Année Universitaire 2006/2007

UNIVERSITE MOULAY ISMAIL
Faculté des Sciences et Techniques
D'Errachidia
Département de Mathématiques

Filière P.C et G.E

Polycopie des cours de programmation (Turbo-Pascal)

(M124/2 et M144/2)

Présenté par :

Mohamed DEROUICH

Table des matières

Chapitre 1

Structure des ordinateurs

1.1 Système informatique

1.1.1 Introduction :

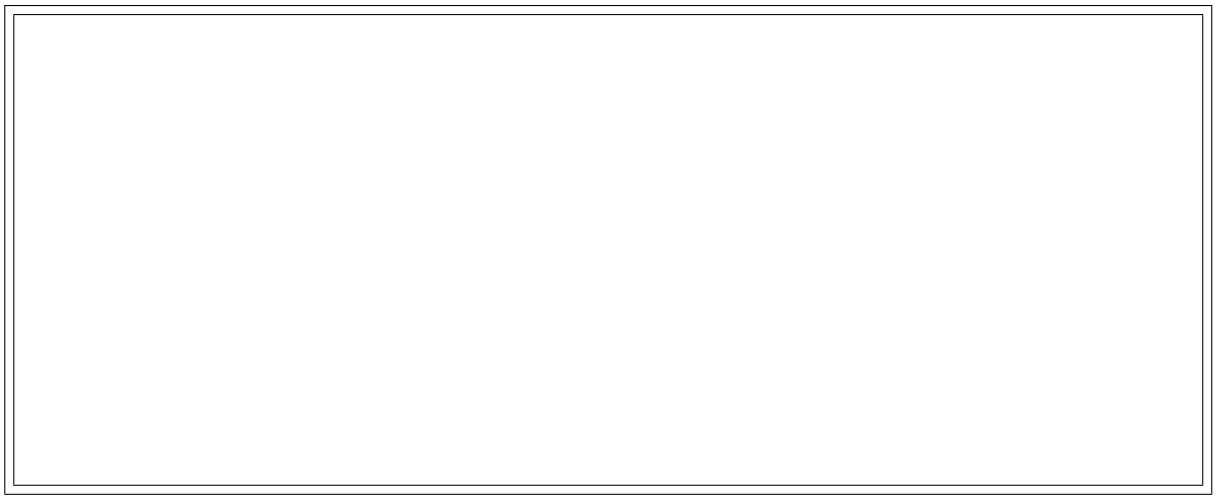
L'informatique est la science de traitement rationnel, notamment à l'aide de machine automatique, de l'information considérée comme le support des connaissances et des communications. En Anglais on trouve les termes : Informatics, computer Science et Electronic Data Processing (traitement électronique des données).

1.1.2 Système informatique :

Pour comprendre comment fonctionnent un ordinateur on va établir une analogie avec la façon dont travail l'homme . prenons comme exemples un employé de bureau :

- ▷ L'employé s'installe au centre de son bureau (Unité Centrale de traitement).
- ▷ On lui remet dans une corbeille le travail à faire (information en entrée).
- ▷ Dans une autre corbeille il devra mettre le travail fait (Information en sortie).

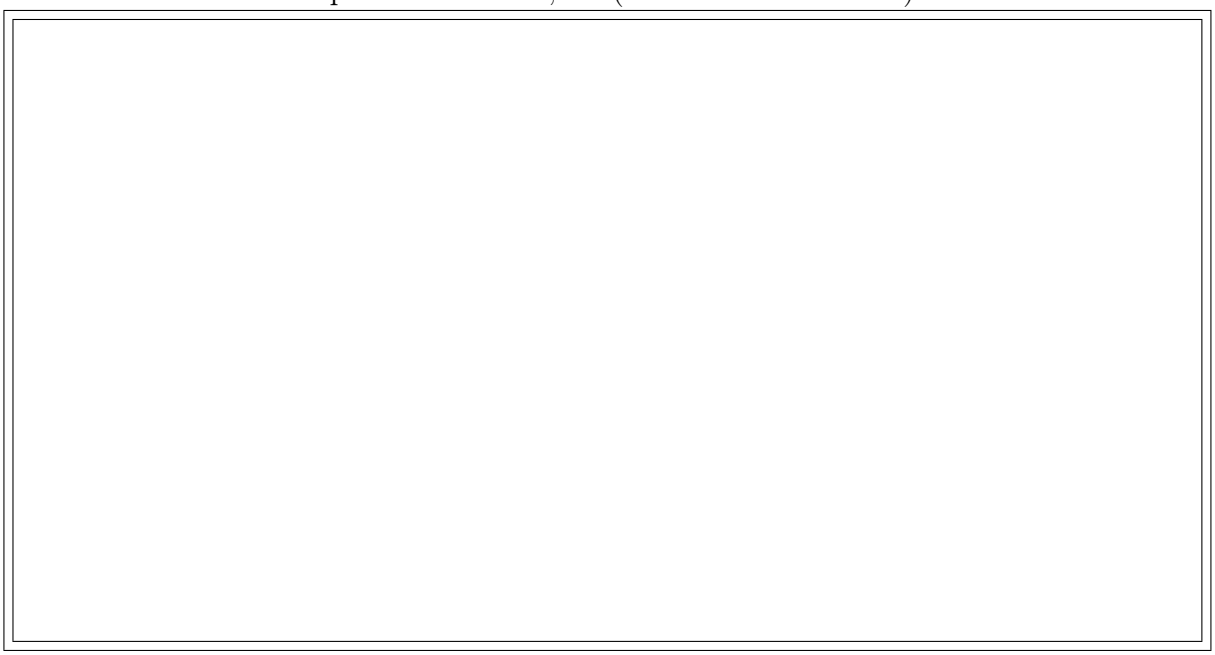
Ainsi nous pouvons donc considérer l'employé comme un système de traitement à qui l'on remet des informations en entrée et qui traite ces informations et nous restitue des informations en sortie.



⋆ Pour effectué sont travail l'employé peut avoir besoin de réaliser des calculs, il dispose d'une calculatrice (unité de calcul).

⋆ Pour ne pas oublier ce qu'on lui demande, il note sur brouillon de instructions qu'on lui demande reçus du directeur (c'est le programme).
Ø Il note certain information sur les données qu'il va traiter (Mémorisation des instructions à exécuter).

⋆ Il a besoin de certain informations comme les prix des produit qui existent sur des catalogues volumineux, les adresses des clients qui existent sur des répertoires classés, ... (Mémoires auxiliaires)



1.1.3 Le matériels informatique :

Dans l'exemples cité on peut distingué deux choses : L'employé de bureau et les instruction à exécuter par l'employé (ce qu'on lui demande).
Ainsi le système informatique est composer de deux parties :

Le matériels : constitués de l'unité centrale et les organes d'entré/sortie.

Les logiciels : Ce sont des programmes (ensembles d'instruction que la machine doit exécuter).

1. **La partie matérielle (Hardware)**
   i)− L'unité centrale ou le Processeur (En anglais CPU Central Processing Unit) c'est dans cette partie que l'information est traitée, l'unité central est constituée de deux organes :
   - L'unité arithmétique et logique : qui permet d'effectuer des opérations sur les données telles que Addition, Soustraction, Multiplication, Division et les opération logique .
   - Unité de commande : Dirige toutes les opérations qui se déroulent dans l'ordinateur .

   ii)− Une Mémoire centrale contenant les programme à exécuter et les données à traiter.
   iii)−Les organes périphériques : ils comprennent :
   - Les organes d'entrées comme : clavier, la souris, lecteur de disquettes,...
   - Les organes de sorties comme : qui fournis des résultats comme : Ecran, Imprimante, haut-parleur, ...

2. **Partie logiciels**
   i)− Logiciels : est un (ensemble) de programme(s) qui permet(tent) d'utiliser la partie matériels de l'ordinateur , on distingues deux familles de logiciels :
   ii)− Logiciels de base :Il permet la gestion du matériels (l'ordinateur et ses périphériques) . il est généralement fourni par le constructeur, parmis les logiciels de base on trouve les systèmes d'exploitations (Dos, Windows, Linux,..) , les compilateur des langages (Pascal, C,C++,...)
   iii)− Logiciels d'applications : logiciels outils (Traitement de texte Tableur,..) ou toutes programmes d'un utilisateurs.

1.2 Système d'exploitation :

1.2.1 Définitions et concepts :

Sans ses logiciels un ordinateur n'est qu'un métal, grâce à ses logiciels, il peut mémoriser, traiter et restituer des informations.

Les logiciels se répartissent en deux grandes catégories : les programmes systèmes qui permettent le fonctionnement de l'ordinateur et les programmes d'applications qui résolvent les problèmes des utilisateurs.

Le système d'exploitation et le programme fondamental des programmes systèmes, il contrôle les ressources de l'ordinateur comme la mémoire, l'unité centrale, le disque dur, l'imprimante, Ĕ et fourni la base sur laquelle seront construit les programmes d'applications.

1.2.2 Les fonctions d'un système d'exploitation :

Les fonctions majeurs d'un système d'exploitation sont la gestion des ressources (Mémoires, C.P.U, fichiers partagés, ...), gestion des données, gestion des tâches et fournir une interface de communication entre l'utilisateur et l'ordinateur.

- Gestion des ressources : Il s'agit de l'allocation et libération de ressources telle que le temps C.P.U, Mémoire Central, Le matériel d'entré sortie, afin d'optimiser l'utilisation de ses ressources et d'accroître la disponibilité de l'ordinateur pour chaque utilisateur.
- Gestion des données : Il gère les entrées et sortis des données et leurs emplacements, sauvegarde et récupération de l'information et l'organisation de celle-ci sur disque.
- Gestion des tâches : Une tâche et un ensemble de programmes et de données associées, le système contrôle les taches en exécution
- Fourni une interface à l'utilisateur : L'interface est la partie du logiciel qui communique avec l'utilisateur est Il fournit à usager un ensembles de commande qui contrôle le matériel, cette interface prend en général trois forme :
    * Ligne de commande : l'utilisateur doit taper la commande à exécuter en utilisant un langage de commande bien définit.
    * Interface Menu : permet à l'utilisateur de sélectionné une commande de menus, la sélection se fait par souris ou clavier.
    * Interface graphique : généralement est constituée des éléments suivantes : Icône : Image graphique qui représente un objet (Fichier, Répertoire,Ĕ) Pointeur graphique : Pour sélectionner les icônes, les commandes et pour déplacer les objets sur écran. Menu déroulant, Fenêtre, boites de dialogue,...

Chapitre 2

Généralités sur l'information

2.1 Définition de l'informatique

Le mot informatique a été proposé par Philippe Dreyfus en 1962 ; c'est un mot-valise, formé d'information et d'automatique. L'informatique c'est donc une automatisation de l'information, plus exactement un traitement automatique de l'information. L'information désigne ici tout ce qui peut être traité par l'ordinateur (textes, nombres, images, sons, vidéos,...). L'outil utilisé pour traiter l'information de manière automatique s'appelle un ordinateur. Ce nom a été proposé par Jacques Perret (professeur de Latin à La Sorbonne) en 1954. Ce mot était à l'origine un adjectif qui signifiait "qui met de l'ordre", "qui arrange". L'anglais, plus restrictif, utilise le terme de computer qui peut se traduire par calculateur, machine à calculer. L'informatique désigne donc un concept, une science, tandis que l'ordinateur est un outil, une machine conçue pour réaliser des opérations informatiques

2.2 L'information

Information est tout forme de données représentant un ou plusieurs renseignement . On distingue généralement différents types d'informations : textes, nombres, sons, images, etc .... L'information sous sa forme brute, ne peut être directement traité est analysée par une machine logique telle que l'ordinateur. Alors pour remédier ce problème on l'a traduit dans une alphabet propre à ce dernier. C'est l'opération de codage.

2.3 Le codage de l'information

2.3.1 Définitions :

◇ **Le codage** d'une information consiste à établir une correspondance entre la représentation externe (habituelle) de l'information (ex : le caractère A ou le nombre 36), et sa représentation interne dans la machine, qui est une suite de 0 et 1.

◇ **Le transcodage** est le passage d'un code à un autre pour la même information.

◇ **Le BIT** signifie "binary digit", c'est-à-dire 0 ou 1 en numérotation binaire. C'est la plus petite unité d'information manipulable par une machine. On peut le représenter physiquement :

* par une impulsion électrique qui correspond à la valeur 1 ou une absence d'impulsion qui correspond à la valeur 0.
* par des alvéoles ou des espaces dans une surface (CD-ROM)
* grâce à des bistables, c'est-à-dire des composants qui ont deux états d'équilibre (un correspond à l'état 1, l'autre à 0)

◇ **L'octet** est une unité d'information composée de 8 bits. Il permet de stocker un caractère, tel qu'une lettre, un chiffre ...

i) Le kilo-octet (kO) : 210 = 1024 octets
ii) Le mega-octet (MO) : 220 = 1048576 octets
iii) Le giga-octet (GO) : 230 = 1073741824 octets = 1024 MO
iv) Le tera-octet (TO) : 240 = 1099511627776 octets = 1024 GO

2.3.2 Représentation des données

la base 10 :

Habituellement, on utilise la base 10 pour représenter les nombres, c'est-à-dire que l'on écrit à l'aide de 10 symboles distincts, (les chiffres 0 1 2 3 4 5 6 7 8 9).
Donc pour un nombre $x = a_n a_{n-1} \cdots a_1 a_0$ avec $a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ on a : $x = a_n a_n - 1 \cdots a_1 a_0 = \sum_{i=0}^{n} a_i 10^i$

cas générale :

En générale dans une base b, on utilise b chiffres. Notons a_i la suite des chiffres utilisés pour écrire un nombre $x = a_n a_{n-1} \cdots a_1 a_0$ avec $a_i < b$ alors

x est donné par : $x = a_n a_n - 1 \cdots a_1 a_0 = \sum_{i=0}^{n} a_i b^i$
a_0 est le chiffre de poids faible, et a_n le chiffre de poids fort.

le code binaire pur (CBP) :

On utilise la représentation binaire car elle est simple, facile à réaliser techniquement à l'aide de bistable (système à deux états réalisés à l'aide de transistors). Enfin, les opérations arithmétiques de base sont facile à exprimer en base 2. En binaire on a : $b = 2$ et $a_i \in \{0, 1\}$: 2 chiffres binaires, ou bits.

le code hexadécimal :

En notation hexadécimale on dispose de 16 symboles : 10 chiffres 0,1, Ĕ,9 et 6 symboles A,B,Ĕ,F . Donc en hexadécimal, b=16,
$a_i \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$ (avec A représente 10, B 11, Ĕ et F 15).

Exemples 2.3.1 *:*
En base 10 :

$$(1996)_{10} = 1 \times 10^3 + 9 \times 10^2 + 9 \times 10^1 + 6 \times 10^0$$

En base 2 :

$$(101)_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$$

En base Hexadécimal :

$$(AB)_{16} = 10 \times 16^1 + 11 \times 16^0 = 160 + 11 = (171)_{10}$$

Remarque 2.3.1 *:*
La notation $()_b$ indique que le nombre est écrit en base b.

Le code ASCII

La mémoire de l'ordinateur conserve toutes les données sous forme numérique. Il n'existe pas e méthode pour stocker directement les caractères. Chaque caractère possède donc son équivalent en code numérique : c'est le code ASCII (American Standard Code for Information Interchange - traduisez " Code Americain Standard pour l'Echange d'Informations"). Le code ASCII de base représentait les caractères sur 7 bits (c'est-à-dire 128 caractères possibles, de 0 à 127). Le code ASCII a été étendu à 8 bits (un octet) pour pouvoir coder plus de caractères (on parle d'ailleurs de code ASCII étendu...). Ce code attribue les valeurs 0 à 255 (donc codées sur 8 bits, soit 1 octet) aux lettres majuscules et minuscules, aux chiffres, aux marques de ponctuation et aux autres symboles(voir le tableau de code ASCII)

- *Les codes 0 à 31 ne sont pas des caractères. On les appelle caractères de contrôle car ils permettent de faire des actions telles que :*
  - *retour à la ligne (CR) ;*
  - *Bip sonore (BEL)*
- *Les codes 65 à 90 représentent les majuscules ;*
- *Les codes 97 à 122 représentent les minuscules.*

Remarque 2.3.2 *:*
il suffit de modifier le 5^{me} *bit pour passer de majuscules à minuscules, c'est-à-dire ajouter 32 au code ASCII en base décimale.*

2.3.3 Passage de la base 10 vers une base quelconque

pour passer de base 10 vers une base quelconque on procède par divisions successives. On divise le nombre par la base , puit le quotient par la base, et ainsi de suite jusqu' a l'obtention d'un quotient nul. la suite des restes obtenus correspond aux chiffres dans la base visée,

Exemple 2.3.2 *: soit à convertir* $(44)_{10}$ *vers la base 2 :*

$$\begin{aligned} 44 = 22 \times 2 + 0 &\implies a_0 = 0 \\ 22 = 11 \times 2 + 0 &\implies a_1 = 0 \\ 11 = 5 \times 2 + 1 &\implies a_2 = 1 \\ 5 = 2 \times 2 + 1 &\implies a_3 = 1 \\ 2 = 1 \times 2 + 0 &\implies a_4 = 0 \\ 1 = 0 \times 2 + 1 &\implies a_5 = 1 \end{aligned}$$

Donc : $(44)_{10} = (101100)_2$

2.3.4 Les opérations en binaire

L'addition en binaire :

L'addition en binaire se fait avec les mêmes règles qu'en décimale : On commence à additionner les bits de poids faible (les bits de droite) puis on a des retenues lorsque la somme de deux bits de mêmes poids dépasse la valeur de l'unité la plus grande (dans le cas du binaire : 1) ; cette retenue est reportée sur le bit de poids plus fort suivant... Par exemple : $01101 + 01110 = 11011$ *(décimal* $13 + 14 = 27$*)*

$$\begin{array}{cccccc} & 0 & 1 & 1 & 0 & 1 \\ + & 0 & 1 & 1 & 1 & 0 \\ \hline & 1 & 1 & 0 & 1 & 1 \end{array}$$

La multiplication en binaire

La multiplication se fait entre bits de même poids, avec le même système de retenue qu'en décimale. La table de multiplication en binaire est très simple : $0\times0=0; 0\times1=0; 1\times0=0; 1\times1=1$. *Par exemple :*$01101\times00110 = 10011100$ *(décimal* $13\times6=78$)

$$\begin{array}{cccccccc} & 0 & 1 & 1 & 0 & 1 & & \\ \times & 0 & 0 & 1 & 1 & 0 & & \\ \hline & 0 & 1 & 1 & 0 & 1 & 0 & \\ 0 & 1 & 1 & 0 & 1 & & & \\ \hline 1 & 0 & 0 & 1 & 1 & 1 & 0 & \end{array}$$

Chapitre 3

Introduction à la programmation Pascal

3.1 Introduction :

Un ordinateur est une machine qui exécute un certain nombre d'opérations comme par exemple :

- *addition, soustraction, multiplication,*
- *afficher les résultats des opérations sur l'écran,*
- *comparer des nombres.*

Sa puissance vient du fait qu'il peut être programmé, c'est à dire que l'on peut lui donner, à l'avance, la séquence (la suite ordonnée) des ordres à effectuer l'un après l'autre. Le grand avantage de l'ordinateur est sa rapidité. Par contre, c'est le programmeur qui doit TOUT faire. L'ordinateur ne comprenant que des ordres codés en binaire (le langage machine), et donc pour faciliter la programmation, des langages dits "évolués" ont été mis au point.

3.2 Langages de programmation

Comme pour les langues naturelles (Français, Anglais, Russe, Chinois, etc.), il existe un grand nombre de langages de programmation. Certains sont adaptés à des domaines particuliers, d'autres dépendent de certains types d'ordinateurs, etc.

3.2.1 Définitions

Définition 3.2.1
Un langage de programmation est un symbolisme conventionnel assurant la communication entre la machine (ordinateur) et l'utilisateur. Il possède un

vocabulaire bien défini. On distingue trois types de langage de programmation : langage machine interne, langage machine externe et langage évolué.

Définition 3.2.2
Un programme est une suite d'instructions élémentaire, qui vont être exécutées dans l'ordre par l'ordinateur. Ces instructions correspondent à des actions très simples, comme additionner deux nombre, lire ou écrire une case mémoire, etc ...

Définition 3.2.3
Un algorithme est la données d'une suite fini d'étape (l'étape est une information élémentaire, information général). A exécuter dans un ordre donné pour résoudre un problème.

Remarque 3.2.1

1. *Un algorithme doit exprime dans un langage compris par le lecteur.*
2. *un programme est un algorithme exprimé dans un langage de programmation.*

3.2.2 Langage Machine Interne (LMI)

Le langage machine interne est le seul langage qui directement compris par l'ordinateur. Ses instructions ne sont rien d'autre que des séquences de "0" et de "1".

3.2.3 Langage Machine Externe (LME)

Ce langage possède exactement la même structuration que le langage machine interne à l'exception d'une différence au niveau de la représentation de ses instructions. Les opérations de ces dernières, dans le langage machine interne sont représentées par des codes hexadécimaux ou binaires, alors que dans le langage machine externe elles sont symbolisées par des mnémonique significatifs.

Exemple 3.2.1
Addition :$\Longrightarrow$ ADD ; soustraction : $\Longrightarrow$ SUB ; Comparaison :$\Longrightarrow$CMP.

3.2.4 Langages Evolués (LE) :

Ils sont appelés aussi langage de haut niveau. Leur degré d'évolution est maximum (très proche des langage naturels. Les symboles mis en jeu, sont plus explicite que ceux de langage machines antérieurement vus.

3.2.5 Exemples de langages évolués (les plus connus)

- **Fortran :** *c'es le plus ancien des langage évolué. (domaines scientifiques)*
- **Basic :** *il a été élaboré dans un cadre pédagogique. (simple et très limité)*
- **Pascal :** *langage structuré, efficace pour les structures arborescentes (en hommage au philosophe et mathématicien français Blaise Pascale)*
- **C :** *il ressemble beaucoup au langage Pascal mai il est très puissant et ouvert par rapport à ce dernier. (efficace pour le traitement des chaînes de caractères)*
- **Visual Basic :** *(interfaces graphiques)*
- **Java :***(efficace pour des programmes en rapport avec Internet)*

3.2.6 Systèmes de traduction :

Assembleur :

C'est un programme qui convertit ou traduit une séquence d'unstrictions (programme du LME en un programme du LMI (directement exécutable par l'ordinateur)

Compilateur :

Le compilateur est un programme important chargé de traduire un programme écrit en LE (programme source) en un autre exprimé en Lm (programme objet). L'ordinateur , par la suit, n'aurait à exécuter que le programme objet.

Interpréteur :

l'interpréteur possède une fonction presque analogue à celle du compilateur sauf qu'il traduit et exécute le programme instruction par instruction
N.B :
Le compilateur par opposition à l'interpréteur, traduit tout le programme avant son exécution.

3.3 Introduction à la programmation avec Turbo-PASCAL

Le langage Pascal offre une très bonne approche de la programmation. Très utilisé dans le milieu scolaire et universitaire, il permet d'acquérir des

notions solides que l'on retrouve dans tous les autre langage Le PASCAL est un langage compilé, c'est à dire qu'il faut :

1. *entrer un texte dans l'ordinateur (à l'aide d'un programme appelé EDITEUR),*
2. *le traduire en langage machine (c'est à dire en codes binaires compréhensibles par l'ordinateur) : c'est la compilation*
3. *et éventuellement l'exécuter sur machine.*

3.3.1 Types de données :

Une variable est une zone dans la mémoire vive de l'ordinateur, dotée d'un nom et d'un type. Le nom de la variable permet d'accéder au contenu de la zone mémoire ; le type spécifie la nature de ce qui peut être stocké dans la zone mémoire (entier, réel, caractère, etc..). Pascal supporte quatre types de variables de base. Ces types sont :

1. *integer (entier)*
2. *char (caractère)*
3. *boolean (booléen)*
4. *real (réel)*

3.3.2 types de données scalaires :

entiers :

On appelle variable entière, une variable dont le contenu est une valeur entière (dont l'écriture ne nécessite pas l'emploi de virgule). En turbo-pascal, on distingue, pour les entiers, cinq types différent : INTEGER, LONGINT, SHORTINT, BYTE et WORD. Et chaque type définit un ensemble d'entiers borné. Ainsi, les intervalles associés à chacun d'eux sont :

Désignation	Bornes	Place en mémoire
Integer	[-32 768, 32 767]	2 octets
Byte	[0 ,255]	1 octet
Word	[0 , 65535]	2 octets
Shorting	[-128 , 127]	1 octet
longint	[-2147483648 , 2147483647]	4 octets

Les opérations et les fonctions pouvant être appliquées aux entiers sont, à titre d'exemple :
1) $+$, $*$, DIV, MOD, ABS, INC, DEC, SQR, $SUCC$, $PRED$,
2) opérateurs relationnels : $>=$, $<=$, $>$, $<$, $<>$, ...

Exemple 3.3.1 *:*
i) − $7DIV2 = 3$ *(quotient de la division euclidienne).*
ii) − $7MOD2 = 1$ *(le reste de la division euclidienne).*
iii) − $SQR(3) = 9$ *(le carré de 3)*

Réels :

Les réels, en machine, ne sont en fait qu'un sous-ensemble de l'ensable Q des rationnels. Comme pour les entiers , les réels sont regroupés selon leur grandeur, en cinq types : Real, Single, Double, Extended et Comp. Leurs intervalles respectif sont :

Désignation	Bornes	Place en mémoire
real	[2.9E-39 , 1.7E38]	6 octets
sigle	[1.5E-45 , 3.4E38]	4 octets
double	[5.0E-324 , 1.7E308]	8octets
Extended	[1.9E-4951 , 1.1E4932]	10 octets
Comp	[-2E+063+1, 2E+063+1]	8 octets

Pour ces nombres, on dispose de deux notations, décimale et scientifique.
***Notation décimale :**

(1)	(2)	•	(3)

1. **(1)**= *signe,*
2. **(2)**= *partie entière,*
3. **(3)**= *partie fractionnaire*
4. *(•)= virgule décimale*

Exemple 3.3.2 *: -13.301 ; 205.12 (la mise du signe (+) est facultatif)*
***Notation scientifique :**

(1)	E	(2)

1. **(1)** = *Mantisse (ou caractéristique), exprimée en notation décimale,*
2. **(2)** = *Exposant (c'est un entier) positif ou négatif de la puissance 10*
3. **(E)** = *C'est l'initiale de Exposant.*

Exemple 3.3.3 *:*
$-i) - 6.1E - 10 = -6.1 \times 10^{-10} = -0.00000000061$
$-ii) 1.34E5 = 1.34 \times 10^{5} = 134000$

Ces nombres réels peuvent faire l'objet d'opérandes pour les opération suivantes :
ABS, SQR, SQRT, SIN, COS, ARCTAN, LN, EXT, ROUND, TRUNC, <= , >=, >, <, ...

caractères (CHAR) :

déclaration : VAR liste de variables : CHAR ;
ces variables contiennent UN caractère.CHAR est le type représentant tous les caractères du code ASCII(lettres, chiffres,...). Comme fonctions opérant sur les CHAR,on cite : ORD, PRED, SUCC, CHR, ⋯

Exemple 3.3.4 *:*
- ORD (caractère) = code ASCII de ce caractère.
- CHR (caractère) = caractère équivalent à ce code
- PRED (caractère) = caractère précédent
- SUCC (caractère) = caractère suivant

Booléens :

Une variable logique ou booléenne ne peut prendre que deux valeurs :TRUE et FALSE.
Des opérateurs logiques, on rappelle :
* *Les opérateurs relationnels(<,>,>=,<=,<>)*
* *La conjonction AND*
* *La conjonction OU*
* *La négation NOT*

Exemple 3.3.5 *: L'expression, (condition 1) AND (condition 2), est TRUE si (condition 1) et (condition 2) sont simultanément TRUE sinon elle est FALSE.*

Remarque 3.3.1 *:*
Les cas restants des type scalaires, seront évoqués aux prochains chapitres.

3.3.3 Identification d'objets :

Le programmeur va devoir nommer tous les objets manipulé dans son programme. Ce nom est dit **identificateur**.
En Turbo pascal, on distingue trois catégories d'identificateur : les mots clés réservés (MCR), les mots clés standards (MCS) et les identificateurs propres à l'utilisateur (MU).

♢ **Mots clés réservés :** *Ces mots sont propres au langage et ne peuvent en aucun cas être redéfinis par l'utilisateur pour son propre intérêt (identification de ses propres objets).*
Exemple : *Program, TYPE, VAR, FUNCTION,PROCEDURE, LABEL, BEGIN, END,...*

♢ **Mots clés standards :** *Il s'agit de mots prédefinits ; cependant, il n'est pas complètement interdit au programmeur de les adopter pour représenter d'autre objet. Cette possibilité est déconseillée afin d'éviter toute sorte de confusion au sien d'un programme.*
Exemple : *ABS, MOD, DIV,....*

♢ **Identification définis par le programme :** *Ce sont des identificateurs que le programmeur, selon la nature de sont programme, peut introduire pour nommer des objets tels que programme, types, variables, constantes, fonctions, procédures ....chaque identificateur doit respecter les conditions suivantes :*
- *son caractère initial doit être une lettre alphabétique*
- *il ne peut comprendre que des caractère alphanumériques et le trait de soulignement(_).*
- *seuls les 63 premiers caractères qui seront tenus en compte.*

Exemple 3.3.6 *:*

Noms	validation
Begin	non valide(car c'est réservé)
3SOT	non valide (lettre initiale est 3)
SOT34AB	valide
Dollar $	non valide
I LHAM	non valide (il y a un 'blanc')
Terre +ciel	non valide (il y a +)
jour_nuit	valide
Jour-nuit	non valide
αrayon	non valide (il y a α)

Remarque 3.3.2 *:*
Si les 63 premiers caractères sont identiques pour deux identificateurs alors ces derniers sont exactement égaux c'est-à-dire ils représentent le même objet.

3.3.4 Déclaration de constantes et de variables scalaires :

En Turbo Pascal (T.P), tout objet (variable ou constante), avant de se permettre son emploi au sein d'un programme, il est nécessaire de le déclarer.

déclartion de constantes :

*Elle se fait grâce au mot clé réservé "***CONST** *" de façon suivante :*

```
CONST
     <nom de la constante 1> = sa valeur ;
        ....
        ....
      <nom de la constante n> = sa valeur ;
```

Exemple 3.3.7 *:*

```
CONST
    PI = 3.14 ;  {constante réelle}
    Alpha = 0.22 ;  {constante réelle}
    Beta = 'A';  {constante caractère}
    Nom = 'ASMAE' ;  {constante chaîne}
    ENT = 19 ;  {constante entière}
```

Déclaration de variables :

*Cette déclaration consiste à préciser le nom et le type de chaque variable. Elle s'entame toujours par le mot réservé "***VAR***" et se fait de la manière illustrée ci-dessous :*

```
VAR
 <nom de la variable> : type ;
    {ou dans le cas de plusieurs variable de même type}
<nom de la variable1>,<nom de la variable2>....: type ;
```

Exemple 3.3.8 *:*

```
VAR
  I, J, R :integer ;
  A, B, C : real ;
  X, Y, Z : CHAR ;
  LG, KX, M, N : BYTE ;
  Reponse, Etat,L: BOOLEAN ;
```

Remarque 3.3.3 *:*
En Turbo-Pascal, on ne différencie pas entre une lettre en majuscule et son équivalent en minuscule.

3.3.5 structure générale, d' programme :

En générale, un programme écrit en Turbo-Pascal, possède la structure syntaxique suivante :

* *L'entente du programme* { **nom du programme ;**
* *Bloc De déclaration* {
  [CONST {déclaration de constantes}]
  [TYPE {déclaration de types}]
  [VAR {déclaration de variables}]
  [FUNCTION {déclaration de fonctios}]
  [PROCEDURE {déclaration de procedures}]
* *Instructions exécutables ou corps du programme* {
  BEGIN
  instruction1 ;
  instruction2 ;
  ... ...
  instructionn ;
  END.

N.B :
Les déclarations mises entre crochets ne sont qu'optionnelles ; autrement dit leur emploi dépend des besoins de l'utilisateur en matière de données.
Un programme est composé, comme illustré ci-dessus, de trois parties distinctes :

* *l'entête :comprend le nom du programme.*
* *Le bloc de déclaration : Toute déclaration effectuée doit figurer à l'intérieur de ce bloc.*
* *Le bloc d'instructions exécutables : Ce bloc, délimité par begin et end, comporte l'ensemble des instructions qui réalisent l'objectif du programme.*

Exemple 3.3.9 *: un programme simple qui affiche les mots 'Bonjour. Comment allez-vous ?' sur l'écran.*
N.B : Le mot-clé **writeln** *écrit le texte sur l'écran.*

```
program MONPREMIER ;
  begin
      writeln ('Bonjour.  Comment allez-vous ?')
  end.
```

Remarque 3.3.4 *:*

1. *Si on a à utiliser des unités, on les appelle directement après l'en-tête par la clause 'uses', comme suit : USES nom de l'unité 1, nom de l'unité 2, ... ;*
2. *Tout commentaire est mis entre accolade { ...} ou entre (* ... *) ; il est ignoré lors de la compilation.*

3. *";" est appelé séparateur d'instructions*
4. *"." est l'indicateur de fin du programme.*

Exemple 3.3.10 *:*

```
Programme exemp1
   CONST
         Semaine=7 ;
         Mois=30 ;
         Electron= -1.6E-19 ;
         Ville='ERRECHIDIA' ;
         Lettre='?' ;
     VAR
        Nbre1, Nbre2, Nbre3 : Integer ;
          X, y, z, w, v :  Real ;
          C : CHAR ;
          Answer : Boolean ;
   Begin
      {programme vide ;}
   END.
```

3.4 Outils de communication

Dans cette partie , on expose quelque instructions nécessaire à l'institution d'un pont de communication entre la machine et l'utilisateur. De même, le déroulement d'exécution des expressions scalaires sera éclairci afin d'éviter toute erreur surtout de raisonnement.

3.4.1 Expression scalaires :

Expression arithmétique :

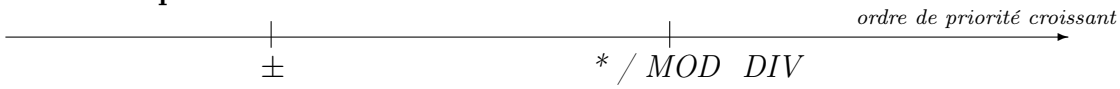
une expression arithmétique est une représentation mathématique constituée d'opérateur et d'opérandes arithmétiques (réels ou entiers). Son évaluation s'effectue en respectant les règles suivantes :

1. *L'opérateur le plus prioritaire est le premier à être exécuté.*
2. *Si deux opérateurs ont le même ordre de priorité, alors celui de gauche est executé le premier.*
3. *Un opérateur mis entre parenthéses, est prioritaire.*

Les opérateur aithméques :

- **, +, - : respectivement la multiplication, l'addition et la soustraction entre reels ou entiers.*
- */ : La division rélle (ex : 5/2=2.5)*
- *Div : La division entiére (ex : 5Div2=2)*
- *MOD : Le reste d'une division entiére (ex : 5MOD2=1).*

Ordre de priorité :

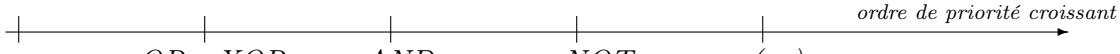


Exemple 3.4.1 *:*
Déroulement de l'évaluation de :
*Step1 21/2+4*3 Div 2-5 MOD 4 Div 5*
*Step2 10.5+4*3 Div 2-5 MOD 4 Div 5*
Step3 10.5+12 Div 2-5 MOD 4 Div5
Step4 10.5+6-5 MOD 4 Div 5
Step5 10.5+6-1 Div5
Step6 10.5+6-0
Step7 16.5-0
Step8 16.5

Expressions Boolèennes :

Une expression boolèenne est une écriture logique qui peut prendre la valeur True ou bien False. Ses opérateur classés suivant leur ordres de priorité, sont :

Ordre de priorité :



<=, >=, OR, XOR AND NOT (...)
<, >,
=, <>,
∈

Exemple 3.4.2 *:*
(X=4) OR (X >= 0) AND (Y <= 0)

Remarque 3.4.1 *:*

- ⋆ *La règle considérée pour l'évaluation des expressions arithmétique reste valable pour les expressions boolèennes.*
- ⋆ *Pour rendre plus prioritaire une opération. Il suffit de l'encadrer par des parenthèses.*

Priorité des opérateurs :

On classe les différents opérateurs par ordre de priorité, les opérateurs de plus forte priorité étant réalisés avant ceux de plus faible priorité. Voici la table des priorité classées par ordre décroissant, les opérateurs sur une même ligne ayant une priorité égale.

1) ( ) fonction( )
2) + - not
3) * / div mod and
4) + - or
5) = <> < <= >= >

3.4.2 Instructions de base :

Affectation :

le signe d'affectation en Turbo Pascal est " := " et on a l'instruction suivante :

Variable := valeur ;

"valeur" peut-être une constante, une variable déjà affectée ou une expression évaluable. Si le dernier cas se présente l'expression sera tout d'abord évaluée puis affectée à " variable".
N.B : *" variable" et "valeur" doivent avoir le même type à l'exception d'un cas où on a une valeur de type entier affectée à une variable de type réel.*

Exemple 3.4.3 *:*

```
   Program       affectation ;
      CONST    A=32 ;
      VAR      X,Y,Z : real ;
               I, j : integer ;
BEGIN
    X:=40;   {on charge X par 40}
    Y:= X-3*A; {Y est affectée par une expression}
     I := A ;   {I est effectué par une constante }
     J := A div 3 ;
     Z:=  J/I+TRUNC(Y)/3;
     ....
END.
```

Remarque 3.4.2 *:*
Quelques fonction prédéfinies :

TRUNC : La partie entière d'un nombre (TRUNC(14.3)=14)
ROUND : l'entier le plus proche d'un nombre (Round(0.9)=1)
SQR : le carré d'un nombre (SQR(4)=16
SQRT : le racine carrée d'un nombre (SQRT(4)=2)
EXP : l'exponentiel d'un nombre ($exp(x) = e^x$)
COS : cosinus
SIN : sinus
ARCTAN : arctangente

3.4.3 Affichage et saisie de données :

Lecture(Read) :

L'instruction de lecteur en Turbo Pascal s'écrit :
Read (VAR1, VAR2,...,VARn) ;
Où VAR1, VAR2, ...,VARn sont les noms des variables à lire .
A "READ", s'ajoute une autre instruction dite **Readln** *qui assure la fonction de READ et le déplacement du curseur à la tête de la ligne suivante. Sa syntaxe est pareille à celle de READ :*
READLN (VAR1, ...,VARn) ;

Exemple 3.4.4 *:*
Cas 1 :
READ(X,Y) ; READ(Z ,U,V) ;
A l'exécution, on entre à la ligne (1) : 3 4 5 6 8 entrée ou ↩
Après exécution : X=3, Y=4, Z=5, U=6 et V=8.
Cas2 :
READLN(X,Y) ; READ(Z ,U,V) ;
A l'exécution : on on tape au clavier :
Ligne(1) : 3 4 5 6 8 entrée ou ↩
Ligne(2) : 10 11 30 entrée ou ↩
Après exécution : X=3, Y=4, Z=10, U=11 et V=30.
Ecriture(write) :
La syntaxe de cette instruction est donnée par :
Write('Message éventuel', Result1,..., Resultn) ;
Avec :

* *Result1,..., Resultn sont des résultats éventuels que, peut-être, il faudrait afficher sur l'écran.*
* *'Message éventuel', c'est un commentaire accompagnant les résultats dans un but d'orientation ou d'éclaircissement.*

Remarque 3.4.3 *:*

- *Les result i (i=1,...,n), après exécution de l'instruction : "Write('Message éventuel', Result1,..., Resultn) ;", seront affichés l'un à coté de l'autre.*
- *Si 'READLN ' coopère avec 'READ ' pour enrichir l'action de lecture alors de même l'écriture est renforcée par 'WRITELN'. Celle-ci, après chaque ordre d'écriture, fait passer le curseur au début de la ligne suivante (retour chariot).*

Exemple 3.4.5 *:*
i)− write :

```
 e :=0.1;   r:=20.1 ;
write('l"erreur est :',e );
write('le résultat est :',r );
```

Après exécution, on a sur l'écran :

```
L'erreur est 0.10000 E00 Le résultat est : 20.1000
```

ii)− writeln :

```
e :=0.1 ;r :=20.1 ;
writeln('l"erreur est :', e );
writeln('le résultat est :', r );
```

Après exécution, on a sur l'écran :

```
L'erreur est 0.10000 E00
le résultat est : 20.1000
```

Remarque 3.4.4 *:*

1. *l'instruction "writeln (chose) ; " est équivalente à :* $\begin{cases} \textbf{write(chose) ;} \\ \textbf{writeln ;} \end{cases}$

2. *write (result : n) ; result est entier*
   n : est le nombre de colonnes qu'on choisit volontairement pour écrire l'entier 'result'.
3. *write (result :n :m) ; result est réel*
   n : nombre de colonnes réservées pour l'affichage de la partie entière.
   m : le nombre de colonnes réservées pour l'affichage de la partie fractionnaire.

4. *si n, dans le cas entier ou réel, est inférieur au nombre de chiffres de la partie entière, on l'ignore.*

Exemple 3.4.6 *:*

```
Program Essai ;
     VAR data1,data2 :real ;
     Car :CHAR;
    i,j : integer ;
Begin
    Writeln('donnez votre première donnée :')
    Readln (data1) ;
   Write ('donnez la deuxième :') ;
   Readln(data2) ;
   i :=trunc (data1+data2) ;
   j :=trunc (data1-data2) ;
  car:= CHR(abs(l*j));
  write ('Le caractère qui a pour code ASCII', ABS(I*J), 'est :' , car)
END.
```

N.B *: le point virgule situé juste avant END est facultatif.*

Chapitre 4

Instructions de répétition et de test

Ce sont des instructions qui contrôlent le nombre de répétition d'exécution d'une ou de plusieurs autre instructions, suite à un teste de validation d'une condition clé. On en distingue "while... do ..." , "Repeat ... until ... ", et "For ... do ... ". Aussi, existe-il d'autres instruction qui permettent le contrôle d'une simple exécution (sans répétition) toujours en se référant à la vérité d'une condition, telles que : "if ... then ... else ...",...

4.1 Instructios de répétition :

4.1.1 While... do... :

Cette instruction signifie "tant que ... faire". Sa syntaxe est donnée ci-dessous par :

While (condition) do (instruction) ;

Elle permet de répéter l'exécution de **(instruction)** *tant que* **(condition)** *reste valide*

Remarque 4.1.1 *:*

* *instruction peu-être simple ou composée.*
* *Une instruction est dite composée si elle est constituée de plusieurs instruction rassemblées entre begin et end, sinon elle est simple.*

Exemple 4.1.1 *:*

```
Program moyenne ;{ programme qui calcule la moyenne des N premiers entiers}
     VAR S,I,N : integer ;
```

```
          M: real;
Begin
  Write('Donner N') ;
  Readln(N) ;
  I :=0 ;S :=0 ;
  While I < N do
     begin
        S : = S + I;
        I :=I+1
     END ;
   M: =S/N;
   Writeln ("la moyenne est :',M :2 :3") ;
   Readin
END.
```

4.1.2 Repeat... until... :

Cette instruction signifie répéter .... jusqu' à . Pour l'employer, on n'a qu'à respecter la syntaxe :

```
Repeat instruction 1 ;
       Instruction 2 ;
           ...
       instruction n ;
until    (condition) ;
```

Elle permet comme le WHILE de répéter l'exécution des **(instructions)** *jusqu' à ce que* **(condition)** *soit validée (true).*

Exemple 4.1.2 *:*

```
Program  Bonjour ;
   VAR  C :CHAR ;
   Begin
      Repeat
        Writeln ('Good morning Youssef') ;
        Readln ;
        Writeln ('How are you, Happy ?  (Y)es or (N)o') ;
        Realdln (C) ;
     Until C='Y';
     Writeln ('Off........!!! I'm too !') ;
```

```
    Readln
  END.
```

Remarque 4.1.2 *:*
Quoique les deux instruction "repeat ..." et " while ... " réalisent la même tâche : répétition, leurs stratégies de manœuvre sont différentes. En effet, deux points non communs peuvent être clairement notés :

1- *'while', avant toute exécution, teste la vérité de sa condition cependant 'repeat' n'effectue ce test qu'à la fin de chaque répétition.*

2- *Lors du déroulement de l'exécution de 'while', sa condition est True, alors que pour 'repeat' c'est le cas inverse (la condition est False).*

4.1.3 For ... Do ... :

Lorsqu'on sait à l'avance le nombre de fois l'exécution d'une ou de plusieurs instructions sera répétée, on fait appel à l'ordre " For... Do... ". Son emploi se fait comme suit :

For <var entière > : = <valmin> to <valmax> Do
<instruction> ;

ou bien

For <var entière > : = <valmax> downto <valmin > Do
<instruction> ;

Avec :
i) " <var entiére > " : c'est une variable dite de contrôle ; elle doit-être de type entier.
ii) " <valmin> " et " <valmax>" : ce sont les valeurs entières, minimale et maximale de " <var entiére > ".
iii) "< instruction>" : peut-être simple composée.

Exemple 4.1.3 *:*

```
Program  for-to ;
  Uses        CRT;
     Const     Min = 9 ; Max = 15 ;
     VAR       i,j integer ;
               S :real ;
  Begin
    Clrscr;
    S := 0 ;
    for i := Min to Max do
```

```
    S := S+1/i ;
  J := 30;
  for i := 20 to j do
    begin
      writeln ('You are welcome in our world !') ;
      Writeln ('Please, set down and begin your work !');
   End;
    S := 0 ;
  For j := 15 down to 1 Do
    S := S+1/j ;
  Writeln ('la somme des inverses des entiers allant de 1 à 15 est', S :2 :3) ;
 Readln
END.
```

Remarque 4.1.3 *:*
Le nombre de répétition pour 'FOR' est donnée à l'avance, pour 'repeat' et 'while', il ne l'est pas.

4.2 Instructions conditionnelles :

4.2.1 If ... Then ... Else ...

C'est la traduction de "Si ... alors ... sinon ...". Sa syntaxe est donnée par :

If (condition) then (instruction 1) else (instruction 2) ;

*Si (condition) est True, (instruction 1) sera exécutée sinon (instruction 2) le sera. Dans le cas où (instruction 2) n'existe pas ; "***If ...Then ... Else ...** *" se réduit à "***If (condition) Then (instruction 1) ;***"*

Exemple 4.2.1 *:*

```
Program devinette ;
     VAR       J,C,CC :CHAR ;
     Begin
         J := 'O' ;
         Cc:= 'P' ;
         While J = 'O' do
            begin
               Write ('Devinez le caractère cc') ;
                  Readln (c) ;
```

```
                    If  c = cc then
                    writeln ('c"est GAGNE !')
                else
                    Writeln ('si vous voulez recommencer, tapez O !')
                    Read (J)
            END ;
        Readln
    END.
```

4.2.2 Cose ... of ...END :

C'est une généralisation de l'instruction "If... Then ... Else...". Son intérêt réside dans le fait qu'elle permet de tester et d'exécuter une instruction parmi plusieurs. Son mode d'emploi est donné ci-dessous :

```
CASE (var_test) of                ou       case (var_test) of
    cas 1 : instruction 1 ;                    cas 1:instruction 1 ;
    cas 2 : instruction 2 ;                    cas 2:instruction 2 ;
              ...                                        ...
    cas n : instruction n;                     cas n : instruction n
    else instruction (n+1)                 END;
END;
```

(var_ test) est une variable ; cas 1,...,cas n, sont les valeurs éventuelles de(var_ test).

Remarque 4.2.1 *:*
i)− "CASE ..." résume les instruction conditionnelles suivantes :

```
if var_test = cas 1 then instruction 1 ;
if var_test = cas 2 then instruction 2 ;
            ....
if var_test = cas n then instruction n ;
```

ii)− Remarquez qu'il n'y a pas de ; devant le ELSE.

Exemple 4.2.2 *:*

```
    Program         chiffre ;
        Var i :integer ;
     Begin
        Writeln ('donnez un chiffre 0...9') ;
        Readin (i) ;
```

```
    Case i of
       0: writeln('c"est Zéro');
       1: writeln('c"est Un');
       2: writeln('c"est Deux');
       3: writeln('c"est Trois');
                 ....
       9: writeln('c"est Neuf');
  END ;
  Readln
END.
```

Chapitre 5

Types de données structurés.

Ce sont des types de données composés de types élementaires tels que integer, real, Char, type énuméré, type intervalle. Certains d'entres eux représentent des informations homogène ; quant aux autres, leurs données sont hétérogènes. On en distingue le type tableau (ARRAY), le type enregistrement (RECORD), le type fichier (file) .....

5.1 Types scalaires définis par le programmeur :

5.1.1 Type intervalle :

ce type décrit une suite de valeurs naturellement classées (nombres entiers ou caractères) limitée de part et d'autre par deux bornes constantes extrêmes. Pour le déclarer, on n'a qu'à appeler :

Type nom=valinf..valsup ;

Exemple 5.1.1 *:*

```
   ...
Type mois=1 .. 12; {intervalle mois}
jour=1.. 31; {intervalle jour}
lettre_majus='A' .. 'Z' {intervalle alphabet majuscule }
lettre_minus='a'.. 'z' {intervalle alphabet minuscule }
chiffre=0..9 ;{intervalle chiffre}
Var  m: mois;
    j: jour;
    Mlettre : lettre_majus;
    lettre : lettre_minus ;
    c: chiffre;
    ....
```

5.1.2 Type énuméré :

il représente une liste de constantes dont chacune, automatiquement, se repère par sa position au sien de cet ensemble. La première constante correspond à la position 0. Le type énuméré se déclare comme suit :

Type nom=(const1, const2,...) ; *{Elles sont classées selon leur succession}*

Exemple 5.1.2 *:*

```
type Semaine=(lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche);
    Annee_solaire=(Jan, Fev, Mar, Avr, Mai, Juin, Juil, Aout, Sept,
                   Oct, Nov, Des);
    Annee_lunaire=(MOHARAM, SAFARE,JOUMADAI, JOUMADAII, RABIEI, RABIEII,
          RAJAB, CHAABAN, RAMADAN, CHOUAL, THI EL KIADA, THI EL HIJJA);
      ..........
```

5.2 type tableau (ARRAY) :

c'est un type prédéfini qui offre la possibilité de construire un tableau de donnée (sous forme de vecteur ou matrice). Pour déclarer des variables sous ce type, on procède comme suit :

```
TYPE VECTEURE=Array[type intervalle ou énuméré] of type;
     MATRICE= Array[type intervalle ou énuméré, type intervalle
                                                ou énuméré] of type;
VAR       X,Y : VECTEURE;
          V,W : MATRICE;
Ou bien:
VAR  X,Y :Array[type intervalle ou énuméré] of type;
    V,W= Array[type intervalle ou énuméré, type intervalle ou énuméré] of type;
```

Commentaire :
i) Vecteur : c'est un nom, à titre d'exemple, attribué au type ainsi défini .
ii) Type intervalle ou énuméré : Il définit les indices des composantes du tableau ou de la matrice.
iii) Type : c'est le type des donnés rangées le vecteur ou dans la matrice.

Exemple 5.2.1 *:*

```
VAR   Vect1,  Vect2 : Array[1..13] of real ;{vecteur}
      Mat1, Mat2, Mat3 : Array[3..4,6..10] of integer ;{matrice}
      X,Y,Z : Array[1..5,1..7,1..4] of CHAR ;{vecteur matrice}
```

Exemple 5.2.2 *:*

```
TYPE     TABVEC = Array[1..7] of CHAR ;
         TABMAT = Array[1..5,1..6] of real ;
         TABVECMAT = Array[1..7,1..7,1..10] of integer ;
VAR      X,Y : TABVEC ;
         Matrice1,Matrice2 :TABMAT ;
         VM1,VM2 ,MV3 :TABVECMAT ;
```

Pour manipuler ou représenter les composantes d'un tableau, on respecte les notations suivantes :

- *X[i] ou X(.i.) avec I =1....7 (cas du vecteur X déclaré ci-avant en Exmple4.2.2)*
- *matrice1[i,j] ou matrice1 (.i,j.) avec i = 1...5, j = 1...6 (cas de la matrice déclaré en Exmple4.2.2)*
- *VM1[i,j,k] ou VM(.i,j,k.) avec I = 1...7 , j = 1...7 , k = 1...10 (cas d'un vecteur de matrice déclaré en Exmple4.2.2*

i, j et k sont des variables de type integer, intervalle ou énuméré. Elles jouent le rôle des indices.

Exemples 5.2.3 *:*

```
Exemples1 :
 TYPE intervalle = 1..12 ;
      semaine=(Lun, MAR, MER, JEU, VEN, SAM, DIM);
      Vect1: Array[intervalle] of real;
      Vect2: Array[semaine] of integer;
      Vect3: Array['A'..'Z'] of CHAR;
      MAT: Array[intervalle, semaine, 'A'..'Z'] of bayte ;

           MAT [1,Lun, 'A'] est élément de MAT
           Vect1 [3]   est élément  de Vect1
           Vect2 [DIM] est élément  de Vect2
           Vect3 ['X']    est élément  de Vect3

Exemple2 :
  Program produit_scalaire;
     uses CRT;
     const MAX=10;
           MIN=1;
     VAR V,W : Array[MIN..MAX] of real;
         I: integer;
         ps: real ;
```

```
  begin
    Writeln ('saisie des vecteurs V et W');
    ps = 0;
    For i: = MIN to MAX Do
     Begin
       write ('v(',i:2,' ='); readln (v[i]);
       write ('w(',i:2,' ='); readln (w[i]);
       ps:=ps+v[i]*w[i];
    END;
   Writeln (' Le produit scalaire calculé est:', ps:2:3) Readln
END.
```

Remarque 5.2.1 *:*
Pour minimiser la mémoire réservée aux composantes d'un tableau déclaré, on adjoint à "ARRAY" un mot de compactage, "Packed", comme ci-dessous :

```
TYPE Nom =Packed ARRAY[type intervalle] of type;
VAR      variable: Nom;
```

Ou bien:

```
VAR  variable: Packed ARRAY[type intervalle] of type;
```

5.3 Chaîne de caractères (string) :

une variable ou une constante de ce type sont des suites de caractères, de tailles différentes (au plus 255 caractères). Une constante est toujours mise entre qoutes ('..'). Une variable se déclare de la façon suivante :

```
VAR  variable,.... :string[entier];{la longueur maximale de la chaîne=entier}
```

ou bien :

```
VAR variable,..... :string; {dans ce cas la chaîne peut avoir une longueur
                                        de 255 caractères}
```

avec 'entier' varie entre 1 et 255
Comme opérateurs s'applique sur les chaînes, on note + (réunion) , =, <>, <, >, >=, <= (comparaison) , lenght(longueur de la chaîne).

Exemple 5.3.1 *:*

```
Program chaine ;
  VAR x,y :string[10] ;
      i,j : integer ;
```

```
Begin
 Writeln('Entrer une chaine 1') ;
 Readln(x) ;
 Writeln('Entrer la chaine 2') ;
 Readln(y) ;
 i=length(x) ;
 j=length(y) ;
 Writeln('la longueur de la chaine 1 est :', i, et de la chaine 2est :', j);
 Writeln('la longueur de', x+y,'est :' i+j) ;
{si on veut afficher les chaines sur deux colonnes l'une à coté de l'autre}
   For i :=1 to 10 do
      Writeln(x[i],'    ',y[i]) ;
 Readln
END.
```

5.4 Type Enregistrement (RECORD) :

le type record représente l'ensemble de tous les enregistrement. C'est un type composé de plusieurs éléments dits "champs" dont chacun possède son propre type. On procède pour le décaler de la manière suivante :

```
TYPE nom=RECORD
      Champ1: type1;
           ...
      Champn: typen;
   END;
VAR enregistrement : nom ;

Ou bien :
VAR enregistrement: RECORD
      Champ1: type1;
          ....
      Champn: typen;
   END;
```

Avec :
i) nom : nom de type enregistrement ainsi défini
ii) enregistrement : nom d'une variable déclarée sous ce même type
iii) champ1...champn : ce sont les noms des composantes de enregistrement, dits "champs".

Exemple 5.4.1 *:*

```
TYPE Etudiant=Record
        Nom, prénom: string [10];
        Numero: integer;
        Note: real
     END;
     Formation = Record
         NBRAN: integer;
         Nom:string;
     END;
 VAR    E1,E2: Etudiant;
        F1, F2, F3:formation;
        Date:record
          mois, jours, an : integer
        END;
        Vect: Array[1..30] of Etudiant;
```

*Pour manipuler un champ, on l'appelle par : "***(nom de la variable enregistrement).(nom du champ)***", tels que par exemple*
Date.mois, F1.NBRAN, E1.Note ...
Dans le cas ou certains champs sont conditionnellement déclarés, on agit pour mentionner cet aspect comme ci-dessous :

```
 TYPE    nom=Record
           Champ1:type1;
           .
           .
           .
           case condition : type of
                  cas1(champs11:type11);
                  cas2(champs22:type22);
                  .
                  .
                  .
                 casn(champsnn:typenn);
           end;
```

afin de ne pas répéter à chaque appel de champs le nom de enregistrement et alléger ainsi l'écriture du programme, on utilise l'instruction "with....do...." dont la syntaxe est suivante :

with(nom de enregistrement) do (instruction) ;

Exemple 5.4.2 *: Sans "with" :*

```
Program DATE ;
    TYPE DAT =Record
           Jour, mois, annee :integer ;
         END ;
    VAR DT :DAT ; Begin
         DT.jour :=7 ;
         DT .mois :=7 ;
         DT.annee :=2006;
    Writlen(DT.jour, '/', DT.mois, '/', DT.annee) ;
    Readln
END.
```

Exemple 5.4.3 *: Avec "with" :*

```
Program DATE ;
  TYPE DAT = Record
         Jour, mois, annee :integer ;
       END ;
 VAR DT :DAT ;
 Begin
    With DT do begin
        jour :=7 ;
         mois :=7 ;
         annee :=2006
         Writlen(jour, '/',mois, '/', annee) ;
    END ;
   Readln
 END.
```

5.5 Ensemble(SET) :

Le type SET est le domaine de tous les ensemble qui seront déclarés. Un ensemble est une collection d'éléments quelconque mais de types scalaire. Ces élément ne sont pas manipulés individuellement mais en ensemble. Pour déclarer des variable de ce type, on procède de la manière suivante :

```
      Type     <nom de l'ensemble>= set of <type scalaire> ;
      VAR     VARIABLE : <nom de l'ensemble>,
Ou bien:
      VAR     VARIABLE : set of <type scalaire> ;
```

Remarque 5.5.1 *:*
Type scalaire ne doit pas comporter plus de 256 valeurs (exemple :CHAR, type intervalle, type énuméré)

Exemple 5.5.1 *:*

```
TYPE lettre = set of CHAR ;
VAR OO,XX,ZZ,alphanum :lettre ;

Soit par exemple :
 XX :=['A','C','E'] ;
 YY:=['U','V','O'] ;
 ZZ :=['A','Z','K'] ;
 OO :=[ ] ;{ensemble vide}
 Alphanum :=['A'..'Z','a'..'z','0'..'9'] ;
```

*Les opérateur qu'on peut applique aux ensembles, sont : + (union), * (intersection), - (différence), = (égalité), <> (inégalité),<= ou >=(inclusion) et IN (appartence)*

XX+YY=['A','C','E','U','V','O']
*XX*ZZ=['A'], XX*YY=[ ]*
*XX*ZZ ⊂ XX+YY est true*
'A' in XX est true
XX=YY est false, XX<>YY est true

5.6 Fichier(File) :

Un fichier est une collection d'élément homogène. Sa déclaration se fait comme suit :

```
Type nom=file of type ;
VAR varfile,...:nom;

Ou bien :
  VAR varfile, .... : file of type ;
```

Avec type : est un type quelconque mais autre que file.

Exemple 5.6.1 *:*

```
 Technicien= Record
        Nom, prenom :string ;
        Matricule :integer ;
 END ;
VAR Tech :file of technicien ;
```

Annexe A

Exemples du calcul numérique

A.1 Interpolation d'une fonction continue par un polynôme

Position du problème :
Le but de cet exercice est de compléter un programme permettant d'interpoler une fonction continue par un polynôme (sect. 1.2, 1.3, 1.4 de la polycopie du calcul numérique). Soit $f : [a, b] \rightarrow \mathbb{R}$ *une fonction continue donnée et soit* $t_0 < t_1 < t_2 <, \cdots, < t_n$, *(n+1) points de [a,b]. On cherche un polynôme p de degré n tel que*

$$P(t_j) = f(t_j), 0 <= j <= n.$$

D'après le cours le polynôme p est donné par :

$$p(t) = \sum_{j=0}^{n} f(t_j)\varphi_j(t),$$

Où $\varphi_0, \varphi_1, \cdots, \varphi_n$ *est la base de Lagrange de* $\mathbb{P}$ *associée à* $t_0, t_1, \cdots, tn$. *Les fonction de base de Lagrange son définies par*

$$\varphi_j(t) = \prod_{\substack{i=0 \\ i \neq j}}^{n} (t - t_i)/(t_j - t_i)$$

Travail demandé : *A partir la donnée de a, b, f, n le programme Lagrange permet de calculer le polynôme p qui interpole f aux points* $t_j, 0 \leq j \leq n$ *, équidistribués uniformément sur l'intervalle* $[a, b]$. *Ce programme est incomplet, en particulier vous devez programmer la formule (1). Une fois le programme écrit et compilé. Son exécution aura pour conséquence de calculer la valeur de p à des points donner par l'utilisateur et compare le résultat avec la valeur de f à ce point.*
Exemple :*(expemle 1.2 de la polycopie du calcul numérique)*

```
Program LAGRANGE;
                   {Etant donne un entier n et une fonction continue f,
                    Le programme interpole la fonction f par un polynôme
                    p de degré n aux points t[0],t[1], Ě,t[n]
                    paramètre :
                        n      : degré du polynôme p (interpolant de Lagrange)
                        f          : la fonction à interpoler
                        t[j]       : est le j-eme point d'interpolation }
uses crt;
 Const NMAX=1000;
  Var   t,y         : Array [1..NMAX] of real;
        I, J, N, K, M : Integer;
        x,tt, pol_k,pol,phi_j  :real;
        rep:char;
Begin clrscr;
  write('donner le nombr de ponts N:');
  readln(N);
  For I:=0 to N Do
  Begin
    write('donner x(',i,')=');
    readln(t[I]) ;
     write('donner f(',i,')=');
    readln(y[I]) ;
  End;
  { N := 80;
  {Les points d'interpolation t[j] sont equidistribues entre -1 et +1}
  For I:=0 to N Do
  Begin
    t[I] := -1+(2*I)/N ;
    Writeln(t[I]);
    readln;
  End;}
  rep:='o';
  while rep='o'do
  begin
   write('donner x:');
   readln(x);
   {M:=100;
  For K := 0  to M Do
    begin
        x_k:=-1+(2*K)/M;}
        pol:=0;
```

```
        For J := 0 to N  Do
         {calcule de phi_j (la j_eme fonction de base de Lagrange)
              au point x (formule de la polycopie)}
          begin
             phi_j:=1;
             For I := 0 to N Do
                begin
                   If I<>J  then
                     phi_j:={A COMPLETER};
                 end;
                 tt:=t[j];
                 {Evalue p au point x}
             pol:=pol+phi_j*y[j]                {(3+4*tt*tt-tt);}
          end;
        pol_k:=pol;
        writeln('x=',x,'     pol_k=',pol_k,'   fx=',3+4*x*x-x);
        write('voulez-vous faire un auter aissai (O)ui ou (N)on?');
        readln(rep);
    end;
end.
```

A.2 Intégration numérique

A.2.1 Formule du trapèze

Position du problème :
Soit $f : [a, b] \to \mathbb{R}$ une fonction continue donnée. Le but de cet exercice est de compléter et d'utiliser un programme permettant d'approximer

$$\int_a^b f(x)dx(1)$$

en utilisant la formule du trapèze à 2 points d'intégration (chap. 3 de la polycopie du calcul numérique). Pour ce faire l'intervalle $[a, b]$ est partitionné en N intervalles $[x_i, x_{i+1}], i = 0, 1, 2, \cdots, N-1$ de même longueur $h = (b - a)/N$.
Travail demandé :
A partir de la donnée de f, a, b, N le programme trapeze permet d'approcher (1). Ce programme est incomplet, en particulier vous devez :

1. *monter que*
2. *programmer la formule (2).*

Une fois le programme écrit et compilé. Son exécution aura pour conséquence de calculer (d'approximer) (1).

```
 Program  trapeze;
           {Etant donné un intervalle [a,b] et une fonction continue f,
           le programme approche l'intégrale de f entre a et b.
           Paramètres :
             a, b    : l'intervalle [a, b]
             n       : l'intervalle [a, b] est partitionné en N intervalles
             f       : la fonction à intégrer}

  VAR n,i:integer;
      a,b,integral,s,h,fx,fa,fb:real;
{Function  f(x:real):real;
 begin
    f:=exp(x)*cos(x);
end;
Function df(x:real):real;
 begin
    df:= exp(x)*cos(x)-exp(x)*sin(x);
end;}

 BEGIN
    write('Entrer la valeur de a=');
    readln(a);
    write('Entrer la valeur de b=');
    readln(b);
            n :=100 ;
    writeln('METHODE DE TRAPEZE');
    {writeln('t  In(f)\t\t  It(f)\t\t  Erc\t\t  Ern\t\t  Ern/Er2n\n");}
    writeln('_____________________________________________________________');
    for i:=1 to 8 do
    begin
    n=:2*n;

    h:=(b-a)/(n+1);
    s:=0;
        for i:=1 to n do  {Formule de trapèze A COMPLETER }
            fx:=exp(a+i*h)*cos(a+i*h);
            s:=s+fx;
        fa:=exp(a)*cos(a);
        fb:=exp(b)*cos(b);
        intgral:={A COMPLETER };
```

```
    Writeln(Integral,'      ',-12.0703463164-Integral);
    writeln('_____________________________________________________________');
    end;
readln;
 END.
```

A.2.2 formule de Simpson

Position du problème :
Soit $f : [a, b] \to \mathbb{R}$ une fonction continue donnée. Le but de cet exercice est de compléter et d'utiliser un programme permettant d'approximer

$$\int_a^b f(x)dx(1)$$

en utilisant la formule de Simpson à 2 points d'intégration (chap. 3 de la polycopie du calcul numérique). Pour ce faire l'intervalle $[a, b]$ est partitionné en N intervalles $[x_i, x_{i+1}], i = 0, 1, 2, \cdots, N-1$ de même longueur $h = (b - a)/N$.
Travail demandé :
A partir de la donnée de f, a, b, N le programme Simpson permet d'approcher (1). Ce programme est incomplet, en particulier vous devez : 1

1. *monter que*
2. *programmer la formule (2).*

Une fois le programme écrit et compilé. Son exécution aura pour conséquence de calculer la valeur de (1), et compare le résultat avec le résultat du programme trapeze.

```
Program  Simpson;
         {Etant donné un intervalle [a,b] et une fonction continue f,
         le programme approche l'intégrale de f entre a et b.
         Paramètres :
            a, b    : l'intervalle [a, b]
            n       : l'intervalle [a, b] est partitionné en N intervalles
            f       : la fonction à intégrer}
VAR n,I,p,j:integer;
    a,b,integral,S,L,h:real;
{Function  f(x:real):real; begin
   f:=exp(x)*cos(x);
end; Function df(x:real):real; begin
    df:= exp(x)*cos(x)-exp(x)*sin(x);
{-2*exp(x)*(cos(x)+sin(x));} end;}
```

```
BEGIN
   write('Entrer la valeur de a=');
   readln(a);
   write('Entrer la valeur de b=');
   readln(b);
   n :=100 ;
   writeln('METHODE DE SIMPSON');
   {printf("\t  In(f)\t\t  It(f)\t\t  Erc\t\t  Ern\t\tErn/Erc\n");}
   writeln('_____________________________________________________________');
   for i:=1 to 8 do
   begin
           p:=n;
   n:=2*n;
   h:=(b-a)/(2*p);
   S:=0;
   L:=0;
       If p>1 then    {Formule de simpson A COMPLETER}
       begin
             for i:=1 to p-1 do
           S:={A COMPLETER};
       End;
       for j:=1 to p do
           L:={A COMPLETER};

       Intgral:=(h/3)*(f(a)+f(b)+2*S+4*L);
   writeln(Integral,'       ',-12.0703463164-Integral);
   writeln('_____________________________________________________________');
   end ;
   readln ;
END.
```

A.3 Méthode de Newton

Position du problème :

Le but de cet exercice est de compléter et d'utiliser un programme permettant de résoudre l'équation $f(x) = 0$ grâce à la méthode de Newton.
La méthode
L'approche graphique est la suivante : On choisit une valeur d'abscisse raisonnablement proche du vrai zéro. On remplace alors la courbe par sa tangente et on calcule le zéro de l'approximation affine associée à la tangente (ce qui se réalise facilement avec l'algèbre élémentaire). Ce zéro de la tangente sera

généralement plus proche du zéro de la fonction, et la méthode peut être réitérée.

En pratique, voici les opérations à poser pour $f : [a, b] \to \mathbb{R}$, fonction définie et dérivable et sur l'intervalle $[a, b]$, et à valeurs réelles. Choisissons une valeur arbitraire x_0 (le plus près du zéro est le mieux). On définit alors, par récurrence pour chaque nombre naturel n :

$$x_{n+1} = x_n - f(x_n)/f'(x_n)$$

où f' désigne la dérivée de la fonction f.
On peut prouver que, si f' est continue et si le zéro inconnu a est isolé, alors il existe un voisinage de a tel que pour toutes les valeurs de départ x_0 dans ce voisinage, la suite (x_n) va converger vers a. De plus, si $f'(a) \neq 0$, alors la convergence est quadratique, ce qui signifie intuitivement que le nombre de chiffres corrects est approximativement doublé à chaque étape.
Travail demandé :
A partir du programme Newton et la donnée de f, x_0 , ce programme résoud l'équation $f(x) = 0$. Ce programme est incomplet, en particulier vous devez programmer la formule (2). Exemples Considérons le problème de trouver le nombre positif x vérifiant $cos(x) = x^3$. Reformulons la question pour introduire une fonction devant s'annuler : on recherche le zéro de $f(x) = cos(x) - x3$.
La dérivation donne $f'(x) = -sin(x) - 3x^2$.
Comme $cos(x) \leq 1$ pour tout x et $x^3 > 1$ pour $x > 1$, nous savons que notre zéro se situe entre 0 et 1. Nous essayons une valeur de départ de $x_0 = 0, 5$.

et les 12 premiers chiffres de cette valeur coïncident avec les 12 premiers chiffres du vrai zéro.

```
Program newton;
uses crt;
 Var   x0, x, eps,fx0,dfx0:real;
       I, ITMAX: Integer;
Begin clrscr; write('donner x0=');
readln(x0);
write('donner eps=');
readln(eps);
write('donner itmax=');
readln(itmax);
for i:=1 to itmax do
  begin
    fx0:=cos(x0)-x0*x0*x0;
    dfx0:=-sin(x0)-3*x0*x0;
    x:=x0-fx0/dfx0;
    if abs(x-x0)<=eps then
       begin
         writeln('racine=',x);
         break;
       end
    else
       begin
         if i=itmax then
             writeln('la methode n''est pas satisfaite')
         else
            begin
                x0:=x;
                writeln(i, '      ', x);
           end;
       end;
   end;
```

```
   readln;
End.
```