

---

***Support de Cours*** Algorithmes et structure de données 1

Niveau 1<sup>ère</sup> année cycle préparatoire intégré scientifique

***Etablissement*** ISSAT Sousse

***Année universitaire*** 2020-2021

---



Ministère de l'Enseignement Supérieur et de la Recherche Scientifique



Université de Sousse

Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Cours **Algorithmique et structures de données 1**

**Dr. Nessrine ELLOUMI**

*Assistante vacataire à ISSAT Sousse*

**Dr. Aicha BEN MAKHLOUF**

*Assistante contractuelle à ISSAT Sousse*

**Niveau : 1<sup>ère</sup> année cycle préparatoire intégré scientifique**

**Année universitaire**

**2020-2021**

**Directeur de Département  
Informatique  
Moeiz ROUIS**



# Algorithmique et structures de données I

## Enseignantes:

**Dr. Aicha BEN MAKHLOUF**  
**Dr. Nessrine ELLOUMI**

**Niveau: 1<sup>ère</sup> année Préparatoire MPI**

Année universitaire 2020 – 2021

1

## Objectifs du cours

### A la fin de ce cours l'étudiant doit:

- Acquérir les connaissances fondamentales de l'algorithmique
- Connaître la structure générale d'un algorithme
- Comprendre les notions de base pour écrire un algorithme
- Comprendre les structures conditionnelles et les traitements itératifs (Boucles)
- Appliquer les concepts des Fonctions et des Procédures
- Maîtriser les type Tableau / Matrice ...
- Comprendre les algorithmes de tri (Tri à bulles, tri par sélection, tri par insertion..)
- Comprendre le type Enregistrement
- Découvrir, le principe récursif, comprendre son fonctionnement et l'appliquer dans la résolution des problèmes.
- Manipulation des objets dynamiques (Déclaration, Création, Accès, Suppression d'un objet dynamique)

2

## Plan du cours

- **Chapitre 1:** Introduction et notions de base
- **Chapitre 2:** Les types et les actions simples
- **Chapitre 3:** Les structures conditionnelles et les structures répétitives
- **Chapitre 4:** Les sous-programmes
- **Chapitre 5:** Les chaînes de caractères
- **Chapitre 6:** Le type tableau
- **Chapitre 7:** Les enregistrements
- **Chapitre 8:** La récursivité
- **Chapitre 9:** Les pointeurs

3



Université de Sousse  
Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Algorithmique et structures de données

## Chapitre 1 : Introduction et notions de base

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

4

## Plan du chapitre :

- **Partie I: Introduction à l'algorithmique**

1. Définitions
2. Cycle de vie d'un programme
3. Notions de bases
4. Les étapes de résolution d'un problème
5. Les spécificités d'un algorithme
6. Les instructions de base
7. Environnement algorithmique

- **Partie II : Généralités et notions de Base**

1. Structure générale d'un Algorithme
2. Le nom de l'algorithme
3. Notion de variable
4. Notion de constante
5. Notion de type

5

## I. INTRODUCTION À L'ALGORITHMIQUE

### 1. Définitions :

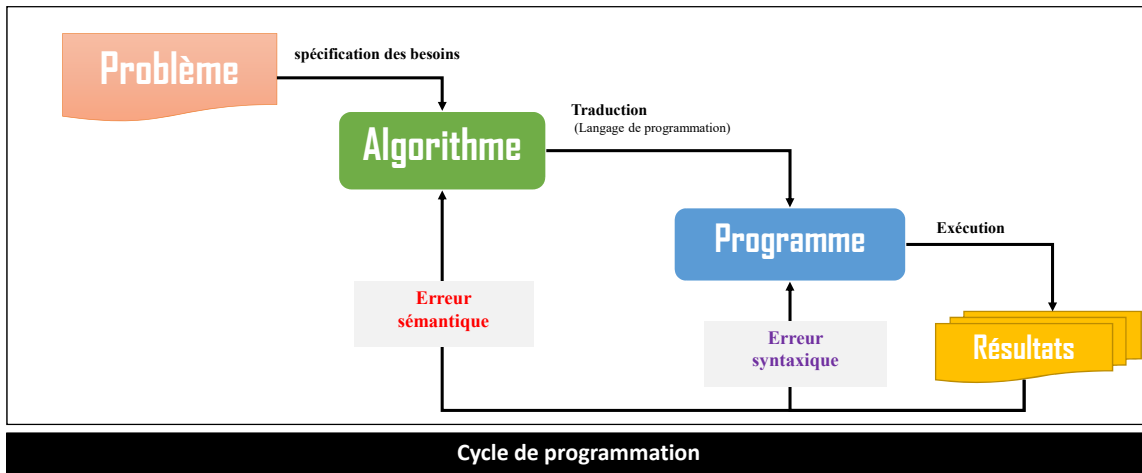
- **Utilisateur** : est une personne autorisée à exploiter et à manipuler un programme.
- **Programme (Logiciel)** : est une liste d'instructions d'une solution informatisée d'un problème donné.
- On distingue deux types de logiciels tels que:
  - Logiciels de base : système d'exploitation
  - Programmes : Logiciel de traitement de texte, Conception graphique, Edition de son, comptabilité, etc...

6

## I. INTRODUCTION À L'ALGORITHMIQUE

### 2. Cycle de vie d'un programme :

- Le développement d'un programme nécessite le suivi d'un cycle composé de différentes étapes.



7

## I. INTRODUCTION À L'ALGORITHMIQUE

### 2. Cycle de vie d'un programme :

- La résolution d'un problème posé consiste à définir les différents besoins demandés par l'utilisateur.
- La spécification des besoins est la partie **pertinente** dans un cycle de programmation.
- L'analyse des besoins permet de définir les spécifications formelle d'un algorithme sous forme de données (Conditions d'entrées, sorties), d'actions (Fonctions, Procédures) et résultats (Solution attendue).
- Cette phase consiste à comprendre le problème et déterminer les entrées et les résultats attendus sous forme d'une première solution appelée : **Algorithme**.

8

## I. INTRODUCTION À L'ALGORITHMIQUE

### 2. Cycle de vie d'un programme :

- La deuxième étape consiste à **traduire l'algorithme** à l'aide d'un langage de programmation choisi par le programmeur (Java, Python, C++, etc...).
- L'étape de traduction, exige **le respect** de la syntaxe du langage de programmation.
- La troisième étape permet **d'exécuter** les instructions écrites par le programmeur dans l'étape précédente.
- L'étape **d'exécution** peut générer des erreurs:
  - ✓ Erreurs syntaxiques : ce qui entraîne des corrections en général **simples** (erreur de syntaxe du langage de programmation). Le retour vers le programme est fréquent.
  - ✓ Erreurs sémantiques : ce qui entraîne des corrections difficiles à découvrir. Le programme produit des résultats inattendus. Le retour vers l'étape de spécification des besoins est inévitable.

9

## I. INTRODUCTION À L'ALGORITHMIQUE

### 3. Notions de bases:

#### Définitions :

#### • Algorithme :

- Un algorithme : est une représentation formelle permettant de résoudre un problème. Cette représentation contient deux parties essentielles :
  - ✓ Une première partie (l'entête) définit l'ensemble des variables et/ou les objets à utiliser.
  - ✓ Une deuxième partie (le corps) contenant une suite d'actions (procédures et fonctions) qu'on devra effectuer pour obtenir un résultat déterminé à partir d'une situation donnée.

#### • Programme :

- Un programme est un algorithme codé dans un **langage** compréhensible par l'ordinateur à l'aide d'un compilateur (traducteur).
- Un programme est une suite d'instructions (écrites en un **langage** de programmation) **exécutées** par l'ordinateur pour atteindre un objectif donné (résultat).

10

## I. INTRODUCTION À L'ALGORITHMIQUE

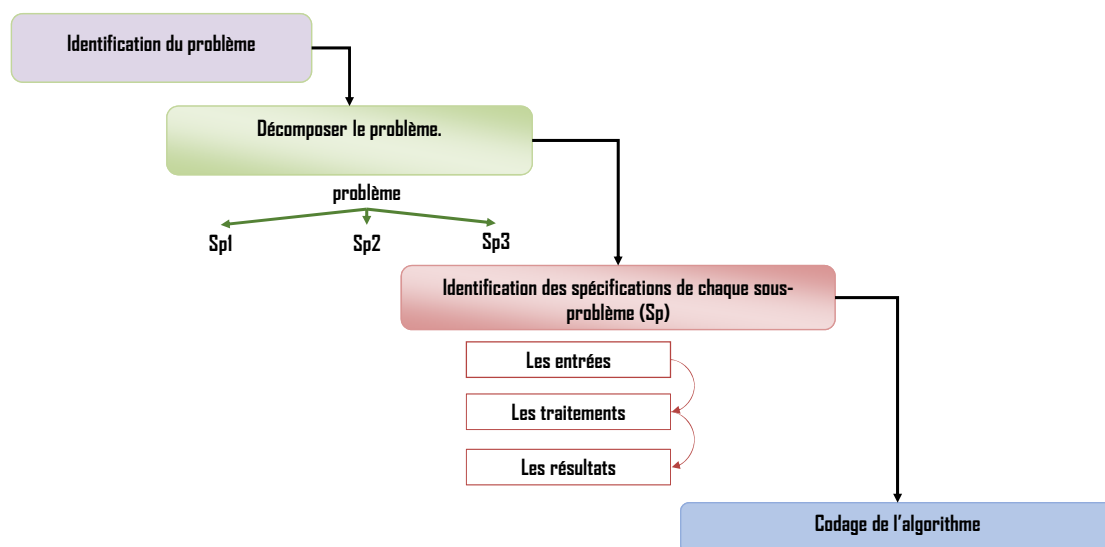
### 3. Notions de bases:

- Les termes « programme » et « algorithme » sont des **représentations formelles** de besoins pour résoudre un problème posé par un utilisateur.
- Un « **algorithme** » c'est la représentation d'une séquence d'instructions en langage algorithmique (abstrait).
- Un « **programme** » c'est la représentation d'une séquence d'instructions écrites suivant un langage de programmation exécuté par un ordinateur.

11

## I. INTRODUCTION À L'ALGORITHMIQUE

### 4. Les étapes de résolution d'un problème

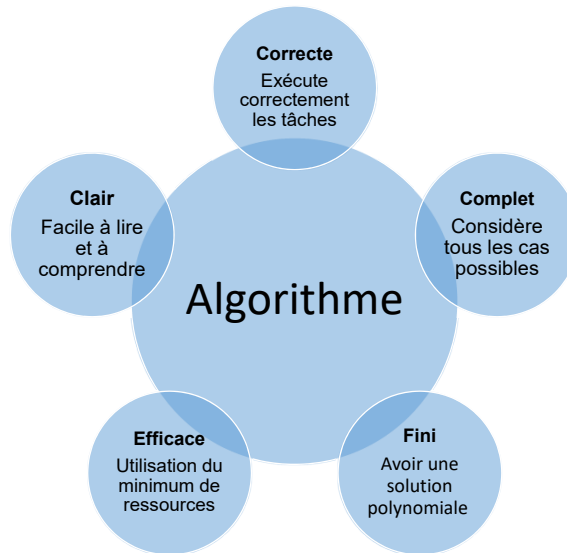


12



## I. INTRODUCTION À L'ALGORITHMIQUE

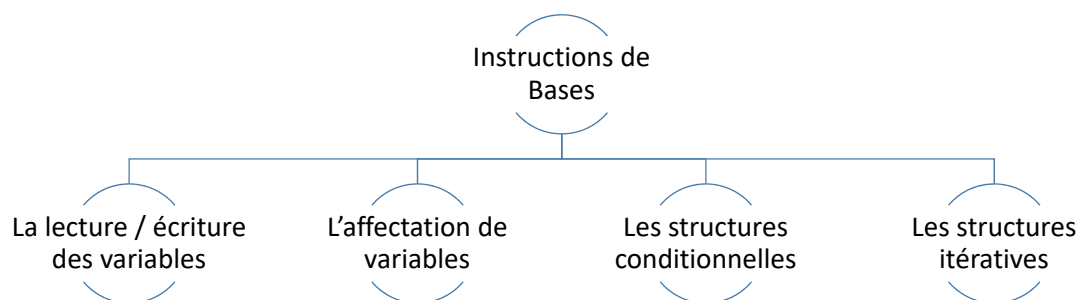
### 5. Les spécificités d'un algorithme :



13

## I. INTRODUCTION À L'ALGORITHMIQUE

### 6. Les instructions de bases



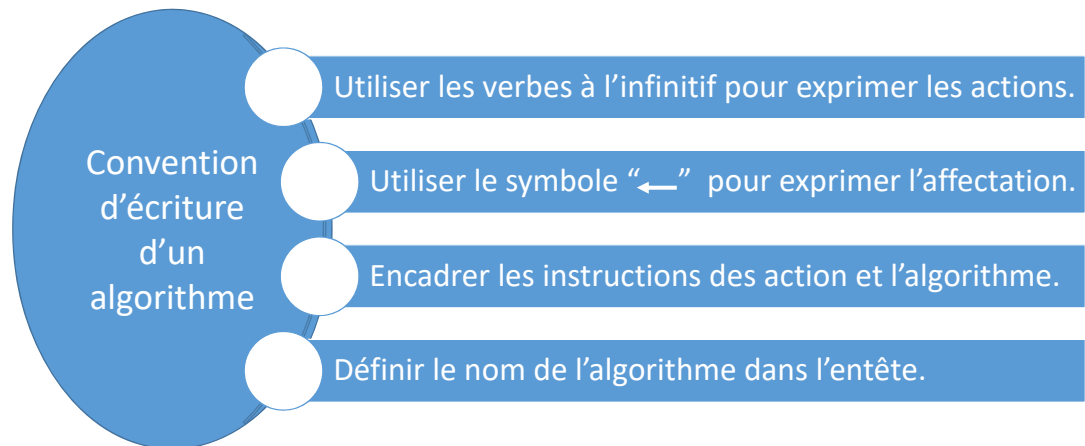
#### Important

Un algorithme c'est la combinaison des quatre instructions de base

14

## I. INTRODUCTION À L'ALGORITHMIQUE

### 7. Environnement algorithmique

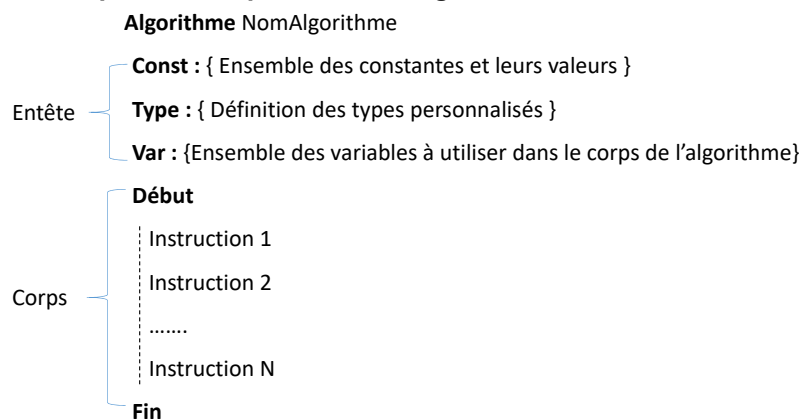


15

## I. INTRODUCTION À L'ALGORITHMIQUE

### 7. Environnement algorithmique

#### • Exemple d'une squelette d'un algorithme



- Les mots **Const**, **Type** et **Var** sont des mots clés qui peuvent exister ou non dans un algorithme selon les besoins.
- Les mots **Algorithme**, **Début** et **Fin** représentent les mots clés qui doivent exister dans un algorithme.

16

## I. INTRODUCTION À L'ALGORITHMIQUE

### 7. Environnement algorithmique

**Exemple** : Algorithme pour calculer la surface d'un rectangle

**Algorithme** CalculSurface1

**Const :**

Log = 4,32 ; Lag = 3,77

**Var :**

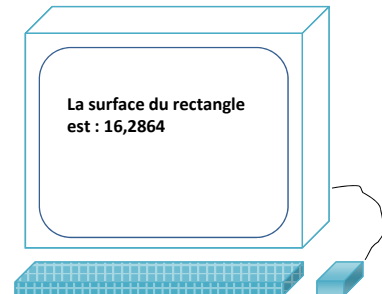
Surf : réel

**Début**

Surf  $\leftarrow$  Log \* Lag

Ecrire ("La surface du rectangle est :" , Surf)

**Fin**



17

## I. INTRODUCTION À L'ALGORITHMIQUE

### 7. Environnement algorithmique

**Exemple** : Algorithme pour calculer la surface d'un rectangle

**Algorithme** CalculSurface2

**Var :**

Surf : réel ; Log : réel ; Lag : réel ;

**Début**

Ecrire ("Quel est la longueur en (m)")

Lire (Log)

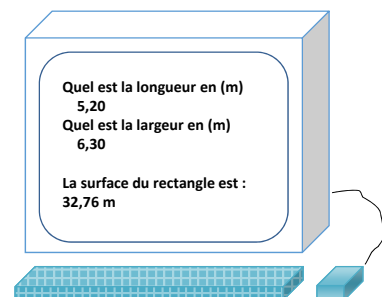
Ecrire ("Quel est la largeur en (m)")

Lire (Lag)

Surf  $\leftarrow$  Log \* Lag

Ecrire ("La surface du rectangle est :" , Surf, "m")

**Fin**



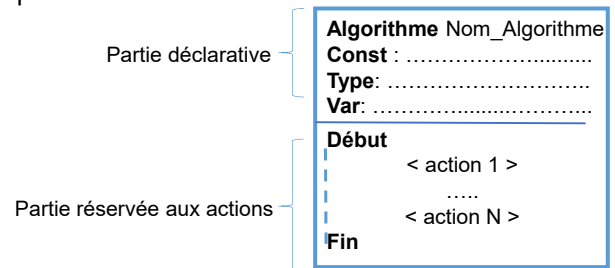
18

## II. GÉNÉRALITÉS ET NOTIONS DE BASE

### 1. Structure générale d'un Algorithme : Convention à adopter

Dans un algorithme, on trouve deux parties essentielles :

- Une partie déclarative (Entête): définissant l'ensemble des structures de données et/ou objets à utiliser qui seront impliqués par les différentes actions de l'algorithme. Cette partie contient les mots clés **Const** (constantes), **Type** et **Var** (Variables) qui peuvent exister ou non dans un algorithme selon les besoins.
- Une partie réservée aux actions (Corps): contenant la suite d'actions qu'on devra effectuer pour résoudre le problème. Cette partie est délimitée par les deux mots-clés Début et Fin.



19

## II. GÉNÉRALITÉS ET NOTIONS DE BASE

### 2. Notion de constante:

- Une constante est une variable définie par son nom et sa valeur et dont l'état reste inchangé tout le long d'un algorithme (seule la consultation est autorisée).
- Exemple:

```
Const:
    PI = 3,14
    x = 1
    ANNEE = 2021
```

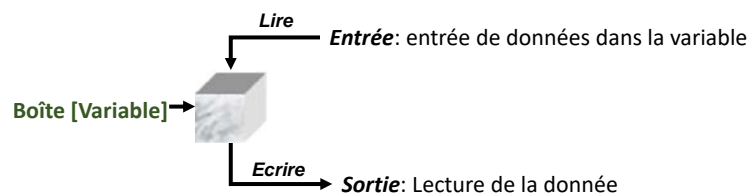
- La constante est déclarée suivant les règles de constitution d'un identificateur algorithmique.
- Il est interdit d'utiliser les symboles exemple ( **$\pi=3,14$** )
- La déclaration des variable mathématiques se fait à travers des caractères alphabétiques exemple (**PI=3,14**).

20

## II. GÉNÉRALITÉS ET NOTIONS DE BASE

### 3. Notion de variable :

- Une variable est caractérisée par:
  - ✓ son nom
  - ✓ son adresse
  - ✓ son contenu .
- Une variable ne peut contenir qu'une **seule valeur dans un instant T**.
- L'affectation d'une nouvelle valeur dans la variable écrase la valeur ancienne.



21

## II. GÉNÉRALITÉS ET NOTIONS DE BASE

### 4. Notion de type:

- Une variable contient des valeurs selon son domaine de définition.
- Un domaine de définition est appelé **type de la variable**.
- Un type est caractérisé par :
  - Ses bornes (inférieures et supérieures).
  - Les opérations effectuées sur les variables de type.
- Un domaine de définition doit être fini.
- Les bornes inférieures et supérieures dépendent des performances techniques de la machine (microprocesseur, bus, etc.) et celles du langage de programmation.

22

## II. GÉNÉRALITÉS ET NOTIONS DE BASE

### 4. Notion de type:

- les types simples: Supportés et reconnus par la majorité des langages de programmation. Il ne sont pas déclaré dans la partie réservée aux types.
- Les types complexes: ce sont des types personnalisés et construits à partir des types simples mais qu'il faut les déclarer dans la partie réservée aux types comme les tableaux et les enregistrements.

→ Exemple d'un type personnalisé:

**Type:**

planetes = { "Mercure", "Venus", "Terre", "Mars", "Jupiter" }

Algorithme	Nom_Algorithme
Const :	.....
<b>Type:</b>	.....
Var:	.....
<b>Début</b>	
	< action 1 >
	< action 2 >
	.....
	< action N >
<b>Fin</b>	

23



Université de Sousse  
Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Algorithmique et structures de données

## Chapitre 2 : LES TYPES ET LES ACTIONS SIMPLES

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

24

## TYPES ET ACTIONS SIMPLES

### • Plan du chapitre :

#### Partie I : Types

1. I. Introduction
2. II. Type Entier
3. III. Type Réel
4. IV. Type Caractère
5. V. Type Booléen

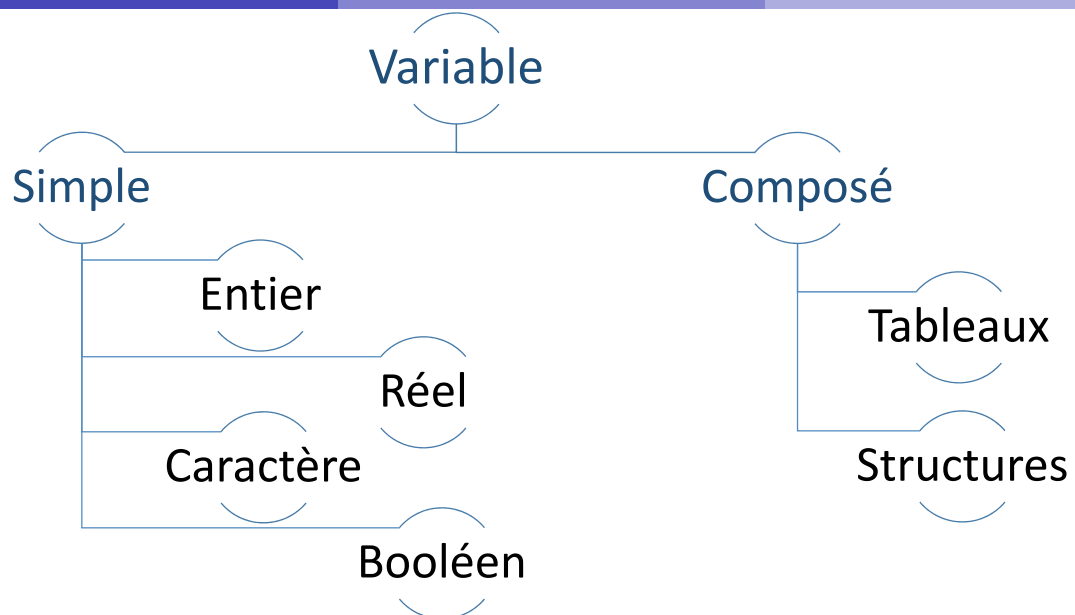
#### Partie II : Actions

1. Introduction
2. Action de lecture
3. Action d'écriture
4. Action d'affectation

25

## I. TYPES SIMPLES

### 1. Introduction



26

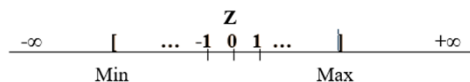
## I. TYPES SIMPLES

### 2. Type Entier

#### 1. Définition :

- Un type entier est défini dans un intervalle d'entiers fini, avec borne inférieure et borne supérieure prédéfinies, inclus dans l'ensemble (**Z**)
- Une variable de type entier peut être positif, négatif ou nul.

Ensemble de valeurs des nombres entières:



- Les valeurs de Min et Max varie en fonction des performances Matérielle de la machine d'exécution et du langage de programmation utilisé.
- **Exemple:** l'intervalle des variables de type entier en langage C entre:

[Min = -32768, ..... ,Max = 32767]

27

## I. TYPES SIMPLES

### 2. Type Entier

#### 2. Représentation algorithmique :

- La déclaration du type entier se fait dans la partie déclarative de l'algorithme (Var)
- Il faut indiquer devant le nom de la variable (Nomvar) son type (Entier).

**Var :**

Nomvar : **Entier** (Nomvar représente le nom de la variable qui sera utilisée dans l'algorithme)

- Il est possible de déclarer plusieurs variables dans la même ligne en les séparant par une virgule.

**Var :**

Nomvar1, Nomvar2, ....., NomvarN : **Entier**

Algorithme nominalgo

Const .....

Type .....

Var .....

**C'est ici !!!!**

Début

<action1>

28



## I. TYPES SIMPLES

### 2. Type Entier

#### 3. Opérations mathématiques sur le type Entier :

- Opérations arithmétiques.
- Opérations algorithmiques (lecture, l'écriture, etc).
- Opérations arithmétiques sur le type entier:

Opération	Symbole	Exemple
Addition	+	A=5, B=7 ; A+B donne 12
Soustraction	-	A=5, B=7 ; A-B donne -2
Multiplication	*	A=5, B=7 ; A* B donne 35
Division (entière)	DIV	A=5, B=7 ; A DIV B donne 0
Reste de la Division entière	MOD	A=5, B=7 ; A MOD B donne 5
Racine carrée	SQRT	X=2; SQRT(X) donne 1,41421
Valeur absolue	ABS	X=-3; ABS(x) donne 3
Opérateurs de comparaison	(<, <=, >, >=, <>, =)	

29

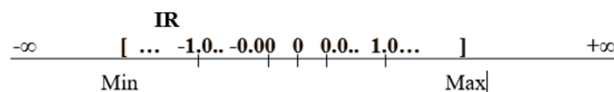
## I. TYPES SIMPLES

### 3. Type Réel

#### 1. Définition :

- Un type Réel est défini dans l'intervalle fini des réels, avec une borne inférieure et une borne supérieure prédéfinies, inclus dans l'ensemble ( $\mathbb{R}$ ).

→ Ensemble de valeurs des nombres réels  $\mathbb{R}$  :



- Les valeurs de Min et Max varient en fonction des performances matérielle de la machine d'exécution et du langage de programmation utilisé.

30

## I. TYPES SIMPLES

### 3. Type Réel

#### 2. Représentation algorithmique :

- La déclaration du type réel se fait dans la partie déclarative de l'algorithme (Var)
- Il faut indiquer devant le nom de la variable (Nomvar) son type (Réel).

**Var :**

Nomvar : **Réel** (Nomvar représente le nom de la variable qui sera utilisée dans l'algorithme)

- Il est possible de déclarer plusieurs variables dans la même ligne en les séparant par une virgule.

**Var :**

Nomvar1, Nomvar2, ....., NomvarN : **Réel**

Algorithme nomalgo  
Const .....  
Type .....  
Var .....

**C'est ici !!!!**

Début  
<action1>

31

## I. TYPES SIMPLES

### 3. Type Réel

#### 3. Ecriture d'un nombre réel :

- Une écriture standard:
  - ✓ Une partie **entière** et une partie **décimale**
  - ✓ Séparation par **un point**.
- Une écriture scientifique:

	Nombre Réel	Représentation
Écriture standard	5	5.0
	2,69	2.69
	-5,25	-5.25
Écriture scientifique	6,24	624 * e -2

32

## I. TYPES SIMPLES

### 4. Type Réel

#### 4. Opérations mathématiques sur le type Réel :

- Opérations arithmétiques
- Opérations algorithmiques (lecture, l'écriture, etc).
- Opérations arithmétiques:

Opération	Symbole	Exemple
Addition	+	A=5.5, B=7.0 ; A+B renvoie 12.5
Soustraction	-	A=5.5, B=7.0 ; A-B renvoie -2.5
Multiplication	*	A=5.5, B=7.0 ; A* B renvoie 38.5
Division normale	/	A=5.5, B=7.0 ; A / B renvoie 0.79
<b>Fonctions prédéfinies</b>		
Carré(X)	Retourne le carré de X	
RacineCarré(x)	Retourne la racine carré de X si X>=0, sinon elle retourne erreur	

33

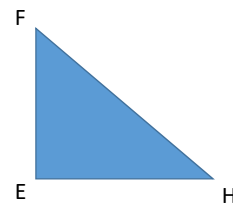
## I. TYPES SIMPLES

### 3. Type Réel

**Application :** Ecrire un algorithme qui permet de calculer le côté FH d'un triangle EFH rectangle en E.

$$FH^2 = EH^2 + EF^2$$

```
Algorithme Calcul_coteFH
Var: fh,eh,ef : réel
Début
    Écrire ("Quel est le longueur de EH :")
    Lire(eh)
    Écrire (" 'Quel est le longueur de EF :")
    Lire(ef)
    fh ← RacineCarré(carre(eh)+carre(ef))
    Écrire(fh)
Fin
```



34

## I. TYPES SIMPLES

### 4. Type Caractère

#### 1. Définition :

- Le type **Caractère** est ensemble de caractères comportant :
  - Les 26 lettres alphabétiques en majuscules ('A' jusqu'à 'Z') et en minuscules ('a' jusqu'à 'z')
  - Les 10 chiffres ('0' jusqu'à '9').
  - Les caractères spéciaux.

#### Remarque :

- Chaque caractère est délimitée par deux apostrophes ' ' : 'A', 'a', '+', '9' '@'....

35

## I. TYPES SIMPLES

### 4. Type Caractère

#### 2. Représentation algorithmique :

- La déclaration du type caractère se fait dans la partie déclarative de l'algorithme (Var)
- Il faut indiquer devant le nom de la variable (Nomvar) son type (Caractère).

**Var :**

Nomvar : **Caractère** (Nomvar étant la variable qui sera utilisée dans l'algorithme)

- Il est possible de déclarer plusieurs variables dans la même ligne en les séparant par une virgule.

**Var :**

Nomvar1, Nomvar2, ....., NomvarN : **Caractère**

Algorithme nomalgo

Const .....

Type .....

Var .....

**C'est ici !!!!**

Début

<action1>

36

## I. TYPES SIMPLES

### 4. Type Caractère

#### 3. Ordre d'apparition des Caractère :

- Les caractères sont ordonnés selon leurs codes ASCII.
- '0' < '1' < '2' < '3' < '4' < '5' < '6' < '7' < '8' < '9' < 'A' < 'B' ..... < 'Y' < 'Z' < 'a' < 'b' < ..... < 'y' < 'z'.

#### 4. Les Opérations sur le type Caractère :

- Les opérations sur les caractères : Suiv, Prec
- Les opérations algorithmiques : lecture, écriture, etc.
- Les opérations Suiv et Prec s'appliquent sur les variables ou des constantes de type caractère.

Opération	Symbole	Exemple
Suivant	Suiv	Si A='C' alors Suiv(A) renvoie 'D'
Précédent	Prec	Si A='C' alors Prec(A) renvoie 'B'

37

## I. TYPES SIMPLES

### 5. Type Booléen ou Logique

#### 1. Définition :

- Un type Booléen ou logique est un ensemble de deux éléments dont la valeur de l'un est l'opposé de l'autre (**VRAI** et **FAUX**).

Nombres binaires	0 et 1
variable réponse	Oui et Non
variable logique	Vrai et Faux
l'état d'une composant	Allumé et éteint
l'état d'un objet	Ouvert et Fermé

38

## I. TYPES SIMPLES

### 5. Type Booléen ou Logique

#### 2. Représentation algorithmique :

- La déclaration du type Booléen se fait dans la partie déclarative de l'algorithme (Var)
- Il faut indiquer devant le nom de la variable (Nomvar) son type (Booléen).

**Var :**

Nomvar : **Booléen** (Nomvar étant la variable qui sera utilisée dans l'algorithme)

- Il est possible de déclarer plusieurs variable dans la même ligne en les séparant par une virgule.

**Var :**

Nomvar1, Nomvar2, ....., NomvarN : **Booléen**

**Algorithme** nomalgo

**Const** .....

**Type** .....

**Var** .....

**C'est ici !!!!**

**Début**

<action1>

39

## I. TYPES SIMPLES

### 5. Type Booléen ou Logique

#### 3. Opérateurs Logiques :

- Une variable de type booléen peut subir les opérateurs suivants :
  - **NON (NOT):** la négation.
  - **ET (AND):** la conjonction.
  - **OU (OR):** la disjonction.

X	Y	Non X	X Et Y	X Ou Y
Faux	Faux	Vrai	Faux	Faux
Faux	Vrai	Vrai	Faux	Vrai
Vrai	Faux	Faux	Faux	Vrai
Vrai	Vrai	Faux	Vrai	Vrai

40

## I. TYPES SIMPLES

### 5. Type Booléen ou Logique

#### lois de Morgan :

➤  $\text{NON } (X \text{ ET } Y) \iff \text{NON } X \text{ OU NON } Y$

➤  $\text{NON } (X \text{ OU } Y) \iff \text{NON } X \text{ ET NON } Y$

L'ordre de priorité des opérateurs logiques est comme suit :

- la négation **NON** est prioritaire à la conjonction **ET**.
- la conjonction **ET** est prioritaire à la disjonction **OU**.

#### Exemple :

$\text{NON } X \text{ ET } Y$  : on doit appliquer d'abord **NON X** puis la conjonction **ET**

41

## I. TYPES SIMPLES

### 5. Type Booléen ou Logique

#### Application:

Soient les propositions  $u, v, w$ . Simplifier l'expression suivante :  $(u \text{ ET } v) \text{ OU NON } v$ .

- ✓ 1.  $\text{NON } v$
- ✓ 2.  $(u \text{ ET } v)$
- ✓ 3.  $2 \text{ OU } 1$

#### Exemples

**Algorithme E1**  
Var:  
  V1, V2 : Booléen  
Début  
  V1 ← vrai  
  V2 ← Non(V1)  
Fin

**V2 =faux**

**Algorithme E2**  
Var:  
  M : Entier  
  Test : Booléen  
Début  
  M ← 50  
  Test=(M>50)  
Fin

**test = faux**

42

## II. ACTIONS SIMPLES

### 1. Introduction

- Parmi les actions qu'on peut appliquer dans un algorithme : l'affichage, la saisie et l'affectation.
- La saisie est une opération d'entrée qui permet de placer dans une case mémoire une valeur communiquée par l'utilisateur à l'ordinateur.
- Cette opération s'effectue à travers des périphériques d'entrée (clavier, souris, stylo optique, etc.) vers la mémoire centrale.
- Une action de saisie est représentée par le terme '**Lire**'. Elle peut concerner une variable ou plusieurs variables.

43

## II. ACTIONS SIMPLES

### 2. L'action de saisie

#### 2. Représentation algorithmique :

- Pour lire une variable l'action de saisie est noté comme suit : **Lire (Nomvariable)**
- La saisie des variables peut être appliquée :
  - Soit une seule lecture pour l'ensemble de variables en les séparant par une virgule:

**Lire (Nomvariable1, Nomvariable2, ....., NomvariableN).**

- Soit une lecture pour chaque variable :

**Lire (Nomvariable1)  
Lire (Nomvariable2)  
.....  
Lire (NomvariableN).**

44



## II. ACTIONS SIMPLES

### 2. L'action de saisie

#### 3. Remarques :

- L'exécution de cette action implique l'arrêt momentané de l'exécution de l'algorithme en attente d'une valeur donnée par l'utilisateur pour produire des résultats attendus, si la touche Entrée (Enter) a été frappée, l'exécution reprend.

- Il est **interdit** de :

Si  $PI=3.14$  déclarée en tant que **constante**, alors **Lire(PI)** est impossible.

- Lire les opérations arithmétiques ou logiques.

**Lire(a+b)** est impossible.

- Lire un message.

**Lire("Bonjour")** est impossible

45

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### 1. Définition :

- L'affichage permet à l'utilisateur d'afficher les résultats d'un programme sur le(s) périphérique(s) de sortie, tels que : écran, imprimante, etc.

#### 2. Représentation algorithmique :

- Une action d'affichage peut se faire sur une ou plusieurs variables ou sur des constantes.
- Une action d'affichage peut **regrouper** des variables, des constantes, des expressions et des messages.

46

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### 3. Affichage sur une ou plusieurs variables :

- Pour une variable l'action d'affichage est noté comme suit :

**Ecrire (Nomvariable)**

- Pour plusieurs variables, deux formats d'écriture peuvent être appliqués :

➤ Soit une écriture multiple :

**Ecrire (Nomvariable1, Nomvariable2, ....., NomvariableN)**

➤ Soit une écriture par variable :

**Ecrire (Nomvariable1)**

.....

**Ecrire (NomvariableN)**

47

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### Ecriture d'une constante :

- Cette opération consiste à afficher la valeur d'une constante.
- Ecrire(Valconstante) ou Ecrire(Nomconstante),

**Ecrire (Valconstante)**

#### Exemples :

**Ecrire("3.14") , Ecrire(PI) , ...**

ou bien

**Ecrire("3.14")**

.....

**Ecrire(PI)**

48

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

- Une expression est un ensemble de variables ou constantes reliées par des opérateurs, et équivalent à une seule valeur.

**Ecrire**(expression)

#### Exemples:

**Ecrire**( $a+b*c$ ) avec a, b et c des **entiers**

**Ecrire**(a ou b et c) avec a, b et c des **booléens**

49

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### • Application :

Écrire un algorithme qui permet de calculer la somme de deux entiers a et b entrés à partir du clavier.

**Algorithme Somme**  
**Var:** a,b,S : entier  
**Début**  
    Lire(a)  
    Lire(b)  
     $S \leftarrow a+b$   
    Écrire(S)  
**Fin**



**Algorithme Somme**  
**Var :** a,b : entier  
**Début**  
    Lire(a)  
    Lire(b)  
    Écrire(a+b)  
**Fin**

*Gain d'une variable et d'une instruction*

50

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

- Un message est une suite de caractère ayant un sens et délimité par deux apostrophes.

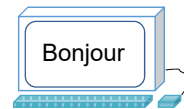
`Ecrire("message")`

#### Application :

Quelle est la différence entre `Ecrire("Bonjour")` et `Ecrire(Bonjour)` ?

**`Ecrire("Bonjour")`**

→ cette opération affichera à l'écran le mot Bonjour.



**Bonjour ← 200**

**`Ecrire(Bonjour)`**

→ cette opération affichera à l'écran le contenu de la variable Bonjour.



51

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### 4. Ecriture mixte:

- Dans certains programmes, l'affichage doit être très clair.

Par exemple, on veut afficher la ligne suivante : **Le Montant de la Facture est = 1250 DT.**

- Pour obtenir un tel affichage, on doit présenter l'action d'écriture comme suit :
- **`Ecrire("Le Montant de la Facture est = ", MNTFACT, "DT")`**

Où MNTFACT est la variable qui contient ce montant.

**`Écrire("message", Nomvariable, constante, "message",...)`**

52

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

#### 5. Application directe :

- On reprend l'algorithme permettant de faire la somme de deux entiers.

**Algorithme Somme**

**Var: a,b : entier**

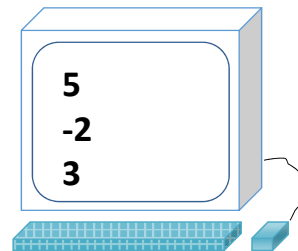
**Début**

**Lire(a)**

**Lire(b)**

**Écrire(a+b)**

**Fin**



- Avec un tel algorithme, l'affichage n'est pas clair surtout pour des utilisateurs qui n'ont pas d'idées sur le problème résolu ! Il faut donc améliorer la présentation de l'affichage.

53

## II. ACTIONS SIMPLES

### 3. L'action d'affichage

- Il suffit d'indiquer des messages. L'algorithme devient alors :

**Algorithme Somme**

**Var: a,b : entier**

**Début**

**Écrire ("Donner un entier a :")**

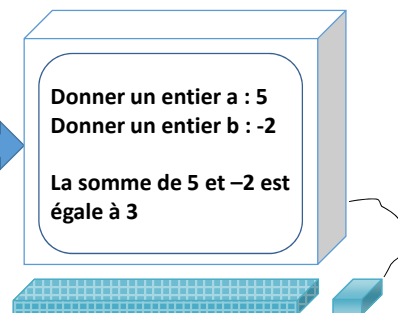
**Lire(a)**

**Écrire ("Donner un entier b :")**

**Lire(b)**

**Écrire("La somme de" ,a, "et" ,b, "est égale à", a+b)**

**Fin**



- Cet affichage est plus clair et l'utilisateur peut comprendre facilement que ce programme réalise la somme de deux entiers.

54

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### 1. Définition :

- L'opération d'affectation consiste à mettre une valeur dans une variable.
- L'affectation est schématisée par le symbole  $\leftarrow$

#### 2. Représentation Algorithmique :

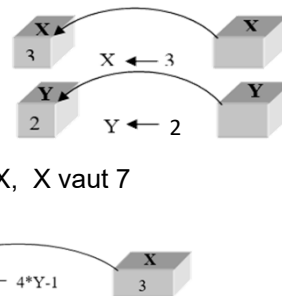
##### Exemples :

Nomvariable  $\leftarrow$  Valeur

$X \leftarrow 3$  -- on a affecté la valeur 3 à la variable X

$Y \leftarrow 2$  -- on a affecté la valeur 2 à la variable Y

$X \leftarrow 4 * Y - 1$  -- on a affecté l'expression  $4 * Y - 1$  à la variable X, X vaut 7



55

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Remarque:

- $A \leftarrow 5$  : cela veut dire mettre la valeur 5 dans la variable A
- On dit également, que:
  - A reçoit la valeur 5
  - on affecte 5 dans la variable A
  - on affecte à la variable A la valeur 5.
- $A \leftarrow B$  : cela veut dire mettre **le contenu** de la variable B dans la variable A.
- $A + B \leftarrow S$  : c'est une opération **impossible** ( $A + B$  n'est pas un nom de variable !!).
- Il faut distinguer entre  $A \leftarrow B$  et  $B \leftarrow A$
- L'affectation n'est pas commutative.

56

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Application Directe :

- Donner pour chaque opération la valeur contenue dans chaque variable :

Tableau : Exemples d'affectation

	<b>A</b>	<b>B</b>	<b>C</b>
<b>A ← 1</b>	1	--	--
<b>B ← A+3</b>	1	4	--
<b>A ← 3</b>	3	4	--
<b>B ← 5</b>	3	5	--
<b>C ← A+B</b>	3	5	8
<b>A ← 2</b>	2	5	8
<b>C ← B-A</b>	2	5	3

57

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exercices d'application:

##### Exercice N°1:

Écrire un algorithme qui permet de saisir deux entiers puis les permuter.

**Algorithme** permute2

**Var:** a, b, x : entier

**Début**

Lire(a)

Lire(b)

x ← a

a ← b

b ← x

Écrire(a)

Écrire(b)

**Fin**

58

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exercices d'application:

**Exercice N°2:** Écrire un algorithme qui permet de saisir quatre entiers puis les permute.

```
Algorithme permute2
Var: a,b,c,d,x : entier
Début
    Écrire("donner la valeur de a") Lire(a)
    Écrire("donner la valeur de b") Lire(b)
    Écrire("donner la valeur de c") Lire(c)
    Écrire("donner la valeur de d") Lire(d)
    x ← a
    a ← d
    d ← c
    c ← b
    b ← x
    Écrire(a)
    Écrire(b)
    Écrire(c)
    Écrire(d)
Fin
```

59

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Les Expressions Numériques:

- Une expression numérique peut contenir les éléments suivants :
  - des constantes
  - des variables
  - des opérateurs numériques : +, -, \*, /, DIV, MOD
  - des parenthèses
- Cette expression ne peut pas apparaître dans un programme **toute seule** ; elle doit être **affectée à une variable** ayant le même **type** que cette expression.

60



## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exemples d'expression :

- $A+B$
  - $3+2-B$
  - $4$
  - $(A+B)*C-2$
- Dans un programme, on trouve  $X \leftarrow (A+B)*C-2$  par exemple.
  - Si on veut évaluer le résultat de  $X$ , comment doit-on procéder ?
    - Etape1:  $(A+B)$
    - Etape 2: Etape1  $*C$
    - Etape 3: Etape2  $- 2$

61

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exemples d'expression :

- En fait, il y a une certaine priorité à respecter au niveau des opérateurs arithmétiques :
  - Priorité 1 : Fonctions
  - Priorité 2 : Parenthèses
  - Priorité 3 : Multiplications et Divisions
  - Priorité 4 : Additions et Soustractions

#### Remarque :

- Si dans une expression, on rencontre des opérateurs ayant la même priorité, on commencera par l'opérateur situé le plus à **gauche** dans l'expression.

62

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exercice d'application 1:

- Remplir le tableau suivant en respectant la priorité des opérateurs

**Tableau :** Exemples d'évaluation des expressions.

	A	B	C	X
A ← 1	1	--	--	--
B ← 2	1	2	--	--
C ← 3	1	2	3	--
X ← A+B*C	1	2	3	7
X ← (A+B)*C	1	2	3	9
X ← A+B/C	1	2	3	1.67
X ← (A+B)/C	1	2	3	1

63

## II. ACTIONS SIMPLES

### 4. L'action d'affectation

#### Exercices d'application 2:

- Écrire un algorithme qui affiche le quotient et le reste de la division entière de deux nombres entiers entrés au clavier.

**Algorithme** conv\_sec

**Var:** nbr1, nbr2, reste, quotient : Entier

**Début**

Ecrire ("donner deux nombres entière :")

Lire (nbr1,nbr2)

quotient ← nbr1 Div nbr2

reste ← nbr1 Mod nbr2

Ecrire("le quotient de la division de",nbr1,"avec",nbr2,"=", quotient,"et reste =",reste)

**Fin**

64

# Algorithmique et structures de données

## CHAPITRE 3 : LES STRUCTURES CONDITIONNELLES ET LES STRUCTURES RÉPÉTITIVES

Année universitaire 2020 – 2021

65

### LES STRUCTURES CONDITIONNELLES & LES STRUCTURES RÉPÉTITIVES

#### Plan du chapitre :

##### • Partie I: Les Structures Conditionnelles

1. Introduction
2. Structure conditionnelle à un choix
3. Structure conditionnelle à deux choix
4. Structure conditionnelle imbriquée
5. Structure à choix multiple

##### • Partie II: les structures répétitives

1. Définition
2. La boucle Pour
3. La boucle Répéter
4. La boucle Tant Que

66

## I. LES STRUCTURES CONDITIONNELLES

### 1. Introduction :

- Dans plusieurs situations, le problème nécessite l'étude de différentes situations qui est difficile de les traiter par les séquences d'actions simples.
- Etant donné ces diverses situations (et avant d'exécuter le programme) on ne sait pas quel cas de figure on doit exécuter, donc on doit prévoir tous les cas possibles.
- Les **structures conditionnelles** permettent de traiter des différents cas, en se basant sur des **conditions**:

SI la voie de gauche est libre  
ALORS je poursuis ma route  
SINON je m'arrête

67

## I. LES STRUCTURES CONDITIONNELLES

### 1. Introduction :

- Les structures conditionnelles sont capables de prendre en charge les différents traitements relatifs aux différentes conditions en exécutant exclusivement le traitement qui respecte une certaine **condition**.
- On distingue plusieurs formes de structures conditionnelles :
  - *Structure conditionnelle à un seul choix.*
  - *Structure conditionnelle à deux choix.*
  - *Structure conditionnelle imbriquée.*
  - *Structure conditionnelle à choix multiple.*

68

## I. LES STRUCTURES CONDITIONNELLES

### 2. Structure conditionnelle à un choix

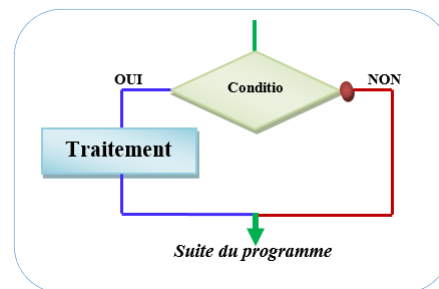
- **Définition :**

Pour cette structure, si la condition logique est **satisfaite** (vrai), le traitement sera exécuté.

Dans le cas contraire, rien ne se passe.

- **Syntaxe générale :**

```
Si (Condition_Satisfaite) Alors
    <Traitement>
Finsi
```



69

## I. LES STRUCTURES CONDITIONNELLES

### 2. Structure conditionnelle à un choix

- **Remarques:**

- Une condition est une expression logique (peut être présentée sous forme de comparaison en utilisant les opérateurs : <, >, <=, >=, =, <>).
- La condition peut être "simple", ou "composée".
- Dans le cas d'une condition composée, on doit utiliser un des opérateurs logiques (ET, OU, NON).
- Une action conditionnelle permet d'évaluer une condition. Si sa valeur est vrai, on exécute 1 ou plusieurs actions.

- **Exemple:** On veut demande d'afficher la moyenne d'un candidat âgé plus de 18 ans.

```
Si (Age > 18) Alors
    Ecrire (Moyenne)
Finsi
```

- **Exemple d'une condition composée:**

```
Si (n < 0 ET m < 0) OU (n > 0 ET m > 0) Alors
    Ecrire ("Le produit entre n et m est positif")
Finsi
```

70

## I. LES STRUCTURES CONDITIONNELLES

### 3. Structure conditionnelle à deux choix

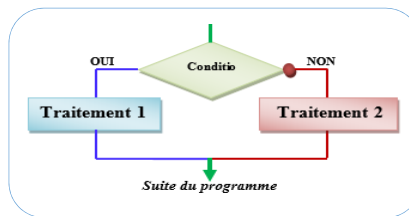
- **Définition :**

Cette structure fait appel au maximum à deux traitements.

- **Syntaxe générale :**

```
Si (Condition_Satisfaite) Alors
    <Traitement1>
Sinon
    <Traitement2>
Finsi
```

Chaque traitement peut être formé d'une ou plusieurs actions.



71

## I. LES STRUCTURES CONDITIONNELLES

### 3. Structure conditionnelle à deux choix

- **Exercices d'application :**

1) Ecrire un algorithme qui lit un nombre entier et qui affiche sa valeur absolue

```
Algorithme absolue
Var: n,a : entier
Début
    Ecrire("Donner un entier:")
    Lire(n)
    Si (n>0) alors
        a ← n
    Sinon
        a ← (- n)
    Finsi
    Ecrire("la valeur absolue de ", n ," est: ", a)
Fin
```

72

## I. LES STRUCTURES CONDITIONNELLES

### 3. Structure conditionnelle à deux choix

- Exercices d'application :

2) Écrire un algorithme qui permet de calculer  $x = (-1)^n + n$

```
Algorithme Calcul
Var: n,x : entier
Début
    Lire(n)
    Si ( n mod 2= 0 ) Alors
        x ← 1 + n
        Ecrire(x)
    Sinon
        x ← -1 + n
        Ecrire(x)
    Finsi
Fin
```

73

## I. LES STRUCTURES CONDITIONNELLES

### 4. Structure conditionnelle imbriquée :

- Définition :

Cette structure fait appel à plusieurs traitements (**plus que deux traitements**).

L'exécution du traitement  $T_i$  entraîne automatiquement le **non exécution** du traitement  $T_{i+1}$ .

- Syntaxe générale :

```
Si (Condition_Satisfaite 1) Alors
    <Traitement1>
Sinon
    Si (Condition_Satisfaite 2) Alors
        <Traitement2>
    Sinon
        Si (Condition_Satisfaite N) Alors
            <TraitementN>
        Sinon
            <TraitementN+1>
        Finsi
    Finsi
Finsi
```

74

## I. LES STRUCTURES CONDITIONNELLES

### 4. Structure conditionnelle imbriquée :

- **Exercice d'application :**

- **Exercice N°1 :** Écrire un algorithme qui permet de résoudre une équation de 1<sup>er</sup> degré :  $ax+b=0$ .

- **Solution:**

```
Algorithme Equation1
Var: a,b,x : Réel
Début
    Lire(a)
    Lire(b)
    Si ( a <> 0 ) Alors
        x ← -b/a
        Écrire(x)
    Sinon
        Si ( b <> 0 ) Alors
            Écrire("Pas de Solutions")
        Sinon
            Écrire("Une infinité de solutions")
        Finsi
    Finsi
Fin
```

75

## I. LES STRUCTURES CONDITIONNELLES

### 4. Structure conditionnelle imbriquée :

- **Tournage à la main:**

On vous donne l'algorithme ABC et on demande de faire son tournage à la main:

```
Algorithme ABC
Var: a,b,c : entier
Début
    Lire(a)
    Lire(b)
    Lire(c)
    Si ((a<5) ou (b>2) et (c>3)) Alors
        a ← 1
        Si (a-b>0) Alors
            c ← 0
        Finsi
        b ← b+c
    Sinon
        a ← c
    FinSi
    c ← b+c
    Écrire(a,b,c)
Fin
```

	a	B	c	Affichage
Cas 1	2	0	3	1 0 0
Cas 2	8	3	7	1 10 17
Cas 3	7	1	4	4 1 5

76



## I. LES STRUCTURES CONDITIONNELLES

### 5. Structure conditionnelle à choix multiple :

- **Définition :**

- Cette structure est une représentation simplifiée des conditions imbriquées.
- Elle permet d'exécuter un bloc d'actions parmi plusieurs traitements selon des conditions.
- Le choix du bloc d'actions se fait suivant la valeur d'un **sélecteur**.

- **Syntaxe générale :**

```
Selon Sélecteur Faire
  Val1 : <Traitement1>
  Val2 : <Traitement2>
  .....
  ValN : <TraitementN>
  Sinon <Traitement_erreur>
Fin Selon
```

77

## I. LES STRUCTURES CONDITIONNELLES

### 5. Structure conditionnelle à choix multiple :

- **Remarques :**

- Le sélecteur est un type énumérable, doit être un **entier** ou un **caractère** (mais pas chaîne).
- La valeur du sélecteur doit être connue (soit par **une lecture**, soit par **une affectation**) avant l'exécution du **Selon**.
- Val\_i est l'ensemble des valeurs qui sont séparées par des virgules ou l'intervalle des valeurs (v\_i .. v\_f)
- Les traitements relatifs aux valeurs peuvent comporter également **d'autres structures conditionnelles**.
- Si la valeur du sélecteur est différente des valeurs proposées le traitement **sinon du Selon** est exécuté.

78

## I. LES STRUCTURES CONDITIONNELLES

### 5. Structure conditionnelle à choix multiple :

#### • Exercice d'application 1:

Ecrire un algorithme qui lit deux réels, une opération arithmétique (+, -, \*, /) et affiche le résultat de l'opération.

```
Algorithme Arithmétique
Var:
Nb1, Nb2 : réel
Op : Caractère
Lire (Nb1, Nb2)
Lire(Op)
Selon Op faire
    '+' : Ecrire (Nb1 + Nb2)
    '-' : Ecrire (Nb1 - Nb2)
    '*' : Ecrire (Nb1 * Nb2)
    '/' :
        Si (Nb2 = 0) alors
            Ecrire "Opération indéterminée"
        Sinon
            Ecrire (Nb1 / Nb2)
        Finsi
    Sinon : Ecrire "Opération inexistante"
Fin Selon
Fin
```

79

## I. LES STRUCTURES CONDITIONNELLES

### 5. Structure conditionnelle à choix multiple :

#### • Exercice d'application 2:

Ecrire un algorithme qui lit la valeur d'une température de l'eau et d'afficher son état : liquide ( $1 \leq \text{température} \leq 100$ ), solide ( $-100 \leq \text{température} \leq 0$ ) ou gaz ( $101 \leq \text{température} \leq 200$ ).

```
Algorithme EtatLiqu
Var: t : entier
Début
    Lire(t)
    Selon (t) faire
        -100 .. 0 : Écrire("État Solide")
        1..100 : Écrire("État Liquide")
        101..200 : Écrire("État Gaz")
        Sinon Écrire("État Inexistant")
    Fin Selon
Fin
```

80

## II. LES STRUCTURES RÉPÉTITIVES

### 1. Définition :

- Les structures répétitives permettent d'exécuter plusieurs fois une même série d'instructions en fonction d'une **condition** donné.
- Il existe deux types de répétitions :
  - **L'itération** : Le nombre de répétitions d'un traitement est connu à l'avance:
    - La boucle **Pour**.
  - **La Répétition Conditionnelle** : Le nombre de répétitions d'un traitement n'est pas connu et dépend d'une condition booléenne :
    - La boucle **Tant que**.
    - La boucle **Répéter**.
- Le choix de la structure répétitive dépend du problème qu'on veut résoudre.

81

## II. LES STRUCTURES RÉPÉTITIVES

### 2. La boucle pour :

- Cette boucle itère le même traitement pour une plage de valeurs entières comprises entre une borne inférieure et une borne supérieure.
- La mise à jour étant automatique, l'arrêt du traitement de la boucle **Pour** se réalise lorsqu'on dépasse l'une des bornes.
- La boucle Pour est conseillée si le nombre d'itérations à faire **est connu à l'avance**.
- **Représentation algorithmique** :

```
Pour i de Val_initiale à Val_finale, (Pas = Val_pas) Faire  
    <Traitement>  
Fin Pour
```

- **i** : variable de type entier (en général) servant de compteur.
- **Val\_initiale** : valeur initiale que va prendre i.
- **Val\_finale** : valeur finale que prendra i.
- **Pas** : contient une valeur entière indiquant la valeur de l'incrément de i (mise à jour de i).

82

## II. LES STRUCTURES RÉPÉTITIVES

### 2. La boucle pour :

- **Exercice d'application :**

Ecrire l'algorithme qui permet de saisir les moyennes des N étudiants et de calculer la moyenne générale.

```
Algorithme CalculMoyGeneral
Var:
  N,i : entier
      MOY : réel (* la moyenne d'un étudiant *)
      MOYGen : réel (* la moyenne générale *)

Début
  Écrire("Ecrire le nombre des étudiants")
  Lire(N)
  MOYGen ← 0
  Pour i de 1 à N Faire
    lire (MOY)
    MOYGen ← MOYGen+MOY
  Fin Pour
  MOYGen ← MOYGen / N
  Écrire(MOYGen)
Fin
```

83

## II. LES STRUCTURES RÉPÉTITIVES

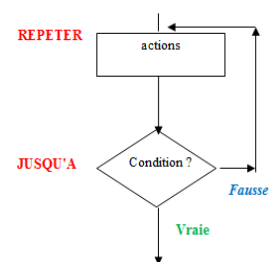
### 3. La boucle Répéter :

- **Définition :**

- Cette boucle permet de répéter un traitement 1 ou plusieurs fois quelle que soit la valeur de **la condition d'arrêt**.
- Quant la condition d'arrêt est atteinte, on sort de la boucle Répéter.
- Il faut initialiser correctement les variables de la condition d'arrêt et de les mettre à jour à la fin de **chaque itération** (pour pouvoir **reboucler**).

- **Représentation algorithmique :**

```
Répéter
<Traitement>
<mise_à_jour_condition_arrêt>
Jusqu'à (Condition_Arrêt_Atteinte)
```



84

## II. LES STRUCTURES RÉPÉTITIVES

### 3. La boucle Répéter :

- **Exercice d'application :**

Écrire un algorithme qui lit un nombre entier et s'arrête dès qu'il devienne supérieur à 100.

```
Algorithme LectureNombreEntier
Var:
  X : entier
Début
  Répéter
    écrire("Entrer une valeur de X")
    lire(X)
  Jusqu'à (X>100)
Fin
```

85

## II. LES STRUCTURES RÉPÉTITIVES

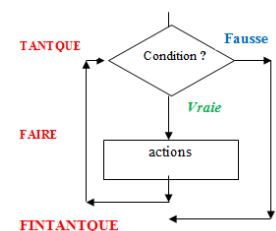
### 4. La boucle Tant que :

- **Définition :**

- La boucle Tant que permet de répéter l'exécution d'un bloc d'actions lorsque **la condition d'exécution** est **vérifiée** (0 ou plusieurs fois).
- Tant que la condition est vraie, le bloc d'actions est exécuté et on revient pour répéter la même chose. Dès que la condition **n'est plus vérifiée**, l'exécution s'arrête..
- Si **la condition d'exécution** reste inchangée (**non mise à jour**), on risque de reboucler indéfiniment et par conséquent le programme se bloque (**boucle infinie**).

- **Syntaxe :**

```
Initialisation des variables de la condition d'exécution
Tant que (Condition_vérifiée) Faire
  <Traitement>
  <mise_à_jour_condition_exécution>
Fin Tant que
.....
```



86

## II. LES STRUCTURES RÉPÉTITIVES

### 4. La boucle Tant que :

- **Exercice d'application :**

Ecrire un algorithme qui oblige l'utilisateur de saisir un entier positif.

```
Algorithme saisiePositif
Var:
  n : entier
Début
  n ← - 1
  Tant que (n<0)
    Ecrire("Donner une valeur positive de n: ")
    Lire(n)
  Fin tant que
  Ecrire("Bravo, vous avez saisi l'entier positif: ",n)
fin
```

87

## II. LES STRUCTURES RÉPÉTITIVES

### 4. La boucle Tant que :

- **Exercice d'application récapitulatif :**

En utilisant les 3 schémas itératifs, on vous demande d'écrire les algorithmes correspondants au calcul du factoriel d'un entier N donné.

```
Algorithme CalculFac
Variable N, i, factoriel : entier
Début
  lire(N)
  factoriel ← 1
  pour i de 1 à N faire
    factoriel ← factoriel * i
  Fin Pour
Fin
```

```
Algorithme CalculFac
Variable N, i, factoriel : entier
Début
  lire(N)
  factoriel ← 1
  i ← 1
  Répéter
    factoriel ← factoriel * i
    i ← i+1
  jusqu'à (i>N)
Fin
```

```
Algorithme CalculFac
Variable N, i, factoriel : entier
Début
  lire(N)
  factoriel ← 1
  i ← 1
  Tant que (i<=N) faire
    factoriel ← factoriel * i
    i ← i+1
  Fin Tant que
Fin
```

88

## II. LES STRUCTURES RÉPÉTITIVES

### 5. Remarques générales :

- Le <Traitement> d'une boucle peut contenir lui aussi une autre boucle. On l'appelle dans ce cas **des boucles imbriquées**.
- La différence entre la boucle **REPETER** et la boucle **TANT QUE** est que le traitement spécifié est exécuté au moins une fois dans le premier cas et peut être 0 ou plusieurs fois dans le deuxième cas.
- La *condition d'arrêt* de la boucle **REPETER** est l'inverse de la *condition d'exécution* de la boucle **TANTQUE**.

**Tableau Comparatif des Trois Schémas itératifs :**

Critère	Répéter	Tant que	Pour
Traitement	Au moins 1 fois	De 0 à N fois	De Val_initiale jusqu'à Val_finale
Initialisation	Oui avant Répéter	Oui avant Tant que	Oui : Val_initiale
Mise à jour variable	Oui après traitement	Oui après traitement	Oui, Automatique en fonction du Pas
Sortie de la boucle	Condition_arrêt atteinte	Condition_exécution non vérifiée	Compteur dépassant les bornes
Type de la condition	Expression logique avec ET, OU, etc.	Expression logique avec ET, OU, etc.	Une seule variable de type Entier

89



Université de Sousse  
Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Algorithmique et structures de données

## CHAPITRE 4 : LES SOUS-PROGRAMMES

Année universitaire 2020 – 2021

90

## LES SOUS-PROGRAMMES

### Plan du chapitre :

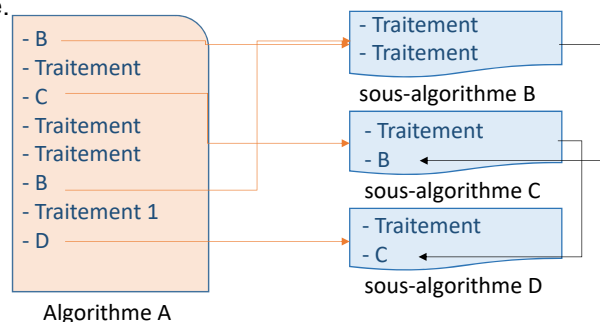
1. Introduction
2. La portée des variables
3. Les fonctions
4. Les procédures

91

## LES SOUS-PROGRAMMES

### 1. Introduction :

- Les sous-programmes sont des moyens qui offre aux programmeurs l'abstraction d'un problème à travers une séquence d'instructions.
- Un sous-programme est une structure algorithmique permettant de considérer une suite de traitements comme une entité indépendante appelée à partir du corps de l'algorithme principal ou d'un autre sous-algorithme.



Exemple l'algorithme A fait référence aux sous-algorithmes B, C et D.

92



## LES SOUS-PROGRAMMES

### 1. Introduction :

- Un sous-programme (*fonction ou procédure*) à la même structure qu'un programme, les seules différences importantes sont les notions de paramètres et de variables locales.
- Les sous-algorithmes présentent trois avantages :
  - Éviter la duplication des actions effectuant un traitement donné lorsque ce traitement est nécessaire en plusieurs endroits d'un même algorithme (ex : appel de B deux fois dans A).
  - Réutiliser des parties de l'algorithme déjà testées dans d'autres algorithmes (appel de C une fois par D, et une autre fois par A).
  - Améliorer la clarté des algorithmes ; il s'agit de décomposer chaque problème en petits sous-problèmes qui pourraient être développées et modifiées indépendamment (chaque sous-algorithme doit traiter un sous-problème donné).

93

## LES SOUS-PROGRAMMES

### 1. Introduction :

#### Exemple :

- On considère le sous programme (la procédure) **Affichage** et le **programme principal** suivants :

**Procédure Affichage** ( t:entier, n:entier)

**Var:**

c : entier

**DEBUT**

t ← t-1

n ← n-1

c ← t + n

Ecrire (c)

**Fin Affichage**

**ALGORITHME** Afficher

**Var** : a,b : entier

**DEBUT**

Lire(a)

Lire(b)

Affichage(a,b)

Ecrire(a,b)

**FIN**

94

## LES SOUS-PROGRAMMES

### 1. Introduction :

- En appelant le sous-programme `affichage(a,b)`, les valeurs de `a` et de `b` vont-elles changer après l'exécution de ce sous-programme ?
- Selon le comportement des variables lors de l'appel.
- Le comportement d'une variable dans un sous-programme, est appelé le mode de passage du paramètre. On compare deux types de passage de paramètres:
  - ✓ Un mode de passage par valeur.
  - ✓ Un mode de passage par variable.

95

## LES SOUS-PROGRAMMES

### 1. Introduction :

#### ➤ Le mode de passage par valeur :

- Les paramètres ne sont pas précédés d'aucun mot clé.
- La communication de valeurs se fait dans un seul sens.

*Programme appelant* → *Programme appelé*

#### ➤ Le mode de passage par variable:

- Les paramètres sont précédés par le mot clé `VAR` dans leur déclaration.
- Toute modification de la valeur du paramètre entraîne la modification du paramètre qui lui correspond.
- La communication de valeurs se fait dans les deux sens:

*Programme appelant* ↔ *Programme appelé*

96

## LES SOUS-PROGRAMMES

### 2. La portée des variables:

- L'appel d'un sous-algorithme nécessite une précision des variables à utiliser dans le sous-algorithme **appelé**. Ces variables sont classées en quatre types différents :

#### 1. Variable Locale :

- Une variable locale a une portée qui se limite dans le programme ou le sous-programme dans lequel est définie. Elle ne peut être ni vue, ni utilisée, par les **autres** sous-programmes.

```
SOUS-ALGORITHME Code_Ascii  
VAR : CA : entier, CC : car  
DEBUT  
  Lire (CC)  
  CA ← ORD(CC)  
  Ecrire (CA)  
FIN Code_Ascii
```

Les variables **CA** et **CC** déclarées dans le bloc VAR sont des variables locales au sous-algorithme : elles sont allouées dans la mémoire au début du sous-algorithme et libérées à sa fin.

97

## LES FONCTIONS ET LES PROCEDURES

### 2. La portée des variables :

#### 2. Variable Globale :

- Une variable globale a une portée qui s'étend sur la structure générale du programme. Elle est reconnue dans tout le programme. Par défaut, une variable est définie comme locale.
- Une variable globale est précédée par le terme **GLOBAL** lors de sa déclaration.
- Dans l'exemple du code ASCII, si on suppose que le caractère à traiter est déclaré dans l'algorithme principal comme une variable **globale**, on aura alors l'algorithme et le sous-algorithme suivants :

```
SOUS-ALGORITHME Code_Ascii  
VAR: CA : entier  
DEBUT  
  CA ← ORD (CC)  
  Ecrire ("le code ASCII de" ,CC, "est", CA)  
FIN Code_Ascii
```

```
ALGORITHME Caractère  
GLOBAL CC : Car  
DEBUT  
  Lire (CC)  
  Code_Ascii  
FIN
```

98

## LES FONCTIONS ET LES PROCEDURES

### 2. La portée des variables :

#### 3. Variable formel :

- C'est une variable déclarée lors de la définition d'un sous-algorithme.
- Ce type est utilisé pour la communication entre le sous-algorithme appelant et le sous-algorithme appelé.

#### Exemple :

- Dans l'exemple du code ASCII, la variable CC est utilisé comme un paramètre formel pour le sous-algorithme Code\_Ascii :

```
SOUS-ALGORITHME Code_Ascii(CC : Car )  
VAR CA : entier  
DEBUT  
CA ← ORD (CC)  
Ecrire ("le code ASCII de ",CC, "est", CA)  
FIN Code_Ascii
```

99

## LES FONCTIONS ET LES PROCEDURES

### 2. La portée des variables :

#### 4. Variable Réel :

- Les variables réels sont déclarées dans un sous-algorithme appelant.
- Ces variable permet de communiquer les valeurs nécessaires à l'exécution d'un sous-algorithme appelé.
- Les paramètres réels d'un sous-algorithme appelant et les paramètres formels du sous-algorithme appelé doivent être de mêmes types respectivement, listés dans le même ordre et de même nombre.

Exemple: Dans l'exemple du code ASCII, l'algorithme principal peut appeler le sous-algorithme plus qu'une fois, à chaque fois avec une nouvelle variable.

```
ALGORITHME Caractère  
VAR CC1, CC2 : Car  
DEBUT  
Lire (CC1, CC2)  
Code_Ascii (CC1)  
Code_Ascii (CC2)  
FIN
```

100

## LES SOUS-PROGRAMMES

### 3. Les fonctions

#### 1. Définition :

- Une fonction est un sous-programme, qui fournit un résultat unique de type simple.
- Dans une fonction, les instructions qui produisent le résultat doivent être affecté au nom de la fonction.

#### 2. Représentation algorithmique :

```
FONCTION Nom_fonction( {Liste de variables formels} ) : type_de_retour
VAR: {déclaration des variables locales}
DEBUT
    {Ensemble d'instructions}
    Nom_fonction ← résultat
FIN Nom_fonction
```

101

## LES SOUS-PROGRAMMES

### 3. Les fonctions

#### Exemple 1:

Écrire un sous-programme (fonction) qui permet de calculer la valeur absolue d'un entier N.

#### Solution :

```
FONCTION Val_abs( N:entier ) : entier
VAR:
a: entier
DEBUT
    Si (N>0) Alors
        a ← N
    Sinon
        a ← - N
    Finsi
    Val_abs ← a
FIN Val_abs
```

102

## LES SOUS-PROGRAMMES

### 3. Les fonctions

#### Exemple 2:

Écrire un sous-programme (fonction) qui permet de calculer la moyenne de deux réels x et y.

#### Solution :

```
FONCTION Moyenne( x:réel, y:réel ) : réel
VAR:
a: entier
DEBUT
    Moyenne ← (x+y)/2
FIN Moyenne
```

103

## LES SOUS-PROGRAMMES

### 3. Les fonctions

#### 3. Appel de la fonction :

- Il existe deux façons d'appel :
  - Dans une affectation : `Variable ← Nom_fonction( {paramètres réels} )`
  - Dans un affichage : `Ecrire ("le resultat est : ", Nom_fonction ( {paramètres réels} ))`
- Variable et Nom\_fonction doivent avoir le même type.
- Les variables utilisées dans le programme appelant sont appelées variables formels.
- L'ordre des variables formels doit être respecté lors de l'appel.

104

## LES SOUS-PROGRAMMES

### 4. Les procédures

#### 1. Définition :

- Une procédure est un sous-programme qui permet la résolution d'un problème précis et qui peut avoir *plusieurs résultats*.

#### 2. Représentation algorithmique :

```
Procédure Nom_procedure (Var1 : type1, ... VarN : typeN, Res1 : Type1, ..., ResN : typeN)
Début
    Traitements
Fin
```

105

## LES SOUS-PROGRAMMES

### 4. Les procédures

#### Exemple :

- Écrire un sous-programme (procédure) qui permet d'afficher la somme de deux entiers a et b.

#### Solution :

```
Procédure Affiche_somme (a : entier, b:entier)
Var:
S: entier
Début
    S ← a+b
    Écrire(S)
Fin Affiche_somme
```

106

## LES SOUS-PROGRAMMES

### 4. Les procédures

#### 3. Appel de la procédure :

Une procédure est appelée par son nom et ses paramètres effectifs, dans le programme appelant.

Nom\_procedure (var1, var2, ..., varN)

#### Exemple :

```
Procédure Somme ( t:entier, n:entier, S : entier)
DEBUT
  S ← t+n
  Ecrire (S)
Fin Somme
ALGORITHME Affiche
VAR : a,b,c : entier
DEBUT
  Écrire("Donner un entier a: "), Lire(a)
  Écrire("Donner un entier b: "), Lire(b)
  Somme(a,b,c)
  Ecrire(a,b,c)
FIN
```

107

## LES SOUS-PROGRAMMES

### 4. Les procédures

#### Exemple : Procédure sans paramètres.

```
Procédure message ()
Var:
Mess: chaine
Début
  Lire (Mess)
  Ecrire (Mess)
Fin message
```

108





# Algorithmique et structures de données

## CHAPITRE 5: LES CHÂÎNES DE CARACTÈRES

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

109

### LES CHÂÎNES DE CARACTÈRES

#### • Plan du chapitre :

- I. Définition
- II. Déclaration
- III. Opérations de manipulation
- IV. Les fonctions relatives aux chaînes de caractères
- V. Les procédures relatives aux chaînes de caractères

110

## I. Définition :

- Une chaîne de caractère est constitué d'une suite de caractères placés entre guillemets.
- Une chaîne de caractère peut être vide (""), de longueur 1 (un seul caractère), ou de longueur supérieure à 1 et inférieure à 255 caractères).

- Exemple :

Ch:chaîne

Ch ← "Bonjour"

Ch est une variable de type chaîne de caractères dont la longueur est égale à 7.

111

## II. Déclaration :

- Une chaîne de caractères peut être de taille prédéfinie (au maximum 255 caractères).

Var: Ch1, Ch2 ,..., ChN : chaîne

- Une chaîne de caractères peut être de taille limité

Var: Ch1, Ch2 ,..., ChN : chaîne [taille]

Avec  $0 < \text{Taille} \leq 255$

112

### III. Opérations de manipulation

- Les opérations permises sur les chaînes de caractères sont les suivantes :
- Lecture / Ecriture / Affectation : Ces actions peuvent être effectuées pour toute la chaîne ou bien élément par élément.
- Exemple

Lire (ch) : pour lire toute la chaîne ch.

Lire (ch[i]) : pour lire le caractère numéro i de la chaîne ch.

113

### IV. Les fonctions relatives aux chaînes de caractères

- **LONG** : la fonction LONG retourne la longueur de la chaîne (le nombre de caractères). Le résultat de cette fonction est de type entier.

#### **LONG (chaîne)**

- La comparaison des chaînes de caractères se fait avec les opérateurs de comparaisons (=, <>, <, <=, >, >=).
- La comparaison de deux chaînes se base sur l'ordre lexicographique en comparant les caractères de même position des deux chaînes.
- La comparaison de deux chaînes se fait du gauche à droite, caractère par caractère.

#### Exemples

"BALLES" < "BALLON" < "BAR" < "Bar" < "bar"

"BAL" < "BALLES"

114

#### IV. Les fonctions relatives aux chaînes de caractères

- **CONCAT** : La fonction CONCAT permet de former une seule chaîne à partir de deux chaînes placées l'une à côté de l'autre selon l'ordre de leur apparition.
- Le résultat de cette fonction est de type chaîne de taille maximale 255 caractères.

**CONCAT (ch1, ch2)**

Exemples

**CONCAT ("AUTO", "MOBILE") retourne "AUTOMOBILE"**

115

#### IV. Les fonctions relatives aux chaînes de caractères

- **SCH** : La fonction SCH extrait une sous chaîne à partir d'une autre chaîne.
- **SCH (ch, P, L)**: permet d'extraire, à partir d'une chaîne *ch*, une sous chaîne de longueur *L* à partir de la position *P*.

- Exemple

**SC1 ← SCH ("AUTOMOBILE", 1, 4)**: SC1 aura comme valeur "AUTO"

**SC2 ← SCH ("AUTOMOBILE", 5, 10)**: SC2 aura comme valeur "MOBILE"

- **POS** : permet de chercher la position de la première occurrence d'une chaîne dans une autre.
- **POS (ch, ch1)** : Retourne la position de la première occurrence de la chaîne *ch1* dans la chaîne *ch*.
- Si *ch1* existe dans *ch* cette fonction renvoie une valeur *n* ( $0 < n \leq \text{LONG}(ch)$ ), si *ch1* n'existe pas dans *ch* cette fonction renvoie 0.

116

## V. Les procédures relatives aux chaînes de caractères

Syntaxe	Rôle	Type	
		Paramètres d'entrée	Résultat
Efface (Ch, p, n)	Supprime n caractères de Ch à partir de la position P.	Chaîne, Entier, Entier	Chaîne
Insère (Ch1, Ch2, p)	Insère une chaîne Ch1 dans une autre Ch2 à la position p.	Chaîne/Caractère, Chaîne, Entier	Chaîne
Convch (n, Ch)	Convertit une valeur numérique n en une chaîne Ch.	Numérique, Chaîne	Chaîne
Valeur (Ch, n, err)	Convertit une chaîne de caractères Ch en une valeur numérique N.	Chaîne/Caractère, Numérique, Entier	Numérique, Entier

117



Université de Sousse  
Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Algorithmique et structures de données

## CHAPITRE 6 : LE TYPE TABLEAU

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

118

## LE TYPE TABLEAU

### • Plan du chapitre :

- I. Définition
- II. Les tableaux à une dimension (Vecteur)
- III. Techniques utilisées avec les tableaux
- IV. Les tableaux a deux dimensions (Matrice)
- V. Méthodes de tri

119

### I. Introduction :

- Un tableau est une structure de données qui permet de regrouper un nombre fini d'éléments **de même type**.
- Il y a deux types de tableaux :
  - Les Vecteurs.

--	--	--	--	--	--	--	--

- Les Matrices.


120

## I. Introduction :

### Remarques :

- Un tableau est caractérisé par :
  - Son nom.
  - Sa taille (Nombre de cases).
  - Ses éléments
- Un tableau est formé par un nombre fini d'élément.
- L'accès à un élément du tableau se fait à l'aide d'un indice.

121

## II. Les tableaux à une dimension :

### 1. Définition :

- Un vecteur permet d'organiser des éléments de même type.
- Il regroupe ces éléments dans une structure fixe et permet d'accéder à chaque élément par son indice.

➤ Tableau A :

2	40	90	51	98	37	53	29
---	----	----	----	----	----	----	----

122

## II. Les tableaux à une dimension :

### 2. Représentation algorithmique :

**Type:**

Nomtableau = Tableau [début .. fin] de type\_élément

**Var:**

N\_var : Nomtableau  
Indice : entier

**Exemple :**

- Déclaration d'un tableau T d'entier

**Type:**

Tab = Tableau [1..40] de entier

**Var:**

T : Tab  
i : entier (\* indice du tableau T \*)

123

## II. Les tableaux à une dimension :

- Exemple de tableau de réel: chaque élément du tableau contient une valeur de type réel.
- Le type de l'indice du tableau doit être de type entier.

	01	02	03	04	05	06	07	08	09	10
T	15.5	09.2	09.7	11.0	10.5					18.6

- Le premier élément du vecteur T contient la valeur 15.5; on écrit  $T[1] \leftarrow 15.5$
- Le deuxième élément du vecteur T contient la valeur 08.5; on écrit  $T[2] \leftarrow 08.5$
- Le  $i^{\text{ème}}$  élément du vecteur T est noté  $T[i]$  contient une valeur de type réel.
- $T[i]$  représente le contenu de la  $i^{\text{ème}}$  case du tableau T.

124



## II. Les tableaux à une dimension :

### Remarques :

- Le type de l'indice du tableau doit être de type entier.
- La taille du tableau doit être fini et déclaré.
- Un tableau, s'il est rempli, ne doit pas faire l'objet d'une lecture supplémentaire sauf s'il s'agit d'une opération de mise à jour où on doit modifier le contenu des éléments du tableau.
- A supprimé

125

## II. Les tableaux à une dimension :

### 3 Opérations de base sur un tableau :

- Afin de représenter les différentes opérations sur le tableau, on prend l'exemple d'un tableau d'entiers.

#### a. Initialisation d'un Tableau :

```
Algorithme Initialisation_Tableau  
Const:  
N=30  
Type:  
TableauEtudiant = Tableau [1..N] de entier  
Var:  
Tetud : TableauEtudiant  
i: entier  
Début  
    Pour i de 1 à N faire  
        T[i] ← 0  
    Finpour  
Fin
```

126

## II. Les tableaux à une dimension :

### b. Remplir un Tableau :

```
Algorithme Remplir_Tableau
Const:
N=30
Type:
TableauEtudiant = Tableau [1..N] de entier
Var:
Tetud : TableauEtudiant
i: entier
Début
    Pour i de 1 à N faire
        Lire (T[i])
    Finpour
Fin
```

Lire(T[i]) permet d'entrer une valeur à partir du clavier dans le i<sup>ème</sup> élément du tableau.

127

## II. Les tableaux à une dimension :

### c. Affichage des éléments d'un tableau :

- Pour afficher les éléments d'un tableau, on utilise l'algorithme suivant:

```
Algorithme Affichage_Tableau
Const:
N=30
Type:
TableauEtudiant = Tableau [1..N] de entier
Var:
Tetud : TableauEtudiant
i: entier
Début
    Pour i de 1 à N faire
        Écrire (T[i])
    Finpour
Fin
```

128

## II. Les tableaux à une dimension :

### d. Addition des éléments de deux tableaux :

- Etant donnée de deux tableaux T1 et T2 et on veut calculer la somme élément par élément.
- **Condition nécessaire** : T1 et T2 de même taille.

```
Algorithme Addition_Tableaux
Const:
N=60
Type:
TableauEnt = Tableau [1..N] de entier
Var:
T1, T2, T3 : TableauEnt
i: entier
Début
    Pour i de 1 à N faire
        T3[i] ← T1[i]+T2[i]
    Finpour
Fin
```

129

## II. Les tableaux à une dimension :

### e. Multiplication des éléments de deux tableaux :

```
Algorithme Multiplication_Tableaux
Const:
N=60
Type:
TableauEnt = Tableau [1..N] de entier
Var:
Tab1, Tab2, Tab3 : TableauEnt
i: entier
Début
    Pour i de 1 à N faire
        Tab3[i] ← Tab1[i]*Tab2[i]
    Finpour
Fin
```

130

### III. Recherches utilisées avec les tableaux :

#### **1. Méthodes de recherche dans un tableau :**

##### a. Introduction :

- La recherche séquentielle.
- La recherche dichotomique : Une recherche plus optimisée que la recherche séquentielle.

131

### III. Techniques utilisées avec les tableaux :

#### **1. Recherche dans un tableau :**

##### b. Recherche Séquentielle :

```
Algorithme Recherche_Séquentielle
Const:
N=40
Type:
TableauEnt = Tableau [1..N] de entier
Var:
Tab : TableauEnt
i, X: entier
B : Booléen
Début
  Lire (X)
  i ← 1
  B ← Faux
  Tant que (i ≤ N) et (B = Faux) faire
    Si (T[i] <> X) alors
      i ← i + 1
    Sinon
      B ← vrai
  Finsi
  Fin tant que
  Si (B = vrai) alors
    Ecrire (X, "appartient à T")
  Sinon
    Ecrire (X, "ne se trouve pas dans T")
  Finsi
Fin
```

132

### III. Techniques utilisées avec les tableaux :

#### 1. Recherche dans un tableau :

##### c. Recherche Dichotomique :

La recherche dichotomique nécessite que le tableau entré soit Rempli et Trié.

→ Cette recherche réduit le temps d'exécution des traitements

D'abord on doit tester si la valeur rechercher (X) est égale, inférieure ou bien supérieure à la valeur du milieu.

➤ Si X est inférieur à cette valeur.

la recherche continue dans la partie gauche du tableau avec la modification du borne supérieure,

➤ Sinon la recherche continue dans la partie droite du tableau avec modification de la valeur de la borne inférieure.

- L'algorithme s'arrête lorsque X est égale à la valeur du milieu ou quand le borne supérieure soit supérieur au borne inférieure du tableau.

133

### III. Techniques utilisées avec les tableaux :

#### 1. Recherche dans un tableau :

##### c. Recherche Dichotomique :

```
Algorithme Recherche_Dichotomique
Const:
N=40
Type:
TableauEnt = Tableau [1..N] de entier
Var:
Tab : TableauEnt
i,inf, sup, milieu, X: entier
Début
  Lire (X)
  inf ← 1
  sup ← N
  Répéter
    milieu ← (inf + sup) div 2
    Si (X < T(mil)) alors
      sup ← mil - 1
    Sinon
      inf ← mil + 1
    Finsi
  Jusqu'à (X = T[mil] ou (inf > sup)
Fin
```

134

### III. Techniques utilisées avec les tableaux :

#### 2. Comparaison de deux Tableaux :

- On doit s'assurer que les deux tableaux ont le même type et de même taille.
- La comparaison se fait élément par élément.

```
Algorithme Comparaison
Const:
N=10
Type:
TableauEnt = Tableau [1..N] de entier
Var:
Tab1, Tab2 : TableauEnt
i : entier
B: Booléen
Début
i ← 1
B ← vrai
Tant que ( i <= N ) et ( B = vrai ) faire
    Si (Tab1[i] <> Tab2(i)) alors
        B ← faux
    Sinon
        i ← i + 1
Finsi
Fin Tant que
Si (B= vrai) alors
    Ecrire ("Tab1 et tab2 sont différents")
Finsi
Fin
```

135

### IV. Les Tableaux à deux dimensions : Les Matrices

#### 1. Définition:

- Un matrice est une structure de données ayant le même type.
- Cette structure est caractérisée par son nom, nombre de lignes et nombre de colonnes.

#### 2. Représentation Algorithmique :

```
Const:
Nbr_lig = 50
Nbr_col=30
Type:
Mat = Tableau [1..Nbr_lig, 1..Nbr_col] de type_élément
Var:
M: Mat
i,j:entier
```

136

## IV. Les Tableaux à deux dimensions : Les Matrices

### 3. Opérations de base sur une Matrice :

- On suppose que le Type\_Elément de la matrice est Entier.

#### a. Initialiser une matrice :

```
Pour i de 1 à nbre_ligne faire
  Pour j de 1 à nbre_colonne faire
    Mat[i,j] ← 0
  Finpour
Finpour
```

#### b. Remplir une matrice :

```
Pour i de 1 à nbre_ligne faire
  Pour j de 1 à nbre_colonne faire
    Lire (Mat[i,j])
  Finpour
Finpour
```

137

## IV. Les Tableaux à deux dimensions : Les Matrices

#### c. Parcourir une matrice :

- Afficher tous les éléments de la matrice Mat:

```
Pour i de 1 à nbre_ligne faire
  Pour j de 1 à nbre_colonne faire
    Ecrire (Mat[i,j])
  Finpour
Finpour
```

#### d. Additionner deux matrices :

**Condition nécessaire:** les deux matrices Mat1 et Mat2 doivent avoir le même type et la même taille.

```
Pour i de 1 à nbre_ligne faire
  Pour j de 1 à nbre_colonne faire
    Mat3[i,j] ← Mat1[i,j]+Mat2[i,j]
  Finpour
Finpour
```

138

## V. Méthodes de tri

### **PROBLÈME :**

- Etant donné un tableau d'entiers T, on veut trier T dans l'ordre croissant
- Un **algorithme de tri** permet d'organiser une suite d'éléments selon un ordre déterminé.
- Les éléments à trier font partie d'un ensemble muni d'une relation d'ordre.

139

## V. Méthodes de tri

### **1. TRI À BULLES**

#### **PRINCIPE :**

- Comparaison 2 à 2 des éléments adjacents
- Permutation des éléments s'ils ne sont pas ordonnés.
- Le programme s'arrête lorsqu'on parcourt le tableau sans faire de permutation.
- Comme les bulles, les plus grands éléments remontent en fin de liste.

140



## V. Méthodes de tri

### 1. TRI À BULLES

#### PRINCIPE :

- Le principe du tri à bulles consiste à trier un tableau en ordre croissant, **tout élément est plus petit que celui qui le suit.**
- Chaque élément d'un tableau, et comparer avec l'élément qui le suit.
- Si l'élément (i+1) inférieur à l'élément (i) on permute ces deux éléments.
- Et on recommence jusqu'à ce que l'on n'ait aucune modification à effectuer.

141

## V. Méthodes de tri

### 1. TRI À BULLES

#### ALGORITHME :

```
Procédure TriBulles (var T: tab; N: entier)
Var:
i : entier
Changement: booléen
Début
    Changement ← Vrai
    TantQue (Changement = Vrai) Faire
        Changement ← Faux
        Pour i de 1 à (n -1) Faire
            Si (T[i] > T[i + 1]) Alors
                Permuter (T[i], T[i+1])
            Changement ← Vrai
        FinSi
    Finpour
    Fin TantQue
Fin TriBulles
```

Exemple :

45	122	12	3	21	78	64	53	89	28	84	46
----	-----	----	---	----	----	----	----	----	----	----	----

142

## V. Méthodes de tri

### 2. TRI PAR SÉLECTION

#### **PRINCIPE :**

- Le principe du tri par sélection consiste à chercher le plus petit élément du tableau et le mettre dans la première case du tableau.
- Puis, on commence à partir du deuxième élément et on cherche le plus petit élément du tableau entre la position 2 et n-1.
- Exemple:

45	122	12	3	21	78	64	53	89	28	84	46
----	-----	----	---	----	----	----	----	----	----	----	----

3	122	12	45	21	78	64	53	89	28	84	46
---	-----	----	----	----	----	----	----	----	----	----	----

3	12	122	45	21	78	64	53	89	28	84	46
---	----	-----	----	----	----	----	----	----	----	----	----

143

## V. Méthodes de tri

### 2. TRI PAR SÉLECTION

#### **PRINCIPE :**

- On recommence à chercher le plus petit élément à partir du troisième (puisque les deux premiers sont maintenant bien placés), et on le place correctement, en l'échangeant, ce qui donnera in fine :

3	12	21	45	122	78	64	53	89	28	84	46
---	----	----	----	-----	----	----	----	----	----	----	----

- Et cetera, et cetera, jusqu'à l'avant dernier.

144

## V. Méthodes de tri

### 2. TRI PAR SÉLECTION

#### ALGORITHME :

```

Procédure TriSelection (var T: tab; N:entier)
Var:
i,p : entier
Début
Pour i de 1 à N - 1 faire
    p ← posmin (T, i, N)
    Si (T[p]<T[i]) alors
        Permuter (T[p], T[i])
    Finsi
Finpour
Fin TriSelection
    
```

```

Fonction posmin (T: tab; d, f: entier): entier
Var:
p,i : entier
Début
p ← d
Pour i de d+1 à f faire
    si T[i] < T[p]
        p ← i
    finsi
Finpour
posmin ← p
Fin posmin
    
```

145

## V. Méthodes de tri

### 3. TRI PAR INSERTION

#### PRINCIPE:

- La liste étant trié jusqu'à l'élément j-1
- Insérer l'élément j à sa place parmi les j premiers éléments.

#### ALGORITHME:

```

Procédure TriInsertion (var T: tab; N:entier)
Var:
i , cle : entier
Début
Pour j de 2 à N faire
    cle ← T[j]
    i ← j-1
    Tant que (i>0 et T[i]> cle ) faire
        T[i+1] ← T[i]
        i ← i-1
    Fin Tant que
    T[i+1] ← cle
Finpour
Fin TriInsertion
    
```

Exemple :

45	122	12	3	21	78	64	53	89	28	84	46
----	-----	----	---	----	----	----	----	----	----	----	----

146



# Algorithmique et structures de données

## CHAPITRE 7: LES ENREGISTREMENTS

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

147

### LES ENREGISTREMENTS

#### Plan du chapitre :

1. Introduction
2. Déclaration
3. Manipulation
4. Tableaux d'enregistrements

148

## LES ENREGISTREMENTS

### 1. Introduction

- Les enregistrements ou structures sont des structures de données dont **les éléments peuvent être de type différent**.
- Un enregistrement contient des éléments appelés **champs**.
- Le type d'un enregistrement est appelé **type structuré**.
- Après la définition d'un type structuré (enregistrement), on peut déclarer une ou plusieurs variables de ce type.
- La déclaration des types structurés se fait dans la partie **Type** de l'algorithme principale.
- **Exemple:** les champs : nom, prénom, sexe, nombre d'enfant sont relatives à un employé d'une entreprise.

149

## LES ENREGISTREMENTS

### 2. Déclaration

#### • Syntaxe :

##### Type

```
nom_type = enregistrement
nom_champ1: type_champ1
...
nom_champn: type_champn
Fin
```

#### • Exemple:

##### Type

```
tpersonne = enregistrement
nom : chaîne
prénom : chaîne
âge : entier
Fin
```

150

## LES ENREGISTREMENTS

### 2. Déclaration

- La déclaration des variables de type enregistrement est similaire à la déclaration des variables de type simple.

- **Syntaxe**

Var:

nom\_variable : nom\_type

- **Exemple:**

```
TYPE
Etudiant = enregistrement
  Nom, Prénom : chaîne[30]
  Moyenne : réel
  Age : entier
Fin enregistrement

Var :
Etd : Etudiant
```

151

## LES ENREGISTREMENTS

### 3. Manipulation

- Les enregistrements sont manipulés à travers ses champs.
- L'accès aux champs d'un enregistrement se fait à travers son nom précédé par une **point**.

- **Exemple:** Nom\_variable de type enregistrement . nom\_champ

- Pour accéder au champ moyenne et nom de la variable Etd, on écrit:

Etd.moyenne : correspond à la moyenne de l'étudiant Etd.

Etd.nom : correspond au nom de l'étudiant Etd.

152

## LES ENREGISTREMENTS

### 3. Manipulation

- Le nom d'un champ est TOUJOURS précédé par le nom de l'enregistrement auquel il appartient.
- Les opérations (affectation, saisie, affichage,...) sont appliquées sur les champs d'un enregistrement.
- **Exemple 1:**
- Ecrire un programme (Manip\_personne) qui permet de saisir les données des personnes pers1 et pers2, puis affiche la différence d'âge entre ces deux personnes.

153

## LES ENREGISTREMENTS

### 3. Manipulation

#### Algorithme Manip\_personne

##### Type

tpersonne = enregistrement  
nom : chaîne  
prénom : chaîne  
âge : entier

##### Fin

##### Var:

pers1, pers2 : tpersonne

##### Début

```
Ecrire ("Entrez le nom puis l'âge de la personne 1" )
Lire (pers1.nom, pers1.age)
Ecrire("Entrez le nom puis l'âge de la personne 2")
Lire (pers2.nom, pers2.age)
Écrire ("La différence d'âge entre ", pers1.nom, " et ", pers2.nom, " est de ")
Si (pers1.age > pers2.age) alors
    Écrire (pers1.age – pers2.age, " ans ")
Sinon
    Écrire (pers2.age – pers1.age, " ans ")
finsi
```

##### Fin

154

## LES ENREGISTREMENTS

### 3. Manipulation

#### Passage d'un enregistrement en paramètre d'un sous-programme

- On peut passer un enregistrement en paramètre d'une fonction ou d'une procédure.
- **Exemple de fonction qui renvoie la différence d'âge entre deux personnes**

**Fonction Différence** (pers1, pers2 : tpersonne) : entier

**Début**

**Si** (pers1.age > pers2.age) alors  
        Différence ← ( pers1.age – pers2.age )

**Sinon**

        Différence ← ( pers2.age – pers1.age )

**Finsi**

**Fin Différence**

155

## LES ENREGISTREMENTS

### 4. Tableau d'enregistrements

- Il est possible de traiter plusieurs enregistrements à travers la création d'un tableau d'enregistrement.
- **Exemple:** Déclaration d'un tableau de personne pour traiter un groupe de personnes.

**Const : Nbp = 20 // nombre de personnes du groupe**

**Type**

    Personne = enregistrement

    nom: chaîne

    age: entier

**Fin**

Tab= tableau[1..Nbp] de Personne

**Var:**

groupe: Tab

156



## LES ENREGISTREMENTS

### 4. Tableau d'enregistrements

- Chaque élément du tableau représente un enregistrement qui contient plusieurs variables de type différent.
- L'accès à un enregistrement se fait à travers son indice dans le tableau.
- **groupe[2]** représente la deuxième personne dans le tableau **groupe**
- **groupe[2].nom** représente le nom de la deuxième personne dans le tableau **groupe**

	Nom	âge
1		
2		
3		
4		
5		

Indices du tableau

Nom des champs

157



Université de Sousse  
Institut Supérieur des Sciences Appliquées et de Technologie de Sousse



# Algorithmique et structures de données

## CHAPITRE 8 : LA RÉCURSIVITÉ

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

158

# LA RÉCURSIVITÉ

- **Plan du chapitre :**

- I. La notion de la récursivité
- II. Mécanisme de fonctionnement de la récursivité
- III. Principe de la récursivité
- IV. Types d'algorithmes récursifs
  - A. La récursivité directe
  - B. La récursivité indirecte
- V. Fonctionnement d'algorithmes récursifs
- VI. Performances de la récursivité

159

## I. Notion de récursivité

### Définition et exemples

- La récursivité est un concept que l'on retrouve fréquemment dans la vie quotidienne : des histoires à l'intérieur d'histoires, des films à l'intérieur de films, des tableaux dans des tableaux, etc.
- **Origine:** Le mot « récursion » vient du latin *recursare* qui signifie courir en arrière, revenir.
- **Système récursif:** Lorsqu'un système contient une autoréférence (ou une copie de lui-même), on dit que ce système est récursif.
- **Exemples:**



L'effet droste (image en abyme)



Les poupées russes



Dans le domaine végétal, le romanesco

160

## I. Notion de récursivité

### En informatique

- En informatique, la récursivité est le fait qu'une fonction ou une procédure fasse **appel à elle-même**, c'est à dire qui contient dans ses instructions un appel à elle-même.
- La récursivité permet de traduire facilement des solutions de problèmes qui ont une certaine forme récurrente en algorithmes récursifs

### Syntaxe:

```
Procédure Precur (paramètres)
Début
  ...
  Precur (valeurs)
  ...
Fin
```

161

## I. Notion de récursivité

### Etude d'un exemple : la fonction factorielle

- Rappelons tout d'abord que  $n! = 1 \times 2 \times 3 \times \dots \times n$
- On montre alors facilement la relation de récurrence  $n! = n \times (n-1)!$  Si l'on sait calculer  $(n-1)!$  on connaîtra donc la valeur de  $n!$
- Or, toujours d'après la formule de récurrence,  $(n-1)! = (n-1) \times (n-2)!$  On est donc ramené au calcul de  $(n-2)!$  Et ainsi de suite jusqu'à  $1!$  dont on connaît la valeur 1.

#### Solution itérative

```
Fonction Facto( n : Entier) : Entier
Variables
  i, f : Entier
Début
  f ← 1
  Pour i de 2 à n Faire
    f ← f * i
  FinPour
  Facto ← f
Fin
```

#### Solution récursive

La relation de récurrence est définie par :

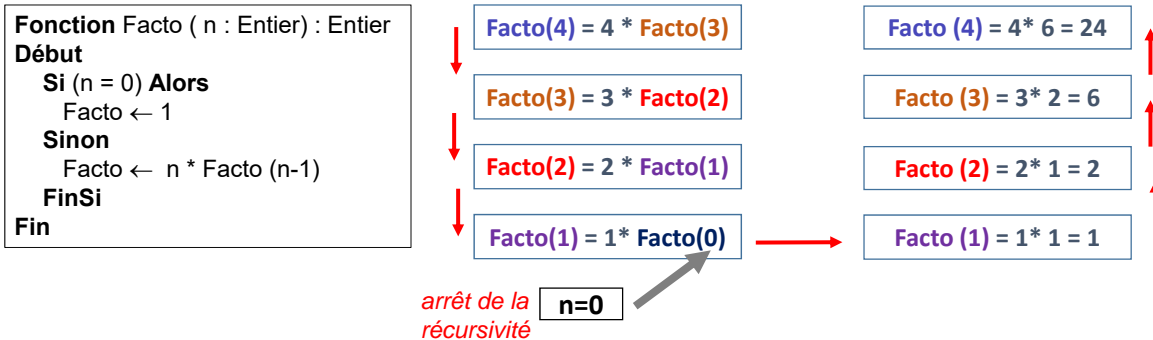
$0! = 1$   
 $n! = n \times (n-1)!$

```
Fonction Facto( n : Entier) : Entier
Début
  Si (n = 0) Alors
    Facto ← 1
  Sinon
    Facto ← n * Facto(n-1)
  FinSi
Fin
```

162

## II. Mécanisme de fonctionnement de la récursivité

- Considérons le calcul de  $4!$  par la fonction récursive définie ci-dessous :



163

## III. Principe de la récursivité

Dans toute définition récursive, on doit avoir :

- un certain nombre de cas dont la résolution est connue, ces « cas simples » formeront les cas d'arrêt de la récursivité, appelés *base de la récursivité* ;
- un moyen de se ramener d'un cas « compliqué » à un cas « plus simple ».
- Difficultés** : Il faut être sûrs que l'on retombe toujours sur un cas connu, c'est-à-dire sur une condition d'arrêt ; il faut s'assurer, alors, que la fonction est complètement définie, c'est-à-dire, qu'elle est définie sur tout son domaine d'application.
- Moyen** : existence d'un ordre strict tel que la suite des valeurs successives des arguments invoqués par la définition soit strictement monotone et finit toujours par atteindre une valeur pour laquelle la solution est explicitement définie.

164

## IV. Les types d'algorithmes récursifs

- Un algorithme est dit récursif lorsqu'il s'appelle lui-même dans sa définition (*récursivité directe*), et éventuellement lorsqu'il s'appelle par l'intermédiaire d'autres algorithmes appelés par lui-même (*récursivité indirecte*).

### A. Récursivité Directe

- Dans le cas d'une récursivité directe, l'appel récursif d'un sous-algorithme **P** se fait dans le sous-algorithme **P** lui-même.
- Cet appel peut être fait une seule fois (*récursivité simple*) ou plusieurs fois (*récursivité multiple*).

### B. Récursivité indirecte:

- Lorsqu'un sous-algorithme **P** fasse appel à un sous algorithme **Q**, qui lui-même fait appel au sous-programme **P**.

165

## IV. Les types d'algorithmes récursifs

### A. Récursivité Directe

#### • Récursivité simple

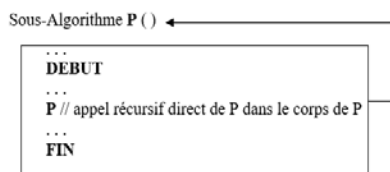
#### • Exemple :

La fonction puissance  $f(x)=x^n$  peut être définie récursivement comme suit :

$$x^n = 1 \quad \text{si } n = 0$$

$$x^n = x \times x^{n-1} \quad \text{si } n \geq 1$$

L'algorithme correspondant s'écrit :



```
Fonction PUISSANCE (x, n : entier) : entier
VAR
P : entier
DEBUT
SI (n = 0) alors
P ← 1
SINON
P ← x * PUISSANCE(x, n-1)
FINSI
PUISSANCE ← P
FIN PUISSANCE
```

166

## IV. Les types d'algorithmes récursifs

### A. Récursivité Directe

- **Récursivité multiple**

- Une définition récursive peut contenir plus qu'un appel récursif.

- **Exemple :**

- Nous voulons calculer ici les combinaisons  $C_n^p$  en se servant de la relation suivante :

$$C_n^p = \begin{cases} 1 & \text{si } p = 0 \text{ ou } p = n \\ C_{n-1}^p + C_{n-1}^{p-1} & \text{sinon} \end{cases}$$

Sous-Algorithmes **P** ( ) ←

```

.....
DEBUT
.....
P // appel récursif direct de P dans le corps de P
.....
P // appel récursif direct de P dans le corps de P
.....
FIN
    
```

```

Fonction COMBINAISON (n, p : entier) : entier
VAR
C : entier
DEBUT
SI (p = 0 ou p = n) alors
    C ← 1
SINON
    C ← COMBINAISON (n-1, p) + COMBINAISON (n-1, p-1)
FINSI
COMBINAISON ← C
FIN COMBINAISON
    
```

167

## IV. Les types d'algorithmes récursifs

### B. Récursivité indirecte

- Parfois, il est possible qu'un sous-algorithme **P** fasse appel à un sous-algorithme **Q**, qui lui-même fait appel au sous-programme **P**. On appelle cela *récursivité indirecte*. (En fait, la suite d'appels qui part de **P** peut passer par plusieurs autres sous-programmes avant d'arriver de nouveau à **P**).

Sous-Algorithmes **P** ( )

```

.....
DEBUT
.....
Q // appel récursif indirect de P dans le corps de Q
.....
FIN P
    
```

Sous-Algorithmes **Q** ( )

```

DEBUT
...
P // appel récursif indirect de Q dans le corps de P
...
FIN Q
    
```

- La procédure **P** utilise pour sa définition la procédure **Q** qui, elle-même, utilise **P** pour se définir.

168

## IV. Les types d'algorithmes récursifs

### B. Récursivité indirecte

- **Exemple** : un exemple simple de récursivité indirecte est la définition récursive des nombres pairs et impairs.
- Un nombre  $n$  positif est pair si  $n-1$  est impair ;
- un nombre  $n$  positif est impair si  $n-1$  est pair.

$$\text{pair}(n) = \begin{cases} \text{Vrai} & \text{si } n = 0 \\ \text{impair}(n-1) & \text{sinon} \end{cases}$$

$$\text{impair}(n) = \begin{cases} \text{Faux} & \text{si } n = 0 \\ \text{pair}(n-1) & \text{sinon} \end{cases}$$

```

Fonction PAIR ( $n$  : entier) : booléen
VAR
  L : booléen
DEBUT
  SI ( $n = 0$ ) alors
    L ← vrai
  SINON
    L ← IMPAIR ( $n-1$ )
  FINSI
  PAIR ← L
FIN PAIR
    
```

```

Fonction IMPAIR ( $n$  : entier) : booléen
VAR
  L : booléen
DEBUT
  SI ( $n = 0$ ) alors
    L ← faux
  SINON
    L ← PAIR ( $n-1$ )
  FINSI
  IMPAIR ← L
FIN IMPAIR
    
```

169

## IV. Les types d'algorithmes récursifs

### C. Récursivité terminale et non terminale

- Un appel récursif est dit **terminal** s'il n'est suivi d'aucune instruction.
- Un appel récursif est dit **non terminal** s'il est suivi d'un traitement

appel **terminal**

```

Procédure PREC (U)
Début
Si C alors
  i1
  PREC (f(U))
Sinon
  i2
Finsi
Fin
    
```

appel **non terminal**

```

Procédure PREC(U)
Début
Si C alors
  i1
  PREC (f(U))
  i2
sinon
  i3
Finsi
Fin
    
```

170

## V. Fonctionnement d'un algorithme récursif

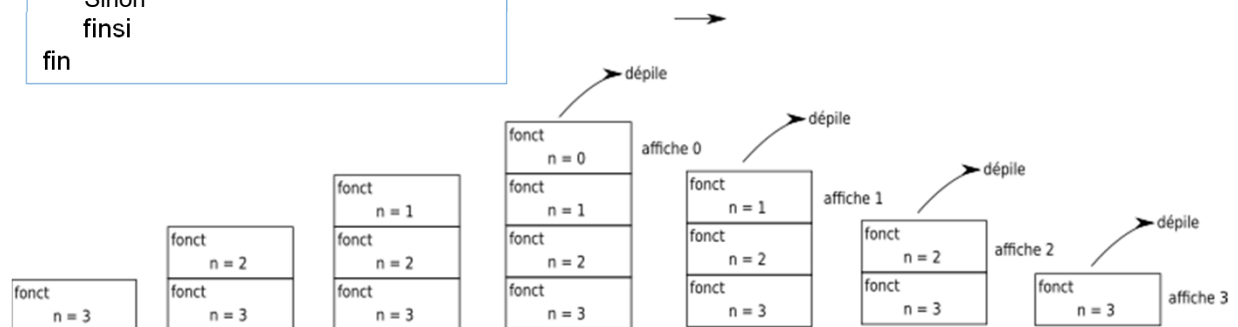
- A chaque appel récursif, une procédure/fonction quitte momentanément son environnement (l'ensemble de variables et leurs valeurs) et crée un nouvel environnement avec les nouvelles valeurs d'appel. Au retour de cet appel, et pour continuer son exécution la procédure/fonction récupère son environnement initial.
- Les différents environnements seront sauvegardés (en mémoire) dans une structure particulière appelée *pile*.

171

## V. Fonctionnement d'un algorithme récursif

```
Procédure fonct (n : entier)
  si (n=0)
    Écrire (0)
  sinon
    fonct (n-1)
    Ecrire (n)
  Sinon
  fin
```

```
Algorithme Affichage
Début
  fonct (3)
fin
```



172



## VI. Performances de la récursivité

- Les problèmes de **mémoire** et de **rapidité** sont les critères qui permettent de faire le choix entre la solution **itérative** et la solution **récursive**.
- Pour le calcul de la fonction factorielle, les différences entre les deux solutions ne se voient pas trop.
- Pour l'utilisation de la mémoire, il y aura dépassement dans les calculs avant la saturation de la mémoire.

173

## VI. Performances de la récursivité

### Etude d'un exemple: La suite fibonacci

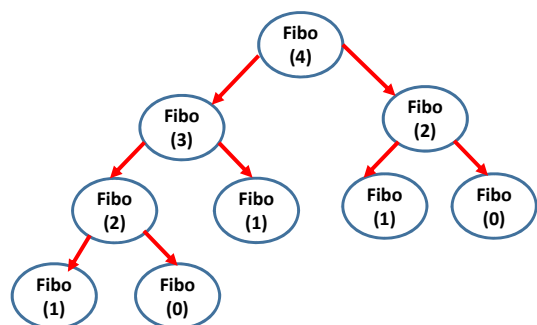
- Le calcul de la suite de fibonacci est le parfait **contre-exemple de l'efficacité** de la solution récursive.
- La relation de récurrence est définie par :

$$f(0) = 1$$

$$f(1) = 1$$

$$f(n) = f(n-2) + f(n-1) \text{ pour } (n > 1)$$

```
Fonction Fibo( n : Entier ) : Entier
Début
  Si ( n = 0 ) ou ( n = 1 ) Alors
    Fibo ← 1
  Sinon
    Fibo ← Fibo( n - 2 ) + Fibo( n - 1 )
  FinSi
Fin
```



174

## VI. Performances de la récursivité

### Etude d'un exemple: La suite fibonacci

```
Fibo(4) = Fibo(3) + Fibo(2)
        = (Fibo(2) + Fibo(1)) + Fibo(2)
        = ((Fibo(1) + Fibo(0)) + Fibo(1)) + Fibo(2)
        = ((1 + Fibo(0)) + Fibo(1)) + Fibo(2)
        = ((1 + 1) + Fibo(1)) + Fibo(2)
        = (2 + Fibo(1)) + Fibo(2)
        = (2 + 1) + Fibo(2)
        = 3 + Fibo(2)
        = 3 + (Fibo(1) + Fibo(0))
        = 3 + (1 + Fibo(0))
        = 3 + (1 + 1)
        = 3 + 2
        = 5
```

```
Fonction Fibo(n : Entier) : Entier
Variables
i, fn, fn_1, fn_2 : Entier
Début
    fn_1 ← 1
    fn_2 ← 1
    Pour i de 2 à n Faire
        fn ← fn_1 + fn_2
        fn_2 ← fn_1
        Fn_1 ← fn
    FinPour
    Fibo ← fn
Fin
```

- Le nombre d'appels récursifs est donc **très élevé** (9 pour Fibo(4), 15 pour Fibo(5), 25 pour Fibo(6) et 21891 pour Fibo(20)), d'autant plus qu'il existe une méthode **itérative** simple qui calcule simultanément Fibo(n) et Fibo(n-1).

175

## VI. Performances de la récursivité

### Avantage et inconvénient des fonctions récursives

#### Avantages:

- Écriture de la fonction proche du langage des mathématiques et de la récurrence.
- Facilité de prouver que l'algorithme est correct grâce à la démonstration par récurrence.
- Programmes élégants.

#### Inconvénients:

- Chaque appel provoque l'empilage des paramètres de la fonction.
- S'il y a beaucoup de niveaux de récursion, cela peut provoquer facilement le débordement de la pile de la machine.
- Les différents empilages (y compris pour les valeurs de retour) et dépilages ralentissent le programme.
- Les appels récursifs qui s'enchainent peuvent devenir très compliqués et produire plusieurs fois le même calcul, ce qui à nouveau ralentit fortement le programme.

176



# Algorithmique et structures de données

## CHAPITRE 9 : LES POINTEURS

1<sup>er</sup> Préparatoire MPI

Année universitaire 2020 – 2021

177

## LES POINTEURS

### • Plan du chapitre :

- I. Adresse et accès aux variables
- II. Le type pointeur
- III. Déclaration d'un pointeur
- IV. Initialisation d'un pointeur
- V. Allocation dynamique
- VI. Pointeurs et tableaux
- VII. Pointeurs et enregistrements

178

## I. Adresse et accès aux variables

### A. Notion d'adresse

- La mémoire est constituée d'un ensemble de cases de 8 bits appelés octets (byte)
- Une variable : selon son type (donc sa taille), va ainsi occuper une ou plusieurs de ces cases.
- Exemple:
  - une variable de type caractère occupera un seul octet
  - une variable de type entier occupera 8 bits ou 16 bits
- Chacun de ces emplacements mémoire (appelées blocs) est identifié par un numéro. Ce numéro s'appelle adresse.
- On peut accéder à une variable de 2 façons:
  - ➔ Adressage direct
  - ➔ Adressage indirect

179

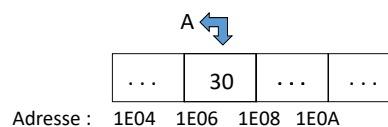
## I. Adresse et accès aux variables

### B. Accès aux variables - Adressage direct

- Pour stocker les informations nous utilisons des variables.
- La valeur d'une variable se trouve dans une case de la mémoire de l'ordinateur.
- Une variable est accessible à partir de son nom.

Variables :  
A : entier

Début  
A ← 30

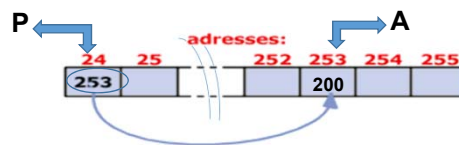


180

## I. Adresse et accès aux variables

### B. Accès aux variables - Adressage indirect

- Pour trouver des informations sur la variable A nous pouvons utiliser l'adresse de cette variable à travers une variable spéciale appelé un pointeur P.
- Soit A une variable renfermant la valeur 200, et P un pointeur qui contient l'adresse de A. En mémoire, A et P peuvent se présenter comme suit :



Dans l'adressage indirect, l'accès au contenu d'une variable se fait via un pointeur qui renferme l'adresse de la variable.

181

## II. Le type pointeur

- Pour accéder à une variable via son adresse, il suffit donc de stocker l'adresse de la variable dans un **pointeur**.
- Un pointeur permet d'accéder directement à la mémoire, de faire référence à toute entité, sans la nommer.
- Le type d'un pointeur précise le type d'objet pointé (dans la mémoire).
- Un type pointeur est représenté par le symbole **\***.

182

### III. Déclaration d'un pointeur

- **Déclaration :**

nom: \* type pointé

- **Exemple :**

Variables

**p**: \* entier //pointeur sur un entier

- **p** désigne une variable pointeur pouvant contenir uniquement l'adresse d'une variable de type **entier**.
- Si **p** contient l'adresse d'une variable entière A, on dira alors que **p** pointe vers A.

183

### IV. Initialisation d'un pointeur

- On peut initialiser une variable de type pointeur avec trois méthode.
  - Initialisation **NULL** on utilisant un pointeur NULL.
  - Initialisation avec une **adresse d'une variable existante**
  - Initialisation par **allocation dynamique**.

Pour initialiser un pointeur il faut utilisé une seule méthode.

184

#### IV. Initialisation d'un pointeur

##### A. Initialisation d'un pointeur vide (NULL)

- Si on veut que le pointeur pointe sur une case vide, il faut l'initialiser à NULL

**p**: \* entier

**p** ← NUL

**Attention:** Il faut obligatoirement initialiser un pointeur. Sinon, le pointeur (non initialisé) va se pointer n'importe où. Ce pointeur appelé «**pointeur pendant**», va écraser un emplacement mémoire quelconque en lui affectant une nouvelle valeur.

185

#### IV. Initialisation d'un pointeur

##### B. Initialisation par des données

- Pour affecter l'adresse d'une variable a un pointeur on utilise l'opérateur **&**.
- EXEMPLE

**variable:**

pvar: \* entier

var: entier

**Début**

var ← 23 (L'adresse de la variable var est: 0x3ffd14)

pvar ← &var (l'adresse de var est affecter à la variable pvar) On dit que pvar pointe sur la variable var.

0x3ffd10	0x3ffd12	ptoto
0x3ffd11		
0x3ffd12	18	toto
0x3ffd13		
0x3ffd14		
...		

**Attention:** Le pointeur pointe sur l'adresse d'une variable.

Un pointeur doit avoir le même type de la variable pointé. Un pointeur de type réel pointe sur une variable réel.

186

#### IV. Initialisation d'un pointeur

##### B. Initialisation par des données

- L'utilisation des pointeur permet d'afficher la valeur de la variable pointée.
- **Déréférencer** : c'est extraire la valeur de la variable pointé.
- **Exemple :**

Var:

a : réel

**p** : \*réel

Début

**a** ← 0.5

**p** ← &a // **p** pointe sur a

écrire (**\*p**)



**\*p** ← 1.75 // **a** vaut maintenant 1.75

Si on veut déréférencer un pointeur il faut qu'il ne soit pas NULL.

187

#### IV. Initialisation d'un pointeur

##### C. Allocation dynamique

- Souvent, nous devons travailler avec des données dont nous ne pouvons prévoir le nombre et la grandeur lors de l'écriture du programme.
- Cette opération consistant à réserver un espace-mémoire au fur et à mesure que nous en avons besoin pendant l'exécution du programme, s'appelle **allocation dynamique**.
- On utilise l'opérateur « **Allouer** » qui va réserver de la mémoire mais ne l'initialise pas!
- Exemple

**variables**

**P**. \* entier // **P** est une variable de type pointeur

Début

**allouer P** // **P** va contenir l'adresse de la mémoire allouée

Début

**\*P** ← 123

Etat initial	Action	Etat final
P <input type="text"/>	Allouer (P)	P <input type="text" value="@"/> → <input type="text" value="123"/>

188



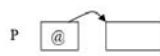

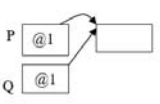

## IV. Initialisation d'un pointeur

### C. Allocation dynamique

- Pour éviter la saturation de la mémoire, il est faut désallouer les emplacements alloués en à travers l'action **libérer**.

libérer **P**

- La libération d'un pointeur implique la suppression de la variable dynamique référencée par le pointeur **P** et non pas le pointeur lui-même.

Etat initial	Action	Etat final
	Libérer (P)	
	Libérer (P)	

189

## V. Allocation dynamique

- Les variables **dynamiques** sont créées, modifiées et supprimées au fur et à mesure des besoins au cours de l'exécution d'un algorithme.
- Les variables dynamiques ne sont pas déclarées dans la partie déclaration de variables
- Elles ne peuvent pas être référencées par un identificateur direct.
- Elles sont référencées par le **pointeur**.
- Le pointeur est une **variable statique** qui contient l'adresse en mémoire d'une autre variable (**dynamique**). Son rôle est de repérer, référencer ou désigner un autre objet.
- Une variable **dynamique** est un objet créé pendant l'exécution d'un programme. Elle a un type et une valeur mais n'a pas de nom.

190

## V. Allocation dynamique

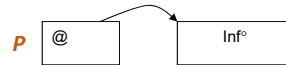
- Un type pointeur est défini par le symbole \* suivi du type de la **variable dynamique**.

Exemple :

*P : pointeur vide*

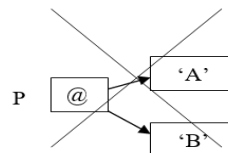
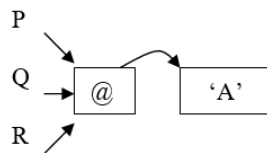


*P : pointeur qui pointe sur une case contenant une information*



Remarque :

- un objet de type dynamique peut être pointé par un ou plusieurs pointeurs.
- Alors qu'un seul pointeur ne peut pointer que sur un seul objet.



191

## VI. Pointeurs et tableaux

**Un tableau** est en fait **un pointeur** sur une zone mémoire où sont stockées toutes les cases.

→ Ainsi pour un tableau d'entier t de longueur N :

- t est un pointeur sur la première case du tableau, par conséquent, t[1] et \*t désignent tous les deux le contenu de la première case du tableau ;
- t+2 est donc un pointeur sur la troisième case du tableau, par conséquent, t[3] et \*(t+2) désignent le contenu de la troisième case du tableau.
- t+i est, de façon générale, un pointeur sur la i + 1-ième case du tableau, par conséquent, t[i+1] et \*(t+i) désignent tous les deux le contenu de la i + 1-ième case du tableau.

192

## VII. Pointeurs et enregistrements

- On peut utiliser un pointeur pour mémoriser l'adresse d'un enregistrement.

### **Exemple:**

pointpers: \*personne

- On peut accéder aux champs de l'enregistrement pers en utilisant l'opérateur \*. ou ->

Écrire(\*pointpers.nom)

ou bien

Écrire(pointpers -> nom)

193

## Références

- [1] Dr. M. AMAD Algorithmique et Structures de Données Cours et Travaux Dirigés Support destiné aux étudiants de niveau Première et deuxième année Licence, 2016
- [2] Christine Froidevaux, Marie-Claude Gaudel, Michèle Soria, Types de données et algorithmiques, EDISCIENCE International, 1994.
- [3] Thomas H. Cormen, Introduction to Algorithms, Edition Dunod, 2ème édition, 2002.

194