Algorithmique : Cours et Exercices en Programmation Pascal

Book · April 2018

CITATIONS
0

READS
26,160

5 authors, including:

Rabia Amour
University of Science and Technology Houari Boumediene
24 PUBLICATIONS **496** CITATIONS
SEE PROFILE

Fermous Rachid
Faculty of Sciences and Technology, Djilali Bounaama University
6 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Ait Ouarabi Mohand
University of Science and Technology Houari Boumediene
9 PUBLICATIONS **35** CITATIONS
SEE PROFILE

Moufida Benzekka
École Normale Supérieure de Kouba
11 PUBLICATIONS **53** CITATIONS
SEE PROFILE

Université des Sciences et de la Technologie Houari Boumediene
Les Cours de la Faculté de Physique

ALGORITHMIQUE

Cours, Exercices et Programmation Pascal
Première Année Universitaire
Domaines : SM, ST *et* MI

Rédigé par :

M. Rabia AMOUR	M.C.A	USTHB
M. Rachid FERMOUS	M.C.B	UDBKM
M. Mohand AIT OUARABI	M.A.A	USTHB
Mme. Moufida BENZEKKA	M.C.B	ENS KOUBA
M. Smain YOUNSI	M.C.B	ESSA ALGER

2017-2018

Préface

Ce polycopié de cours et de travaux pratiques regroupe un certain nombre de notions de base sur l'algorithmique. Ce cours d'algorithmique, destiné particulièrement aux étudiants de première année Sciences de la Matière (SM), fut assuré pendant pratiquement une dizaine d'années par le Laboratoire de Physique Théorique, affilié à la Faculté de Physique (USTHB).
Le présent polycopié est scindé en deux grandes parties. La première partie illustre l'essentiel du cours d'algorithmique, à savoir, le langage algorithmique qui couvre toutes les instructions de base constituant un algorithme, les structures conditionnelles et itératives, les tableaux à une et à deux dimensions et enfin les fonctions et procédures.
Ce cours est rédigé dans un style simple, clair et riche en terme d'exemples, que nous souhaitons compréhensible. La deuxième partie de ce fascicule est consacrée aux travaux pratiques. Ces derniers englobent l'ensemble des exercices traités, en séance de TP, par les étudiants de première année SM durant l'année universitaire 2012/2013. Ces séries d'exercices couvrent toutes les sections présentées en cours.
Pour ce faire, nous avons fait appel au langage Pascal. Ce dernier est un langage facile à manipuler par l'étudiant débutant et possède une structure très proche de celle du langage naturel.
La première série de TP présente un ensemble d'exercices de base permettant à l'étudiant de se familiariser, tout d'abord avec les différentes instructions de base de programmation Pascal et de traiter quelques problèmes en utilisant les structures conditionnelles.
Les deux dernières séries de TP incluent des problèmes plus complexes, nécessitant l'exploitation des structures itératives (boucles: Pour, Tant que et Répéter) et les tableaux, à savoir les vecteurs et les matrices.
Nous tenons à remercier spécialement:
Madame le Professeur F. CHAFA, Ex-Doyenne de la Faculté Physique, pour la confiance qu'elle nous a accordée afin de gérer et enseigner le module d'algorithmique.
Madame le Professeur Lynda AMIROUCHE, Ex-coordinatrice du module, avec laquelle nous avons entrepris l'enseignement de l'algorithmique.
M. BOUKHALFA Sofiane pour sa précieuse aide surtout dans la partie programmation.
Nos remerciements vont également à tous les enseignants des facultés de Physique et d'informatique, ayant assuré le cours, TD et le TP du module algorithmique durant l'année universitaire 2012/2013.

Dédicaces

Ce polycopié est dédié à la mémoire du défunt Monsieur le Professeur Mouloud Tribeche.

Table des Matières

Introduction

3.1 Généralités

Depuis que l'homme a ressenti le besoin de différencier et de dénombrer, soit les êtres de son espèces, soit des objets et des animaux qu'il chassait ou qui représentaient un danger sur sa vie, il a créé des outils qui l'aider à faire les calculs. Depuis des milliers d'années différents outils ont été utilisé, le plus anciens est probablement l'os. La main fût utilisée par la plupart des sociétés ; elle sert à nos jours. Succédèrent une variété d'objets et de techniques tels que les entailles sur le bois, les entassements de cailloux et d'autres objets. Des outils plus complexes furent inventés au fur et à mesure, tel l'abaque, ensuite le boulier (utilisé à nos jours), la pascaline (Le mot calcul vient du Latin Calculus, qui signifie caillou), etc.
Avant d'entamer les définitions de l'informatique, nous allons faire un petit rappel chronologique non exhaustif d'un certain nombre d'évènements majeurs ayant contribué au développement ou carrément à la fondation de l'informatique, telle que nous la connaissons aujourd'hui.
Du caillou au Bit ; des millénaires d'interrogations, de découvertes et d'apprentissage.
Ci-dessous quelques évènements marquants sur l'apparition et le développement de l'informatique et de l'ordinateur :

- 780-850: Abu Abdullah Muhammad bin Musa Al-Khawarizmi[1].
- 1642 : l'invention de la Pascaline par Blaise Pascal.
- 1821 : invention de la machine à différences par Charles Babbage.
- 1840 : Ada Lovelace (mathématicienne) nomme le processus logique d'exécution d'un programme : Algorithme, en l'honneur à Al-Khawarizmi.
- 1854: Boole publie "An Investigation Into the Laws of Thought", ouvrage fondateur de l'algèbre de Boole. 1946 : premier ordinateur l'ENIAC.
- 1947 : apparition du transistor (laboratoires Bell Téléphone)

1. 825 ”livre sur le calcul avec les nombres hindous”, 830 ”livre sur les mathématiques”

- 1950 : invention de l'assembleur à l'université de Cambridge.
- 1957 : lancement du premier compilateur FORTRAN, par John Backus et son équipe (IBM).
- 1958 : Jack St. Clair Kilby invente le circuit intégré[2].
- 1964 : lancement de la série 360 d'ordinateurs d'IBM ; ordinateurs compatibles entres eux.
- 1968 : apparition du Langage PASCAL, créé par Niklaus Wirth.
- 1971 : commercialisation du Intel 4004 ; premier microprocesseur Intel (4 Bits, 108 KHz, 60000 instructions par seconde, composé de 2300 transistors en technologie de 10 microns).
- 1973 : lancement du mini-ordinateur multitâches (temps-réel) et multi-utilisateur, le HP 3000, par Hewlett-Packard.
- 1981 à nos jours : c'est l'âge des micro-ordinateurs et des supercalculateurs.

3.2 Informatique

Le mot Informatique est un Néologisme construit à partir des mots information et automatique par P. Dreyfus en 1962. C'est la discipline qui s'occupe du traitement automatique de l'information.
D'après le petit Robert de langue Française : informatique est un nom féminin qui désigne la science de l'information ; *" c'est l'ensemble des technique de la collecte, du tri, de la mise en mémoire, de la transmission et de l'utilisation des informations traitées automatiquement à l'aide de programmes (logiciels) mis en oeuvre sur ordinateur "*.
L'informatique se scinde en deux parties : le matériel et le logiciel, la partie logiciel, communément connue " Software ".

`Hardware (le Matériel)`

C'est un emprunt de l'anglais ; il désigne toute la partie électronique et les composants qui constituent un ordinateur.

3.3 Ordinateur

Nom donné en 1955 aux premières machines d'IBM par J.Perret. Toutes les autres langues utilisent le terme "calculateur" et non "ordinateur". C'est une machine à traiter électroniquement

2. Le premier circuit intégré a été produit en septembre 1958, mais les ordinateurs l'utilisant ne sont apparus qu'en 1963.

les données. Il permet l'automatisation de tâches et de calculs et leur exécution.
Le micro-ordinateur standard, commercialisé de nos jours, est composé d'un ensemble d'éléments:

1. Périphériques d'entrée : clavier, souris (caméra, scanner, lecteur de code barre, etc.)
2. Périphériques de sortie : écran (imprimante) Périphériques Entrée/Sortie : lecteur/graveur de disque, lecteur de disquette, etc.
3. Unité Centrale : elle désigne le processeur ; dans le commerce, elle désigne le boitier qui intègre la carte mère contenant le microprocesseur, la mémoire centrale, le disque dur et d'autres composants.

Figure 3.1: Ordinateur.

L'ordinateur moderne est conçu selon l'architecture de John Von Neumann[3], c'est-à-dire composé de quatre parties distinctes: l'unité arithmétique et logique (UAL), l'unité de contrôle (UC), la mémoire centrale (MC) et les entrées-sorties (E/S).

3. Le terme " architecture de Von Neumann " est néanmoins considéré comme injuste vis-à-vis des collaborateurs de John Von Neumann, notamment John William Mauchly et John Eckert qui ont utilisé ce concept pendant leurs travaux sur l'ENIAC et il est donc maintenant plutôt proscrit.

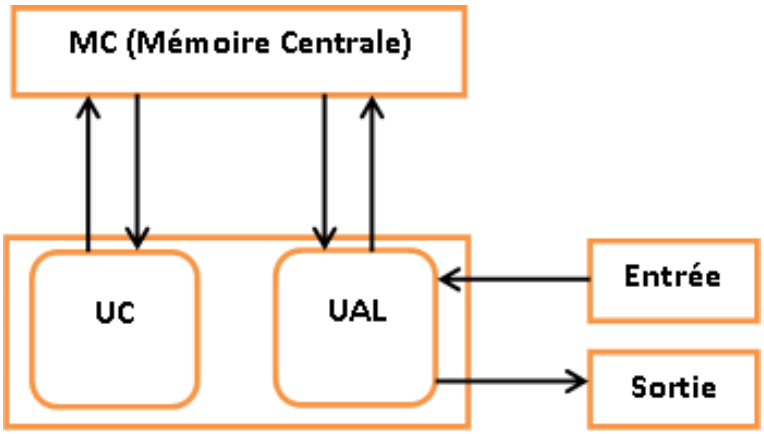


Figure 3.2: Schéma d'une mémoire centrale.

3.3.1 Unité Centrale

Appelée aussi processeur, a pour rôle d'exécuter les programmes. Elle est composée de l'unité arithmétique et logique (UAL) et de l'unité de contrôle.

- L'unité arithmétique et logique réalise une opération élémentaire (addition, soustraction, multiplication, etc.) du processeur à chaque top d'horloge.
- L'unité de commande contrôle les opérations sur la mémoire (lecture/écriture) et les opérations à réaliser par l'UAL selon l'instruction en cours d'exécution.

Pour pouvoir effectuer les opérations sur des données et exécuter des programmes l'UC doit disposer d'un espace de travail. Cette espace de travail s'appelle la mémoire centrale.

3.3.2 Microprocesseur (processeur)

Inventé en 1971 par Ted Hoff à Santa Clara dans la Silicon Valley pour le compte d'Intel, contient des circuits électroniques intégrés imprimés sur une seule pastille de silicium, communément appelé "puce électronique".
De nos jours le nouveau microprocesseur d'intel, l'Ivy Bridge (Fig. 3.3) contient 1,4 milliard de transistors concentrés sur un die d'une surface de 160 mm^2.

Figure 3.3: Micro-processeur.

3.3.3 Mémoire centrale (MC)

Elle représente l'espace de travail de l'ordinateur. C'est l'organe principal de stockage des informations utilisées par le processeur. Pour exécuter un programme il faut le charger (copier) dans la mémoire centrale. Le temps d'accès à la mémoire centrale et sa capacité sont deux éléments qui influent sur le temps d'exécution d'un programme (performance d'une machine).

Types de mémoires centrales

Il existe deux grandes familles de mémoires centrales : les mémoires statiques (SRAM) et les mémoires dynamiques (DRAM):

- Les mémoires statiques sont à base de bascules de type D. Elles possèdent un faible taux d'intégration mais un temps d'accès rapide (Utilisation pour les mémoires caches).
- Les mémoires dynamiques sont à base de condensateurs. Elles possèdent un très grand taux d'intégration, elles sont plus simples que les mémoires statiques mais avec un temps d'accès plus long.

3.3.4 RAM (Random Access Memory)

Une mémoire à accès instantané en lecture-écriture, elle est volatile, c'est-à-dire que les données disparaissent avec l'interruption de l'alimentation en courant électrique. Elle fait partie des mémoires dites dynamiques, à cause du coût excessivement élevé des mémoires statiques.

3.3.5 ROM (Read Only Memory)

Mémoire à lecture seule, c'est à dire qu'elle n'est utilisée que pour en lire des données initialement stockées; ce principe de mémoire ROM est assez utilisé pour stocker des programmes nécessaires au fonctionnement d'appareils ; dans le cas de l'ordinateur, il est fortement utilisé pour contenir le BIOS.

3.3.6 Mémoire Cache (antémémoire)

C'est une mémoire fabriquée à base de module de *SRAM*. Elle offre des temps d'accès de l'ordre de 2 à 5 nanosecondes. Elle sert à stocker les instructions et données fréquemment demandées par le microprocesseur. Au début, les mémoires caches résidaient à l'extérieur du processeur ; actuellement elles sont intégrées dans la puce du microprocesseur. De nos jours, la mémoire cache processeur est structurée en trois niveaux (L1, L2 et L3), sa taille est de l'ordre de quelques Mégaoctets (3 à 12 Mo)

3.3.7 Les unités informatiques

1. **Octet** : nom commun de genre masculin, d'origine latine " Octo " qui signifie " huit ". Il est définit comme une unité de mesure de la quantité d'information numérique ; c'est une séquence de huit bits, permettant de représenter 256 valeurs ou combinaisons.
2. **Bit** = chiffre binaire qui représente 0 ou 1.
3. **Byte** = 8 Bits = 1 Octet.

En 1998, les spécialistes de la métrologie ont décidé de normaliser les préfixes des unités utilisées en informatique, et ont décidé que les préfixes Kilo, Méga, Giga, Téra, etc. correspondent aux mêmes multiplicateurs que dans tous les autres domaines, soit des puissances de 10:

```
1 kilooctet (ko) = 103 = 1 000 octets
1 mégaoctet (Mo) = 106 octets = 1 000 ko = 1 000 000 octets
1 gigaoctet (Go) = 109 octets = 1 000 Mo = 1 000 000 000 octets
1 téraoctet (To) = 1012 octets = 1 000 Go = 1 000 000 000 000 octets
```

La nécessité d'utiliser le système d'unité à base de puissance de 2 a conduit à la création de nouveaux préfixes qui permettent de représenter les puissances de 2 :

```
1 kibi-octet (kio) = 210 octets = 1024 octets
1 mébi-octet (Mio) = 220 octets = 1024 Kio
1 gibi-octet (Gio) = 230 octets = 1024 Mio
1 tébi-octet (Tio) = 240 octets = 1024 Gio
1 pébi-octet (Pio) = 250 octets = 1024 Tio
```

3.3.8 Software (Logiciel)

On désigne par software la partie logicielle, qui peut être : des systèmes d'exploitation, des langages de programmation, ou bien de simples logiciels spécialisés.

3.3.9 Systèmes d'exploitation

Souvent appelé OS (Operating System), c'est l'ensemble de programmes destinés à interagir avec un ordinateur et exploiter ses ressources pour effectuer des tâches.

3.3.10 Systèmes d'exploitation

Exemple :

MS-DOS, Windows [3.11, 98, Milinium, NT, 2000, XP, Vista, Seven, Windows 8, Server], Unix, Linux, Mac OS, etc.

3.4 Langages de programmation

Ce sont des notations artificielles, destinées à exprimer des algorithmes et produire des programmes. D'une manière similaire à une langue naturelle, un langage de programmation est fait d'un alphabet, un vocabulaire et des règles de grammaire.

Exemple :

Assembleur, Basic, Pascal, Turbo Pascal, Perl, Ada, Algol, Visual Basic, Delphi, C, $C++$, C, $Java$, $Html$, XML, $D - Base$, $Matlab$, $Scilab$, $Fortran$, etc.).

3.4.1 Logiciels spécialisés

Ce sont des programmes destinés à réaliser des tâches spécifiques.

Exemple :

Adobe, Microsoft Office, Dreamweaver, Comsol Multiphysics, Ansys, Wien2k, Latex, Winedt, Scientifique Workplace, Carine Cristallographie, Maud, Labview, etc.

3.4.2 Algorithme

L'algorithme provient du mathématicien perse Al Khawarizmi(Al-Khawarizmi ,né en 783, originaire de Khiva dans la région du Khwarezm, dans l'actuel Ouzbékistan, mort vers 850 à Bagdad, est un mathématicien, géographe, astrologue et astronome perse, membre des Maisons de la sagesse. Ses écrits, rédigés en langue arabe, ont permis l'introduction de l'algèbre en Europe. Sa vie s'est déroulée en totalité à l'époque de la dynastie Abbasside.) (9^{eme} siècle). Abu Abdullah Muhammad Ibn Mussa Al-Khawarizmi passe pour être le père de la théorie des algorithmes ainsi que de l'algèbre (de l'arabe " Al-jabr " signifiant compensation).
Un algorithme est un ensemble d'actions à faire, en utilisant des données initiales, pour aboutir à la solution d'un problème.
La résolution d'un problème peut se faire d'une multitude de manières, et à chaque manière va correspondre un Algorithme.
L'écriture d'un Algorithme ne nécessite pas des connaissances d'un Langage de programmation ; le langage courant (Naturel) suffit.

3.4.3 Quelques mots sur l'algorithmique

De nos jours, l'algorithmique est associée à la programmation informatique. Cependant, elle ne date pas d'hier puisque les premiers algorithmes remontent à environ 1800 ans avant J.C avec les babyloniens, ensuite Euclide (PGCD) et beaucoup d'autres. Contrairement à ce que l'on pourrait penser, les algorithmes ne se traitent pas qu'avec des nombres ; il en existe énormément qui traitent d'autres données, comme l'algorithme génétique, les algorithmes de jeux, etc. Les algorithmes ne se décrivent pas avec un langage de programmation, leur écriture ne nécessite pas un ordinateur. La résolution de problèmes par le biais d'algorithmes va énormément faciliter le passage à la programmation, en évitant les erreurs et en économisant le temps d'exécution.

Langage algorithmique

Un algorithme est une suite d'actions, qui une fois exécutée correctement conduit à un résultat attendu. Selon Larousse *l'algorithme est un ensemble de règles opératoires dont l'enchaînement permet de résoudre un problème au moyen d'un nombre fini d'opérations* . Autrement dit, un algorithme décrit les étapes à suivre pour réaliser un travail.

4.1 Structure générale d'un algorithme

La présentation d'un algorithme sous forme de bloc est très proche du langage de programmation. Sa structure est la suivante :

```
Algorithme <identificateur_nom> ; {En-tête}

Variables
          <identificateur> : <Type> ; {Partie déclarations}

Début
          <partie actions> ; {Corps de l'algorithme}
Fin.
```

Notons que chaque partie de l'algorithme possède des mots clés spécifiant la nature ainsi que l'étape de description.

4.1.1 En-tête

L'en-tête d'un algorithme est de la forme suivante :

```
        Algorithme <identificateur_nom>;
```

où *Algorithme* est un mot clé indiquant le début d'un algorithme (en Pascal on utilise le mot Program) et *identificateur_nom* c'est le nom donné par le programmeur. Généralement, on choisit un nom évoquant le rôle de l'algorithme. Notons que le point-virgule (;) indique la fin de l'en-tête. Il permet de séparer l'en-tête du reste de l'algorithme.

4.1.2 Partie déclaration

Elle contient la déclaration de tous les objets manipulés (constantes et variables) par un algorithme. Elle associe à chaque objet un nom (identificateur), un type et éventuellement une valeur (pour les constantes).

Identificateurs :

C'est un nom que l'on attribue à toute entité manipulée dans un programme. Les identificateurs sont choisis librement, par l'utilisateur, toutefois ils obéissent à certaines règles :

1. Un nom doit commencer par une lettre et non par un chiffre.
   Exemple 1 : d1 ou D1 et non 1D;
2. Doit être constitué uniquement de lettres, de chiffres et du soulignement (éviter les caractères de ponctuation et les espaces),
   Exemple 2 : SM2013, USTHB et non SM 2013, U S T H B.
3. Doit être différent des mots clés réservés au langage (par exemple en Pascal: *Var*, *begin*, *sqrt*, *write* ...).

Types de base (simples) :

Appelés aussi types élémentaires, ces types sont à valeur unique. On peut distinguer les types standards tels que les types entiers, booléens, caractères et chaîne de caractères et les types non standards tels que les types énumérés et les intervalles.

1. **Le type entier :** un objet de type entier prend ses valeurs dans l'ensemble des entiers relatifs Z. En effet, un entier s'écrit comme suit : $[+/-] < Chiffre >$. Notons que le signe [ ] caractérise un élément optionnel.
   Exemple 3 : $+20$, -10, 2013 sont des valeurs entières valides.
2. **Le type réel :** un objet de type réel prend ses valeurs dans l'ensemble R. En effet, un réel s'écrit comme suit : $[+/-]$ <partie entière>.<partie fractionnaire> E <entier>; E désigne la puissance de 10.
   Exemple 4 : $+12.05$, $-0.05 \equiv -5E-2$, $-2.5 \times 10^5 \equiv -2.5E+5$.
3. **Le type booléen :** appelé aussi type logique, contient deux valeurs seulement : vrai et faux.
   Exemple 5 : $Res \leftarrow X < Y$. *Res* est de type booléen, il reçoit vrai si $X < Y$, sinon faux.

4. **Le type caractère :** un objet de type caractère prend ses valeurs dans l'ensemble des caractères alphabétiques minuscules et majuscules, numériques et caractères spéciaux $*, +, /, ?, <, > etc$. Les opérations qu'on peut appliquer sur le type caractère sont les opérations de comparaison $(>, <, =, <=, etc)$ ainsi que la concaténation qui donne une chaîne de caractères. La représentation d'un caractère impose que ce dernier soit mis entre deux guillemets.
   Exemple 6 : $'A' < 'B' < 'C'$.
5. **Le type chaîne de caractères :** un objet de type chaîne de caractères est composé d'un ensemble d'objets de type caractère.
   Exemple 7 : 'USTHB' , 'Section', 'SM2013'.
   Les opérations qu'on peut appliquer sur le type chaîne de caractères sont les mêmes appliquées sur un des objets de type caractère.
   Exemple 8 : $'Sections' + ' ' + 'SM' = 'Sections\ \ SM'$

Déclaration des objets (constantes et variables) :

1. **La déclaration des constantes :** une constante est une variable dont la valeur ne change pas au cours de l'exécution du programme. Elle peut être un nombre, un caractère, ou une chaîne de caractères.
   Syntaxe:

```
Constante <identificateur> = <valeur de la constante>;
```

   Exemple 9 : Constante:
   Pi = 3.14
   Ln2 = 2.93
2. **La déclaration des variables :** une variable sert à stocker la valeur d'une donnée dans un langage de programmation. Elle désigne un emplacement mémoire dont le contenu peut changer au cours de l'exécution d'un programme (d'où le nom de variable). Chaque emplacement mémoire a un numéro qui permet d'y faire référence de façon unique : c'est l'adresse mémoire de cette cellule. La déclaration des variables commence par le mot clé Variables.
   Syntaxe:

```
Variables <identificateur> : <type de la variable>;
```

   Exemple 10 :

```
    Variables
```

```
        Prix : Réel ;
        Nombre_etudiant : Entier ;
        Nombre_groupe : Entier;
        Nom_section : Caractère ;
        Nom_etudiant : Chaîne de caractères ;
```

4.1.3 Partie instructions

Appelée aussi partie d'actions; elle commence par le mot clé *Début* et se termine par *Fin*. Ces deux bornes constituent deux parenthèses, l'une ouvrante et l'autre sortante, délimitant un bloc appelé corps de l'algorithme. Ce bloc regroupe toutes les instructions nécessaires pour traiter un problème donné.

Expression :

Une expression (située à droite de la flèche) peut être une valeur, une variable ou une opération constituée de variables reliées par des opérateurs. Il existent trois types d'expressions:

1. **L'expression de base :** appelée aussi élémentaire, représente les valeurs que peut prendre une constante ou une variable.
   Exemple 11 : $-10, 1, +50$, *Vrai* et *Faux*.
2. **L'expression arithmétique:** elle est formée par des combinaisons d'objets numériques (entier et réel) et des opérateurs arithmétiques. Une expression arithmétique donne un résultat numérique dont le type est entier ou réel.
   Exemple 12 : $a * 2$, $(a + 3) * b/c$, $k * q_1 * q_2/r * r$.
3. **L'expression logique:** elle est formée d'objets de type booléen et d'opérateurs logiques. Une expression booléenne donne un résultat booléen (vrai ou faux).
   Exemple 13 : $(X \leq Y)$ et B est une expression logique formée de deux variables booléennes.

Opérateur :

C'est un signe qui relie deux valeurs, pour produire un résultat. Il existe plusieurs types d'opérateurs.

1. **Les opérateurs arithmétiques :** Ils permettent le calcul de la valeur d'une expression arithmétique. Le tableau suivant montre quelques types d'opérateurs arithmétiques.

Symboles	Opérations
+ , -	Addition et soustraction
* , /	Multiplication et division
div	Division entière
mod	Modulo (reste de la division entière)

2. **Les opérateurs rationnels :** ils permettent de comparer deux valeurs de même type (numérique ou caractère) en fournissant un résultat booléen (vrai ou faux).

Symboles	Relation
$>$	Supérieur
$<$	Inférieur
$>=$	Supérieur ou égal
$<=$	Inférieur ou égal
$=$	Égal
$<>$	Différent

Exemple 14 : *Comp* $\leftarrow A > B$. *Comp* doit être de type booléen. A et B sont de même type.

3. **Les opérateurs logiques :** ces opérateurs sont appliqués à des données de type booléen et renvoient un résultat de type booléen.

Opérateurs logiques	Fonction
Et	réalise une conjonction entre deux valeurs booléennes
Ou	réalise une disjonction entre deux valeurs booléennes
Non	réalise une négation d'une valeur booléenne

Le tableau de vérité:

Soit A et B deux variables booléennes. Le tableau suivant illustre les différentes valeurs de vérité que l'on obtient en combinant les valeurs de A et B à l'aide des opérateurs logiques.

A	B	Non A	Non B	A et B	A ou B
Vrai	Vrai	Faux	Faux	Vrai	Vrai
Vrai	Faux	Faux	Vrai	Faux	Vrai
Faux	Vrai	Vrai	Faux	Faux	Vrai
Faux	Faux	Vrai	Vrai	Faux	Faux

Instructions de base (simples) :

L'instruction simple est une opération élémentaire, c'est à dire, l'ordre le plus basique (simple) que peut comprendre un ordinateur. Principalement, elle comporte deux éléments : l'action et les éléments sur lesquels l'action va être effectuée. Par exemple, une instruction demandant l'addition des nombres a et b fera intervenir l'opérateur d'addition et les valeurs identifiées par a et b. Nous distinguons deux instructions simples : l'affectation et les opérations d'entrée et de sortie.
Notons qu'il existe d'autres types d'instructions, à savoir, l'instruction structurée (structure conditionnelle et itérative) et l'instruction composée qui peut contenir à la fois des instructions simples et structurées.

1. **L'affectation:** elle consiste à attribuer une valeur à une variable (c'est-à-dire remplir le contenu d'une zone d'une mémoire). L'affectation est réalisée au moyen de l'opérateur $\longleftarrow$ (ou = en C et := en Pascal).
   Exemple 15 : $X \leftarrow 5$, signifie attribuer la valeur 5 à la variable X.
   Exercice 1 :
   Quelles sont les valeurs successives prises par les variables X et Y suite aux instructions suivantes :
   $X \leftarrow 6$, $Y \leftarrow (-4)$, $X \leftarrow (-X+3)$, $X \leftarrow (Y-5)$ et $Y \leftarrow (X+Y)$.
   Solution 1 :

X	6	6	-3	-9	-9
Y		-4	-4	-4	-13

   Remarque 1 :
   (a) L'affectation est une expression qui retourne une valeur. Il est donc possible d'écrire: X $\leftarrow$ Y $\leftarrow$ Z+2, ce qui signifie: on affecte la valeur Z+2 à la variable Y, puis à la variable X la valeur de Y (résultat de l'affectation Y $\leftarrow$ Z+2). La valeur finale de X est Z+2.
   (b) L'affectation est différente d'une équation mathématique

   - les opérations j $\leftarrow$ j+1 et j $\leftarrow$ j-1 sont respectivement désignés par incrémentation et décrémentation.
   - On n'affecte jamais une valeur à une équation.

2. **L'opération d'entrée (lecture):** elle permet d'entrer des données à partir d'un périphérique d'entrée (clavier).
   Syntaxe:

```
Lire (Identificateur_variable)
```

Cette dernière écriture est équivalente à *Identificateur_variable* = *valeur* lue à partir du clavier. L'instruction lire permet d'initialiser une ou plusieurs variables de même type à la fois. elle diffère de l'affectation qui ne se fait pas à partir du clavier.

Exemple 16 :

Variables A, B : <type>

Lire (A), Lire (B) ou bien Lire (A,B).

Remarque 2 : Lorsqu'un programme rencontre l'instruction *lire*, il s'arrête de s'exécuter jusqu'à ce qu'on termine l'introduction et la validation de toutes les valeurs à partir du clavier.

3. **L'opération de sortie (écriture):** elle permet d'afficher la valeur d'une expression sur un périphérique de sortie (écran). Une expression peut être un nombre, une variable numérique, un résultat d'une opération entre plusieurs variables, ou une chaîne de caractères quelconque (qui peut contenir des lettres accentuées et des espaces) : dans ce dernier cas, il est nécessaire de mettre la chaîne de caractères entre deux apostrophes.

   Syntaxe:

```
Écrire (expression) ;
```

Exemple 17 :

Ecrire (a) ;

Ecrire ('Section SM') ;

Exercice 2 :

Écrire un algorithme qui demande à l'utilisateur d'introduire un nombre entier, puis calcule et affiche le carré de ce nombre.

Solution 2 :

Algorithme Carré;

Variables A, B : entier;

Début

écrire('entrer la valeur de A');

lire(A);

$B \leftarrow A * A$;

écrire('le carré de ', A, 'est :', B);

Structures conditionnelles et itératives

5.1 Structures conditionnelles

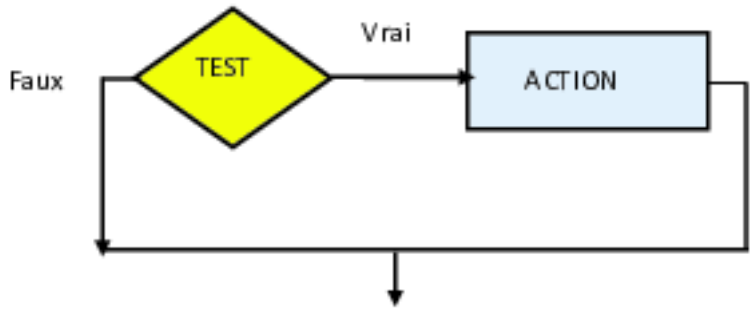
Ce sont des instructions qui permettent d'exécuter une ou plusieurs actions en fonction d'une condition.

Types d'instructions conditionnelles :

1. **L'instruction alternative simple (si ... alors ...)**
   Ce type d'instruction présente une seule possibilité; elle ne s'exécute que dans le cas où la condition est vérifiée. Le cas contraire entraîne une sortie immédiate de l'instruction en ignorant le reste des actions qui viennent.
   Syntaxe:

```
Si < Cond >  Alors < Bloc d'instructions >
```

   Où *Cond* est une expression booléenne.
   Organigramme 1 : Fonctionnement:



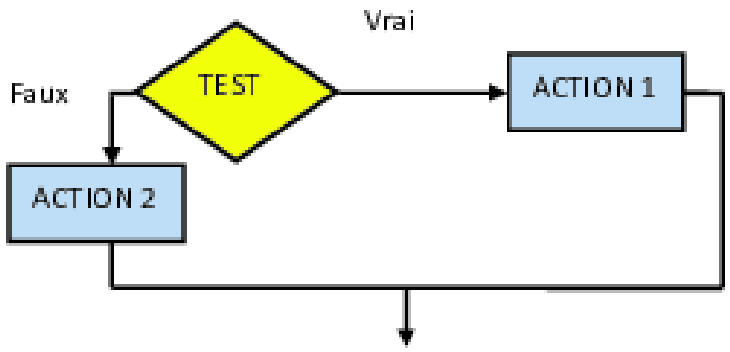
1-Évaluer la condition *Cond*.
2-Si *Cond* est **vraie**, le bloc d'instructions sera exécuté.
3-Si *Cond* est **fausse**, le bloc d'instructions ne sera pas exécuté.
Exemple 18 :

```
Si (y<0) alors
      Ecrire ('y est négatif')
FinSi
```

2. **L'instruction alternative multiple (si ... alors ... sinon)**
Cette instruction offre un choix entre deux possibilités, selon le résultat du test de la condition.
Syntaxe:

```
  Si < Cond >  Alors < Bloc d'instructions 1 >
               Sinon < Bloc d'instructions 2 >
Finsi
```

Organigramme 2 : Fonctionnement:



1-Évaluer la condition *Cond*.
2-Si *Cond* est **vraie**, toutes les instructions du bloc 1 seront exécutées.
3-Si *Cond* est **fausse**, toutes les instructions du bloc 2 seront exécutées.
Exemple 19 : Afficher le signe d'un nombre entier x.

```
Si (x>= 0) alors
   Ecrire(x, ' est positif')
Sinon
   Ecrire(x, ' est négatif')
FinSi
```

3. **L'instruction alternative imbriquée**

Le traitement de certains problèmes qui présentent plusieurs possibilités (choix multiples) implique l'ouverture de plusieurs voies, correspondant à des tests imbriqués les uns dans les autres.

Syntaxe:

```
 Si <Cond 1> Alors <instructions 1>
            Sinon  Si <Cond 2>  Alors <instructions 2>
                                Sinon Si <Cond 3> Alors <instructions 3>
                                                  Sinon <instructions 4>
                                      Finsi
                    Finsi
 Finsi
```

Fonctionnement:

1- Evaluer la condition *Cond1*

2-Si *Cond 1* est **vraie**, le bloc d'instructions 1 sera exécuté.

3-Si *Cond 1* est **fausse**, on évalue *Cond 2*.

4-Si *Cond 2* est **vraie**, le bloc d'instructions 2 sera exécuté.

5-Si *Cond 2* est **fausse** on évalue *Cond 3*.

6-Si *Cond 3* est **vraie**, le bloc d'instructions 3 sera exécuté.

7-Si *Cond 3* est **fausse**, le bloc d'instructions 4 sera exécuté.

Exemple 20 : Comparaison entre deux entiers positifs x et y.

```
Si (x = y) alors
               Ecrire (x '=' y);
           sinon Si (x > y) alors
                                Ecrire (x '>' y);
                            sinon
                                Ecrire (x '<' y);
                  FinSi
           FinSi
Fin.
```

4. **L'instruction** *Cas...Vaut...*
   Elle permet de faire un choix parmi plusieurs cas possibles, suivant la valeur d'une expression.
   Syntaxe:

```
Cas <expression> vaut
  <valeur 1> : <Action 1>;
  <valeur 2> : <Action 2>;
  <valeur 3> : <Action 3>;
       .           .
       .           .
  <valeur n> : <Action n>
  Autres ou sinon : <Action>;
Fin cas ;
```

   Exemple 21 : L'affichage du libellé de la saison en connaissant le numéro du mois.

```
Cas numéro vaut
  01, 02, 12 : écrire ('hiver');
  03, 04, 05 : écrire ('printemps');
  06, 07, 08 : écrire ('été');
  09, 10, 11 : écrire ('automne')
  Sinon : écrire ('numéro du mois incorrect')
Fincas;
```

5.2 Structures itératives

Les instructions itératives, appelées aussi instructions ou structures répétitives, permettent d'exécuter plusieurs fois une instruction. Ce type de construction est appelé boucle. Dans une boucle, le nombre de répétitions peut être connu, fixé à l'avance, comme il peut dépendre d'une condition permettant l'arrêt et la sortie de cette boucle.

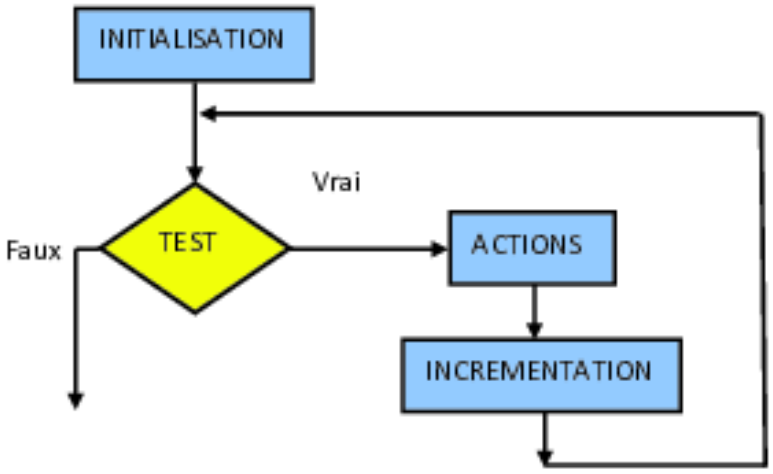
Cas où le nombre de répétitions est inconnu :

1. **La boucle tant que**
   Dans cette structure, le bloc d'actions n'est exécuté qu'après la vérification de la condition.
   Syntaxe:

```
TANT QUE <Cond>  FAIRE
        < Bloc d'instructions >
```

```
FIN TANT QUE
```

Organigramme 3 :

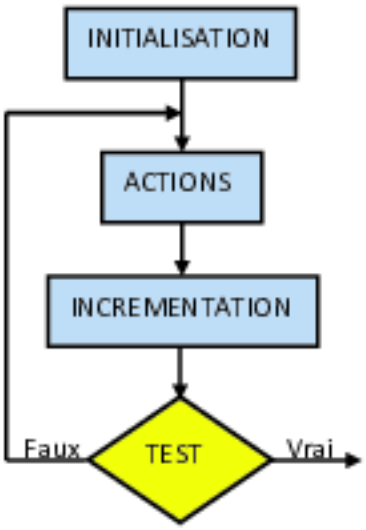


Fonctionnement:
1-Au début de chaque itération, la condition est évaluée.
2-Si *Cond* est **vraie**, le bloc est exécuté sinon on sort de la boucle.
3-Le Bloc est répété tant que *Cond* est Vraie.
4-Si *Cond* est **fausse** au début, aucune instruction ne s'exécuterait.
Exemple 22 : L'algorithme qui calcule la somme de N nombre entiers $(N >= 2)$ peut être écrit comme suit:
Algorithme Somme;
Variables N, I : entier;

Début
Ecrire ('Introduire le nombre de valeurs, N'); Lire (N);
$Som \leftarrow 0$;
$I \leftarrow I$;
Tant que $(I <= N)$ faire
$Som \leftarrow Som + I$;
$I \leftarrow I + 1$;

Finfaire
Ecrire ('La somme est Som=', Som);
Fin.

2. **La boucle répéter**
Cette boucle permet de répéter les instructions jusqu'à la vérification de la condition.
Organigramme 4 :



Syntaxe:

```
RÉPÉTER
     < Bloc d'instructions >
JUSQU'À <COND>
```

Fonctionnement:
1- Le bloc d'instructions est répété jusqu'à ce que la condition *Cond* soit Vraie.
2- A la fin de chaque itération, l'expression logique *Cond* est évaluée.
3- Le bloc d'instructions est répété au moins une fois.
Exemple 23 : L'algorithme qui permet l'affichage d'un message plusieurs fois peut être écrit comme suit:
Algorithme Répéter;
Variables n, I : entier;

Début
Ecrire('entrer la valeur de n');
Lire(n);
$I \leftarrow 1$;
Répéter
Ecrire ('Faculté de Physique USTHB 2013');
$I \leftarrow I + 1$;
Jusqu'à $(I > n)$;
Fin.

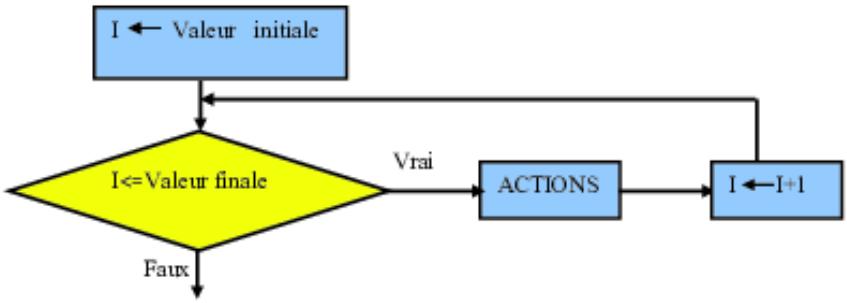
3. **La différence entre la boucle Tant que et Répéter :**
   - Les instructions de la boucle Répéter sont exécutées au moins une fois, car l'évaluation de la condition vient après le passage dans la boucle. Cependant, les instructions de la boucle Tant que peuvent ne jamais être exécutées si la condition n'est pas vérifiée, lors de la première évaluation.
   - La boucle Répéter s'exécute jusqu'à ce que la condition soit vérifiée. Donc la condition permet la sortie de la boucle Répéter. La boucle Tant que s'exécute tant que la condition est vérifiée. Donc la condition permet le passage dans la boucle Tant que.
   - Une séquence d'instructions nécessite une délimitation par *Debut* ... *Fin* dans le cas d'une boucle Tant que; ce qui n'est pas le cas de la boucle répéter.

Cas où le nombre de répétitions est connu :

1. **La boucle Pour**
   Elle permet d'exécuter une séquence d'instructions un nombre de fois connu à priori. Cette répétition se fait grâce à une variable compteur ou indice qui progresse dans un intervalle, pris sur un ensemble dénombrable et ordonné (entiers, char).
   Syntaxe:

```
POUR i ALLANT de valeur initiale A valeur finale FAIRE
        < Bloc d'actions >
FIN POUR
```

   Organigramme 5 :



Fonctionnement:

- Pour chaque valeur de la variable de contrôle (Compteur) qui varie de la valeur initiale à la valeur finale avec un pas égale à 1, le bloc sera exécuté.
- La sortie de cette boucle s'effectue lorsque le nombre souhaité de répétitions est atteint.
- Dans cette boucle, l'incrémentation (ajouter 1) est faite automatiquement.

Exemple 24 : Calcul de la somme des 20 premiers entiers positifs.
Algorithme Somme;
Variables S, I : entier;

Début
$S \leftarrow 0$;
Pour $I \leftarrow 1$ à 20
faire $S \leftarrow S + I$;
Finfaire
Ecrire ('La somme est S=', S);
Fin.

2. **La boucle imbriquée**
   Les instructions d'une boucle peuvent être des instructions itératives. Dans ce cas, on aboutit à des boucles imbriquées.
   Exemple 25 :

```
Pour i allant de 1 à 4
       Pour j allant de 1 à i
                 Ecrire ('S')
       FinPour
Ecrire ('M 2013')
```

```
    FinPour
```

Résultat final: pour i=4, le résultat est *SSSSM* 2013.

Remarque 3 : Comment choisir le type de boucle à utiliser ?

- Si on peut déterminer le nombre d'itérations avant l'exécution de la boucle, il est plus judicieux d'utiliser la boucle Pour.
- S'il n'est pas possible de connaître le nombre d'itérations avant l'exécution de la boucle, on fera appel à l'une des boucles Tant Que ou Répéter.
- Pour le choix entre Tant Que et Répéter :
- Si on doit tester la condition de contrôle avant de commencer les instructions de la boucle, on utilisera Tant Que.
- La valeur de la condition de contrôle dépend d'une première exécution des instructions.

Tableaux, fonctions et procédures

6.1 Tableaux

Les variables utilisées, jusqu'à présent, étaient toutes de type simple prédéfini (entier, réel, caractère ou logique). Dans cette partie, nous allons voir une autre manière de représenter les données, c'est le type *structuré Tableau*. Une variable de type tableau n'est plus une variable simple, mais une structure de données, regroupant des informations de même type. Nous distinguons deux types de tableaux: les tableaux à une dimension (Vecteurs) et les tableaux à plusieurs dimensions (Matrices).

6.1.1 Tableaux à une dimension (vecteurs)

Définition :

On appelle un vecteur V d'ordre n, noté $V(n)$, un ensemble de n éléments de même type disposés dans un tableau à n cases. Un vecteur possède un identificateur (nom du vecteur) et un indice qui indique le numéro de la case.
Exemple 26 : Soit T un vecteur de 6 cases

12	-4	1	-8	6	100

$T[i]$ est le contenu (l'élément) de la case N° i.
$T[3]$ est égal à 1, c'est à dire l'élément de la case N° 3 est 1.

Déclaration d'un vecteur :

La déclaration d'une variable de type tableau à une dimension (vecteur) se fait par :

```
Variables <identificateur_tab>: tableau [Borne_inf..Borne_sup] de <Type>.
Où:
- <identificateur_tab> est le nom du vecteur.
- <Borne_inf> est le numéro de la première case du vecteur.
```

```
- <Borne_sup> est le numéro de la dernière case du vecteur.
- <Type> est le type des éléments du vecteur.
```

Il est nécessaire de rajouter, dans la partie déclaration, la définition de l'indice de parcours (de la borne inférieure du vecteur à la borne supérieure) qui est une variable simple de type entier ou caractère.

```
 <identificateur_indice> : <Type>.
```

Exemple 27 :

```
Variables  T : Tableau [1..100] de caractères
           i : entier
           V : Tableau ['A'..'F'] de réels
           j : caractère
```

T est un vecteur de 100 cases destinées à contenir des éléments de type caractère, indicé par une variable entière i.
V est un vecteur de 6 cases (nombre de lettres comprises entre A et F) destinées à contenir des éléments de type réel, indicé par une variable caractère j.

Remarque 4 :

1- La déclaration du tableau suivant: T: tableau [1..N] est fausse car N n'est pas initialisé.

2- Si on a plusieurs tableaux (vecteurs) de même type et de même type, on peut les déclarer tous ensembles en les séparant par des virgules.

Exemple 28 : Soient T_1, T_2 et T_3 trois vecteurs de même type (entiers), leur déclaration peut être faite comme suit : Variables T_1, T_2, T_3 : Tableau [1..10] d'entiers.

Initialisation d'un élément d'un vecteur (affectation) :

L'initialisation d'un élément du vecteur se fait comme suit

$$< identificateur > [Indice] \longleftarrow < expression >$$

Exemple 29 : Soit T un vecteur de 4 éléments entiers

$T[1] \longleftarrow -5$

$T[2] \longleftarrow 2$

$T[3] \longleftarrow T[1] - T[2]$

$T[4] \longleftarrow T[3] + 6$

Le déroulement de cet exemple nous donne le vecteur suivant:

-5	2	-7	-1

Lecture et affichage des éléments d'un vecteur :

Nous avons vu que la lecture d'un nombre à partir du clavier se fait par l'instruction *Lire(x)*; où x est une variable simple. Il en est de même pour le remplissage d'un vecteur. Ses cases étant des variables simples, on leur attribue une à une des valeurs à partir du clavier par les instructions :

```
Lire (T[1]),  Lire (T[2]),. . . Lire (T[n])
```

Remarquer que l'instruction *Lire* est répété n fois. Afin d'éviter la répétition de l'instruction *Lire*, nous utilisons une des structures itératives que nous avons vu précédemment. Le nombre d'itérations étant connu (n), pour cela nous utilisons la boucle *Pour*.

Santaxe

```
Pour  i  allant  de  1  à  n faire
        Lire (T[i])
```

De même que pour le remplissage d'un vecteur, l'affichage de son contenu se fait par l'instruction :

```
Pour  i  allant  de  1  à  n faire
        Écrire (T[i])
```

Quelques opérations sur les vecteurs :

Dans un tableau de données numériques, il est possible de faire plusieurs sortes de traitements, à savoir, la recherche d'un minimum ou d'un maximum, le calcul de la somme et/ou de la moyenne des éléments, le tri d'un tableau, etc.

1. **Somme des éléments d'un tableau :**
   Pour calculer la somme des nombres contenus dans un vecteur (éléments du vecteur), il faut ajouter un à un le contenu des cases, depuis la première jusqu'à la dernière. Le résultat final nous donne un nombre et non un vecteur.
   Exemple 30 : L'algorithme qui calcule et affiche la somme des éléments d'un vecteur T de 8 cases peut être écrit comme suit:

   Algorithme Somme
   Variables T : tableau [1..8] d'entiers
   S, i : entiers
   Début
   Ecrire (' Introduire les éléments du tableau ')

Pour $i \longleftarrow$ 1 à 8 faire

Lire ($T[i]$)

S $\longleftarrow$ 0

Pour $i \longleftarrow$ 1 à 8 faire

S $\longleftarrow$ S + $T[i]$

Ecrire (' La somme des éléments du tableau est S= ', S)

Fin.

Application: Soit T le vecteur suivant :

-5	2	-7	10	25	7	-1	3

Le déroulement de la somme des éléments de ce vecteur s'effectue comme suit :

i	$T[i]$	Somme (S)
1	-5	-5
2	2	-3
3	-7	-10
4	10	0
5	25	25
6	7	32
7	-1	31
8	3	34

La somme finale des éléments de ce vecteur (T) est 34.

Remarque 5 :

Pour calculer la moyenne des éléments d'un vecteur, il faut d'abord calculer la somme de ses éléments comme indiquer précédemment, puis diviser sur le nombre de valeurs (cases) que contient le vecteur.

2. **Recherche du maximum des éléments d'un vecteur :**

La recherche d'un maximum des éléments d'un vecteur se fait comme suit:

1- on fait entrer les éléments du vecteur;

2- on suppose que la première case contient le maximum;

3- on compare le contenu de la deuxième case avec le maximum précédent. Si celui-ci (l'élément de la deuxième case) est supérieur, il devient le maximum;

4- on continue la même opération avec les cases suivantes.

Exemple 31 : L'algorithme qui recherche le maximum des éléments d'un tableau T de 8 cases peut s'écrire sous la forme:

Algorithme Maximum
Variables T : tableau [1..8] d'entiers
Max, i : entiers
Début
Ecrire ($'$ Introduire les éléments du tableau $'$)
Pour $i \longleftarrow$ 1 à 8 faire
Lire ($T[i]$)
Max $\longleftarrow$ $T[1]$
Pour $i \longleftarrow$ 2 à 8 faire
Si $T[i] >$ Max alors
Max $\longleftarrow$ $T[i]$
Ecrire ($'$ Le Maximum des éléments du tableau est Max= $'$, Max)
Fin.

Application: Nous reprenons le vecteur (T) précédent

-5	2	-7	10	25	7	-1	3

Le déroulement de la recherche du Maximum des éléments de ce tableau est le suivant :

i	$T[i]$	Maximum (Max)
1	-5	-5
2	2	2
3	-7	2
4	10	10
5	25	25
6	7	25
7	-1	25
8	3	25

Le Maximum des éléments de ce tableau est 25.

Remarque 6 : Nous procédons de la même manière pour la recherche du plus petit élément (Minimum) du vecteur.

6.1.2 Tableaux à deux dimensions (matrices)

Les tableaux manipulés jusqu'à présent sont à une dimension (vecteurs). Par ailleurs, lorsqu'un traitement utilise plusieurs tableaux à une dimension ayant le même nombre d'éléments et subis-

sant le même traitement, on utilise un seul tableau à deux dimensions (matrice).

Définition :

On appelle matrice A d'ordre $m \times n$, notée $A(m,n)$, un ensemble de $m \times n$ éléments rangés dans des cases disposées en m lignes et n colonnes.
Notons que l'élément d'une matrice, noté $A[i,j]$, est repéré par deux indices; le premier indique la *ième* ligne et le second indique la *jème* colonne.
Exemple 32 : Soit A une matrice d'ordre 3×4

12	-4	1	-8
3	-20	0	-6
-5	-4	10	7

$A[i,j]$ est le contenu (l'élément) de la ième ligne et la j^{eme} colonne.
$T[1,1] = 12$, c'est à dire l'élément de la première ligne et la première colonne.
$T[2,3] = 0$ et $T[3,4] = 7$.

Déclaration d'une matrice :

La déclaration d'une variable de type tableau à deux dimensions (matrice) se fait par l'une des deux syntaxes suivantes:

```
Variables <iden_mat>: tableau [B_inf_l..B_sup_l, B_inf_c..B_sup_c]
de <Type>.

ou bien par:

Variables <iden_mat>:tableau [B_inf_l..B_sup_l] de tableau
[B_inf_c..B_sup_c] de <Type>.

              Où - <iden_mat> est le nom de la matrice;
                 - <B_inf_l> est le numéro de la première ligne;
                 - <B_sup_l> est le numéro de la dernière ligne;
                 - <B_inf_c> est le numéro de la première colonne;
                 - <B_sup_c> est le numéro de la dernière colonne;
                 - <Type> est le type des éléments de la matrice;
```

Notons que la première syntaxe est celle qu'on utilise souvent.
De même que pour les vecteurs, on doit déclarer les indices de parcours des lignes et des colonnes qui peuvent être de type entier ou caractère.

```
<idetificateur_indice_ligne> : <Type>.
<idetificateur_indice_colonne> : <Type>.
```

Exemple 33 :

```
Variables  M : Tableau [1..10,1..4] de réels
           i,j : entiers
           ou bien
           M : Tableau [1..10] de tableau [1..4] de réels
           i,j : entiers
```

M est une matrice de 10 lignes et de 4 colonnes, ses cases sont destinées à contenir des éléments de type réel.
$M[i,j]$ est le contenu de la case qui se trouve à la ligne i et la colonne j.
Remarque 7 :
Si on a plusieurs matrices de même ordre et de même type, on peut les déclarer à la fois en les séparant par des virgules.

Exemple 34 : Soient M_1, M_2 et M_3 trois matrices d'ordre 5×4 de même type (réel), leur déclaration peut être faite comme suit : Variables M_1, M_2, M_3 : Tableau [1..5,1..4] de réels.

Initialisation d'un élément d'une matrice :

Comme dans le cas d'un vecteur, l'initialisation d'un élément d'une matrice se fait par l'affectation selon la syntaxe suivante

$$< identificateur > [Indice - ligne, indice - colonne] \longleftarrow < expression >$$

Exemple 35 : Soit M une matrice d'ordre 2×3
$M[1,1] \longleftarrow -5$
$M[1,2] \longleftarrow 2$
$M[1,3] \longleftarrow 10$
$M[2,1] \longleftarrow M[1,2] + 6$
$M[2,2] \longleftarrow M[1,1] + M[2,1]$
$M[2,3] \longleftarrow M[1,1] + M[2,2]$
Le déroulement de cet exemple nous donne la matrice suivante:

-5	2	10
8	3	-2

Lecture et affichage des éléments d'une matrice :

Une matrice de m lignes et de n colonnes est identique à m vecteurs superposés de n cases chacun. Par conséquent, le remplissage d'une matrice se fait, ligne par ligne ou colonne par colonne, par l'instruction suivante:

```
Pour i allant de 1 à m faire
    Pour j allant de 1 à n faire
        Lire (A[i,j])
```

Où l'on utilise deux boucles *Pour* imbriquées où la première boucle englobe la seconde. L'indice i est utilisé pour le parcours des lignes et j pour les colonnes.

Déroulement du remplissage

- Initialement l'indice i de la première boucle prend la valeur 1 (première ligne).
- L'indice j de la boucle englobée progresse de 1 à n et l'instruction *Lire* est exécutée n fois, réalisant ainsi le remplissage (lecture) de la première ligne.
- L'indice i est incrémenté en prenant la valeur 2 (deuxième ligne).
- A nouveau, l'indice j progresse de 1 à n et l'instruction *Lire* est exécutée encore n fois, en effectuant ainsi le remplissage de la seconde ligne.
- De la même manière, l'opération du remplissage se poursuivra jusqu'à la dernière ligne ($i = m$).
- De même que pour le remplissage d'une matrice, l'affichage de son contenu se fait par l'instruction:

```
Pour i allant de 1 à m faire
    Pour j allant de 1 à n faire
        Écrire (A[i,j])
```

Quelques opérations sur les matrices :

Tout comme dans le cas des vecteurs, plusieurs opérations peuvent être appliquées sur les matrices. Nous illustrons dans ce qui suit le calcul de la somme des éléments de la matrice et la recherche du maximum de la matrice.

1. **Somme des éléments d'une matrice :** le calcul de la somme des éléments d'une matrice de m lignes et de n colonnes est le même que celui des vecteurs. L'addition des éléments de la matrice se fait ligne par ligne, et le résultat final nous donne un nombre et non une matrice.

   Exemple 36 : Algorithme qui calcule et affiche la somme (S) des éléments d'une matrice M d'ordre 2×3.

Algorithme Som

Variables M : tableau [1..2,1..3] d'entiers

S, i, j : entiers

Début

Ecrire ($'$ Introduire les éléments de la matrice M $'$)

Pour $i \longleftarrow$ 1 à 2 faire

Pour $j \longleftarrow$ 1 à 3 faire

Lire ($M[i, j]$)

S $\longleftarrow$ 0

Pour $i \longleftarrow$ 1 à 2 faire

Pour $j \longleftarrow$ 1 à 3 faire

S $\longleftarrow$ S + $M[i, j]$

Ecrire ($'$ La somme des éléments de la matrice M est S= $'$, S)

Fin.

Déroulement de l'algorithme:

- Entrée des éléments de la matrice.
- Initialisation de la variable S.
- Addition des éléments de la matrice ligne par ligne.

Application: Soit M une matrice d'ordre 2×3

-5	2	10
8	-3	8

Le déroulement de la somme des éléments de cette matrice s'effectue comme suit :

i	j	$M[i, j]$	Somme (S)
1	1	-5	-5
1	2	2	-3
1	3	10	7
2	1	8	15
2	2	-3	12
2	3	8	20

La somme finale des éléments de la matrice M est 20.

2. **Recherche du maximum des éléments d'une matrice :**

Le principe est le même que pour la recherche du maximum des éléments d'un vecteur; seule

la méthode de parcours diffère. Les étapes de la recherche d'un maximum d'une matrices est les suivantes:
1- On fait entrer les éléments de la matrice.
2- On utilise la variable Max. On suppose que l'élément de la première ligne et la première colonne est le maximum et on l'affecte à cette variable.
3- Tant qu'il reste des lignes à examiner faire

3.1- Tant qu'il reste des éléments à examiner sur la ligne courante faire

3.1.1- Si l'élément courant est supérieur au maximum alors

- Affecter cet élément à la variable Max

- Sinon conserver la valeur de Max.

3.1.2- Passer à l'élément suivant

3.1.3- Refaire l'étape 3.1.

3.2- Passer à ligne suivante.

3.3- Refaire toutes les étapes de 3.

On continue la même opération avec le reste des colonnes de la première ligne puis avec les colonnes du reste des lignes de la matrice.
Exemple 37 : Algorithme qui recherche le maximum des éléments d'une matrice M d'ordre 2×3.

Algorithme Max

Variables M : tableau [1..2,1..3] d'entiers
Max, i, j : entiers

Début

Ecrire ($'$ Introduire les éléments du tableau $'$)
Pour $i \longleftarrow$ 1 à 2 faire
Pour $j \longleftarrow$ 1 à 3 faire
Lire ($M[i,j]$)
Max $\longleftarrow$ $M[1,1]$
Pour $i \longleftarrow$ 1 à 2 faire
Pour $j \longleftarrow$ 1 à 3 faire
Si $M[i,j] >$ Max alors

Max $\longleftarrow M[i,j]$

Ecrire ($'$ Le Maximum des éléments de la matrice est Max= $'$, Max)

Fin.

Application: Nous reprenons la matrice (M) précédente

-5	2	10
8	-3	8

Le déroulement de la recherche du maximum des éléments de la matrice M est le suivant:

i	j	$M[i,j]$	Maximum (Max)
1	1	-5	-5
1	2	2	2
1	3	10	10
2	1	8	10
2	2	-3	10
2	3	8	10

Le Maximum des éléments de cette matrice est $Max = 10$.

Remarque 8 :

Nous procédons de la même manière pour la recherche du Minimum d'une matrice.

6.2 Fonctions et procédures

Nous savons que la construction d'un algorithme, pour résoudre un problème donné, repose sur l'écriture d'un certain nombre d'instructions dans un ordre séquentiel et logique.

Il apparaît fréquemment qu'un même ensemble d'instructions se réalise à divers emplacements de l'algorithme. Pour éviter la peine de recopier une même série d'instructions et aussi pour diminuer la taille de l'algorithme, on fera de cet ensemble un *sous-algorithme* définis séparément de l'algorithme principal.

6.2.1 Notion de sous-algorithme

La notion de sous-algorithme représente toute la puissance du langage algorithmique. En fait, c'est la possibilité de structurer encore davantage l'algorithme en créant de nouveaux ordres, définis indépendamment de l'algorithme principal, utilisables dans le corps de l'algorithme. Cela permet d'avoir un algorithme beaucoup plus petit et bien lisible. On distingue deux types de sous-algorithmes : les fonctions et les procédures.

6.2.2 Notion de fonction

Une fonction est un sous-algorithme (sous-programme en langage informatique) qui fournit un résultat à partir des données qu'on lui apporte. Pour créer une fonction, il faut déclarer l'existence de cette fonction, lui donner un nom (identificateur), définir son type (c'est à dire le type du résultat qu'elle renvoie) et, enfin, décrire le traitement qu'elle permet de réaliser.

Déclaration d'une fonction :

La déclaration des fonctions se fait à la fin de la partie déclaration de l'algorithme principal et suit le format suivant:

```
<Fonction> <iden_fonction>: (arguments: type ): <Type du résultat renvoyé>.
                        Début
                         <Corps de la fonction>
                         <iden_fonction>:=<expression>
                        Fin;
```

Où

```
- <Fonction> est le mot clé introduisant la déclaration de la fonction;
-<iden_fonction> est le nom de la fonction;
-(arguments: type) est la déclaration du type des arguments de la fonction;
- <Type du résultat renvoyé> est le type du résultat fourni par la fonction;
- <Corps de la fonction> est l'ensemble des instructions de la fonction.
   Il est délimité par ''début'' et ''fin'';
- <iden_fonction>:=<expression> est le renvoi du résultat qui se fait
  obligatoirement par l'intermédiaire du nom de la fonction, grâce
  à l'opérateur d'affectation.
```

Remarque 9 :
Tous les paramètres formels d'une fonction sont des paramètres d'entrée.
Exemple 38 : Fonction qui calcule la somme de deux nombres.

Fonction Som (A, B:réels):réel
Début
Som $\longleftarrow A + B$
Fin.

Dans cet exemple, les arguments A et B sont utilisés pour calculer leur somme. Ces arguments appelés aussi les paramètres formels.

Les paramètres formels servent à décrire un traitement donné à l'intérieur de la fonction. On pourra également y trouver des variables déclarées à l'intérieur de la fonction appelée *variables locales* et qui par conséquent, ne seront connues que de cette fonction.

Exemple 39 :

Fonction Som (A, B:réels):réel
  Variables C: réel
Début
    $C \longleftarrow 5$ {C est une variable locale de la fonction}
    Som $\longleftarrow (A + B) * C$
Fin.

Remarque 10 :

Les variables sont appelées *locales* lorsqu'elles sont utilisées dans le sous-algorithme où elles sont déclarées. Elles seront dites *globales* si elles sont utilisées dans des sous-algorithmes de niveau inférieur.

Exemple 40 : Soit SA_1 un sous-algorithme contenant deux sous-algorithmes SA_2 et SA_3. Les variables déclarées dans le sous-algorithme SA_1 sont:

- locales à SA_1,
- globales à SA_2 et SA_3.

L'appel d'une fonction :

Une fois la fonction est définie, il est possible, à tout endroit de l'algorithme principal, de faire appel à elle en spécifiant son nom suivi par la liste des paramètres effectifs.

Syntaxe:

```
<iden_variable>:= <iden_fonction>(paramètres effectifs)
```

Notons que le résultat est affecté à une variable de même type que le résultat renvoyé par la fonction.

Exemple 41 : Algorithme utilisant une fonction qui permet de calculer la surface d'un rectangle.

Algorithme Surface

Fonction Rectangle (A,B:réels):réel

Début

Rectangle $\longleftarrow A * B$ {A et B sont les paramètres formels}

Fin.

Variables S, Ln, Lr : Réels

Début

Lire (Ln, Lr)

$S \longleftarrow$ Rectangle (Ln, Lr) {Ln et Lr sont les paramètres effectifs}

Ecrire ('la surface du rectangle est $S =$', S)

Fin.

6.2.3 Notion de procédure

Une procédure est une fonction qui ne renvoie pas une valeur : elle effectue donc un traitement particulier sans fournir un résultat à partir des données qu'on lui apporte.

Déclaration d'une procédure :

Les procédures se déclarent et s'utilisent exactement comme les fonctions excepté le fait qu'elles ne renvoient aucune valeur et qu'elles, par conséquent, n'ont pas de type.
Syntaxe:

```
<Procédure> <iden_procédure>: (arguments: type )
            Début
             <Corps de la procédure>
            Fin;
```

Où

```
- <Procédure> est le mot clé introduisant la déclaration de la
  procédure;
- <iden_procédure> est le nom de la procédure;
- (arguments: type) est la déclaration du type des paramètres formels
  de la procédure;
- <Corps de la fonction> est l'ensemble des instructions de la procédure.
  Il est délimité par ''début'' et ''fin''.
```

Exemple 42 : Déclaration ou définition d'une procédure qui calcule le minimum de deux réels.

Procédure Minimum (a, b:réels)
variables min : entier
Début
Si $a \leq b$ alors min $\longleftarrow a$
sinon min $\longleftarrow b$
Ecrire ($'$ le minimum est $'$, min)
Fin.

L'appel d'une procédure :

Tout comme le cas d'une fonction, l'appel d'une procédure se réalise, à tout endroit de l'algorithme principal, par l'occurrence du nom de cette procédure suivi par la liste des paramètres effectifs.
Syntaxe:

```
<iden_procédure>(paramètres effectifs)
```

Exemple 43 : Algorithme utilisant une procédure qui permet de calculer le factoriel d'un entier positif.

Algorithme factoriel

Procédure facto (n:entier)
Variables y, i : entier
Début
$y \longleftarrow 1$
Pour $i \longleftarrow 1$ à n faire
$y \longleftarrow y * i$
Ecrire ('le factoriel de ce nombre entier est', y)
Fin.
Variables A: entier
Début
Lire (A)

facto (*A*)

Fin.

Programmation
Exercices et solutions

Série TP N^{o}1: Exercices avec solutions

Dans cette partie d'application, nous allons proposer un ensemble de séries d'exercices traités, en séance de TP, par les étudiants de la première année SM, année universitaire 2012-2013. Ces séries d'exercices englobent tous les chapitres présentés en cours. Pour ce faire, nous avons fait appel au langage Pascal. Ce dernier est un langage facile à manipuler et possède une structure très proche de celle du langage naturel. Cette première série permettra à l'étudiant, tout d'abord, de se familiariser avec les différentes instructions de base de programmation en Pascal et de passer, par la suite, aux structures conditionnelles.

Exercice 3 : (***Write, Writeln,readln***)

1. Taper le programme suivant :

```
Program affichage;
 begin
  write('Faculte de Physique');
  write('USTHB');
 end.
```

2. Compiler (*Compile* ou Alt + F9) et exécuter (*Run* ou *Ctrl + F9*). (Pour voir l'écran d'affichage, appuyer sur *Alt+F5).*
3. Rajouter au programme l'instruction *Readln* juste avant END. Compiler et exécuter. Que remarquez-vous ?
4. Remplacer *Writeln* par *Write*, compiler et exécuter. Que remarquez-vous ? Quelle est la différence entre *Write* et *Writeln* ?
5. Remplacer 'USTHB' par 'Je suis étudiant à l'USTHB'. Que se passe-t-il ? Comment faire pour afficher la phrase en entier ?

Solution 3 :

2- La commande *Compile* ou *Alt + F9* sert à traduire le programme en langage machine (binaire).

- La commande *Run* ou *Ctrl + F9* sert à executer le programme.

- La commande *Alt+F5* sert à visualiser l'écran d'affichage.

3- En rajoutant au programme l'instruction *Readln* juste avant END, permet de voir directement,

après execution bien sûr, l'écran d'affichage sans faire appel à la commande *Alt+F5*.
4- La commande *writeln*, contrairement à *write*, permet au programme d'afficher, après execution, ligne par ligne.
5- Pour tenir compte du caractère typographique (apostrophe dans le mot l'USTHB), il faut l'écrire deux fois (voir le programme ci-dessous).

```
Program affichage;
  begin
   writeln('Faculté de Physique');
   writeln('Je suis étudiant à \emph{l''USTHB}');
   readln;
  end.
```

Exercice 4 : (*utilisation de: read, readln, affectation (:=) et opérateur**)
Écrire un programme Pascal qui permet de calculer et afficher la surface (S) d'un carré de longueur L. L étant un nombre entier.

Solution 4 :

```
Program surface;
 var l,s:integer;
 begin
      writeln('l = ');
      readln(l);
       s:=sqr(l);
      writeln('s = ',s);
      readln;
 end.
```

Exercice 5 : (***read, readln, write, writeln,affectation*** *(:=), variable de type entier*)
Écrire un programme Pascal qui permet de faire la permutation entre deux nombres entiers.
Solution 5 :

```
Program permutation;
  var x,y,t: integer;
 begin
  writeln('donner la valeur de x = '); readln(x);
  writeln('donner la valeur de y = '); readln(y);
  t:=x;
  x:=y;
  y:=t;
  writeln('la nouvel valeur de x après permutation = ',x);
  writeln('la nouvel valeur de y après permutation = ',y);
```

```
  readln;
end.
```

Exercice 6 : (*Introduction des opérateurs arithmétiques* [*, /, +, -])
Écrire un programme Pascal qui permet de déterminer et d'afficher la position (x, y), à l'instant t, d'un mobile en mouvement dans un plan donné par les équations paramètriques suivantes:

$$\begin{cases} x = 2t \\ y = \frac{3t^2}{2} - 4t \end{cases}$$

Solution 6 :

```
Program position;
var t,x,y:real;
    begin
    writeln('Donner la valeur du temps t = ');
    readln(t);
    x:=2*t;
    y:=(3*sqr(t)/2)-4*t;
    writeln('position est', x ,y);
    readln;
    end.
```

Exercice 7 : (*instructions : if then else, mod, div et opérateurs [+,-,/,*]*)
Écrire un programme qui demande un temps T (entier) exprimé en secondes, et qui le transforme en heures, minutes, secondes. Exemple: $T = 23513$ seconds $\Rightarrow$ 6 heures 31 minutes 53 seconds.
Solution 7 :

```
Program heure; var t:integer;
    H,rest,mn,s:integer;
 begin
    writeln('\'{e}crire votre temps en secondes');
    readln(t);
    H:=t div 3600;
    rest:=t mod 3600;
    mn:=rest div 60;
    s:=rest mod 60;
    writeln('l''heure est: ', H, ' heures ',mn,' minutes',s, ' seconds ');
 end.
```

Exercice 8 : (*if then else simple, if then else imbriqué*)
Écrire un programme Pascal qui demande à l'utilisateur d'introduire deux nombres entiers, et lui

affiche si leur produit est négatif ou positif, dans les deux cas suivants:

1. Le produit nul n'est pas considéré.
2. Le produit nul est inclus.

Indication: on ne doit pas calculer le produit de ces deux nombres.
Remarque: Les deux commandes *uses crt* et *clrscr*, que vous verrez dans les prochains programmes, servent à effacer l'écran d'affichage après chaque exécution du programme.
Solution 8 :

1. Cas où le produit nul n'est pas considéré:

```
Program exercice6;
    uses crt;
    var a,b,prod:integer;
    begin
    clrscr;
    write(' a= '); readln(a);
    write(' b= '); readln(b);
    prod:=a*b;
    if prod>0 then writeln(' le produit est positif')
              else if prod=0 then writeln(' le produit est nul')
                             else writeln(' le produit est n\'{e}gatif');
    readln
end.
```

   Il existe une autre façon différente de faire les choses, et qui est la suivante:

```
Program exercice6;
    uses crt;
    var a,b:integer;
 begin
    clrscr;
    write(' a= '); readln(a);
    write(' b= '); readln(b);
    if (a>0) and (b>0) or (a<0) and (b<0)
       then writeln(' le produit est positif')
    else if (a>0) and (b<0) or (a<0) and (b>0)
       then writeln(' le produit est négatif');
    readln
end.
```

2. Cas où le produit nul est inclus:

```
Program exercice6;
```

```
   uses crt;
   var a,b,prod:integer;
 begin
   clrscr;
   write(' a= '); readln(a);
   write(' b= '); readln(b);
   prod:=a*b;
   if prod=0 then writeln(' le produit est nul')
             else if prod>0 then writeln(' le produit est positif')
                            else writeln(' le produit est négatif');
   readln
end.
```

Dans ce cas, on peut, aussi, procéder autrement :

```
Program exercice6;
   uses crt;
   var a,b:integer;
 begin
   clrscr;
   write(' a= '); readln(a);
   write(' b= '); readln(b);
   if (a=0) or (b=0) then writeln(' le produit est nul')
   else if (a>0) and (b>0) or (a<0) and (b<0)
        then writeln(' le produit est positif')
           else writeln(' le produit est n\'{e}gatif');
   readln
 end.
```

Exercice 9 : (*if then else imbriqué, opérateurs logiques and et or*)
Soit a, b et c trois paramètres (de type entier) d'un triangle:

1. Si $a = b = c$ alors triangle équilatéral;
2. Si $a = b$ ou $a = c$ ou $b = c$ alors triangle isocèle;
3. Si $a <> b <> c$ alors triangle scalène (quelconque).

Écrire un programme Pascal qui permet d'afficher si un triangle est équilatéral, isocèle ou scalène.

Solution 9 :

```
Program EX07;
    uses crt;
    var a,b,c:integer;
 begin
```

```
   clrscr;
   write(' a = '); readln(a);
   write(' b = '); readln(b);
   write(' c = '); readln(c);
   if (a=b) and (b=c) then  writeln(' Triangle équilatéral ')
                      else  if (a=b) or (a=c) or (b=c)
                        then writeln(' Triangle isocèle')
                          else writeln(' Triangle normal');
   readln
end.
```

Exercice 10 : (*if then else, affectation (:=), opérateurs [+,-,/,*]*)
Écrire le programme correspondant à la résolution d'une équation de premier degré dans R:

$$ax + b = 0$$

où a, b sont des réels donnés par l'utilisateur.
Solution 10 :

```
Program equation_degre_1;
var a,b:real;
    x:real;
begin
  writeln('Résolution de l''équation ax+b=0');
  write('Donner le coefficient a : '); readln(a) ;
  write('Donner le coefficient b : '); readln(b) ;
  if a<>0
     then begin
          x:=-b/a;
          writeln('L''équation admet une solution unique x= ',x);
          end
     else if b=0 then writeln('Infinité de solutions')
                 else writeln('Pas de solution');
  readln
end.
```

Exercice 11 :
Écrire le programme correspondant à la résolution d'un systeme de deux équations de premier degré dans R:

$$\begin{aligned} a_1x_1 + b_1 &= c_1 \\ a_2x_2 + b_2 &= c_2 \end{aligned}$$

où *a, b* sont des réels donnés par l'utilisateur.

Solution 11 :

```
program systeme_deux_equations_degre_1;
 uses crt;
 var a1,b1,c1,a2,b2,c2:real;
 delta,x1,x2:real;
 begin
 clrscr;
 {*résolution du système de deux equations suivantes:*}
                    {* a1+b1x=c1  *}
                    {* a2+b2x=c2  *}
 write(' a1 = '); readln(a1);
 write(' b1 = '); readln(b1);
 write(' c1 = '); readln(c1);
 write(' a2 = '); readln(a2);
 write(' b2 = '); readln(b2);
 write(' c2 = '); readln(c2);
 delta:=a1*b2-a2*b1;
 {*1 ère Condition sur delta*}
 if delta=0 then  writeln(' l''equation admet une infinité de solutions')
                    else
                    begin
                    x1:=(c1*b2-c2*b1)/delta;
                    writeln('x1 = ',x1);
                    x2:=(a1*c2-a2*c1)/delta;
                    writeln('x2 = ',x2);
                    end;
     readln
     end.
```

*Remarque:*En Pascal, toute expression insérée entre deux accolades et deux étoiles est un commentaire qui ne sera pas pris en compte par le programme lors de son exécution. Résultat après exécution:

```
a1 = 1
b1 = 3
c1 = 5
a2 = 4
b2 = 6
c2 = 7
x1 = -1.50000000000000E+000
x2 =  2.16666666666667E+000
```

Exercice 12 :
Écrire le programme correspondant à la résolution d'une équation de seconde degré dans R:

$$ax^2 + bx + c = 0$$

où $a \neq 0$, b, c sont des réels donnés par l'utilisateur. Le but est de bien comprendre comment utiliser plusieurs conditions imbriquées. Rappelons qu'une condition ne sera alors exécutée que si la précédente le permet.
Solution 12 :

```
Program equation_degre_2;
  uses crt;
  var a,b,c:real;
     delta,x1,x2,x3:real;
 begin
  clrscr;
  {*Résolution de l'équation: a^2+bx+c=0*}
  write(' a = '); readln(a);
  write(' b = '); readln(b);
  write(' c = '); readln(c);
  if a<>0 then   {*Première condition sur a*}
                        begin
                        delta:=sqr(b)-(4*a*c);
                        if delta>0 then {*1 ère condition sur delta*}
                                  begin
                                  x1:=((-b+sqrt(delta))/(2*a));
                                  writeln('x1 = ',x1);
                                  x2:=((-b-sqrt(delta))/(2*a));
                                  writeln('x2 = ',x2);
                                  end
                        else if delta=0 then {*2 ème condition sur delta*}
                                        begin
                                        x3:=-b/(2*a);
                                        writeln('x3 = ',x3);
                                        end
                             else {*3 ème condition sur delta*}
                                        writeln(' l''équation n''admet
                                        pas de solutions  réelles');
                        end;
 readln
 end.
```

Série TP N^{o}2: Exercices avec solutions

Cette série de TP est réservée aux structures itératives. L'étudiant va pouvoir résoudre des problèmes plus complexes en faisant appel aux structures itératives, à savoir, la boucle *Pour*, *Tant que* et *Répéter*.

Exercice 13 :

Les variables i et N sont de type entier. La variable N doit être affichée et lue en entrée. Ecrire un programme Pascal qui calcule et affiche en sortie :

1. la somme de N nombres entiers définie comme suit :

$$S = 1 + 2 + 3 + ... + N = \sum_{k=1}^{N} k$$

2. le produit de N nombres entiers défini comme suit :

$$P = 1 * 2 * 3 * ... * N = \prod_{k=1}^{N} k$$

NB: Utiliser les différentes structures itératives (While, Repeat et For).

Solution 13 :

1- En utilisant la boucle *For* (*Pour*):

```
Program boucleFor;
        Var k,N,Som:integer;
            Prod:real;
  begin
        write('entrez un entier N= ');
        readln(N);
        Som:=0;
        Prod:=1;
        for k:=1  to N do begin
                          Som:=Som+k;
                          Prod:=Prod*k;
                          end;
```

```
        writeln('la somme des ', N,' premiers nombres vaut : ', Som);
        writeln('le produit des ', N,' premiers nombres vaut : ', Prod);
        readln
 end.
```

2- En utilisant la boucle *While* (*Tantque*):

```
Program BoucleWhile;
        Var k,N,Som:integer;
            Prod:real;
 begin
        write('entrez un entier: N= ');
        readln(N);
        Som:=0;
        Prod:=1;
        k:=1;
        while k<=N do begin
                      Som:=Som+k;
                      Prod:=Prod*k;
                      k:=k+1;
                      end;
        writeln('la somme des ', N,' premiers nombres vaut : ', Som);
        writeln('le produit des ', N,' premiers nombres vaut : ', Prod);
        readln
 end.
```

3- En utilisant la boucle *Repeat* (*Rpter*):

```
Program BoucleRepeat;
        Var k,N,Som:integer;
            Prod:real;
 begin
        write('entrez un entier N = ');
        readln(N);
        Som:=0;
        Prod:=1;
        k:=1;
        repeat
              Som:=Som+k;
              Prod:=Prod*k;
              k:=k+1;
        until k>N;
        writeln('la somme des ', N,' premiers nombres vaut : ', Som );
        writeln('le produit des ', N,' premiers nombres vaut : ', Prod );
```

```
        readln
 end.
```

Exercice 14 :
Les variables i et N sont de type entier. La variable N doit être affichée et lue en entrée.
Écrire un programme Pascal qui calcule et affiche en sortie la somme et le produit des N premiers nombres entiers (impaire) définie comme suit :

$$S = 1 + 3 + 5...$$

$$P = 1 * 3 * 5 * ...$$

Solution 14 :

```
Program Somme_Produit_Nombres_Impaires;
        Var k,N:integer;
            Prod,Som:real;
 begin
        write('entrez un entier N = ');
        readln(N);
        Som:=0;
        Prod:=1;
        k:=1;
        repeat
              Som:=Som+k;
              Prod:=Prod*k;
              k:=k+2;
        until k>N;
        writeln('la somme des ', N,' premiers nombres vaut : ', Som );
        writeln('le produit des ', N,' premiers nombres vaut : ', Prod );
        readln
 end.
```

Exercice 15 :
Les variables i et N sont de type entier. La variable N doit être affichée et lue en entrée.
Écrire un programme Pascal qui calcule et affiche en sortie la somme et le produit des N premiers nombres entiers (paire) définie comme suit :

$$S = 2 + 4 + 6...$$

$$P = 2 * 4 * 6 * ...$$

Solution 15 :

```
Program Somme_Produit_Nombres_Paires;
        Var k,N:integer;
            Prod,Som:real;
    begin
        write('entrez un entier N = ');
        readln(N);
        Som:=0;
        Prod:=1;
        k:=2;
        repeat
              Som:=Som+k;
              Prod:=Prod*k;
              k:=k+2;
        until k>N;
        writeln('la somme des ', N,' premiers nombres vaut : ', Som );
        writeln('le produit des ', N,' premiers nombres vaut : ', Prod );
        readln
   end.
```

Entraînement: reprendre ce programme en utilisant la boucle While.

Exercice 16 :

Écrire un programme Pascal qui calcule et affiche la somme alténative suivante:

$$S = 1 - 2 + 3 - 4 + ... + (-1)^{N+1}N = \sum_{i=1}^{N}(-1)^{i+1}i$$

Solution 16 :

```
program somme_alternative;
    var i,N,sign:integer;
        s:real;
 begin
    writeln('choisir une valeur pour N =');
    readln(N);
    s:=0;
    sign:=1;
    for i:=1 to N do begin
```

```
                    s:=s+sign*i;
                    sign:=-sign;
                    end;
   writeln('la somme s= ',s);
   readln
end.
```

Exercice 17 :
Écrire un programme Pascal qui calcule et affiche le factoriel d'un nombre entier N.

$$N! = N * (N-1) * (N-2) * ... * 2 * 1$$

Solution 17 : 1- Avec l'incrémentation du compteur:

```
Program factorielle;
   var i,N:integer;
       fact:real;
 begin
   writeln('choisir une valeur pour N =');
   readln(N);
   fact:=1;
   for i:=1 to n do begin
                    fact:=fact*i;
                    writeln (i,'! = ',fact);
                    end;
   writeln('N ! =',fact);
   readln
 end.
```

2- Avec la décrémentation du compteur:

```
Program factorielle;
    var i,N:integer;
       fact:real;
 begin
   writeln('choisir une valeur pour N =');
   readln(N);
   fact:=1;
   for i:=N downto 1 do begin
                        fact:=fact*i;
                        writeln (i,'! = ',fact);
                        end;
```

```
    writeln('N ! =',fact);
    readln
 end.
```

Exercice 18 :
Écrire un programme Pascal qui calcule et affiche la $N^{\text{ième}}$ puissance d'un nombre réel X qui doit être lu et affiché en entrée.

$$X^N = \underbrace{X * X * ... * X}_{N \text{ fois}}$$

Solution 18 :

```
Program puissance;
   var i,N,x:integer;
       puiss:real;
 begin
    writeln('choisir une valeur pour x =');
    readln(x);
    writeln('choisir une valeur pour N =');
    readln(N);
    puiss:=1;
    for i:=1 to N do begin
                     puiss:=puiss*x;
                     writeln('x a la puissance ',i, = ',puiss);
                     end;
    writeln('x a la puissance N = ',puiss);
    readln
 end.
```

Exercice 19 :
Étant donné X et N deux entiers lus en entrée, écrire un programme Pascal qui permet de calculer et afficher la somme S_1 où

$$S_1 = \ln(X+1) = X - \frac{X^2}{2} + \frac{X^3}{3} - \frac{X^4}{4} + ... + (-1)^{N+1}\frac{X^N}{N}$$

Solution 19 :

```
Program developpement_logarithme;
      var i,N,sign:integer;
          som,puiss,x:real;
 begin
```

```
    writeln('choisir une valeur pour x =');
    readln(x);
    writeln('choisir une valeur pour N =');
    readln(N);
    puiss:=1;
    sign:=1;
    som:=0;
    for i:=1 to N do begin
                     puiss:=puiss*x;
                     som:=som+sign*(puiss/i);
                     writeln('som = ',som);
                     sign:=-sign;
                     end;
   writeln('la somme s= ',s);
   readln
end.
```

Exercice 20 :

Étant donné X et N deux entiers lus en entrée, écrire un programme Pascal qui permet de calculer et afficher la somme S_2 où

$$S_2 = 1 - X + \frac{X^2}{2!} - \frac{X^3}{3!} + \frac{X^4}{4!} + ... + (-1)^N \frac{X^N}{N!}$$

Solution 20 :

```
Program somme_2;
      var i,N,sign,j,k,fact:integer;
          som,puiss,x:real;
 begin
    writeln('choisir une valeur pour x =');
    readln(x);
    writeln('choisir une valeur pour N =');
    readln(N);
    fact:=1;
    puiss:=1;
    sign:=-1;
    som:=1;
    for i:=1 to N do begin
                     fact:=fact*i;
                     puiss:=puiss*x;
                     som:=som+sign*(puiss/fact);
```

```
                    writeln('som = ',som);
                    sign:=-sign;
                    end;
  writeln('la somme s= ',s);
  readln
end.
```

Exercice 21 :

Étant donné X et N deux entiers lus en entrée, écrire un programme Pascal qui permet de calculer et afficher la somme S_3 où

$$S_3 = \cos X = 1 - \frac{X^2}{2!} + \frac{X^4}{4!} + ... + (-1)^N \frac{X^{2N}}{(2N)!}$$

Solution 21 :

```
Program developpement_cosinus;
       var i,N,sign,j,k,fact:integer;
           som,puiss,x:real;
 begin
    writeln('choisir une valeur pour x =');
    readln(x);
    writeln('choisir une valeur pour N =');
    readln(N);
    puiss:=1;
    sign:=-1;
    som:=1;
    for i:=1 to N do begin
                    puiss:=puiss*(x*x);
                    fact:=1;
                    k:=2*i;
                    for j:=1 to K do begin
                                     fact:=fact*j;
                                     writeln('fact= ',fact);
                                     end;
                    som:=som+sign*(puiss/fact);
                    writeln('som = ',som);
                    sign:=-sign;
                    end;
  writeln('la somme s= ',s);
  readln
end.
```

Série TP N^{o}3: Exercices avec solutions

Les deux premières séries de TP ont permis à l'étudiant, premièrement, d'acquérir les notions de base de la programmation et, deuxièmement, de résoudre certains problèmes à l'aide d'un programme numérique simple en faisant appel aux structures conditionnelles et itératives. Afin de tester et même de progresser les capacités de l'étudiant en programmation, nous avons proposé, dans cette troisième série, un ensemble de problèmes plus complexes qui nécessite l'utilisation des tableaux à une et à deux dimensions (vecteurs et matrices).

Exercice 22 :

Soient deux vecteurs réels V_1 et V_2 d'ordre 6. Les éléments de ces vecteurs doivent être affichés et lus (remplis) en entrée.

Écrire un programme Pascal qui calcule et affiche la somme des deux vecteurs $V_1 + V_2$.

Solution 22 :

```
program somme_deux_vecteurs;
   var V1,V2,V3:array[1..6] of real;
           i:integer;
   begin
     for i:=1 to 6 do begin
                      writeln('V1[',i,'] = ');
                      readln(V1[i]);
                      end;

     for i:=1 to 6 do begin
                      writeln('V2[',i,'] = ');
                      readln(V2[i]);
                      end;

     for i:=1 to 6 do
     V3[i]:=V1[i]+V2[i];

     for i := 1 to 6 do
     writeln('V3[',i,'] = ',V3[i]);
     readln
```

```
   end.
```

Exercice 23 :

Écrire un programme Pascal permettant de chercher et afficher l'élément maximum du vecteur V_1 et d'indiquer sa position.

Solution 23 :

```
program maximum;
 type tabular=array[1..6] of real;
 var V1:tabular;
    i,position:integer;
     max:real;
 begin
     for i:=1 to 6 do begin
                     writeln('V1[',i,'] = ');
                     readln(V1[i]);
                     end;

     max:=V1[1];
     position:=1;
     for i:=2 to 6 do
     if V1[i]>=max then begin
                       max:=V1[i];
                       position:=i;
                       end;
     writeln('L''élément maximum du vecteur = ',max);
     writeln('L''élément maximum se trouve à la case = ',position);
     readln
 end.
```

Exercice 24 :

Écrire un programme Pascal qui calcule le produit scalaire des deux vecteurs $V_1 * V_2$.

Solution 24 :

```
program produit_scalaire_deux_vecteurs;
  var V1,V2:array[1..6] of real;
       i:integer;
       scal:real;
  begin
     for i:=1 to 6 do begin
                     writeln('V1[',i,'] = ');
                     readln(V1[i]);
                     end;
```

```
    for i:=1 to 6 do begin
                    writeln('V2[',i,'] = ');
                    readln(V2[i]);
                    end;

    scal:=0;
    for i:=1 to 6 do
    scal:=scal+V1[i]*V2[i];

    writeln('Le produit scalaire = ',scal);
    readln
  end.
```

Exercice 25 :

Considérons deux matrices M_1 et M_2 de même ordre $(n \times p)$ dont les éléments sont de type entier. Écrire un programme pascal qui calcule et affiche la matrice $M_3 = M_1 + M_2$.

Solution 25 :

```
program somme_deux_matrices;
  const n=3;
        p=4;
  var M1,M2,M3:array[1..n,1..p] of real;
        i,j:integer;
  begin
    for i:=1 to n do
    for j:=1 to p do begin
                    writeln('M1[',i,',',j,'] = ');
                    readln(M1[i,j]);
                    end;
    for i:=1 to n do
    for j:=1 to p do begin
                    writeln('M2[',i,',',j,'] = ');
                    readln(M2[i,j]);
                    end;

    for i:=1 to n do
    for j:=1 to p do
                    M3[i,j]:=M1[i,j]+M2[i,j];


    for i:=1 to n do
```

```
    for j:=1 to p do
                    writeln('M3[',i,',',j,'] = ',V3[i,j]);

    readln
  end.
```

Exercice 26 :

Soient A et B deux matrices réelles d'ordre 3×4 et 4×5 respectivement. Les éléments de ces deux matrices doivent être affichés et lus en entrée.

Écrire un programme Pascal qui permet de calculer la transposée A^t de la matrice A.

Solution 26 :

```
program Matrice_Transpose; const n=3;
      p=4;
  var M:array[1..n,1..p] of real;
      Mt:array[1..p,1..n] of real;
      i,j:integer;
  begin
    writeln('Introduire les éléments de la matrice M');
    for i:=1 to n do
    for j:=1 to p do begin
                    writeln('M[',i,',',j,'] = ');
                    readln(M1[i,j]);
                    end;
    for i:=1 to n do
    for j:=1 to p do begin
                    Mt[j,i]:=M[i,j];
                    end;

    writeln('Affichage des éléments de la matrice transposée Mt');
    for i:=1 to p do
    for j:=1 to n do begin
                    writeln('Mt[',i,',',j,'] = ',Mt[i,j]);
                    end;
    readln
  end.
```

Exercice 27 :

1- Écrire un programme Pascal qui calcule la moyenne arithmétique suivante:

$$Moy = \overline{X} = \sum_{i=1}^{N} X_i / N$$

telles que les X_i sont des variables réelles qui doivent être affichées et lues en entrée.
2- Étendre le programme précédent pour calculer les grandeurs statistiques connues telles que : la variance et les moyennes géométrique, harmonique et quadratique.

Solution 27 :

```
program Moyenne_variance;
  const N=10;
  var Note:array[1..N] of real;
      i:integer;
      S1,S2,S3,S4,a,Variance,h,g,q:real;
  begin
     for i:=1 to N do begin
                      writeln('Note[',i,'] = ');
                      readln(Note[i]);
                      end;
{*calcul de la moyenne arithmétique*}
     S1:=0;
     for i:=1 to N do
                      S1:=S1+Note[i];
                      a:=S1/N;
     writeln('moyenne arithmétique : ',a);

{*calcul de la Variance*}
     for i:=1 to N do
                      Variance:=sqr(Note[i]-a)/N;
     writeln('La Variance = ',Variance);

{*calcul de la moyenne géométrique*}
     S2:=0 ;
     for i:=1 to N do
                      S2:= S2+ln(note[i]) ;
                      g:=exp(S2/N) ;
     writeln('moyenne géométrique : ',g) ;
{*calcul de la moyenne harmonique*}
     S3:=0;
     for i:=1 to N do
                      S3:=S3+1/note[i];
                      h:=N/S3;
     writeln('moyenne harmonique : ',h) ;
{*calcul de la moyenne quadratique*}
     S4:=0;
     for i:=1 to N do
```

```
                    S4:=S4+note[i]*note[i];
                    q:=sqrt(S4/N);
   writeln('moyenne quadratique : ',q);
   readln
end.
```

Liste de Références

L. AMIROUCHE et S. KERIREM, " *Introduction à l'Algorithmique*" (2009).

S. GRAINE,"*Algorithmique et structures de donnée*", les éditions l'Abeille (2001).

D. MOKHTARI, " *Le Pascal Facile*", Edition HAMDI, Algérie (1999).

E. MORVANT, "*Algorithmique et programmation Pascal*", Saint-Louis (2009).