Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Seddik Benyahia, Jijel
Faculté des Science et de la Technologie
Département d'Electronique

- Support de cours -

Informatique 1

Réalisé par :

Dr. Mohammed TAMOUM

Destiné aux étudiants de première année licence Sciences et Technologie

Année Universitaire
2017/2018

DOMAINE SCIENCES ET TECHNOLOGIE	**PROGRAMME "Informatique1"** Volume horaire semestriel 45h00 Volume horaire hebdomadaire 3h00 (1H30 cours et 1h30 TP) Semestre 1 - 15 semaines-	1ère ANNEE SOCLE COMMUN Coef : 02 Crédits : 04

Programme	Nombre de semaines
Chapitre 1 : Introduction à l'informatique 1- Définition de l'informatique 2- Evolution de l'informatique et des ordinateurs 3- Les systèmes de codage des informations 4- Principe de fonctionnement d'un ordinateur 5- Partie matériel d'un ordinateur 6- Partie système Les systèmes de base (syst. d'exploitation : Windows, Linux, Mac OS,...) Les langages de programmations les logiciels d'application	05
Chapitre 2 : Notions d'algorithme et de programme 1- Concept d'un algorithme 2- Représentation en organigramme 3- Structure d'un programme 4- Démarche et analyse d'un problème 5- Structure des données Constantes et variables Types de données 6- Les opérateurs L'opérateur d'affectation Les opérations arithmétiques Les opérateurs relationnels Les opérateurs logiques Les priorités dans les opérations 7- Les opérations d'entrée/sortie 8- Les structures de contrôle Les structures de contrôle conditionnel Les structures de contrôle répétitives	07
Chapitre 3 : Les variables Indicées 1- Les tableaux unidimensionnels Représentation en mémoire Operations sur les tableaux 2- Les tableaux bidimensionnels Représentation en mémoire Opérations sur les tableaux bidimensionnels	03

Chapitre 1 : Introduction à l'informatique

1. Définition de l'informatique

Le mot **INFORMATIQUE** est composé de 2 mots : **INFORMA**tion et automa**TIQUE**.

L'informatique est la science du traitement de l'information automatiquement utilisant une machine appelée : **Ordinateur**.

On a 2 types d'information essentielle :

Les données (data) : qu'on doit saisir à l'ordinateur pour en avoir des résultats.

Les instructions (commandes) : ce sont les opérations effectuées par 1 ordinateur.

2. Evolution de l'informatique et des ordinateurs

4 générations :

1945-1957 : Première génération : Tubes à vide et Commutateurs

1958-1964 : Deuxième génération : Transistors

1965-1980 : Troisième génération : Circuits Intégrés

1980-ce jour : Quatrième génération : Le microprocesseur

3. Les systèmes de codage des informations

Dans un système de base B, tout nombre entier positif N peut se décomposer de la forme suivante :

$N = a_nB^n + a_{n-1}B^{n-1} + ... + a_2B^2 + a_1B^1 + a_0B^0 = \sum_{i=0}^{n} a_i.B^i$

avec $0 \leq a_i \leq B-1$ (chiffres)

ou encore : $N = a_n\, a_{n-1}.....a_2a_1a_0$ (notation condensée)

Exemples :

$(6123)_{10} = 6x10^3 + 1x10^2 + 2x10^1 + 3x10^0$

$(10101)_2 = 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0$

En informatique les systèmes les plus utilisés sont les suivants :

Système	Base	Symboles
Décimal	10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Binaire	2	0, 1
Octal	8	0, 1, 2, 3, 4, 5, 6, 7
Hexadécimal	16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

Les informations traitées par l'ordinateur sont toujours représentées sous forme binaire car ce système peut correspondre à deux états physiques parfaitement distincts.

Le courant passe ⇨ 1

Le courant ne passe pas⇨ 0

Une information élémentaire correspond donc à un chiffre binaire 0 ou 1 appelé **BIT** (**B**inary dig**IT**s).

Un mot de 8 bits est appelé octet (byte) : unité utilisée pour mesurer la capacité d'une mémoire (l'information). On utilise souvent des multiples :

- 1 Kilo-octet = 1 Ko = 2^{10} octets = 1 024 octets ≈ 10^3 octets
- 1 Méga-octet = 1 Mo = 2^{20} octets = 1 048 576 ≈ 10^6 octets
- 1 Giga-octet = 1 Go = 2^{30} octets = 1 073 741 824 ≈ 10^9 octets
- 1 Tera-octet = 1 To = 2^{40} octets = 1 099 511 627 776≈ 10^{12} octets
- 1 Peta-octet = 1 Po = 2^{50} octets = 1 125 899 906 842 624 ≈ 10^{15} octets

Les conversions :

Pour nous qui raisonnons en système décimal, il se pose donc un problème de conversion entre les différentes bases

Problème fondamental : le changement de base.

Il y a deux méthodes à retenir :

Conversion : *base B - système décimal* :

On développe le nombre selon les puissances de la base B.

Exemple 1 : Binaire ⇨ Décimal

$(1010)_2 = 1x2^3 + 0x2^2 + 1x2^1 + 0x2^0$

$= 8 + 2$

$= (10)_{10}$

Exemple 2 : Hexadécimal ⇨ Décimal

$(A24)_{16} = Ax16^2 + 2x16^1 + 4x16^0$

$= 10x16^2 + 2x16^1 + 4x16^0$

$= (2596)_{10}$

Exemple 3 : Base 5 ⇨ Décimal

$(40132)_5 = 4x5^4 + 0x5^3 + 1x5^2 + 3x5^1 + 2x5^0$

$= 2500 + 0 + 25 + 15 + 2$

$= (2542)_{10}$

Conversion : *décimal - base B* :

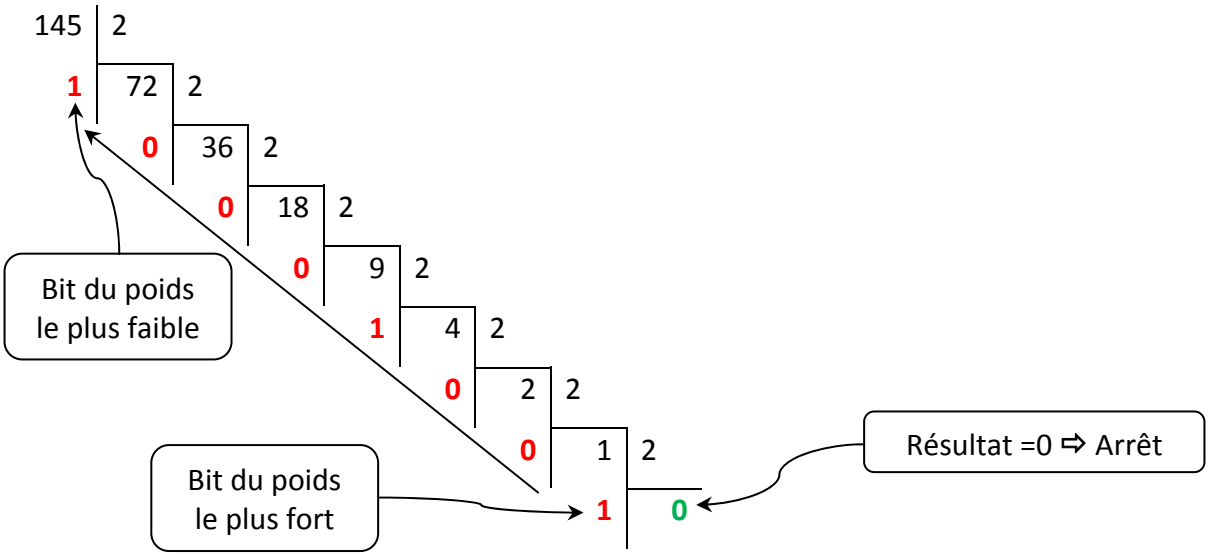
On applique le principe de la division euclidienne successive :

N = B x Q + R avec 0 ≤ R < B

On fait des divisions des quotients successifs par B jusqu'à l'obtention d'un résultat nul.

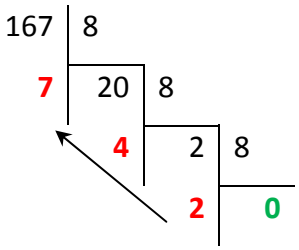
La liste **inversée** des restes ainsi obtenus constitue la décomposition recherchée.

Exemple 1 : Décimal ⇨ Binaire



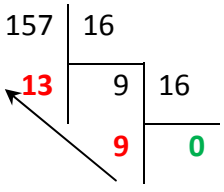
Ainsi, on a : $(145)_{10} = (10010001)_2$

Exemple 2 : Décimal ⇨ Octal



Ainsi, on a : $(167)_{10} = (247)_8$

Exemple 3 : Décimal ⇨ hexadécimal



Donc : $(157)_{10} = (9D)_{16}$

Le passage d'une base B vers une autre base Q (B et Q $\neq$ 10) ⇨ 2 manières :

- On peut utiliser la base 10 comme base intermédiaire.
- Si B est une puissance de Q ($B = Q^m$) ⇨ utiliser la technique de regroupement de bits par groupes de m bits.

Il est recommandé de bien connaître la correspondance des 16 premiers nombres dans les quatre bases.

Décimal	Binaire	Octal	Hexadécimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	1 0000	20	10

Octal ⇔ Binaire : (en base 8 = 2^3 ⇨ 3 bits)

- Regrouper les bits du nombre binaire en groupes de 3 en partant de la droite, si le dernier groupe ne contient pas 3 bits, ajouter des ZEROs.
- Trouver l'équivalent octal de chaque groupe de 3 bits.

Exemples :

$(011\ 000\ 111)_2$ ⇨ $(307)_8$

$(110\ 010\ 101\ 001\ 011\ 100)_2$ ⇨ $(625134)_8$

$(72)_8$ ⇨ $(111\ \mathbf{0}10)_2$

Le zéro en **rouge et gras** a été ajouté car $(2)_8$ en binaire ne comporte que deux chiffres.

Hexadécimal ⇔ Binaire

> *Cette fois-ci, chaque chiffre du nombre en base hexadécimale peut être représenté par un nombre de 4 chiffres en binaire*

On regroupe cette fois-ci les bits (binaires) par 4, car nous sommes en base 16, et $16 = 2^4$ ⇨ 4 bits.

On complète à gauche par des zéros si nécessaire.

Exemples :

$(B5E)_{16} \Rightarrow (1011\ 0101\ 1110)_2$

$(1100\ 0111)_2 \Rightarrow (C7)_{16}$

$(11\ 1000\ 1110\ 1111\ 1001\ 1010)_2 \Rightarrow (38EF9A)_{16}$

$(1\ 11\ 00\ 10\ 01)_2 \Rightarrow (13021)_4$

Octal ⇔ Hexadécimal

Dans ce cas, il est plus simple de passer par la base binaire, puis de reconvertir dans la base désirée, plutôt que d'utiliser la division successive.

Exemples :

$(307)_8 \Rightarrow (011\ 000\ 111)_2 = (1100\ 0111)_2 \Rightarrow (C7)_{16}$

$(333)_{16} \Rightarrow (0011\ 0011\ 0011)_2 = (001\ 100\ 110\ 011)_2 \Rightarrow (1463)_8$

Ainsi, on convertit chaque chiffre octal en un nombre binaire de 3 bits (conversion octal ⇔ binaire), puis on regroupe les bits par 4, pour passer en hexa (conversion binaire ⇔ hexa).

Exemples de différentes conversions

$(13021)_4 \Rightarrow (1\ 11\ 00\ 10\ 01)_2 = (111\ 001\ 001)_2 \Rightarrow (711)_8$

$= (1\ 1100\ 1001)_2 \Rightarrow (1C9)_{16}$

$(13021)_4 = (?)_{16}$

$16 = 4^2 \Rightarrow$ On utilise des groupes de 2 chiffres.

$\Rightarrow (1\ 30\ 21)_4 = (1C9)_{16}$

$(112102)_3 = (?)_9 = (?)_{27}$

$9 = 3^2 \Rightarrow$ On utilise des groupes de 2 chiffres.

$\Rightarrow (11\ 21\ 02)_3 = (472)_9$

$27 = 3^3 \Rightarrow$ On utilise des groupes de 3 chiffres.

$\Rightarrow (112\ 102)_3 = (EB)_{27}$

Base 4	Base 16
00	0
01	**1**
02	2
03	3
10	4
11	5
12	6
13	7
20	8
21	**9**
22	A
23	B
30	**C**
31	D
32	E
33	F

Base 3	Base 9
00	0
01	1
02	**2**
10	3
11	**4**
12	5
20	6
21	**7**
22	8

Base 3	Base 27
000	0
001	1
002	2
010	3
011	4
012	5
020	6
021	7
022	8
100	9
101	A
102	**B**
110	C
111	D
112	**E**
120	F

Base 3	Base 27
121	G
122	H
200	I
201	J
202	K
210	L
211	M
212	N
220	O
221	P
222	Q

4. Principe de fonctionnement d'un ordinateur

Les systèmes informatiques comportent deux composants :

- Le Hardware (le matériel)
- Le Software (le logiciel) (programme).

Le logiciel est l'ensemble des programmes rendant utilisable le matériel.

4.1. Hardware (Partie matériel d'un ordinateur)

Le hardware est la partie qui représente les composants matériels avec lesquels est construit un ordinateur. L'ordinateur comporte les éléments (unités) suivants :

a. Unité centrale : elle regroupe :

- *Le processeur ou microprocesseur* : Il représente le cerveau de l'ordinateur, car c'est lui qui se charge réellement du traitement de l'information. Il contient :
  - L'Unité Arithmétique et Logique (UAL)
  - L'Unité de Commande et de Contrôle (UCC)
- *La mémoire centrale* : Qui permet de stocker l'information (les programmes et les données) durant le traitement.

b. Les périphériques :

Ce sont les accessoires nécessaires pour faire entrer et recevoir les informations. On cite principalement ; le clavier, la souris, le scanner, l'écran, l'imprimante, les haut-parleurs, …

Périphériques d'Entrée *Périphériques de Sortie*

c. Les unités d'échange (ou d'entrée/sortie) :

Ce sont tous les composants qui permettent de véhiculer l'information entre les différents éléments de l'ordinateur. Elles sont représentées essentiellement par un ensemble de registres, de bus et d'autres circuits (cartes ; graphique, son, …).

d. Supports de stockage :

Pour stocker l'information d'une manière durable (exemple : disque dur, CD, DVD, flash disk, …)

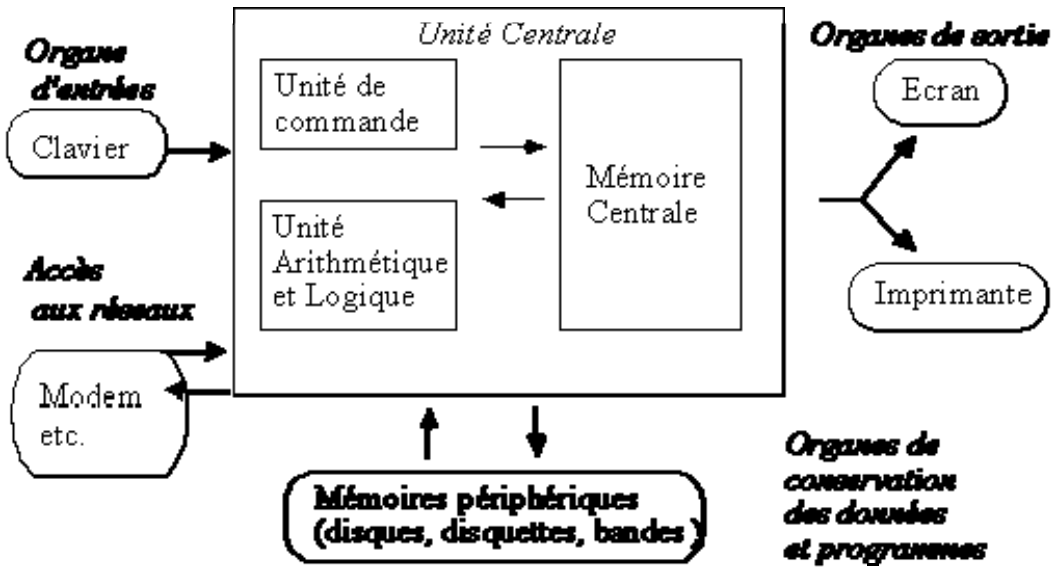


Figure I.1 : Schéma de base d'un ordinateur

4.2. Software (Partie système)

*On appelle **Software** la partie logique de l'ordinateur, elle est constituée de tous les programmes qui permettent de faire fonctionner la machine ainsi que ceux qui permettent à l'utilisateur d'exploiter l'ordinateur selon ses besoins*

Un programme est une série d'instructions écrites dans un langage que l'ordinateur comprend.

Le terme logiciel est une généralisation du terme programme.

Un logiciel peut regrouper plusieurs programmes.

L'ensemble de ces programmes peut être repartie en 2 sous ensembles :

- Les programmes de base.
- Les programmes d'application.

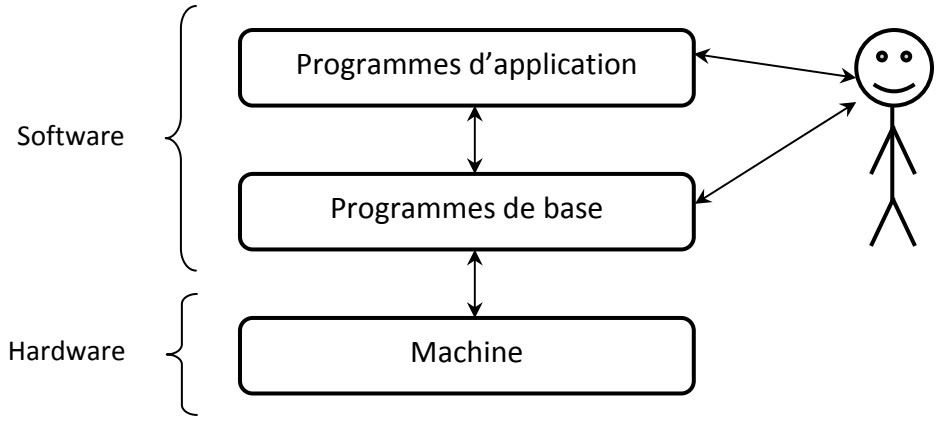


Figure I.2 : Parties d'un système informatique

a. Les programmes de base (appelés systèmes d'exploitation) : puisque c'est grâce à eux que la machine est exploitable.

Ils servent d'intermédiaire entre l'utilisateur et la machine d'une part, et entre les programmes d'application et la machine d'autre part. Les systèmes les plus connus sont : Windows, DOS, Unix, Linux, Mac OS, ...

b. Les programmes d'application

Ce sont les programmes qui permettent de répondre aux besoins de l'utilisateur. Ils accomplissent des tâches bien précises.

On peut citer comme exemple :

- Microsoft Word : qui est un logiciel de traitement de texte
- Microsoft Excel : qui est un tableur (pour le calcul)
- Microsoft Access : qui est un gestionnaire de bases de données (SGBD)
- Pascal, Delphi, C++, Fortran, VB, Java, … : langages de programmation
- Photoshop : qui est logiciel de création et de retouche (traitement) d'image
- McAfee, Avira, BitDefender, Avast, Kaspersky, … : antivirus
- Solitaire, Freecell, dame de pique, … : jeux

Chapitre 2 : Notions d'algorithme et de programme

1. Concept d'un algorithme

L'algorithme est une description d'un traitement automatisé de données destiné à être réalisé sur un ordinateur, après avoir traduit cette description dans un langage de programmation. (Description en langage naturel de la suite des actions effectuées par un programme) (Description d'une suite d'actions à effectuer, dans un ordre donné, pour parvenir un résultat) (Suite finie d'instructions permettant de donner la réponse à un problème)

Exemple :

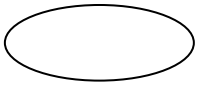
1. Ouvrir la porte du garage
2. Prendre la clef
3. Ouvrir la porte avant gauche
4. Entrer dans la voiture
5. Mettre au point mort
6. Mettre le contact
7. Débrayer
8. Enclencher la marche arrière
9. Desserrer le frein à main
10. Embrayer doucement
11. …

Pour résoudre un sujet d'examen *:*

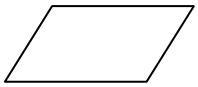
1. *Lire tout le sujet*
2. *Choisir l'exo le plus facile*
3. *Cas de Pb, passer à l'exo suivant jusqu'à la fin*
4. *Puis revenir aux autres exo et leurs donner le tp qu'il faut*
5. *Fin*

- *Recette culinaire*
- *Montrer le chemin à quelcun*
- *Résoudre une équation du $2^{ème}$ degré*

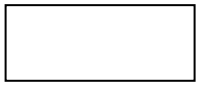
2. Représentation en organigramme : c'est la traduction graphique de l'algorithme.



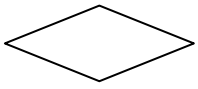
Début et fin



E/S (Ecriture/Lecture)



Action (ou instruction)



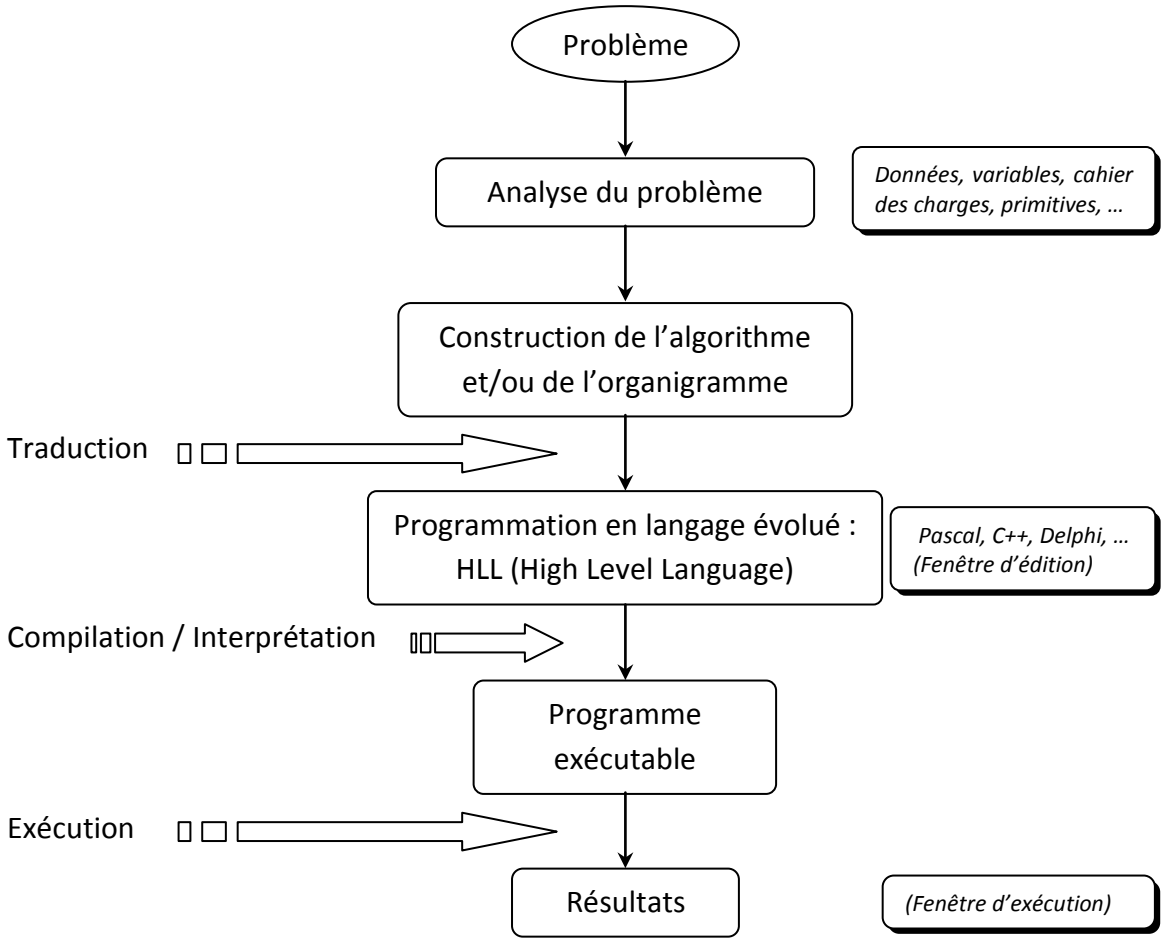
Condition (ou question) (test)

Direction (sens)

Il est plus clair que l'algo, mais dès que l'algo se complique, il devient très difficile et même impossible.
Il est parfois appelé Algorigramme ou Ordinogramme

3. Démarche et analyse d'un problème

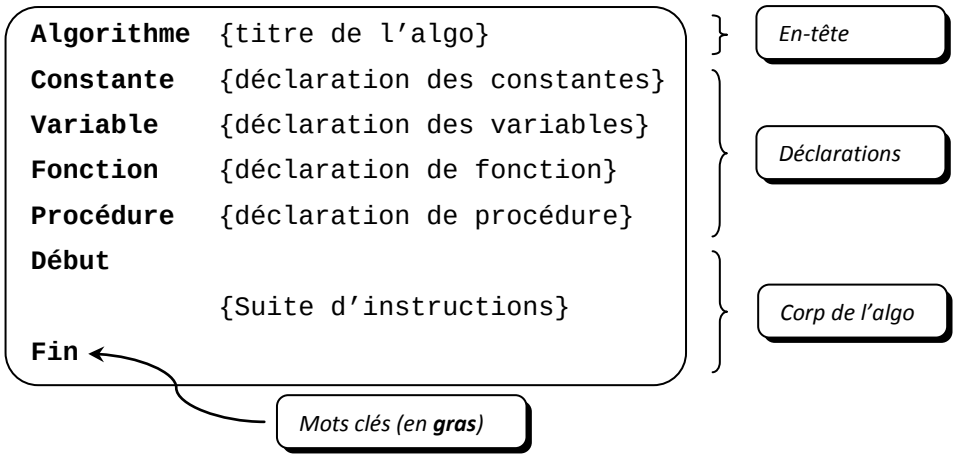
Pour résoudre un problème par ordinateur, il faut passer par les étapes suivantes :



Résolution d'un problème par ordinateur

4. Structure d'un programme

Voici la structure générale d'un algorithme :



Et voici celle d'un programme Pascal :

```
Program     {nom du programme} ;
Uses        {unités (ou bibliothèques) utilisées} ;
Const       {déclarations des constantes} ;
Type        {déclaration des types} ;
Function    {déclaration de fonction} ;
Procedure   {déclaration de procédure paramétrée} ;
Var         (* déclaration des variables *) ;
Procedure   (* déclaration de procédure simple *) ;
Begin
            {Instructions du programme principal} ;
End.
```

NB : L'ordre indiqué doit être impérativement respecté.

- **Uses** : Indique qu'on utilise des fonctions et outils contenus dans d'autres fichiers de programme (nommés des unités) que celui dans lequel on travaille.
- **Type** : Cette clause sert à définir des types de données particulières (par exemple : les jours de la semaine, les couleurs, …).

*Attention : Le mot réservé **Type** ne doit être écrit qu'une seule fois.*

*Par exemple : Pour pouvoir utiliser la procédure « **Clrscr** » mettre en début de programme la commande : **Uses crt** ; Sous Windows : **Uses wincrt** ;*
***Clrscr** : procédure sans paramètre ; efface le contenu de l'écran et positionne le curseur au coin supérieur gauche.*
***Procedure** et **Function**: On verra plus tard. L'essentiel est de savoir que ces procédures et fonctions sont situées AVANT le bloc principal **Begin/End**.*

Grammaire du Pascal :

- Un identificateur (nom d'une constante, variable, fonction, programme, …) est un mot composé de lettres, de chiffres et du caractère « _ » et commençant obligatoirement par une lettre.
- Un identificateur ne peut être égal à un mot réservé du langage **Pascal** ! Les mots réservés (mots clés) varient d'un compilateur à l'autre (**Turbo Pascal, Free Pascal**, **GnuPascal**, **Delphi**, . . .) et d'une version à l'autre.
- Les identificateurs ne doivent pas excéder **127** signes (selon compilateur) (1 lettre au minimum).

Exemple : LMD, D45, ST_2, aaa1.

- **Turbo Pascal** ne différencie aucunement les majuscules des minuscules. Ainsi les identificateurs *Ma_Variable* et *ma_variable* sont considérés comme équivalents.
- Un programme principal débute toujours par **Begin** et se termine par **End.** (avec un point). Alors qu'un sous-programme (fonction, procédure, bloc conditionnel...) commence lui aussi par **Begin** mais se termine par **End ;** (sans point mais avec un *point-virgule*).

- Chaque commande doit se terminer avec un *point-virgule* (**;**). Il n'y a pas d'exception à la règle hormis **Begin** et l'instruction précédent **End** ou **Else**.
- Il est toléré de mettre plusieurs instructions les unes à la suite des autres sur une même ligne du fichier mais il est recommandé de n'en écrire qu'une par ligne : c'est plus clair et en cas de bogue, on s'y retrouve plus aisément. De plus, s'il vous arrive d'écrire une ligne trop longue, le compilateur vous le signifiera en l'erreur « Error 11: Line too long ». Il vous faudra alors effectuer des retours à la ligne comme le montre l'exemple suivant :

```
WriteLn('Fichier: ', file,
' Date de création:', datecrea,
' Utilisateur courant:', nom,
' Numéro de code:', Round(ArcTan(x_enter)*y_old):0:10) ;
```

- Les commentaires sont des éléments du texte d'un programme qui sont ignorés par le compilateur. Ils servent à apporter des renseignements sur le programme.

  En **Pascal**, il y a trois façons d'écrire des commentaires :

  1. commentaires sur une seule ligne : ils débutent par **//**

     ```
     // Commentaire sur une s e u l e    l i g n e
     ```

  2. commentaires sur plusieurs lignes : encadrés par **{** et **}**

     ```
     { commentaire sur
     p l u s i e u r s
     l i g n e s }
     ```

  3. commentaires sur plusieurs lignes : encadrés par **(*** et ***)**

     ```
     (* commentaires
     sur p l u s i e u r s
     l i g n e s *)
     ```

N'hésitez pas à insérer des commentaires dans votre code, cela vous permettra de comprendre vos programmes un an après les avoir écrit, et ainsi d'autres personnes n'auront aucun mal à réutiliser vos procédures, fonctions ... Vous pouvez aussi mettre en commentaire une partie de votre programme.

5. Structure des données

5.1. Types de données

Un programme peut être amené à manipuler plusieurs types de données :

Types simples (de base) :

- ***Booléen*** : Valeur pouvant être soit « Vraie » soit « Fausse ».
- ***Entier*** : Valeur numérique entière pouvant être signée ou non signée. (5,-20, …)
- ***Réel*** : Valeur numérique codée avec Mantisse et Exposant ($M.10^E$). ($1,3.10^{20}$, $1,6.10^{-19}$, …)
- ***Caractère*** : Octet correspondant à un code ASCII. ("A", "m", "@", "5", …)
- ***Chaînes de caractères*** : Ensemble de caractères. ("Mohammed", "Salam", …)

Types composés (complexes)

Tableau de données : Ensemble fini de données de même type. (Matrices)

(2, 7, 11)

$\begin{pmatrix} 1,3 \\ 3,8 \\ 7,2 \end{pmatrix}$

"a"	"#"
"4"	"Z"

Enregistrement (Structure) : Ensemble de données de types différents correspondant à un même objet. (Etudiant : Nom, Prénom, Age, Filière, Adresse, Moyenne, Admi, …)

Les types les plus utilisés en **Pascal** sont :

- ***Integer*** (nombres entiers).
- ***Real*** (nombres réels).
- ***Char*** (caractère) permet de représenter les caractères. Les constantes de type caractère s'écrivent entre apostrophe :'A', '3' ou '*'.
- ***String*** (chaînes de caractères).
- ***Boolean*** prend deux valeurs possibles: *TRUE* et *FALSE*.

Petite liste-exemple très loin d'être exhaustive : (Annexe)

Désignation	Description	Bornes	Place en mémoire
Real	nombres réels (avec 11 décimales)	$2.9\ 10^{-39}$ **et** $1.7\ 10^{+38}$	6 octets
Single	réel	$1.5\ 10^{-45}$ **et** $3.4\ 10^{+38}$	4 octets
Double	réel (avec 15 décimales)	$5.0\ 10^{-324}$ **et** $1.7\ 10^{+308}$	8 octets
Extended	réel	$1.9\ 10^{-4951}$ **et** $1.1\ 10^{+4932}$	10 octets
Comp	réel	$-2\ 10^{+63}+1$ **et** $2\ 10^{+63}+1$	8 octets
Integer	nombres entiers (sans virgule)	-32 768 **et** 32 767	2 octets
Longint	entier	-2 147 483 648 **et** 2 147 483 647	4 octets
Shortint	entier	-128 **et** 127	1 octet
Word	entier	0 **et** 65 535	2 octets
Byte	entier	0 **et** 255	1 octet
Long	entier	$(-2)^{31}$ **et** $(2^{31})-1$	4 octets
Boolean	variable booléenne (valeurs logiques)	TRUE **ou** FALSE	1 octet
Array [1..10] Of xxx	tableau de 10 colonnes fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
Array [1..10, 1..50, 1..13] Of xxx	tableau en 3 dimensions fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
String	chaîne de caractères		256 octets
String [y]	chaîne de caractère ne devant pas excéder y caractères		y+1 octets
Text	fichier texte		
File	fichier		
File Of xxx	fichier contenant des données de type xxx (Real, Byte...)		
Char	nombre correspondant à un caractère ASCII codé	0 et 255	1 octet
Pointeur	adresse mémoire		4 octets
Datetime	format de date		
Pathstr	chaîne de caractère (nom complet de fichier)		
Dirstr	chaîne de caractère (chemin de fichier)		
Namestr	chaîne de caractère (nom de fichier)		
Extstr	chaîne de caractère (extension de fichier)		

5.2. Constantes et variables

a. Constante : Donnée manipulée par un programme et ne pouvant pas être modifiée.

Les constantes se déclarent comme suit :

En **algorithmique** :

```
Constante   <Nom_Constante> = <Valeur>
```

En **Pascal** :

```
Const   <Identificateur> = <Valeur> ;
```

L'utilisation de constantes en programmation est vivement conseillée. Elles permettent :

- une notation plus simple (*Exemple* **: Pi** à la place de **3.141592653**).
- la possibilité de modifier simplement la valeur spécifiée dans la déclaration au lieu d'en rechercher les occurrences, puis de modifier dans tout le programme.

Exemple :

```
Const       pi = 3.141592653 ;                  {type numérique}
            DEUX = 2 ;
            VRAI = true;                        {type booléen}
            FAUX = false;
            CAR_A  = 'A';                       {type caractère}
            PHRASE = 'Salamo 3alikom';    {type chaîne de caractères}
            LARG = 640 ;
            HAUT = 480 ;
            NB_PIX = LARG*HAUT ;
```

Il est possible de déclarer plusieurs constantes.
NB : Ne pas oublier le point virgule à la fin de chaque déclaration.

b. Variable : Donnée manipulée par un programme et pouvant être modifiée. Ce peut être :

- Une donnée d'entrée (exemple : A, B, C)
- Un résultat final de calcul (exemple : X)
- Un résultat intermédiaire de calcul (exemple : Δ)

Equation du 2ème degré : $A.X^2+B.X+C=0$

Déclarer une variable c.-à-d. ; Allouer un espace mémoire (réserver une case mémoire et lui donner un nom).

Les variables se déclarent comme suit :

En **algorithmique** :

```
Variable   <Nom_Variable> : <Type>
```

En **Pascal** :

```
Var   <Identificateur> : <Type> ;
```

NB :
- ***Var*** *ne peut apparaître qu'une seule fois.*
- *Possibilité de grouper plusieurs variables pour le même type (séparation des variables par une virgule).*

Exemple :

```
Var
      x : Real ;
      i, j, n : Integer ;
      FLAG : Boolean ;
      s , t : String ;
      C1 , C2 : Char ;
```

6. Les opérateurs

6.1. Les opérateurs arithmétiques

Addition	**+**
Soustraction	**-**
Multiplication	*****
Division réelle	**/**

Div : renvoie le quotient de la division entière x **Div** y
Mod : renvoie le reste de la division entière x **Div** y

- *Pour les entiers : **+, -, *, Div, Mod,***
- *Pour les réels : **+, -, *, /,***

6.2. Les opérateurs relationnels

Egalité	**=**
Différent	**<>**
Inférieur strict	**<**
Inférieur ou égale	**<=**
Supérieur strict	**>**
Supérieur ou égale	**>=**

Ayant un résultat booléen

6.3. Les opérateurs logiques

Not : le "non"
And : le "et" logique des maths
Or : le "ou" logique
Xor : le "ou" exclusif

Pour les booléens.
*Aussi ; **Nand, Nor***

Tableau de vérité

A	B	Not A	Not B	A And B	A Or B	A Xor B	A Nand B	A Nor B
F	F	V	V	F	F	F	V	V
F	V	V	F	F	V	V	V	F
V	F	F	V	F	V	V	V	F
V	V	F	F	V	V	F	F	F

6.4. Les priorités dans les opérations

*En mathématiques une question se pose avec un calcul tel que **2 + 3 * 4** : quel est le résultat de ce calcul ?*
*Est-ce **20** (calcul **2 + 3 = 5** puis multiplication par **4**) ou **14** (ajouter **2** à **3 * 4**) ?*
Le même problème se pose en programmation. Il en existe deux solutions :

- Dans le cas d'une expression comportant des parenthèses, les parenthèses sont naturellement considérées en premier.
- En l'absence de parenthèses, des règles de priorités interviennent. Celles propres à **Pascal** sont :
  a. L'évaluation des fonctions est l'opération la plus prioritaire.
  b. Les opérateurs dits multiplicatifs (*** / Div Mod …**) sont plus prioritaires que les opérateurs dits additifs (**+ - …**).
  c. En cas de même priorité, l'expression est évaluée de gauche à droite.

Niveaux de priorité des opérateurs :

Niveau 1 : **Not**.
Niveau 2 : *****, **/**, **Div**, **Mod**, **And**.
Niveau 3 : **+**, **-**, **Or**, **Xor**.
Niveau 4 : **=**, **<**, **>**, **<=**, **>=**, **<>**.

Exemples :

`2+3*4`	donne **14** car la sous-expression **3*4** est d'abord évaluée (règle **b**: priorité de * par rapport à **+**)
`(2+3)*4`	**2+3** donne **5**, multiplié par **4** donne **20**
`2+3+4*2*5`	donne **45** (**4 * 2** puis multiplier par **5** donne **40**, puis **2** + **3** donne **5**, ensuite ajouter **40** à **5**)
`2+(3+4*2)*5`	donne **57** (**4 * 2** puis ajouter **3** donne **11**, multiplier par **5** puis ajouter **2**)
`4*Sin(π/2)*5.0`	calcul de **Sin (π/2)** puis multiplication par **4**, ensuite par **5.0** (règle **a**) (ça donne **20.0**)
`2*3+4 Div 3`	donne **7** car la sous-expression **2 * 3** est d'abord évaluée (règles **b** puis **c**), puis la sous-expression **4 Div 3** l'est (règles **b** puis **c**) enfin l'addition des résultats partiels **6** et **1** est effectuée
`(2*3+4)Div 3`	donne **3** car la sous-expression **2 * 3** est d'abord évaluée (règles **b**), puis ajouter **4** donne **10** (règles **b**), enfin **10 Div 3** est effectuée
`2*(3+4)Div 3`	donne **4** car la sous-expression **3 + 4** est d'abord évaluée donne **7**, puis multiplier **7 * 2** l'est (règles **b** puis **c**) donne **14** enfin **14 Div 3** est effectuée
`2*(3.0+4)Div 3`	donne **ERREUR** car la sous-expression **3.0 + 4** est d'abord évaluée donne **7.0**, puis multiplier **7.0 * 2** l'est (règles **b** puis **c**) donne **14.0** on ne peut pas effectuer l'opération **14.0 Div 3** (**14.0** est un réel)
`20/2*3`	**30.0** (Réel)

6.5. L'opérateur d'affectation

Une affectation consiste à attribuer une valeur à une variable.

La syntaxe générale est la suivante :

En **algorithmique** :

```
Nom_Variable ⇦ <Expression>
```

En **Pascal** :

```
<Identificateur> := <Expression> ;
```

<Expression> peut être :

- Une valeur constante — *exp* : **`surface := 40 ;`**
- Une autre variable — *exp* : **`Donnee := ValeurMemorisee ;`**
- Le résultat d'une fonction — *exp* : **`Resultat := Sqrt(Nombre) ;`**
- Un calcul portant sur ces différents éléments — *exp* : **`Surface := (Pi*Sqr (D))/4 ;`**

Exemples :

```
Algorithme Affectation
Variable A, B : Entier
Début
A ⇦ 1
B ⇦ A + 3
A ⇦ 3
Fin
```

A	B
1	?
1	4
3	4

Trace de l'Algorithme

Trace *: l'exécuter instruction par instruction (le faire tourner à la main) L'ordre des instructions est primordial.*

```
Début
A ⇦ "Salam"
B ⇦ "Merci"
Fin
```

Chaîne de caractère. ***B*** *contient « M e r c i »*

```
Début
A ⇦ "Salam"
B ⇦ Merci
Fin
```

Autre variable. ***B*** *contient le contenu de la variable* ***Merci****.*

7. Les opérations d'entrée/sortie

Permettent à la machine de dialoguer avec l'utilisateur.

7.1. Sortie (Ecriture, Affichage)

C'est l'instruction d'affichage et s'utilise comme suit :

En **algorithmique** :

```
     Ecrire (Nom_Variable)
Ou   Ecrire ("<Message>")
```

Ecrire *s'appelle aussi* ***Afficher****.*

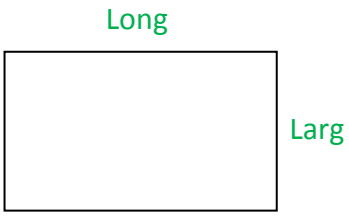
En **Pascal** :

```
     Write (<Identificateur>) ;
Ou   Write (<Valeur>) ;
Ou   Write ('<Message>') ;
Ou   Write ('<Message>',<Identificateur>) ;
```

Exemples :

```
Write('Bienvenue à tous') ;
Write ( 10 ) ;      {écrit la valeur 10 à l'écran}
Write (10, 4) ; {écrit la valeur 10 suivie de la valeur 4 sur l'écran}
Write ( 10 + 4 ) ; {écrit la valeur de l'expression, à savoir 14, sur l'écran}
Delta := 7 ;
Write('la valeur de Δ est',delta) ;
Write ('L''age de l''étudiant est 20 ans.') ;
Write(delta) ;
Write('delta') ;
```

Exemple : Ecrire un programme (Pascal et Algorithme) pour le calcul du périmètre d'un rectangle de longueur 12 cm et de largeur 7 cm.



En **algorithmique** :	En **Pascal** :
Algorithme Rectangle **Variable Long, Larg, Peri : Réel** **Début** **Long ⇦ 12** **Larg ⇦ 7** **Peri ⇦ (Long + Larg) * 2** **Ecrire (Peri)** **Fin**	**Program Rectangle ;** **Var Long, Larg, Peri : Real ;** **Begin** **Long := 12 ;** **Larg := 7 ;** **Peri := (Long + Larg) * 2 ;** **Write (Peri) ;** **End.**

Pour améliorer la présentation, nous fournissons des textes écrits entre guillemets " " (en Algo) ou entre apostrophes ' ' (en Pascal) ; comme :

```
Ecrire ("Le périmètre du rectangle est ", Peri, "cm")
```

Son exécution produit la ligne suivante :

```
Le périmètre du rectangle est 38 cm
```

- *Si le texte à afficher contient une apostrophe, il faut alors la doubler (en Pascal).*
- *Les différents noms de variables doivent êtres séparés par des virgules.*
- *L'identificateur est remplacé par la valeur de la variable correspondante au moment de l'exécution de l'instruction.*
- *La valeur peut être une constante ou une expression.*

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

Exemple :

```
WriteLn ('Texte avec renvoi à la ligne') ;
```

***Remarque** : Les arguments d'une instruction **WriteLn** sont des expressions.*

7.2. Entrée (Lecture, Saisie)

*L'algorithme précédent pourrait être adapté à d'autres rectangles (d'autres **Long** et **Larg**) il est plus pratique d'utiliser l'action **Lire** qui permet de choisir la valeur à affecter à une variable au moment de l'exécution de l'algorithme.*

C'est l'instruction de saisie et s'utilise comme suit :

En **algorithmique** :

```
Lire (Nom_Variable)
```

En **Pascal** :

```
Read (<Identificateur>) ;
Ou   Read (V1, V2, V3, …) ;
```

Dès que le programme rencontre une instruction **Read**, l'exécution s'interrompt, attendant la frappe d'une valeur au clavier.

Remplaçons, dans notre programme, les 2 premières lignes par :

```
Read (Long) ;
Read (Larg) ;
```

Pour le confort de l'utilisateur, il est conseillé de précéder chaque instruction **Read** d'une instruction **Write** qui le prévient que le programme attend une valeur, et lui précise laquelle. Exemple :

```
Write ("Veuillez fournir la longueur du rectangle en cm")
Read (Long) ;
```

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

NB :

- **Readln(...) ;** ➔ passage à la ligne suivante en ignorant ce qui reste sur la ligne.
- **Readln ;** peut être employé sans paramètre.

Exemples :

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

*L'équivalent de la commande **ReadLn** est **ReadKey** qui donne une valeur à une variable de type « Char » (caractère **ASCII**).*

Syntaxe :

```
x := ReadKey ;
```

*Il existe une équivalence à cette commande très utile pour sortir d'une boucle : **KeyPressed**.*

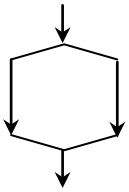
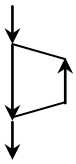
Syntaxe :

```
Repeat
...
commandes
...
Until KeyPressed ;
```

Exemple : Ecrire un programme pour le calcul de la surface et du périmètre d'un cercle. (Pour la maison)

```
Program Surface_Perimetre_Cercle ;
Const Pi=3.14159 ;
Var Rayon, Surface, Peri : Real ;
Begin
Write ('Entrer le rayon du cercle R = ') ;
Read (Rayon) ;
Surface := Pi * Sqr (Rayon) ; {Ou Pi * Rayon * Rayon}
Peri := Pi * 2 * Rayon ;
Writeln ('La surface du cercle est ', Surface) ;
Writeln ('Le périmètre du cercle est ', Peri) ;
End.
```

8. Les structures de contrôle (structures ou primitives algorithmiques)

Séquencement linéaire	*Structures conditionnelles (ou de choix)*	*Structures répétitives (ou itératives) (Boucles)*
		

8.1. Les structures de contrôle conditionnelles (ou de choix)

En fonction d'une condition, le programme exécute des opérations différentes.

8.1.1. L'alternative (Le test - Le choix simple - La sélection - If ... Then ... Else ...)

Syntaxe en **algorithmique** :

```
Si Condition Alors
      Instruction1(s)
Sinon
      Instruction2(s)
FinSi
```

Syntaxe en **Pascal** :

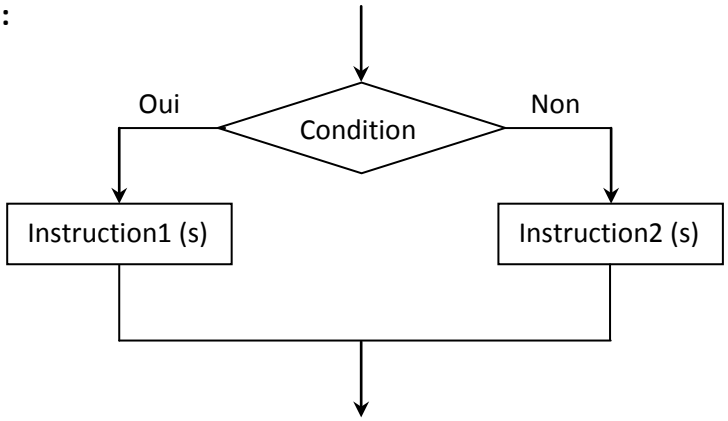
```
If Condition(s) Then Instruction1(s) Else Instruction2(s) ;
```

Ou

```
If Condition Then
      Instruction1(s)
Else
      Instruction2(s) ;
```

Si la condition est *VRAIE*, l'instruction **Then** est exécutée, *Sinon* l'instruction **Else** est exécutée si elle existe.

L'organigramme :



Remarques :

- Surtout pas de point virgule immédiatement avant le **Else** !!! Le **`If ... Then ... Else ...`** est une unique instruction composée, sans "**;**" au milieu.
- L'instruction "test" peut se limiter à **If ... Then** ; si le test est négatif alors aucune instruction ne sera effectuée et la suite du programme sera lue … (on peut omettre le **Else**).
- S'il y a plus d'une instruction à exécuter (instruction composée : *suite d'instructions*), il faut les encadrer par "**Begin**" et "**End**" :

```
If condition Then
      Begin
            Instructions ;
            ...
            Instructions ;
            Instructions
      End
Else
      Begin
            Instructions ;
            ...
            Instructions ;
            Instructions
      End;
```

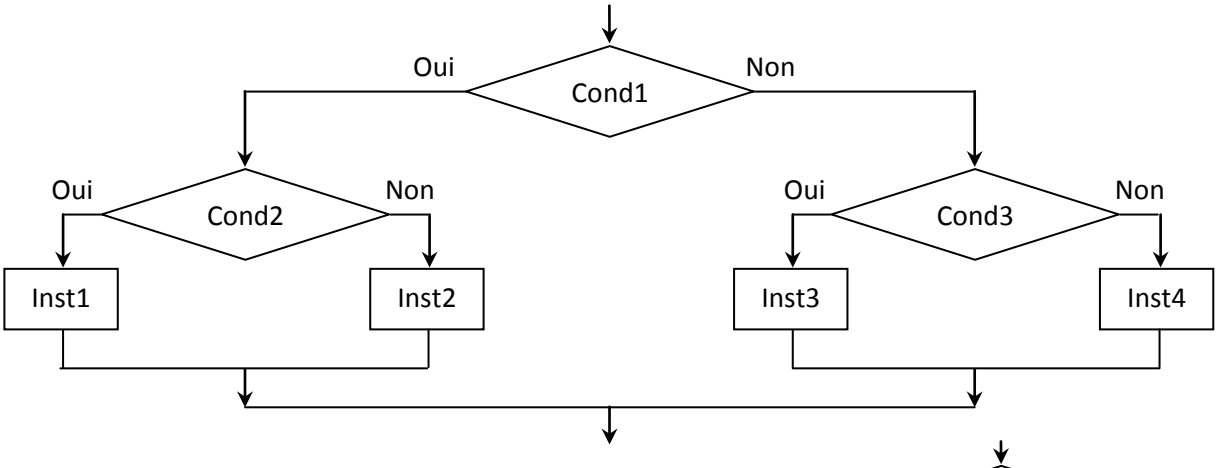
Exemple :

```
If (a>b) And (b>c) Then
Begin
      Writeln('premier :', a) ;
      Writeln('deuxième :', b) ;
      Writeln('troisième :', c)
End ;
```

On peut imbriquer plusieurs "**If**". Utiliser des indentations (marges décalées) pour une meilleure présentation.

Exemple :

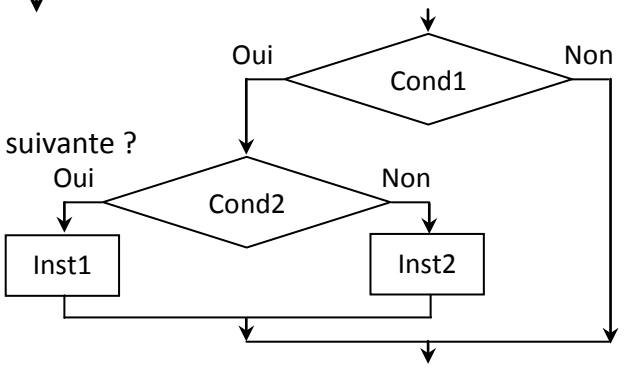
```
If Cond1 Then If Cond2 Then Inst1        {Cond1 et Cond2}
                  Else Inst2             {Cond1 et pas Cond2}
      Else If Cond3 Then Inst3           {pas Cond1 mais Cond3}
                  Else Inst4 ;           {ni Cond1 ni Cond3}
```



Remarque :

Quel est l'effet de la partie de programme suivante ?

```
If Condition_1 Then
      If Condition_2 Then
            Instruction_1
      Else Instruction_2 ;
```

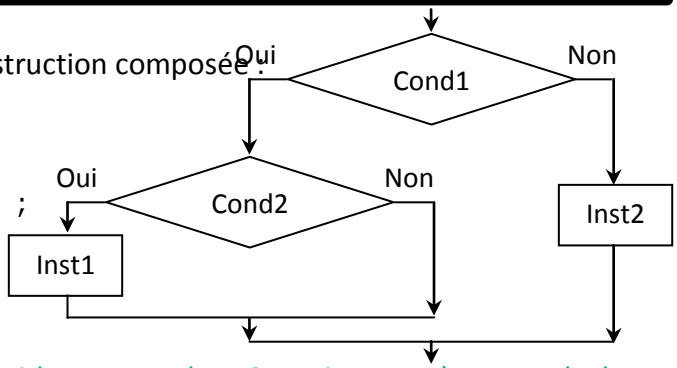


Cette situation est ambiguë. Par convention, la partie **Else** correspond à l'instruction **If** *la plus proche*, ici **If** condition_2 **Then …**

> *Un **Else** se rapporte toujours au dernier **Then** rencontré auquel un **Else** n'a pas encore été attribué.*

Le contraire s'obtient à l'aide d'une instruction composée :

```
If Condition_1 Then
Begin
      If Condition_2 Then
            Instruction_1 ;
End
Else Instruction_2 ;
```



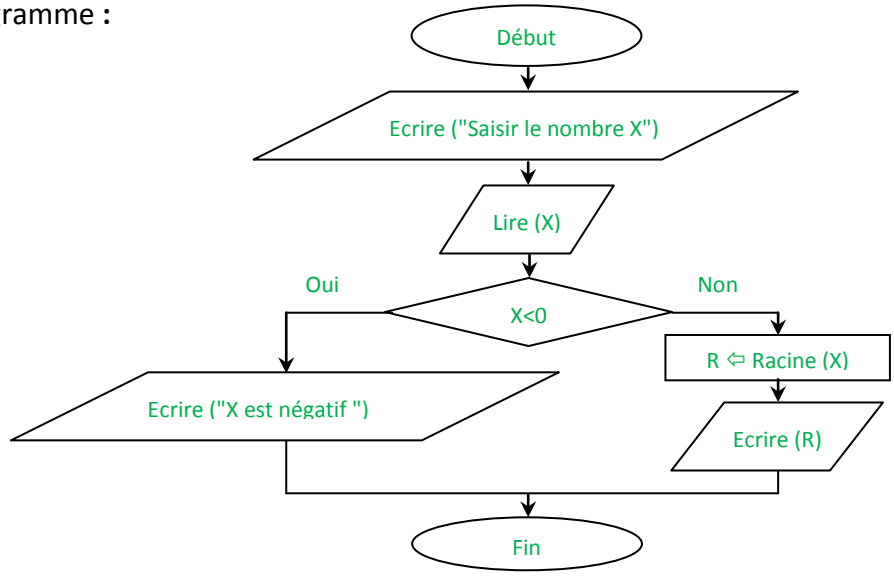
Exemple : Ecrire un programme (Algorithme, Pascal et Organigramme) pour calculer et afficher la racine carrée.

En **algorithmique** :	En **Pascal** :

`Algorithme Racine_Carree`
`Variable X, R : Réel`
`Début`
`  Ecrire ("Saisir le nombre X")`
`  Lire (X)`
`  Si X<0 Alors`
`  Ecrire ("X est négatif")`
`  Sinon`
`  R ⇦ Racine (X)`
`  Ecrire (R)`
`  FinSi`
`Fin`
	`Program Racine_Carree ;`
	`Var X, R : Real ;`
	`Begin`
	`  WriteLn ('Saisir le nombre X') ;`
	`  ReadLn (X) ;`
	`  If X<0 Then`
	`  WriteLn ('X est négatif')`
	`  Else`
	`  Begin`
	`  R := Sqrt (X) ;`
	`  WriteLn (R) ;`
	`  End ;`
	`End.`

Pour les nombres complexes et si X<0, on affiche : `Write ('i', Sqrt (-X)) ;`

L'organigramme **:**



Exemple : (Pour la maison)

Ecrire un programme devant donner l'état de l'eau selon sa température, doit pouvoir choisir entre 3 réponses possibles : Solide, Liquide ou Gazeuse.

En **algorithmique** :	En **Pascal** :

`Algorithme Temperature_Eau`
`Variable Temps : Entier`
`Début`
`  Ecrire ("Entrer la température de l'eau")`
`  Lire (Temps)`
`  Si Temps ≤ 0 Alors`
`  Ecrire ("C'est de la glace")`
`  Sinon`
`  Si Temps < 100 Alors`
`  Ecrire ("C'est du liquide")`
`  Sinon`
`  Ecrire ("C'est de la vapeur")`
`  FinSi`
`  FinSi`
`Fin`
	`Program Temperature_Eau ;`
	`Var Temps : Integer ;`
	`Begin`
	`  WriteLn ('Entrer la température de l''eau') ;`
	`  ReadLn (Temps) ;`
	`  If Temps <= 0 Then`
	`  WriteLn ('C''est de la glace')`
	`  Else`
	`  If Temps < 100 Then`
	`  WriteLn ('C''est du liquide')`
	`  Else`
	`  WriteLn ('C''est de la vapeur') ;`
	`End.`

8.1.2. La structure de choix, Cas - Parmi (Sélection Multiple : le Case...Of)

Une donnée est comparée successivement à des valeurs constantes :

*Souvent, avec les types énumérés, on veut faire un traitement qui dépend de la valeur. Pour cela, on peut bien sûr utiliser des instructions **If**, mais cela a tendance à rendre le programme plus lourd que nécessaire. Lorsqu'on veut faire un traitement par cas, il y a une instruction beaucoup plus pratique, l'instruction **Case**.*

*On parle de **type énuméré** pour désigner un type dans lequel on peut donner tous les éléments les uns à la suite des autres. Pour l'instant, on en a vu deux, le type **Integer** et le type **Char**, mais on en verra d'autres dans la suite du cours*

Syntaxe en **algorithmique** :

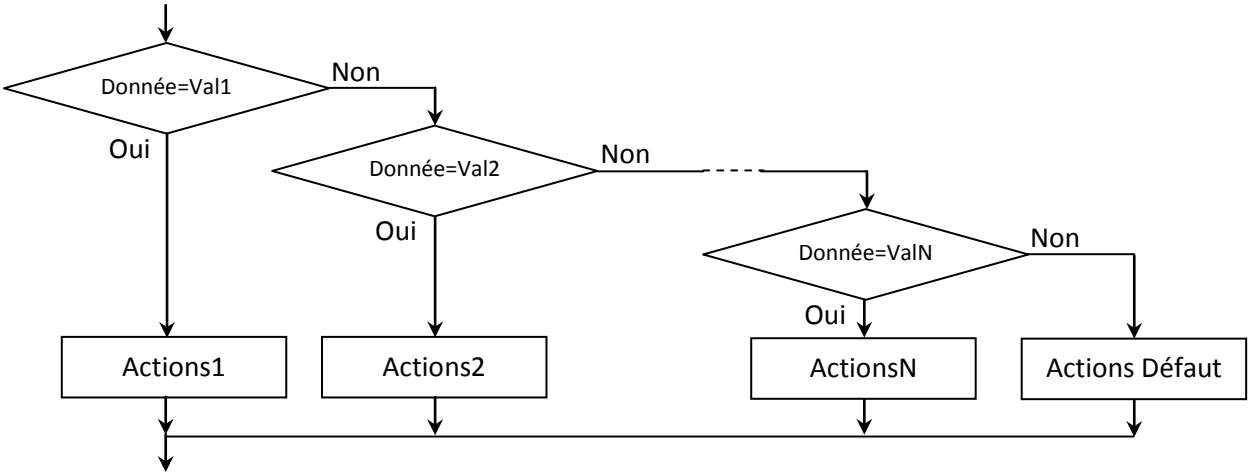
```
Cas où Donnée Vaut
   Valeur1 : Actions1
   Valeur2 : Actions2
   ...
   ValeurN : ActionsN
   Autre : Actions défaut
FinCas
```

Syntaxe en **Pascal** :

```
Case <Expression> Of
   Valeur 1 : Instruction(s) 1 ;
   Valeur 2 : Instruction(s) 2 ;
   ...
   Valeur n : Instruction(s) n ;
   Else Instruction(s) par défaut ;
End ;
```

*En **Turbo Pascal**, on accepte une **Valeur** particulière qui est **Else** (et doit être placée en dernier), pour prévoir le cas où Expression n'appartient à aucun des cas cités au dessus. En **Ms-Pascal** on utilise de même **Otherwise**.*

L'organigramme **:**



Remarques :

- La partie « Action Défaut » peut ne pas exister.
- Plusieurs valeurs différentes peuvent être regroupées sur une même ligne si les actions correspondantes sont identiques.
- S'il y a plusieurs instructions, on doit les encadrer par « **Begin** » et « **End ;** ».

*Opportunité du **Case** :*
- *Si structures **If... Then... Else...** imbriquées.*
- *Si égalités testant la valeur d'une expression.*
- *Très utile pour la **lisibilité du programme** !!!*

***Case** peut être remplacée par **If imbriquée**.*
L'inverse n'est possible que si les tests concernent une seule variable, et que les tests concernant cette variable est soit une égalité soit un intervalle.

Exemple :

Affichage de la nature d'un caractère.

En **algorithmique** :	En **Pascal** :

`Algorithme Nature_Caract`
`Variable C : Caractère`
`Début`
`  Ecrire ("Tapez un caractère")`
`  Lire (C)`
`  Cas Où C Vaut`
`  "A".."Z" : Ecrire ("Lettre majuscule")`
`  "a".."z" : Ecrire ("Lettre minuscule")`
`  "0".."9" : Ecrire ("Chiffre")`
`  Autre : Ecrire ("Symbole")`
`  FinCas`
`Fin`
	`Program Nature_Caract ;`
	`Var C : Char ;`
	`Begin`
	`  WriteLn ('Tapez un caractère') ;`
	`  ReadLn (C) ;`
	`  Case C Of`
	`  'A'..'Z' : WriteLn ('Lettre majuscule') ;`
	`  'a'..'z' : WriteLn ('Lettre minuscule') ;`
	`  '0'..'9' : WriteLn ('Chiffre')`
	`  Else WriteLn ('Symbole') ;`
	`  End ;`
	`End.`

*Attention : Là, le point-virgule avant le **Else** est **FACULTATIF**. Mais pour plus de sécurité afin de ne pas faire d'erreur avec le bloc **If**, choisissez systématiquement d'omettre le point-virgule avant un **Else**.*

Exemple 2 :

```
Program case_age;
Var age:Integer;
Begin
      Write('Entrez votre âge : ') ;
      Readln(age) ;
      Case age Of
            18 : Writeln('18 ans');
            0..17 : Writeln('Petits enfants...') ;
            60..99 : Writeln('Les vieux')
            Else Writeln('Vous êtes d''un autre âge...') ;
      End ;
End.
```

Note : On peut effectuer un test de plusieurs valeurs en une seule ligne par séparation avec une virgule si on souhaite un même traitement pour plusieurs valeurs différentes. Ainsi la ligne :

```
0..17 : Writeln('Petits enfants …') ;
```

Peut devenir :

```
0..10, 11..17 : Writeln('Petits enfants …') ;
```

Ou encore :

```
0..9, 10, 11..17 : Writeln('Petits enfants …') ;
```

Ou même :

```
0..17, 5..10 : Writeln('Petits enfants …') ;
```

{Cette dernière ligne est stupide mais correcte !}

8.2. Les structures de contrôle répétitives (ou itératives) (Boucles)

Souvent un algorithme pose une question à laquelle l'utilisateur doit répondre par « OUI » ou par « NON » et propose un modèle de réponse, par exemple : "Voulez-vous conserver ces résultats ? O/N". Lorsque la réponse est incorrecte (elle n'est ni O ni N), elle doit être rejetée, et la question doit être posée à nouveau.

On pourrait essayer avec un « SI » comme suit :

```
Algorithme Question
Variable Rep : Caractère
Début
  Ecrire ("Voulez-vous un livre ? O/N")        .     .     .     (1)
  Lire (Rep)     .     .     .     .     .     .     .     .     (2)
  Si (Rep ≠"O") et (Rep ≠"N") Alors.     .     .     .     .     (3)
    Ecrire ("Vous devez répondre par O ou N")  .     .     .     (4)
    Lire (Rep)   .     .     .     .     .     .     .     .     (5)
  FinSi    .     .     .     .     .     .     .     .     .     (6)
Fin
```

Ainsi, lorsque la réponse n'est ni O ni N, un message d'erreur apparait.

Quoi faire ?

*Reproduire le même « **SI** », après le second **Lire (Rep)**, permettrait de régler une deuxième erreur, mais pas une troisième … L'algorithme ne devrait passer à la suite que lorsque la variable **Rep** contient, soit la lettre « **O** », soit la lettre « **N** ».*

La bonne solution consiste à utiliser une structure itérative (boucle).

Une itération consiste à exécuter un bloc d'instructions un certain nombre de fois. Dans la plupart des cas, on ne connait pas le nombre de répétitions.

Nous allons examiner différentes manières de faire répéter une séquence d'instructions, appelée « corp de la boucle ».

8.2.1. La boucle (structure) "While ... Do" (TantQue ... Faire)

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est **VRAI**.

Syntaxe en **algorithmique** :

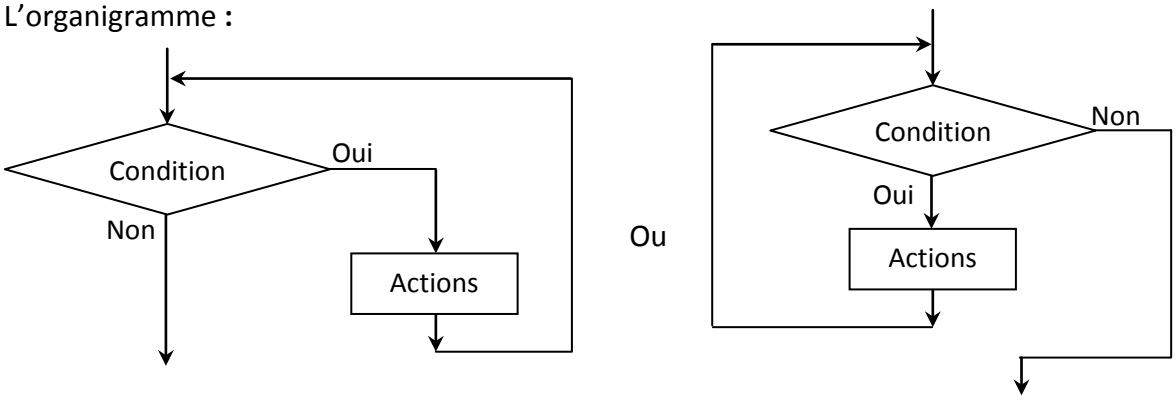
```
TantQue Condition Faire
      Actions
FinTantQue
```

Séquence (bloc d'instructions)

Syntaxe en **Pascal** :

```
While <Condition> Do
      <Instruction> ;
```

L'organigramme **:**



Lorsque le corps de la boucle contient plus d'une instruction (comme pour l'instruction **IF**) il faut l'encadrer par les termes « **Begin** » et « **End ;** ».

Illustration avec notre exemple :

Une première approximation de la solution consiste à écrire :

```
Algorithme Question2
Variable Rep : Caractère
Début
   Ecrire ("Voulez-vous un livre ? O/N")        .     .     .     (1)
   TantQue (Rep ≠"O") et (Rep ≠"N") Faire .     .     .     .     (2)
      Lire (Rep)   .     .     .     .     .     ..    .     .     (3)
   FinTantQue     .     .     .     .     .     .     .     .     (4)
Fin
```

C'est le squelette de l'algorithme correct (manque les muscles les organes pour qu'il fonctionne correctement).
Son principal défaut est de provoquer une erreur à chaque exécution :
*La variable **Rep** n'a pas été affectée avant l'entrée dans la boucle.*
On teste donc une variable qui n'a pas de valeur, ce qui provoque une erreur et l'arrêt immédiat de l'exécution.
*Pour éviter ceci, Il faut que la variable **Rep** ait déjà été affectée avant qu'on en arrive au premier tour de boucle*
*⇨ **2** possibilités :*
***1ère Possibilité** : On peut faire une première lecture da la variable **Rep** avant la boucle. Dans ce cas, celle-ci ne servira qu'en cas de mauvaise saisie lors de cette première lecture. On ajoute dans l'algorithme précédent entre les lignes **(1)** et **(2)** :*

```
Lire (Rep)
```

Erreur ⇨ La variable **Rep** n'a pas été affectée avant l'entrée dans la boucle.

2ème Possibilité : (fréquemment employée) consiste à affecter arbitrairement la variable avant la boucle. Provoquer l'entrée obligatoire dans la boucle.

L'affectation doit donc faire en sorte que la condition soit mise à **VRAI** pour déclencher le 1er tour de la boucle.

Dans notre exemple, on peut donc affecter **Rep** avec n'importe quelle valeur à part « **O** » et « **N** ».

On ajoute par exemple entre les lignes **(1)** et **(2)** l'instruction :

```
Rep ⇦ "X"
```

Remarque : Il existe une différence importante dans les structures logiques des 2 possibilités.
*Pour la 1ère possibilité, la boucle sera exécutée uniquement dans l'hypothèse d'une mauvaise saisie initiale. Si l'utilisateur saisit une valeur correcte à la 1ère demande de **Rep**, l'algorithme passera sur la boucle sans entrer dedans.*
Avec la 2ème solution (possibilité), l'entrée de la boucle est forcée, et l'exécution de celle-ci au moins une fois est rendue obligatoire à chaque exécution du programme.

On peut ajouter des affichages de libellés qui font encore un peu défaut. L'algorithme devient :

```
Algorithme Question2
Variable Rep : Caractère
Début
   Rep ⇦ "X"
   Ecrire ("Voulez-vous un livre ? O/N")
   TantQue (Rep ≠"O") et (Rep ≠"N") Faire
      Ecrire ("Vous devez répondre par O ou N")
      Lire (Rep)
   FinTantQue
   Ecrire ("Saisie acceptée")
Fin
```

Traduction en Pascal** : On doit encadrer les 2 instructions de **TantQue** par **Begin** et **End ;

*Si on ajoute dans la boucle (par exemple avant **FinTantQue**) l'instruction :*

```
Rep ⇦ "X"
```

⇨ Boucle INFINIE

Remarque (Conclusion) :

Si la 1ère évaluation de la condition fournit la valeur **FAUX**, le corps de la boucle n'est exécuté (le programme ne rentre jamais dans la boucle). Et si la séquence (les actions) ne change pas la valeur de la condition et si celle-ci a la valeur **VRAI**, la séquence sera répétée sans que l'on passe jamais à la suite : boucle Infinie, il est donc indispensable de prendre les 2 précautions suivantes :

- Initialiser les variables qui interviennent dans la condition avant d'aborder la boucle.
- S'assurer que la séquence est capable de faire évoluer la valeur de la condition vers la valeur **FAUX**, ce qui permettra l'arrêt de la boucle.

8.2.2. La boucle "Repeat ... Until" (Répéter... Jusqu'à)

Dans une boucle « **TantQue** », la condition est évaluée avant d'exécuter la 1ère instruction du bloc contenu dans la boucle (séquence).

On peut souhaiter dans certains cas, évaluer la condition après l'exécution de la dernière instruction de ce bloc. Un tel algorithme correspond à une boucle « **Répéter... Jusqu'à** ».

Syntaxe en **algorithmique** :

```
Répéter
      Actions
Jusqu'à Condition
```

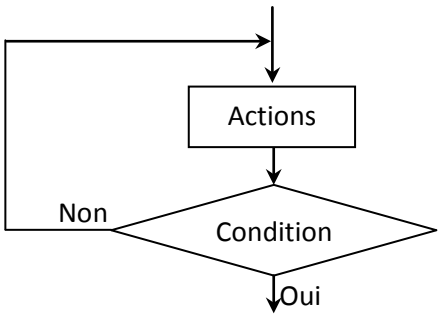
Syntaxe en **Pascal** :

```
Repeat
      <Instruction(s)> ;
Until <Condition> ;
```

Ou

```
Repeat < Instruction1; ... ; InstructionN > ; Until < Condition > ;
```

L'organigramme **:**



Remarques :

1. Ici, même si le corps de la boucle contient plus d'une instruction, il n'est pas nécessaire de l'encadrer par « **Begin** » et « **End ;** ».
2. Le bloc d'instruction sera répété jusqu'à ce que la condition soit vérifiée.
3. Au contraire de la boucle « **TantQue** », l'utilisation de la boucle « **Répéter... Jusqu'à** » garantit que le bloc sera exécuté **au moins une fois** puisque le test a lieu après son exécution.

4. La signification de la condition n'est plus la même :
   - Avec « **TantQue** » ⇨ Condition pour continuer la répétition (test d'entrée).
   - Avec « **Répéter... Jusqu'à** » ⇨ Condition d'arrêt (test de sortie).

Note :
*La commande **Inc** permet d'incrémenter une variable d'une certaine valeur. La commande **Dec** permet au contraire de décrémenter une variable d'une certaine valeur. Ces commandes permettent d'éviter la syntaxe :*

Variable := Variable + 1 et **Variable := Variable - 1.**

Syntaxe :

```
Inc (Variable, Nombre) ;
Dec (Variable, Nombre) ;
```

Remarque :
*On utilise généralement les instructions **While** ou **Repeat** lorsque l'on ne connaît pas, à l'avance, le nombre d'itérations.*

Exemple 1 : Refaire l'exemple précédent en utilisant « **Répéter... Jusqu'à** ».

*Avec **Répéter ... Jusqu'à** :*

```
Algorithme Question3
Variable Rep : Caractère
Début
   Ecrire ("Voulez-vous un livre ? O/N")
   Répéter
      Ecrire ("Vous devez répondre par O ou N")
      Lire (Rep)
   Jusqu'à Rep ="O" ou Rep ="N"
   Ecrire ("Saisie acceptée")

Fin
```

Traduction en Pascal

Exemple 2 : Répéter la multiplication de 2 nombre.

```
Algorithme Multiplication
Variable    a, b, P : Réel
            c : Caractère
Début
   Répéter
      Ecrire ("Saisir les nombres a et b")
      Lire (a,b)
      P ⇦ a*b
      Ecrire ("Le produit est",P)
      Ecrire ("Encore un calcul ? NON touche N ; OUI autre touche")
      Lire (c)
   Jusqu'à c = "N"

Fin
```

*Avec **TantQue** :*

```
c ⇦ "k"
TantQue c≠"N" Faire
...
```

8.2.3. La boucle "For" (Pour)

Il est pratique d'utiliser la boucle « **Pour** » si le nombre de passage dans la boucle est connu au préalable, une variable particulière **i** appelée **Variable d'itération ou de contrôle** (indice) sert à compter le nombre de répétitions du bloc d'instructions.

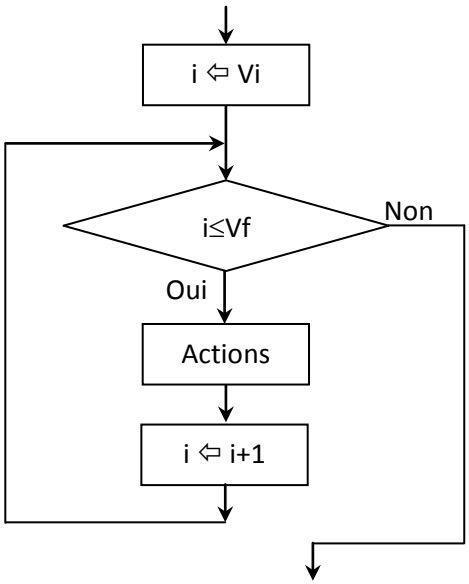
Syntaxe en **algorithmique** :

```
Pour i = Vi à Vf Faire
      Actions
FinPour
```

Syntaxe en **Pascal** :

```
For i := Vi To Vf Do
      Instruction(s) ;
```

L'organigramme **:**



Vi : Valeur initiale

Vf : Valeur finale

i : indice (variable d'itération, variable de contrôle, compteur) : Entier, on peut utiliser un autre caractère.

Remarques :

1. Le nombre de passage dépond de **Vi** et **Vf** (le nombre de passage = **Vf - Vi + 1**).
2. Si **Vi** > **Vf**, le bloc d'instructions n'est jamais exécuté.
3. S'il y a plus d'une instruction à exécuter, il faut les encadrer par **Begin** et **End :**

```
For i := Vi To Vf Do
Begin
      Instructions ;
End ;
```

4. La boucle « **Pour** » ne convient pas à l'écriture d'un algorithme de lecture de réponse **OUI** ou **NON**.
5. La variable **i** peut être non seulement de type entier, mais aussi de tout type sur lequel est défini un ordre, par exemple : le type caractère dans lequel le suivant de "**A**" est "**B**", etc.
6. La variable d'itération **i** change automatiquement de valeur à chaque itération (s'incrémente), la valeur de l'incrément par défaut est égale à 1. Certains langages autorisent la définition d'un pas d'itération différent de 1. Il suffit de le préciser dans l'algorithme, comme le montre l'exemple suivant :

```
Pour i = Vi à Vf Pas de P Faire
      Actions
FinPour
```

7. La valeur du pas **P** peut être négative ; dans ce cas **Vi>Vf**.

```
Ou
Pour i = Vi en descendant jusqu'à Vf Faire
            Actions
FinPour
```

8. E Pascal, lorsque la *valeur initiale* est supérieure à la *valeur finale*, le **To** doit être remplacé par **Downto** comme suit (syntaxe) **:**

```
For i:= Vi Downto Vf Do
      Instruction(s);
```

Exemple 1 : Affichage d'une ligne d'étoiles (80 étoiles).

En **algorithmique** :	En **Pascal** :

Algorithme Etoiles
Variable i : Entier
Début
  Pour i=1 à 80 Faire
  Ecrire ("*")
  FinPour
Fin
	Program Etoiles ;
	Var i : Integer ;
	Begin
	  For i :=1 to 80 Do
	  Write ('*') ;
	End.

Exemple 2 :

```
Program Compteur2 ;
Var n : Integer ;
Begin
      For n:=1 To 10 Do
            Writeln(n,' a pour triple ',3*n)
End.
```

Exemple 3 :

```
Program Compteur3 ;
Var n : Integer ;
Begin
      For n:=10 Downto 1 Do
            Writeln(n,' a pour triple ',3*n)
End.
```

Étape d'exécution :

1) *Evaluation et vérification des bornes (les bornes sont évaluées avant l'exécution du* ***For****).*
2) *Pour chaque valeur consécutive :*
   - *Affectation à la variable de contrôle.*
   - *Exécution de l'instruction à subir.*
   - *La borne supérieure est incluse*

Remarque : cette structure algorithmique peut être remplacée par une structure « **TantQue** » ou « **Répéter** ». ⇨

*(Comparer les organigrammes des boucles « **Pour** » et « **TantQue** »)*

Exemple :

```
Algorithme Etoiles_TantQue
Variable i : Entier
Début
   i ⇦ 1
   TantQue i≤80 Faire
      Ecrire ("*")
      i ⇦ i+1
   FinTantQue
Fin
```

Ou ***TantQue i ≠ 81 Faire***
Ou ***TantQue i < 81 Faire***

Initialisation

Incrémentation

*Gagnées avec « **Pour** »*

Traduction en Pascal

```
Algorithme Etoiles_Répéter
Variable i : Entier
Début
   i ⇦ 1
   Répéter
      Ecrire ("*")
      i ⇦ i+1
   Jusqu'à i>80 Faire
Fin
```

Ou ***Jusqu'à i=81 Faire***
Ou ***Jusqu'à i≥81 Faire***

Remarque :

- On pourrait bien programmer toutes les situations de boucles uniquement avec un « **Tantque** » (ou avec « **Répéter** »). Le seul intérêt du « **Pour** » est d'épargner un peu de fatigue au programmeur, en lui évitant de gérer lui-même la progression de la variable i qui lui sert de compteur (incrémentation automatique). Autrement dit, la structure « **Pour** » est un cas particulier de « **TantQue** » (ou « **Répéter** »).

- Les structures « **TantQue** » (ou « **Répéter** ») sont employées là où l'on ne connaît pas d'avance le nombre de répétitions, comme par exemple :
  - Le contrôle d'une saisie (O ou N).
  - La gestion des tours d'un jeu (Tant que la partie n'est pas finie on recommence).
  - La lecture des enregistrements d'un fichier de taille inconnue (bases de données).
- Les structures « **Pour** » sont employées là où on connaît le nombre de répétitions d'avance.

Les boucles dans les boucles ⇨ OUI (boucles imbriquées)
Mais pas de boucles croisées.

Le nombre de répétitions = Vf-Vi+1

Remarque :
*Les boucles "**For**" sont utiles mais il faut prendre quelques précautions quand on les utilise :*

1. *Il ne faut jamais changer la valeur de l'indice d'une boucle dans l'instruction. Autrement, si quand même besoin, on utilisera une autre variable qui prend la valeur de l'indice ; ou on utilise les autres instructions itératives.*
2. *La valeur de l'indice est limitée à l'instruction de la boucle (elle est perdue dès que l'on sort de la boucle). C'est une variable muette (sa valeur en sortie de boucle est indéterminée).*
3. *"Valeur Initiale" et "Valeur Finale" peuvent être des expressions, donc si la valeur de ces expressions est modifiée dans la boucle, le nombre d'itérations **ne change pas !***

*Ne pas suivre la première règle peut donner un programme valide (qui marche bien), mais il y a des risques d'erreurs et il est possible de rendre le programme incompréhensible. Par exemple, que fait la boucle **For** suivante :*

```
Program mauvaiseboucle;
(* Exemple de programme où on change le valeur de l'indice à l'intérieur d'une boucle For *)
Var lettre1 : Char;
Begin
      For lettre1 := 'A' To 'E' Do
            Begin
                  Writeln(lettre1); lettre1:= 'A' end
```

Exemple : Examinons l'algorithme suivant :

```
Algorithme Actuce
Variable Truc : Entier
Début
  Pour Truc=1 à 20 Faire     .     .     .     .     .     .     (1)
    Truc ⇦ Truc*3     .     .     .     .     .     .     .     (2)
    Ecrire ("passage numéro", Truc)     ..    .     .     .     (3)
  FinPour   .     .     .     .     .     .     .     .     .     (4)
Fin
```

- La ligne (1) augmente la valeur de **Truc** de **1** à chaque passage.
- La ligne (2) triple la valeur de **Truc** à chaque passage.

⇨ Contradiction ⇨ Plantage ⇨ éviter ce genre d'erreur (éviter de manipuler au sein d'une boucle « **Pour** » la variable qui sert de compteur à cette boucle).

Exemple : Calculer la somme des N premiers nombres entiers (avec les différentes boucles)

1ere méthode

```
Algorithme Suite
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
Pour i = 1 à N Faire
  Som ← Som + i
FinPour
Ecrire ("La somme est : ", Som)
Fin
```

Som ← 1
Pour i = 2 à N **Faire**

2ème méthode sans boucles (c'est une suite arithmétique) :

```
Algorithme Suite2
Variables N, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← (1+N)*N/2
Ecrire ("La somme est : ", Som)
Fin
```

3ème méthode

```
Algorithme Suite3
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
i ← 1
TantQue i ≤ N Faire
  Som ← Som + i
  i ← i + 1
FinTantQue
Ecrire ("La somme est : ", Som)
Fin
```

Ou: **TantQue** i ≠ N + *1* **Faire**
Ou: **TantQue** i < N + *1* **Faire**

4ème méthode

```
Algorithme Suite4
Variables N, i, Som : Entier
Debut
Ecrire ("Entrez un nombre : ")
Lire (N)
Som ← 0
i ← 1
Répéter
  Som ← Som + i
  i ← i + 1
Jusqu'à i > N
Ecrire ("La somme est : ", Som)
Fin
```

Ou bien : ***Jusqu'à*** *i = N + 1*
Ou aussi : ***Jusqu'à*** *i ≥ N + 1*

- *Traduction en Pascal*
- *En déduire la factorielle F*

Avec :

F : Real
Write (F :0 :0)

Chapitre 3 : Les variables Indicées (tableaux)

1. Notion de tableau

*Soit un entier **n** positif.*
Supposons qu'on veuille saisir n valeurs réelles afin de calculer leur moyenne, ou de trouver leur minimum, ou de les afficher par ordre croissant.
Pour des valeurs petites de n, on peut déclarer n variables réelles pour résoudre le problème. Mais si n est assez grand, on se rend compte que cela devient impropre, fastidieux, voire impossible.
Il faudrait, dans ce cas, utiliser une variable permettant de représenter les n valeurs. Le type de données de cette variable serait le type tableau.

Définition :

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

Déclaration :

Pour déclarer un tableau, il faut donner :

- Son nom (identificateur)
- Ses bornes : la borne inférieure correspond à l'indice minimal et la borne supérieure correspond à l'indice maximal.
- Le type des éléments le composant.

Syntaxe :

En **algorithmique** :

```
Type nom = Tableau[<indice min> .. <indice max>] de <type des comp>
```

ou

```
Variable nom : Tableau[<indice min> .. <indice max>] de <type des comp>
```

Exemple :

```
Variable t : Tableau[1..10] de réels
```

En **Pascal** :

```
Type nom = Array[<indice min> .. <indice max>] Of <type des composants> ;
```

ou

```
Var nom : Array[<indice min> .. <indice max>] Of <type des composants> ;
```

Exemple :

```
Var  t : Array [1..10] Of real ;
```

Schématiquement, on va représenter la variable **t** comme suit :

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

Dans la mémoire centrale, les éléments d'un tableau sont stockés de façon linéaire, dans des zones contiguës.

*Le tableau ci-dessus est de dimension **1** (couramment appelé **Vecteur**), nous verrons un peu plus loin que l'on peut représenter des tableaux à **2** dimensions, voire même plus.*

*Un **vecteur** est aussi appelé **Tableau Unidimensionnel** ou encore **Liste***

L'élément n° **i** sera représenté par l'expression **t[i]**.

Dans notre exemple, **t[i]** peut être traité comme une variable réelle. On dit que le tableau **t** est de taille **10**.

*Un type énuméré est une séquence ordonnée d'identificateurs. (On parle de type énuméré pour désigner un type dans lequel on peut donner tous les éléments les uns à la suite des autres. Pour l'instant, on en a vu deux, le type **Integer** et le type **Char**, mais on en verra d'autres dans la suite du cours).*

Syntaxe :

```
TYPE Identificateur = (id1, id2,..., idn) ;
```

Exemples :

```
Type
     COULEUR = (jaune, vert, rouge, bleu, marron);
     SEMAINE = (dim, lun, mar, mer, jeu, ven, sam) ;
     REPONSE = (oui, non, inconnu);
     SEXE    = (masculin, féminin);
     VOYELLE = (A, E, I, O, U);
```

Exemple avec référence à un ***type*** existant *:*

```
Var
       JOUR    :  semaine  ;
       A, B, C :  real     ;
       I, J, K :  integer  ;
       CONGE   :  week_end ;
       VIVANT  :  boolean  ;
```

Exemple avec *déclaration locale explicite :*

```
Var
       LETTRE :  'A'  . .  'Z' ;
       FEUX   :  (vert, orange, rouge) ;
```

Exemple de déclaration de « constante », « type » et « variable » :

```
Const
       JOUR_MAX =  31 ;
       AN_MIN   =  2001 ;
       AN_MAX   =  2100 ;
Type
       SIECLE  =  AN_MIN  . .  AN_MAX ;
       SEMAINE = (dimanche, lundi, mardi, mercredi, jeudi, vendredi,
                  samedi) ;
       ANNEE   = (janvier, fevrier, mars, avril, mai, juin, juillet,
                  aout, septemble, octobre, novembre, decembre);
Var
       MOIS       : annee ;
       JOUR       : semaine ;
       N_JOUR     : 1 .. jour_max ;
       AN         : siecle ;
       OUVRABLE   : dimanche .. jeudi ;       {Jours_de_tarvail}
       N_ETUDIANT : 1 ..  maxint ;
```

2. Création d'un tableau (Ecriture dans un tableau)

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

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

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

Si on devait saisir les valeurs, il faudrait écrire :

```
For i := 1 to 10 do     Read(t[i] );
```

3. Affichage d'un tableau

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

Exercice d'application

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

```
Type tab = Array[1..20] Of Integer ;
Var   t1, t2 : tab ;
      i : Integer ;
Begin
      { saisie de t1 }
      For i :=1 To 20 Do
      Begin
            Write('donner t1[',i,']:');
            Readln(t1[i]);
      End;
      { création de t2 }
      For i :=1 To 20 Do
            t2[i] := 2*t1[i];
      { affichage de t2 }
      For i :=1 To 20 Do
            Writeln('t2[',i,']=', t2[i]);
End.
```

NB :

- *Quand on ne connait pas à l'avance le nombre d'éléments exactement, il faut majorer ce nombre, quitte à n'utiliser qu'une partie du tableau.*
- *Il est **préférable** de définir un **TYPE** tableau réutilisable, plutôt que de déclarer à chaque fois, en **VAR**, le tableau en entier.*
- *Il est **interdit** de mettre une variable **dans l'intervalle** de définition du tableau.*

4. Traitement d'un tableau

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

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

```
Var t : Array[1 .. n] Of Integer ;
```

4.1. Somme des éléments d'un tableau

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

```
S := 0 ;
For i :=1 To n Do
      S := S + t[i];
Writeln('la somme des éléments de t est:', S);
```

4.2. Minimum d'un tableau

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

```
min := t[1] ;
For i :=2 To n Do
      If t[i] < min Then  min := t[i];
Writeln('Le minimum des elements de t est:', min) ;
```

4.3. Test d'appartenance

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

```
appartient := false;
For i:=1 To n Do
      If t[i]=x Then    appartient := true;
If appartient then
      Write('x appartient à t')
Else  Write('x n''appartient pas à t');
```

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

```
i := 1 ;
While (t[i]<>x) and (i<=n) Do
    i:=i+1;
If i>n Then
    Write('x  n''appartient  pas  à
t')
Else
    Write('x appartient à t');
```

```
i := 0 ;
Repeat
    i:=i+1;
Until (t[i]=x) or (i>n);
If i>n Then
    Write('x  n''appartient  pas  à
t')
Else
    Write('x appartient à t');
```

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

4.4. Ajout d'un élément

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

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

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

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

```
For i := n Downto k+1 Do
      t[i] := t[i-1];
t[k] := x;
```

4.5. Suppression d'un élément

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

```
For i := k To n Do
      t[i] := t[i+1];
```

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

5. Tableaux de caractères

5.1. Notion de chaîne de caractères

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

Exemples : "Salam", "A", "Nous somme en 2014"

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

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

5.2. Le type STRING de Pascal

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

Exemple :

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

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

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

Les fonctions prédéfinies les plus usuelles sont :

- **Length(s : STRING) : integer;**

Retourne la longueur courante de **s**.

- **Copy(s : STRING ; p : integer ; l : integer): STRING;**

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

- **`Concat(s1,s2 : STRING): STRING ;`**

Renvoie la chaîne résultant de la concaténation de **s1** et de **s2**.

- **`Pos (ssch : STRING ; ch : STRING): Byte;`**

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

CHR (65) = 'A'
ORD ('A') = 65 *: Code* ***ASCII*** *en Décimal*
ORD (S[i]) = ?

Exemple : Ecrire un programme qui demande à l'utilisateur de faire entrer 2 chaines de caractères, qui fait la concaténation des 2 chaines et qui affiche la chaine résultante et nous donne la longueur des 3 chaines.

```
Program Les_strings ;
Uses
   WinCrt ;     { Autorise les Writeln, Readln etc. }
Var
   s, s1, s2 : String ;
Begin
   Write('Entrez une chaîne caractères : ') ;
   Readln(s1) ;
   Write('Entrez en une autre : ') ;
   Readln(s2) ;
   s:= Concat(s1,' ',s2) ;    { en turbo pascal, il faudrait écrire : s := s1 + ' ' +
s2 }
   Write('La longueur de la 1ère chaîne est ') ;
   Writeln(Length(s1)) ;
   Write('La longueur de la 2ème chaîne est ') ;
   Writeln(Length(s2)) ;
   Writeln ;
   Writeln('La chaîne finale est : ', s );
   Writeln('Sa longueur est : ', Length(s):3 ) ;
End.
```

6. Les tableaux à deux dimensions (Matrice)

Pour traiter les notes obtenues par un étudiant à **10** épreuves on peut utiliser un tableau de **10** réels. Pour traiter les notes obtenues par **30** étudiants, on pourrait utiliser **30** tableaux de **10** réels chacun.

Mais puisqu'on va effectuer très probablement les mêmes traitements sur ces tableaux, il est préférable de les regrouper dans une seule variable qui sera un tableau de **30** lignes et **10** colonnes.

Chaque élément de ce tableau multidimensionnel sera identifié par **2** indices : la position indiquant la ligne et la position indiquant la colonne.

Déclaration :

En **algorithmique** :

```
variable t : Tableau[1..30, 1..10] de reels
```

En **Pascal** :

```
var t : Array[1..30, 1..10] of real ;
```

En général :

```
Var identificateur : Array[1..M, 1..N] Of type_éléments ;
```

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

Un tel tableau sera représenté sous la forme suivante :

T[1,1]	t[1,2]	t[1,3]	…	t[1,N]
t[2,1]	t[2,2]	t[2,3]	…	t[2,N]
t[3,1]	t[3,2]	t[3,3]	…	t[3,N]
…	…	…	…	…
t[M,1]	t[M,2]	t[M,3]	…	t[M,N]

NB :

- *Dimension du tableau ci-dessus : **2** (**matrice**)*
- *Nombre de cases disponibles : **M x N***
- *Le tableau **t** est bidimensionnel ⇨ tous les **t[i,j]** sont des variables (**i = 1 … M, j = 1 … N**) de même type*
- *Accès au contenu d'une case : **t[i,j]***

Exemples de déclarations de tableaux :

1. Cas le plus classique

```
type  LISTE = array [1..100] of real ;
      TABLEAU = array [1..10,1..20] of integer ;
      CHAINE = array [1..80] of char ;
```

2. Cas plus complexes

```
type  POINT = array [1..50,1..50,1..50] of real ;
      MATRICE = array [1..M,1..N] of real ;
      SYMPTOME = (FIEVRE, DELIRE, NAUSEE)  ;
      MALADE = array [symptome] of boolean ;
      CODE = 0..99 ;
      CODAGE = array [1..N_VILLE] of code ;
```

3. Définition récursive

```
type  CARTE = array [1..M] of array [1..N] of real ;
var   LIEU : carte ;
```

⇨ Accès à un élément par : `LIEU [I] [J]`

Exemple de programme :

Nous désirons réaliser un programme permettant l'initialisation d'une **matrice unité** de dimension **10**.

Il s'agit donc d'une matrice à **10** colonnes et **10** lignes, ne comportant que des **0**, sauf sur sa diagonale où il n'y a que des **1**.

```
Program Matrice_Unite ;
const l_MAX = 10 ;
  c_MAX = 10 ;
Type TAB = array [1 .. l_max, 1 .. c_max] of integer ;
Var I, J : Integer ;
  MAT : tab ;
Begin
  for I := 1 to l_max do
  begin
    for J := 1 to c_max do
    begin
      if I = J then
          MAT [I, J] := 1
      else
          MAT [I, J] := 0 ;
      write (MAT [I, J]) ;
    end ;
    writeln ;
  end ;
End.
```

Exercice d'application

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

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

Remarque : Les ensembles en **Pascal**

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

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

Exemple de définition de types ensembles :

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

Annexes

Code ASCII

Caractère	Code		
	Déc.	Oct.	Hex.
NUL	0	0	0
SOH	1	1	1
STX	2	2	2
ETX	3	3	3
EOT	4	4	4
ENQ	5	5	5
ACK	6	6	6
BEL	7	7	7
BS	8	10	8
HT	9	11	9
LF	10	12	A
VT	11	13	B
FF	12	14	C
CR	13	15	D
SO	14	16	E
SI	15	17	F
DLE	16	20	10
DC1	17	21	11
DC2	18	22	12
DC3	19	23	13
DC4	20	24	14
NAK	21	25	15
SYN	22	26	16
ETB	23	27	17
CAN	24	30	18
EM	25	31	19
SLB	26	32	1A
ESC	27	33	1B
FS	28	34	1C
GS	29	35	1D
RS	30	36	1E
US	31	37	1F
space	32	40	20
!	33	41	21
"	34	42	22
#	35	43	23
$	36	44	24
%	37	45	25
&	38	46	26
'	39	47	27
(	40	50	28
)	41	51	29
*	42	52	2A

Caractère	Code		
	Déc.	Oct.	Hex.
+	43	53	2B
,	44	54	2C
-	45	55	2D
.	46	56	2E
/	47	57	2F
0	48	60	30
1	49	61	31
2	50	62	32
3	51	63	33
4	52	64	34
5	53	65	35
6	54	66	36
7	55	67	37
8	56	70	38
9	57	71	39
:	58	72	3A
;	59	73	3B
<	60	74	3C
=	61	75	3D
>	62	76	3E
?	63	77	3F
@	64	100	40
A	65	101	41
B	66	102	42
C	67	103	43
D	68	104	44
E	69	105	45
F	70	106	46
G	71	107	47
H	72	110	48
I	73	111	49
J	74	112	4A
K	75	113	4B
L	76	114	4C
M	77	115	4D
N	78	116	4E
O	79	117	4F
P	80	120	50
Q	81	121	51
R	82	122	52
S	83	123	53
T	84	124	54
U	85	125	55

Caractère	Code		
	Déc.	Oct.	Hex.
V	86	126	56
W	87	127	57
X	88	130	58
Y	89	131	59
Z	90	132	5A
[	91	133	5B
\	92	134	5C
]	93	135	5D
^	94	136	5E
_	95	137	5F
`	96	140	60
a	97	141	61
b	98	142	62
c	99	143	63
d	100	144	64
e	101	145	65
f	102	146	66
g	103	147	67
h	104	150	68
i	105	151	69
j	106	152	6A
k	107	153	6B
l	108	154	6C
m	109	155	6D
n	110	156	6E
o	111	157	6F
p	112	160	70
q	113	161	71
r	114	162	72
s	115	163	73
t	116	164	74
u	117	165	75
v	118	166	76
w	119	167	77
x	120	170	78
y	121	171	79
z	122	172	7A
{	123	173	7B
|	124	174	7C
}	125	175	7D
~	126	176	7E
DEL	127	177	7F

Jeu de caractères et symboles de Pascal

Jeu de caractères :
Les caractères suivants sont utilisés pour écrire tout programme **Pascal**:
a) les **chiffres** : 0 1 2 3 4 5 6 7 8 9
b) les **lettres** : a b c... z A B C... Z
c) les **caractères spéciaux** suivants :

(	parenthèse gauche	<	inférieur à (plus petit que)
)	parenthèse droite	=	égale à
[	crochet gauche	>	supérieur à (plus grand que)
]	crochet droit	:	deux-points
{	accolade gauche	;	point-virgule
}	accolade droite	.	point
+	plus	,	virgule
-	moins	'	apostrophe
*	fois, étoile	^	chapeau, flèche
/	divisé, slash		

d) le caractère « espace » appelé aussi « blanc » et le caractère de « tabulation » (HT).
e) le caractère « _ » (souligné).

Remarque :
Tout caractère imprimable peut appartenir à une constante caractère, à une chaîne de caractères ou à un commentaire!
Symboles :
a) les **symboles à un caractère** sont les caractères spéciaux.
b) les **symboles à deux caractères** (accolés) sont les suivants :

:=	affection
..	intervalle (de...à...)
<=	inférieur ou égal à (plus petit ou égal à)
<>	différent de
>=	supérieur ou égal à (plus grand ou égal à)
(*	début commentaire
*)	fin de commentaire

Mots réservés de Pascal

(Petite liste-exemple)

And	Array	Begin	Case	Const	Div	Do
Downto	Else	End	External	File	For	Forward
Function	Goto	If	In	Label	Mod	Nil
Not	Of	Or	Packed	Procedure	Program	Record
Repeat	Set	Then	To	Type	Until	Var
While	With					
Absolute	Boolean	Byte	Char	Comp	Double	Extended
Implementation	Inline	Integer	Interface	Interrupt	Longint	Real
Shl	Shortint	Shr	Single	String	Subrange	Unit
Uses	Word	Xor				

Identificateurs prédéfinis de Pascal

Abs	*Arctan*	*Boolean*	*Char*	*Chr*	*Cos*	*Dispose*
Eof	*Eoln*	*Exp*	*False*	*Get*	*Input*	*Integer*
Ln	*Maxint*	*New*	*Odd*	*Ord*	*Output*	*Pack*
.Page	*Pred*	*Put*	*Read*	*Readln*	*Real*	*Reset*
.Rewrite	*Round*	*Sin*	*Sqr*	*Sqrt*	*Succ*	*Text*
.True	*Trunc*	*Unpack*	*Write*	*Writeln*		

Remarque :
Chaque version de **Pascal** possède des identificateurs prédéfinis supplémentaires!

Types de données élémentaires en Pascal

(Petite liste-exemple)

Désignation	Description	Bornes	Place en mémoire
Real	nombres réels (avec 11 décimales)	$2.9\ 10^{-39}$ **et** $1.7\ 10^{+38}$	6 octets
Single	réel	$1.5\ 10^{-45}$ **et** $3.4\ 10^{+38}$	4 octets
Double	réel (avec 15 décimales)	$5.0\ 10^{-324}$ **et** $1.7\ 10^{+308}$	8 octets
Extended	réel	$1.9\ 10^{-4951}$ **et** $1.1\ 10^{+4932}$	10 octets
Comp	réel	$-2\ 10^{+63}+1$ **et** $2\ 10^{+63}+1$	8 octets
Integer	nombres entiers (sans virgule)	-32 768 **et** 32 767	2 octets
Longint	entier	-2 147 483 648 **et** 2 147 483 647	4 octets
Shortint	entier	-128 **et** 127	1 octet
Word	entier	0 **et** 65 535	2 octets
Byte	entier	0 **et** 255	1 octet
Long	entier	$(-2)^{31}$ **et** $(2^{31})-1$	4 octets
Boolean	variable booléenne (valeurs logiques)	TRUE **ou** FALSE	1 octet
Array [1..10] Of xxx	tableau de 10 colonnes fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
Array [1..10, 1..50, 1..13] Of xxx	tableau en 3 dimensions fait d'éléments de l'ensemble défini xxx (Char, Integer...)		
String	chaîne de caractères		256 octets
String [y]	chaîne de caractère ne devant pas excéder y caractères		y+1 octets
Text	fichier texte		
File	fichier		
File Of xxx	fichier contenant des données de type xxx (Real, Byte...)		
Char	nombre correspondant à un caractère ASCII codé	0 et 255	1 octet
Pointeur	adresse mémoire		4 octets
Datetime	format de date		
Pathstr	chaîne de caractère (nom complet de fichier)		
Dirstr	chaîne de caractère (chemin de fichier)		
Namestr	chaîne de caractère (nom de fichier)		
Extstr	chaîne de caractère (extension de fichier)		

Quelques opérations :

Syntaxe	Utilisation	Type des variables	Description
Inc(a);	Procédure	intervalle ou énuméré	Le nombre **a** est incrémenté de **1**
Inc(a,n);	Procédure	intervalle ou énuméré	Le nombre **a** est incrémenté de **n**
Dec(a);	Procédure	intervalle ou énuméré	Le nombre **a** est décrémenté de **1**
Dec(a,n);	Procédure	intervalle ou énuméré	Le nombre **a** est décrémenté de **n**
Trunc(a)	Fonction	tout scalaire	Prise de la partie entière du nombre **a** sans arrondis
Int(a)	Fonction	a:Real Int(a):Longint	Prise de la partie entière du nombre **a** sans arrondis
Frac(a)	Fonction	Real	Prise de la partie fractionnaire du nombre **a**
Round(a)	Fonction	a:Real Round(a):Longint	Prise de la partie entière du nombre a avec arrondi à l'unité la plus proche
Pred(x)	Fonction	intervalle ou énuméré	Renvoie le prédécesseur de la variable **x** dans un ensemble ordonnée
Succ(x)	Fonction	intervalle ou énuméré	Renvoie le successeur de la variable **x** dans un ensemble ordonnée
Hight(x)	Fonction	tous	Renvoie la plus grande valeur possible que peut prendre de la variable **x**
Low(x)	Fonction	tous	Renvoie la plus petite valeur possible que peut prendre de la variable **x**
Odd(a)	Fonction	a:Longint Odd(a):Boolean	Renvoie **true** si le nombre **a** est impair et **false** si **a** est pair
SizeOf(x)	Fonction	x:tous SizeOf(x):Integer	Renvoie le nombre d'octets occupés par la variable **x**

Procédures prédéfinies :

Appel de la procédure	Actions effectuées
Dispose (pt);	rendre au tas la zone mémoire pointée par *pt^*. Après exécution, *pt* a une valeur indéfinie
Dispose (pt, c1, c2, ..., cn);	doit être utilisée à la place de *Dispose (pt)* si *pt* a été créée par *New (pt, c1, c2, ..., cn)* (même *ci* !)
Get (f);	lit l'élément suivant du fichier *f* ouvert en lecture (possible que si *eof (f)* est fausse !) et l'affecte à *f^*
New (pt);	alloue une zone mémoire, pointée par *pt*
New (pt, c1, c2, ..., cn);	alloue une zone mémoire, pointée par *pt*. Cette forme doit être utilisée dans le cas des enregistrements à variante, pour ne réserver que la place strictement nécessaire aux variantes correspondant aux valeurs *ci*
Pack (t1, i, t2);	soit t1 de type **Array** [m1..n1] **Of** T et t2 de type **Packed Array** [m2..n2] **Of** T alors l'effet de *pack* est équivalent à l'instruction : **For** j:= m2 **To** n2 **Do** t2[j] := t1 [j - m2 + i];
Page (f);	provoque un saut de page sur le fichier (de caractères) *f*
Put (f);	écrit l'élément *f^* à la fin du fichier *f* ouvert en écriture
Read (f, v1, ..., vn);	lit n valeurs dans le fichier de caractères *f* et les affecte à *v1, v2..., vn.* Si *f* est omis, cela équivaut à *Read (input, v1, ..., vn);*
Readln (f, v1, ..., vn);	comme *Read (...);* mais passe ensuite à la ligne suivante
Reset (f);	prépare la lecture du fichier *f* à partir du début de *f*
Rewrite (f);	réinitialise le fichier *f* et le prépare pour l'écriture
Unpack (t2, t1, i);	soit t1 de type **Array** [m1..n1] **Of** T et t2 de type **Packed Array** [m2..n2] **Of** T alors l'effet de *Unpack* est équivalent à l'instruction : **For** j:= m2 **To** n2 **Do** t1[j - m2 + i] := t2[j];
Write (f, v1, ..., vn);	écrit les n valeurs *v1, v2, ..., vn* dans le fichier *f.* Si *f* est omis, cela équivaut à *Write (output, v1, ..., vn);*
Writeln (f, v1, ..., vn);	comme *Write (...);* mais passe ensuite à la ligne suivante

Fonctions prédéfinies :

Appel de fonction	Actions effectuées
Abs (Valeur_Numerique) de	donne la valeur absolue (de type *Integer,* resp. *Real*) *Valeur_Numerique* (de type *Integer,* resp. *Real*)
Arctan (Angle_en_Radians) type	Arctg(*Angle_en_Radians*) *(Angle _en_Radians* de *Real)* avec - pi/2 <= *Angle_en_Radians* <= pi/2
Chr (Valeur_Entiere)	donne le caractère dont le code ASCII est *Valeur _Entiere* (Entier compris entre 0 et 127 (ou 255))
Cos (Angle_en_Radians)	Cosinus de *Angle_en_Radians* (de type *Real*)
Eof (f)	donne la valeur *True* si la fin du fichier *f* est atteinte, *False* sinon. L'abréviation *Eof* signifie *eof (input)*
Eoln (f)	donne la valeur *true* si la fin de la ligne courante du fichier *f* est atteinte, *False* sinon. L'abréviation *Eoln* signifie *Eoln (input)*
Exp (x)	"e puissance x" avec x de type *Integer* ou *Real*
Ln (x)	"log naturel de x" avec x de type *Integer* ou *Real*
Odd (Valeur)	donne la valeur *True* si *Valeur* (de type *Integer*) est impaire, *False* sinon
Ord (Valeur)	- donne le code ASCII de *Valeur* si *Valeur* de type *Char* - donne le numéro d'ordre de *Valeur* si *Valeur* est d'un type Enuméré - donne 0 si *Valeur* vaut *False*, 1 si la *Valeur* vaut *True* - donne l'adresse de l'élément pointé par *Valeur* si *Valeur* est d'un type pointeur
Pred (Valeur)	donne le prédécesseur de *Valeur* (d'un type Scalaire). Provoque une erreur si le prédécesseur n'existe pas!
Round (Valeur)	donne l'arrondi (de type *Integer*) de *Valeur* (de type *Real*)
Sin (Angle_en_Radians)	sinus de *Angle_en_Radians* (de type *Real*)
Sqr (Valeur)	donne le carré (de type *Integer*, resp. *Real*) de *Valeur* (de type *Integer*, resp. *Real*)
Sqrt (Valeur)	donne la racine carrée (de type *Real*) de *Valeur* (de type *Integer* ou *Real*). Valeur doit être >=0!
Succ (Valeur)	donne le successeur de *Valeur* (d'un type Scalaire). Provoque une erreur si le successeur n'existe pas!
Trunc (Valeur)	donne la partie entière (de type *Integer*) de *Valeur* (de type *Real*)

Remarque :

Chaque version de **Pascal** comporte des procédures et fonctions prédéfinies supplémentaires !

On peut voir que les fonctions "tangente" et "factorielle" n'existent pas.

Références Bibliographiques

[1] Jean MAYSONNAVE, « Introduction à l'algorithmique générale et numérique - DEUG Sciences : Résumés de cours », Edition Masson, 1996.

[2] Ives GRANJON, « Tavaux dirigés - Informatique : Algorithmique en Pascal et en langage C - DEUG Sciences : Résumés de cours », Edition Dunod, 1999.

[3] Salah FENNI, « Exercices en Turbo Pascal », Edition Chebba, 2000.

[4] Olivier LECARME, « Pascal : Langages d'écriture de systèmes », Techniques de l'Ingénieur, traité Informatique, H 2260.

[5] Adeline CREPIEUX, « Introduction à l'informatique et à la programmation », Cours de DEUG U1, Université de la méditerranée, 2002. Cours disponible à : http://www.cpt.univ-mrs.fr/~crepieux/infoDEUG.html

[6] Hugo ETIEVANT, « Cours de Turbo Pascal 7.0 : Le cours aux 100 exemples », 2004. Cours disponible à : http://cyberzoid.developpez.com

[7] Philipe TRIGANO, Dominique LENNE, « Algorithmique et programmation », Université de Technologie de Compiègne, 2008. Cours disponible à :

https://moodle.utc.fr/file.php/316/Cours/Poly_NF01_-_ete_2008.pdf

http://www4.utc.fr/~nf01

[8] Christophe DARMANGEAT, « Algorithmique et programmation pour non-matheux : Cours complet », Université Paris 7, 2008. Cours disponible à :

http://www.pise.info/index.htm