



الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي و البحث العلمي

Université Abou Bekr Belkaid
Tlemcen Algérie



جامعة أبي بكر بلقايد

تلمسان الجزائر

Cours N°1

Concepts de Base

*****Mr.BACHIR *****Mr.MOUSSA*****



1. Définition de l'Informatique

- Le mot ***informatique***, proposé par l'ingénieur français Philippe DREYFUS, en 1962, est une contraction des mots ***information*** et ***automatique***.
- Définition accepté par l'Académie Française : "*Science du traitement rationnel, notamment par machines automatiques, de l'information considérée comme le support des connaissances humaines et des communication dans les domaines techniques, économiques et sociaux*".
- L'informatique désigne l'ensemble des sciences et techniques en rapport avec le traitement de l'information.



1. Définition de l'Informatique (Suite)

- L'informatique n'est pas fondamentalement liée à l'utilisation des ordinateurs. Surtout elle se fonde sur des études théoriques de logique, de mathématiques, de linguistique, de grammaire formelle, de compilation et bien évidemment de structure d'ordinateur.

À cet égard, Edsger Dijkstra (Mathématicien et informaticien néerlandais du XXe siècle) disait :
" *L'informatique n'est pas plus la science des ordinateurs que l'astronomie n'est celle des télescopes.* "



2. Branches de l'Informatique

L'informatique est subdivisée en de nombreuses branches plus ou moins spécialisées dont on peut citer:

- ❑ **Informatique formelle ou analytique:** branche de l'informatique la plus proche des sciences exactes
- ❑ **Informatique systématique et logique:** qui étudie l'architecture des systèmes informatiques
- ❑ **Informatique physique et technologique:** qui s'attache à l'étude et à la réalisation des composants et sous-ensembles électroniques
- ❑ **Informatique méthodologique:** qui se rapporte aux recherches en méthodologie de la programmation
- ❑ **Informatique appliquée:** qui s'occupe concrètement de l'application de l'informatique dans les divers domaines de la vie économique, culturelle et sociale



3. Étapes de Résolution d'un Problème Informatique

Pour résoudre un problème informatique, il faut:

- ❑ **Analyser ce problème:** définir avec précision les résultats à obtenir, les informations dont on dispose, ...
- ❑ **Déterminer les méthodes de résolution:** il s'agit de déterminer la suite des opérations à effectuer pour obtenir à partir des données la solution au problème posé. Cette suite d'opérations constitue un **algorithme**.
- ❑ **Formuler l'algorithme définitif:** cette étape doit faciliter la résolution sur ordinateur par l'expression de l'algorithme dans un formalisme adéquat.
- ❑ **Traduire l'algorithme dans un langage de programmation adapté.**



4. Notion d'Algorithme

□ Exemples d'algorithmes :

Exemple1: *Préparation d'un litre de glace*

- ½ litre de lait
 - 6 œufs
 - 200 g de sucre glacé
 - 2 cuillérées de café soluble
- 1) Faire bouillir le lait
 - 2) Battre les jaunes d'œufs
 - 3) Verser dessus le lait bouillant en remuant avec une spatule



4. Notion d'Algorithme (suite)

□ Exemples d'algorithmes :

Exemple2: *Tri d'un jeu de cartes suivant la couleur*

1) Prendre la première carte

2) La carte est-elle rouge?

Si oui, poser la carte sur le premier tas

Sinon, poser la carte sur le second tas

3) Reste-t-il des cartes?

Si oui, prendre la carte suivante et continuer sous 2

Sinon, fin du tri



4. Notion d'Algorithme (suite)

□ Exemples d'algorithmes :

Exemple3: Calcul des racines d'un polynôme du 2nd ordre: $a x^2 + b x + c = 0$, $\forall (a,b,c) \neq 0$

1) Saisir les valeurs de (a, b, c)

2) On calcule $\Delta = b^2 - 4ac$

Si $\Delta < 0$ alors pas de racine dans IR

Si $\Delta = 0$ alors racine double $x = -\frac{b}{2a}$

Si $\Delta > 0$ alors deux racines :

$$x_1 = \frac{-b - \sqrt{\Delta}}{2a} \quad x_2 = \frac{-b + \sqrt{\Delta}}{2a}$$



4. Notion d'Algorithme (suite)

- Un *algorithme* est une suite d'actions qui, correctement exécutées donneront le résultat désiré (attendu).
- Un *algorithme* est le résultat de la décomposition d'un problème complexe en opérations élémentaires à exécuter en plusieurs étapes successives.
- Un *algorithme* est toujours exécuté par un *processeur*. Il peut être une personne, un dispositif électronique, mécanique ou un ordinateur. C'est toute entité en mesure de comprendre et d'exécuter les actions constituant un *algorithme*.
- L'ensemble des objets (éléments) nécessaires à la réalisation d'un travail décrit par un algorithme est appelé *environnement*.



4. Notion d'Algorithme (suite)

❑ Définitions:

- ✓ Un algorithme est une séquence (suite) d'actions élémentaires, qui exécutées par un processeur bien défini réalisera un travail bien précis (demandé).
- ✓ Un algorithme est une suite de règles, de raisonnements ou d'opérations, qui transforment des grandeurs données (données d'entrée) en d'autres grandeurs (données de sortie).

Entrée:

Normalement, un algorithme possède une ou plusieurs données d'entrée [input data], c-à-d des valeurs qui sont connues avant son exécution et sur lesquelles l'algorithme est appliqué.



Sortie:

Un algorithme possède une ou plusieurs données de sortie [output data], c-à-d des valeurs produites par lui-même. Ces données sont en relation exactement spécifiée avec les données d'entrée.



4. Notion d'Algorithme (suite)

□ Propriétés:

- L'algorithme doit tenir compte de tous les cas possibles.
Il traite le cas général et les cas particuliers
- Il contient toujours un nombre fini d'actions
- L'ordre des actions est important (exécution séquentielle)
- Chaque action doit être définie avec précision, sans aucune difficulté
- Certaines actions peuvent être raffinées (décomposées)
- L'algorithme n'est pas nécessairement unique
- Il doit produire le résultat désiré



4. Notion d'Algorithme (suite)

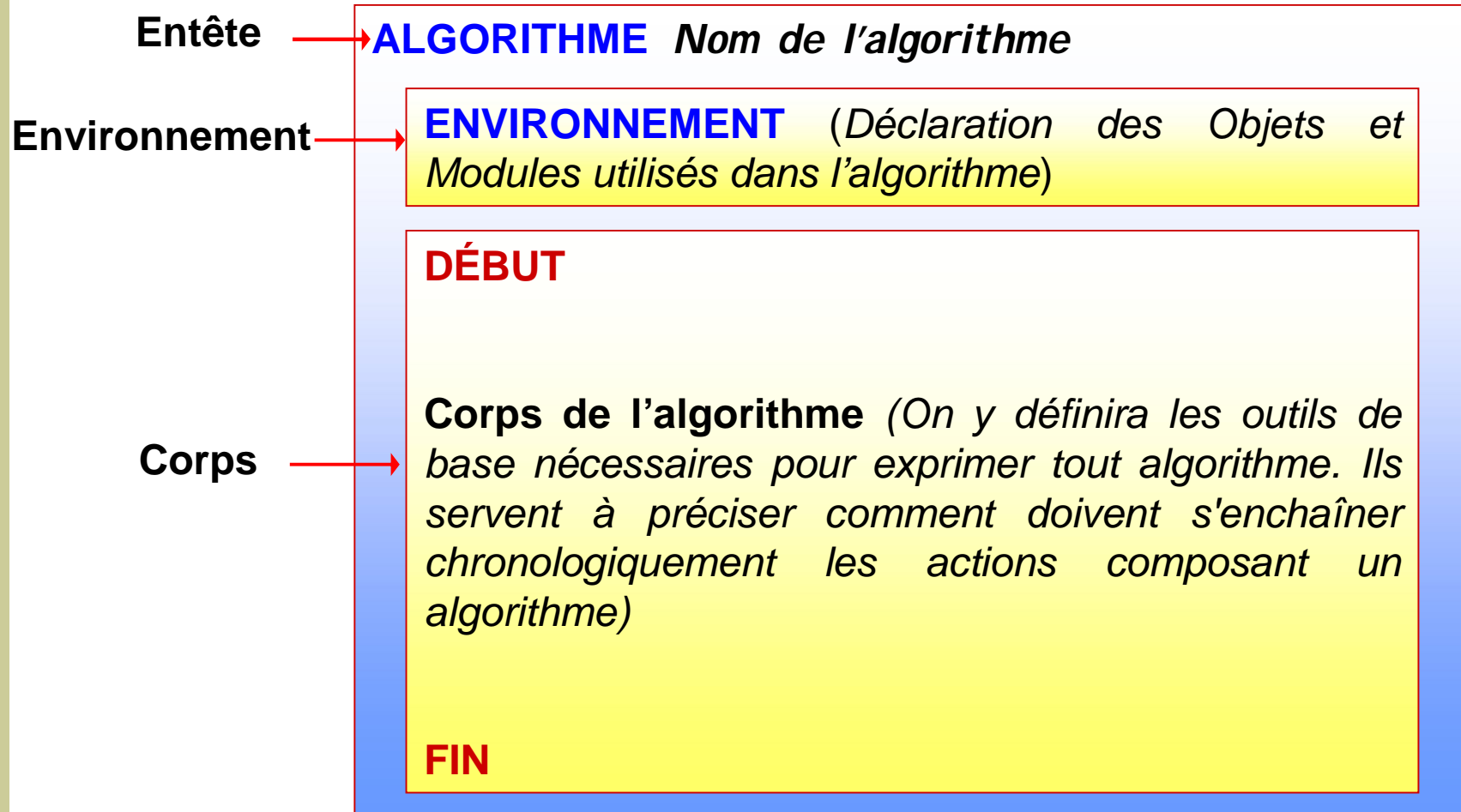
□ Formalisme algorithmique:

Un formalisme algorithmique est un ensemble de conventions (ou de règles) dans lequel on exprime toute solution d'un problème donné.



4. Notion d'Algorithme (suite)

❑ Structure générale d'un algorithme:





4. Notion d'Algorithme (suite)

□ Exemple 1 : *Addition de deux nombres réels*

Algorithme Addition

Variables utilisées:

A, B, Somme : nombres Réels

- 1) Début
- 2) Lire (A,B)
- 3) Somme=A+B
- 4) Écriture (Somme)
- 5) Fin

□ Exemple 2 : *Calcul des racines d'un polynôme du 2^{ème} ordre :*

$$a x^2 + b x + c = 0 , \forall (a,b,c) \neq 0$$



5. Notion d'Organigramme

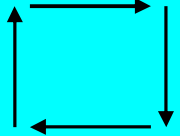


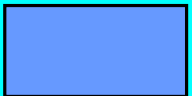
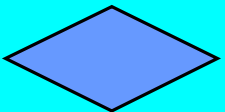
□ Définitions:

- ✓ Un *organigramme* est un schéma symbolique conventionnel qui illustre les étapes d'un algorithme et leurs relations.
- ✓ Nous utilisons l'*organigramme* parce qu'une représentation graphique aide à la compréhension.
- ✓ L'*organigramme* est un schéma fonctionnel qui présente les différentes parties d'un programme les unes à la suite des autres en utilisant des symboles graphiques pour visualiser l'exécution du programme et le cheminement des données.



5. Notion d'Organigramme (suite)

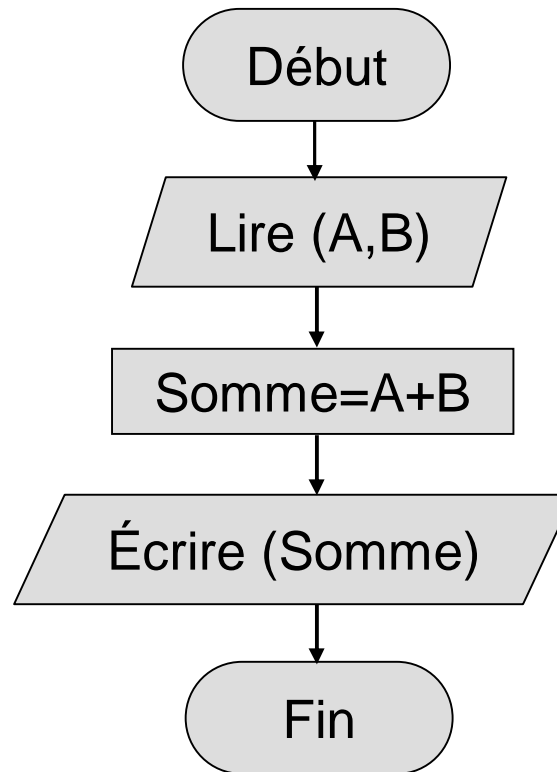
□ Principaux Symboles d'un Organigramme:

Noms	Symbole	Définition
Flèches		Elles indiquent le sens du traitement (haut, bas, gauche, droite).
Début / Fin		Ce symbole indique le début ou la fin de l'organigramme
Entrée / Sortie		Ce symbole indique les données d'entrées et de sorties
Boite de traitement		Elle indique un traitement spécifique qui peut être exécuté
Boite de décision (Test)		Elle permet d'envoyer le traitement sur un chemin ou sur un autre, selon le résultat du test



5. Notion d'Organigramme (suite)

□ **Exemple 1** : *Addition de deux nombres réels*



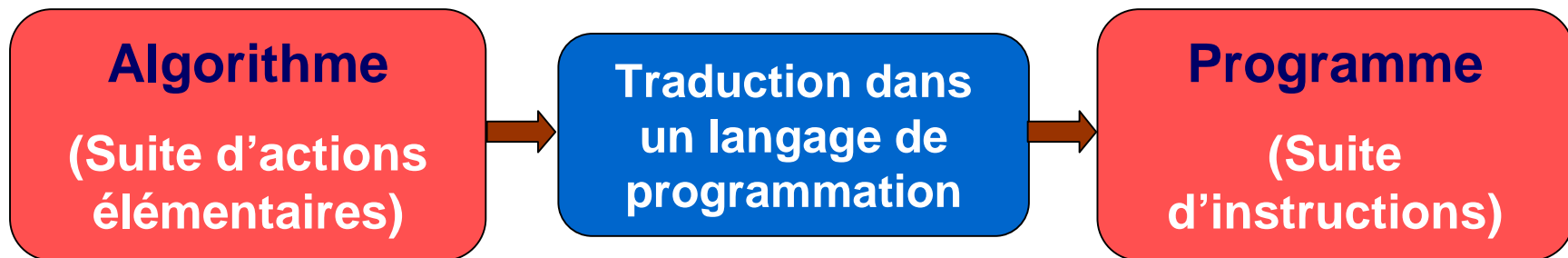
□ **Exemple 2** : *Calcul des racines d'un polynôme du 2nd ordre :*
 $a x^2 + b x + c = 0 , \forall (a,b,c) \neq 0$



6. Programmes et Langages de Programmation

□ Notion de Programme:

Un programme est une séquence d'instructions écrites dans un langage de programmation traduisant un algorithme. Chacune de ses instructions spécifie l'opération que doit exécuter l'ordinateur.





6. Programmes et Langages de Programmation

□ Langage de Programmation:

- Un langage de programmation est un langage artificiel comprenant un ensemble de caractères, de symboles et de mots régis par des règles qui permettent de les assembler, utilisé pour donner des instructions à une machine.
- Les langages de programmation permettent de définir les ensembles d'instructions effectuées par l'ordinateur lors de l'exécution d'un programme.
- Il existe plusieurs langages de programmation, la plupart d'entre eux étant réservés à des domaines spécialisés. Exemple: *Fortran, C, C++, Java, Html, Pascal ...*



6. Programmes et Langages de Programmation

❑ **Compilateur:**

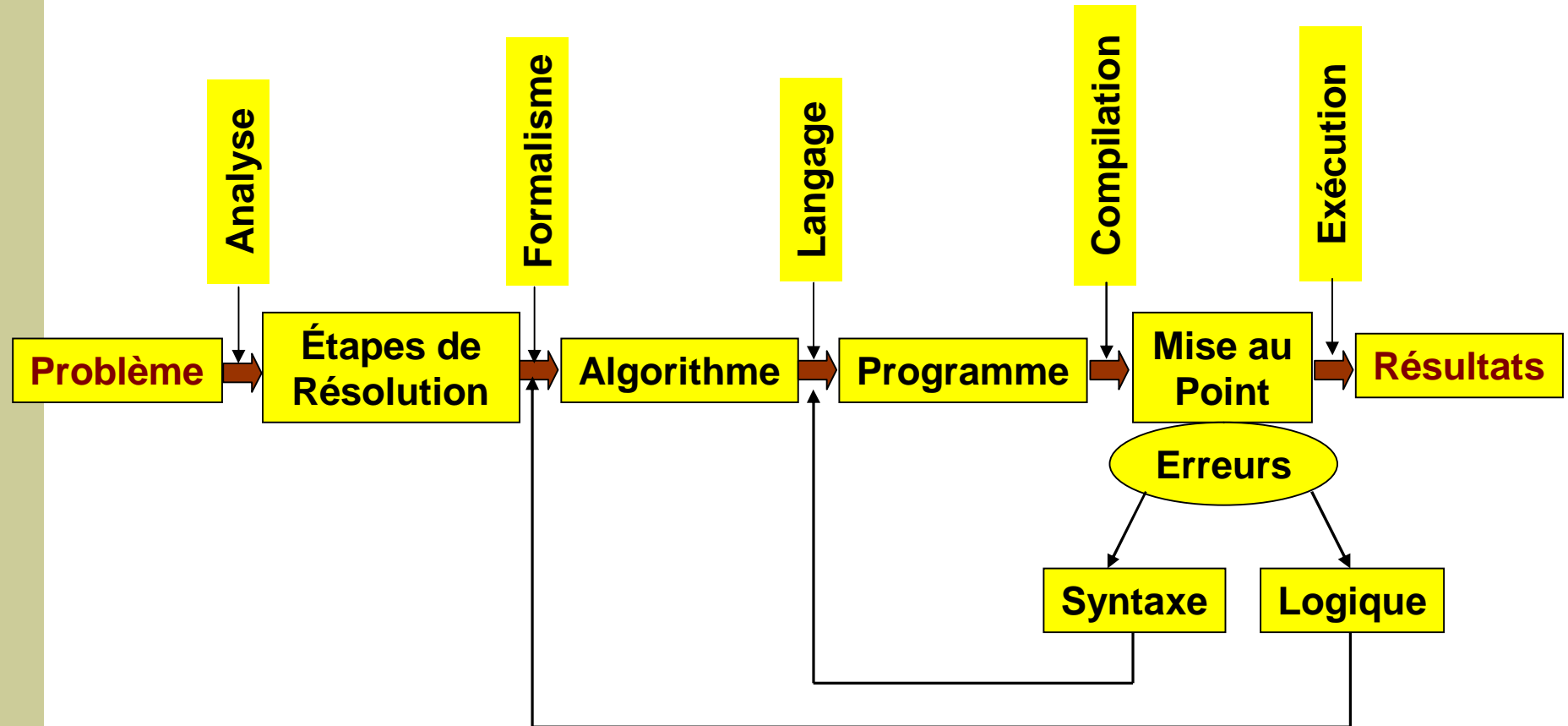
- Tout langage possède un compilateur ou du moins un interpréteur.
- Il sert à traduire le programme écrit avec le langage (programme source) en langage machine (codes) afin qu'il soit compris par l'ordinateur.
- Il permet aussi d'analyser le programme source pour détecter les erreurs de syntaxe commises par le programmeur.





6. Programmes et Langages de Programmation

□ Du problème au résultat:





7. Langage Pascal (Turbo Pascal)

□ Langage Pascal:

- Ce langage a été créé en 1969 à l'école polytechnique de ZURICH par N. WIRTH. Il a été conçu pour permettre d'enseigner la programmation comme une science.
- Ce langage est capable de supporter la programmation structurée et la conception descendante de logiciels.
- Le langage Pascal est un langage compilé c-à-d qu'il faut:
 - ✓ Entrer un texte à l'aide d'un Éditeur (Programme Source)
 - ✓ Le traduire en langage machine à l'aide du compilateur (Programme Compilé)
 - ✓ Exécuter ce programme (Programme Exécuté)



7. Langage Pascal (Turbo Pascal) (suite)

□ Turbo Pascal:

- Turbo Pascal est un environnement de développement intégré pour le langage Pascal.
- Il se compose de plusieurs éléments : un éditeur, un linker, un débogueur et, en plus, de diverses fonctions pour le chargement et la sauvegarde de programmes.
- La première version de Turbo Pascal compatible PC est apparue en 1983



7. Langage Pascal (Turbo Pascal) (suite)

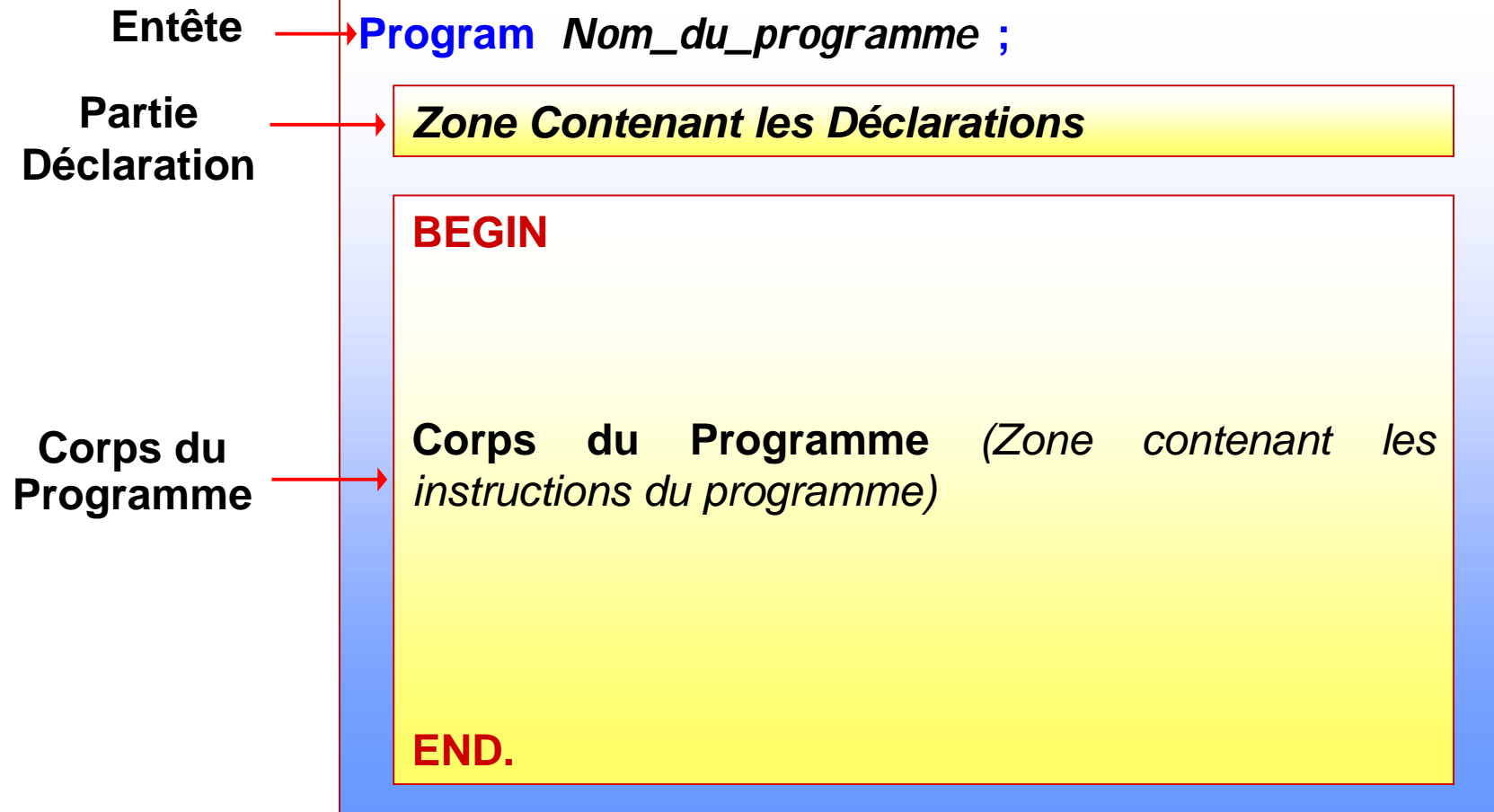
□ Turbo Pascal:

- La dernière version de Turbo Pascal, la version 7 (1991), existait en deux versions :
 - ✓ Turbo Pascal 7, qui comprenait un EDI pour MS-DOS et des compilateurs pour créer des programmes MS-DOS et DOS étendus ;
 - ✓ Borland Pascal 7, qui comprenait *en plus* un EDI pour Windows, qui permettait de créer des exécutables pour Windows.



7. Langage Pascal (Turbo Pascal) (suite)

□ Structure générale d'un programme Pascal:





7. Langage Pascal (Turbo Pascal) (suite)

- **Exemple:** *Addition de deux nombres réels*

```
Program Addition;  
  
Uses wincrt;  
  
Var A,B,Somme:Real;  
  
    Begin  
  
        Read(A,B);  
  
        Somme:=A+B;  
  
        Write(Somme);  
  
    End.
```



Cours N°2

Règles Générales d'Écriture d'un Programme Pascal



1. Les Identificateurs

- Pour manipuler différents objets dans un programme, il faut leur donner des noms.
- Les noms utilisés pour les objets manipulés sont des **identificateurs**.

Définition :

L'identificateur est un nom symbolique utilisé pour nommer (identifier) un objet dans un programme informatique.



1. Les Identificateurs (suite)

Les « objets » dans un programme sont des :

- ✓ **Nom du programme,**
- ✓ **Constantes,**
- ✓ **Variables,**
- ✓ **Types,**
- ✓ **Procédures,**
- ✓ **Fonctions.**



1. Les Identificateurs (suite)

❑ Règles d'écriture d'un identificateur :

Les identificateurs sont représentés par une suite de lettres et/ou de chiffres avec les **restrictions** suivantes :

- ✓ le premier caractère doit être alphabétique, donc une lettre obligatoirement ;
- ✓ les caractères suivant le premier peuvent être numériques ;
- ✓ le caractère souligné « _ » est permis;
- ✓ les caractères dits « spéciaux » c'est-à-dire l'espace et les symboles : parenthèses, signe plus (+), signe moins (-), signe égal (=), point-virgule (;) sont **interdits** ;



1. Les Identificateurs (suite)

❑ Règles d'écriture d'un identificateur :

- ✓ l'utilisation des accents sur les lettres est interdite ;
- ✓ l'utilisation des mots clés (réservés) du langage est interdite ;
- ✓ l'utilisation des minuscules ou des majuscules est permise parce que *TURBO-PASCAL ne fait pas la différence entre les minuscules et les majuscules.*

Exemples :

ValeurM, Valeur_A, AB, B7, Nom

Sont des identificateurs corrects

Valeur M, Valeur-A, A/B, 7B, Nom\$

Sont des identificateurs incorrects



1. Les Identificateurs (suite)

❑ Les mots clés (mots réservés) du langage :

`and, array, begin, const, div, do, downto, else, end, for, function, if, mod, not, of, or, procedure, program, repeat, then, to, type, until, var, while :`

sont des mots standards imposés par Turbo-Pascal ; leur signification et leur rôle sont parfaitement définis. On les appelle **mots clés** ou **mots réservés**.

Attention ! Un mot clé n'est **JAMAIS** accepté comme identificateur.



1. Les Identificateurs (suite)

❑ Les identificateurs prédéfinis :

- ✓ Les identificateurs prédéfinis sont des mots qui ont une signification *par défaut* en Turbo-Pascal.
- ✓ Il s'agit de : `abs`, `arctan`, `boolean`, `cos`, `exp`, `false`, `integer`, `ln`, `read`, `readln`, `real`, `round`, `sin`, `sqr`, `sqrt`, `true`, `trunc`, `write`, `writeln`.
- ✓ La différence par rapport aux mots clés est que les identificateurs prédéfinis peuvent être utilisés comme identificateurs ordinaires.
- ✓ Par exemple, on peut appeler une variable par `REAL` sans avoir des erreurs de compilation, ***mais cela est tout à fait déconseillé.***



2. Les Séparateurs

Définition :

Un séparateur est un espace ou un caractère ou une série de caractères, destinés à séparer des identificateurs.

- En Turbo-Pascal, les différents mots du langage sont séparés soit par un espace, soit par un signe particulier ou une fin de ligne.
- Dans un programme, deux identificateurs successifs doivent être séparés soit par un espace, soit par une fin de ligne. Sinon, le compilateur renvoie un message d'erreur.



2. Les Séparateurs (suite)

□ Exemples de séparateurs :

Séparateurs	Définition	Exemple
:	séparateur permettant de préciser le type d'une variable	<code>VAR a : REAL;</code>
;	séparateur fin de ligne	<code>PROGRAM Essai;</code>
,	séparateur virgule pour séparer des variables	<code>VAR a,b : REAL;</code>



3. Structure d'Ensemble de la Partie Déclaration

- La partie déclaration peut contenir différentes sortes de déclaration. Ces dernières sont introduites par des mots clés.
- Les mots clés utilisés dans la partie déclaration sont:

Uses

Const

Var

Label

Type

Function

Procedure



3. Structure d'Ensemble de la Partie Déclaration

Mots Clés	Signification	Exemples
Uses	C'est une bibliothèque utilisée par le compilateur la ou existe les fonctions et les procédures prédéfinis du langage	Uses Wincrt;
Const	Pour la déclaration des constantes	Const coeff=5;
Var	Pour la déclaration des variables	Var i:Integer; x,y:Real;
Label	Pour la déclaration des étiquettes	Label 10,20,nom;



3. Structure d'Ensemble de la Partie Déclaration

Mots Clés	Signification	Exemples
Type	Pour la définition de <i>nouveaux types</i>	Type Tab=Array[1..50] of Real;
Function	Pour la définition des <i>fonctions</i>	Function Max(a,b:Real):Real;
Procedure	Pour la définition des <i>procédures</i>	Procedure Min(a,b:Integer);

Chaque déclaration est séparée de la suivante ou du début du programme (**Begin**) par un point virgule (;)

Exemple:

```
Program Essai;  
Uses wincrt;  
Const   Coeff=0.27;  
        Nbr_fois=15.5;  
Var     A,B:Real;  
        i,j:Integer;  
Begin
```



4. Types de Données en Pascal

- La notion de **type** est liée à la notion de **donnée**. C'est l'ensemble des valeurs que peut prendre une donnée.
- Donc le type désigne, pour un langage de programmation, deux ensembles :
 - *un ensemble de valeurs désigné explicitement ou par des valeurs extrêmes,*
 - *un ensemble d'opérations permises par le type.*
- Le langage pascal donne la possibilité d'utiliser :
 - *des types de données **prédéfinis**,*
 - *des types de données **définis par l'utilisateur**.*



4. Types de Données en Pascal (suite)

- Une classification nous permet de saisir trois catégories de types en Pascal :
 - 1) Les **types simples** pour lesquels les valeurs ne sont pas décomposables en constituants plus simples.
 - 2) Les **types structurés** qui couvre quatre catégories de données structurées :
 - Les **tableaux** (le type *ARRAY*)
 - Les **enregistrements** (le type *RECORD*)
 - Les **ensembles** (le type *SET*)
 - Les **fichiers** (le type *FILE*)
 - 3) Le type pointeur



5. Types de Données Simples Standards

Il existe 4 types de données simples standards :

- ✓ Le type **Entier**
- ✓ Le type **Réel**
- ✓ Le type **Caractère**
- ✓ Le type **Booléen**



5. Types de Données Simples Standards (suite)

A. Type Entier (Integer) :

Le type Entier spécifié par l'identificateur standard **INTEGER** correspond à l'ensemble des nombres entiers.

Représentation des valeurs

Les entiers s'écrivent en notation décimale avec éventuellement un signe + ou - devant

Exemple : +23
 23
 -6



5. Types de Données Simples Standards (suite)

A. Type Entier (Integer) :

- Les variables associées au type **INTEGER** ne correspondent pas aux ensembles infinis que l'on rencontre en mathématiques.
- L'ensemble des valeurs des entiers pris par une variable de type **INTEGER** est limité et ces limites sont liées à la longueur des mots mémoire utilisés pour représenter ces nombres.



5. Types de Données Simples Standards (suite)

A. Type Entier (Integer) :

Turbo-Pascal permet l'utilisation de cinq types de données prédéfinis pour le domaine des nombres entiers conformément au tableau suivant :

Type	Intervalle	Longueur
Shortint	-128 .. 127	1 octet (8 bits)
Integer	-32 768 .. 32 767	2 octets
Longint	-2 147 483 648 .. 2 147 483 647	4 octets
Byte	0 .. 255	1 octet
Word	0 .. 65 535	2 octet



5. Types de Données Simples Standards (suite)

A. Type Entier (Integer) :

Opérateurs et Fonctions relatifs au type entier :

Type	Notation	Signification	Exemples
Opérateurs de Comparaison	$<$, $>$, $<>$, $=$, $<=$, $>=$		
Opérateurs Arithmétiques	$+$	Addition	$5+8$ vaut 13
	$-$	Soustraction	$9-10$ vaut -1
	$*$	Multiplication	$2*10$ vaut 20
	Div	Division entière	$(7)\text{Div}(2)$ vaut 3
	Mod	Reste de la division entière	$(7)\text{Mod}(2)$ vaut 1
Fonctions Prédéfinies	Sqr	Élévation au carré	Sqr (4) vaut 16
	Abs	Valeur absolue	Abs (-10) vaut 10
	Succ	Entier suivant	Succ (5) vaut 6
	Pred	Entier précédent	Pred (5) vaut 4



5. Types de Données Simples Standards (suite)

B. Type Réel (Real) :

Le type Réel spécifié par l'identificateur standard **REAL** correspond à l'ensemble des nombres réels.

Représentation des valeurs

Les réels peuvent s'écrire sous forme de notation :

– Décimale (virgule fixe):

Nombre **REAL** avec partie entière et fractionnaire

Exemple : 12.43 -0.45 +1.0

– Exponentielle (virgule flottante):

Nombre **REAL** avec partie entière, fractionnaire et un exposant

Exemple : 20E+2 0.45e-4 -1.5E1



5. Types de Données Simples Standards (suite)

B. Type Réel (Real) :

- Le point fixe d'un nombre réel doit être toujours précédé et suivi d'une valeur. Ainsi 5. est illégal mais 5.0 est correct.
- Comme dans le cas des nombres entiers, l'ensemble des valeurs des réels pris par une variable de type **REAL** est limité et ces limites sont liées à la longueur des mots mémoire utilisés pour représenter ces nombres.



5. Types de Données Simples Standards (suite)

B. Type Réel (Real) :

Turbo-Pascal permet l'utilisation de quatre types de données prédéfinis pour le domaine des nombres réels conformément au tableau suivant :

Type	Intervalle (en valeur absolue)	Longueur
Real	2.9 E-39 à 1.7 E+38	6 octets
Single	1.5 E-45 à 3.4 E+38	4 octets
Double	5.0 E-324 à 1.7 E+308	8 octets
Extended	3.4 E-4932 à 1.1 E+4932	10 octets

Les types `Single`, `Double` et `Extended` ne peuvent pas être utilisés que si l'ordinateur est équipé d'un coprocesseur mathématique du type 8087, 80287, 80387, etc., ou si on dispose d'un émulateur.



5. Types de Données Simples Standards (suite)

B. Type Réel (Real) :

Opérateurs et Fonctions relatifs au type réel :

Type	Notation	Signification	Exemples
Opérateurs de Comparaison	$<$, $>$, $<>$, $=$, $<=$, $>=$		
Opérateurs Arithmétiques	$+$	Addition	$x+y$
	$-$	Soustraction	$x-y$
	$*$	Multiplication	$x*y$
	$/$	Division	x/y
Fonctions Prédéfinies	Sqrt	Racine carré	Sqrt (x)
	Sqr	Élévation au carré	Sqr (x)
	Abs	Valeur absolue	Abs (x)
	Frac	Partie fractionnaire	Frac (1.35) vaut 0.35
	Int	Partie entière	Int (1.35) vaut 1



5. Types de Données Simples Standards (suite)

B. Type Réel (Real) :

Opérateurs et Fonctions relatifs au type réel :

Type	Notation	Signification	Exemples
Fonctions Prédéfinies	Round	Arrondi à l'entier le plus proche	Round (2.35) vaut 2 Round (2.85) vaut 3 Round (2.50) vaut 3
	Trunc	Éliminer la partie fractionnaire	Trunc (2.35) vaut 2 Trunc (2.85) vaut 2
	Sin	Sinus (en radiant)	Sin (x)
	Cos	Cosinus (en radiant)	Cos (x)
	Arctan	Arctg (en radiant)	Arctan (x)
	Ln	Logarithme népérien	Ln (x)
	Exp	Exponentiel	Exp (x)

Remarque : les fonctions **sin**, **cos**, **arctan**, **ln** et **exp** peuvent être utilisées avec des variables de type entier.



5. Types de Données Simples Standards (suite)

C. Type Caractère (Char) :

Le type **CHAR** correspond à l'ensemble des valeurs qui définit les caractères.

Représentation des valeurs

L'ensemble du type **Char** est formé de plusieurs sous-ensembles :

- Les caractères alphabétiques minuscules ou majuscules :
a , b , c , ... , z , A , B , C , ... , Z
- Les caractères numériques : **0 , 1 , 2 , ... , 9**
- Les caractères spéciaux : **+ , - , * , / , = , ? , (,) , [,] , : , ; , \$, ...**
- L'espace (appelé le blanc)



5. Types de Données Simples Standards (suite)

C. Type Caractère (Char) :

- Turbo-Pascal utilise le code **ASCII** pour représenter les caractères. Les lettres **ASCII** représentent l'abréviation de *American Standard Code for Information Interchange*.
- En conformité avec ce code, un caractère est codé sur un octet, ce qui permet de définir 255 différents caractères.
- Une constante de type **CHAR** s'écrit par **un caractère** encadré d'une paire d'apostrophes

Exemple : `'A'` , `'m'` , `' '` , ... etc

Rem : si le caractère apostrophe doit être écrit sous forme d'une constante, alors il faut le doubler : `''` {constante caractère apostrophe}.



5. Types de Données Simples Standards (suite)

C. Type Caractère (Char) :

Fonctions relatifs au type caractère :

Type	Notation	Signification	Exemples
Fonctions de Conversion	Ord	Le résultat de cette fonction est une valeur entière qui représente le code ASCII d'un caractère	Ord ('X') vaut 88 Ord ('A') vaut 65
	Chr	Le résultat de cette fonction retourne le caractère qui correspond au code ASCII d'un entier	Chr (88) vaut X Chr (65) vaut A
Fonctions de Succession	Succ	Caractère suivant	Succ ('B') vaut C
	Pred	Caractère précédent	Pred ('C') vaut B



5. Types de Données Simples Standards (suite)

D. Type Booléen (Boolean) :

Le **type booléen**, dit type logique, est un type symbolique qui ne peut recevoir que deux valeurs logiques : **TRUE** (Vrai) et **FALSE** (Faux).

Représentation des valeurs

- ✓ Si, dans un programme, une variable doit recevoir des valeurs logiques, elle doit être déclarée de type **BOOLEAN**. Alors, les valeurs possibles de cette variable sont **TRUE** et **FALSE**.
- ✓ La déclaration d'une *constante de type* **BOOLEAN** se fait en affectant la valeur constante **TRUE** ou **FALSE** à la constante.



5. Types de Données Simples Standards (suite)

D. Type Booléen (Boolean) :

- Des valeurs logiques ne peuvent pas être entrées au clavier ; elles doivent être affectées à une variable de type booléen pendant le déroulement du programme donc, de façon dynamique.
- Le contenu des variables ou constantes de type **BOOLEAN** peut être affiché, comme pour les autres variables, avec l'instruction **Write** ou **Writeln**.

Exemple :

```
OK:= True;  
WRITE(OK); {affiche TRUE}  
FIN:= False;  
WRITE(FIN); {affiche FALSE}
```



5. Types de Données Simples Standards (suite)

D. Type Booléen (Boolean) :

Opérateurs relatifs au type booléen :

Type	Notation	Signification	Exemples
Opérateurs de Comparaison	=	Égale	
	<	Inférieur	
	>	Supérieur	
	<=	Inférieur ou égale	
	>=	Supérieur ou égale	
	<>	Différent	
Opérateurs Logique	And	Et (Conjonction)	(A < B)And(C < D)
	Or	Ou (Disjonction)	(A < B)Or(C < D)
	Not	Non (Négation)	Not(A < B)

La règle de comparaison suivante est utilisée pour les valeurs logiques:

FALSE < TRUE



6. L'Affectation

Définition :

- ✓ *L'affectation permet d'attribuer une valeur ou une expression à une variable de même type*
- ✓ *L'instruction d'affectation se note par le symbole " := "*

Exemple: on suppose que **x** est une variable entière

Avant	Affectation	Après
x <input data-bbox="678 932 786 1023" type="text" value="?"/>	x := 10 ;	x <input data-bbox="1357 932 1464 1023" type="text" value="10"/>
x <input data-bbox="678 1086 786 1177" type="text" value="10"/>	x := x + 5 ;	x <input data-bbox="1357 1086 1464 1177" type="text" value="15"/>

Si **x** est de type réel et **y** est de type entier, alors:

x := y ;

Possible

y := x ;

Impossible



7. Les Commentaires

- Comme tout langage évolué, Pascal permet la présence de commentaires dans un **programme source**.
- Les commentaires sont des textes explicatifs destinés aux lecteurs du programme et qui ne seront pas lus par la machine.
- Les commentaires sont ignorés par le compilateur et n'influencent pas l'exécution du programme ; ils sont utilisés seulement pour **documenter le programme**.
- Pour introduire un commentaire dans le programme source, il y a deux possibilités :
 - Utilisation des accolades { }
 - Utilisation des parenthèses (* *)

Exemple :

```
{Ceci est commentaire}
```

```
(* ceci est un autre commentaire *)
```



7. Les Commentaires (suite)

- Les deux symboles sont équivalents dans les sens qu'on peut utiliser un ou l'autre pour écrire de commentaires mais avec une restriction :
un commentaire ouvert par { doit être absolument fermé par }
un commentaire ouvert par (* doit être absolument *fermé* par *)
- Un commentaire peut apparaître dans un programme à n'importe quel endroit où un espace ou une fin de ligne sont permis. Par contre un commentaire ne pourra pas apparaître dans un identificateur ou dans une constante.

Exemple :

```
Program addition;  
(* Programme permettant l'addition de 2 nombre réels *)  
Uses wincrt;  
Var A,B,Somme:Real; { Partie déclaration }  
Begin
```



8. Entées – Sorties

Définition :

Pour faire fonctionner un programme il faut lui fournir des données et prévoir la possibilité de récupérer les résultats. Ces deux opérations portent le nom générique d'opérations d'entrée/sortie. Donc il s'agit de deux opérations distinctes :

- ✓ *Entrée de données (Lecture des données)*
- ✓ *Sortie de résultats (Écriture des résultats)*

- En Pascal, les opérations d'entrée/sortie sont réalisées par deux procédures système :

READ – procédure standard de lecture

WRITE – procédure standard d'écriture



8. Entrées – Sorties (suite)

A. Instructions de Lecture (Read, Readln) :

- Les instructions de lecture permettent à l'utilisateur de saisir des valeurs au clavier (ou à partir d'un fichier) pour qu'elles soient utilisées par le programme.
- Dès que le programme rencontre une instruction **READ** ou **READLN** l'exécution s'interrompt attendant la saisi d'une valeur.

Syntaxe des instructions de lecture

```
Read(Liste_de_variables);
```

```
Readln(Liste_de_variables);
```

Lire la valeur de
`liste_de_variables` et revenir
à la ligne



8. Entées – Sorties (suite)

A. Instructions de Lecture (Read, Readln) :

Exemple :

Supposons qu'on veut saisir 4 variables entières dont les valeurs sont :

A=5 B=10 C=2 D=6

```
Read(A, B) ;
```

```
Readln(C) ;
```

```
Read(D) ;
```

Exécution

5 10 2

6



8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

- Les instructions d'écriture permettent au programme de communiquer des valeurs (ou des messages) à l'utilisateur en les affichant à l'écran (ou sur un fichier).

Syntaxe des instructions d'écriture

```
Write(Liste_de_variables);
```

```
Writeln(Liste_de_variables);
```

Écrire la valeur de `liste_de_variables` et revenir à la ligne

```
Write('Ceci est un Message');
```

Écrire le message écrit entre les apostrophes



8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

Exemple :

Supposons qu'on a : `A=5` est de type réel

```
write('La valeur de A est: ');  
writeln(A);  
write('Ceci est un message affiché à l''écran');
```

Exécution

```
La valeur de A est: 5.0000000000E+00  
Ceci est un message affiché à l'écran
```




8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

❑ Instructions d'affichage par défaut :

Pour chaque information mentionnée dans une instruction d'écriture, on peut choisir entre :

- ✓ Laisser le langage Pascal imposer sa présentation : on parle alors de "**Format d'affichage par défaut**"
- ✓ Imposer notre propre format d'affichage



8. Entées – Sorties (suite)

B. Instructions d'Écriture (write, writeln) :

❑ Exemple d'affichage par défaut :

```
program affichage;
uses wincrt;
var n,p:Integer;
    x,y:Real;
    C1,C2:Char;
    ok:Boolean;
Begin
  write('n= ');readln(n);
  write('p= ');readln(p);
  write('x= ');readln(x);
  write('y= ');readln(y);
  write('C1= ');readln(C1);
  write('C2= ');readln(C2);
  ok:=false;
  writeln('Nombre',n);
  writeln('Nombre',n);
  writeln(n, p);
  writeln(n,' ',p);
  writeln(x,y);
  writeln(C1,C2);
  writeln('Cela est',ok);
End.
```

Exécution

```
n= 3
p= 125
x= -34.5e6
y= 2
C1= A
C2= i
Nombre3
Nombre 3
3 125
-3.4500000000E+07 2.0000000000E+00
Ai
Cela estFALSE
```



8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

❑ Règles générales de l'affichage par défaut :

Le tableau suivant présente les cases utilisés pour l'affichage par défaut de chaque type de données:

Nature de l'Expression	Cases utilisés pour son affichage
Entière (Integer)	Sa propre longueur
Réelle (Real)	17
Caractère (Char)	1
Booléenne (Boolean)	4 (pour True) ou 5 (pour False)



8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

❑ Imposition d'un format d'affichage :

- Tous les types d'expressions figurant dans une instruction `write` (ou `writeln`), peuvent se voir imposer un gabarit (nombre de cases) par une indication de la forme (`:g`) où `g` représente une expression entière quelconque.
- De plus, pour les expressions de type réel, on peut imposer un gabarit de la forme (`:g:d`) qui correspond à la forme point fixe avec `d` décimales.



8. Entées – Sorties (suite)

B. Instructions d'Écriture (`write`, `writeln`) :

- ❑ Imposition d'un format d'affichage :

Exemple :

Supposons qu'on a : `A=25.35` est de type réel

```
writeln('La valeur de A est: ',A);  
writeln('La valeur de A est: ',A:10);  
writeln('La valeur de A est: ',A:9:3);
```

Exécution 

```
La valeur de A est:  2.5350000000E+01  
La valeur de A est:  2.535E+01  
La valeur de A est:  25.350
```



Cours N°3

Structures de Contrôles

Cours élaboré par Mrs BEADAHMANE, BOUFATAH & BRAHMI



1. Introduction

Définition :

Une structure de contrôle sert à contrôler le déroulement d'un traitement.

■ Un traitement peut s'exécuter de différentes manières:

✚ Séquentielle تسلسليا

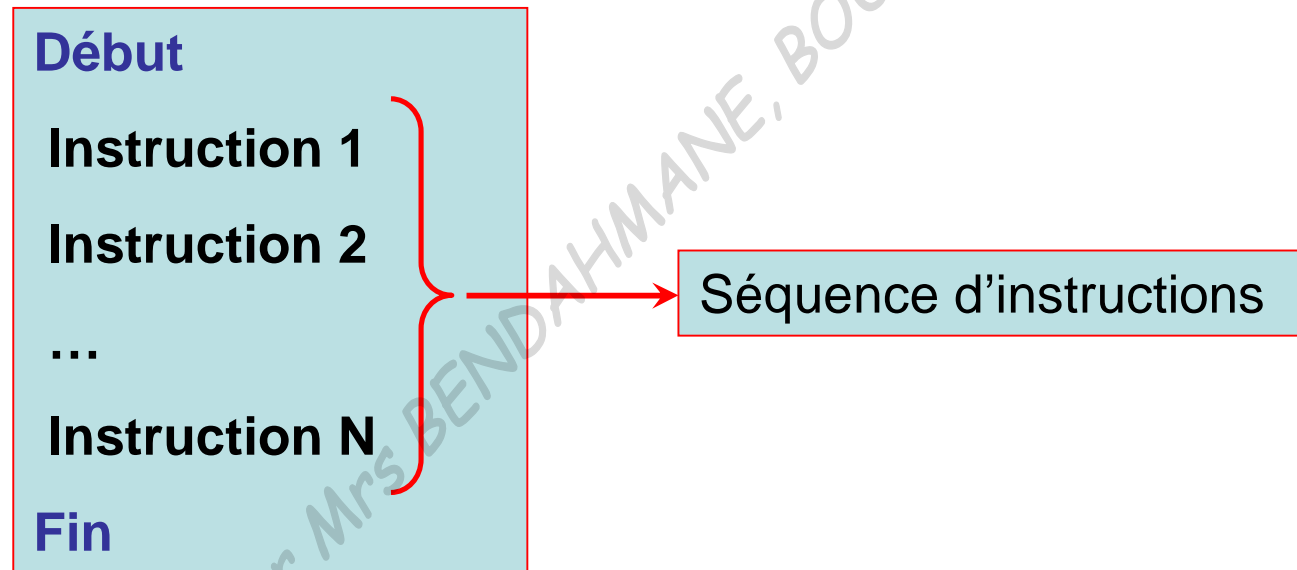
✚ Alternative (condition) تناوبيا

✚ Répétitive (boucle) تكريريا



2. Le traitement séquentiel

Le traitement séquentiel est une suite d'instruction qui s'exécutent l'une à la suite de l'autre.



Exemple: Addition de deux nombres A et B



2. Le traitement séquentiel (suite)

```
Program Addition;  
Uses wincrt;  
Var A,B,Somme:Real;  
  
Begin  
    Read(A);  
    Read(B);  
    Somme:=A+B;  
    Write(Somme);  
End.
```

Suite d'instructions
s'exécutant
séquentiellement



2. Le traitement séquentiel (suite)

Dans un programme, les instructions sont exécutées dans l'ordre de leur apparition, donc de façon séquentielle. Mais, **“l'intelligence”** d'un programme informatique provient essentiellement:

- De la possibilité de faire des choix entre plusieurs possibilités de traitement en fonction de différents critères (condition);
- De la possibilité d'exécuter rapidement une série d'instructions de façon répétitive (boucle).



3. Le traitement alternatif (structures de choix)

- L'instruction de choix permet la sélection entre deux possibilités (appelée sélection binaire).
- La condition s'exprime sous la forme:
 - d'une expression logique booléenne simple (condition simple).
 - ou combinée, plusieurs conditions composées avec des opérations logiques ET, OU et NON.



3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if ... then)

La structure conditionnelle simple se présente sous la forme suivante:

SI (condition) alors
séquence
FSI {lire Fin SI}

Principe :

Si la condition est vérifiée (vraie) alors la séquence d'instructions s'exécute. Dans le cas contraire, ne rien faire.



3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if ... then)

Traduction en Pascal:

- Instruction simple:

```
IF <condition> then <séquence>;
```

- Bloc d'instructions:

```
IF <condition> then  
    begin  
        <séquence>  
    end;
```



3. Le traitement alternatif (suite)

A. Structure conditionnelle simple (if ... then)

Exemple: Comparaison de deux variables

Algorithme comparaison

Variables utilisées:

A, B : nombres réels

1) Début

2) Lire (A)

3) Lire (B)

4) SI $A > B$ ALORS Ecrire ('A est supérieure à B) FSI

5) Fin



3. Le traitement alternatif (suite)

Programme: Première écriture

```
Program comparaison ;  
Uses wincrt ;  
Var A,B : REAL ;  
  
  BEGIN  
  
    READLN(A) ;  
  
    READLN(B) ;  
  
    IF A > B THEN WRITELN (A,' est superieure a ',B) ;  
  
  END.
```



3. Le traitement alternatif (suite)

Programme: Deuxième écriture

```
Program comparaison ;  
  
Uses wincrt ;  
  
Var A,B : REAL ;  
  
BEGIN  
  
    READLN(A) ;  
  
    READLN(B) ;  
  
    IF A > B THEN  
  
        begin  
  
            WRITELN (A, ' est superieure a ',B) ;  
  
        end ;  
  
END.
```




3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if ... then ... else)

La structure conditionnelle composée se présente sous la forme suivante:

SI (condition) ALORS

séquence 1

SINON

séquence 2

FSI

Principe :

Si la condition est vérifiée alors la séquence1 s'exécute.
Dans le cas contraire, c'est la séquence2 qui va s'exécuter.



3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if ... then ... else)

Traduction en Pascal:

```
■ IF <condition> THEN <séquence1>
```

```
  ELSE <séquence2>;
```

```
■ IF <condition> THEN
```

```
  begin
```

```
    <séquence1>
```

```
  end
```

```
ELSE
```

```
  begin
```

```
    <séquence2>
```

```
end;
```



3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if ... then ... else)

Attention ! On ne pose **JAMAIS** un point virgule (;) avant un **ELSE**.



3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if ... then ... else)

Exemple: Comparaison de deux variables (amélioré)

Algorithme comparaison

Variables utilisées:

A, B : nombres réels

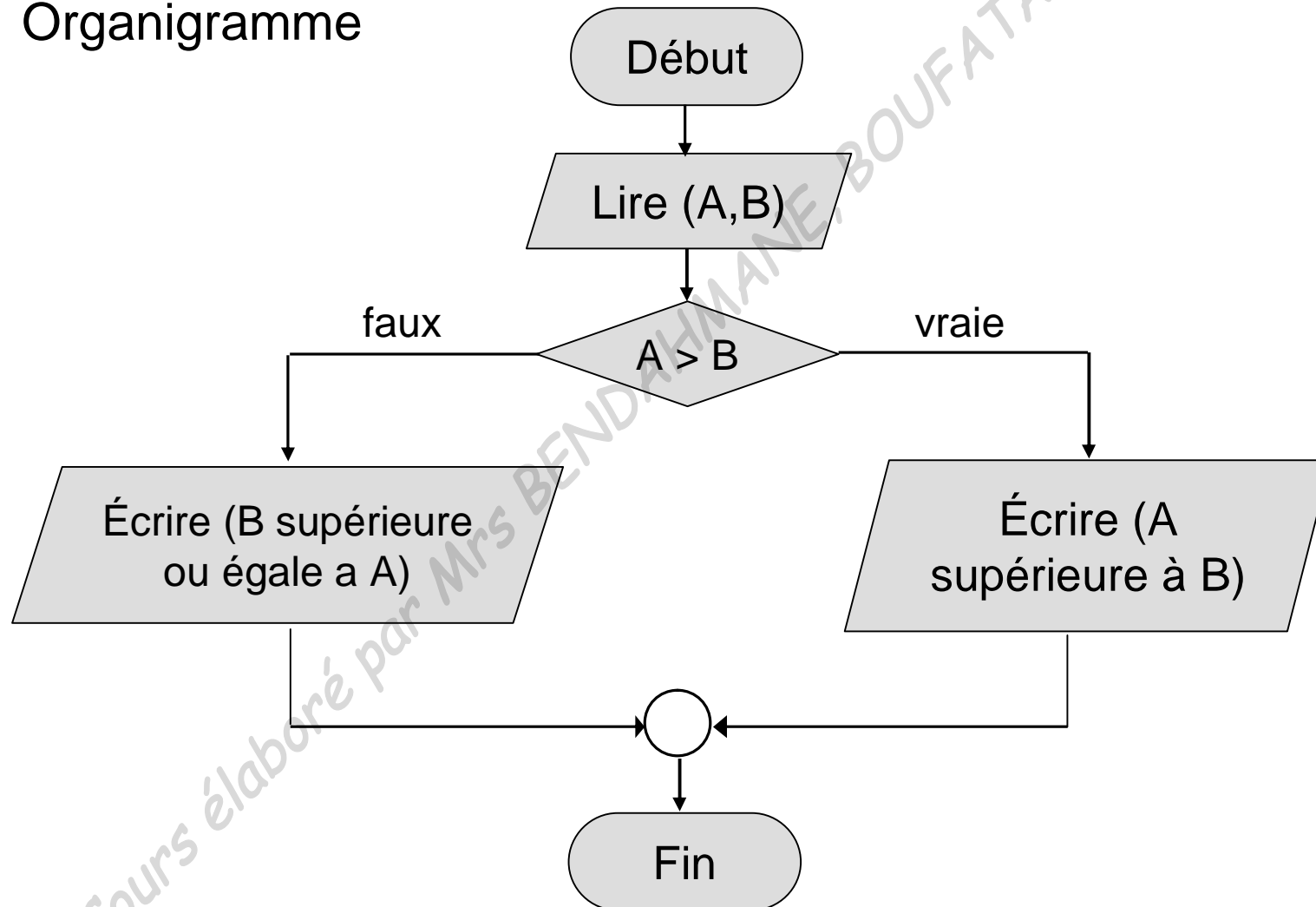
- 1) Début
- 2) Lire (A)
- 3) Lire (B)
- 4) SI $A > B$ ALORS Écrire ('A est supérieure à B')
- 5) SINON Écrire ('B est supérieure ou égale à A') FSI
- 6) Fin



3. Le traitement alternatif (suite)

B. Structure conditionnelle composée (if ... then ... else)

Organigramme





3. Le traitement alternatif (suite)

Programme: Première écriture

```
Program comparaison ;  
Uses wincrt ;  
Var A,B : REAL ;  
  
BEGIN  
  
    READLN(A) ;  
  
    READLN(B) ;  
  
    IF A > B THEN WRITELN (A,' est supérieure a ',B)  
    ELSE WRITELN (B,' est supérieure ou égale a ',A) ;  
  
END.
```



Programme: Deuxième écriture

```
Program comparaison ;
Uses wincrt;
Var A,B : REAL;
  BEGIN
    READLN(A);
    READLN(B);
    IF A > B THEN
      begin
        WRITELN (A,' est supérieure a ',B);
      end
    else
      begin
        WRITELN (B,' est supérieure a ',A);
      end;
    END.
```



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

Définition :

L'instruction CASE utilise la valeur d'une expression de type ordinal pour opérer le choix d'une instruction parmi d'autres (choix multiple) en vue de son exécution.

- Cette instruction compare la valeur d'une variable de type entier ou caractère à toute une série de valeur.
- Suivant la valeur effective de la variable différentes séquences d'instructions sont exécutées.
- Une séquence par défaut peut être prévue dans le cas où la variable n'est égale à aucune des valeurs énumérées.



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

AU CAS OU (**variable ou expression**)

choix 1 : <séquence 1>;

choix 2 : <séquence 2>;

...

choix n : <séquence n>;

SINON <séquence par défaut>;

FCAS

Remarque :

Cette structure de contrôle présente plusieurs cas d'exécutions du traitement mais un seul cas sera exécuté.



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

Traduction en Pascal:

■ Première écriture:

CASE <expression> **OF**

Choix 1 : <séquence 1>;

Choix 2 : <séquence 2>;

...

Choix n : <séquence n>;

END;



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

Traduction en Pascal:

■ Deuxième écriture:

CASE <expression> **OF**

Choix 1 : <séquence 1>;

Choix 2 : <séquence 2>;

...

Choix n : <séquence n>;

ELSE <séquence par défaut>;

END;



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

Exemple: Calcul du produit, somme et la moyenne de trois réelles suivant un choix

Algorithme menu

Variables utilisées:

NB1, NB2, NB3 : nombres réels

Choix : caractère

1) Début

2) Lire (NB1, NB2, NB3)

3) Lire (Choix)

4) AU CAS OU (Choix)

‘1’ : écrire (‘Le produit est : ’, $nb1*nb2*nb3$);

‘2’ : écrire (‘La somme est : ’, $nb1+nb2+nb3$);

‘3’ : écrire (‘la moyenne est : ’, $(nb1+nb2+nb3)/3$);

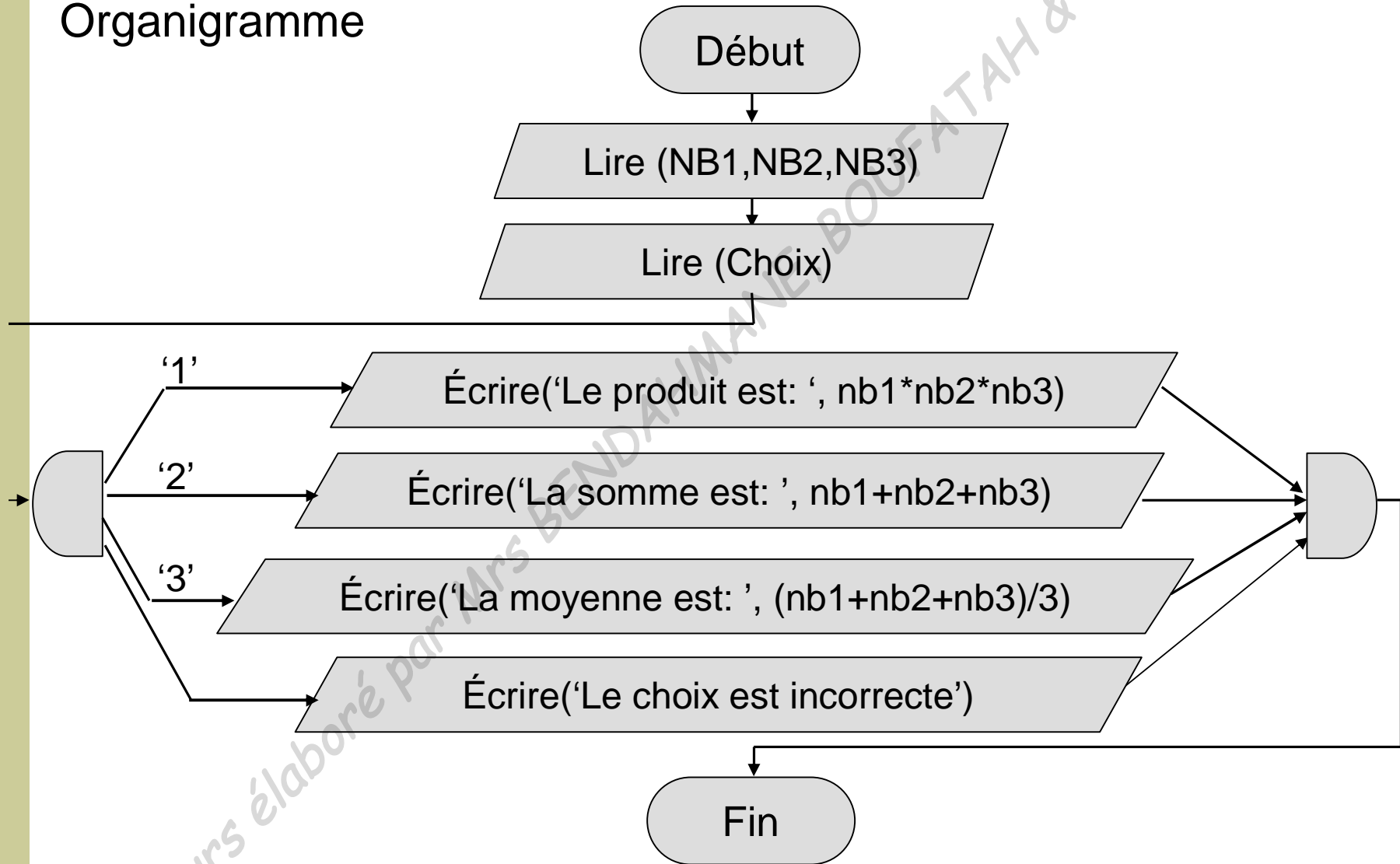
5) SINON écrire (‘Choix incorrecte) FCAS

6) Fin



3. Le traitement alternatif (suite)

Organigramme





3. Le traitement alternatif (suite)

Programme

```
Program menu ;
Uses wincrt;
Var NB1,NB2,NB3 : REAL;
    Choix : CHAR;
BEGIN
    WRITELN('Entrer trois nombres :');
    READLN(NB1,NB2,NB3);
    WRITELN('1 pour la multiplication. ');
    WRITELN('2 pour la somme. ');
    WRITELN('3 pour la moyenne. ');
    READLN(Choix);
    CASE Choix of
        '1': WRITELN ('Le produit est : ',NB1*NB2*NB3);
        '2': WRITELN ('La somme est : ',NB1+NB2+NB3);
        '3': WRITELN ('La moyenne est : ',(NB1+NB2+NB3)/3);
        ELSE WRITELN ('Le choix est incorrecte');
    END;
END;
END.
```



3. Le traitement alternatif (suite)

C. Structure conditionnelle composée (case ... of)

Remarque:

- Les valeurs d'une instruction CASE sont des constantes mais elles peuvent définir un domaine (intervalle). Exemple:

CASE L OF

5 .. 20 : <instruction 1 >;

30 .. 40 : <instruction 2>;

END;



4. Le traitement répétitif (les boucles)

Dans un programme, il arrive souvent qu'une ou plusieurs instructions doivent être exécuter plusieurs fois dans une structure répétitive appelée boucle.

Dans une boucle, le nombre de répétitions:

- Peut être connu, fixé à l'avance.
- Il peut dépendre d'une condition permettant l'arrêt et la sortie de cette boucle.

On dispose de trois structures pour contrôler un traitement répétitif:

- La boucle "*Tant que ... faire*".
- La boucle "*Pour ... faire*".
- La boucle "*Répéter ... faire*".



4. Le traitement répétitif (suite)

A. La boucle "Tant que ... faire" (while ... do)

La boucle "*Tant que ... faire*" se présente sous la forme suivante:

Tant que (condition) faire
séquence
Fin tant que

Principe :

Tant que la condition est vérifiée la séquence s'exécute. Elle ne s'arrêtera que lorsque la condition est invérifiable.



4. Le traitement répétitif (suite)

A. La boucle "Tant que ... faire" (while ... do)

Traduction en Pascal:

■ Première écriture:

```
WHILE <condition> DO <séquence>;
```

■ Deuxième écriture

```
WHILE <condition> DO
```

```
begin
```

```
    <séquence>;
```

```
end;
```



4. Le traitement répétitif (suite)

A. La boucle "Tant que ... faire" (while ... do)

Exemple: Afficher les 100 premiers nombre entiers positifs

Algorithme premier_N_E

Variables utilisées:

I : nombre entier

1) Début

2) I=0

3) Tant que I <= 100 faire

4) Écrire(I)

5) I=I+1 Fin tant que

6) Fin



4. Le traitement répétitif (suite)

A. La boucle "Tant que ... faire" (while ... do)

Programme

```
Program premier_N_E;  
Uses wincrt;  
Var   I:Integer;  
  
BEGIN  
    I:=0; {Initialisation}  
    WHILE I<=100 DO  
        begin  
            WRITELN(I);  
            I:=I+1; {Incrémentation}  
        end;  
    END.
```



4. Le traitement répétitif (suite)

A. La boucle "Tant que ... faire" (while ... do)

Attention ! Il faut bien choisir le test d'arrêt de la boucle sinon on obtient une *boucle infinie*.

Exemple: calcul du carré d'un nombre réel.

```
r := 1.5;           {initialisation du nombre réel}
WHILE r > 0 DO   {condition toujours vraie ! étant donné l'initialisation}
r := SQR(r);       {calcul du carré avec la fonction prédéfini SQR}
```

C'est une **boucle infinie** parce que le carré d'un nombre renvoie toujours une valeur positive. La condition $r > 0$ est toujours vraie.



4. Le traitement répétitif (suite)

B. La boucle "Pour ... faire" (For ... To ... do)

La boucle "*Pour ... faire*" se présente sous la forme suivante:

```
Pour compteur = <valeur initiale> à <valeur finale> faire  
  
    <séquence>  
  
Fin Pour
```

Principe :

Pour la variable de contrôle allant de la valeur initiale jusqu'à la valeur finale, exécuter la séquence d'instruction.



4. Le traitement répétitif (suite)

B. La boucle "Pour ... faire" (For ... To ... do)

Traduction en Pascal:

■ Première écriture:

```
FOR <compteur>:= <valeur initiale> TO <valeur finale> DO  
    <séquence>;
```

■ Deuxième écriture

```
FOR <compteur>:= <valeur initiale> TO <valeur finale> DO  
    begin  
        <séquence>;  
    end;
```



4. Le traitement répétitif (suite)

B. La boucle "Pour ... faire" (For ... To ... do)

Exemple: Afficher les 100 premiers nombre entiers positifs

Algorithme premier_N_E

Variables utilisées:

I : nombre entier

1) Début

2) Pour I = 0 à 100 faire

3) Écrire(I)

4) Fin Pour

5) Fin



4. Le traitement répétitif (suite)

B. La boucle "Pour ... faire" (For ... To ... do)

Programme

```
Program premier_N_E;  
Uses winCRT;  
Var I : Integer;  
  
BEGIN  
    FOR I:=0 TO 100 DO  
        begin  
            WRITELN(I);  
        end;  
    END.
```



4. Le traitement répétitif (suite)

C. La boucle "Répéter ... jusqu'à" (Repeat ... until)

La boucle "*Répéter ... jusqu'à*" se présente sous la forme suivante:

Répéter

<séquence>

Jusqu'à <condition vérifiée>

Principe :

La séquence d'instructions délimitée par les mots **Répéter** et **Jusqu'à** est exécutée jusqu'à ce que la condition évaluée à la fin de la boucle soit vraie.



4. Le traitement répétitif (suite)

C. La boucle "Répéter ... jusqu'à" (Repeat ... until)

Traduction en Pascal:

REPEAT

<séquence>;

UNTIL <condition>;

Cours élaboré par Mrs BENDAHMANE, BOUFATAH & BRAHMI



4. Le traitement répétitif (suite)

C. La boucle "Répéter ... jusqu'à" (Repeat ... until)

Exemple: Afficher les 100 premiers nombre entiers positifs

Algorithme premier_N_E

Variables utilisées:

I : nombre entier

1) Début

2) $I = 0$

3) Répéter

4) Écrire(I)

5) $I = I + 1$

6) Jusqu'à $I > 100$

7) Fin



4. Le traitement répétitif (suite)

C. La boucle "Répéter ... jusqu'à" (Repeat ... until)

Programme

```
Program premier_N_E ;  
Uses wincrt ;  
Var I : Integer ;  
  
BEGIN  
    I:=0 ;  
    repeat  
        WRITELN(I) ;  
        I:=I+1 ;  
    until I > 100 ;  
END.
```



4. Le traitement répétitif (suite)

D. Les boucles imbriquées

Les boucles peuvent être imbriquées les unes dans les autres. Une boucle "*tant que ... faire*" peut contenir une autre boucle "*tant que ... faire*", ou une autre boucle "*répéter ... jusqu'à*", ou une autre boucle "*pour ... faire*".

Autrement dit, une boucle peut contenir une autre boucle qui elle-même peut contenir une autre boucle ainsi de suite.



5. Remarques générales sur les boucles

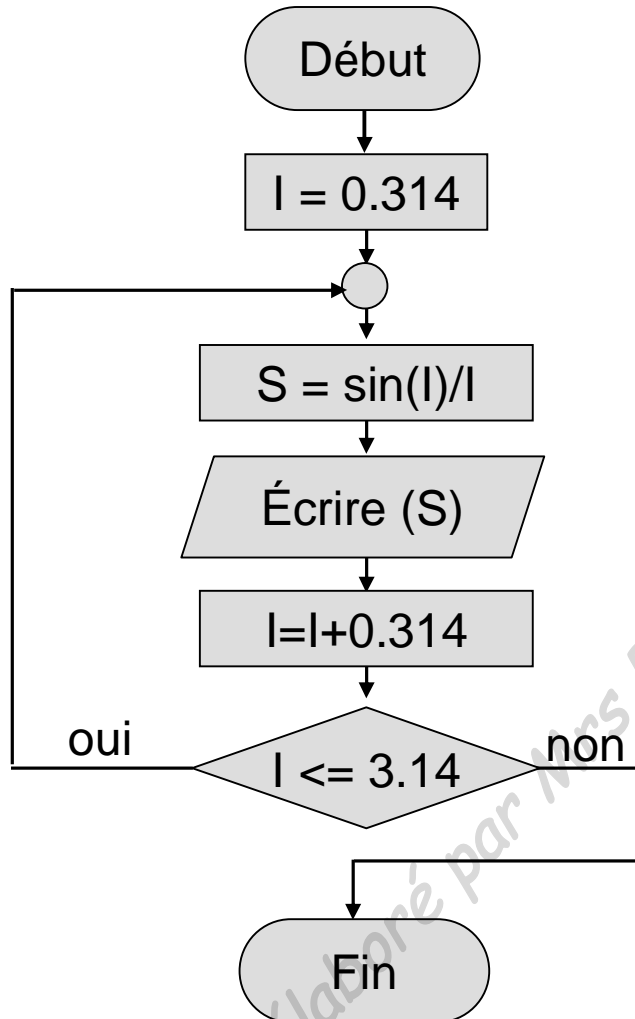
- 1) La boucle "*while*" peut ne pas s'exécuter du tout si la condition d'accès à la boucle n'est pas vérifiée.
- 2) La boucle "*repeat*" s'exécute au moins une fois.
- 3) La boucle "*for*" s'exécute toujours, le nombre d'exécution de la séquence (itérations) est connu d'avance.
- 4) Les boucles "*while*" et "*repeat*" nécessitent toujours une initialisation et une incrémentation du compteur de la boucle.
- 5) La boucle "*for*" est utilisée exclusivement dans le cas où le nombre d'itérations est connu.
- 6) Si le nombre d'instructions de la boucle est supérieur à 1 celles-ci doivent être mises entre un **BEGIN** et un **END**.



6. L'instruction GOTO et la déclaration d'étiquette

En théorie, les structures de choix et de répétition proposées par le Pascal sont suffisantes pour traduire toutes les situations possibles. Mais, le Pascal dispose également d'une instruction de branchement inconditionnel GOTO.

Exemple:



```
Program sinus;  
Uses wincrt;  
Var i,s : real;  
Label 10;  
BEGIN  
    I:=0.314;  
10 : S:=sin(I)/I;  
    WRITELN(S);  
    I:=I+0.314;  
    IF I <= 3.14 then GOTO 10;  
END.
```



Cours N°4

Les Matrices

Cours élaboré par Mrs BENDAHMANE, BOUFATAH & BRAHMI



1. Introduction

Définition :

Une matrice $n \times m$ est un tableau de nombres à n lignes et m colonnes. n et m sont les dimensions de la matrice.

Exemple avec $n = 2$ et $m=3$

$$A = \begin{bmatrix} 1 & -5 & 3 \\ 2 & 1 & 4 \end{bmatrix}$$



1. Introduction (suite)

- Dans une matrice \mathbf{A} , on note \mathbf{A}_{ij} l'élément situé à l'intersection de la ligne i et de la colonne j (la ligne est toujours nommée en premier).

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1m} \\ A_{21} & A_{22} & \cdots & A_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ A_{n1} & A_{n2} & \cdots & A_{nm} \end{bmatrix}$$

- On note $[\mathbf{A}_{ij}]$ la matrice d'élément général \mathbf{A}_{ij} . On a donc: $\mathbf{A} = [\mathbf{A}_{ij}]$



1. Introduction (suite)

- Si $m = 1$ ou $n = 1$, la matrice est appelée **vecteur**. Plus précisément:
 - Si $m = 1$, la matrice est un **vecteur-colonne**.
 - Si $n = 1$, la matrice est un **vecteur-ligne**.
- Si $m = n$, la matrice est appelée **matrice carré**.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$

$$x = [x_1 \quad x_2 \quad \dots \quad x_n]$$



2. Quelques matrices carrées particulières

A. Matrice diagonale

$$D = \begin{bmatrix} D_{11} & 0 & 0 & 0 \\ 0 & D_{22} & 0 & 0 \\ 0 & 0 & D_{33} & 0 \\ 0 & 0 & 0 & D_{44} \end{bmatrix}$$



2. Quelques matrices carrées particulières(suite)

B. Matrice unité

$$I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



2. Quelques matrices carrées particulières(suite)

C. Matrice triangulaire supérieure

$$U = \begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix}$$



2. Quelques matrices carrées particulières(suite)

D. Matrice triangulaire inférieure

$$L = \begin{bmatrix} L_{11} & 0 & 0 & 0 \\ L_{21} & L_{22} & 0 & 0 \\ L_{31} & L_{32} & L_{33} & 0 \\ L_{41} & L_{42} & L_{43} & L_{44} \end{bmatrix}$$



2. Quelques matrices carrées particulières(suite)

E. Matrice Symétrique

Une matrice carrée A est dite symétrique si: $A_{ji} = A_{ij}$.

$$A = \begin{bmatrix} 6 & 1 & 2 & 3 \\ 1 & 5 & -2 & 3 \\ 2 & -2 & 5 & -4 \\ 3 & 3 & -4 & 6 \end{bmatrix}$$



3. Opérations sur les matrices

A. Addition, soustraction

L'addition et la soustraction des matrices se font terme à terme. Les matrices doivent avoir les même dimensions:

$$\begin{bmatrix} 4 & 6 & 2 \\ 0 & 1 & 3 \end{bmatrix} + \begin{bmatrix} -1 & 3 & 6 \\ 2 & -5 & 12 \end{bmatrix} = \begin{bmatrix} 3 & 9 & 8 \\ 2 & -4 & 15 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 6 & 2 \\ 0 & 1 & 3 \end{bmatrix} - \begin{bmatrix} -1 & 3 & 6 \\ 2 & -5 & 12 \end{bmatrix} = \begin{bmatrix} 5 & 3 & -4 \\ -2 & 6 & -9 \end{bmatrix}$$



3. Opérations sur les matrices (suite)

B. Multiplication par un nombre

Chaque terme de la matrice est multiplié par un nombre:

$$3 \times \begin{bmatrix} 4 & 6 & 2 \\ 0 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 12 & 18 & 6 \\ 0 & 3 & 9 \end{bmatrix}$$



3. Opérations sur les matrices (suite)

C. Transposition

La transposée A^T d'une matrice A est la matrice obtenue en échangeant les lignes et les colonnes de A :

$$A = \begin{bmatrix} 4 & 6 & 2 \\ 0 & 1 & 3 \end{bmatrix} \Leftrightarrow A^T = \begin{bmatrix} 4 & 0 \\ 6 & 1 \\ 2 & 3 \end{bmatrix}$$

La transposée d'un vecteur-colonne est un vecteur-ligne.



3. Opérations sur les matrices (suite)

D. Multiplication des matrices

Le produit de la matrice $\mathbf{A}(n \times m)$ par la matrice $\mathbf{B}(m \times p)$ est la matrice $\mathbf{C}(n \times p)$ telle que l'élément C_{ij} est égal au produit scalaire de la ligne i de la matrice \mathbf{A} par la colonne j de la matrice \mathbf{B} :

$$C_{ij} = \sum_{k=1}^m A_{ik} \cdot B_{kj} \quad i = 1..n \quad j = 1..p$$

$$\begin{bmatrix} 4 & 6 & 2 \\ 0 & 1 & 3 \end{bmatrix} \bullet \begin{bmatrix} 2 & 0 \\ 3 & -1 \\ 4 & 3 \end{bmatrix} = \begin{bmatrix} 34 & 0 \\ 15 & 8 \end{bmatrix}$$



3. Opérations sur les matrices (suite)

E. Déterminant d'une matrice carrée 2×2

- Le déterminant *détermine* l'unicité de la solution d'un système d'équations linéaires.
- Le déterminant d'une matrice carrée 2×2 est la quantité:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \Rightarrow \det(A) = |A| = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

- La matrice A est singulière si $\det(A) = 0$, régulière dans le cas contraire.



3. Opérations sur les matrices (suite)

F. Déterminant d'une matrice carrée 3×3

Le déterminant peut se calculer de manière récursive en développant, par exemple, par rapport à la première ligne:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = a \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \begin{vmatrix} d & e \\ g & h \end{vmatrix}$$

$$= a(ei - fh) - b(di - fg) + c(dh - eg)$$

$$= aei - afh - bdi + bfg + cdh - ceg$$



Cours N°5

Les Tableaux

Cours élaboré par Mrs BENDAHMANE, BOUFATAH & BRAHMI



1. Introduction

Définition :

Un tableau est un ensemble de données qui sont toutes du même type (entier ou réel ou caractères ...) et se différencient les une des autres par leur numéro d'indice.

- Le tableau est caractérisé par un identificateur unique (nom du tableau).
- Les tableaux les plus utilisés sont:
 - à une dimension (exemple: listes, vecteurs, ...etc)
 - à deux dimensions (exemple: matrices)



2. Représentation pratique d'un tableau

On représente un tableau par un ensemble de cases repérées par leurs indices (leurs positions dans le tableau).

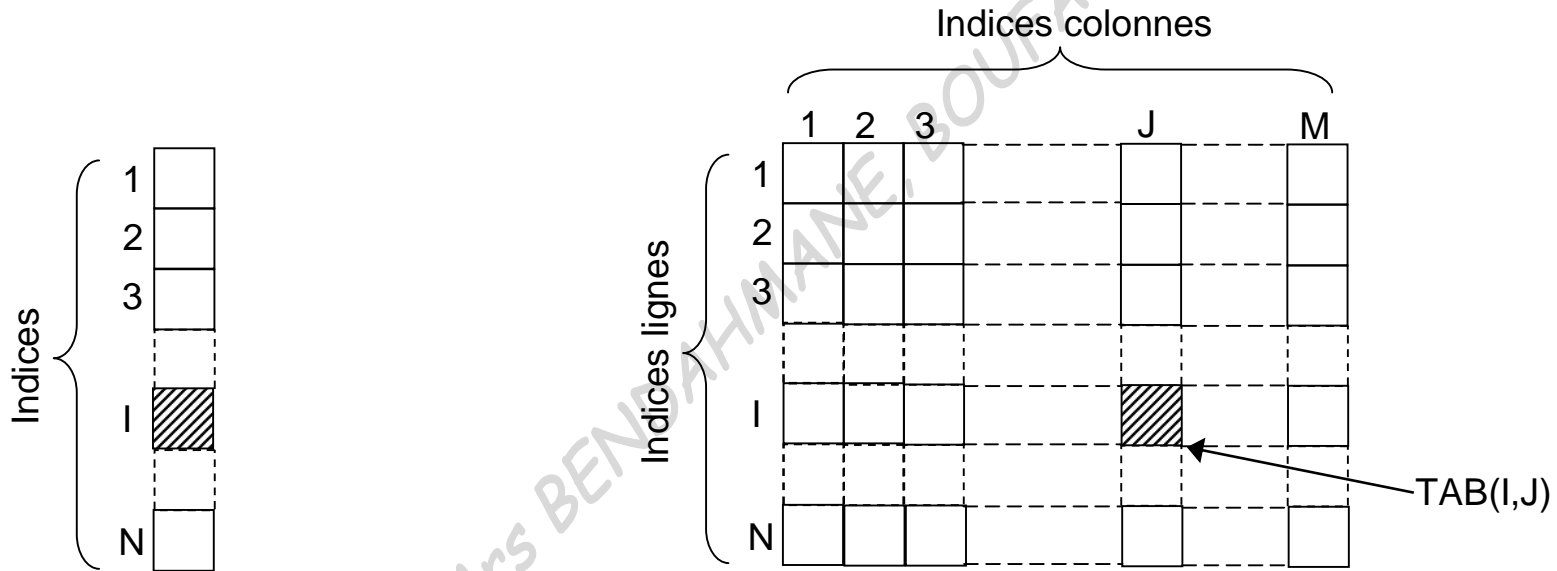


Tableau à 1 dimension

Tableau à 2 dimensions

- Dans un tableau à 1 dimension, $Liste(I)$ représente la $I^{ième}$ case du tableau unidimensionnel Liste.
- Dans un tableau à 2 dimensions, $TAB(I,J)$ représente le contenu de la case située à l'intersection de la $I^{ième}$ ligne et de la $J^{ième}$ colonne.



3. Déclaration d'un tableau

Quatre éléments fondamentaux définissent un tableau:

1. **Son nom:** qui sera un identificateur choisi en respectant les règles usuelles de dénomination des variables.
2. **Le nombre de ses dimensions** (1 dimension ou 2 dimensions ...)
3. **Sa taille:** les valeurs maximales de ses indices.
4. **Le type de données qu'il contient.**

La déclaration algorithmique d'un tableau est sous la forme:

**<Nom du tableau> (valeur maxi 1^{er} indice, valeur maxi 2^{eme} indice)
: type de variable**

Exemple: TAB est un tableau à deux dimensions de nombres réels possédant 10 lignes et 20 colonnes au maximum.

TAB(10,20) : tableau de réels



3. Déclaration d'un tableau (suite)

■ En PASCAL, les tableaux se déclarent dans la partie déclaration d'un programme, en même temps que les autres variables du programme.

■ Pour un tableau de dimension 1, on utilise la syntaxe:

<Nom_tableau> : ARRAY[1..Taille] of <Types de données>;

■ Pour un tableau de dimension 2, on utilise:

**<Nom_tableau> : ARRAY[1..nombre_lignes, 1..nombre_colonnes] of
<Types de données>;**

Exemples:

```
Var LISTE : ARRAY[1..10] of integer;
```

```
Var TAB : ARRAY[1..10,1..20] of real;
```



4. Traitements itératifs sur les tableaux

- Le principal intérêt de l'utilisation d'un tableau est de permettre d'effectuer des traitement répétitives sur l'ensembles des éléments du tableau.
- Par exemple, il suffit de décrire un traitement en invoquant le terme général **TAB(I,J)** et de le placer ce traitement dans une boucle qui fait varier les indices **I** et **J** entre les valeurs 1 et leurs valeurs maximales.
- Pour saisir ligne par ligne les **(N×M)** éléments d'un tableau de nombres, on peut utiliser un simple algorithme suivant:

```
Pour I = 1 à N faire
  Pour J = 1 à M faire
    lire(TAB(I,J))
  Fin Pour
Fin Pour
```

Tout élément d'un tableau peut être utilisé comme une simple variable: lecture, écriture, affectation, utilisation dans une expression ...etc.



4. Traitements itératifs sur les tableaux (suite)

Exemple 1 : Lecture et écriture des éléments réels d'un tableau à 1 dimension Liste(N)

Algorithme tableau;

Variables utilisées:

I,N : nombres entiers

Liste(100) : tableau de réels

- 1) Début
- 2) Lire(N)
- 3) Pour i=1 à N faire
- 4) Lire(Liste(i)) Fin Pour
- 5) Pour i=1 à N faire
- 6) Ecrire(Liste(i)) Fin Pour
- 7) Fin



Programme

```
Program Liste ;  
  
Uses wincrt;  
  
VAR N,I : INTEGER;  
    Liste : ARRAY[1..100] of real;  
  
BEGIN  
    write('Entrer le nombre de lignes N:'); readln(N);  
    writeln('Entrer ligne par ligne les coefficients du tableau  
    Liste:');  
  
    FOR I:=1 TO N DO  
        begin  
            WRITE('Liste(',I,')=');  
            READLN(Liste[I]);  
        end;  
  
    writeln('Les coefficients du tableau Liste(',N,') sont :');  
    for I:=1 to N do  
        writeln('Liste(',I,',')= ',Liste[I]);  
  
END.
```




4. Traitements itératifs sur les tableaux (suite)

Exemple 2 : Lecture et écriture des éléments réels d'un tableau à 2 dimensions TAB(N,M)

Algorithme tableau;

Variables utilisées:

I,J,N,M : nombres entiers

TAB(50,50) : tableau de réels

- 1) Début
- 2) Lire(N,M)
- 3) Pour i=1 à N faire
- 4) Pour j=1 à M faire
- 5) Lire(TAB(I,J)) Fin Pour
- 6) Pour i=1 à N faire
- 7) Pour j=1 à M faire
- 8) Ecrire(TAB(I,J)) Fin Pour
- 9) Fin



Programme

```
Program Tableau ;
Uses wincrt;

VAR N,M,I,J : INTEGER;
    TAB : ARRAY[1..50,1..50] of real;

BEGIN
write('Entrer le nombre de lignes N:'); readln(N);
write('Entrer le nombre de colonnes M:'); readln(M);
writeln('Entrer un a un et ligne par ligne les coefficients du
    tableau TAB:');

FOR I:=1 TO N DO
FOR J:=1 TO M DO
begin
WRITE('TAB(',I,',',J,')= ');
READLN(TAB[I,J]);
end;

writeln('Les coefficients du tableau TAB(',N,',',M,') sont :');
for I:=1 to N do
for J:=1 to M do
writeln('TAB(',I,',',J,')= ',TAB[I,J]);

END.
```



Cours N°6

Le Traitement des Chaînes de Caractères

Cours élaboré par Mrs RENDJIMANE, BOUFATAH & BRAHMI



2. Représentation des variables chaînes

- En Pascal, les variables chaînes sont nommées au moyen d'un identificateur quelconque
- Les constantes chaînes s'écrivent sous la forme d'une suite de caractères encadrées par des guillemets simple ` `
- Si une apostrophe doit apparaître à l'intérieur de la chaîne, elle doit être doublée

Exemple: `ch := '1''affectation';`



3. Déclaration des variables chaînes

- Les variables chaînes se définissent dans la partie déclarative du programme en précisant leur longueur maximale.
- La déclaration d'une chaîne se fait à l'aide du mot réservé "string" suivie d'une constante entière, entre une paire de crochets, dont la valeur doit être comprise entre 1 et 255.

Exemple:

```
Const lg_max = 30;  
Var X:String[15];  
    Y:String[lg_max];
```



4. Initialisation et affectation d'une chaîne

- On peut affecter à toute variable chaîne une expression ou une constante chaîne de caractères en utilisant le symbole d'affectation traditionnel " := "

Exemple:

Pour les déclarations suivantes:

```
Var X:String[10];  
    Y:String[15];
```

Ces affectations sont correctes:

```
X := 'Vitesse';  
  
Y := X;  
  
Y := 'Temps';  
  
X := Y;
```



4. Initialisation et affectation (suite)

- Lorsque la variable réceptrice a une longueur maximal inférieure à la longueur de la chaîne qu'on cherche à lui affecter, celle-ci est simplement tronquée par la droite.

Exemple:

Pour les déclarations suivantes:

```
Var X:String[10];  
    Y:String[15];
```

```
Y:='Accélération';  
X:=Y;
```

On obtient pour la variable x la chaîne **Accélérati**



5. Comparaison des chaînes

- Les comparaisons de chaînes sont basées, comme les comparaisons des caractères, sur l'ordre des codes ASCII de chacun des caractères qui les constituent.
- L'égalité de deux chaînes a lieu lorsqu'elles ont même longueur courante (pas nécessairement même longueur maximale) et qu'elles sont constituées des mêmes suites de caractères.



5. Comparaison des chaînes (suite)

Exemples:

1. Pour les déclarations suivantes:

```
Var X:String[10];  
    Y:String[15];
```

On obtient

```
X:='Vitesse';  
Y:='Vitesse';  
X:=Y;
```

2. 'vecteur' < 'vitesse'

'Physique1' < 'Physique2'

'bon' < 'bonne'

'12' < '2'



6. Lecture - Écriture des chaînes

A. Instruction de Lecture :

- L'instruction `Readln` permet de lire des chaînes de caractères comme n'importe quelle autre variable d'un autre type.
- En ce qui concerne la manière dont la chaîne est délimitée, le langage Pascal cherche à lire un nombre de caractères correspondant à la longueur maximale, mais il s'interrompt à la rencontre d'une validation par la touche 'Entrer'.
- La lecture doit toujours se faire par `Readln` et non pas par `Read`. (La lecture par `Read` exclut la possibilité de lire convenablement plusieurs chaînes de suite).



6. Lecture - Écriture des chaînes (suite)

A. Instruction de Lecture :

Exemple:

Supposons qu'on veut saisir 2 variables chaînes dont les valeurs sont : **X=Accélération** **Y=Vitesse**

Avec:

```
Var X:String[15];  
    Y:String[10];
```

```
Read(X,Y);
```

Exécution

```
AccélérationVitesse
```



6. Lecture - Écriture des chaînes (suite)

B. Instruction d'Écriture :

- Les instructions "write" et "writeln" permettent d'écrire des chaînes de caractères comme n'importe quelle autre variable d'un autre type.



6. Lecture - Écriture des chaînes (suite)

B. Instruction d'Écriture :

Exemple: Rangement de 2 chaînes de caractères (Par ordre alphabétique)

```
Program rangement;
Uses wincrt;
Var mot,mot1,mot2:String[20];
Begin
  Write('Donnez un premier mot: ');
  Readln(mot1);
  Write('Donnez un deuxième mot: ');
  Readln(mot2);
  If mot1 > mot2 Then
    Begin
      mot:=mot1;
      mot1:=mot2;
      mot2:=mot;
    End;
  Writeln('Voici vos deux mots rangés:');
  Writeln(mot1,' ',mot2);
End.
```



7. Lien entre le type Chaîne et le type Caractère

- Le type chaîne et le type caractère sont deux types différents.
- Le type caractère est compatible avec le type chaîne, en revanche le type chaîne n'est pas compatible avec le type caractère.

Exemple:

Pour les déclarations suivantes:

```
Var C:Char;  
    mot:String[1];
```

Ces affectations sont correctes:

```
mot:=C;  
mot:='a';
```

Cette affectation est incorrecte:

```
C:=mot;
```



8. Fonctions et Procédures relatives au type chaîne

A. Fonctions sur les chaînes de caractères :

■ **Fonction Concaténation :**

- Cette opération consiste à juxtaposer deux ou plusieurs chaînes de caractères pour n'en former qu'une seule.
- Elle peut s'obtenir avec l'opérateur "+" ou la fonction "Concat"

Exemple:

Pour les déclarations suivantes:

```
Var A,B,X,Y:String[25];
```

```
A:='Matière';  
B:='Informatique';  
X:=A+' '+B;  
Y:=Concat(A,' ',B);  
Writeln(X);  
Writeln(Y);
```

Exécution

```
Matière Informatique  
Matière Informatique
```

Le résultat d'une concaténation ne peut dépasser 255 caractères. Dans le cas contraire on obtient une erreur à l'exécution.



8. Fonctions et Procédures relatives au type chaîne

(suite)

A. Fonctions sur les chaînes de caractères :

■ **Fonction longueur d'une chaîne (Length) :**

- Cette fonction permet d'obtenir la longueur effective d'une chaîne de caractères (y compris les blancs et les caractères non affichables).
- Le résultat obtenu à partir de cette fonction est de type entier.

Syntaxe: Length(Chaîne)

Exemple:

Pour les déclarations suivantes:

```
mot1 := 'Vitesse';  
mot2 := ' Accélération';  
X := Length(mot1);  
Y := Length(mot2);  
Writeln('X= ', X, ' Y= ', Y);
```

```
Var mot1, mot2: String[15];  
X, Y: Integer;
```

Exécution

X= 7 Y= 13



8. Fonctions et Procédures relatives au type chaîne

(suite)

A. Fonctions sur les chaînes de caractères :

■ **Fonction d'extraction de sous chaîne (Copy) :**

- Cette fonction retourne une chaîne de caractères extraite d'une autre chaîne de caractères .
- La fonction "Copy" permet d'extraire une chaîne d'une longueur donné, à partir d'une position donnée.

Syntaxe: Copy(Chaîne, Position, longueur)

Exemple:

Pour les déclarations suivantes:

```
Var X,Y:String[15];
```

```
X:=`Informatique`;
```

```
Y:=Copy(X,3,6);
```

```
Writeln(`Y= `,Y);
```

Exécution

```
Y= format
```



8. Fonctions et Procédures relatives au type chaîne (suite)

A. Fonctions sur les chaînes de caractères :

■ **Fonction de localisation de sous chaîne (Pos) :**

– Cette fonction permet de situer une sous-chaîne dans une chaîne donnée. Dans le cas où elle s'y trouve, la fonction "Pos" fournit le rang du caractère où commence la sous-chaîne. Dans le cas contraire, elle fournit zéro (0).

Syntaxe: Pos (Sous-Chaîne , Chaîne)

Exemple:

Pour les déclarations suivantes:

```
Var mot1,mot2:String[25];  
X,Y:Integer;
```

```
mot1:='Matière Informatique';  
mot2:='format';  
X:=Pos('Mat',mot1);  
Y:=Pos(mot2,mot1);  
Writeln('X= ',X,' Y= ',Y);
```

Exécution

X= 1 Y= 11



8. Fonctions et Procédures relatives au type chaîne (suite)

B. Procédures sur les chaînes de caractères :

■ **Procédure de suppression de sous chaîne (*Delete*) :**

– Cette procédure permet de supprimer un ou plusieurs caractères d'une chaîne à partir d'une position donnée et avec une longueur donnée.

Syntaxe: `Delete(Chaîne, Position, longueur)`

Exemple:

Pour les déclarations suivantes:

```
Var X:String[25];
```

```
X:='Matière Informatique';
```

```
Delete(X,8,13);
```

```
Writeln('X= ',X);
```

Exécution

X= Matière

Si on cherche à supprimer plus de caractères qu'il n'est possible, il y aura suppression de la partie existante. Si la position indiquée sort des limites de la chaîne, la procédure n'aura aucune action.



8. Fonctions et Procédures relatives au type chaîne (suite)

B. Procédures sur les chaînes de caractères :

■ **Procédure d'insertion de sous chaîne (*Insert*) :**

- Cette procédure permet d'insérer une chaîne de caractères (source) dans une autre chaîne de caractères (destination) à partir d'une position déterminée.

Syntaxe: `Insert(source, destination, Position)`

Exemple:

Pour les déclarations suivantes:

```
Var X,Y:String[25];
```

```
X:= 'Turbo ' ;  
Y:= 'Le Pascal' ;  
Insert(X,Y,4) ;  
Writeln('Y= ',Y) ;
```

Exécution

```
Y= Le Turbo Pascal
```



8. Fonctions et Procédures relatives au type chaîne

(suite)

B. Procédures sur les chaînes de caractères :

■ **Procédure de conversion (*Str*) :**

- Cette procédure effectue la conversion d'un nombre (entier ou réel) en sa représentation sous la forme de chaîne de caractères.

Syntaxe: `Str(nombre, chaîne)`

Exemple:

```
program conversion;
uses wincrt;
var X,Y:string[25];
    S:Real;
Begin
  write('S= ');readln(S);
  STR(S:9:2,X);
  Y:='Num '+X;
  writeln('Y= ',Y);
End.
```

Exécution

S= 10.5

Y= Num 10.50



8. Fonctions et Procédures relatives au type chaîne (suite)

B. Procédures sur les chaînes de caractères :

■ **Procédure de conversion (*val*) :**

- Cette procédure convertit une chaîne de caractères représentant un nombre en sa valeur numérique.
- Le contenu de la chaîne doit correspondre aux règles d'écriture des nombres. Aucun espace ne doit se trouver en première ou en dernière position.
- Après l'appel de cette procédure, si la conversion a été effectuée, une variable code contient la valeur zéro.
- Dans le cas contraire, cette variable code contient la position du premier caractère de la chaîne qui empêche la conversion, et le contenu de la variable n'est pas défini.



8. Fonctions et Procédures relatives au type chaîne

(suite)

B. Procédures sur les chaînes de caractères :

■ **Procédure de conversion (val) : (suite)**

Syntaxe: val(chaîne, nombre, code)

Exemple:

```
program conversion;
uses wincrt;
var  Ch:string[25];
     S,code:integer;

Begin
  write('Ch= ');readln(Ch);
  Val(Ch,S,code);
  writeln('S= ',S);
  writeln('code= ',code);
End.
```

Exécution

```
Ch= 12
S= 12
Code= 0
```




9. Tableau de chaînes de caractères

- Si les chaînes sont des tableaux de caractères, les tableaux de chaînes sont en quelque sorte des tableaux de tableaux.

Exemple:

Pour déclarer un tableau de 10 chaînes de 20 caractères au maximum, on écrit :

```
Var Tab:Array[1..10] of String[20];
```

Nous accédons à chacune des chaînes en utilisant l'élément `Tab[I]` du tableau et nous accédons à n'importe quel caractère `J` de la chaîne `I` à l'aide de `Tab[I,J]`.



Cours N°7

Les Types Définis par l'Utilisateur – Type Ensemble

Cours élaboré par Mrs FENDAH MANE, BOUEATAH & BRAHMI



I. Types Définis par l'Utilisateur

- Jusqu'à maintenant, nous avons vu les types prédéfinis:
 - *Les types simples*: entiers, réels, caractère et booléen
 - *Le type tableau et le type chaînes de caractères*
- Le langage Pascal offre à l'utilisateur la possibilité de définir de nouveaux types de données. C'est ce qu'on appelle ***les types définis par l'utilisateur***.
- Ces nouveaux types permettent à l'utilisateur d'imaginer des traitements à la fois plus performants et plus souples.
- Il existe deux types définis par l'utilisateur:
 - *Le type scalaire par énumération ou le type énuméré*
 - *Le type intervalle*



1. Le Type Énuméré

- Les types énumérés permettent de représenter des valeurs en les énumérant au moyen de leur nom.
- Un type énuméré est un type dont les variables associées n'auront qu'un nombre très limité de valeur (256 au maximum).
- On ne peut alors lui affecter aucune autre valeur que celles prévues dans l'énumération.



1. Le Type Énuméré

A. Définition du type énuméré :

■ On définit une énumération en deux phases:

1) On définit (avant la déclaration des variables) un nouveau type dans lequel on définit l'énumération que l'on souhaite créer (sous forme d'identificateurs), en utilisant la syntaxe suivante:

```
Type <nom_enumeration>=(Val1,Val2,... );
```

2) Une fois l'énumération définie, on déclare des variables, possédant ce type, dans la partie déclarative réservée au variables selon la syntaxe suivante:

```
Var <nom_variable>:<nom_enumeration>;
```



1. Le Type Énuméré

A. Définition du type énuméré :

Exemple :

```
Type jour_semaine = (Samedi,Dimanche,Lundi,Mardi,  
                    Mercredi,Jeudi,Vendredi);  
Var jour : jour_semaine;
```

Dans cet exemple la variable `jour` peut contenir l'une des valeurs prévues dans la définition du type `jour_semaine`, c-à-d: `Samedi`, `Dimanche`, ... , `Vendredi`

Ici il ne s'agit pas de créer une variable chaîne de caractères `jour_semaine` qui peut être égale à l'un des jours de la semaine, mais un nouveau type de variables dont on définit les valeurs possibles.



1. Le Type Énuméré

B. Règles concernant la définition du type énuméré :

- 1) Un identificateur, dans une définition de type énuméré, ne peut pas être un mot réservé.

Exemple :

```
Type Note_Musique = (Do, Re, Mi, Fa, Sol, La, Si);
```

Cette déclaration n'est pas correcte parce que Do est un mot réservé.

- 2) Les constantes dans une déclaration de type par énumération doivent être des identificateurs

Exemple :

```
Type Nombres = (1, 2, 3);
```

Définition fausse

```
Type Nombres = (Un, Deux, Trois);
```

Définition correcte₆



1. Le Type Énuméré

B. Règles concernant la définition du type énuméré :

- 3) Un même identificateur ne peut pas désigner plusieurs choses différentes parce qu'il y a le risque d'ambiguïté.

Exemple :

```
Type Nombres = (Un, Deux, Trois, Quatre, Cinq);  
Numeros = (Un, Deux, Trois);
```

L'identificateur `Un` représente deux entités de type différents.

- 4) Tout identificateur d'une énumération doit être défini avant la déclaration des variable de ce type.

Exemple :

```
Var I, J: Nombres;  
Type Nombres = (Un, Deux, Trois);
```

Définition fausse



1. Le Type Énuméré

C. Propriétés du type énuméré :

- 1) Les types énumérés sont des types ordonnés. L'ordre est défini comme étant celui dans lequel on a énuméré les différentes valeurs du type.

Exemple :

```
Type jour_semaine = (Samedi, Dimanche, Lundi, Mardi,  
                     Mercredi, Jeudi, Vendredi);
```

Samedi < Dimanche < ... < Jeudi < Vendredi

- 2) Les opérateurs relationnels (=, <, >, <=, >=, <>) sont utilisés avec des éléments de même type.
- 3) A chaque valeur énumérée correspond un numéro d'ordre. La première valeur porte le numéro **0**, la seconde le numéro **1** etc.



1. Le Type Énuméré

C. Propriétés du type énuméré :

4) Les fonctions prédéfinies sur les types énumérés sont:

Succ(*expr_type_enumere*) fournit le **successeur** de la valeur donnée par *expr_type_enumere* (erreur si cette expression désigne la dernière valeur énumérée)

Pred(*expr_type_enumere*) fournit le **prédécesseur** de la valeur donnée par *expr_type_enumere* (erreur si cette expression désigne la première valeur énumérée)

Ord(*expr_type_enumere*) fournit le **numéro d'ordre** de la valeur donnée par *expr_type_enumere*

Il existe la fonction réciproque de la fonction **Ord**, c-à-d une fonction qui fait correspondre l'élément du type au rang correspondant. Elle porte simplement le nom du type.



1. Le Type Énuméré

C. Propriétés du type énuméré :

4) Les fonctions prédéfinies sur les types énumérés (suite)

Exemple :

```
Type jour_semaine = ( Samedi, Dimanche, Lundi, Mardi,
                      Mercredi, Jeudi, Vendredi );
```

Succ(Samedi) vaut **Dimanche**

Pred(Jeudi) vaut **Mercredi**

Ord(Samedi) vaut **0**

Ord(Vendredi) vaut **6**

jour_semaine(3) vaut **Mardi**



1. Le Type Énuméré

C. Propriétés du type énuméré :

- 5) Le Pascal n'admet aucune entrée-sortie sur des valeurs d'un type énuméré, c-à-d on ne peut ni lire ni écrire une variable de type énuméré. Néanmoins, il reste possible d'échanger des informations d'un tel type en prévoyant des instructions qui nous permettent de lire ou d'écrire ce type de variables.

Exemple :

Avec la déclaration suivante:

```
Type  Nombres = (Un,Deux,Trois);  
var   Numeros : Nombres;
```

On peut écrire:

```
Case Numeros of  
  Un : writeln('Un');  
  Deux : writeln('Deux');  
  Trois : writeln('Trois');  
End;
```



2. Le Type Intervalle

- Le type Intervalle est un type simple défini par l'utilisateur.
- Un intervalle permet de restreindre le groupe des valeurs d'un type appelé type de base et choisi parmi Integer, Char ou un Type Énuméré (Le type Real est exclu parce qu'il n'est pas de type ordinal).

Exemple :

0..9 est l'intervalle dont le type de base est *integer*

lundi..vendredi est l'intervalle dont le type de base est un
type énuméré

'A'..'Z' est l'intervalle dont le type de base est *char*



2. Le Type Intervalle

A. Définition du type intervalle :

■ On définit un type intervalle en deux phases:

1) On définit (avant la déclaration des variables) un nouveau type intervalle dans lequel on fait état de l'intervalle que l'on souhaite créer, en utilisant la syntaxe suivante:

```
Type <nom_intervalle> = <const_1>..<const_2>;
```

2) Une fois cet intervalle défini, on peut déclarer des variables de ce type selon la syntaxe suivante:

```
Var <nom_variable> : <nom_intervalle>;
```



2. Le Type Intervalle

A. Définition du type intervalle :

■ Remarques:

- ❑ `const_1` et `const_2` sont deux constante de même type tel que: `const_1 <= const_2`
- ❑ La valeur d'une variable de type intervalle ne peut en aucun cas dépasser les bornes de cet intervalle.

Exemple :

```
Type  Majuscule = 'A'..'Z';  
      Chiffre   = 0..20;  
      Nombres  = (Un,Deux,Trois,Quatre);  
      Int_Nombres = Un..Trois;  
Var   Lettre  : Majuscule;  
      X,Y    : Chiffre;  
      Numero : Int_Nombres;
```



2. Le Type Intervalle

B. Propriétés du type intervalle :

- Des variables de type intervalle ont exactement les mêmes propriétés que le type des constantes utilisées dans les bornes de cet intervalle: elles peuvent intervenir dans les mêmes expressions.
- Le seul point de différence est le fait qu'elles ne peuvent pas se voir affecter des valeurs situées en dehors de l'intervalle imposé.



2. Le Type Intervalle

Exemple:

```
Program type_int;  
Uses wincrt;  
Type jour=( Sam,Dim,Lun,Mar,Mer,Jeu,Ven );  
    Nbr_jour=0..6;  
Var J:jour;  
    nombre:Nbr_jour;  
Begin  
    Write('Donnez un nombre de jour (entre 0 et 6): ');  
    Readln(nombre);  
    J:=jour(nombre);  
    If(J=Jeu)or(J=Ven)Then Writeln('C''est un jour de repos')  
    Else Writeln('C''est un jour de la semaine');  
End.
```

Exécution

```
Donnez un nombre de jour (entre 0 et 6): 2  
C'est un jour de la semaine
```

16



3. Type Tableau

- On peut utiliser les tableaux comme étant des types définis par l'utilisateur. Ceci est possible en utilisant la définition suivante:

```
Type <Tableau>=Array[Indice1,Indice2] of <Type_élément>;  
Var <nom_Tableau> : <Tableau>;
```

Exemple :

```
Type Matrice = Array[1..5,1..10] of Real;  
Var A,B,C : Matrice;
```



II. Types Ensembles

- En Pascal, un ensemble est une collection d'objets, comportant des valeurs de même type (maximum 256).
- Comme pour le type intervalle, ces valeurs sont issues de types de base: *Integer*, *Char*, *Boolean*, *Intervalle* ou *un Type Énuméré*.
- Les ensembles sont les mêmes utilisés en mathématique.
- Ils sont donc régis par les mêmes lois classiques de:
 - Réunion
 - Intersection
 - Différence
 - Inclusion
 - Égalité
 - Inégalité
 - Contenance
 - Appartenance



2. Déclaration du Type Ensemble

- On déclare une variable de type ensemble en utilisant la syntaxe suivante:

```
Var <nom_ensemble> : Set of <Type_de_Base>;
```

Exemple :

```
Type Chiffre = 1..10;  
      Nombres = (Un,Deux,Trois,Quatre);  
  
Var   X,Y : Set of Chiffre;  
      Numero : Set of Nombres;  
      Lettre : Set of Char;
```



3. Propriétés du Type Ensemble

- 1) L'affectation de données à une variables déclarée de type ensemble se fait en utilisant les crochets "[]"

Exemple :

Pour la déclaration suivante: `Var Voyelles:Set of Char;`

On peut utiliser l'affectation suivante:

```
Voyelles:=[ 'a', 'e', 'i', 'o', 'u', 'y' ];
```

■ Remarques:

- L'écriture [] indique l'ensemble vide.
- L'ordre des éléments, dans une affectation ou une comparaison d'ensemble, n'est pas pris en considération puisque les ensembles ne sont pas ordonnés.



3. Propriétés du Type Ensemble

2) Les opérations sur les ensembles sont :

■ Réunion "+" :

Exemple :

```
A := [1, 2, 3];
```

```
B := [3, 4, 5];
```

```
C := A+B;
```

```
C vaut [1, 2, 3, 4, 5]
```

■ Différence "-" :

Exemple :

```
A := [1, 2, 3];
```

```
B := [3, 4, 5];
```

```
C := A-B;
```

```
D := B-A;
```

```
C vaut [1, 2]
```

```
D vaut [4, 5]
```



3. Propriétés du Type Ensemble

2) Les opérations sur les ensembles sont : (suite)

■ **Intersection "*" :**

Exemple :

```
A := [1, 2, 3];
```

```
B := [3, 4, 5];
```

```
C := A * B;
```

```
C vaut [3]
```

■ **Inclusion "<=" :**

Exemple :

```
A := [1, 2];
```

```
B := [1, 2, 3];
```

```
A <= B
```



3. Propriétés du Type Ensemble

2) Les opérations sur les ensembles sont : (suite)

■ **Égalité "=" :**

Exemple :

```
A := [1, 2];
```

```
B := [1, 2];
```

```
A = B
```

■ **Inégalité "<>" :**

Exemple :

```
A := [1, 2];
```

```
B := [3, 4, 5];
```

```
A <> B
```




3. Propriétés du Type Ensemble

2) Les opérations sur les ensembles sont : (suite)

■ **Contenance " \geq " :**

Exemple :

```
A := [1, 2];
```

```
B := [1, 2, 3];
```

```
B  $\geq$  A
```

■ **Appartenance "IN" :**

Exemple :

```
A := [1, 2, 3];
```

```
1 IN A
```

3) Il est impossible d'utiliser les procédures "Read" et "Write" avec les variables de type ensemble.



4. Exemple

```
Program ensemble;  
Uses wincrt;  
Var      Maj : Set of Char;  
         Min : Set of Char;  
         Lettre : Char;  
  
Begin  
  Write('Tapez une Lettre: ');  
  Readln(Lettre);  
  Maj:=['A'..'Z'];  
  Min:=['a'..'z'];  
  If (Lettre IN Maj) Then Writeln('C''est une MAJUSCULE')  
  Else If (Lettre IN Min) Then Writeln('C''est une minuscule')  
  Else Writeln('Ce n''est pas une lettre');  
  
End.
```

Exécution

```
Tapez une Lettre: A  
C'est une MAJUSCULE
```

25



Cours N°8

Fonction et Procédure

Cours élaboré par Mrs BENDAHMANE, BOUFATAH & BRAHMI



1. Introduction

- Dans tout langage de programmation évolué, il est possible de définir un bloc d'instructions qui pourra être appelé dans n'importe quelle partie du programme principal en faisant référence uniquement par son identifiant.
- Cette technique de programmation simplifie grandement les algorithmes (programmes) et fait appel à des **sous-programmes** effectuant chacun des tâches précises.
- Il existe deux types de sous-programme:
 - ✚ Les fonctions
 - ✚ Les procédures



2. Déclaration d'un sous-programme

- Pour pouvoir utiliser un sous-programme (appelé aussi sous-routine), il faut d'abord le déclarer.
- La déclaration des procédures et fonctions se fait après toutes les autres déclarations.
- La structure d'un programme contenant un sous-programme est comme suit:

```
entête  
déclaration des :  
    labels;  
    constantes;  
    types;  
    variables;  
définition des sous-programmes;  
BEGIN {Programme Principal}  
    instructions  
END.
```



3. Les Fonctions

Définition :

Une **fonction** est un groupe d'instructions doté d'un nom générique, qui exécute une tâche déterminée ou un algorithme. La fonction est identifiée à un type et restitue une valeur en fin d'exécution.

Exemple :

La fonction exponentielle en langage Pascal est **$y:=\exp(x);$**

La syntaxe fait référence à un nom générique (identificateur) de fonction **exp** et un argument **x**.

Remarque: Certaines fonctions peuvent avoir plusieurs arguments.



3. Les Fonctions (suite)

Déclaration d'une fonction :

La déclaration d'une fonction se fait, en général, après les déclarations des variables en Pascal, en précisant:

1. **Son type**
2. **Son nom**
3. **Une liste d'arguments dont on précise le type**

La déclaration algorithmique d'une fonction est sous la forme:

Fonction <Nom de fonction> (argument1: type1, argument2: type2,...)
: type de résultat

Exemple : **Fonction** somme(x: réel, y: réel, z: réel) : réel



3. Les Fonctions (suite)

Déclaration d'une fonction :

- En Pascal, la ligne de déclaration d'une fonction s'écrit sous la forme suivante:

Function <Nom_fonction> (Argument1: **type1**; Argument2: **type2**; ...)
: <Types de résultats>;

- Cette ligne précède immédiatement le bloc de définition qui commence **obligatoirement** par un "**begin**" et se termine par un "**end**";".
- Les paramètres sont séparés par des points-virgules ";".
- Si plusieurs paramètres sont du même type, on peut les placer ensemble séparés par des virgules et suivies par leur type.
- La valeur renvoyée par la fonction peut être utilisée dans n'importe quelle instruction ou expression compatible avec son type.



3. Les Fonctions (suite)

Exemple 1 : Programme utilisant une fonction qui renvoie le maximum de deux entiers.

Algorithme max_entier

Variables utilisées:

a,b,x : nombres entiers

Fonction maximum(a1,b1 : entier) : entier

Début { de la fonction maximum }

Si $a1 \geq b1$ alors

maximum = a1

Sinon maximum = b1 FSI

Fin { de la fonction maximum }

1) Début { Programme Principal }

2) Lire(a,b)

3) $x = 2 * \text{maximum}(a,b)$

4) Écrire(x)

5) Fin { Programme Principal }



3. Les Fonctions (suite)

Programme Pascal

```
Program max_entier ;
Uses wincrt;

VAR A,B,X : INTEGER;

Function maximum(a1,b1:INTEGER) : INTEGER;
Begin
If a1 >= b1 then maximum:=a1
else maximum:=b1;
END;

BEGIN (*Programme Principal*)

write('Entrer le premier nombre entier A:'); readln(A);
write('Entrer le deuxième nombre entier B:'); readln(B);

x:=2*maximum(A,B);
writeln('La valeur de x: ',x);

END. (*Programme Principal*)
```



3. Les Fonctions (suite)

- Dans cet exemple, les arguments "a1" et "b1" sont utilisés pour calculer le maximum. On les appelle **paramètres formels (effectifs)** de la fonction.
- Les **paramètres formels** sont utilisés uniquement dans le traitement à l'intérieur du sous-programme.
- les variables "A", "B" et "X" sont déclarés dans le programme principal. On les appelle **variables globales**.
- **Une variable globale** déclarée dans le programme principal est reconnue à la fois par le programme principal et par tous les sous-programmes qui sont déclarées par la suite.



3. Les Fonctions (suite)

Exemple 2 : Programme utilisant une fonction qui calcule la surface d'un rectangle.

Algorithme surface_rectangle

Variables utilisées:

surface, largeur, longueur : nombres réels

Fonction surf_rect(L1, L2 : réel) : réel

Début { de la fonction }

surf_rect = L1*L2

Fin { de la fonction }

1) Début { Programme Principal }

2) Afficher ('Entrer la longueur et a largeur du rectangle')

3) Lire(longueur, largeur)

4) surface = surf_rect(longueur, largeur)

5) Afficher('La surface est: ', surface)

6) Fin { Programme Principal }



3. Les Fonctions (suite)

Programme Pascal

```
Program surface_rectangle ;
Uses wincrt;

VAR surface,largeur,longueur : REAL;

Function surf_rect(L1,L2:REAL) : REAL;
Begin
Surf_rect:=L1*L2
END;

BEGIN (*Programme Principal*)

write('Entrer la longueur du rectangle: '); readln(longueur);
write('Entrer la largeur du rectangle: '); readln(largeur);

surface:=surf_rect(longueur,largeur);
writeln('La surface du rectangle: ',surface:10:3);

END. (*Programme Principal*)
```



4. Les Procédures

Définition :

Une procédure est un groupement d'instructions, identifié par un nom, qui exécutent une tâche ou un algorithme déterminé. Elle est un bloc de programme qui travaille sur des données fournies ou contenues dans le bloc de programme. Aucune valeur n'est associée au nom de la procédure.

- Une procédure est une fonction qui ne renvoie aucune valeur.

Exemple :

exp(x): une fonction qui renvoie une valeur.

Effacer_ecran: c'est une tâche à exécuter mais ne renvoie aucune valeur.



4. Les Procédures (suite)

Déclaration d'une procédure :

- La déclaration d'une procédure se fait de la même manière qu'une fonction avec la différence de ne pas préciser son type.
- La déclaration algorithmique d'une fonction est sous la forme:

Procédure <Nom procédure> (argument1: type1, argument2: type2,...)

Exemple : **Procédure** factoriel(n: entier)

- En Pascal, la ligne de déclaration d'une procédure s'écrit sous la forme suivante:

Procedure <Nom_procedure> (Argument1: type1; Argument2: type2; ...);

- Cette ligne précède immédiatement le bloc de définition qui commence **obligatoirement** par un "**begin**" et se termine par un "**end;**".



4. Les Procédures (suite)

Exemple 1 : Algorithme d'une procédure qui calcule le factoriel d'un nombre entier.

Procédure factoriel (n : entier)

Variables utilisées:

fact , i : nombres entiers

Début { de la Procédure }

Si n = 0 alors fact = 1

Sinon

Fact = 1

i = 1

Tant que i <= n faire

Fact = fact * i

i = i+1

Fin tant que

FSI

Afficher('Factoriel de: ', n, ' est ', fact)

FIN {de la Procédure }



4. Les Procédures (suite)

Exemple 1 (suite) :

- Dans cet exemple, on utilise un paramètre formel "n" et deux **variables locales "fact" et "i"**.
- **Une variable locale** est déclarée localement dans la procédure (ou fonction) et sa portée est limitée à la procédure ou (fonction). Elle n'est pas reconnue dans le programme principal ni dans les autres sous-programmes (s'ils existent).

Exemple 2 : Programme qui permet à partir de la saisie de 3 nombres A, B, C: soit de calculer le factoriel de A, soit le factoriel de B ou celui de C par un choix. Le calcul du factoriel doit être effectué par une procédure.



Programme 2 :

```
Program Factoriel_entier;
uses wincrt;

VAR A,B,C: INTEGER;
    ch: CHAR;

Procedure factoriel(n:INTEGER);
var fact,i: INTEGER;

begin (*Debut procedure*)
  IF n = 0 THEN fact :=1
  ELSE
    Begin
      fact:=1;
      i:=1;
      while(i <= n) do
        begin
          fact:=fact*i;
          i:=i+1
        end;
      end;
      writeln('Le factoriel de ',n,' est ',fact);
    end; { *procedure* }
```



Programme 2 (suite) :

```
BEGIN (*Programme Principal*)
  writeln('Entrer trois nombres entiers');
  readln(A,B,C);

  Writeln('Voulez-Vous:');
  writeln('1 - Calculer le factoriel de ',A);
  writeln('2 - Calculer le factoriel de ',B);
  writeln('3 - Calculer le factoriel de ',C);
  write('Entrer votre choix');readln(ch);

  case ch of
    '1': factoriel(A);
    '2': factoriel(B);
    '3': factoriel(C);
    else writeln('Erreur dans votre choix')
  end;
END.
```



4. Les Procédures (suite)

Remarques :

- La fonction est un cas particulier de la procédure. La différence entre fonction et procédure se trouve à deux niveaux :
 - ✚ Au niveau du résultat, la fonction délivre un résultat et un seul.
 - ✚ Au niveau de l'appel, l'appel apparaît toujours dans une expression ou affectation.
- Une procédure peut ne pas posséder aucuns paramètres. Dans ce cas, elle réalise toujours la même action lorsqu'on l'invoque.



5. Procédures et fonctions récursives

Définition :

Une procédure ou une fonction est dite récursive si elle s'appelle d'elle-même.

Exemple Programme permettant de faire le calcul du factoriel sous forme récursive.



5. Procédures et fonctions récursives (suite)

Algorithme Programme permettant de faire le calcul du factoriel sous forme récursive.

Algorithme récursive

Variables utilisées:

n,resultat : nombres entiers

Fonction factoriel(n : entier) : entier

Début { de la fonction}

Si $n \leq 1$ alors factoriel = 1

Sinon factoriel = $n \cdot \text{factoriel}(n-1)$ FSI

Fin { de la fonction}

1) Début { Programme Principal}

2) Lire(n)

3) resultat = factoriel(n)

4) Écrire(resultat)

5) Fin {Programme Principal}



5. Procédures et fonctions récursives (suite)

Programme permettant de faire le calcul du factoriel sous forme récursive.

```
PROGRAM RECURSIVE;  
  
USES WINCRT;  
  
VAR n,resultat : INTEGER;  
  
FUNCTION factoriel(n:INTEGER) : INTEGER;  
  
begin  
if n <= 1 then factoriel:=1  
else factoriel:=n*factoriel(n-1);  
end;  
  
BEGIN  
  
write('Entrer un entier positif: ');readln(n);  
resultat:=factoriel(n);  
writeln('Le factoriel de ',n,' est: ',resultat);  
  
END.
```

Attention: Il faut toujours vérifier qu'en aucun cas on ne puisse avoir une boucle infinie qui bloquerait la machine.



6. Remarque IMPORTANTE

En cas d'utilisation de tableaux comme paramètres formels, il faut déclarer le type de ces tableaux immédiatement après l'entête du programme principal et utiliser le mot clé "**VAR**" pour déclarer ces tableaux dans les parties déclarations des fonctions ou procédures.

Exemple

```
Program exemple;  
Type Matrice = ARRAY[1 .. 100; 1 .. 100] of REAL;  
Procedure Liste(VAR Tableau : Matrice);  
    :  
Function somme(VAR TAB:Matrice) : REAL;  
    :  
    :
```




Cours N°9

Les Fichiers de Type Texte



1. Introduction

- Le type fichier est un type structuré au même titre que le type tableau, à la différence que le tableau a une taille fixe et que le fichier est a priori illimité.
- Les types de variables étudiés précédemment sont utilisés pour décrire des informations stockées en mémoire centrale. La durée de vie de ce type d'information est égale au temps d'exécution du programme.
- Le type fichier permet la manipulation des informations stockées en une mémoire secondaire telle que le disque dur, clé USB ...



1. Introduction

Définition d'un Fichier:

Un fichier est une suite de composantes de même type qui contient de l'information codée en binaires (bits), définis par un identificateur au même titre qu'une variable et enregistrée sur un support de mémoire non volatil.

- Les fichiers sont utilisés pour la communication entre un programme et son environnement, ou entre des programmes différents ; ils permettent aussi de conserver temporairement de grands volumes de données.
- Il existe différents types de fichiers. En ce qui nous concerne nous allons nous limiter aux **Fichiers de Type Texte**.



2. Fichiers Texte

- Un fichier de type texte est constitué d'une suite de caractères affichables et de caractères de contrôle groupés en lignes, comme dans un texte. Chaque ligne est terminée par une marque de fin de ligne qui est une séquence.
- Ces fichiers sont manipulés d'une manière séquentielle c-à-d l'accès à ce type de fichiers peut s'effectuer d'un élément à l'autre, en partant du premier.
- Les fichiers de type texte sont appelés aussi « fichiers ASCII » parce que leur contenu peut être visualisé à l'aide de n'importe quel éditeur de texte.
- Nous déclarons les fichiers de type texte, comme suit:

```
Var <nom_Logique_Fichier> : Text;
```



3. Opérations sur les Fichiers Texte

- Les opérations sur les fichiers de type texte se font dans une séquence précise :

1) Assignation du fichier

2) Ouverture du fichier en lecture, écriture ou ajout

3) Traitement du fichier – opérations de lecture/écriture

4) Fermeture du fichier



3. Opérations sur les Fichiers Texte (Suite)

A. Assignation du fichier : (Assign)

- Un fichier doit avoir deux noms: un **nom logique** (Interne: en mémoire centrale) et un **nom physique** (Externe: sur le disque).
- L'assignation est l'association du nom logique avec le nom physique de ce fichier.
- Toutes les opérations effectuées sur un fichier, concernent le fichier situé sur le disque.

Syntaxe: `Assign (<Nom Logique> , <Nom Physique>) ;`

Exemple: `Assign (Fich , 'D: \LMD \Resulat.dat') ;`

Nom Logique

Nom Physique

Chemin d'Accès



3. Opérations sur les Fichiers Texte (Suite)

B. Ouverture du Fichier en Lecture, Écriture ou Ajout

■ *Ouverture du fichier en Lecture: (Reset)*

- La procédure `Reset` permet d'**ouvrir un fichier existant**, sans écraser son contenu, et positionne le pointeur de fichier au début de ce dernier.
- Cette procédure permet d'ouvrir le fichier en lecture seulement.

Syntaxe: `Reset (<Nom Logique>);`

Exemple:

```
Reset (Fich);
```



3. Opérations sur les Fichiers Texte (Suite)

B. Ouverture du Fichier en Lecture, Écriture ou Ajout

■ *Ouverture du fichier en Écriture: (Rewrite)*

- La procédure `Rewrite` permet la **création d'un nouveau fichier** ou réécrire complètement un fichier existant.
- Le pointeur de fichier se positionne automatiquement au début de ce dernier.

Attention: Si un fichier disque de même nom existe déjà, il est détruit et remplacé par un nouveau fichier vide.

Syntaxe: `Rewrite (<Nom Logique>);`

Exemple: `Rewrite (Fich);`



3. Opérations sur les Fichiers Texte (Suite)

B. Ouverture du Fichier en Lecture, Écriture ou Ajout

■ *Ouverture du fichier en mode Ajout: (Append)*

- C'est une procédure qui permet l'**ouverture d'un fichier en mode ajout**, c-à-d qu'on peut ajouter des lignes de texte à la fin du fichier.
- Après un appel à la procédure `Append`, le pointeur de fichier se positionne à la fin de ce dernier et seule l'écriture est autorisée.

Syntaxe: `Append (<Nom Logique>);`

Exemple:

`Append (Fich);`



3. Opérations sur les Fichiers Texte (Suite)

C. Lecture / Écriture dans un fichier

■ *Lecture dans un fichier:*

- Les procédures `Read` ou `Readln` permettent de lire les valeurs de variables à partir d'un fichier.

Syntaxe:

```
Read(<Nom Logique>, <Liste de variables>);
```

■ *Écriture dans un fichier:*

- Les procédures `Write` ou `Writeln` permettent d'écrire les valeurs de variables dans d'un fichier.

Syntaxe:

```
Write(<Nom Logique>, <Liste de variables>);
```



3. Opérations sur les Fichiers Texte (Suite)

D. Fermeture d'un fichier : (Close)

- `Close` est une procédure standard qui ferme un fichier ouvert.
- Un fichier fermé ne peut faire l'objet de transferts de données.
- Cette procédure permet la fermeture d'un fichier préalablement ouvert par `Reset`, `Rewrite` ou `Append`.
- Tout programme qui utilise des fichiers doit se terminer par la fermeture des fichiers.

Syntaxe: `Close (<Nom Logique>);`

Exemple:

```
Close(Fich);
```



Exemple

```
Program Fichier;  
uses wincrt;  
var binf,bsup,i,N1,N2:integer;  
    x,f:real;  
    Fich1,Fich2:text;  
BEGIN  
    Assign(Fich1,'Don.dat');  
    Reset(Fich1);  
    Read(Fich1,binf,bsup,N1,N2);  
    close(Fich1);  
    Assign(Fich2,'Res.dat');  
    Rewrite(Fich2);  
    For i:=binf to bsup do  
        Begin  
            x:=i/10;  
            f:=sqr(x);  
            writeln(Fich2,x:7:2,' ',f:15:8);  
        end;  
    Close(Fich2);
```



Exemple (suite)

```
Assign(Fich2, 'Res.dat');  
Append(Fich2);  
For i:=N1 to N2 do  
  Begin  
    x:=i/10;  
    f:=sqr(x);  
    writeln(Fich2,x:7:2, ' ', f:15:8);  
  end;  
Close(Fich2);  
end.
```