ;
     4 : Ecrire ('La moyenne est : ', moyenne) ;
     0 : début fin
     Sinon Ecrire('Choix non accepté') ;
    fin ;
   Jusqu'à (choix = 0) ;
 fin ;
Fin.
```

Le programme Pascal :

```
program calculs_menu ;
var
 i, choix : integer ;
 s, n, moyenne, minimum, maximum : real ;
begin
 writeln('Entrez successivement des nombres positifs') ;
```

```
  writeln('Entrez un nombre négatif pour finir') ;
  readln(n);
  if (n >= 0) then begin
    s := n;
    minimum := n;    maximum := n;
    i := 1;
    repeat
      readln(n);
      if (n >= 0) then begin
        if (n < minimum) then minimum := n ;
        if (n > maximum) then maximum := n ;
        s := s + n;
        i := i + 1;
      end;
    until (n < 0);
    moyenne := s/i;
    repeat
      writeln('Choisissez entre : ') ;
      writeln('1 : minimum') ;
      writeln('2 : maximum') ;
      writeln('3 : somme') ;
      writeln('4 : moyenne') ;
      writeln('0 : stop') ;
      writeln('Entrez votre choix') ;
      readln(choix) ;
      case (choix) of
        1 : writeln('Le minimum est : ', minimum) ;
        2 : writeln('Le maximum est : ', maximum) ;
        3 : writeln('La somme est : ', s) ;
        4 : writeln('La moyenne est : ', moyenne) ;
        0 : begin end
        else writeln('Choix non accepté') ;
      end ;
    until (choix=0) ;
  end;
end.
```

Chapitre 5 : Les tableaux et les chaînes de caractères

1. Introduction

Les types présentés jusqu'à maintenant sont des types simples. Il existe d'autres types dits structurés (complexes). Un type structuré est tout type défini à base d'autres types. Commençant par le type qu'on juge le plus important des types structurés, à savoir le type tableau.

2. Le type tableau

Un tableau est une structure homogène composée d'un ensemble d'éléments de même type de données. Format général : Nom_tab : Tableau [indice_min..indice_max] de type_données.
Un tableau possède un nom (Nom_tab). Il est aussi caractérisé par deux indices (indice_min, indice_max). Le type des éléments du tableau est indiqué par type_données.
Par exemple, pour déclarer un tableau Note de dix éléments réels on met : Note : Tableau [1..10] de réel ;. Cet exemple nous a permis de substituer la déclaration de 10 variables Note1, Note2, …, Note10 de type réel par une seule structure, à savoir le tableau Note.
En Pascal on écrit : Note : Array [1..10] of real ;
La capacité d'un tableau (le nombre d'éléments que peut contenir le tableau) ne peut pas être changée au cours d'exécution d'un programme. Il faut donc que la taille du tableau soit suffisamment grande pour la manipulation. Pour une souplesse de programmation, on peut utiliser une constante comme suit :

```
const  Max = 10 ;
var  Note = Array[1.. Max] of real ;
```

Remarques :

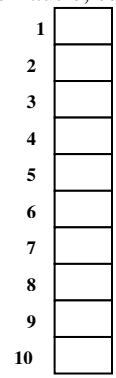
- Les deux indices (indice_min, indice_max) doivent être de type ordinal (scalaire ou discret), généralement entier. Un type est dit ordinal, si ses éléments sont dénombrables, ordonnés avec une valeur min et une valeur max. Par

exemple, en Pascal, les types integer, char et boolean sont des types ordinaux; le type réel n'est pas un type ordinal.

- Les deux indices permettent de calculer le nombre d'éléments du tableau. Dans le cas de type entier, nbr_éléments = indice_max - indice_min + 1

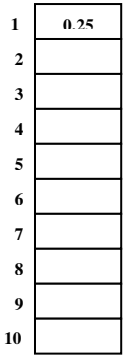
2.1. Manipulation d'un tableau

Un tableau peut être représenté par un ensemble de cases, l'une à côté de l'autre, ou l'une sous l'autre, schématisées comme suit :



Représentation du tableau Note

Un élément du tableau est accessible par sa position (indice) dans le tableau. Le tableau Note est un tableau à une seule dimension, là où chaque élément est accessible par un seul indice. Par exemple Note[1] ← 0.25; permet d'affecter la valeur 0.25 à la première case du tableau. Le tableau Note devient:



L'indice peut être exprimé directement comme un nombre en clair. Il peut aussi être une variable ou une expression calculée.
Soit par exemple :

```
Algorithme exe_tab;
Variables
 T : tableau [1..5] d'entier;
 X : entier;
Début
 X ← 3;
 T[1] ← 7;
 T[X] ← 18;
 T[X-1] ← 4
Fin.
```

Après l'exécution des opérations de l'algorithme, le tableau T est représenté comme suit :

1	**7**
2	**4**
3	**18**
4	
5	

Remarques :

- Pour la manipulation des tableaux, on utilise fréquemment les boucles.
- Pascal permet les déclarations suivantes :

```
var
 A : Array [1..10] of real ;
 B : Array [0..9] of boolean ;
 C : Array[boolean] of boolean ;
 H : Array ['a'..'e'] of Array [1..4] of real;
 E : Array[1..7] of string ;
```

Ainsi, les affectations suivantes sont possibles :

```
A[1] :=1 ;   B[0] :=TRUE ;   C[FALSE]:=TRUE;
H['b'][1]:=3; E[1]:='Samedi';
…
```

Exercice 1 : (Création, initialisation et édition d'un tableau)
Problème : Ecrire l'algorithme, ensuite le programme Pascal, permettant de créer un tableau de dix entiers, d'initialiser ses éléments à 0, ensuite de les afficher.
Solution :

```
Algorithme init_tab ;
Variables
  T : tableau [1..10] d'entier ;
  i : entier ;
Début
  Pour i ← 1 à 10 Faire T[i] ← 0 ;
  Pour i ← 1 à 10 Faire Ecrire(T[i]) ;
Fin.
```

Le programme Pascal :

```
program init_tab ;
var
  T : array [1..10] of integer ;
  i : integer;
begin
  for i := 1 to 10 do T[i] := 0 ;
  for i := 1 to 10 do writeln(T[i]);
end.
```

Exercice 2 : (Création, lecture et affichage d'un tableau)
Problème : Ecrire l'algorithme, ensuite le programme Pascal, permettant de créer un tableau de dix caractères, de lire ses éléments à partir du clavier, ensuite de les afficher.
Solution :

```
Algorithme lire_tab ;
Variables
  T : tableau [1..10] de caractère ;
  i : entier ;
Début
  Pour i ← 1 à 10 Faire Lire (T[i]) ;
  Pour i ← 1 à 10 Faire Ecrire(T[i]) ;
Fin.
```

Le programme Pascal :

```
program lire_tab ;
```

```
var
  T : array [1..10] of char ;
  i : integer;
begin
  for i := 1 to 10 do readln(T[i]);
  for i := 1 to 10 do writeln(T[i]);
end.
```

Exercice 3 : (Recherche dans un tableau)
Problème : Ecrire l'algorithme permettant la recherche d'un nombre lu à partir du clavier dans un tableau de dix réels.
Solution : Il existe plusieurs stratégies pour la recherche dans un tableau. Nous allons voir deux : la recherche en utilisant la technique de Flag, et la recherche dichotomique.
Technique de Flag : Le flag, en anglais, veut dire drapeau. Dans cette première méthode, on a une variable booléenne initialisée à FAUX (drapeau baissé), et qui aura la valeur VRAI dès qu'un événement attendu se produit (drapeau levé). La technique de Flag peut être utilisée dans le problème de recherche et dans n'importe quel problème pareil.
La recherche dichotomique : La dichotomie (« couper en deux » en grec). La recherche dichotomique exige que les éléments du tableau soient ordonnés préalablement. Elle consiste à comparer le nombre à rechercher avec le nombre qui se trouve au milieu du tableau. S'ils sont égaux on arrête la recherche et on déclare que l'élément existe. Sinon, si le nombre recherché est inférieur, on devra continuer la recherche dorénavant dans la première moitié. Sinon, on devra le rechercher dans la deuxième moitié. A partir de là, on prend la moitié du tableau qui nous reste, et on recommence la recherche. La recherche s'arrête quand il ne reste plus de place là où rechercher et on déclare que le nombre recherché n'existe pas dans le tableau.
La recherche en utilisant la technique de Flag :

```
Algorithme Recherche_Flag ;
Variables
  NOMBRES : tableau [1..10] de réel ;
  NBR : réel ;
  Flag : booléen ;
  i : entier ;
```

```
Début
  Ecrire('Entrez les nombres du tableau') ;
  Pour i ← 1 à 10 Faire Lire(NOMBRES[i]) ;
  Ecrire('Entrez le nombre à rechercher : ') ;
  Lire(NBR) ;
  Flag ← FAUX ;
  Pour i ← 1 à 10 Faire Si (NOMBRES[i]=NBR) Alors Flag ←
VRAI ;
  Si (Flag) Alors Ecrire(NBR,' fait partie du tableau')
                  Sinon Ecrire(NBR,' ne fait pas partie du tableau') ;
Fin.
```

La recherche dichotomique :

```
Algorithme Recherche_dichotomique ;
Variables
  NOMBRES : tableau [1..10] de réel ;
  NBR : réel ;   TROUVE : booléen ;
  i, Sup, Inf, Milieu : entier ;
Début
  Ecrire('Entrez les nombres triés du tableau') ;
  Pour i ← 1 à 10 Faire Lire(NOMBRES[i]) ;
  Ecrire('Entrez le nombre à rechercher : ') ;
  Lire(NBR) ;
  Sup ← 10 ;
  Inf ← 1 ;
  TROUVE ← FAUX ;
  Tant que (NON TROUVE) ET (Sup >= Inf) Faire début
     Milieu ← (Sup + Inf) ÷ 2 ;
     Si (NBR = NOMBRES[Milieu]) Alors TROUVE ← VRAI
        Sinon Si (NBR < NOMBRES[Milieu]) Alors  Sup ← Milieu – 1
                  Sinon Inf ← Milieu + 1 ;
  fin ;
  Si TROUVE Alors Ecrire('Le nombre ', NBR, ' existe dans le tableau')
     Sinon Ecrire('Le nombre ', NBR, ' n''existe pas dans le tableau') ;
Fin.
```

Remarque : On note que la recherche dichotomique contient la technique de Flag, car on a besoin d'un test d'arrêt au cas où l'élément recherché est trouvé.

Exercice 4 : (Calcul de la somme, le produit et la moyenne des éléments numériques d'un tableau)

Problème : Ecrire l'algorithme, ensuite le programme Pascal, permettant de calculer la somme, le produit et la moyenne d'un tableau de dix réels.

Solution :

```
Algorithme Som_Moy_Produit_tab ;
Variables
  T : tableau [1..10] de réel ;
  i : entier ;  somme, produit, moyenne : réel ;
Début
  Pour i ← 1 à 10 Faire Lire(T[i]) ;
  somme ← 0;
  produit ← 1 ;
  Pour i ← 1 à 10 Faire début
    somme ← somme + T[i] ;
    produit ← produit * T[i] ;
  fin ;
  moyenne ← somme / 10 ;
  Ecrire('La somme = ', somme) ;
  Ecrire('Le produit = ', produit) ;
  Ecrire('La moyenne = ', moyenne) ;
Fin.
```

Le programme Pascal :

```
program Som_Moy_Produit_tab ;
var
  T : array [1..10] of real ;
  i : integer;
  somme, produit, moyenne : real;
begin
  for i := 1 to 10 do readln(T[i]);
  somme := 0;
  produit := 1 ;
  for i := 1 to 10 do begin
    somme := somme + T[i];
    produit := produit * T[i];
  end;
  moyenne := somme / 10;
  writeln ('La somme = ', somme);
```

```
  writeln ('Le produit = ', produit);
  writeln ('La moyenne = ', moyenne);
end.
```

Exercice 5 : (Recherche du plus petit et plus grand élément dans un tableau)

Problème : Ecrire l'algorithme permettant de déterminer la valeur maximale et la valeur minimale dans un tableau de dix entiers, avec leurs positions dans le tableau.

Solution :

```
Algorithme MaxMin_tab ;
Variables
  T : tableau [1..10] d'entier ;
  max, min, pos_min, pos_max, i : entier ;
Début
  Pour i ← 1 à 10 Faire Lire(T[i]) ;
  max ← T[1] ;
  pos_max ← 1 ;
  Pour i ← 2 à 10 Faire Si (max < T[i]) Alors début
                          max ← T[i] ;
                          pos_max ← i ;
                        fin ;
  min ← T[1] ;
  pos_min ← 1 ;
  Pour i ← 2 à 10 Faire Si (min > T[i]) Alors début
                          min ← T[i] ;
                          pos_min ← i ;
                        fin ;
  Ecrire('La valeur maximale = ', max, ' occupant la position ', pos_max) ;
  Ecrire('La valeur minimale = ', min, ' occupant la position ', pos_min) ;
Fin.
```

2.2. Tri d'un tableau

Problème : soit un tableau de N entiers rangés n'importe comment. On cherche à modifier le tableau de telle manière que les entiers y soient rangés par ordre croissant.

Plusieurs stratégies peuvent être utilisées pour résoudre le problème de tri.

- Tri par sélection.

- Tri à bulles.
- Tri par insertion.
- Tri rapide (*quick sort*).
- Tri fusion (*merge sort*).
- Tri par tas (*heap sort*).

Dans ce qui suit, nous allons voir deux stratégies : le tri par sélection, et le tri à bulles.

Tri par sélection :

Dans ce cas, le tri d'un tableau consiste à mettre en bonne position l'élément numéro 1, c'est-à-dire le plus petit. Puis, on met en bonne position l'élément suivant. Et ainsi de suite, jusqu'au dernier.

```
Algorithme Tri_tab_par_sélection ;
Variables
 T : tableau [1..10] d'entier ;
 x, i, j : entier ;
Début
 Pour i ← 1 à 10 Faire Lire(T[i]) ;
 Pour i ← 1 à 9 Faire
   Pour j ← i+1 à 10 Faire Si (T[i] > T[j]) Alors début
                                    x ← T[i] ;
                                    T[i] ← T[j] ;
                                    T[j] ← x ;
                                   fin ;
 Pour i ← 1 à 10 Faire Ecrire(T[i]) ;
Fin.
```

Le tri à bulles :

L'idée de départ du tri à bulles consiste à se dire qu'un tableau trié par ordre croissant, est un tableau dans lequel tout élément est plus petit que celui qui le suit.

```
Algorithme Tri_tab_bulles ;
Variables
 T : tableau [1..10] d'entier ;
 i, x : entier ;
 Permut : booléen ;
Début
 Pour i ← 1 à 10 Faire Lire(T[i]) ;
 Répéter
   Permut ← FAUX ;
```

```
    Pour i ← 1 à 9 Faire Si (t[i] > t[i+1]) Alors début
                                x ← t[i] ;
                                t[i] ← t[i+1] ;
                                t[i+1] ← x ;
                                Permut ← VRAI ;
                               fin ;
   Jusqu'à Non Permut ;
  Pour i ← 1 à 10 Faire Ecrire(T[i]);
Fin.
```

2.3. Tableau à deux dimensions

Un élément du tableau peut être lui même de type tableau.
Par exemple, Note : Tableau [1..10] de Tableau [1..3] de réel ;
Dans ce cas, un élément du tableau est accessible par deux indices.
Par exemple, Note[4][2] ← 5 ; ou Note[4,2] ← 5 ;
Une deuxième méthode consiste à utiliser un seul tableau à deux dimensions pour les lignes et les colonnes.
Format général : Nom_tab : Tableau [indl_min..indl_max, indc_min..indc_max] de type_données ;
L'exemple suivant permet de déclarer un tableau de réels nommé Note à deux dimensions 3 * 10 :
Note : Tableau [1..3, 1..10] de réel.
En Pascal on écrit : Note : Array [1..3, 1..10] of real ;
Un tableau à deux dimensions peut être représenté par un ensemble de cases organisées en lignes et colonnes, et schématisées comme suit :

	1	2	3	4	5	6	7	8	9	10
1	0.25	11	55	29.06	17.5	0.5	0	43	10	10.25
2	30.77	166	5.44	9.6	7.57	10.5	770	4.83	510	10.5
3	0.62	61	0	2.06	17.5	80.95	0.04	473	10	1.25

Un élément du tableau est accessible par deux indices (ligne et colonne). Par exemple, Note[2,3] :=5.44.
Voyons un deuxième exemple :

```
program matrice;
var   H : Array ['a'..'z', 1..10] of real ;
begin
  H ['b', 3] := 2.25 ;
end.
```

Les tableaux les plus utilisés sont à une ou deux dimensions, mais il est possible de définir des tableaux à trois ou à quatre dimensions, voire plus. Par exemple, Note : Tableau [1..10, 1..3, 'a'..'z', booleen] de réel.

Remarque : on utilise parfois le terme vecteur pour désigner un tableau à une dimension et le terme matrice pour désigner un tableau à deux dimensions.

3. Les chaînes de caractères

Une chaîne de caractères est une suite de caractères. Elle peut être considérée tout simplement comme un tableau de caractères sur lequel on peut effectuer des opérations supplémentaires. Ces opérations dépendent du langage de programmation considéré.

Chaque caractère peut être manipulé en utilisant son indice. Les indices valides pour une chaîne de caractères sont des entiers compris entre 1 et la longueur de la chaîne.

3.1. Déclaration d'une chaîne de caractères

En Pascal, une chaîne de caractères est déclarée en utilisant le mot clé String. Une chaîne de caractères déclarée par String peut prendre au maximum 255 caractères, et elle occupe 256 octets.

Ce langage permet aussi de préciser la taille maximale de la chaîne. Pour une chaîne de 25 caractères maximum, on met String[25]. Une chaîne de caractères déclarée par String[n] peut prendre au maximum n caractères, et elle occupe n+1 octets.

3.2. Manipulation des chaînes de caractères

On peut affecter à toute variable chaîne, une expression chaîne de caractères, en utilisant le symbole d'affectation traditionnel :=.

L'exemple suivant montre différentes possibilités d'affectation sur les chaînes de caractères :

```
var
  CH1, CH2, CH3, CH4, CH5 : string[15] ;
  C : char ;
begin
  CH1 := 'bonjour' ;
  CH2 :='Aujourd''hui';
  C := 'a';
```

```
  CH3 := C;
  CH4 := '';
  CH5 := 'a';
end.
```

Une chaîne de caractères doit être mise entre deux guillemets simples pour la distinguer d'un identificateur de variable. Dans l'exemple précédent, la variable CH1 reçoit ʻbonjourʻ. Si la chaîne contenait elle-même un guillemet, alors il faut le doubler, CH2 :='Aujourd''hui' ;. La variable CH3 reçoit la valeur de la variable de type caractère C (l'inverse n'est pas accepté, i.e. on ne peut pas affecter une chaîne de caractères à une variable de type caractère). La variable CH4 reçoit la chaîne vide et la variable CH5 reçoit un seul caractère (ʻaʻ).
La chaîne de caractères CH1 peut être représentée de la manière suivante :

0	1	2	3	4	5	6	7	8	9	10
•	b	o	n	j	o	u	r			

Le premier caractère de la chaîne a pour indice 1. Le dernier élément correspond à l'indice 7 (=Long(CH1)). Le reste de l'espace réservé à la chaîne sera rempli par des caractères null possédant le code ASCII 0. L'élément de l'indice 0 contient une variable de type Char dont la valeur numérique correspond à la longueur de la chaîne. Le code ASCII du caractère • est 7 (la longueur de la chaîne ʻbonjourʻ).
Les fonctions de manipulation des chaînes de caractères sont détaillées dans le tableau suivant.

Notation en LDA	**Fonction Pascal**	**Type du résultat**	**Signification**
Long(Ch)	LENGTH(Ch)	Entier	Nombre de caractères dans *Ch*
Ch1 + Ch2	CONCAT(Ch1,Ch2) Ch1 + Ch2	Chaîne	Concaténation (juxtaposition) de *Ch1* et *Ch2*
$Ch_{i,l}$	COPY(Ch, i, l)	Chaîne	Extraction, dans *Ch*, de la chaîne de caractères à partir de la position *i* avec une longueur *l*
Ch[i]	Ch[i]	Caractère	Caractère, dans *Ch*, de la postion *i*

Par exemple, l'affectation Ch := concat('aa','bb','cc') ; donne la valeur 'aabbcc' à Ch.
upcase(car) permet de convertir un caractère simple (car) en majuscule. On peut aussi comparer des chaînes de caractères par <, >, =, etc. Le résultat de la comparaison dépend de l'ordre lexicographique.
Remarque : Il est possible de déclarer une constante chaîne de caractères comme suit : const ch = 'Merci';

Exemple :
```
program chaine;
var   ch : string[10] ;
begin
 ch := 'bonjour' ;
 write (length(ch), ' ', Concat(ch, ' Ahmed'), ' ',Copy(ch, 2, 4), ' ', ch[5], ' ', ch[length(ch)], ch:9, ' ') ;
 if (ch > 'merci') then write(ch) else write ('merci');
end.
```

Le programme Pascal précédent affiche :
7 bonjour Ahmed onjo o r bonjour merci

3.3. Tableau de chaînes de caractères

On peut aussi déclarer un tableau de chaînes de caractères qui est un tableau de tableau. Pour déclarer par exemple un tableau de 5 chaînes de 20 caractères on écrit : var tab : array [1..5] of string[20] ;
On accède à chacune des chaînes en utilisant l'élément tab[i] du tableau, et on accède à n'importe quel caractère *j* de la chaîne *i* par tab[i,j] ou tab[i][j].

4. Exercices corrigés

4.1. Exercices

Exercice 1 :
Ecrire un algorithme permettant de remplir deux tableaux. Chacun peut stocker 10 entiers. L'algorithme doit permettre aussi de

calculer la somme des deux tableaux et stocker le résultat dans un troisième tableau. Traduire l'algorithme en Pascal.

Exercice 2 :
Ecrire l'algorithme permettant de remplir un tableau à deux dimensions (3*10) et de déterminer le nombre d'apparition d'une certaine valeur lue à partir du clavier. Traduire l'algorithme en Pascal.

Exercice 3 :
Ecrire l'algorithme permettant de remplir un tableau à deux dimensions (10*20) et de calculer les totaux par ligne et colonne dans les tableaux TotLig et TotCol. Traduire l'algorithme en Pascal.

Exercice 4 :
Ecrire un programme Pascal permettant de lire un tableau de 10 chaînes de 5 caractères maximum, puis affiche le dernier caractère de chaque chaîne.

4.2. Corrigés

Solution 1 :

```
Algorithme Somme_tab ;
Variables
  T1, T2, T3 : tableau [1..10] d'entier ;
  i : entier ;
Début
  Pour i ← 1 à 10 Faire Lire (T1[i]) ;
  Pour i ← 1 à 10 Faire Lire (T2[i]) ;
  Pour i ← 1 à 10 Faire T3[i] ← T1[i] + T2[i] ;
  Pour i ← 1 à 10 Faire Ecrire (T3[i]) ;
Fin.
```

Le programme Pascal :

```
program Somme_tab ;
var
  T1, T2, T3 : array [1..10] of integer ;
  i : integer;
begin
  for i := 1 to 10 do readln(T1[i]) ;
```

```
  for i := 1 to 10 do readln(T2[i]) ;
  for i := 1 to 10 do T3[i] := T1[i] + T2[i] ;
  for i := 1 to 10 do writeln(T3[i]) ;
end.
```

Solution 2 :

```
Algorithme Nbr_app ;
Variables
  T: tableau [1..3, 1..10] d'entier ;
  i, j, x, nb : entier ;
Début
  Pour i ← 1 à 3 Faire Pour j ← 1 à 10 Faire Lire(T[i,j]) ;
  Ecrire ('Donnez une valeur :') ;
  Lire (x) ;
  nb ← 0 ;
  Pour i ← 1 à 3 Faire Pour j ← 1 à 10 Faire Si (x = T[i,j]) Alors
nb ← nb + 1 ;
  Ecrire('Le nombre d''apparition de la valeur ', x, ' dans la
matrice est = ', nb) ;
Fin.
```

Le programme Pascal :

```
program Nbr_app ;
var
  T : array [1..3, 1..10] of integer ;
  i, j, x, nb : integer;
begin
  for i := 1 to 3 do for j := 1 to 10 do readln(T[i,j]);
  writeln('Donnez une valeur :') ;
  readln(x) ;
  nb := 0 ;
  for i := 1 to 3 do for j := 1 to 10 do if (x = T[i,j]) then nb := nb + 1;
  writeln('Le nombre d''apparition de la valeur ', x, ' dans la
matrice est = ', nb) ;
end.
```

Solution 3 :

```
Algorithme Somme_ligne_col ;
Variables
```

```
  T: tableau [1..10, 1..20] de réel ;
  TotLig : tableau [1..10] de réel ;
  TotCol : tableau [1..20] de réel ;
  i, j : entier ;
Début
  Ecrire('Donnez les valeurs de la matrice');
  Pour i ← 1 à 10 Faire Pour j ← 1 à 20 Faire Lire(T[i,j]) ;
  Pour i ← 1 à 10 Faire TotLig[i] ← 0 ;
  Pour i ← 1 à 20 Faire TotCol[i] ← 0 ;
  Pour i ← 1 à 10 Faire Pour j ← 1 à 20 Faire début
    TotLig[i] ← TotLig[i] + T[i,j] ;
    TotCol[j] ← TotCol[j] + T[i,j] ;
  fin ;
  Ecrire('Total lignes');
  Pour i ← 1 à 10 Faire Ecrire(TotLig[i]) ;
  Ecrire('Total colonnes');
  Pour i ← 1 à 20 Faire Ecrire(TotCol[i]) ;
Fin.
```

Le programme Pascal :

```
program Somme_ligne_col ;
var
  T: array[1..10, 1..20] of real ;
  TotLig : array [1..10] of real ;
  TotCol : array[1..20] of real ;
  i, j : integer ;
begin
  writeln('Donnez les valeurs de la matrice');
  for i := 1 to 10 do for j := 1 to 20 do readln(T[i,j]) ;
  for i := 1 to 10 do TotLig[i] := 0;
  for i := 1 to 20 do TotCol[i] := 0 ;
  for i := 1 to 10 do for j := 1 to 20 do begin
              TotLig[i] := TotLig[i] + T[i,j];
              TotCol[j] := TotCol[j] + T[i,j] ;
            end ;
  writeln('Total lignes');
  for i := 1 to 10 do writeln(TotLig[i]);
  writeln('Total colonnes');
  for i := 1 to 20 do writeln(TotCol[i]);
end.
```

Solution 4 :

```
program Saisie ;
var   T : array [1..10] of string[5];
  i : integer;
begin
  writeln('Saisir les chaînes de caractères');
  for i:= 1 to 10 do readln(T[i]);
  writeln('Les derniers caractères des chaînes');
  for i:= 1 to 10 do
    writeln('Le dernier caractère de la chaîne ', T[i], ' est ',T[i][length(T[i])]);
end.
```

Chapitre 6 : Les sous-programmes : Procédures et Fonctions

1. Introduction

Pour avoir des programmes fiables, lisibles et faciles à maintenir, il faut les réduire et les organiser en utilisant les sous-programmes. Ces derniers permettent donc :

1. La réduction de la taille des programmes : il est possible de déterminer les blocs analogues, les substituer par un sous-programme, ensuite appeler le sous-programme par son nom, dans des points d'appel au niveau du programme appelant.
   Soit le programme Pascal suivant :

```
program Calcul_Som ;
var
  x, y, s : integer ;
begin
  writeln('*** bloc 1 *** ') ;
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
  writeln('*** bloc 2 ***') ;
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
end.
```

   Pour réduire le code de ce programme, il est possible d'utiliser un sous-programme (dans cet exemple une procédure) comme suit :

```
program Calcul_Som ;
var
```

```
  x, y, s : integer ;
procedure somme;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
end;
begin
  writeln('*** bloc 1 *** ') ;
  somme;
  writeln('*** bloc 2 ***') ;
  somme;
end.
```

2. L'organisation du code : le problème initial sera découpé en sous-problèmes. Chaque sous-problème sera résolu par un sous-programme.
   Soit le programme Pascal suivant :

```
program addition_multiplication_soustraction ;
var
  x, y, s, p, d : integer ;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
  p := x * y ;
  writeln('Le produit = ', p) ;
  d := x - y ;
  writeln('La différence = ', d) ;
end.
```

   Pour plus d'organisation, il est possible de subdiviser le problème en trois sous-problèmes (addition, multiplication et soustraction). Chaque sous-problème sera résolu par un sous-

programme (dans ce cas une procédure). On obtient le programme suivant :

```
program addition_multiplication_soustraction ;
var
  x, y, s, p, d : integer ;
procedure somme;
begin
  s := x + y ;
  writeln('La somme = ', s) ;
end;
procedure produit;
begin
  p := x * y ;
  writeln('Le produit = ', p) ;
end;
procedure difference;
begin
  d := x - y ;
  writeln('La différence = ', d) ;
end;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  somme;
  produit;
  difference;
end.
```

2. Les sous-programmes

Un programme peut contenir dans sa partie déclaration des étiquettes, des constantes, des types, des variables et des sous-programmes. Un sous-programme est en fait un petit programme composé d'un entête, d'une partie déclaration et d'une partie instructions. Le programme Calcul_Som peut être écrit comme suit :

```
program Calcul_Som ;
var
```

```
  s : integer ;
procedure somme;
var
  x, y : integer ;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
end;
begin
  writeln('*** bloc 1 *** ') ;
  somme;
  writeln('*** bloc 2 ***') ;
  somme;
end.
```

Un sous-programme peut contenir dans sa partie déclaration d'autres sous-programmes qui à leur tour peuvent contenir d'autres sous-programmes, etc. Le programme addition_multiplication_soustraction peut être écrit comme suit :

```
program addition_multiplication_soustraction ;
var  x, y : integer ;
procedure addition_multiplication;
var  s : integer;
  procedure produit;
  var
    p : integer;
  begin
    p := x * y ;
    writeln('Le produit = ', p) ;
  end;
begin
  s := x + y ;
  writeln('La somme = ', s) ;
  produit;
end;
```

```
procedure difference;
var
  d : integer;
begin
  d := x - y ;
  writeln('La différence = ', d) ;
end;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  addition_multiplication ;
  difference;
end.
```

Dans le cas où on a des blocs qui se ressemblent en termes d'instructions, mais ils utilisent des variables différentes, on peut utiliser un sous-programme avec des paramètres (arguments).
Soit le programme Pascal suivant :

```
program Calcul_Som ;
var  x, y, z, h, s : integer ;
begin
  writeln('*** bloc 1 *** ') ;
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  s := x + y ;
  writeln('La somme = ', s) ;
  writeln('*** bloc 2 ***') ;
  writeln('Donnez la première valeur ') ;
  readln(z) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(h) ;
  s := z + h ;
  writeln('La somme = ', s) ;
end.
```

Pour réduire le code de ce programme, il est possible d'utiliser un sous-programme avec deux paramètres. On obtient alors le programme suivant :

```
program Calcul_Som ;
var   x, y, z, h, s : integer ;
procedure somme (a, b : integer);
begin
  writeln('Donnez la première valeur ') ;
  readln(a) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(b) ;
  s := a + b ;
  writeln('La somme = ', s) ;
end;
begin
  writeln('*** bloc 1 *** ') ;
  somme(x, y);
  writeln('*** bloc 2 ***') ;
  somme(z, h);
end.
```

On peut aussi écrire : procedure somme (a: integer; b : integer);

Les paramètres x, y et z, h sont dits effectifs, utilisés lors de l'invocation du sous-programme. Les paramètres a, b sont dits formels, utilisés lors de la déclaration du sous-programme.

Les paramètres formels et effectifs d'un même sous-programme, dans le même programme doivent être de même nombre et de mêmes types, tout en respectant l'ordre des types des paramètres.

Il existe deux types de sous-programmes : Procédures et Fonctions.

Les procédures : Une procédure est un ensemble d'instructions regroupées sous un nom, et qui réalise un traitement particulier dans un programme lorsqu'on l'appelle.

En Pascal, l'entête de la procédure contient le mot clé procedure, suivi du nom de la procédure, et éventuellement des paramètres mises entre parenthèses.

Les fonctions : Une fonction est un sous-programme qui retourne un et un seul résultat au programme appelant. Le point d'appel d'une fonction apparaît toujours dans une expression.

En Pascal, l'entête de la fonction contient le mot clé function, suivi du nom de la fonction, et éventuellement des paramètres mises entre parenthèses, et en fin le type de la fonction.
Si on reprend l'exemple Calcul_Som mais cette fois en utilisant une fonction on obtient :

```
program Calcul_Som ;
var  x, y, s : integer ;
function somme : integer;
begin
  writeln('Donnez la première valeur ') ;
  readln(x) ;
  writeln('Donnez la deuxième valeur ') ;
  readln(y) ;
  somme := x + y ;
end;
begin
  writeln('*** bloc 1 *** ') ;
  s := somme ;
  writeln('La somme = ', s) ;
  writeln('*** bloc 2 ***') ;
  s := somme ;
  writeln('La somme = ', s) ;
end.
```

Voyons un deuxième exemple montrant une fonction avec des paramètres :

```
program Comparaison ;
var  x, y : integer ;
function Sup (a, b : integer) : boolean;
begin
  if (a > b) then Sup := true
    else Sup := false;
end;
begin
  writeln('Donnez la première valeur ') ;   readln(x) ;
  writeln('Donnez la deuxième valeur ') ;   readln(y) ;
  if Sup(x,y) then  writeln(x, ' est supérieure à ', y)
    else writeln(y, ' est supérieure à ', x);
end.
```

Ce programme permet de comparer deux nombres entiers en indiquant quelle est la valeur supérieure.
En Pascal, il existe plusieurs fonctions prédéfinies ou standards permettant de faire des calculs mathématiques, telles que (ABS(), SQRT(), SQR(), EXP(), etc).

3. Les variables locales et les variables globales

Un sous-programme peut dans sa partie déclaration contenir les objets suivants : étiquettes, constantes, types, variables et sous-programmes qui sont propres à lui. Ces objets sont dits objets locaux, et souvent on appelle variables locales celles déclarées au niveau du sous-programme et qui ne peuvent être utilisées qu'au niveau de sa partie instructions. Les variables déclarées au niveau du programme principal sont des variables globales.
Exemple :

```
program IMBR1 ;
var
  A, B : real;
procedure EXT;
var
  C, D : real;
begin
… (* Les variables accessibles sont A, B, C, D*)
end ;
procedure ITN;
var
  E, F : real;
begin
… (* Les variables accessibles sont A, B, E, F*)
end ;
begin
… (* Les variables accessibles sont A, B*)
end.
-----------------------------------------------------------------
program IMBR2 ;
var
  A, B : real;
procedure EXT;
var
```

```
  C, D : real;
procedure ITN;
var
  E, F : real;
begin
… (* Les variables accessibles sont A, B, C, D, E, F*)
end ;
begin
… (* Les variables accessibles sont A, B, C, D*)
end ;
procedure EXT2;
var
  G, H : real;
begin
… (* Les variables accessibles sont A, B, G, H*)
end ;
begin
… (* Les variables accessibles sont A, B*)
end.
```

Pour savoir si un sous-programme peut faire appel à un autre sous-programme, on utilise le même principe. Ainsi, dans le programme IMBR1 précédent, la procédure ITN peut faire appel à la procédure EXT. Dans le programme IMBR2, la procédure EXT2 peut faire appel à la procédure EXT, mais elle ne peut pas appeler la procédure ITN.

Pour enlever l'ambiguïté, il est préférable d'éviter d'utiliser le même identificateur de variable dans différents niveaux. Voyons l'exemple suivant :

```
program Saisie ;
var  i : integer;
procedure S;
 var  i : integer;
 begin
   i := 5;   writeln (' i = ', i);
 end;
begin
 i := 10;
 S;
 writeln (' i = ', i);
```

```
end.
```

Le programme affiche:
i = 5
i = 10

Il est possible de substituer une fonction par une procédure en utilisant une variable globale permettant de récupérer la valeur retournée par la fonction. Soit le programme Pascal suivant :

```
program somme2 ;
var x, y : integer ;
function S (a, b : integer) : integer ;
begin
  S := a + b ;
end;
begin
  writeln ('Donnez la première valeur');
  readln(x) ;
  writeln ('Donnez la deuxième valeur');
  readln(y) ;
  writeln('Somme = ', S(x,y) ) ;
end.
```

La fonction S peut être substituée par une procédure S en ajoutant une variable globale, et cela comme suit :

```
program somme2 ;
var x, y, som : integer ;
procedure S (a, b : integer) ;
begin
  som := a + b ;
end;
begin
  writeln ('Donnez la première valeur');
  readln(x) ;
  writeln ('Donnez la deuxième valeur');
  readln(y) ;
  S(x,y) ;
  writeln('Somme = ', som) ;
end.
```

4. Le passage des paramètres

Voyons l'exemple suivant :

```
program exemple1 ;
var
 I : integer ;
procedure change1(var y : integer);
begin
 y := 1;
end;
begin
 I := 0;
 change1(I) ;
 writeln(I) ;
end.
```

y est un paramètre formel de la procédure change1. Le mot clé var est utilisé pour transmettre la valeur de y vers le programme principal. Par conséquent, la valeur affichée sera 1.
Le passage des paramètres par le mot clé var est dit "par variable" (par adresse ou par référence). Le passage des paramètres sans var est dit "par valeur". Voyons un deuxième exemple :

```
program exemple2 ;
var  I : integer ;
procedure change2(y : integer);
begin
 y := 1;
end;
begin
 I := 0;
 change2(I) ;
 writeln(I) ;
end.
```

Ce programme possède une variable globale I et une procédure change2 déclarée avec un paramètre formel y à passage par valeur. La valeur qui sera affichée en sortie est 0.
Remarques : Quand on utilise le passage par valeur, il est possible d'invoquer le sous-programme par des paramètres

effectifs exprimés sous forme d'expression, par exemple change2(I*4), change2(4).

5. La récursivité (récursion)

5.1. Définition

Une procédure ou fonction est dite récursive si elle fait référence à elle-même.

5.2. Exemple (Calcul de la factorielle)

La factorielle d'un nombre "n" se note "n!", avec n! = n*(n-1)* … *2*1. Ainsi, 5! = 5*4*3*2*1.
La fonction itérative (en utilisant une boucle) pour calculer la factorielle est donnée ci-dessous :

```
function fact (n : integer) : integer ;
var
  i, f : integer;
begin
  f := 1;
  for i :=1 to n do f := f*i ;
  fact := f;
end ;
```

La définition mathématique de la factorielle en tant que formule récurrente est la suivante : pour tout n entier, si n>0 fact(n) = n * fact(n-1), en plus fact(0) = 1. La fonction récursive fact pour le calcul de la factorielle est la suivante :

```
function fact (n : integer) : integer ;
begin
  if n=0 then fact := 1
     else fact := n*fact(n-1) ;
end ;
```

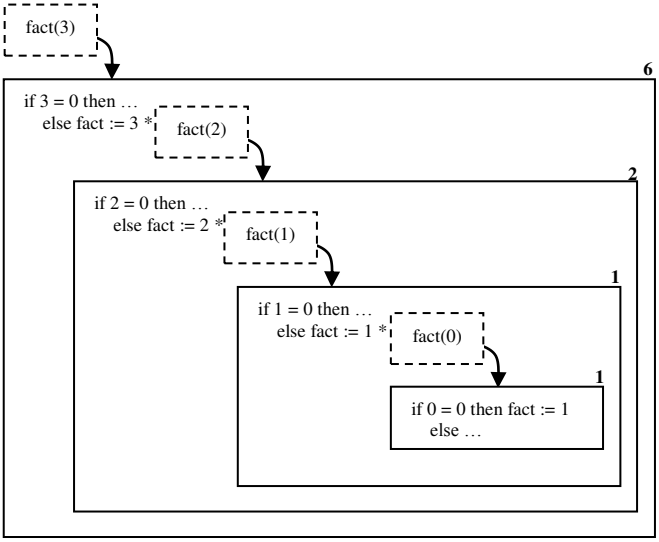
Une fonction récursive doit vérifier les deux conditions suivantes :

1. L'existence d'un critère d'arrêt. Le critère d'arrêt dans la fonction récursive fact est (n=0).

2. Les paramètres de l'appel récursif doivent converger vers le critère d'arrêt.

Le mécanisme interne et le stockage des paramètres et des variables locales pour la fonction récursive fact peuvent être représentés comme suit.

Pour n = 3.



La consommation de l'espace mémoire est élevée puisque la pile d'activation (en anglais Stack, une zone mémoire réservée à chaque programme, destinée aux variables locales des sous-programmes et aux paramètres passés par valeur aux sous-programmes) va contenir n + 1 fois le contexte de la fonction.

6. Exercices corrigés

6.1. Exercices

Exercice 1 :

Ecrire en Pascal votre propre fonction ABS(). Nommez la Absolue. La fonction Absolue doit retourner la valeur absolue d'un paramètre entier.

Exercice 2 :

1. Qu'affiche le programme Pascal suivant :

```
program Somme ;
var
  x, y, Som : integer;
procedure S(a, b : integer);
begin
  Som := a + b ;
  a := a + b;
  b := 2 ;
end;
begin
  x := 2 ;  y := 9 ;
  S(x,y);
  writeln(x, ' + ', y ,' = ', Som);
end.
```

2. Qu'affiche programme précédent, mais cette fois-ci si on remplace S(a, b : integer) par S(a : integer ; var b : integer), par S(var a : integer ; b : integer), et enfin par S(var a, b : integer).

Exercice 3 :

Ecrire en Pascal la fonction itérative, ensuite la fonction récursive permettant de calculer le PGCD (le plus grand diviseur commun) de deux entiers.

Exercice 4 :

Soit la procédure suivante contenant une boucle simple :

```
procedure compter ;
var
  i : integer ;
begin
  for  i := 1 to 10 do writeln(i) ;
end ;
```

Exprimez cette procédure par une autre récursive, faisant le même travail :

6.2. Corrigés

Solution 1 :

```
program Val_ABS ;
var  x : integer;
function Absolue(a : integer) : integer;
begin
 if (a >= 0) then Absolue := a
   else Absolue := -a ;
end;
begin
 writeln('Donnez une valeur ');
 readln(x);
 writeln('La valeur absolue de x est = ', Absolue(x));
end.
```

Solution 2 :

1. Le programme affiche : 2 + 9 = 11.
2. Le programme affiche :
   - Pour S(a : integer ; var b : integer) : 2 + 2 = 11.
   - Pour S(var a : integer ; b : integer) : 11 + 9 = 11.
   - Pour S(var a, b : integer) : 11 + 2 = 11.

Solution 3 :

La fonction itérative de calcul du PGCD est la suivante :

```
function PGCD(m, n : integer) : integer;
var
 r : integer ;
begin
 while NOT (m mod n = 0) do  begin
    r := m mod n ;
    m := n ;
    n := r ;
 end;
 PGCD := n ;
end ;
```

La version récursive de cette fonction est la suivante :

```
function PGCD(m, n : integer) : integer;
begin
```

```
  if m mod n = 0 then PGCD := n
     else PGCD := PGCD(n, m mod n);
end ;
```

Solution 4 :

```
procedure compter(i : integer) ;
begin
  if (i = 10) then writeln(i)
  else begin
    writeln(i) ;
    compter(i + 1) ;
  end ;
end ;
```

La procédure récursive compter sera invoquée dans le programme principal par compter(1).

Chapitre 7 : Les types définis par l'utilisateur

1. Introduction

Les types qu'on a vu jusqu'à maintenant (entier, réel, caractère, booléen, tableau et chaîne de caractères) sont prédéfinis, c.-à-d. qu'ils existent dans les langages de programmation. Ces langages offrent à l'utilisateur la possibilité de définir de nouveaux types de données. En langage Pascal, les types définis sont introduits par le mot clé TYPE dans la partie déclaration du programme.

2. Types simples définis par l'utilisateur

2.1. Le type énuméré

Le type énuméré permet de citer explicitement les valeurs que peut prendre une variable. Ces valeurs sont spécifiées par des identificateurs.
En Pascal, le type énuméré peut avoir différentes représentations en mémoire. Ça dépend du compilateur. Par défaut, Turbo Pascal utilise une représentation 1 octet, permettant ainsi 256 valeurs différentes. Free Pascal utilise par défaut 4 octets pour le stockage des valeurs énumérées. Cette espace d'encodage des valeurs énumérées peut être personnalisé grâce à la directive {$Z} : {$Z1} pour un octet, {$Z2} pour 2 octets et {$Z4} pour 4 octets.
Ceci ressemble à la représentation des valeurs booléennes sur 1 octet : false (0) et true(1).
Exemple :

```
Type couleur= (vert, noir, blanc, jaune) ;
var c,d : couleur ;
```

Dans cet exemple, nous avons défini le type couleur. Ensuite, nous avons déclaré deux variables c et d de ce type. Une variable de type couleur ne peut prendre qu'une des valeurs : vert, noir, blanc, jaune. On peut mettre directement var c, d : (vert, noir, blanc, jaune) ;

Une variable de type énuméré ne peut être ni lue ni écrite par les instructions read ou write, mais on peut manipuler cette variable de différentes manières :

- L'affectation. Par exemple, c := vert ; d := blanc ;.
- Les valeurs d'un type énuméré sont ordonnées. Ce qui permet aussi d'utiliser les fonctions prédéfinies PRED, SUCC et ORD. PRED(noir) retourne vert, SUCC(noir) retourne blanc et ORD(noir) retourne 1.
- La comparaison selon l'ordre de déclaration. Par exemple, l'expression logique (noir<vert) donne false.
- On peut utiliser une variable de type énuméré
  dans une boucle. Par exemple, for c := vert to jaune do … ;,
  ou dans une instruction de sélection. Par exemple, case c of
  vert : … ;
  noir : … ;
  blanc :… ;
  jaune : … ;
  end ;

2.2. Le type intervalle

Le type intervalle est utilisé pour déclarer une variable qui prend ses valeurs dans un intervalle limité par une borne inférieure et une borne supérieure :

Exemple :

```
Type
  niveau = 1..10 ;
  couleur= (vert, noir, blanc, jaune) ;
var
  x : niveau ;
  y : 'a'..'f' ;
  cl : vert..blanc ;
…
```

La variable x peut prendre une valeur de 1 à 10, la variable y peut prendre un caractère de 'a' à 'f', et la variable cl peut prendre comme valeur vert, noir ou blanc.

Les deux bornes doivent être de même type simple, non réel, avec (la borne inférieure <= la borne supérieure). Ces bornes font partie de l'intervalle.
Les opérations applicables au type intervalle sont celles applicables à des variables de même type que les bornes.
En plus, les instructions suivantes :

- Inc() (incrémentation de la variable passée en paramètre),
- Dec() (décrémentation de la variable passée en paramètre),
- Succ() (renvoie le successeur de la variable passée en paramètre),
- Pred() (renvoie le prédécesseur de la variable passée en paramètre),
- Ord() (renvoie l'index de la variable dans l'intervalle auquel elle appartient)

s'appliquent au type intervalle, qu'il soit de type entier, caractère ou énuméré.
Pour le type intervalle (subrange en anglais), Pascal réserve l'espace nécessaire pour la borne inférieure et la borne supérieure. Par exemple, pour l'intervalle 1..255, Pascal réserve 1 octet. Pour l'intervalle 20..270, Pascal réserve 2 octets.
Soit le programme Pascal (écrit sur un compilateur Free Pascal) :

```
Program Exemple;
type
  T1  = (Rover, Jaguar, Honda);
  T2  = Rover..Honda;
  T3 = 1..255;
  T4 = 10..270;
  T5 = 'a'..'d';
BEGIN
writeln('T2  size = ', SizeOf(T2));  { SizeOf retourne la taille d'une
variable ou un type}
writeln('T3  size = ', SizeOf(T3));
writeln('T4  size = ', SizeOf(T4));
writeln('T5  size = ', SizeOf(T5));
readln
END.
```

Le programme affiche :

T2 size = 4
T3 size = 1
T4 size = 2
T5 size = 1

3. Types structurés définis par l'utilisateur

3.1. Le type ensemble

Un ensemble est une collection d'objets de même type.
Exemple :

```
Type
  couleur = (vert, noir, blanc) ;
  cl = set of couleur ;
var
  cl1, cl2 : cl ;
begin
cl1 := [vert, blanc] ;
cl2 := [vert, noir] ;
…
```

Dans cet exemple, nous avons défini un type énuméré couleur et un type ensemble cl associé au type couleur. On dit que couleur est le type de base de cl. Ensuite, nous avons déclaré cl1 et cl2 deux variables de type ensemble cl. cl1 et cl2 peuvent prendre comme valeurs des sous ensembles de cl. On peut mettre directement var cl1, cl2 : set of (vert, noir, blanc);
Un type de base peut être un type énuméré, un type intervalle d'entier ou de caractère, le type caractère, ou le type booléen. Par exemple, var chiffre : set of 1..10.
On peut affecter à une variable de type ensemble, un ensemble de valeurs mises entre deux crochés [et] en utilisant le symbole d'affectation :=, comme c'est le cas dans l'exemple précédent pour les variables cl1 et cl2. L'écriture cl1 := [] indique que la variable cl1 reçoit l'ensemble vide. L'écriture cl1 := [vert..blanc] indique que la variable cl1 reçoit l'ensemble des valeurs vert, noir et blanc.
On peut effectuer plusieurs opérations sur les ensembles : L'union (cl1+cl2 donne [vert, noir, blanc]) ; la différence ou le

complément (cl1-cl2 donne [blanc]) ; l'intersection (cl1*cl2 donne [vert]) ; les tests booléens sont aussi possibles (=, <>, <=, >=). Par exemple, l'expression logique (cl1=cl2) retourne la valeur FALSE. On peut également tester l'appartenance d'un élément à un ensemble par l'opérateur IN. Par exemple, l'expression logique (vert IN cl1) retourne la valeur TRUE.
Tant que le numéro d'ordre du dernier élément (la valeur maximal) du type de base ne dépasse pas la valeur 31, le compilateur réserve un espace de 4 octets pour une variable de type ensemble, sinon le compilateur réserve 32 octets. Notons que la valeur minimale du type de base ne doit pas être inferieure à 0.
Soit le programme Pascal suivant (écrit sur un compilateur Free Pascal) :

```
Program Exemple;
type
  T1  = (Rover, Jaguar, Honda);
  T2  = set of Rover..honda;
  T3 = set of 0..31;
  T4 = set of 0..32;
  T5 = set of 10..40;
  T6 = set of 'a'..'d';
BEGIN
writeln('T2  size = ', SizeOf(T2));
writeln('T3  size = ', SizeOf(T3));
writeln('T4  size = ', SizeOf(T4));
writeln('T5  size = ', SizeOf(T5));
writeln('T6  size = ', SizeOf(T6));
END.
```

Le programme affiche :

```
T2  size = 4
T3  size = 4
T4  size = 32
T5  size = 32
T6  size = 32
```

3.2. Le type enregistrement

La notion d'enregistrement (structure) permet de regrouper un ensemble de données de différents types dans un même objet.
Format général :

```
Nom_enreg = enregistrement
  Nom_champ1 : type1 ;
  Nom_champ2 : type2 ;
  ….
Fin_enreg ;
```

Avec Nom_enreg est le nom de l'enregistrement défini par l'utilisateur. Nom_champ1, Nom_ champ2,… sont des variables membres (champs) de l'enregistrement.
On peut déclarer des variables de ce type comme suit : Nom_var : Nom_enreg. On accède à une information en précisant le nom de la variable de type enregistrement, suivi d'une variable membre, généralement séparés par un point comme suit : Nom_var.Nom_ champ1, Nom_var.Nom_ champ2, …

Exemple :
Soit l'exemple Pascal suivant :

```
program personnel ;
type
  personne = record
     nom : string[20] ;
     sit_familiale : (marie, celibataire, divorce) ;
     telephone : string[10] ;
  end ;
var
  employe, X : personne ;
begin
  employe.nom := 'Ali' ;   employe.sit_familiale := celibataire ;
  readln(employe.telephone) ;
  writeln(employe.nom,'  ', employe.telephone) ;
  X := employe ;
end.
```

Dans cet exemple, nous avons défini un type enregistrement nommé personne, possédant les champs nom de type chaîne de caractères, sit_familiale de type énuméré, et telephone de type chaîne de caractères. Ensuite, nous avons déclaré deux variables X

et employe de type enregistrement déjà défini. L'espace mémoire réservé pour chacune des deux variables est égal la somme des espaces réservés pour ses champs.
Les opérations de lecture, écriture, comparaison, etc. ne sont pas applicables directement sur un type enregistrement. Ainsi, les instructions read(employe); write(employe) ; et la comparaison (X<employe) ne sont pas acceptées dans le programme précédent. La seule instruction possible pour manipuler directement une variable de type enregistrement sans passer par ses champs est l'affectation (X := employe). Pour d'autres opérations, la variable employe peut être manipulée champ par champ en regroupant le nom de la variable avec le nom d'un seul champ, comme par exemple employe.nom. La composante obtenue se comporte alors exactement comme toute autre variable de même type que celui du champ correspondant. Maintenant, on peut effectuer sur cette composante une affectation, une lecture, une écriture, une comparaison, etc.
L'exemple décrit au-dessus peut être simplifié en utilisant la structure with…do, et cela comme suit :

```
program personnel;
type
  personne = record
     nom : string[20] ;
     sit_familiale : (marie, celibataire, divorce) ;
     telephone : string[10] ;
  end ;
var
  employe, X : personne ;
begin
  with employe do begin
    nom := 'Ali' ;
    sit_familiale := celibataire ;
    readln(telephone) ;
    writeln(nom,'  ', telephone) ;
  end;
    X := employe;
end.
```

Les variables X et employe peuvent être déclarées directement comme suit :

```
var
 X, employe :  record
    nom : string[20] ;
    sit_familiale : (marie, celibataire, divorce) ;
    telephone : string[10] ;
 end ;
```

Il est aussi possible de déclarer un tableau dont les éléments sont de type enregistrement. Par exemple, var T : array [1..10] of personne ; L'accès au premier élément du tableau s'effectuera comme suit : T[1].nom :='Ali' ; T[1].sit_familiale := celibataire ;…

4. Exercices corrigés

4.1. Exercices

Exercice 1 :
Ecrire un programme Pascal permettant de définir un type énuméré, ayant comme valeurs les couleurs d'un feu pour la gestion de la circulation dans un carrefour. Ensuite, lire une variable de type intervalle et afficher à quelle couleur la valeur de la variable correspond.

Exercice 2 :
Ecrire un programme Pascal permettant de déterminer le nombre de voyelles dans un mot. Utilisez l'ensemble VL = {a, e, u, o, i}.

Exercice 3 :
Présentez en Pascal les types enregistrement suivants :

- Un nombre complexe est défini par une partie réelle et une partie imaginaire.
- Une date est composée d'un numéro de jour (1..31), d'un numéro de mois (1..12) et d'une année.
- Un stage peut être défini par un intitulé (une chaîne de caractères), une date de début et une date de fin (deux dates), un nombre de places (entier).

- Une identité décrivant le nom, le prénom et la date de naissance.
- Une fiche bibliographique est définie par le titre du livre (chaîne), les auteurs (10 auteurs maximum, chacun est défini par nom, prénom et date de naissance), la date de parution, l'éditeur (nom, prénom et date de naissance), le numéro ISBN (chaîne).

Comment peut-on affecter une valeur à :

- Une partie réelle d'une variable X de type complexe.
- Jour d'une variable Y de type date.
- Mois de la date de début d'une variable Z de type stage.
- Année de la date de naissance d'une variable G de type identité.
- Jour de la date de naissance du premier auteur d'une variable H de type fiche bibliographique.

Exercice 4 :

Un type enregistrement dit pere possède les champs Nom de type chaîne de caractères, Date_nais de type chaîne de caractères, Nbr_enf de type entier, et Liste_enf de type chaîne de caractères. Ecrire un programme Pascal permettant de déclarer un tableau de cinq éléments de type enregistrement pere, de lire ses éléments, de les afficher, et enfin d'afficher le nombre total des enfants.

4.2. Corrigés

Solution 1 :

```
program Circulation ;
type
  feu = (vert, orange, rouge) ;
  Inter_feu = 0..2;
var
  num : Inter_feu ;
begin
  writeln('Entrez un numéro entre 0 et 2') ;
  readln(num) ;
  case num of
     ord(vert) : writeln('feu vert');
     ord(orange) : writeln('feu orange');
     ord(rouge) : writeln('feu rouge');
```

```
  end;
end.
```

Solution 2 :

```
program voyelles ;
var
  VL : set of char ;
  mot : string[20] ;
  x, t, nb : integer ;
begin
  VL := ['a', 'A', 'e','E', 'i','I', 'o','O', 'u','U'] ;
  writeln('Entrez un mot de 20 caractères maximum') ;
  readln(mot) ;
  t := length(mot) ;
  nb := 0 ;
  for x := 1 to t do if (mot[x] IN VL) then nb := nb + 1;
  writeln('Nombre de voyelles dans le mot est : ', nb) ;
end.
```

Solution 3 :

```
program solution ;
{déclaration des types}
type
  nombre_complexe = record
      reel : real ;
      imaginaire : real ;
  end;
  date = record
      jour : 1..31;
      mois : 1..12;
      annee : integer;
  end;
stage = record
      intitule : string[50];
      date_debut, date_fin : date;
      nbr_place : integer;
  end;
  identite = record
```

```
    nom : string[15];
    prenom : string[15];
    date_nais : date;
  end;
  fiche_biblio = record
    titre : string[50] ;
    auteurs : array [1..10] of identite ;
    date_parution : date ;
    editeur : identite;
    num_ISBN : string[10];
  end ;

{déclaration des variables}
var
  X : nombre_complexe ;
  Y : date ;
  Z : stage ;
  G : identite ;
  H : fiche_biblio ;

begin {Accès aux champs des variables}
  X.reel := … ;
  Y.jour := … ;
  Z.date_debut.mois := …;
  G.date_nais.annee := … ;
  H.auteurs[1].date_nais.jour := … ;
end.
```

Solution 4 :

```
program enregistrement ;
type
  pere = record
    nom : string [20];
    date_nais : string[10];
    nbr_enf : integer;
    liste_enf : string[100];
  end;
var
  peres : array [1..5] of pere ;
```

```
 j, nbr : integer ;
begin
 nbr := 0;
 for j := 1 to 5 do begin
  writeln('Donnez le nom du père num ', j) ;   readln(peres[j].nom) ;
   writeln('Donnez la date de naissance du père num ', j) ;
readln(peres[j].date_nais) ;
   writeln('Donnez le nombre d''enfants du père num ', j) ;
readln(peres[j].nbr_enf) ;
   nbr :=nbr + peres[j].nbr_enf;
   writeln('Donnez la liste des enfants : ') ;
readln(peres[j].liste_enf) ;
 end ;
 for j := 1 to 5 do begin
   writeln('Voici les informations concernant le père num ', j) ;
   writeln('Nom : ', peres[j].nom) ;
   writeln('La date de naissance : ', peres[j].date_nais) ;
   writeln('Le nombre d''enfants : ', peres[j].nbr_enf) ;
   writeln('La liste des enfants :', peres[j].liste_enf) ;
 end ;
 writeln('Le nombre total des enfants = ', nbr);
end.
```

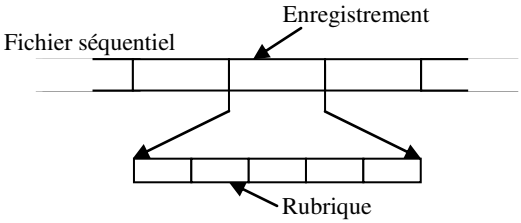
Chapitre 8 : Les fichiers

1. Introduction

Toutes les structures de données que nous avons utilisé jusqu'à maintenant permettent le stockage temporaire (pendant l'exécution du programme) des variables dans la RAM. Les fichiers permettent le stockage permanent des données.

2. Les fichiers

Un fichier est une structure de données permettant le stockage des informations sur un support non volatil (disque dur, flash disk, etc.) pour une durée indéterminée.
Le fichier peut être considéré comme un tableau d'enregistrements stocké sur un disque. Chaque enregistrement contient une collection d'unités logiques d'informations, encore appelées rubriques ou champs.



Quand t-il s'agit d'un ensemble d'enregistrements, on parle de fichier typé, mais il existe d'autres types de fichiers, à savoir les fichiers non typés et les fichiers textes.

3. Types de fichiers

Il existe donc trois types de fichiers : typés, non typés et les fichiers textes :

3.1. Les fichiers textes

Un fichier texte est un fichier séquentiel qui contient une suite de caractères regroupés en lignes. Chaque ligne se termine par un

indicateur de fin de ligne : soit le retour au chariot (Cr) dont le code ASCII est 13, soit le saut de ligne (Lf) dont le code ASCII est 10.
Format général : Variables f : Texte ;
En Pascal : var f : text ;

3.2. Les fichiers typés

Un fichier typé est constitué d'un ensemble d'enregistrements.
Format général : Variables f : Fichier de <Type> ;
En Pascal : var f : file of <Type> ;
Exemple Pascal :

```
type
  TPers = record
    Code : integer ;
    Nom, Prenom : string ;
end ;
var
  personne : file of TPers ;
  f : file of integer ;
…
```

La forme "fichier typé" est la plus favorisée pour la manipulation des fichiers.

3.3. Les fichiers non typés

Un fichier non typé est un fichier dont le contenu est inconnu. Il peut être par exemple un fichier son, une image, un programme exécutable, etc.
Format général : Variables f : Fichier ;
En Pascal : var f : file ;

4. Les méthodes d'accès aux fichiers

Un même fichier est accessible en lecture, écriture ou recherche par des méthodes différentes :

- L'accès séquentiel : signifie que les enregistrements du fichier sont traités en séquence.

- L'accès direct : signifie qu'on peut accéder directement à l'enregistrement choisi en précisant le numéro de cet enregistrement. Chaque numéro représente l'emplacement de l'enregistrement relativement au début du fichier.
- L'accès indexé : signifie que chaque enregistrement du fichier est accessible via une clé d'accès. Une clé est une rubrique du fichier qui identifie de façon unique un enregistrement. La clé est associée à un numéro d'enregistrement qui renvoie l'enregistrement correspondant.

Remarque :
En fait, tout fichier peut être utilisé avec l'un ou l'autre des trois types d'accès. C'est donc dans le programme qu'on choisit le type d'accès souhaité.

5. Manipulation des fichiers

Plusieurs opérations peuvent être alors utilisées pour la manipulation des fichiers, parmi lesquelles on cite :

5.1. Assignation

L'opération d'assignation consiste à associer un nom logique du fichier (spécifié dans la partie déclaration du programme) avec le nom physique du fichier (chemin du fichier sur le disque).
Format général : Assigner (<nom logique>, <nom physique>) ;
En Pascal : assign(<nom logique>, <nom physique>) ;

Remarque : A la fin du chemin de fichier on trouve le nom de fichier, éventuellement suivi d'un point et d'une extension d'au plus 3 caractères.

Exemple Pascal :

```
type
  TPers = record
    Code : integer ;
    Nom, Prenom : string ;
  end ;
var
  personne : file of TPers ;
  f : text ;
```

```
begin
 assign(personne, 'c:\application\pr.dat') ;
 assign(f, 'c:\essai.txt') ;
…
```

5.2. Ouverture

Il est possible d'ouvrir un fichier en lecture sans écraser son contenu et positionner le pointeur au début du fichier. Format géneral : OuvrirL(<nom logique>) ;
En Pascal : reset(<nom logique>) ;
Il est aussi possible d'ouvrir un fichier en écriture. Si le fichier existe déjà, il sera écrasé et son contenu sera perdu. Format général : OuvrirE(<nom logique>) ;
En Pascal : rewrite(<nom logique>) ;

5.3. Lecture et écriture

La lecture consiste à ranger le contenu du composant (enregistrement) courant du fichier dans une variable de même type que les composants du fichier.
Format général : Lire(<nom logique>, <nom_variable>) ;
En Pascal : read(<nom logique>, <nom_variable>) ;
L'écriture permet d'écrire dans le fichier à la position courante, le contenu d'une variable de même type que les composants du fichier.
Format général : Ecrire(<nom logique>, <nom_variable>) ;
En Pascal : write(<nom logique>, <nom_variable>) ;

5.4. Accès aux enregistrements

Un indicateur booléen indique la fin du fichier après un parcours séquentiel.
Format général : Fin_Fichier(<nom logique>).
En Pascal : eof(<nom logique>).
La fonction eof détecte la fin du fichier et retourne la valeur true ou false.
Il est aussi possible d'accéder directement à un enregistrement par son numéro d'ordre (accès direct).
Format général : Rechercher(<nom logique>, <numéro>) ;

En Pascal : seek(<nom logique>, <numéro>) ;
Remarque : La procédure seek permet d'accéder directement à un enregistrement par son numéro. Ce numéro vaut 0 pour le premier élément. seek n'est pas utilisable pour les fichiers Textes.

5.5. Fermeture du fichier

Pour finir, une fois qu'on en a terminé avec un fichier en lecture ou en écriture, il ne faut pas oublier de le fermer.
Format général : Fermer(<nom logique>) ;
En Pascal : close(<nom logique>) ;

6. Exercices corrigés

6.1. Exercices

Exercice 1 :
Ecrire un programme Pascal permettant de créer un fichier typé contenant les enregistrements des employés d'une entreprise. Chaque enregistrement contient : le code, le nom et le prénom de l'employé. L'organisation des enregistrements est séquentielle. Le programme doit permettre aussi l'affichage à l'écran de toutes les informations contenues dans le fichier.

Exercice 2 :
Ecrire un programme Pascal permettant de lire le contenu d'un fichier texte nommé texte.txt.

6.2. Corrigés

Solution 1 :

```
program fichier_seq ;
type
  employe = record
    code : integer ;
    nom, prenom : string[20] ;
    end;
var
  emp : file of employe ;
  e : employe ;
```

```
  reponse : char ;
begin
{ Création du fichier emp }
  assign(emp, 'c:\fichemp.ent');
  rewrite(emp);
  repeat
    writeln('Donnez le code de l''employé ');
    readln(e.code);
    writeln('Donnez le nom de l''employé ');
    readln(e.nom);
    writeln('Donnez le prénom de l''employé ');
    readln(e.prenom);
{ Ecriture sur disque dans le fichier emp }
    write(emp, e);
    writeln('Autre saisie ? o/n ');
    readln(reponse);
  until reponse = 'n';
  close(emp);
{ Consultation séquentielle du fichier emp }
  assign(emp, 'c:\fichemp.ent');
  reset(emp);
  while not eof(emp) do begin
    read(emp, e);
    writeln(' Code : ', e.code, ' Nom : ', e.nom, ' Prénom : ',
e.prenom);
  end;
  close(emp);
end.
```

Solution 2 :

```
program fichier_seq ;
var
  fichier : text ;
  ligne : string[80] ;
begin
  assign(fichier, 'essai.txt');
  {$I-}
  reset(fichier);
  if (ioresult = 0) then begin
```

```
   while not eof(fichier) do begin
     readln(fichier, ligne);
     writeln(ligne);
   end;
   close(fichier);
   end
  else writeln('fichier inexistant ');
end.
```

Conclusion générale

Ce livre constitue un premier volume introduisant les notions de base de l'algorithmique et les structures de données statiques, et initiant à la programmation en Pascal. Dans ce document nous avons apporté une description de la méthode de résolution d'un problème, allant de l'analyse, jusqu'à l'écriture du programme. Nous avons aussi présenté comment écrire un algorithme séquentiel avec des opérations de base et des structures de données (types) simples. C'est à ce niveau qu'une section importante a été consacrée à la présentation du langage Pascal. Nous somme passés ensuite aux structures de contrôle, notamment les structures conditionnelles (simples, composées et de choix multiples) et les boucles (Tant que, Répéter et Pour). Puis, on est revenu aux types structurés, à savoir les tableaux et les chaînes de caractères. Les sous-programmes (fonctions et procédures) visant à organiser et à réduire un programme ont été aussi introduits. Les deux derniers chapitres ont été consacrés respectivement aux types définis par l'utilisateur (cette partie a montré les différentes possibilités du langage Pascal offertes aux utilisateurs pour définir leurs propres types), et au type fichier permettant le stockage permanent des données. Ce cours a été enrichi par des exercices corrigés.

Le prochain volume sera consacré à des structures de données dynamiques, dites aussi récursives, et à des notions avancées d'algorithmique. Il comportera les listes chaînées, les arbres, les graphes, les algorithmes de tri et la complexité des algorithmes.

Index