République Tunisienne

Ministère de l'Enseignement Supérieur et

de la Recherche Scientifique

Support de cours

ALGORITHMIQUE & STRUCTURES DE DONNÉES I

Niveau : 1ère année Licence Fondamentale

Option : Informatique Appliquée à la Gestion

Elaboré par : **MBAREK DHAHRI**

Version 2012

Avant-propos

Pré-requis

- Néant

Objectifs généraux

Ce cours permettra aux étudiants d'analyser un problème donné et de définir l'algorithme traduisant la solution du problème d'une manière rigoureuse et optimisée et prête à être traduite en utilisant un langage de programmation quelconque.

Public Cible

Ce cours est destiné essentiellement aux étudiants de la première année Licence Fondamentale en Informatique Appliquée à la Gestion.

Volume horaire

Ce cours est présenté, de manière hebdomadaire, comme suit :

- **1.5h** du Cours Intégré
- **3h** de Travaux Dirigés
- Soit en total : **21h** de Cours Intégré et **42h** de Travaux Dirigés

Moyens pédagogiques

- Notes de cours
- Tableau
- Polycopiés de Travaux Dirigés

Evaluation

- Coefficient : 2
- Devoir Surveillé : 30%
- Examen final : 70%

Table des matières

CHAPITRE

1

INTRODUCTION A L'ALGORITHMIQUE

Objectifs spécifiques

- Connaître le vocabulaire de base en programmation
- Comprendre la démarche de programmation

Plan du chapitre

1. Le traitement informatique
2. Algorithme et algorithmique
3. Notion de programme

Volume horaire

1 séance de cours intégré

Chapitre 1 Introduction à l'algorithmique

1 Le traitement informatique

L'informatique a envahi tous les domaines de la vie courante. Pratiquement, tout problème est totalement ou en partie informatisé.

Un ordinateur est un ensemble de circuits électroniques qui traite l'information grâce à un programme qu'il mémorise, communique et archive des informations.

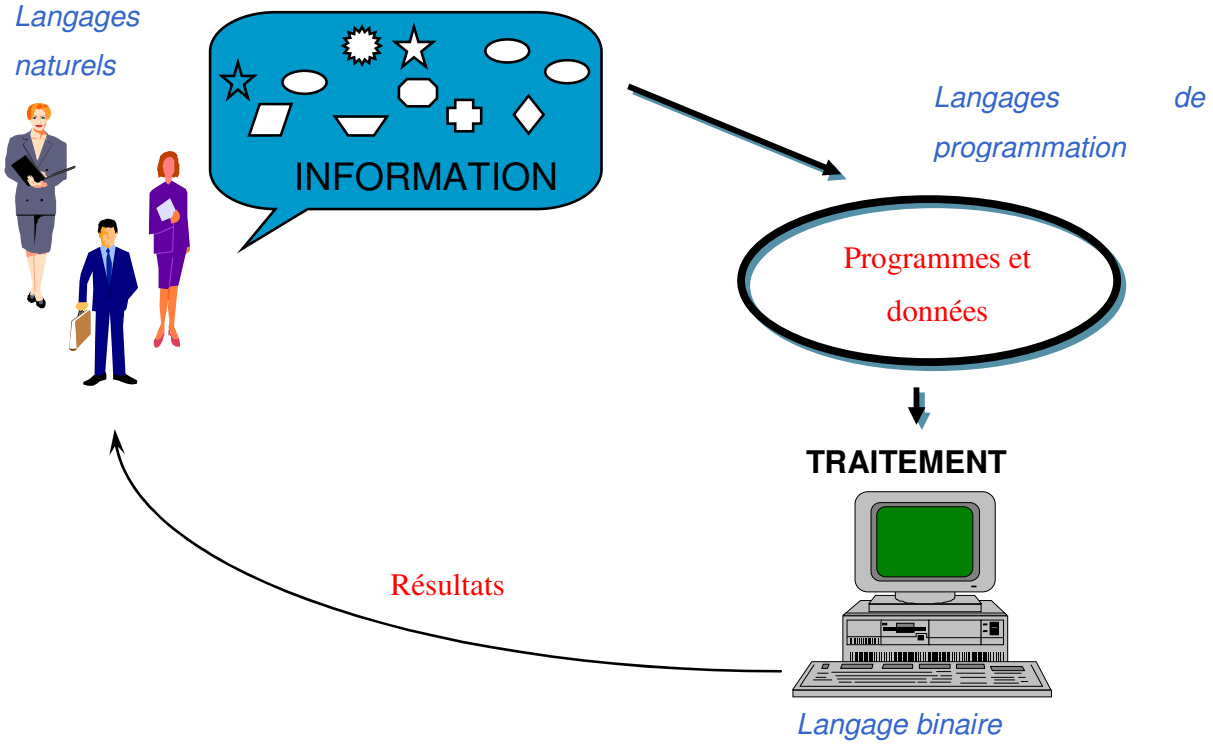
Le traitement de l'information se fait automatiquement et vise à résoudre un problème bien défini.

Les différentes fonctions correspondent, en fait, à trois constituants de l'ordinateur :

* **La mémoire centrale** : contient les programmes systèmes nécessaires au bon fonctionnement de l'ordinateur, et les programmes utilisateurs répondant à un besoin particulier et résolvant un problème rencontré par le dit utilisateur. Ces programmes auront besoin d'un ensemble de données afin d'être exécutés et fournir les résultats escomptés, ces données existent également dans la mémoire centrale.

* **L'unité centrale** : va s'occuper de l'exécution des programmes logés dans la mémoire centrale.

* **Les périphériques** : sont les unités qui assurent la relation de l'ordinateur avec le monde extérieur.



2 Algorithme et algorithmique

2.1 Définitions

* Un *algorithme* est une suite structurée et finie d'actions ou d'instructions servant à résoudre un problème donné.

C'est une marche à suivre :

- ✓ permettant de résoudre de manière certaine un problème donné
- ✓ dont les opérations (ou instructions) sont toutes définies et portent sur des objets (variables ou constantes),
- ✓ dont l'ordre d'exécution des opérations est défini sans ambiguïté et suivant une logique déterminée.

* *L'algorithmique* est l'étape précédant la programmation.

Un algorithme s'exprime dans un langage indépendant des langages de programmation.

2.2 La démarche algorithmique

Cinq étapes pour arriver à un algorithme à partir d'un problème donné :

1. Comprendre le problème.
2. Identifier les données d'entrée et de sortie liées au problème.
3. Formuler le problème en actions élémentaires précises avec les données nécessaires pour le calcul.
4. Développer l'algorithme général suivant les actions.
5. Détailler chaque action de l'algorithme (plus précis - sans ambiguïté).

Ensuite, traduire l'algorithme avec un langage de programmation pour obtenir un programme.

3 Notion de programme

3.1 Définitions

Un programme est une suite logique d'instructions que l'ordinateur doit exécuter.

Remarques :

- ✓ Instructions et données doivent résider en mémoire centrale pour être exécutées.
- ✓ Le processeur se charge d'effectuer les opérateurs arithmétiques et logiques qui transformeront les données en résultats.

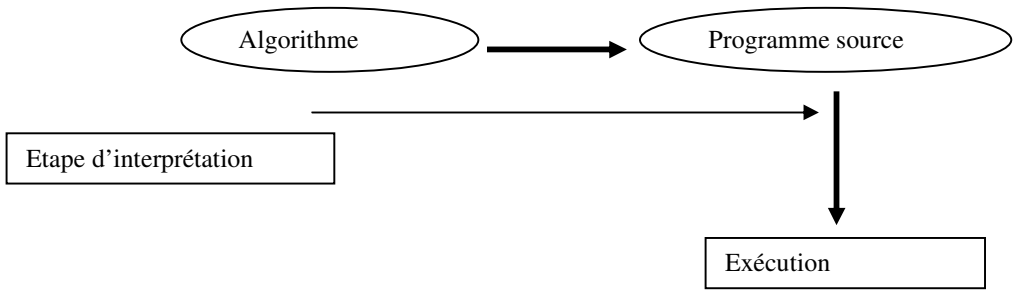
C'est avec un ***langage de programmation*** que nous traduisons notre algorithme en un programme exécutable par l'ordinateur.

Caractéristiques d'un langage de programmation :

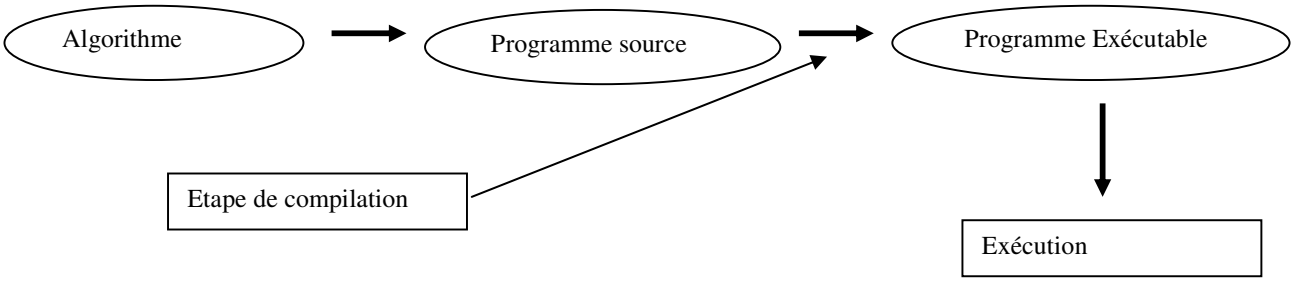
- il est composé d'un vocabulaire obéissant à une sémantique et à une syntaxe ;
- il est souvent muni d'un analyseur permettant de détecter les erreurs faites sur le vocabulaire et la syntaxe ;
- il est souvent muni d'un éditeur de texte permettant au programmeur de saisir son programme ;

L'exécution d'un programme dépend du langage utilisé :

Si le langage est dit **interprété**, la traduction en langage machine se fait au moment de l'exécution et instruction par instruction



Si le langage est dit **compilé**, on génère un autre programme à partir du programme déjà écrit compréhensible par la machine après avoir corrigé toutes les erreurs révélées par le compilateur.



Exemple :

Langage interprété	**Langage compilé**
HTML	PASCAL
BASIC	C
PROLOG ……	C++ …..

3.2 La démarche de programmation

- Phase d'analyse et de réflexion (algorithmique) : on cherche par quel moyen on pourra obtenir les résultats escomptés à partir des données dont on dispose : c'est l'analyse du problème et la recherche d'un algorithme
- Phase de programmation
  - traduction de l'algorithme en programme
- Phase de test
- Phase d'exécution

CHAPITRE

2

TYPES DE DONNEES, VARIABLES ET CONSTANTES

Objectifs spécifiques

- Identifier et manipuler les constantes et les variables.
- Manipuler les types de base.
- Reconnaître les expressions.
- Evaluer des expressions en tenant compte de la priorité des opérateurs.

Plan du chapitre

1. Les variables
2. Les constantes
3. Les types de base
4. Les expressions
5. Exercices d'application
6. Solution des exercices

Volume horaire

1 séance de cours intégré

Chapitre 2 Types de données, Variables et Constantes

1 Les variables

Un programme s'exécute en manipulant des données se trouvant dans la mémoire centrale. Une variable est donc un nom d'un emplacement de la mémoire centrale qui contient des données. Ces données varient au cours de l'exécution du programme. Chaque variable possède un type et est rangée en mémoire à une adresse précise.

D'une manière générale, un nom d'une variable est formé d'une ou plusieurs lettres, les chiffres sont autorisés à condition de ne pas apparaître au début du nom.

En effet, le nom d'une variable doit respecter les 4 règles suivantes:

- il ne doit pas contenir d'espaces
- il ne doit pas contenir des caractères spéciaux (sauf le tiret de soulignement (_))
- il ne doit pas être long
- il ne doit pas commencer par un chiffre

Déclaration :

- Toute variable utilisée dans un programme doit impérativement être déclarée préalablement à son utilisation.
- La partie déclarative des variables est désignée par le terme **Var**.

Exemple : nom et type

Var

somme : Entier

Ainsi : Mont_Fact, P, Quantite et f1 sont tous des noms corrects.

2 Les constantes

Une constante est un objet de valeur invariable. Elle est la réalisation d'une valeur d'un type particulier.

Déclaration :

- Toute constante utilisée dans un programme doit impérativement être déclarée préalablement à son utilisation.
- La partie déclarative des constantes est désignée par le terme **Const**. S'il y a des constantes à déclarer, il faut mettre cette partie avant la partie déclarative des variables.

- Toute constante possède deux attributs :

* un **nom** ou **identificateur** qui sert à la désigner,

* une **valeur**.

Exemple :

Const

TVA = 21

3 Types de base

A chaque variable utilisée dans le programme, il faut associer un type qui permet de définir :

* l'ensemble des valeurs que peut prendre la variable

* l'ensemble des opérations qu'on peut appliquer sur la variable.

La syntaxe de l'action de déclaration est la suivante :

Var Variable 1, Variable 2, … : **Type**

Les principaux types utilisés en algorithmique sont :

- le type entier
- le type réel
- le type caractère
- le type chaîne de caractères
- le type logique ou booléen.

3.1 Le type entier

Les valeurs du type entier forment un sous-ensemble de l'ensemble des nombres entiers relatifs (un sous-ensemble de Z).

Les opérateurs arithmétiques sur les entiers :

On ajoutera aux opérateurs arithmétiques usuels (+, - et *), deux autres opérateurs **DIV** et **MOD** définis ci-dessous :

DIV : *donne le quotient dans la division entière*

Exemple : 5 DIV 3 vaut 1

MOD : *donne le reste de la division entière*

Exemple : 5 MOD 3 vaut 2

3.2 Le type réel

Les valeurs du type réel forment un sous ensemble de l'ensemble des nombres réels IR.

Il existe deux formes de représentation des réels :

* la forme **usuelle** avec le point comme symbole décimal.

Exemples : 0.1 -2.5 +3.14

* la notation **scientifique** selon le format **aEb**, où :

- **a** est la **mantisse**, qui s'écrit sous une forme usuelle
- **b** est **l'exposant** représentant un entier relatif.

Exemple : 20.0E+03

La lettre « E » se lit « dix puissance ».

Les opérateurs arithmétiques sur les réels : +, -, *, / (division réelle)

3.3 Le type booléen (logique)

Dans certains cas, il sera nécessaire d'évaluer des propositions pour déduire si elles sont vraies ou fausses. Une variable est dite booléenne si elle prend une valeur parmi VRAI ou FAUX.

Les opérateurs logiques sur les booléens :

NON : Négation,

ET : Conjonction,

OU : Disjonction,

Table de vérité des opérateurs logiques :

F = FAUX et **V** = VRAI

X	**Y**	**NON** (X)	X **ET** Y	X **OU** Y
F	F	V	F	F
F	V	V	F	V
V	F	F	F	V
V	V	F	V	V

Exemples :

La proposition (5 > 3) ET (12 > 3) a la valeur VRAI

La proposition (7 > 8) ET (1 < 3) a la valeur FAUX

Les opérateurs de comparaison sur les booléens : <, >, <=, >=, =, ≠

3.4 Le type caractère

Le type des caractères appartient à l'une des catégories suivantes :

* les chiffres de 0 à 9

* les lettres de l'alphabet (de A à Z) majuscules et minuscules

* les caractères spéciaux : +, -, /, …

* les caractères non imprimables

Opérateurs sur les caractères : =, ≠, <, <=, > et >=

La comparaison entre les caractères se fait selon leurs codes ASCII (voir annexe).

Exemples : ″A″ < ″G″ est une proposition VRAIE,

″E″ > ″e″ est une proposition FAUSSE.

Remarque :

En plus de ces types prédéfinis, le programmeur a la possibilité de définir lui-même de nouveaux types en fonction de ses besoins.

Exemple

Types

Saison = ("A","H","P","E")

Tnote = 0 .. 20

Variables

s : Saison

note : Tnote

→ La variable s de type saison ne peut prendre que les valeurs "A", "H","P" ou "E".

4 Les expressions

Une expression est toute composition d'opérateurs et d'opérandes réalisant un calcul déterminé. L'opérande est l'élément sur lequel on applique l'opération. Il peut être une constante, une variable, une valeur ou un résultat envoyé par une fonction.

Exemples :

(2 >= 5) est une expression **logique**.

3 + CARRE(X) + Y est une expression **arithmétique**.

1er opérande 2ème opérande 3ème opérande

L'ordre de l'évaluation des différentes parties d'une expression correspond en principe à celle que nous connaissons des mathématiques.

1	Les parenthèses	`()`
2	Les opérateurs unaires	`- NON`
3	Les opérateurs multiplicatifs	`* / MOD DIV`
4	Les opérateurs addititifs	`+ -`
5	Les opérateurs relationnels	`< <= > >= = <>`

Remarque :

- pour les opérateurs de même priorité, on commence par celui qui est le plus à gauche.

Exemple : 5 + 6 - 3

- si on veut imposer un ordre, on doit utiliser les parenthèses

5 Exercices d'application

Exercice 1

1- Quel est l'ordre de priorité des différents opérateurs de l'expression suivante :

((3 * a) – x ^ 2) – (((c – d) / (a / b)) / d)

2- Evaluer l'expression suivante :

5 + 2 * 6 – 4 + (8 + 2 ^ 3) / (2 – 4 + 5 * 2)

3- Ecrire la formule suivante sous forme d'une expression arithmétique : $\frac{(4-ab)^2 - 4ab}{2a-b}$

Exercice 2

Sachant que a = 4, b = 5, c = -1 et d = 0, évaluer les expressions logiques suivantes :

1- (a < b) ET (c >= d)

2- NON (a < b) OU (c ≠ b)

3- NON ((a ≠ b ^ 2) OU (a*c < d))

Exercice 3

Compléter le tableau suivant par le type du résultat :

Opérateur	Type opérande 1	Type opérande 2	Type du résultat
DIV	entier	entier	.....
/	entier	entier	.....
*	réel	entier	.....
-	entier	entier	.....

6 Solution des exercices

Exercice 1

2- 5 + 2 * 6 – 4 + (8 + 2 ^ 3) / (2 – 4 + 5 * 2) = 15

3- ((3 – x * y) ^ 2 – 4 * a * c) / (2 * x – z)

Exercice 2

1- Faux 2- Vrai 3- Faux

Exercice 3

Entier Réel Réel Entier

CHAPITRE

3

LES INSTRUCTIONS SIMPLES

Objectifs spécifiques

- Comprendre les actions algorithmiques simples
- Connaître la structure générale d'un algorithme

Plan du chapitre

1. L'instruction d'affectation
2. Les instructions d'entrée/sortie
3. Structure générale d'un algorithme
4. Exercices d'application
5. Solution des exercices

Volume horaire

1 séance de cours intégré

Chapitre 3 Les instructions simples

1 L'instruction d'affectation

Une instruction d'affectation permet de placer une valeur dans une variable.

Sa syntaxe générale est de la forme :

Variable ← Expression

Ainsi l'instruction A ← 5 permettra de dire "affecter" à la variable A la valeur 5, c'est à dire : ranger dans la variable A la valeur 5

Exemples

N ← 1

NB ← A + B

Une instruction d'affectation réalise les opérations suivantes :

1- elle évalue l'expression située à droite du signe ←

2- si elle est correcte elle détermine sa valeur

3- elle vérifie la compatibilité du type de la valeur de l'expression avec celui de la variable située à gauche du signe d'affectation

4- elle range le résultat, si tout est correct, dans la variable située à gauche de ce signe. Sinon elle produit une erreur.

Une affectation ne modifie jamais la valeur des variables qui apparaissent dans l'expression

Ainsi l'affectation B ← A + 3 détermine la valeur de l'expression (A + 3) et la range dans B, sans pour autant modifier la valeur de A.

Si la variable réceptrice (ici B) contient déjà une valeur, celle-ci sera écrasée et remplacée par celle qui vient d'être déterminée.

Exemple

affectation	A	B
	-	-
B ← 1	-	1
A ← 1	1	1
B ← A + 3	1	~~1~~ 4
A ← 3	~~1~~ 3	4

Exercice 1

Donner, pour chaque instruction la valeur contenue dans chaque variable.

Instructions	Contenu de A	Contenu de B	Contenu de C
A ← 1			
B ← A+3			
B ← 5			
C ← A+B			
A ← 2			
C ← B-A			

Exercice 2

Donner la suite d'instructions pour permuter le contenu de deux variable m et n de deux manières différentes.

2 Instructions d'Entrée/Sortie

Pour qu'un programme présente un intérêt pratique, il devra pouvoir recevoir et communiquer un certain nombre d'informations (données / résultats) par l'intermédiaire d'un périphérique : c'est le rôle des instructions **d'entrée/sortie**.

2.1 Instruction d'écriture *: communication Programme → Utilisateur*

L'instruction d'écriture « **Ecrire** » consiste à écrire une donnée sur un périphérique de sortie (l'écran).

La forme générale de cette instruction est :

Ecrire (expression1, expression2, ...)

Exemples :

Ecrire (var) : affiche sur l'écran le contenu de la variable var

Ecrire (100) : affiche sur l'écran 100

Ecrire ("Bonjour") : affiche sur l'écran le texte Bonjour var

Ecrire ("Bonjour", var) : affiche sur l'écran le texte Bonjour puis le contenu de la variable var

2.1 Instruction de lecture *: communication Utilisateur → Programme*

L'instruction de lecture « **Lire** » a pour rôle de permettre à l'utilisateur d'entrer des valeurs au programme à partir d'un périphérique d'entrée (clavier).

La forme générale de cette instruction est :

Lire (variable1, variable2, ...)

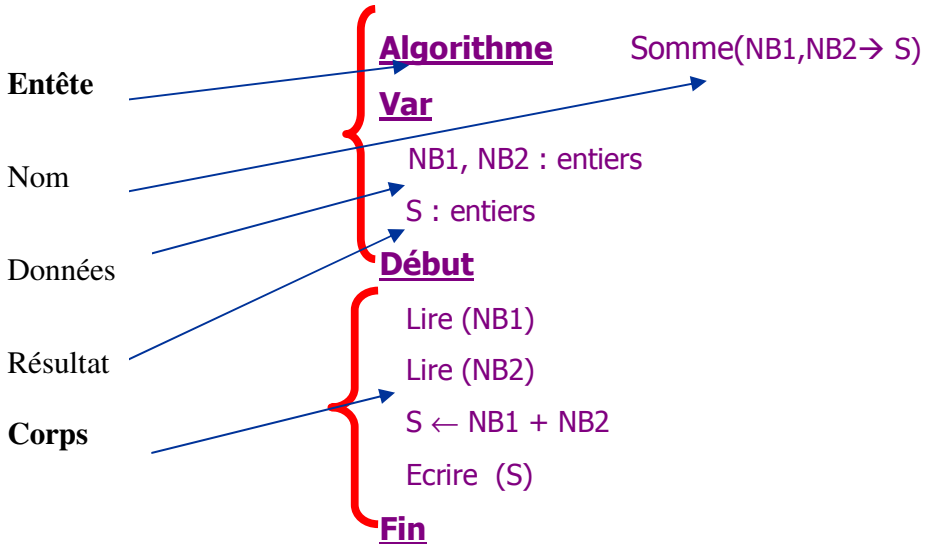
3 Structure générale d'un algorithme

Un algorithme se compose :

- d'un **en-tête** qui précise le nom attribué à l'algorithme, les données de l'algorithme, les résultats fournis par l'algorithme.
- d'un **corps** formé par une suite ordonnée d'actions et/ou d'autres algorithmes, délimité par les mots clés : **début** et **fin**.

```
Algorithme Nom_algorithme
Const
        C1 = Valeur1
        …
Type
        Type1 = définition du type
        …
Var
        V1 : Type1
        …
Début
        Instruction1
        …
Fin.
```

Exemple :



4 Exercices d'application

Exercice 1

Donner toutes les raisons pour lesquelles l'algorithme suivant est incorrect :

1. **Algoritme** Incorrect
2. x, y : Entier

3. z : Réel
4. **Début**
5. z ← x + 2
6. y ← z
7. x*2 ← 3 + z
8. y ← 5y + 3
9. **Fin**.

Exercice 2

Ecrire un algorithme qui permet de lire trois variables A, B et C, puis de faire une permutation circulaire de sorte que la valeur de A passe dans B, celle de B dans C et celle de C dans A. On utilisera une seule variable supplémentaire.

Exercice 3

Écrire un algorithme qui convertit en heures, minutes et secondes une durée de temps donnée en seconde.

5 Solution des exercices

Exercice 1

- ligne 1 : Le mot Algorithme s'écrit avec un « h » au milieu
- ligne 2 : La déclaration des variables commence par le mot « Var »
- ligne 5 : La valeur de x est indéterminée
- Ligne 6 : incompatibilité de type (un résultat réel affecté à une variable de type entier)
- Ligne 7 : Le membre gauche d'une affectation doit être une variable
- Ligne 8 : Il faut écrire 5*y et non 5y.

Exercice 2

```
X ← C
C ← B
B ← A
A ← X
```

Exercice 3

```
Algorithme conversion
Var
  D, H, M, S : entier
Début
  Lire (D)
  H ← D DIV 3600
  M ← D MOD 3600 DIV 60
  S ← D MOD 60
  Ecrire ("heure :", H, "minutes : ", M,
   "secondes : ", S)
Fin
```

CHAPITRE

4

LES STRUCTURES CONDITIONNELLES

Objectifs spécifiques

- Résoudre des problèmes faisant appel aux structures de contrôle conditionnelles
- Choisir la forme adéquate des structures de contrôle conditionnelles

Plan du chapitre

1. Introduction
2. Structure conditionnelle simple
3. Structure conditionnelle imbriquée
4. Structure conditionnelle à choix
5. Exercices d'application
6. Solution des exercices

Volume horaire

1 séance de cours intégré

Chapitre 4 Les structures conditionnelles

1 Introduction

Les instructions d'affectation, d'entrée et de sortie sont insuffisantes pour confronter des situations traitant des conditions. On aura besoin alors de choisir entre 2 ou plusieurs traitements selon la réalisation ou non d'une certaine condition d'où la notion de traitement *conditionnel*.

On distingue deux structures de traitement conditionnel à savoir :

- La structure conditionnelle **simple** dans laquelle on a à choisir entre deux traitements au plus ;
- La structure conditionnelle **imbriquée** dans laquelle on a la possibilité de choisir un traitement parmi plusieurs.

2 Structure conditionnelle simple

2.1 Forme réduite

Syntaxe :

SI <*condition*> **ALORS**
    <*action*>
FINSI

La partie « *action* » est composée d'une ou de plusieurs instruction(s). Cette partie est exécutée si la **condition** est vraie.

Exemple :

ALGORITHME VALEUR_ABSOLUE (A, B → C)
VAR
    A, B, C : REELS
DEBUT
  Lire (A, B)
  C ← A - B
  SI C < 0 **ALORS**
    C ← (-C)
  FINSI
  Ecrire ("La valeur absolue est : ", C)
FIN

2.2 Forme alternative

Syntaxe :

```
SI  <condition>     ALORS
        <action1>
SINON
        <action2>
FINSI
```

Lorsque l'évaluation de la *condition* produit la valeur :

- VRAI : *L'action1* est exécutée
- FAUX : *L'action2* est exécutée.

Action1, comme action2, peuvent être soit une instruction ou un ensemble d'instructions.

Exemple :

ALGORITHME VALEUR_ABSOLUE (A, B → C)

VAR

A, B, C : REELS

DEBUT

Lire (A, B)

```
SI A > B  ALORS
        C ← A - B
SINON
        C ← B - A
FINSI
```

Ecrire ("La valeur absolue est : ", C)

FIN

3 Structure conditionnelle imbriquée

Syntaxe :

```
SI  <condition1> ALORS
        SI <condition2>   ALORS
                <action21>
        SINON
                <action22>
        FINSI
```

```
SINON
        SI <condition3> ALORS
            <action31>
        SINON
            <action32>
        FINSI
FINSI
```

Exemple :

Un robot conduit une voiture. Il peut exécuter trois actions "s'arrêter", "ralentir", "passer" en fonction de la couleur des feux qui sera une variable saisie.

```
ALGORITHME ROBOT
VAR
    couleur : caractère
DEBUT
  Lire (couleur)
  SI couleur = "r" ALORS
    Ecrire ("Je m'arrête")
  SINON   SI couleur = "o"  ALORS
    Ecrire ("Je ralentis")
  SINON
    Ecrire ("Je passe")
  FINSI
FIN
```

4 Structure conditionnelle à choix

La structure de choix permet une présentation plus claire d'un ensemble d'alternatives imbriquées.

Syntaxe :

```
SELON <variable ou expression> FAIRE
    <valeur1>  :  <action1>
    <valeur2>  :  <action2> …
    <valeurn>  :  <actionn>
     SINON   <action>
FINSELON
```

- La séquence d'instructions numéro i sera exécutée si la valeur du sélecteur appartient à la i^{ème} liste de valeurs.

- Le sélecteur est une variable ou une expression de type **scalaire** (le résultat est un entier ou un caractère).

Exemple 1 :

```
ALGORITHME ROBOT
VAR
        couleur : caractère
DEBUT
  Lire (couleur)
  SELON couleur FAIRE
        "r" : Ecrire ("Je m'arrête")
        "o" : Ecrire ("Je ralentis")
        "v" : Ecrire ("Je passe")
  FINSELON
FIN
```

Exemple 2 : Saisir le numéro du mois, puis afficher la saison correspondante

```
ALGORITHME SAISON
VAR
        Mois : entier
DEBUT
  Lire (Mois)
  SELON Mois FAIRE
        12, 1, 2 : Ecrire ("Hiver")
        3..5 : Ecrire ("Printemps")
        6..8 : Ecrire ("Eté")
        9..11 : Ecrire ("Automne")
        SINON       Ecrire ("Erreur")
  FINSELON
FIN
```

5 Exercices d'application

Exercice 1

Ecrire un algorithme qui permet de saisir un entier N, puis d'afficher s'il est divisible par son chiffre de dizaine.

Exercice 2

Ecrire un algorithme permettant de simuler une calculatrice à 4 opérations (+, -, *, et /). Utiliser la structure « **selon** » pour le choix de l'opération à effectuer.

6 Solution des exercices

Exercice 1

```
Algorithme Divisible
Var
N, d : Entier
Début
Ecrire ("Entrer un entier r : ") Lire (N)
d ← N MOD 100 DIV 10
Si (d = 0) Alors
  Ecrire ("Non divisible")
Sinon si N MOD d = 0 alors
     Ecrire ("Divisible")
Sinon
    Ecrire ("Non divisible")
FinSi
Fin.
```

Exercice 2

```
Algorithme calculatrice
Var
a, b : Réel
op : Caractère
Début
Ecrire ("Première opérande : ")  Lire (a)
Ecrire ("Opération : ") Lire (op)
Ecrire ("Deuxième opérande : ") Lire (b)
Selon op Faire
 "+" : Ecrire ("Résultat = ", a + b)
 "-" : Ecrire ("Résultat = ", a - b)
 "*" : Ecrire ("Résultat = ", a * b)
 "/" : Si (b # 0) Alors
          Ecrire ("Résultat = ", a / b)
       Sinon
          Ecrire ("Division par zéro ! ")
       FinSi
  Sinon  Ecrire ("opérateur erroné... ")
FinSelon
Fin.
```

CHAPITRE

5

LES STRUCTURES ITERATIVES

Objectifs spécifiques

- Résoudre des problèmes faisant appel aux structures de contrôle itératives
- Savoir choisir la forme adéquate pour résoudre un problème

Plan du chapitre

1. Introduction
2. Structure Pour … faire
3. Structure Répéter … jusqu'à
4. Structure Tant que … faire
5. Sélection d'une itération
6. Exercices d'application
7. Solution des exercices

Volume horaire

2 séances de cours intégré

Chapitre 5 Les structures itératives

1 Introduction

Pour certaines applications, il est nécessaire de pouvoir exécuter un ou plusieurs traitements un certain nombre de fois, ou jusqu'à ce qu'une condition spécifique soit remplie.

Pour éviter de réécrire le même traitement plusieurs fois, nous pouvons nous servir d'une structure itérative (ou structure répétitive). Une structure itérative est appelée **boucle**.

2 Structure Pour … faire

Syntaxe :

```
POUR Compteur DE <valeur initiale> A <valeur finale> [pas p] FAIRE
        <action>
        …
FINPOUR
```

Cette structure permet de répéter une (ou plusieurs) action(s) un *nombre de fois connu d'avance*.

Remarques :

- Le compteur est de type Entier. La valeur initiale et la valeur finale sont des variables (ou constantes) de type Entier.
- Le paramètre p est le pas *d'incrémentation* c'est-à-dire la valeur d'augmentation progressive du compteur. Il doit être un Entier.
- L'incrémentation de la variable *Compteur* se fait automatiquement de la valeur du pas p. Si cette dernière n'est pas spécifiée, l'incrémentation se fait de **1**.

Exemple 1 : afficher la table de multiplication de 5

```
Algorithme TABLE_MULT5
Var
  NBR    : Entier
DEBUT
  POUR NBR DE 1 A 10 FAIRE
        Ecrire  (NBR, " * 5 = ", NBR*5)
  FINPOUR
FIN
```

Exemple 2 : Calcul du factoriel d'un entier positif

Algorithme Factoriel

Var

n, i, f : Entier

DEBUT

Ecrire ("Entrer un entier positif : ") Lire(n)

f ← 1

POUR i **DE** 2 **A** n **FAIRE**

f ← f * i

FINPOUR

Ecrire (n, "! = ", f)

FIN.

3 Structure Répéter … jusqu'à

Syntaxe :

REPETER

<action>

…

JUSQU'A <condition>

- Cette structure permet la répétition d'une (ou de plusieurs) action(s) jusqu'à ce qu'une condition soit vérifiée.
- La condition n'est testée qu'après une *première exécution* de l'action. Cette dernière est alors exécutée *au moins une fois*.

Exemple 1 : Saisir un entier strictement positif

REPETER

Ecrire ("Donner un nombre > 0")

Lire (N)

JUSQU'A N > 0

Exemple 2 : Calcul du factoriel d'un entier positif à l'aide de répéter

f ← 1

i ←1

```
REPETER
        f ← f * i
        i ← i + 1
JUSQU'A i > n
```

4 Structure Tant que … faire

Syntaxe :

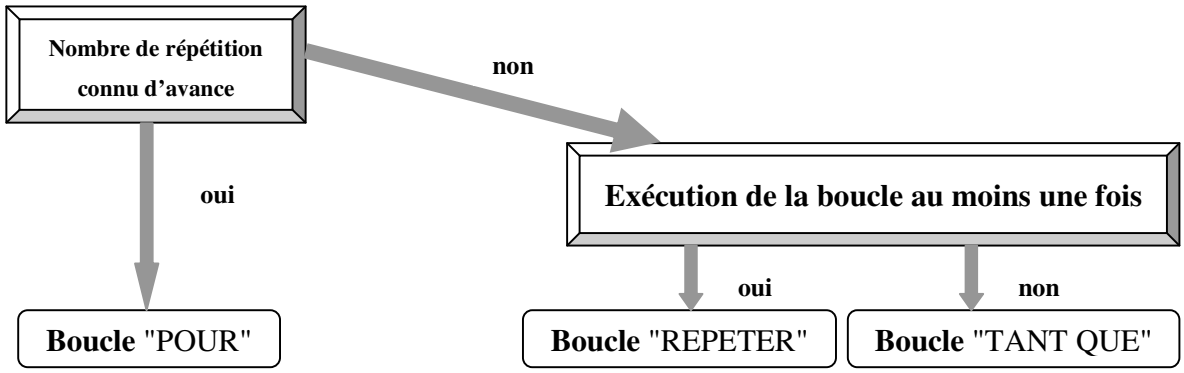
```
TANT QUE  <Condition>  FAIRE
          <action>
          …
FINTANTQUE
```

- Cette structure permet la répétition d'une (ou de plusieurs) action(s) tant qu'une condition est satisfaite; dès que celle-ci cesse d'être vérifiée, l'exécution de l'action s'arrête.
- La condition est testée *avant la première exécution de l'action.*
- La structure TANT QUE est utilisée lorsque l'action peut être exécutée **0 à N fois.**
- On s'arrête lorsque la condition est **fausse.**

Exemple : afficher à l'écran une suite d'entiers saisis au clavier, ainsi que le nombre d'entiers saisis.

```
Algorithme COMPTER_ENTIER
Var
        N, NB : Entier
DEBUT
        Ecrire ("Entrer un nom (ou 0 pour arrêt) :")
        Lire (N)
        NB ← 0
        TANT QUE (N ≠ 0)  FAIRE
                NB ← NB + 1
                Ecrire  ("Entrer un nom (ou 0 pour arrêt) :")
                Lire (N)
        FINTANTQUE
        Ecrire  ("Vous avez saisi ", NB, " entier(s)")
FIN
```

5 Sélection d'une itération



6 Exercices d'application

Exercice 1

Ecrire un algorithme qui permet de saisir deux entiers M et N, puis de déterminer et d'afficher leur **PGCD**.

NB : le PGCD de M et de 0 est M et tout diviseur de M et N est aussi diviseur du reste de M par N.

Exercice 2

Ecrire un algorithme qui lit un entier positif et vérifie si ce nombre est **premier** ou non.

NB : Un nombre est dit premier lorsqu'il n'est divisible que par 1 ou par lui-même.

7 Solution des exercices

Exercice 1

```
Algorithme PGCD
Var a, b : Entier
Début
Ecrire ("Entrer la valeur de a : ") L ire(a)
Ecrire ("Entrer la valeur de b : ") L ire(b)
Tant que (b ≠ 0) Faire
    r ← a mod b
    a ← b
    b ← r
FinTantQue
Ecrire ("PGCD = ", a)
Fin.
```

Exercice 2

```
i ← 2
Verif ← Vrai
Tant que (i <= A div 2) et (verif = Vrai) Faire
        Si (A Mod i = 0) Alors
                Verif ← Faux
        Sinon
                i ← i + 1
        FinSi
FinTantQue
Si Verif alors
   Ecrire ("premier")
Sinon   Ecrire ("non premier")
FinSi
```

CHAPITRE

6

LES TABLEAUX

Objectifs spécifiques

- Comprendre l'utilisation du type tableau
- Manipuler des tableaux

Plan du chapitre

1. Introduction
2. Tableau unidimensionnel
3. Tableau bidimensionnel
4. Exercices d'application
5. Solutions des exercices

Volume horaire

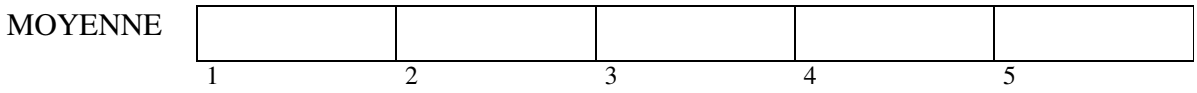
1 heure de cours intégré

Chapitre 6 Les tableaux

1 Introduction

Supposons que nous voulons calculer puis trier dans l'ordre décroissant les moyennes de cinq élèves. Nous utilisons habituellement cinq variables de type réel Moy1, Moy2, Moy3, Moy4 et Moy5. Nous remarquons que ce nombre de variables deviendra encore beaucoup plus grand si nous avons 30 élèves. Cela rendra les algorithmes très complexes. Cependant, ces variables moyennes ont toutes le même rôle (moyenne d'un élève), nous pouvons les regrouper sous le même nom **MOYENNE**.

Ce regroupement formera une nouvelle structure dite **tableau** ou **vecteur** de 5 éléments.



Moy1 correspondra au 1er élément du tableau MOYENNE

Moy2 correspondra au 2ème élément du tableau MOYENNE

…

Moy5 correspondra au 5ème élément du tableau MOYENNE

2 Tableau unidimensionnel

Un tableau à une dimension, appelé aussi **vecteur**, est une structure de données constituée d'un nombre fini d'éléments de même type et directement accessibles par leurs indices ou indexes.

2.1 Déclaration d'un tableau

Pour définir une variable de type tableau, il faut préciser :

- le *nom* (identifiant du tableau)
- *l'indice* (généralement de type entier ou caractère)
- le *type des éléments* (entier, réel, caractère, etc.)

On note :

```
Nom_tab : Tableau [BorneInf..BorneSup] de Type_éléments
```

Exemple :

```
Tnote : Tableau [1..30] de Réel
```

2.2 Accès à un élément du tableau

La syntaxe générale pour accéder à un élément d'un tableau est:

```
Nom_Tableau [Indice_De_L_Element]
```

L'indice de l'élément est une valeur comprise entre **BorneInf** et **BorneSup.**

Exemple :

Accès au 20ème élément du tableau Tnote : Tnote [20]

2.3 Opérations élémentaires sur un tableau

2.3.1 Remplissage d'un tableau

Un tableau peut être rempli *élément par élément* à l'aide d'une série d'affectations :

```
T[1] ← Valeur1
T[2] ← Valeur2
…
T[n] ← Valeurn
```

Il est également possible de lire les éléments du tableau à partir du clavier à l'aide d'une boucle « **Pour** » :

```
Pour i de 1 à n Faire
        Ecrire ("T [", i, "] : ")
        Lire (T[i])
FinPour
```

2.3.2 Affichage d'un tableau

L'affichage des éléments d'un tableau se fait également élément par élément.

```
Pour i de 1 à n Faire
        Ecrire ("T [", i, "] : ", T[i])
FinPour
```

2.3.3 Somme des éléments d'un Tableau

```
S ← 0
Pour i de 1 à n Faire
        S ← S + T[i]
FinPour
```

3 Tableau bidimensionnel

Un tableau bidimensionnel est appelé aussi **matrice**. Elle est composée de lignes et de colonnes.

3.1 Déclaration d'une matrice

```
Nom_mat : Tableau [BInfLigne..BSupLigne] [BInfCol..BSupCol] de Type_éléments
```

Exemple :

```
M : tableau [1..4] [1..5] de entier
```

M contient 4 lignes. Chaque ligne contient 5 colonnes.

3.2 Accès à un élément d'une matrice

La syntaxe générale pour accéder à un élément d'une matrice est:

```
Nom_Mat [Indice_Ligne] [Indice_Colonne]
```

Indice_Ligne est une valeur comprise entre **BInfLigne** et **BSupLigne.**

Indice_colonne est une valeur comprise entre **BInfCol** et **BSupCol.**

Exemple :

M [2] [4] désigne l'élément situé à la $2^{ème}$ ligne et la $4^{ème}$ colonne.

3.3 Opérations élémentaires sur une matrice

3.3.1 Remplissage d'une matrice

Il est nécessaire d'utiliser deux boucles imbriquées correspondant chacune à une dimension :

```
Pour i de 1 à n Faire
        Pour j de 1 à m Faire
                Ecrire ("M [", i, "] [", j, "] : ")
                Lire (M [i] [j])
        FinPour
FinPour
```

3.3.2 Transposition d'une matrice carrée

Une matrice carrée est une matrice à n lignes et n colonnes.

L'opération de transposition consiste à inverser les lignes et les colonnes en effectuant une symétrie par rapport à la diagonale principale de la matrice.

Exemple :

M

2	8	9
6	3	7
1	4	5

Devient

2	6	1
8	**3**	4
9	7	**5**

```
Pour i de 1 à n Faire
        Pour j de (i + 1) à n Faire
                aux ← M [i] [j]
                M [i] [j] ← M [j] [i]
                M [j] [i] ← aux
        FinPour
FinPour
```

3.3.3 Somme des éléments d'une matrice

```
S ← 0
Pour i de 1 à n Faire
        Pour j de 1 à m Faire
                S ← S + M [i] [j]
        FinPour
FinPour
```

3 Exercices d'application

Exercice 1

Soit T un tableau contenant n éléments de type entier. Ecrire un algorithme qui détermine le plus petit et le plus grand élément de ce tableau.

Exercice 2

Soient M1 et M2 deux matrices à n lignes et m colonnes. Ecrire un algorithme calcule les éléments de la matrice M3 tel que M3 = M1+M2

Exemple

M1

2	3	6
1	4	5

M2

0	6	8
-1	2	7

M3

2	9	14
0	6	12

4 Solution des exercices

Exercice 1

```
Algorithme mimax
Const N = 20
Var T : Tableau [1..N] d'entier
  i, min, max : entier
Début
        Pour i de 1 à N Faire
                Ecrire ("T [", i, "] : ")
                Lire (T[i])
        FinPour
        min ← T [1]
        max ← T [2]
        Pour i de 2 à N Faire
                Si T [i] < min alors
                   min ← T [i]
                Sinon Si T [i] > max alors
                   max ← T [i]
                FinSi
        FinPour
       Ecrire ("Min = ", min, "Max =", max)
Fin
```

Exercice 2

```
Algorithme somme_matrice
Type Mat = Tableau [1..20] [1..10] d'entier
Var M1, M2, M3 : Mat
  n, m, i, j : entier
Début
      Ecrire ("Nombre de lignes :") Lire (n)
      Ecrire ("Nombre de colonnes :") Lire (m)
      // remplissage des matrices
        Pour i de 1 à n Faire
          Pour j de 1 à m faire
                Ecrire ("M1 [", i, "] [", j, "]: ")
                Lire (M1 [i] [j])
                Ecrire ("M2 [", i, "] [", j, "]: ")
                Lire (M2 [i] [j])
          FinPour
        FinPour
        // calcul de sommes
        Pour i de 1 à n Faire
          Pour j de 1 à m faire
           M3 [i] [ j] ← M1 [i] [ j] + M2 [i] [ j]
          FinPour
        FinPour
        Pour i de 1 à n Faire
          Pour j de 1 à m faire
                Ecrire (M3 [i] [j])
          FinPour
        FinPour
Fin
```

CHAPITRE

7

LES CHAINES DE CARACTETRES

Objectifs spécifiques

- Manipuler des chaînes de caractères
- Construire des algorithmes qui traitent des chaînes de caractères

Plan du chapitre

1. Introduction
2. Opérations sur les chaînes
3. Exercices d'application
4. Solutions des exercices

Volume horaire

1 heure de cours intégré

Chapitre 7 Les chaînes de caractères

1 Introduction

Une chaîne de caractères est une suite de caractères. Elle est considérée comme un tableau de caractères. Elle est délimitée par deux guillemets. La chaîne ne contenant aucun caractère est appelée chaîne **vide**.

La déclaration d'une chaîne de caractères se fait suivant la syntaxe suivante:

Nom_Variable_Chaine : **chaine** [longueur]

Exemple :

Nom : chaine [25]

Pour accéder à un caractère de la chaîne, il suffit d'indiquer le nom de la chaîne suivi d'un entier entre crochets qui indique la position du caractère dans la chaîne.

En général, ch [i] désigne le $i^{ème}$ caractère de la chaîne ch.

Exemple :

Si la variable Nom contient "Moha**m**ed", alors Nom [5] renvoie "m"

2 Opérations sur les chaînes

2.1 Entrée et sortie standard de chaînes

- **Ecriture de chaînes**

Pour l'écriture de chaînes sur la sortie standard, on utilise l'instruction **écrire**.

- Lecture de chaînes

Pour la lecture des chaînes à partir de l'entrée standard, on utilise l'instruction **lire**.

Exemple:

```
ch : chaine [30]
Ecrire ("Donner une chaine de caractères:")
Lire (ch)
Ecrire ("La chaine lue est:", ch)
```

2.2 Copie d'une chaîne de caractères dans une autre

La fonction **Copier_chaine** copie le contenu d'une chaîne de caractères (chaine_source) vers une autre chaîne (chaine_destination).

La syntaxe est :

Copier_chaine (chaine_destination, chaine_source)

Exemple:

ch : chaine [30]

Copier_Chaine (ch, "Algorithme")

2.3 Longueur d'une chaîne

La fonction **Longueur_chaine** renvoie le nombre de caractères contenus dans une chaîne.

La syntaxe est :

Longueur_Chaine (chaine)

Exemple:

ch : chaine [20]

L : entier

Copier_Chaine (ch, "chapitre")

L ← Longueur_Chaine (ch)

Ecrire (L)

Ecran

8

2.4 Concaténation d'une chaîne à une autre

La concaténation de chaînes de caractères permet d'ajouter le contenu d'une chaîne de caractères à celui d'une autre chaîne. La fonction qui assure ce mécanisme est: **Concaténer_Chaine**. Sa syntaxe est :

Concaténer_Chaine (chaine_Destination, chaine_source**)**

Le résultat de la concaténation va se trouver au niveau de la chaîne destination.

Exemple:

Nom_Fichier : chaine [14]

Ext: chaine [5]

Copier_Chaine (Nom_Fichier, "Projet")

Copier_Chaine (extension, ".exe")

Concaténer_Chaine (Nom_Fichier, extension);

Écrire ("Le nom complet du fichier est: ", Nom_Fichier);

Ecran

Projet.exe

2.5 Comparaison de deux chaînes de caractères

Pour comparer deux chaînes, il y a la fonction **Comparer_Chaine**. Cette fonction admet deux paramètres chaine1 et chaine2.

Comparer_Chaine est une fonction qui retourne:

- **0** si chaine1 = chaine2

- Une valeur **positive** si chaine1 > chaine2
- Une valeur **négative** si chaine1 < chaine2.

Exemple :

```
S1, S2 : chaine [30]
Écrire ("Saisir 2 chaines")   Lire (S1, S2)
Si (Comparer_Chaine (S1, S2) = 0) alors
        Ecrire ("Les 2 chaînes sont égales")
Sinon
        Écrire ("les 2 chaînes ne sont pas égales")
Fin si
```

3 Exercices d'application

Exercice 1

Ecrire un algorithme qui lit une chaîne de caractères puis affiche son inverse.

Exemple : Si la chaîne entrée est "algo", l'algorithme doit afficher "ogla".

Exercice 2

Ecrire un algorithme qui permet de saisir une chaîne de caractère, puis de calculer et afficher le nombre de caractères majuscules et minuscules.

4 Solutions des exercices

```
Algorithme inverse   // exercice 1
Var  i, L : Entier
ch1, ch2 : Chaîne
Début
Ecrire ("Entrer une chaîne : ") L ire (ch1)
L ← Longueur_chaine (ch1)
ch2 ← ""
Pour i de 1 à L Faire
      ch2 ← ch1 [i] + ch2
FinPour
Ecrire ("Inverse de la chaîne = ", ch2)
Fin.
```

```
Algorithme compte  // exercice 2
Var  i, maj, min : Entier
ch : Chaîne
Début
Ecrire ("Entrer une chaîne : ") L ire (ch)
maj ← 0        min ← 0
Pour i de 1 à Longueur_chaine (ch1) Faire
   Selon ch [i] Faire
      "A".."Z" : maj ← maj +1
      "a".."z" : min ← min + 1
   FinSelon   FinPour
Ecrire ("nbre maj = ", maj, "nbre min=", min)
Fin.
```

CHAPITRE

8

LES ENREGISTREMENTS

Objectifs spécifiques

- Définir la structure enregistrement
- manipuler des variables de type enregistrement
- Mettre à profit la structure enregistrement pour résoudre des problèmes

Plan du chapitre

1. Définition
2. Déclaration des variables de type enregistrement
3. Manipulation des variables de type enregistrement
4. Tableaux et enregistrements
5. Exercices d'application
6. Solutions des exercices

Volume horaire

1 heure de cours intégré

Chapitre 8 Les enregistrements

1 Définition

Les variables que nous avons jusqu'à présent utilisées ne se constituent que d'un seul type de données (Entier, Réel, Caractère, etc.).

Les tableaux constituent une extension puisque nous y déclarons une variable composée de plusieurs éléments de même type.

Un **enregistrement** (ou **structure**) permet de rassembler un ensemble d'éléments de types différents sous un nom unique. On définit ainsi un **type composé**.

A titre d'exemple, une date, une adresse ou nombre complexe peuvent être considérés comme des enregistrements.

2 Déclaration d'une variable de type enregistrement

On définit le type structure comme suit:

```
Type Nom_Structure = structure
        Champ 1 : type
        …
        Champ n : type
Fin structure
```

Exemples:

Définition de la structure **Etudiant** :

```
Type Etudiant = structure
        Nom : chaine [50]
        Prenom : chaine [25]
        Adr : chaine [50]
        DateNaiss : chaine [10]
Fin structure
```

Définition de la structure **Date** :

```
Type Date = structure
        Jour : entier
        Mois : entier
        Annee : entier
Fin structure
```

Pour déclarer une variable de type structure :

```
Var nom_variable : Type_structure
```

Exemple :

Var E : Etudiant → permet de déclarer une variable E de type étudiant

Remarque :

Les champs d'une structure peuvent avoir l'un des types déjà définis y compris les **tableaux** et les **structures**.

Revenons à l'exemple de la structure « *Etudiant* » et supposons que l'adresse n'est pas une chaîne de caractères mais elle est une structure qui est définie comme suit:

Définition de la structure **Adresse** :	Définition de la structure **Etudiant** :
Type Adresse = structure	Type Etudiant = structure
Rue: chaine [50]	Nom : chaine [50]
Ville : chaine [20]	Prenom : chaine [25]
CP : entier	Adr : **Adresse**
Fin structure	DateNaiss : **Date**
	Fin structure

3 Manipulation des variables de type enregistrement

3.1 Affectation

Pour faire référence à un champ, on sépare le nom de la variable de type la structure concernée de celui du champ visé, avec l'opérateur (**.**).

```
Nom_Variable_Enreg.champ ← valeur
```

Exemple :

```
Var Adr : Adresse
Adr.Rue ← "Ibn Rochd"
Adr.Ville ← "Nabeul"
Adr.CP ← 8000
```

Remarque :

Il est possible d'affecter une variable enregistrement dans une autre à condition qu'ils aient la même structure.

Exemple :

VAR Adr1, Adr2 : Adresse

Il est possible d'écrire :

Adr1 ← Adr2 ⟺

```
Adr1.Rue ← Adr2.Rue
Adr1.Ville ← Adr2.Ville
Adr1.CP ← Adr2.CP
```

3.2 Lecture / Ecriture

La lecture des valeurs des différents champs d'une variable enregistrement se fait comme suit :

Lire (variable.champ)

L'écriture des valeurs des différents champs d'une variable enregistrement se fait comme suit :

Ecrire (variable.champ)

Exemple :

VAR Adr : Adresse

Lecture de la variable **Adr** :	Ecriture de la variable **Adr** :
Lire (Adr.Rue)	Ecrire (Adr.Rue)
Lire (Adr.Ville)	Ecrire (Adr.Ville)
Lire (Adr.CP)	Ecrire (Adr.CP)

4 Tableaux et enregistrements

Il est possible de déclarer un tableau dont les éléments sont de type enregistrement. On définit tout d'abord l'enregistrement, puis on déclare le tableau.

Exemple :

Type **Personne** = structure
    nom : chaine[30]
    num_tel : chaine[10]
    num_cb : entier
fin structure

Tab = tableau [1..100] de **Personne**

5 Exercices d'application

Exercice 1

Ecrire un algorithme qui permet de saisir et d'afficher un enregistrement nommé **LIVRE** caractérisé par : titre (chaîne de 30 c), nombre de pages (entier), nom de l'auteur (chaîne de 25 c), date d'édition (Enregistrement).

Exercice 2

Ecrire un algorithme qui permet de saisir les informations de N (5 <= N< = 35) **élèves** (Num, Nom, Prénom, Moyenne, Mention), puis d'afficher les élèves ayant une mention donnée et leur pourcentage par rapport au nombre total des élèves.

6. Solutions des exercices

Exercice 1

```
Algorithme livre
Type     t_date = structure
            jour : 1..31
            mois : 1..12
            annee : entier
       Fin structure
       LIVRE = structure
            titre : chaine [30]
            NP : entier
            Nom_Aut : chaine [25]
            Date_Ed : t_date
      Fin structure
var l : livre
Début
 Ecrire ("titre : ")  Lire (l.titre)
 Ecrire ("nbre de pages : ")  Lire (l.np)
 Ecrire ("nom de l"auteur : ") Lire (l.nom_aut)
 Ecrire ("jour : ")     Lire (l.Date_Ed .jour)
 Ecrire ("mois : ")  Lire (l.Date_Ed .mois)
 Ecrire ("année : ") Lire (l.Date_Ed.annee)
// affichage
Ecrire ("titre : ", l.titre)
Ecrire ("nbre de pages : ", l.np)
Ecrire ("nom de l"auteur : ", l.nom_aut);
Ecrire ("Date : ", l.Date_Ed.jour, "/",
l.Date_Ed.mois, "/",  l.Date_Ed.annee)
Fin
```

Exercice 2

```
Algorithme mention
Const  n = 35
type eleve = structure
            num : Entier
            Nom, Prenom : chaine[30]
            Moyenne : réel
            Mention : chaine [20]
Fin structure
tab = tableau [1..n] d'eleve
var t : tab
   ne, i : entier
   m : chaine [20]
Début
Ecrire ("donner la mention : ")   Lire (m)
ne ← 0
Pour i de 1 à  n Faire
      Si t[i].mention = m alors
         Ecrire ("n° : ", t[i].num)
         Ecrire ("nom : ", t[i].nom)
         Ecrire ("prénom : ", t[i].prenom)
         Ecrire ("moyenne : ", t[i].moyenne)
         Ecrire ("mention : ", t[i].mention)
         ne ← ne + 1
       FinSi
  FinPour
  p ← (ne / n) * 100
  Ecrire ("pourcentage : ", p)
Fin
```

CHAPITRE

9

LES ALGORITHMES DE TRI ET DE RECHERCHE

Objectifs spécifiques

- Connaître les différentes méthodes de tri.
- Utiliser les différentes méthodes de tri pour résoudre des problèmes.
- Connaître les différentes méthodes de recherches.
- Savoir choisir la méthode de recherche la plus adaptée au problème traité.

Plan du chapitre

1. Les algorithmes de tri
2. Les algorithmes de recherche
3. Exercice d'application
4. Solution de l'exercice

Volume horaire

3 Séances de cours intégré

Chapitre 9 Les algorithmes de tri et de recherche

1 Les algorithmes de tri

L'opération tri consiste à ordonner une suite de valeurs suivant une relation d'ordre (ordre croissant et décroissant).

Il y a plusieurs façons ou techniques pour effectuer un tri. On va considérer trois tris :

- tri par sélection
- tri à bulle
- tri par insertion

1.1 Tri par sélection *(par minimum)*

C'est la méthode de tri la plus simple, elle consiste à :

- chercher l'indice du plus petit élément du tableau T [1..n] et permuter l'élément correspondant avec l'élément d'indice 1
- chercher l'indice du plus petit élément du tableau T [2..n] et permuter l'élément correspondant avec l'élément d'indice 2
- …
- chercher l'indice du plus petit élément du tableau T [n-1..n] et permuter l'élément correspondant avec l'élément d'indice (n-1).

```
ALGORITHME Tri_Selection
Const n = 20
Var T : tableau [1..n] d'entier
    i, min, ind, temp : entier
DEBUT
    Pour ind de 1 à (n – 1) faire
        min ← ind
        Pour i de ind + 1 à n faire
            Si T [i] < T [min] alors
                min ← i
            FinSi
        FinPour
        temp ← T [ind]
        T [ind] ← T [min]
        T [min] ← temp
    FinPour
FIN
```

Trace d'exécution

	1	*2*	*3*	*4*	*5*
Tableau initial	6	4	2	3	5
Après la 1ère itération	2	4	6	3	5
Après la 2ème itération	2	3	6	4	5
Après la 3ème itération	2	*3*	4	*6*	5
Après la 4ème itération	2	*3*	4	5	*6*

1.2 Tri à bulles

La méthode de tri à bulles nécessite deux étapes :

- Parcourir les éléments du tableau de 1 à (n–1) ; si l'élément i est supérieur à l'élément (i+1), alors on les permute.

- Le programme s'arrête lorsqu'aucune permutation n'est réalisable après un parcours complet du tableau.

```
ALGORITHME Tri_Bulle
Const n = 20
Var T : tableau [1..n] d'entier
        i, temp : entier
        permute : booléen
DEBUT
        Répéter
                permute ← faux
                Pour i de 2 à n faire
                        Si T [i - 1] > T [i] alors
                                permute ← vrai
                                temp ← T [i - 1]
                                T [i - 1] ← T[i]
                                T [i] ← temp
                        FinSi
                FinPour
        Jusqu'à non (permute)
FIN
```

Trace d'exécution

	1	*2*	*3*	*4*	*5*
Tableau initial	6	4	3	5	2
Après le 1ère parcours	4	*3*	*5*	*2*	*6*
Après le 2ème parcours	*3*	*4*	*2*	*5*	*6*
Après le 3ème parcours	3	*2*	4	*5*	6
Après le 4ème parcours	*2*	*3*	4	5	*6*

1.3 Tri par insertion

Le principe du tri par insertion consiste à insérer un par un les éléments en les plaçant correctement: on insère le second élément du tableau dans sa place au niveau du sous tableau constitué du premier élément, on insère ensuite le troisième élément du tableau dans sa place au niveau du sous tableau constitué du premier et deuxième élément et ainsi de suite jusqu'au dernier élément.

```
ALGORITHME Tri_Insertion
Const  n = 20
Var    T : tableau [1..n] d'entier
       temp, ind, j : entier
DEBUT
   Pour ind de 2 à n faire
       temp ← T [ind]
       j ← ind
       Tant que (j – 1) > 0 et T [j-1] > temp
       faire
              T [j] ← T [j-1]
              j ← j - 1
       FinTantQue
       T [j] ← temp
  FinPour
FIN
```

Trace d'exécution

	1	*2*	*3*	*4*	*5*
Tableau initial	6	4	3	5	2
Après la 1ère itération	4	*6*	*3*	5	*2*
Après la 2ème itération	*3*	*4*	6	*5*	*2*
Après la 3ème itération	*3*	*4*	*5*	*6*	*2*
Après la 4ème itération	2	*3*	4	5	*6*

2 Les algorithmes de recherche

La recherche d'une information est une opération fréquemment rencontrée dans les traitements des suites de valeurs. Il y a deux types de recherche dans un tableau :

- recherche *séquentielle*
- recherche *dichotomique*

2.1 Recherche séquentielle

La recherche séquentielle consiste à comparer l'élément à chercher « **x** » aux différents éléments du tableau jusqu'à trouver x ou atteindre la fin du tableau.

```
Algorithme Recherche_Seq
Const  n = 40
Var    T : tableau [1..n] d'entier
       x, i : entier
DEBUT
       Lire (x)
       i ← 0
       Répéter
              i ← i + 1
       Jusqu'à (T[i] = x) ou (i = n)
       Si (T[i] = x) Alors
          Ecrire ("Indice = ", i)
       Sinon
           Ecrire ("Elément introuvable…")
       FinSi
FIN
```

2.2 Recherche dichotomique

Hypothèse : le tableau doit être trié

Le principe consiste On regarde l'élément au milieu du tableau :

- S'il est égal à la valeur cherchée, l'élément cherché est existant.
- S'il est inférieur à la valeur recherchée, il ne reste à traiter que la moitié droite du tableau
- S'il est supérieur à la valeur recherchée, il ne reste à traiter que la moitié gauche du tableau

On continue ainsi la recherche en diminuant à chaque fois de moitié le nombre d'éléments du tableau restants à traiter.

```
ALGORITHME Recherche_Dich
Const  n = 20
Var    T : tableau [1..n] d'entier
       x, binf, bsup, mil : entier
DEBUT
       Écrire ("Donner la valeur à rechercher:")  lire (x)
       binf ← 1
       bsup ← n
       Répéter
            mil ← (binf + bsup) DIV 2
            // il s'agit d'une division entière
            Si T[mil] > x alors
                  bsup← mil – 1
            Sinon
                  binf ← mil + 1
            FinSi
       Jusqu' à (T[mil] = x) ou (bsup < binf)
       Si T[mil] = x alors
            Écrire (x, "EXISTE ")
       Sinon
            Écrire (x, "N'EXISTE PAS ")
       FinSi
FIN
```

3 Exercice d'application

Etant donné un tableau A de n nombres triés par ordre croissant. Ecrire un algorithme qui permet de lire un entier R et l'insérer dans sa bonne position.

Exemple : R = 6

	1	2	3	4	5	6
A	3	4	5	8	10	
A devient	3	4	5	6	8	10

4 Solution de l'exercice

Algorithme inserer

Const nmax = 40

Var A : tableau [1..nmax] d'entier

R, i, j, n : entier

DEBUT

Ecrire ("Taille : ") Lire (n)

Lire (T [1])

Pour i de 2 à n faire

Répéter

Lire (T [i])

Jusqu'à (T [i] > T [i-1])

FinPour

Lire (R)

n ← n + 1

T [n] ← R

j ← n

Tant que (j – 1) > 0 et T [j - 1] > R faire

T [j] ← T [j - 1]

j ← j - 1

FinTantQue

T [j] ← R

FIN

CHAPITRE

10

PROCEDURES ET FONCTIONS

Objectifs spécifiques

- Décomposer un problème en sous-problèmes élémentaires.
- Présenter les solutions sous forme de procédures et fonctions.

Plan du chapitre

1. Introduction
2. Définition et syntaxe
3. Modes de passage de paramètres
4. Exercices d'application
5. Solutions des exercices

Volume horaire

2 Séances de cours intégré

Chapitre 10 Procédures et fonctions

1 Introduction

L'analyse modulaire consiste à **diviser** un problème en sous-problèmes de **difficultés moindres**. Ces derniers peuvent être divisés à leur tour en d'autres sous-problèmes jusqu'à ce qu'on arrive à un niveau abordable de difficulté. Pour chacun des sous-problèmes, on écrit un module appelé **sous-programme**.

Ainsi, la résolution du problème sera composée d'un algorithme principal et d'un certain nombre de sous-programmes. L'algorithme principal a pour but d'organiser l'enchaînement des sous-programmes.

L'intérêt de l'analyse modulaire est:

- Répartir les difficultés entre les différents sous problèmes
- Faciliter la résolution d'un problème complexe
- Améliorer la qualité d'écriture du programme principal
- Minimiser l'écriture du code source dans la mesure où on utilise la technique de la réutilisation.

2 Définition et syntaxe

2.1 Définition

- Un *sous-programme* est une unité fonctionnelle formée d'un bloc d'instructions et éventuellement paramétré, que l'on déclare afin de pouvoir l'appeler par son nom en affectant des valeurs à ses paramètres (s'ils existent). Les données fournies au sous-programme et les résultats produits par ce dernier sont appelés des **arguments** ou des **paramètres**. Un sous-programme peut être une **procédure** ou une **fonction**.

- Une *procédure* est un sous-programme ayant un nombre de paramètres, contenant un certain nombre d'instructions et admettant zéro ou plusieurs résultats.

- Une *fonction* est un sous-programme ayant un nombre de paramètres, contenant un certain nombre d'instructions et admettant au maximum un résultat unique affecté à son nom.

2.1 Syntaxe

La syntaxe de définition d'une fonction est:

```
FONCTION Nom_Fonction (Paramètres_Formels) : Type_Valeur_Retour
```

```
// Déclaration des variables
Début
        Instructions
        Retourner Résultat
Fin
```

Exemple : Une fonction « min » qui retourne le minimum de 2 entiers a et b.

```
Fonction min (a, b : Entier) : Entier
Var
        c : Entier
Début
        Si (a <= b) Alors
                c ← a
        Sinon
                c ← b
        FinSi
        Retourner c
Fin
```

L'appel d'une fonction se fait comme suit:

```
        Var_Résultat ← Nom_Fonction (Parametres_Effectifs)
```

Exemples :

```
        x ←3
        y ←5
        z ←min (a, b)
        ou bien z ← min (3, 5)
```

La syntaxe de définition d'une procédure est:

```
PROCEDURE Nom_Procédure (Paramètres_Formels)
// Déclaration des variables
Début
        Instructions
Fin
```

Exemple : Afficher un caractère un certain nombre de fois.

Procédure Affiche (c : Caractère, n : Entier)

Var

i : Entier

Début

Pour i de 1 à n Faire

Ecrire(c)

FinPour

Fin

L'appel d'une procédure se fait comme suit:

```
Nom_Procedure (Parametres_Effectifs)
```

Exemple : l'appel de procédure ReptCar ("*",6) doit afficher "******".

Remarques :

- Les paramètres spécifiés dans la définition d'un sous-programme sont qualifiés de paramètres *formels*, par contre les paramètres qui seront transmis à un sous-programme lors de l'appel sont appelés des paramètres *effectifs*.

- Les paramètres formels et les paramètres effectifs doivent correspondre en *nombre*, en *type* et en *ordre*. Les noms peuvent se différer.

3 Modes de passage de paramètres

La substitution des paramètres effectifs aux paramètres formels s'appelle **passage de paramètres**. Elle correspond à un transfert d'informations entre le programme ou le module *appelant* et la procédure *appelée*.

Notons d'abord que les paramètres d'une procédure peuvent être de type **donnée**, **résultat** ou **donnée-résultat**.

* Un paramètre **donné** est une information nécessaire pour réaliser une tâche. Il ne subit aucun changement au cours de l'exécution du sous-programme.

* Un paramètre **résultat** est un aboutissement de l'action, sa valeur n'est pas significative avant le début de l'exécution de la procédure.

* Un paramètre **donnée-résultat** est à la fois une donnée et un résultat, c'est à dire une information dont la valeur sera modifiée par la procédure.

Le tableau suivant indique pour chaque type de paramètre le sens de transfert et le mode de passage utilisé :

Type de paramètre	Sens de transfert	Mode de passage
Donnée	Programme → Procédure	Par valeur
Résultat	Programme ← Procédure	Par variable (adresse)
Donnée-Résultat	Programme ←→ Procédure	

En algorithmique, le mot **Var** inséré devant un paramètre formel signifie que ce paramètre est passé par variable (paramètre de type résultat ou donnée-résultat).

Exemple : une procédure « permut » qui permet d'échanger les valeurs de 2 entiers a et b.

Procédure permut (**Var** a, b : Entier)

Var x : Entier

Début

x ← a

a ← b

b ← x

Fin

4 Exercices d'application

Exercice 1

Ecrire un algorithme qui détermine puis affiche le nombre de combinaisons de p objets parmi n (n et p sont deux entiers naturels strictement positifs (avec $n \geq p$)).

$$C_p^n = \frac{n!}{p!(n-p)!}$$

Exercice 2

Ecrire une fonction qui détermine puis affiche X^n (avec X de type réel, n entier positif).

5 Solution des exercices

Exercice 1

Algorithme comb

Var n, p, f1, f2, f3, c : entier // Variables globales

Fonction fact (n : entier) : entier

Var f, i : entier // Variables locales

Début

f ← 1

```
  Pour i de 1 à n Faire
     f ← f * i
  FinPour
  Retourner (f)
Fin
Procédure saisie (var x, y : entier)
Début
   Répéter
        Ecrire ("donner un entier n : ")        Lire (n)
        Ecrire ("donner un entier p : ")        Lire (p)
    Jusqu'à (n >= p) Et (p > 0)
Fin
Début
    saisie (n, p)
    f1 ← fact (n)
    f2 ← fact (p)
    f3 ← fact (n-p)
    c ← f1 div (f2*f3)
   Ecrire ("Le nombre de combinaison de ", p, " objets parmi ", n, " est : ", c)
Fin.
```

Exercice 2

```
Fonction Puissance (x : réel, n : entier) : réel
Var i : entier
    p : réel
Début
  p ←1
  Pour i de 1 à n Faire
     p ← p * x
 FinPour
  Retourner (p)
Fin
```

CHAPITRE

11

Récursivité

Objectifs spécifiques

- Définir la récursivité.
- Résoudre des problèmes faisant appel à la récursivité.

Plan du chapitre

1. Notion de récursivité
2. Etude d'un exemple : la fonction factorielle
3. Choix entre itération et récursivité
4. Récursivité terminale
5. Récursivités multiple et imbriquée
6. Récursivité croisée
7. Exercices d'application
8. Solutions des exercices

Volume horaire

2 Séances de cours intégré

Chapitre 11 Récursivité

1 Notion de récursivité

La récursivité est une technique de programmation alternative à l'itération qui permet de trouver, lorsqu'elle est bien utilisée, des solutions très élégantes à un certain nombre de problèmes.

- Un programme est dit **récursif** s'il s'appelle lui-même.
- Un programme récursif est donc forcément une **fonction** ou une **procédure** (il doit pouvoir s'appeler).
- Puisqu'un programme récursif s'appelle lui-même, il est impératif qu'on prévoie une **condition d'arrêt** à la récursion, sinon le programme ne s'arrête jamais !
- Utiliser la récursivité revient à :
  - Traiter le ou les cas particuliers représentés par une ou plusieurs **conditions d'arrêts**
  - Traiter le ou les cas généraux représentés par les **appels récursifs**

Forme générale d'une Procédure	**Forme générale d'une fonction**
Procédure **récursive** (paramètres) DEBUT SI (condition d'arrêt) ALORS Traitement du point d'arrêt SINON instructions {optionnel} **récursive** (paramètres changés) instructions {optionnel} FINSI FIN	Fonction récursive (paramètres) : type_retour DEBUT SI (condition d'arrêt) ALORS instructions {optionnel} **retourner** (résultat) SINON instructions {optionnel} **retourner récursive** (paramètres changés) FINSI FIN

2 Etude d'un exemple : la fonction factorielle

Solution itérative

```
Fonction Factorielle1 (n : Entier) : Entier
Var
        i, f : Entier
Début
        f ←1
        Pour i de 2 à n Faire
                f ← f * i
        FinPour
        Retourner (f)
Fin
```

Solution récursive

Pour trouver la factorielle, on peut écrire :

$$\begin{cases} \textbf{factorielle}\ (n) = n * \textbf{factorielle}\ (n-1) \\ \textbf{factorielle}\ (0) = 1 \end{cases}$$

Cette relation donne la fonction récursive suivante :

Fonction **Factorielle2** (n : Entier) : Entier

Début

Si (n = 0) Alors

Retourner (1)

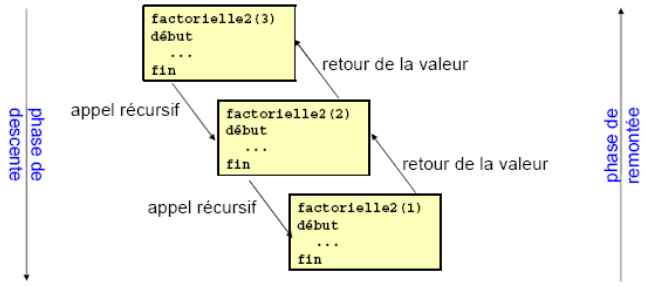
Sinon

Retourner (n * **Factorielle2** (n - 1))

FinSi

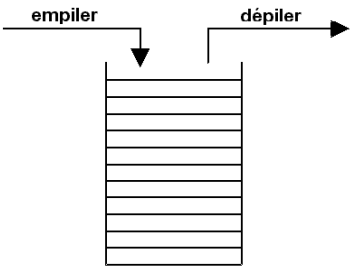
Fin

Appel récursif :



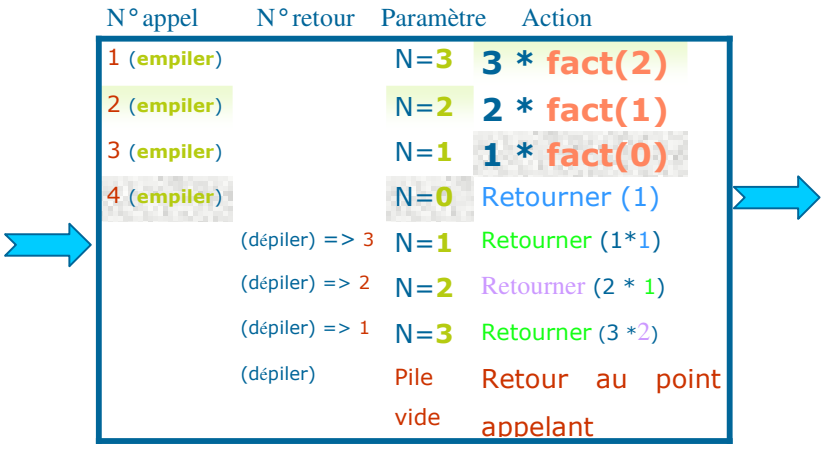
Chaque appel récursif provoque la sauvegarde des valeurs des paramètres d'appel dans la ***pile d'exécution***.

Pile d'exécution



L'empilement des appels ne doit pas excéder la **taille** de la pile (si la pile est pleine et qu'un nouvel appel est lancé, il y a **débordement** de pile (Stack Overflow).

Exécution de factorielle (3) :



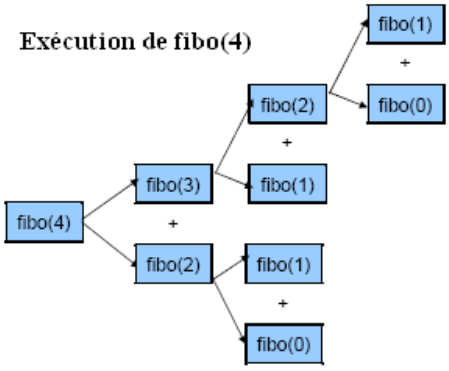
3 Choix entre itération et récursivité

- L'exécution d'une version récursive d'un programme est moins rapide que celle de la version itérative, même si le nombre d'instructions est le même (à cause des appels récursifs).

- La version récursive d'un algorithme peut parfois conduire à exécuter bien plus d'instructions que la version itérative.

Exemple : calcul des termes de la suite de Fibonacci (certains termes de la suite seront calculés plusieurs fois).

```
Fonction fibo (n : entier) : entier
Début
        Si ((n = 1) ou (n = 0)) alors
                Retourner 1
        Sinon
                retourner fibo (n - 1) + fibo (n - 2)
        FinSi
Fin
```



- Ecrire un programme sous forme récursive est souvent plus facile et rapide que sous forme itérative, en particulier pour des calculs récursifs, ou lorsqu'on utilise des structures de données récursives (listes, arbres).

Conclusion : ne pas s'interdire d'écrire en récursif pour se faciliter la vie, mais garder un œil sur l'efficacité.

Remarque : on peut toujours (théoriquement) *dérécursiver* un algorithme récursif, c'est-à-dire le transformer en algorithme itératif, mais ce n'est pas toujours facile.

4 Récursivité terminale

Un algorithme est dit **récursif terminal** si aucun traitement n'est effectué à la remontée d'un appel récursif (sauf le retour de la valeur).

Contre exemple : forme récursive non terminale de la factorielle

```
Fonction factorielle (entier n) : entier
Début
        Si (n = 1) alors retourner 1
        Sinon retourner n*factorielle (n-1)
        FinSi
Fin
```

Exemple : forme récursive terminale de la factorielle

```
Fonction factorielle (n : entier, resultat : entier) : entier
Début
        Si (n = 1) alors retourner resultat
        Sinon retourner factorielle (n-1, n * resultat)
        FinSi
Fin
// Factorielle (4,1) renvoie la factorielle de 4
```

- Un *algorithme récursif terminal* est en théorie plus efficace (mais souvent moins facile à écrire) que son équivalent non terminal : il n'y a qu'une phase de descente et pas de phase de remontée.
- En *récursivité terminale*, les appels récursifs n'ont pas besoin d'êtres empilés car l'appel suivant remplace simplement l'appel précédent dans le contexte d'exécution.

5 Récursivités multiple et imbriquée

- Le calcul de la suite de Fibonacci requiert une récursivité **multiple** (plusieurs appels récursifs dans la fonction).
- On peut également faire de la récursivité **imbriquée** : un appel récursif fait alors appel à un autre appel récursif

Exemple : la suite d'Ackerman

A (m, n) = n + 1 si m = 0,

A (m, n) = A (m - 1, 1) si n = 0 et m > 0

A (m, n) = A (m - 1, A (m, n - 1)) sinon

```
Fonction Ackerman (m : entier, n : entier) : entier
Début
        Si (m = 0) alors retourner n+1
        Sinon
        Si ((m>0) et (n=0)) alors retourner Ackerman (m - 1, 1)
        Sinon retourner Ackerman (m - 1, Ackerman (m, n - 1))
        FinSi
        FinSi
Fin
```

6 Récursivité croisée

On peut écrire des algorithmes récursifs où les appels récursifs se croisent d'une fonction à l'autre, c'est la récursivité **croisée**.

Exemple :

// cette fonction renvoie vrai si l'entier est pair, faux sinon. on suppose que n >= 0 fonction **estPair** (n : entier) : booléen début si (m = 0) alors retourner VRAI sinon retourner **estImpair** (n-1) FinSi fin	// cette fonction renvoie vrai si l'entier est impair, faux sinon. on suppose que n >= 0 fonction **estImpair** (n : entier) : booléen début si (m = 0) alors retourner FAUX sinon retourner **estPair** (n-1) FinSi fin

7 Exercices d'application

Exercice 1

Ecrire une fonction récursive qui retourne le PGCD de deux entiers A et B en utilisant l'algorithme d'Euclide qui s'appuie sur les propriétés suivantes :

PGCD (A, B) = A Si B = 0 PGCD (A, B) = PGCD (B, A mod B) Sinon

Exercice 2

Ecrire une fonction récursive qui vérifie si une chaîne de caractères est un **palindrome** ou non.

Exemple : "radar" est une chaine palindrome.

8 Solutions des exercices

Exercice 1

Fonction PGCD_Euc (a, b : Entier) : Entier

Début

Si (b = 0) alors

    Retourner (a)

Sinon

    Retourner (PGCD_Euc (b, a mod b))

FinSi

Fin

Exercice 2

Fonction Palindrome (ch : Chaîne) : Booléen

Début

Si (Longueur_chaine (ch) < 2) Alors

    retourner (Vrai)

Sinon si (ch [1] = ch [longueur_chaine (ch)]) alors

    Retourner (Palindrome (copie (ch, 2, longueur_chaine (ch) – 2)))

Sinon

    Retourner (Faux)

FinSi

Fin

Travaux Dirigés

TD n° 1 : Types de données, Variables et Constantes

Exercice 1

Soient : **A = 4; B = 6 ; C = 3**, remplir le tableau suivant :

Instruction	Type	Valeur (S)
S ← A + B		
S ←11 mod 3 + 5.2		
S ← (A ≤ B) ET (C ≠ B)		
S ← (A+B*C Mod A)		
S ← A+B/C*5+B		
S ← NON (12 ≠ 3 * 16.8 / 4) ET VRAI		

Exercice 2

Quelles seront les valeurs des variables A, B et C après l'exécution de chacune des instructions suivantes :

A ← 5

B ← 3

C ← A + B

A ← 2

C ← B – A

Exercice 3

Quelles seront les valeurs des variables X, Y et Z après l'exécution de chacune des instructions suivantes :

X ← 3

Y ← 10

Z ← X + Y

Y ← X + Y

X ← Z

Exercice 4

Quelles seront les valeurs des variables X et Y après l'exécution de chacune des instructions suivantes :

X ← 5

Y ← 7

X ← 8

Y ← X

Les instructions : X ← Y, Y ← X

Permettent-elles d'échanger les valeurs des 2 variables X et Y ?

Exercice 5

Donner la trace de l'algorithme suivant :

0. **début** séquence
1. a ← 15>10
2. b ← a et (10≥20)
3. c ← 3
4. a ← (c ≤ 3) et a et b
5. c ←(c mod 2) = (4 div c)
6. **fin** séquence

Séq	a	b	c
1			
2			
3			
4			
5			

Donner le type de a :……

TD n° 2 : Les instructions simples

Exercice 1

Quels sont les résultats produits par l'algorithme suivant :

```
Algorithme ABC
VAR
Val, Double : entier
Début
    Val ← 231
    Double ← Val*2
    Ecrire (Val)
    Ecrire (Double)
Fin
```

Exercice 2

Ecrire un algorithme qui permet de lire un entier composé de trois chiffres puis d'afficher son complément à dix en calculant le complément à dix de ses chiffres.

Exemple : N = 126

Résultat : 984 car le complément à 10 de 1 est 9, le complément de 2 à 10 est 8 et le complément de 6 à 10 est 4.

Exercice 3

Ecrire un algorithme qui permet de :

- lire un entier positif **n** de 2 chiffres
- lire un entier positif **p** de 3 chiffres
- former et afficher un entier **res** de 5 chiffres à partir des deux entiers **n** et **p** et ceci en intercalant l'entier **p** entre les deux chiffres de **n**.

Exemple : **n** = 14 et **p** = 258, l'entier **res** sera égal à 1**258**4

Exercice 4

Ecrire un algorithme qui lit :

- le prix (PRIXHT) hors taxe d'un article
- le nombre d'articles

- le taux de la TVA

Et qui affiche le prix (PRIXTTC) toute taxes comprises.

Exercice 5

Modifier l'algorithme de l'exercice précédent pour que les résultats soient affichés comme suit:

Prix Hors Taxes :

Nombre d'articles :

Taux de la TVA :

Prix TTC :

Exercice 6

Ecrire un algorithme qui saisit un entier **N** de trois chiffres, puis détermine le nombre correspondant écrit à l'envers.

Exemple : **123** devient **321**

Exercice 7

Ecrire un algorithme qui convertit une distance **d** exprimée en pouces en son équivalents mètres et affiche le résultat. (1 pouce = 2.54 cm)

TD n° 3 : Les structures conditionnelles

Exercice 1

A, B, C étant des variables numériques, on considère la séquence algorithmique suivante :

```
DEBUT
    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
        C ← B + C
      FINSI
      Ecrire (A, B, C)
FIN
```

Donner les valeurs de A, B et C après exécution de cette séquence, dans les deux cas suivants :

1er Cas : A = 4 B = 1 C = 4

2ème Cas : A = 4 B = 0 C = 4

Exercice 2

Les tarifs d'affranchissement d'une lettre sont les suivants :

- en dessous de 20 g : 180 millimes
- à partir de 20 g, mais au dessous de 50g : 350 millimes
- à partir de 50g : 650 millimes

Ecrire un algorithme qui calcule le tarif d'affranchissement selon le poids.

Exercice 3

Ecrire un algorithme qui demande trois nombres à l'utilisateur et qui affiche le texte « Le plus petit est : », suivi de la valeur du plus petit des trois nombres.

Exercice 4

Ecrire un algorithme qui permet de résoudre l'équation ax + b = 0 dans l'ensemble des réels : distinguer les différents cas.

Exercice 5

Ecrire un algorithme qui permet de calculer la moyenne d'un élève ayant passé un concours de 3 matières à coefficients égaux (coef = 1). En fin il affichera « refusé » si sa moyenne < 10 ou « admis avec mention bien » si sa moyenne >= 14 ou « admis avec mention assez bien » si sa moyenne >= 12 ou « admis avec mention passable » si sa moyenne < 12

Exercice 6

Ecrire un algorithme qui permet de résoudre l'équation $ax^2 + bx + c = 0$

Exercice 7

Ecrire un algorithme qui lit un caractère au clavier puis affiche s'il s'agit d'une lettre minuscule, d'une lettre majuscule, d'un chiffre ou d'un caractère spécial.

TD n° 4 : Les structures itératives

Exercice 1

Ecrire un algorithme qui permet d'afficher tous les entiers de trois chiffres de la forme **cdu** tel que, pour chaque entier, la somme de ses chiffres (**c + d + u**) est un diviseur du produit des chiffres (**c * d * u**).

Exemple : l'entier **514** vérifie cette propriété, en effet, (5+1+4)=10 est un diviseur de (5*1*4)=20

Exercice 2

Ecrire un algorithme fournissant tous les nombres de trois chiffres, tel que la somme des cubes de ses chiffres soit égale à lui-même.

Exemple: $153 = 1^3 + 5^3 + 3^3$

Exercice 3

Ecrire un algorithme permettant d'afficher tous les entiers naturels formés de quatre chiffres dont la somme donnera un entier d'un seul chiffre.

Exemple : La somme des chiffres de l'entier 2004 est égale à 6 (2 + 0 + 0 + 4 = 6) et par conséquent l'entier 2004 sera affiché.

Exercice 4

Ecrire un algorithme qui lit un entier positif n puis affiche tous ses diviseurs.

Exercice 5

Ecrire un algorithme qui calcule le PPCM (Plus Petit Commun Multiple) de 2 entiers A et B en utilisant la méthode suivante :

- Permuter, si nécessaire, les données de façon à ranger dans A le plus grand des 2 entiers
- Chercher le plus petit multiple de A qui est aussi multiple de B.

Exemple : PPCM (6,8) = PPCM (8,6) = 24.

Exercice 6

Ecrire un algorithme qui permet de lire un entier naturel N, puis de déterminer et d'afficher l'entier S formé par la somme des chiffres de N.

Exemple: Pour N = 2012, la valeur de S est 2 + 0 + 1 + 2 = 5

Exercice 7

Ecrire un algorithme qui calcule et affiche les n premiers termes de la suite de Fibonacci.

La suite de Fibonacci est définie par :

- $F_0 = 1$
- $F_1 = 1$
- $F_n = F_{n-2} + F_{n-1}$ pour $n > 1$

Exercice 8

Un nombre **parfait** est un nombre présentant la particularité d'être égal à la somme de tous ses diviseurs, excepté lui-même.

Le premier nombre parfait est 6 = 3 + 2 + 1.

Ecrire un algorithme qui affiche tous les nombres parfaits inférieurs à N (N > 100).

Exercice 9

```
Algorithme Inconnu
Var    I, S, N : Entier
Début
        Lire (N)
        S ←0
        Pour I De 0 A N Pas 2 Faire
                S ← S + I
        FinPour
        Ecrire (S)
Fin
```

Reprendre cet algorithme en utilisant la structure itérative Tant Que.

TD n° 5 : Les tableaux

Exercice 1

Ecrire un algorithme qui affiche les éléments d'un tableau T de N entiers (3 < N < 20) dans l'ordre des indices décroissants (de droite à gauche).

Exercice 2

Ecrire un algorithme qui calcule le nombre d'occurrences de la valeur **val** dans un tableau **T** de N entiers (3 < N < 20).

Exercice 3

Ecrire un algorithme qui calcule le maximum et le minimum des éléments d'une matrice.

Exercice 4

Ecrire l'analyse d'un programme permettant de déterminer et d'afficher tous les diviseurs ainsi de tous les multiples d'un entier p donné, dans une partie d'un tableau **T** de **N** entiers donnés. Cette partie est délimitée par deux indices **Ind_inf** et **Ind_sup**.

Avec (0 < Ind_inf < Ind_sup ≤ N).

Exemple :

T	25	32	**43**	**4**	**32**	**72**	**80**	**15**	**24**	**2**	**48**	**56**	10	14
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
			▲							▲				
			Ind_inf								Ind_sup			

Pour Ind_inf = 3, Ind_sup = 10 et p = 8

L'algorithme affichera :

- Les diviseurs de 8 sont : 4 2
- Les multiples de 8 sont : 32 72 80 24

Exercice 5

Ecrire l'analyse d'un programme, qui à partir d'un tableau **T** de **N** entiers (5 <= N <= 30), affiche toutes les séquences strictement croissantes, de ce tableau, ainsi que leur nombre.

Exemple : Pour un tableau T de 15 éléments

1	2	5	3	12	25	13	8	4	7	24	28	32	11	14
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Les séquences strictement croissantes sont :

(1,2,5), (3,12,25), (13), (8), (4,7,24,28,32),(11,14).

Le nombre de séquences est 6.

Exercice 6

Ecrire un algorithme qui effectue la mise à 0 de la diagonale principale d'une matrice carrée d'ordre N.

Exercice 7

Ecrire un algorithme qui multiplie une matrice de **m** lignes et **n** colonnes par un réel X donné.

Exercice 8

Ecrire un algorithme qui remplit et affiche un tableau à deux dimensions contenant les **n** premières lignes du *triangle de Pascal*.

Chaque élément du triangle de pascal est obtenu par la formule :

T [1, 1] = 1 T [n, n] = 1

T [L, C] = T [L-1, C] + T [L-1, C-1]

Exemple : n = 6

T	1	2	3	4	5	6
1	1					
2	1	1				
3	1	2	1			
4	1	3	3	1		
5	1	4	6	4	1	
6	1	5	10	10	5	1

Exercice 9

Ecrire un algorithme qui calcule le nombre d'apparition d'un entier **X** dans une matrice carrée d'ordre **N**.

TD n° 6 : Les chaînes de caractères

Exercice 1

Ecrire un algorithme qui compte les occurrences d'une lettre dans un mot.

Exemple : la lettre "I" apparaît deux fois dans le mot "INFORMATIQUE".

Exercice 2

Ecrire un algorithme qui calcule la position d'une lettre dans un mot. Si la lettre est absente on affiche la valeur 0.

Exercice 3

On appelle **bigramme** une suite de 2 lettres. Ecrire un algorithme qui calcule le nombre d'occurrences d'un bigramme donné dans une chaîne de caractères.

Exercice 4

Ecrire un algorithme qui permet de tester si un mot est un palindrome ou non. Un mot est dit palindrome s'il se lit de la même manière de gauche à droite ou de droite à gauche.

Exemple : LAVAL, AZIZA, AFIFA, …

Exercice 5

Ecrire un algorithme qui permet de compter le nombre de mots dans une phrase. On suppose que la phrase commence obligatoirement par une lettre et les mots sont séparés par un seul espace.

Exercice 6

Ecrire un algorithme qui détermine et affiche le mot le plus long dans une phrase donnée.

TD n° 7 : Les enregistrements

Exercice 1

Calculer et afficher, au moyen de type enregistrement, la somme de deux nombres complexes $C1(a+ib) et\ C2(c+id)$ tout en utilisant la formule suivante :

Addition : $(a+ib)+(c+id)=(a+c)+(b+d)i$

Exercice 2

Calculer et afficher, au moyen de type enregistrement, la soustraction de deux nombres complexes $C1(a-ib) et\ C2(c-id)$ tout en utilisant la formule suivante :

Soustraction : $(a+ib)-(c+id)=(a-c)+(b-d)i$

Exercice 3

On désire informatiser la gestion d'une bibliothèque. Ainsi, pour chaque livre on dispose des informations suivantes :

- Le nom du livre ''nom_livre''
- Le nom de l'auteur ''nom_auteur''
- Le code du livre ''code_livre''

1- Ecrire un algorithme qui permet la saisie et l'affichage de l'enregistrement d'un seul livre.

2- Supposons que le nombre de livres dans une bibliothèque est égal à N ($5 \leq N \leq 27$). Refaire 1-.

Exercice 4

On suppose que le nombre des élèves dans la classe 4ème science de l'informatique est N avec $20 \leq N \leq 40$.

Ecrire un algorithme qui permet de ranger dans un tableau les informations nécessaires pour chaque élève (nom, âge, moyenne) puis :

- d'afficher le nom et la moyenne du 1er de la classe
- d'afficher la moyenne M de la classe ainsi que le nombre et les informations des élèves ayant une moyenne supérieure à M

TD n° 8 : Les algorithmes de tri et de recherche

Exercice 1

Ecrire un algorithme qui permet de trier par ordre décroissant les éléments d'un tableau **A** de n entiers positifs dans un nouveau tableau B de même dimension.

n étant un entier vérifiant 5 < n < 25.

On utilisera la démarche suivante :

1. chercher le maximum de A
2. placer ce maximum dans B
3. remplacer le maximum par –1 dans A
4. refaire les étapes 1, 2 et 3 jusqu'à ce que le tableau A soit entièrement composé de –1.

Exercice 2

Soient un tableau **TM** de taille **N** contenant des réels représentant les **moyennes** des élèves d'une classe, et un tableau **VN** contenant leurs **noms** (avec 5 < N <= 20).

On désire remplir les deux tableaux, ensuite trier, dans le même tableau TM, uniquement les moyennes >= 10 dans l'ordre **décroissant** tout en gardant celles inférieures à 10 à leurs places. On devra utiliser la méthode de tri par **sélection**. Finalement, on affichera les noms ainsi que les moyennes des 3 élèves ayant les moyennes >= 10 les plus élevées.

Dans le cas où il y a moins de 3 moyennes qui sont >= 10, afficher seulement celles qui le sont.

Exemple : Pour N = 8 et les deux tableaux suivants :

TM	14.53	**9.57**	**8.34**	10.13	16.59	**8.91**	12.83	**9.03**
	1	2	3	4	5	6	7	8

VN	"Ahmed"	**"Slim"**	**"Ali"**	"Rim"	"Afef"	**"Sami"**	"Safa"	**"Imed"**
	1	2	3	4	5	6	7	8

Après exécution du programme, les tableaux deviennent :

TM	16.59	**9.57**	**8.34**	14.53	12.83	**8.91**	10.13	**9.03**
	1	2	3	4	5	6	7	8

VN	"Afef"	**"Slim"**	**"Ali"**	"Ahmed"	"Safa"	**"Sami"**	"Rim"	**"Imed"**
	1	2	3	4	5	6	7	8

Et le programme affichera :

Rang N° 1 : Afef **Moyenne** = 16.59

Rang N° 2 : Ahmed **Moyenne** = 14.53

Rang N° 3 : Safa **Moyenne** = 12.83

Exercice 3

Ecrire un algorithme qui permet de dire si un élément **X** existe dans une matrice d'entiers ou non.

Exercice 4

Soit T un tableau de N lettres minuscules (6 <= N <=100) et soient D et N deux entiers qui répondent aux conditions suivantes :

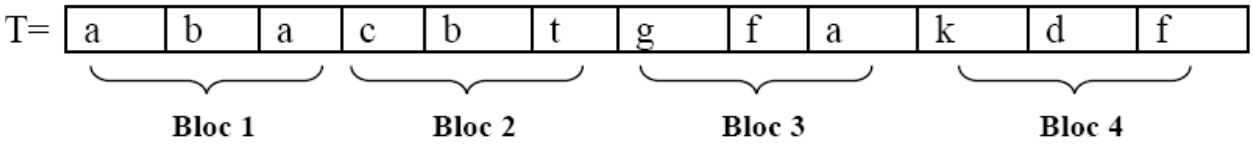
1. D est un entier diviseur de N strictement supérieur à 1.
2. M est un entier tel que N = M * D.

On se propose de trier les D éléments des M blocs disjoints qui constituent le tableau T.

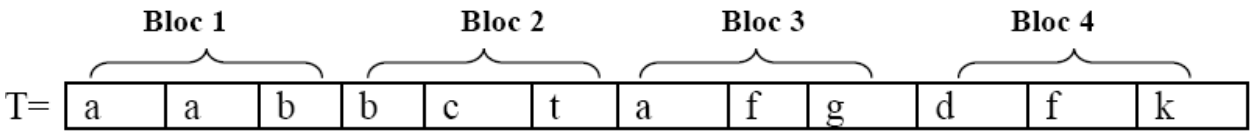
Ecrire un algorithme permettant de :

- lire les deux entiers N et D qui répondent aux conditions 1 et 2.
- Remplir le tableau T par N lettres minuscules.
- Trier dans l'ordre croissant, les éléments de chaque bloc du tableau T.
- Afficher le tableau T après le tri.

Exemple : Si N = 12 et D = 3 (donc M=4) et que le tableau T contient les éléments suivants :



Après le tri de chacun des éléments des blocs, le tableau T sera égal à :



TD n° 9 : Procédures et fonctions

Exercice 1

1. Ecrire une fonction qui prend en paramètre un entier et retourne combien de fois il est divisible par 2.
2. Ecrire un algorithme qui lit un entier et affiche le nombre de fois il est divisible par 2.

Exercice 2

1. Ecrire une procédure qui permet de remplir un tableau.
2. Ecrire une fonction qui retourne vrai si un tableau est trié et Faux sinon.
3. Ecrire un algorithme qui lit les valeurs d'un tableau et affiche si ce tableau est trié ou non

Exercice 3

Ecrire une fonction qui prend en paramètre une chaîne et un caractère et retourne la dernière apparition du caractère dans la chaîne, elle retourne 0 si le caractère n'existe pas.

Exercice 4

Ecrire un algorithme qui permet de vérifier si un tableau unidimensionnel d'entiers et de 10 éléments saisis au clavier est bien symétrique ou non.

Exemple :

3	5	6	9	8	8	9	6	5	3

: Tableau symétrique

4	6	8	7	4	2	6	9	6	5

: Tableau non symétrique

L'algorithme utilisera les sous programmes suivants :

- Une procédure permettant la saisie des éléments du tableau.
- Une fonction booléenne « Symétrique » qui pour un tableau donné retourne s'il est symétrique ou non.

Exercice 5

Soit un tableau T1 de n éléments ($1 \leq n \leq 100$). Les éléments de T1 sont des entiers naturels de trois chiffres. On se propose de remplir un tableau T2 de la façon suivante :

T2 [i] est égal à la somme des carrés des chiffres de T1 [i].

Exemple : Si T1 [i] = 254 alors T2 [i] = $2^2 + 5^2 + 4^2 = 45$

Ecrire un algorithme qui permet de saisir les éléments de T1, de remplir puis d'afficher le tableau T2.

Exercice 6

On se propose d'écrire un algorithme permettant de remplir deux tableaux T1 et T2 de N entiers à deux chiffres chacun ($2 < N < 15$) puis de former un tableau T tel que un élément T [i] est le résultat de la fusion des deux éléments T1 [i] et T2 [i] selon le principe suivant:

- insérer le chiffre des dizaines du plus petit nombre parmi T1 [i] et T2 [i], entre les deux chiffres du plus grand nombre parmi T1 [i] et T2 [i].
- mettre le chiffre des unités du plus petit nombre parmi T1 [i] et T2 [i], à droite du nombre obtenu.

Exemples: - Pour T1 [i] = 52 et T2 [i] = 36, T [i] sera égal à 5326

- Pour T1 [i] = 13 et T2 [i] = 47, T [i] sera égal à 4173

Exercice 7

On se propose d'écrire un algorithme qui permet de chercher, dans une matrice carrée de N * N éléments, un entier X donnée. La matrice est remplie par des entiers saisis au clavier et appartenant à l'intervalle (0,100). N est entier donné dans l'intervalle (5,20).

Une valeur est considérée trouvée dans la matrice si elle est un élément de la 1ère ou de la dernière ligne ou de la 1ère ou de la dernière colonne ou de la première ou de la deuxième diagonale.

Exemple :

Soit la matrice suivante M=

Pour : N =7 et X = 25

La fonction retourne **faux.**

Pour N = 7 et X = 50

La fonction retourne **vrai.**

10	20	15	26	48	26	35
12	18	25	14	10	26	17
2	40	50	47	18	32	17
18	12	5	18	9	36	59
98	10	14	17	12	3	18
2	0	25	15	41	10	100
26	35	14	74	15	12	8

TD n° 10 : Récursivité

Exercice 1

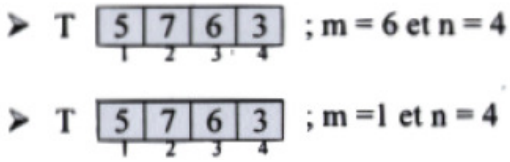
Soient T un vecteur de n entiers, m un autre entier et f une fonction définie de la façon suivante :

f (m, n, T) = VRAI si (T [n] = m)

= f (m, n-1, T) sinon pour tout n ≠ 0.

f (m, 0, T) = FAUX

1. Donner la trace d'exécution de la fonction f pour les cas suivants :



2. En déduire le rôle de la fonction f.
3. Ecrire un algorithme récursif de la fonction f.
4. Donner sous forme d'un algorithme la version itérative de la fonction f.

Exercice 2

Ecrire une fonction récursive qui retourne la somme des chiffres d'un nombre N donné.

Exemple : Si N = 417 alors 4 + 1 + 7 = 12

Exercice 3

Ecrire une fonction récursive qui détermine le $K^{ième}$ chiffre à partir de la droite d'un entier n>0.

Exemples : le $3^{ème}$ chiffre de 89**7**52 est 7

Le $5^{ème}$ chiffre de **2**1327 est 2

Exercice 4

Ecrire une fonction récursive qui retourne le nombre d'occurrences d'un caractère C dans la chaine CH.

Exercice 5

Ecrire une fonction récursive qui retourne VRAI si une chaîne CH est tautogramme.

Un « **tautogramme** » est une chaîne dont chacun de ses mots commence par la même lettre (sans distinction entre majuscule et minuscule).

Exemple : ch = « **L**e **l**ion **l**ape **l**e **l**ait **l**entement. » est un tautogramme

Exercice 6

Ecrire une fonction récursive qui permet de vérifier l'existence d'un entier E dans un tableau T de N entiers.

Version 1 : Recherche **séquentielle**

Version 2 : Recherche **dichotomique**

Exercice 7

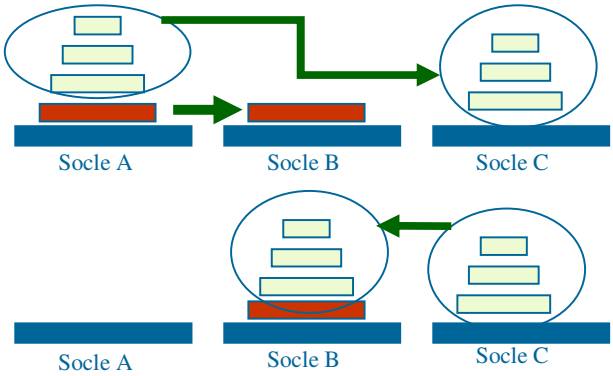
Le problème des tours de Hanoi est un grand classique de la récursivité car la solution itérative est relativement complexe. On dispose de 3 tours appelées A, B et C. La tour A contient n disques empilés par ordre de taille décroissante qu'on veut déplacer sur la tour B dans le même ordre en respectant les contraintes suivantes :

- On ne peut déplacer qu'un disque à la fois
- On ne peut empiler un disque que sur un disque plus grand ou sur une tour vide.

Ainsi, le paramétrage de la procédure déplacer sera le suivant :

Procédure **déplacer** (n : Entier ; A, B, C : Caractère)

Exemple : n = 4



Proposez une version récursive de la procédure « **déplacer** » qui permet de déplacer n disques (1 <= n <= 64) d'un socle A sur un socle B, en utilisant un socle intermédiaire C.

Bibliographie

[1] Thomas H. Cormen, Charles E. Leireson, Ronald L Rivest et Clifford Stein, « Introduction à l'algorithmique », cours et exercices 2ème cycle Ecoles d'ingénieurs », Edition Dunod, 2ème édition, Paris 2002

[2] Michael GRIFFITHS, « ALGORITHMIQUE ET PROGRAMMATION », Hernes, 1992

[3] S ROHAUT, « Algorithmique et Techniques fondamentale de programmation », Edition Eni 2007.

[4] http://algo.developpez.com/cours/

Tableau des codes ASCII décimaux

0	NUL	**44**	,	**88**	X	**132**	ä	**176**	▒	**220**	▄
1	SOH	**45**	-	**89**	Y	**133**	à	**177**	▒	**221**	¦
2	STX	**46**	.	**90**	Z	**134**	å	**178**	▓	**222**	Ì
3	ETX	**47**	/	**91**	[	**135**	ç	**179**	│	**223**	▀
4	EOT	**48**	0	**92**	\	**136**	ê	**180**	┤	**224**	Ó
5	ENQ	**49**	1	**93**	]	**137**	ë	**181**	Á	**225**	ß
6	ACK	**50**	2	**94**	^	**138**	è	**182**	Â	**226**	Ô
7	BEL	**51**	3	**95**	_	**139**	ï	**183**	À	**227**	Ò
8	BS	**52**	4	**96**	`	**140**	î	**184**	©	**228**	õ
9	TAB	**53**	5	**97**	a	**141**	ì	**185**	╣	**229**	Õ
10	LF	**54**	6	**98**	b	**142**	Ä	**186**	║	**230**	µ
11	VT	**55**	7	**99**	c	**143**	Å	**187**	╗	**231**	þ
12	FF	**56**	8	**100**	d	**144**	É	**188**	╝	**232**	Þ
13	CR	**57**	9	**101**	e	**145**	æ	**189**	¢	**233**	Ú
14	SO	**58**	:	**102**	f	**146**	Æ	**190**	¥	**234**	Û
15	SI	**59**	;	**103**	g	**147**	ô	**191**	┐	**235**	Ù
16	DLE	**60**	<	**104**	h	**148**	ö	**192**	└	**236**	ý
17	DC1	**61**	=	**105**	i	**149**	ò	**193**	┴	**237**	Ý
18	DC2	**62**	>	**106**	j	**150**	û	**194**	┬	**238**	¯
19	DC3	**63**	?	**107**	k	**151**	ù	**195**	├	**239**	´
20	DC4	**64**	@	**108**	l	**152**	ÿ	**196**	─	**240**	-
21	NAK	**65**	A	**109**	m	**153**	Ö	**197**	┼	**241**	±
22	SYN	**66**	B	**110**	n	**154**	Ü	**198**	ã	**242**	‗
23	ETB	**67**	C	**111**	o	**155**	ø	**199**	Ã	**243**	¾
24	CAN	**68**	D	**112**	p	**156**	£	**200**	╚	**244**	¶
25	EM	**69**	E	**113**	q	**157**	Ø	**201**	╔	**245**	§
26	SUB	**70**	F	**114**	r	**158**	×	**202**	╩	**246**	÷
27	ESC	**71**	G	**115**	s	**159**	ƒ	**203**	╦	**247**	¸
28	FS	**72**	H	**116**	t	**160**	á	**204**	╠	**248**	°
29	GS	**73**	I	**117**	u	**161**	í	**205**	═	**249**	¨
30	RS	**74**	J	**118**		**162**	ó	**206**	╬	**250**	·
31	US	**75**	K	**119**	w	**163**	ú	**207**	¤	**251**	¹
32	Space	**76**	L	**120**	x	**164**	ñ	**208**	ð	**252**	³
33	!	**77**	M	**121**	y	**165**	Ñ	**209**	Ð	**253**	²
34	„	**78**	N	**122**	z	**166**	ª	**210**	Ê	**254**	■
35	#	**79**	O	**123**	{	**167**	º	**211**	Ë	**255**	
36	$	**80**	P	**124**	|	**168**	¿	**212**	È		
37	%	**81**	Q	**125**	}	**169**	®	**213**	ı		
38	&	**82**	R	**126**	~	**170**	¬	**214**	Í		
39	'	**83**	S	**127**	DEL	**171**	½	**215**	☼		
40	(	**84**	T	**128**	Ç	**172**	¼	**216**	Ï		
41	)	**85**	U	**129**	ü	**173**	¡	**217**	┘		
42	*	**86**	V	**130**	é	**174**	«	**218**	┌		
43	+	**87**	W	**131**	â	**175**	»	**219**	█		