**République Algérienne Démocratique et Populaire**

**Ministère de l'Enseignement Supérieur et de la Recherche Scientifique**

**Université Abderrahmane Mira de Bejaia**

**Faculté des Sciences Economiques, de Gestion et Commerciales**

**Département des Sciences Commerciales**

*Habilitation à Diriger des Recherches (HDR)*

**Polycopié de Cours du Module Informatique II (INF 2)**

# Informatique II (INF 2)

**Présenté par**

Dr. Zair BOUZIDI

**Objectifs de l'enseignement :**

**Familiarisation** avec de **Nouveaux Concepts d'Algorithmique**, de **Programmation** et de l'**Outil informatique**.

**Année** 2022 – 2023

# Table de Matières

# Listes de Tables

# Listes de Figures

# Introduction Générale

Le langage Pascal est un langage de programmation structuré développé dans les années 1970 pour l'enseignement de la programmation. Il est principalement utilisé pour enseigner les concepts de base de la programmation, tels que les structures de données et la programmation orientée objet.

Il n'est pas aussi populaire que Python, C ou Java en raison de sa popularité limitée en tant que langage de production pour les entreprises. Cependant, il est encore utilisé dans certains domaines spécialisés, tels que la programmation scientifique et la recherche.

Les avantages de Pascal incluent une syntaxe claire et structurée qui facilite la compréhension et la maintenance du code, ainsi que la capacité à gérer de manière efficace les données complexes. Les inconvénients incluent des performances inférieures par rapport à d'autres langages plus modernes et une quantité limitée de bibliothèques et de ressources disponibles.

Il existe de nombreux projets réalisés avec Pascal, y compris des applications commerciales, des jeux et des outils de développement. Il est encore utilisé pour enseigner les concepts de base de la programmation et peut-être une excellente option pour les débutants dans le développement de logiciels.

Ce polycopié pédagogique s'adresse aux étudiants de deuxième année du Département des Sciences Commerciales de la Faculté des Sciences économiques, de Gestion et Commerciales de l'Université Abderrahmane Mira de Béjaia. Il vise à rendre stimulant l'apprentissage des outils de base de l'informatique, à savoir [1, 2, 3] :

+ les concepts de base du langage algorithmique et de l'organigramme, tels que :
    * la notion de variable,
    * de type de variable,
    * affectation d'une valeur à une variable,
    * expression,
    * algorithme,
    * instance,
    * heuristique,
    * efficacité d'un algorithme,

* notion de complexité,
* langage algorithmique et Organigramme,
* notion de codage et de sctructures de contrôle,
* et notion de structure de données (Tableaux, Matrices, ...),

+ Initiation à un Langage de Programmation Pédagogique à l'instar de Pascal
+ Quelques exemples d'algorithmes et de leur traduction en Pascal.

Le présent document couvre 14 séances de Travaux Pratiques, à raison d'une heure et demi par séance (01h30/ séance). L'ensemble des cours proposés se réfèrent au programme national des classes de Seconde Année des Sciences Commerciales.

Ce polycopié pédagogique intègre trois (3) compétences discursives suivantes [1, 3] :

- compréhension logique,
- programmation écrite et
- programmation sur machine.

Il est alors question de développer la compétence logique à travers l'écrit et la programmation écrite chez l'étudiant.

L'apprentissage de l'informatique répond à des exigences bien spécifiques, aussi bien à l'écrit, notamment : questions grammaticales, syntaxiques qu'à la Programmation (sur machine).

Le protocole suivi pour apprendre la programmation aux étudiants prévoit dans un premier temps que les travaux demandés soient relatifs au domaine des sciences économiques, et dans un second temps, que la première version soit corrigée par l'enseignant à travers une approche qui consiste à exploiter l'erreur en tant qu'outil d'apprentissage.

Cette pédagogie de l'erreur consiste d'abord à identifier les erreurs de programmation (grammaticales, syntaxiques et lexicales) commises par l'étudiant. Il lui est demandé de les analyser, les corriger et les améliorer avec l'accompagnement de l'enseignant qui supervise.

L'expérimentation de cette méthode auprès des étudiants de deuxième année vise un apprentissage responsable via lequel l'étudiant apprend des astuces de programmation et acquiert un certain degré d'autonomie et un réflexe d'autocorrection et surtout de Programmation.

Il importe de préciser que l'ensemble des cours qui sollicitent et mobilisent les compétences informatiques de programmation sont dispensés sur une séance.

En plus, des travaux sont répartis en goupes de 2 ou 3 étudiants à préparer chaque 15 jours ou carrément mensuels que ce soit à exposer ou à présenter à l'ensemble des étudiants (qui sera évalué).

Certaines activités requièrent une approche coopérative : cette démarche amène l'étudiant à recenser ses lacunes et consolider ses acquis.

En d'autres termes, cette méthode l'aidera à discerner les points forts des points

faibles et lui faire acquérir des compétences communicationnelles, telles que gérer son stress, parler en public et occuper de l'espace. à ce titre, une liste non exhaustive de sujets choisis par les apprenants pour l'épreuve orale pourrait révéler les orientations thématiques de ces derniers.

Plusieurs approches sont convoquées dans l'enseignement/apprentissage de l'économique à l'école, notamment les deux approches communicatives et motivationnelles qui sont essentiellement centrées sur l'apprenant. Elles ont pour principal but de créer un contexte qui favoriserait l'interaction et cultiverait la motivation chez des sujets ambitieux, et d'autres pas. En somme, l'approche motivationnelle se traduit par un système de récompense au lieu de sanction, de participation au lieu d'abstention et d'implication au lieu de marginalisation.

L'évaluation semestrielle des acquis est réalisée sous forme de contrôles continus d'examens de moyenne durée. à la fin de ce document pédagogique, s'ajoute une grille d'évaluation des enseignements qui a été élaborée par mes soins me permettant ainsi de faire un tour d'horizon, annuellement, sur ma pédagogie dans le but de l'améliorer, garantir un enseignement de qualité et un meilleur rendement.

**Objectifs de l'enseignement**

Cette matière permet d'initier les étudiants à [1, 2, 4, 5] :

- la familiarisation avec les systèmes de numérations de l'ordinateur,
- la conception d'algorithmes,
- la conception d'organigrammes, et
- la logique de programmation.

**Connaissances préalables recommandées**

Les connaissances préalables recommandées consistent à la maîtrise :

- du français,
- des méthodes de calcul usuelles, et
- des concepts de boules

**Contenu de la matière**

Le contenu de ce polycopié consiste à :

1- lRésolution de problèmes à l'aide de l'ordinateur

2- Structure de la programmation

3- Les algorithmes et

4- Les boucles

# Chapitre 1

# Généralités et Notions de Base

## Summary



## 1.1 Introduction

Plusieurs approches sont convoquées dans l'enseignement/apprentissage de l'économique à l'école, notamment les deux approches communicatives et motivationnelles qui sont essentiellement centrées sur l'apprenant. Elles ont pour principal but de créer

un contexte qui favoriserait l'interaction et cultiverait la motivation chez des sujets ambitieux, et d'autres pas. En somme, l'approche motivationnelle se traduit par un système de récompense au lieu de sanction, de participation au lieu d'abstention et d'implication au lieu de marginalisation.

L'évaluation semestrielle des acquis est réalisée sous forme de contrôles continus d'examens de moyenne durée. à la fin de ce document pédagogique, s'ajoute une grille d'évaluation des enseignements qui a été élaborée par mes soins me permettant ainsi de faire un tour d'horizon, annuellement, sur ma pédagogie dans le but de l'améliorer, garantir un enseignement de qualité et un meilleur rendement.

**Objectifs de l'enseignement**

Cette matière permet d'initier les étudiants à [1, 2, 4, 5] :

- Logique de programmation
- conception d'organigramme
- familiarisation avec les systèmes de numérations de l'ordinateur

Connaissances préalables recommandées :

La maîtrise du français , des méthodes de calcul usuelles , des concepts de boules

**Contenu de la matière**

1- Résolution de problèmes à l'aide de l'ordinateur

2- Structure de la programmation

3- Les algorithmes

4- Les boucles

## 1.2 Algorithmique

L'algorithmique consiste à construire des algorithmes qui seront appelés 'a être traduits dans un langage de programmation tel que Pascal [2, 3, 6].

### 1.2.1 Algorithme

Un algorithme est une procédure de calcul bien définie qui prend en entrée un ensemble de valeurs et qui délivre en sortie un ensemble de valeurs [2, 5, 6, 7, 8].

Pour informatiser une application (faire réaliser par ordinateur, une tâche qui était réalisée par l'Homme), il faut [2, 3, 6] :

- détailler suffisamment les étapes de résolution du problème, pour qu'elle soit exécutable par l'homme

- et transférer la résolution en une suite d'étapes si élémentaire et simple à exécuter (pouvant être codée en un programme dans un langage compréhensible par ordinateur).

Toute suite d'étapes si élémentaire et simple soit-elleà exécuter s'appelle un **ALGORITHME**.

Un Algorithme est le résultat d'une **démarche logique** (résolution du problème) en vue d'une mise en œuvre pratique sur ordinateur pour obtenir des **résultats concrets** (en passant par par un langage de programmation).

Il consiste en une succession d'opérations, fidèlement exécutées, pour produire le résultat désiré.

C'est une suite d'actions élémentaires (instructions) à effectuer par un automate pour arriver à un résultat déterminé.

L'algorithmique est la logique d'écrire des algorithmes qui permet de [2, 5, 6, 7, 8] :

- connaître la résolution manuelle du problème ;
- utiliser les capacités de l'ordinateur (actions élémentaires à assurer), et
- la logique d'exécution des instructions.

### 1.2.2 Etapes de Résolution de Problèmes

Les étapes de Résolution de Problèmes sont les suivantes [2, 7, 8, 9] :

- 1 Comprendre l'énoncé du problème
- 2 Décomposer le problème en sous-problèmes plus simples à résoudre
- 3 Associer, à chaque sous-problème, une spécification :
  - -des données nécessaires ;
  - des données résultantes ;
  - La démarche à suivre pour arriver au résultat en partant d'un ensemble de données

4. Et élaborer un algorithme.

**Exemple :**

Cet algorithme permet d'afficher sur lécran la phrase suivante :

La valeur de 3*5 est 15

**Exemple :**

Trier une suite de nombres entiers dans l'ordre croissant.

**Entrée :**

Suite de n nombres entiers :

$$(a_1, a_2, ..., a_n)$$

**Sortie :**

Une permutation de la suite donnée en entrée

$$(\check{a}_1, \check{a}_2, ..., \check{a}_n)$$

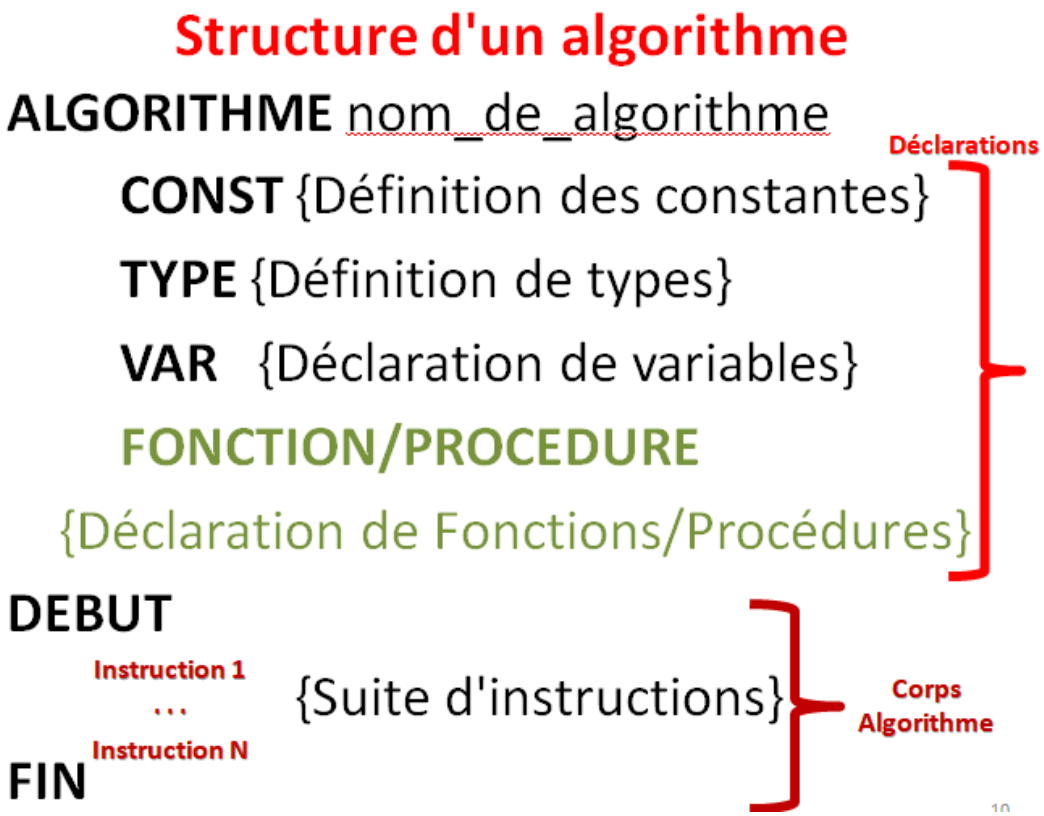



FIGURE 1.1 – Structure d'un algorithme.

telle que

$$\check{a}_1 \leq \check{a}_2 \leq, ..., \leq \check{a}_n.$$

A partir de la suite

$$(6, 9, 2, 4),$$

un algorithme de tri fournira le résultat

$$(2, 4, 6, 9).$$

### 1.2.3 Instance

Une valeur particulière de l'ensemble des valeurs données en entrée est appelée **instance du problème**.

**Exemple**

La valeur

$$(6, 9, 2, 4)$$

est une instance du problème.

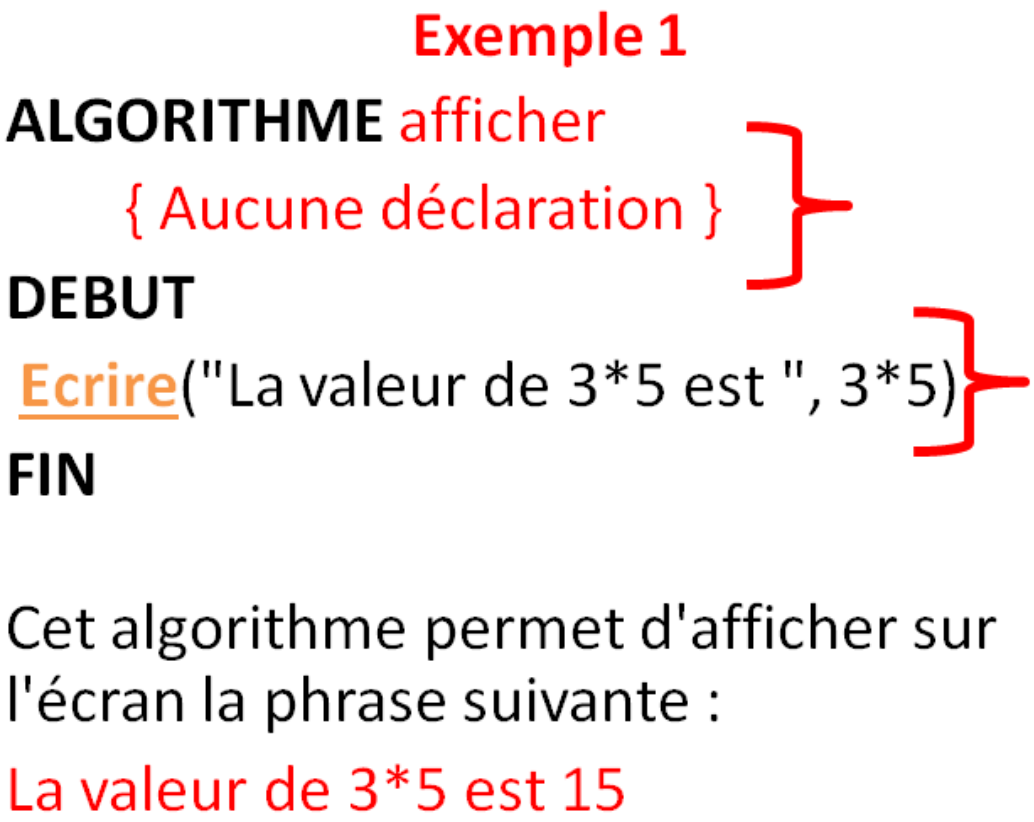



FIGURE 1.2 – Exemple d'algorithme.

### 1.2.4 Exemple de Résolution de Problème

On veut écrire l'algorithme qui permet de lire 2 valeurs et de calculer la moyenne :
Pour cela, il faut d'abord Rechercher la solution en 2 phases, à savoir [2, 3, 6] :
- Comprendre comment résoudre manuellement,
- Définir données, formules de calcul et résultats
- Problème : (A + B) / 2

### 1.2.5 Algorithme Correct

Un algorithme est correct si pour toute instance du problème il se termine et produit une sortie correcte [2, 5].

Les algorithmes peuvent être spécifiés en langage humain ou tout langage informatique.

**Exemple :**

On veut écrire l'algorithme qui permet de saisir 3 notes d'un étudiant dans trois matières, avec les coefficients respectifs 2, 3 et 1 et de calculer la moyenne.

Dans ce qui suit nous utiliserons un langage proche du langage naturel.

Nous donnerons une implémentation en Pascal.

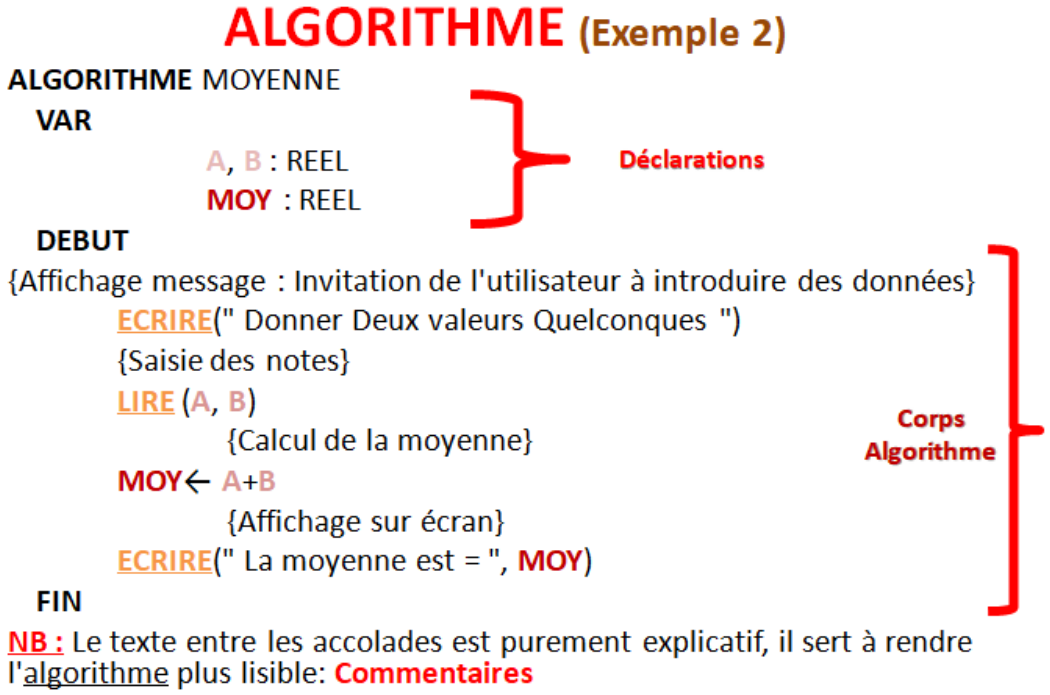



FIGURE 1.3 – Exemple d'algorithme.

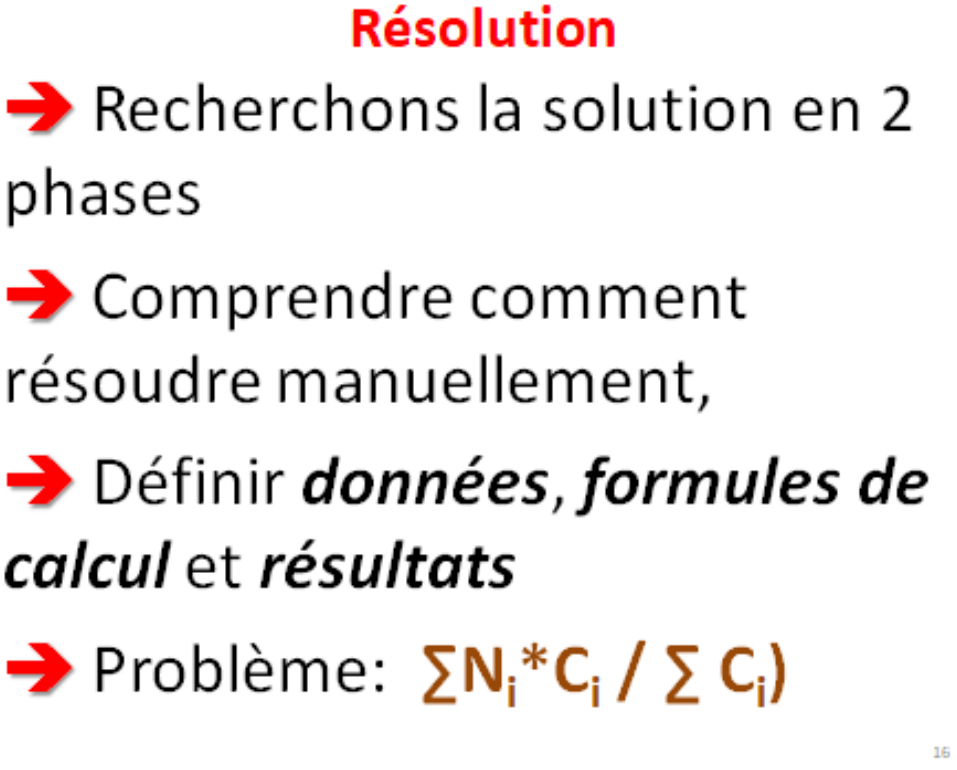



FIGURE 1.4 – Resolution de problème du calcul de la moyenne.

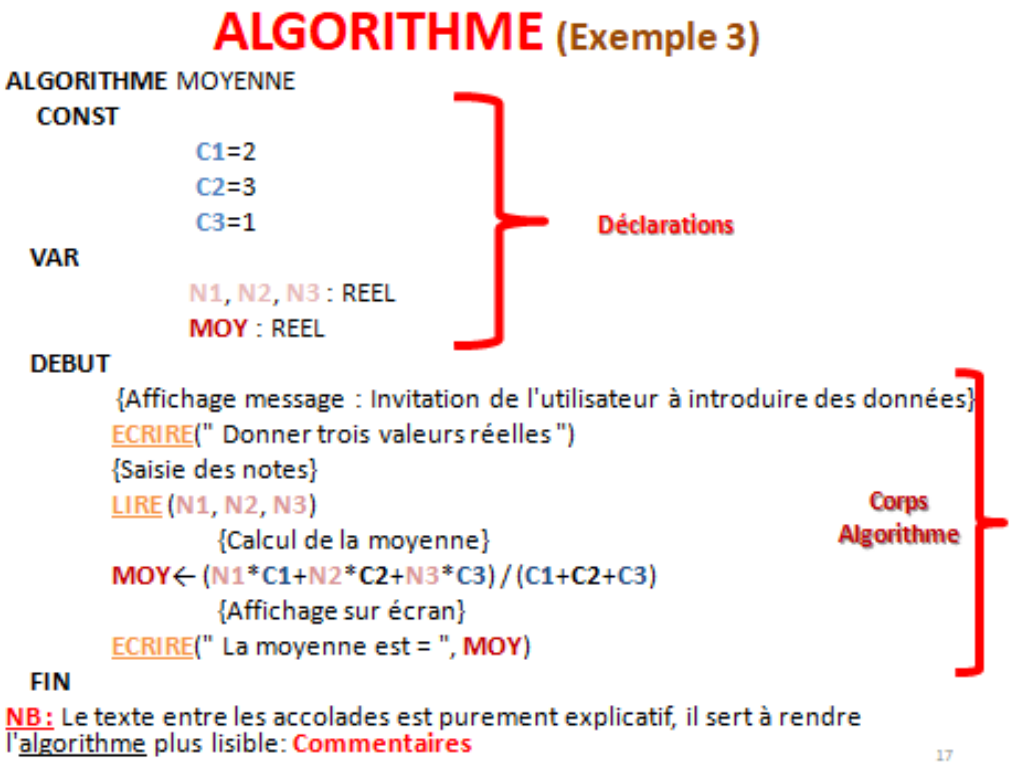



FIGURE 1.5 – Resolution de l'Exemple en obtenant un Algorithme.

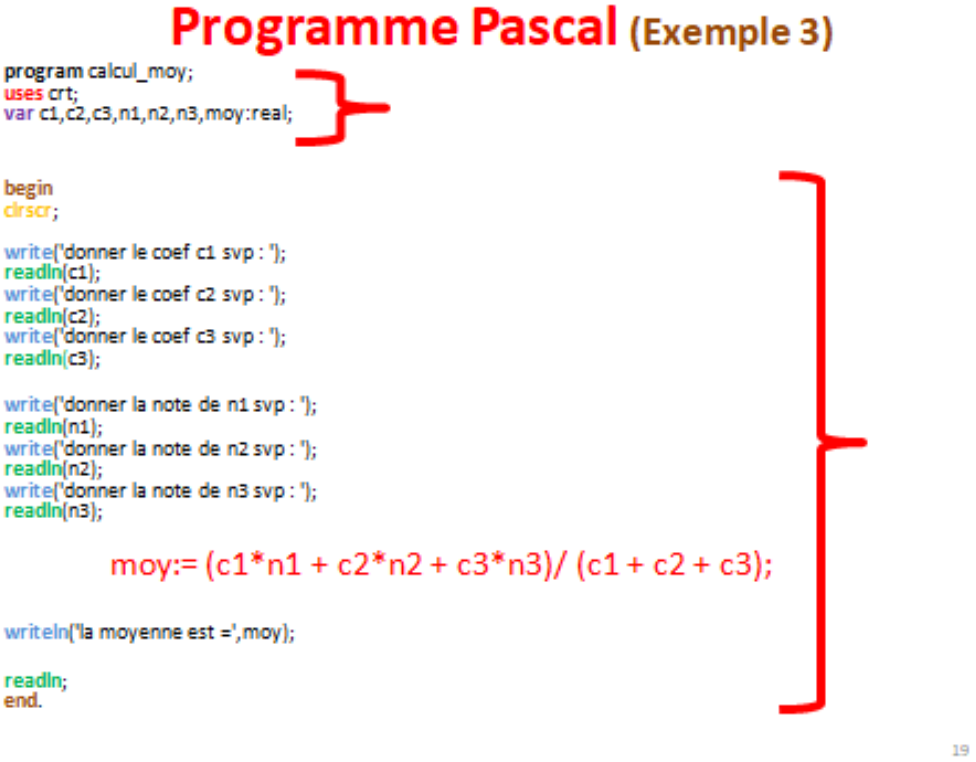



FIGURE 1.6 – Resolution de l'Exemple en obtenant son programme Pascal.

### 1.2.6 Structure de Données

Une structure de données est un moyen de stocker et d'organiser des données pour faciliter leur stockage, leur utilisation et leur modification [2, 3, 6, 10].

### 1.2.7 Applications

De nombreux problèmes nécessitent des algorithmes [5, 6, 8] :

- Bioinformatique
- Moteur de recherche sur Internet
- Commerce électronique
- Affectation de tâches
- ...

L'efficacité d'un algorithme est mesurée par son coût (complexité) en temps et en mémoire [2, 5, 6, 7, 8].

## 1.3 Notion de complexité

L'efficacité d'un algorithme est fondamentale pour résoudre effectivement des problèmes [2, 3, 6, 10].

**Exemple**

Supposons que l'on dispose de deux ordinateurs. L'ordinateur A est capable d'effectuer 109 instructions par seconde. L'ordinateur B est capable d'effectuer 107 instructions par seconde.

Considérons un même problème (de tri par exemple) dont la taille des données d'entrées est n. Pour l'ordinateur A, on utilise un algorithme qui réalise $2n^2$ instructions. Pour l'ordinateur B, on utilise un algorithme qui réalise 50nlog(n) instructions.

Pour traiter une entrée de taille 106 : l'ordinateur A prendra 2000s et l'ordinateur B prendra 100s.

Ainsi, même si la machine B est médiocre, elle résoudra le problème 20 fois plus vite que l'ordinateur A.

### 1.3.1 Complexité d'un Algorithme

La complexité d'un algorithme est en temps, le nombre d'opérations élémentaires effectuées pour traiter une donnée de taille n, en mémoire, l'espace mémoire nécessaire pour traiter une donnée de taille n [2, 3, 6, 10].

Dans ce cours, nous considèrerons que la complexité des instructions élémentaires les plus courantes sur un ordinateur ont un temps d'exécution que l'on considèrera dans ce cours comme constant égal à 1.

Les instructions élémentaires sont : addition, multiplication, modulo et partie entière, affectation, instruction de contrôle. Ce qui intéresse fondamentalement l'algorithmique c'est l'ordre de grandeur (au voisinage de l'infini) de la fonction qui exprime le nombre d'instructions.

Un programme est un algorithme codé (traduit) dans un langage compréhensible par l'ordinateur à l'aide d'un compilateur [11, 12] (traducteur) (Pascal, ...)

### 1.3.2 Langage Algorithmique

Il est nécessaire de disposer d'un langage qui soit non lié à l'implémentation. Ceci permet une description plus précise des structures de données ainsi qu'une rédaction de l'algorithme plus souple et plus **lisible** [2, 5, 6, 7, 8]. Le langage algorithmique est un exemple de ce qui peut être utilisé et qui sera utilisé dans ce cours. Il est composé de chaînes de caractères alphanumériques, de signes opératoires (+,-,*,/,<,<=,>=,>,<>,==,=,ou,non,et), de mot-clés réservés, et de signes de ponctuation : "=, ;,(,), début, fin, //.

**Remarque**

Pascal aussi utilise des marqueurs de début (Begin) et de fin (End).

## 1.4 Identifiant

est un Nom, dommé par l'utilisateur, de [2, 3, 5, 6, 7, 8, 11, 12] :

— Constante,

— Variable,

— Algorithme (Programme),

— Fonction,

— ou Procédure,

qui est une suite de caractères alphanumériques dont le premier caractère est, **obligatoirement**, alphabétique, peut accepter (_) (souligné) au milieu.

**Exemple :**

A_1, Nom_User, Nom sont des identifiants.

Mais 3A, _Nom ne sont pas acceptés comme identifiants.

**Remarque :**

Evitez les caractères spéciaux suivants : !, ?, *, =, ....

### 1.4.1 Type

Un **type** est un triplet composé :

- d'un nom,
- d'un ensemble de valeurs,
- d'un ensemble d'opérations définies sur ces valeurs.

### 1.4.2 Types de base

Les types de base sont [2, 3, 10] :

- entier,
- caractères,
- booléen,
- réel

**Booléens** Une variable de type booléen prend comme valeur VRAI ou FAUX.

Les opérations usuelles sont **ET**, **OU** et **NON** qui sont données dans les tables qui suivent.

**Entiers** Une variable de type entier peut prendre comme valeur l'ensemble des nombres entiers signés. Les opérations associées sont les opérations usuelles +,-,*,/.

**Réels** Une variable de type réel peut prendre comme valeur l'ensemble des nombres réels. Les opérations associées sont les opérations usuelles +,-,*,/.

**Attention**

Les valeurs

o "1" qui est un caractère,

o 1 qui est un entier,

o 1. qui est un réel

sont différentes et ne seront pas codés de la même manière dans la mémoire de la machine.

### 1.4.3 Variable

Une variable est un triplet composé d'un type (déjâ défini), d'un identifiant et d'une valeur.

$$\textbf{var}\mathit{NomDeVariable} : \textbf{Type};$$

Type est à prendre pour l'instant dans l'ensemble : {entier, car, booléen, réel}

### 1.4.4 Expression

Les Expressions sont constituées à l'aide de variables déjà déclarées, de valeurs, de parenthèses et d'opérateurs du (des) type(s) des variables concernées[11, 12].

### 1.4.5 Affectation

L'affectation est l'instruction qui permet de stocker une valeur dans une variable. On écrit

**NomDeVariable** := **ExressionDuTypeDeLaVariable**;

**Remarque**

Toute variable doit être déclarée et recevoir une valeur initiale.

**Caractères** Une variable de type car peut prendre comme valeur l'ensemble des caractères imprimables. On notera les valeurs entre guillemets. On considère souvent que les caractères sont ordonnés dans l'ordre alphabétique.

**Affectation :**

Une variable est modifiée dans unalgorithme / programme à l'aide de l'affectation ($\leftarrow$) :

**Exemple :**

$$Moy \leftarrow 0;$$

ou d'une incrémentation :

$$I \leftarrow I + 1;$$

ou d'une décrémentation :

$$I \leftarrow I - 1;$$

TABLE 1.1 – Table Logique **ET**

| **ET** | **Faux** | **Vrai** |
|---|---|---|
| **Faux** | Faux | Faux |
| **Vrai** | Faux | Vrai |

TABLE 1.2 – Table Logique **OU**

| **OU** | **Faux** | **Vrai** |
|---|---|---|
| **Faux** | Faux | Vrai |
| **Vrai** | Vrai | Vrai |

TABLE 1.3 – Table Logique **NON**

| **NON** | |
|---|---|
| **Faux** | **Vrai** |
| **Vrai** | **Faux** |

### 1.4.6 Comparaison

Les opérateurs $<, \leq, =, \neq, >, \geq$ permettent de comparer les valeurs de type entier, réel et caractère. Le résultat de cette comparaison est une valeur booléenne.

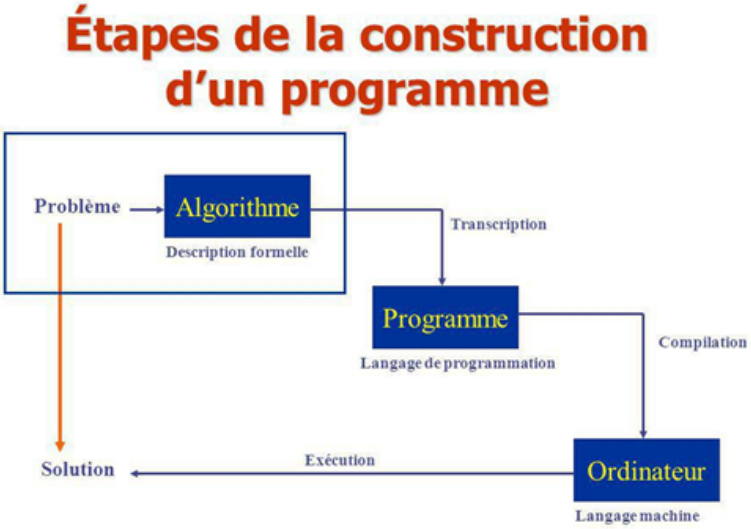



FIGURE 1.7 – L'algorithme, une des étapes dans l'élaboration d'un Programme.

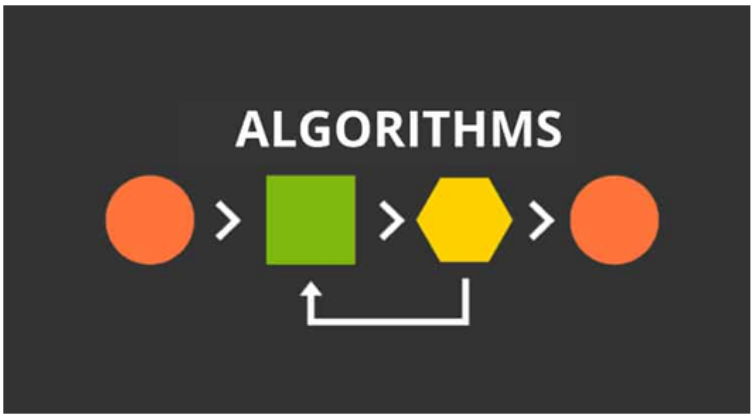
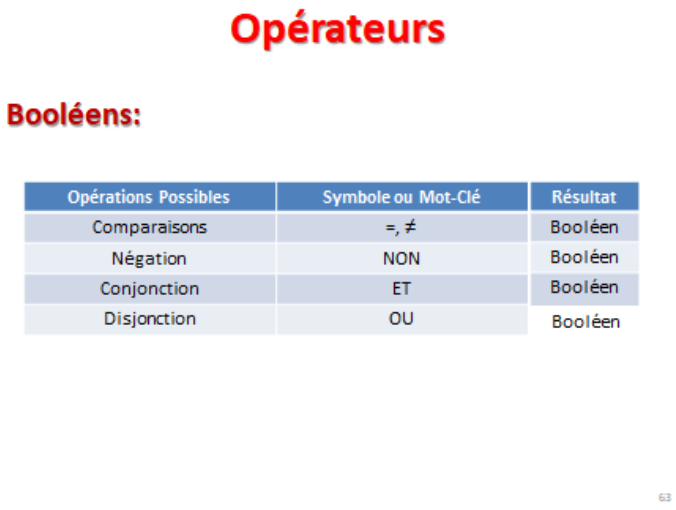



FIGURE 1.8 – Exemple d'algorithme.



## Opérateurs

### Booléens:

| Opérations Possibles | Symbole ou Mot-Clé | Résultat |
|---|---|---|
| Comparaisons | =, ≠ | Booléen |
| Négation | NON | Booléen |
| Conjonction | ET | Booléen |
| Disjonction | OU | Booléen |

63

FIGURE 1.9 – Opérateurs Booléens.

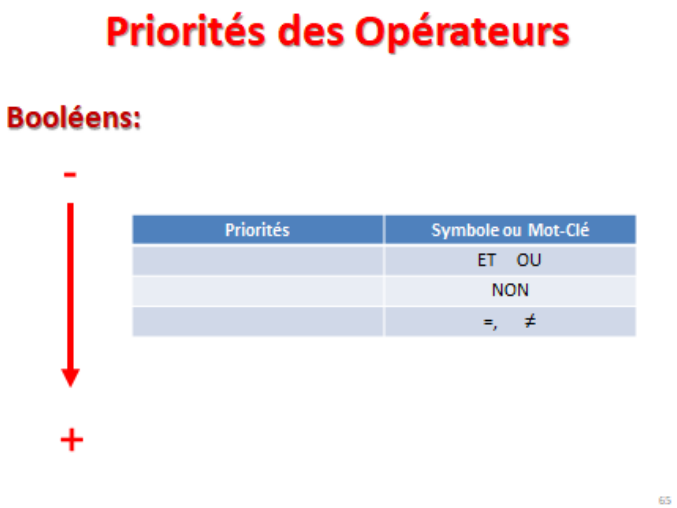



FIGURE 1.10 – Opérateurs Booléens.

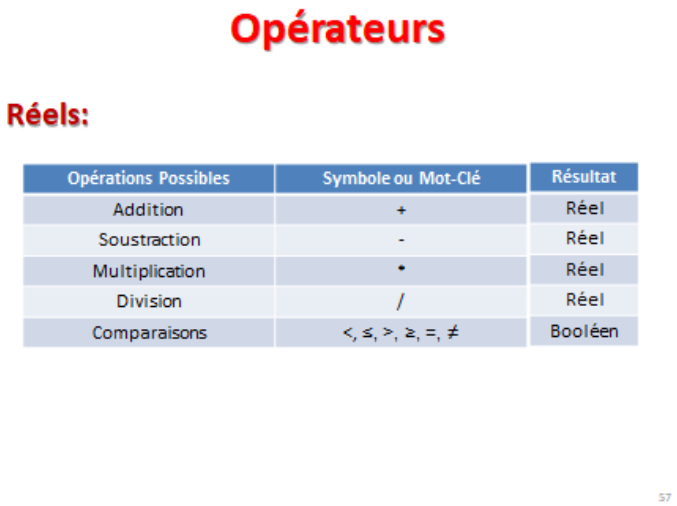



FIGURE 1.11 – Opérateurs Rléels.

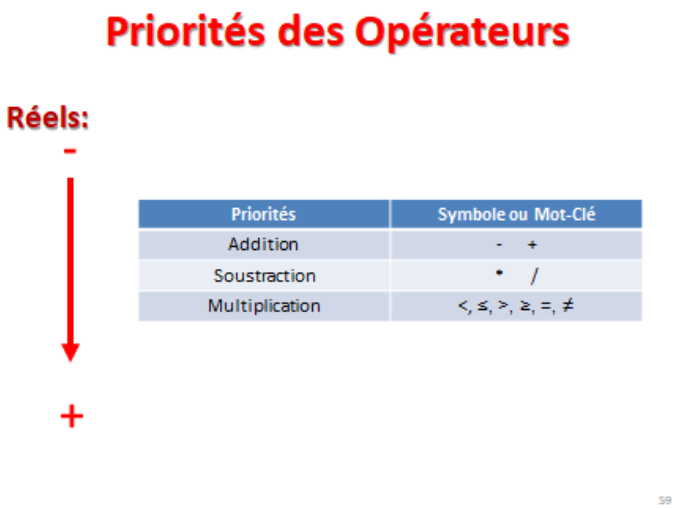



FIGURE 1.12 – Opérateurs Booléens.

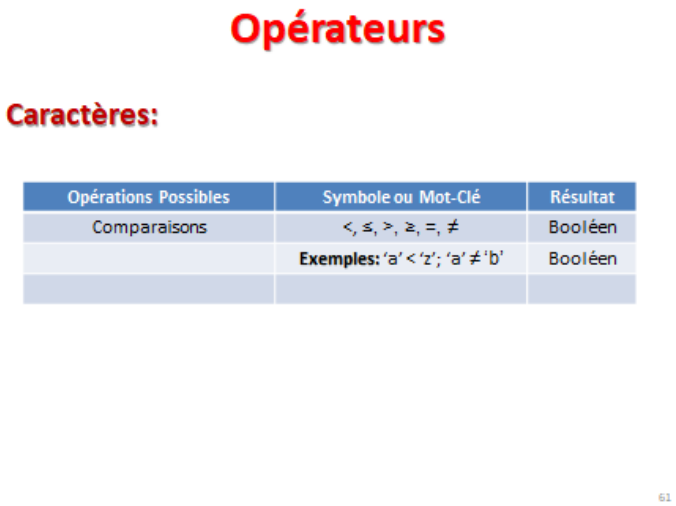
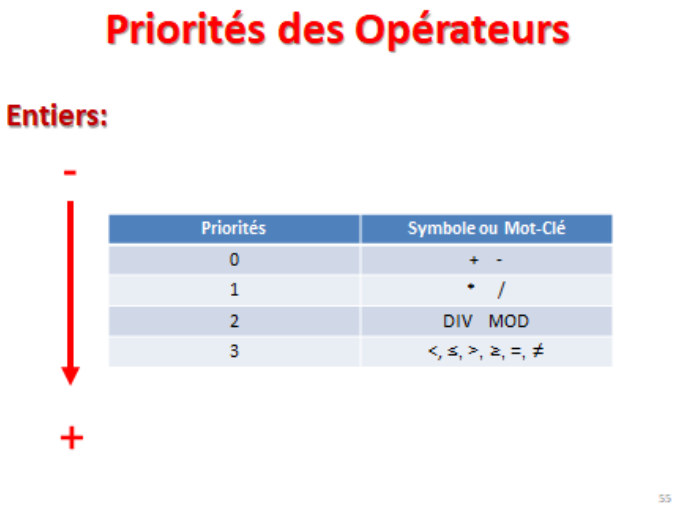



FIGURE 1.13 – Opérateurs Caracteres.




| Priorités | Symbole ou Mot-Clé |
|---|---|
| 0 | + - |
| 1 | * / |
| 2 | DIV MOD |
| 3 | <, ≤, >, ≥, =, ≠ |

FIGURE 1.14 – Opérateurs Booléens.


FIGURE 1.15 – Abstraction de l'Univers Réel.

# Chapitre 2

# Structures de Contrôles

## Summary



## 2.1 Structures de Contrôle

Il y a trois structures principales de contrôle qui permettent de construire des algorithmes [2, 3, 6, 10] :

## 2.2 Bloc d'instruction

**Bloc instructions**
**Begin**
Instruction 1
Instruction 2
.....
Instruction N
**End**

**Exemple :**

```
Program MonProg;
Var
    i : integer;
    j : integer;
Begin
    i := 1;
    j := 'Bonjour a tous ';
End
```

**Exemple :**

```
Program MonPremierProg;
Var
    s : string;
Begin
    s := 'Salut a tous ';
    writeln(s);
    readln;
End
```

**Exemple :**

```
Program MonDeuxiemeProg;
Var
    s , Phrase : string;
Begin
    writeln(' Entrez un mot : ');
    readln( s );
    Phrase := s;
    writeln(' Entrez un deuxieme mot : ');
    readln( s );
    Phrase := Phrase + s;
    writeln(' Entrez un troisieme mot : ');
    readln( s );
    Phrase := Phrase + s;
    writeln(' La Phrase Complete est : ');
    writeln( Phrase );
End
```

**Exemple :**

Ecrire Un Algorithme Qui Permet d'introduire Un Nombre Et d'afficher Son Double Ainsi Que Sa Moitié.

```
Program Exo_serie;
uses wincrt;
var
    A : integer;
Begin
    write('Donner un Nombre Entier : ');
    readln(A);
    writeln('Le Double de ', A, ' est ', 2*A, ' Sa MoitiÃ© est ', A/2)
End.
```

**Exemple :**

Ecrire un algorithme (ou un programme) qui lit le prix HT (Hors Taxe) d'un article, le nombre d'articles et le taux de la TVA (Taxe sur la Valeur AjoutÃ©e) puis affiche le prix total TTC (Toutes Taxes Comprises) de tous les articles.

```
Algorithme Affiche_PrixTTC;
Var PrixHT, Nb_Articles, PrixTTC, TauxTVA : rÃ©el;
Début
    Ecrire ( "Donner le Prix Hors Taxe : ");
    Lire (PrixHT);
    crire ( " Donner le Nombre d'articles : ");
    Lire (Nb_Articles);
    Ecrire ( "Donner le Taux TVA : ");
    Lire (TauxTVA);
    PrixTTC ← PrixHT * ( 1 + TauxTVA ) * Nb_articles;
    Ecrire ( " Le Prix Total TTC est : ", PrixTTC);
Fin.
Program Affiche_PrixTTC;
uses wincrt;
var
    PrixHT, Nb_Articles, PrixTTC, TauxTVA : real;
Begin
    write( " Donner le Prix Hors Taxe : " );
    readln(PrixHT);
    write( " Donner le Nombre d'articles : " );
    readln(Nb_Articles);
    write( " Donner le Taux TVA : " );
```

```
    readln(TauxTVA);
    PrixTTC := PrixHT * ( 1 + TauxTVA ) * Nb_articles;
    write( ' Le Prix Total TTC est : ', PrixTTC);
End.
```

## 2.2.1 Action Conditionnelle

### 2.2.1.1 Alternative

Il existe trois sortes d'actions alternatives [2, 3, 6, 10] :

- action alternative simple :
- action alternative complexe :
- action alternative multiple (Selon Que Cas)

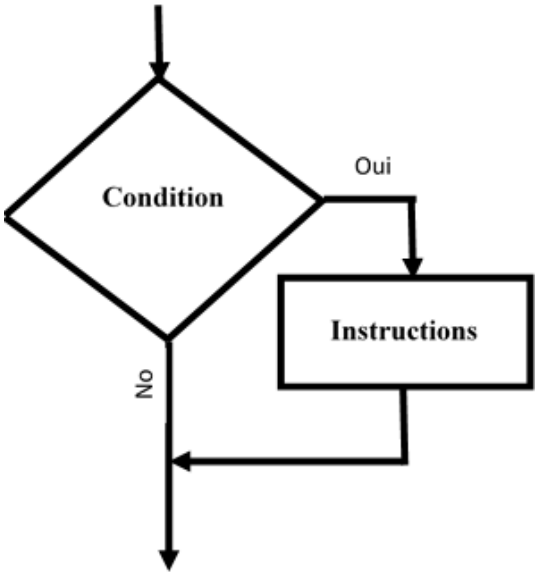



FIGURE 2.1 – Exemple d'Action Alternative Simple.

### 2.2.1.2 Alternative simple :

**Si** Condition
**Alors**
    Instructions
**Finsi**

### 2.2.1.3 Alternative multiple :

**Si** Condition
**Alors**

Instruction 1

**Sinon**

Instruction 2

**Finsi**

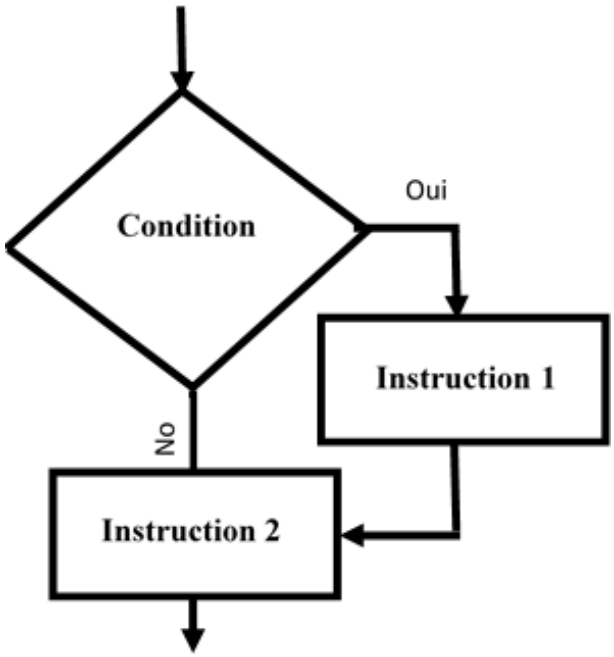



FIGURE 2.2 – Exemple d'Action Alternative complexe.

**Exemple :**

Ecrire un algorithme (ou un programme) qui récupère la moyenne d'un étudiant en fin d'année et qui, suivant que celle-ci soit inférieure à 10 ou non, affiche le message "Ajourné" ou "Admis".

**Algorithme** Affiche_Moy ;

**Var**

Moy : réel ;

**DÃ©but**

Écrire( ' Donner la Moyenne : ' ) ;

Lire(Moy) ;

**SI** Moy >= 10.0

**alors**

Ecrire ( ' Admis (e) ' ) ;

**sinon**

Ecrire ( ' Ajourné (e) ' ) ;

**FinSi**

**Fin**

**Exemple :**

Lire et la Moyenne et Afficher le résultat d'un étudiant.

**Algorithme** Affiche_Moy ;
**Var**
Moy : réeel ;
**Début**
    Écrire( ' Donner la Moyenne : ' ) ;
    Lire(Moy) ;
    **SI** Moy >= 10.0
    **alors**
        Ecrire ( ' Admis (e) ' ) ;
    **sinon**
        Ecrire ( ' Ajourné (e) ' ) ;
    **FinSi**
**Fin**

**Exemple :**

Ecrire un algorithme (ou un programme) qui récupère la moyenne d'un étudiant en fin d'année et qui, suivant que celle-ci soit inférieure à 10 ou non, affiche le message "Ajourné" ou "Admis".

**Program** Affiche_Moy ;
**uses** wincrt ;
**var**
    Moy : real ;
**Begin**
    write( ' Donner la Moyenne : ' ) ;
    readln(Moy) ;
    **if** Moy >= 10.0 **then**
        writeln(' ==============> Admis (e)')
    **else**
        writeln(' ==============> AjournÃ© (e)')
**End.**

**Exemple :**

Ecrire un algorithme (ou un programme) qui demande à l'utilisateur d'introduire deux nombres non nuls et sans faire la multiplication l'informe si leur produit est positif ou négatif.

**Algorithme** Signe_Produit_2_Nbres_Non_Nuls ;
**Var** Nbre1, Nbre2 : réel ;
**Début**

```
Ecrire (' Donner 2 Nombres non Nuls : ' ) ;
Lire (Nbre1, Nbre2) ;
Si (Nbre1 < 0) alors
    Si (Nbre2 < 0) alors
        Ecrire ( ' Produit des 2 Nombres est Positif ' ) ;
    Sinon
        Ecrire( ' Produit des 2 Nombres est NÃ©gatif ' ) ;
    Finsi
Sinon
    Si (Nbre2 < 0) alors
        Ecrire ( ' Produit des 2 Nombres est Négatif ' ) ;
    Sinon
        Ecrire ( ' Produit des 2 Nombres est Positif ' ) ;
    Finsi
Finsi
Ecire(b) ;
```

**Fin.**

**Exemple :**

Ecrire un algorithme (ou un programme) qui demande à l'utilisateur d'introduire trois valeurs entières puis affiche le maximum entre ces trois valeurs.

```
Algorithme Max_3_Nbres ;
Var Nbre1, Nbre2 : réel ;
Début
    Ecrire ( ' Donner 3 Nombres :' ) ;
    Lire (Nbre1, Nbre2, Nbre3) ;
    Si (Nbre1 < Nbre2) alors
        Si (Nbre3 < Nbre2) alors
            Ecrire ( ' Le Max des 3 Nbres est : ' , Nbre2) ;
        Sinon
            Ecrire (' Le Max des 3 Nbres est : ' , Nbre3) ;
        Finsi
    Sinon
        Si (Nbre3 < Nbre1) alors
            Ecrire ( ' Le Max des 3 Nbres est : ' , Nbre1) ;
        Sinon
            Ecrire ( ' Le Max des 3 Nbres est : ' , Nbre3) ;
        Finsi
    Finsi
```

Ecire(b) ;
**Fin.**
**Selon que Cas**
ExpressionBooléenne
**Cas 1** Instruction 1
**Cas 2** Instruction 2
. . . . . .
**Autrement** Instruction N
**FinSelonQue**

### 2.2.2 Action Itérative

Il existe trois actions itératives différentes [2, 3, 6, 10] :
- Boucle Tant Que
- Boucle Pour Faire
- Boucle Répéter
- Répétition

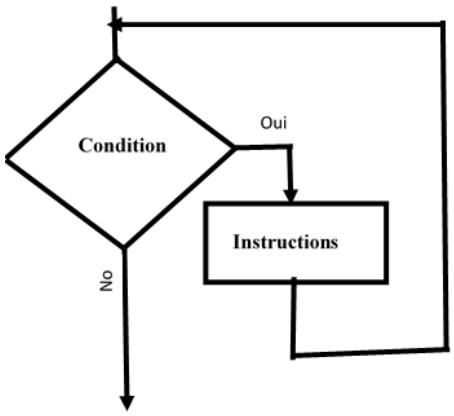



FIGURE 2.3 – Exemple d'Action Itérative.

#### 2.2.2.1 Action Itérative : Boucle Tant Que

L'instruction **exit** permet d'arrêter la répétition.

o **le bloc d'instruction peut ne pas être exécuté :**

**Tant Que** Condition **Faire**

    Instructions

**FinTantQue**

**Exemple :**

Ecrire un algorithme (ou un programme) qui demande un nombre compris entre 20 et 30, jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 30, on fera apparaître un message : Plus petit , et inversement, Plus grand si le nombre est inférieur à 20. Une fois la réponse convienne, il affiche "MerciÂ ".

```
Algorithme Nbre_entre_20_30 ;
Var Nbre : réel ;
Début
    Ecrire ( ' Donner 1 Nombre : ' ) ;
    Lire (Nbre) ;
    TantQue Non((Nbre > 20) et (Nbre < 30)) Faire
        Si (Nbre < 20) alors
            Ecrire ( ' Plus Grand ! ' ) ;
        Sinon
            Ecrire ( ' Plus Petit ! ' ) ;
        Finsi
        Lire (Nbre) ;
    FinTantQue
    Ecrire ( ' Merci !!! ' ) ;
    Ecire(b) ;
Fin.
```

**Programme Pascal :**

```
Program Nbre_entre_20_30 ;
uses wincrt ;
var
    Nbre : integer ;
Begin
    write( ' Donner Un Nombre : ' ) ;
    readln(Nbre) ;
    While (Nbre < 20) or (Nbre > 30) Do
    begin
        if Nbre < 20 then
            writeln( ' Plus Grand ! ' )
```

```
            else
                writeln( ' Plus Petit ! ' ) ;
            write( ' Donner Un Nombre : ' ) ;
            readln(Nbre)
        end ;
        writeln( ' Merci !!! ' )
    End.
```

#### 2.2.2.2 Action Itérative : Boucle Pour

o **le bloc d'instruction peut ne pas être exécuté et il y a une variable indicatrice :**

**Pour** Compteur **Allant de** CompteurDébut **à** CompteurFin **Faire**
    Instructions
**FinPour**

**Exemple :**

Ecrire un algorithme (ou un programme) qui demandeÂ un nombre entier en entrée, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suitÂ dans le cas où l'utilisateur entre le nombre 7 :

7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70

**Algorithme avec la Boucle Pour :**

```
Algorithme Table_Multiplication_Nbre ;
Var
    Nbre , i : entier ;
Début
    Ecrire ( ' Donner 1 Nombre Entier : ' ) ;
    Lire (Nbre) ;
    Pour i := 1 à 10 Faire
        Ecrire (Nbre,  , i , ' = ' Nbre * i) ;
    FinPour
Fin.
Program Table_Multiplication_Nbre ;
uses wincrt ;
var
```

```
    Nbre , i : integer ;
Begin
    write( ' Donner Un Nombre Entier : ' ) ;
    readln(Nbre) ;
    For i := 1 to Nbre Do
        writeln( Nbre,  , i, -, Nbre * i)
End.
Algorithme Factoriel_N ;
Var
    N , i , Fact : entier ;
Début
    Ecrire ( ' Donner 1 Nombre Entier :' ) ;
    Lire (N) ;
    Fact ← 1 ;
    Pour i := 1 à N Faire
        Fact ← Fact * i ;
    FinPour
    Ecrire ( ' Le Factoriel de ' , N , ' est : ', Fact) ;
Fin.
Program Factoriel_N ;
uses wincrt ;
var
    N , i , Fact : integer ;
Begin
    write( ' Donner Un Nombre Entier : ' ) ;
    readln(N) ;
    Fact := 1 ;
    For i := 1 to N Do
        Fact := Fact * i ;
    writeln( ' Le Factoriel de ' , N, ' est : ', Fact)
End.
Program Factoriel_N ;
Var
    N , i , Fact : integer ;
Begin
    write( ' Donner Un Nombre Entier : ' ) ;
    readln(N) ;
    Fact := 1 ;
```

```
    For i := N Downto 1 Do
        Fact := Fact * i;
    writeln(' Le Factoriel de ', N, ' est : ', Fact)
End.
Program Factoriel_N_While;
uses wincrt;
var
    N , i , Fact : integer;
Begin
    write( ' Donner Un Nombre Entier : ' );
    readln(N);
    Fact := 1;
    i := 1;
    While i <= N Do
    Begin
        Fact := Fact * i;
        i := i + 1
    end;
    writeln( ' Le Factoriel de ', N, ' est : ', Fact)
End.
```

#### 2.2.2.3 Action Itérative : Boucle While ...Do

```
Program Table_Multiplication_Nbre;
uses wincrt;
var
    Nbre , i : integer;
Begin
    write( ' Donner Un Nombre Entier : ' );
    readln(Nbre);
    i := 1;
    While ( i <= Nbre) Do
    begin
        writeln( Nbre,  , i, -, Nbre * i);
        i := i + 1
    end
End.
```

#### 2.2.2.4 Action Itérative : Boucle Répéter

o **le bloc d'instruction est exécuté au moins une fois :**

**Répéter**
    Instructions
**Jusqu'à** ExpressionBooléenne
**FinRépéter**

**Programme Pascal avec la boucle Repeat ... Until :**

```
Program Nbre_entre_20_30_Repeat;
uses wincrt;
var
    Nbre : integer;
Begin
    write('Donner Un Nombre : ');
    readln(Nbre);
    Repeat
        if Nbre < 20 then
            writeln('Plus Grand!')
        else
            writeln('Plus Petit!');
        write('Donner Un Nombre : ');
        readln(Nbre)
    Until (Nbre > 20) and (Nbre < 30);
    writeln('Merci!!!')
End.
```

# Chapitre 3

# Structures de Données

## Summary



## 3.1 Tableaux

Un tableau est une table d'association à clé unique telle que [4, 5, 10, 13] :

o le nombre d'éléments de la table (dimension ou taille) est constant,

o l'accès aux éléments s'effectue directement par la clé,

o les valeurs minimum et maximum des clés sont des constantes.

On écrit en algorithmique

$$nom\_tableau = \textbf{tableau}[min\_indice..max\_indice]de\textbf{type\_predefini};$$

ce qui signifie que [2, 3, 6, 10] :

- les éléments ont pour type le type_predefini

- les indices des éléments vont de min_indice à max_indice, avec :

$$min\_indice < max\_indice$$

La taille du tableau est donc :

$$max\_indice - min\_indice + 1.$$

Pour accéder à un élément d'un tableau T d'indice I, on écrit

$$T[I].$$

La complexité de l'accès à un élément du tableau est

$$O(1).$$

### 3.1.1 Somme des éléments d'un tableau d'entiers

**fonction***somme*(**ref***T* : **tableau**[*min_indice..max_indice*]*d'***entiers**) : **entier**;

var s,i :entier ;
**début**
s=0 ;
**pour** i **allant de** min_indice **à** max_indice **faire**
s=s+T[i]
**finpour**
retourner(s)
**fin**
Complexité : O(n)

### 3.1.2 Recherche d'un élément

**Propriété**
Soit i,j deux entiers, i<=j.
Soit T un tableau d'éléments d'indice variant entre i et j. Pour tout élément e, appartenant au tableau T, on a T[i]=e ou e est dans T[i+1..j]

**fonction***cherche*(**ref***T* : **tableau**[*min_indice..max_indice*]*d'lment*; **val***e* : *lment*) : **entier**;

var
i : entier ;
**début**
**pour** i **allant de** min_indice **à** max_indice **faire**
si T[i]==e alors
retourner(i)
**finpour**
retourner()
**fin**
**finfonction**
Complexité minimum : O(1) maximum : O(n) Source Python

### 3.1.3 Recherche de l'indice du premier élément minimum

On suppose que le tableau contient des éléments comparables (l'ensemble des éléments est muni d'une relation d'ordre).
Choisissons ici, pour simplifier les notations, des entiers.
**Propriété**
Soit i,j deux entiers, i<=j.
Soit T un tableau d'entiers d'indice variant entre i et j.
Soit m l'élément minimum du tableau, on a T[i]=m ou m est dans T[i+1..j]

**fonction***minimum*(**ref***T* : **tableau**[*min_indice..max_indice*]*d'***entier**) : **entier**;

var
i,sauv :entier ;
**début**
sauv= min_indice ;
**pour** i **allant de** min_indice+1 **à** max_indice **faire**
si T[i]

## 3.2 Matrices

### Les **Déclaration**

Une matrice M de dimension nxm est un tableau de dimension n dont chaque élément est un tableau de dimension m [4, 5, 10, 13].

On peut donc déclarer la matrice sous la forme suivante :

$$varM : tableau[1..n]detableau[1..m]d'lments;$$

### 3.2.1 Initialisation

**fonction***initMatrice*(**ref***M* : **tableau**[1..*n*]**de tableau**[1..*m*]

*d'lments*; **val***valeurInitiale* : *lment*) : **vide**;

```
var i,j :entier ;
début
pour i allant de 1 à n faire
pour j allant de 1 à m faire
M[i][j]=valeurInitiale
finpour
finpour
retourner()
finfonction
Complexité : O(nm)
```

### 3.2.2 Somme de deux matrices réelles

Les

**fonction** $sommeMatrice$(**ref** $M1, M2$ : **tableau**[1..$n$]**de tableau**[1..$m$]$de$**réels**) :

**tableau**[1..$n$]$de$**tableau**[1..$m$]$de$**réels**;

var i,j :entier ;
var M :tableau[1..n] de tableau [1..m] de réels ;
**début**
pour i allant de 1 à n faire
pour j allant de 1 à m faire
M[i][j]=M1[i][j]+M2[i][j] ;
finpour
finpour
retourner(M)
finfonction
Complexité : O(nm)

# Chapitre 4

# Notions de Procédures & Fonctions

## Summary



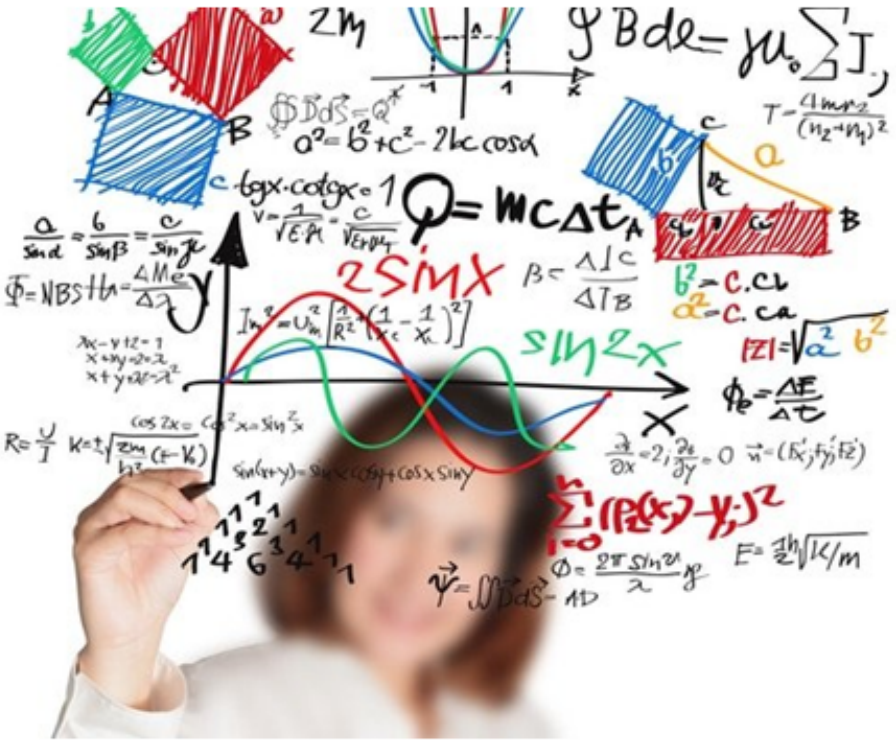


FIGURE 4.1 – L'abstraction du Monde Réel.

## 4.1 Procédures

Les procédures[1] sont des sous-programmes écrits avant le programme principal mais appelés depuis ce programme principal, d'une autre procédure ou même d'une autre fonction. Le nom d'une procédure ou d'une fonction est un identifiant [4, 5, 10, 13, 14].

Procedure

Syntaxe :

**Program** nom_de_programme ;
**Procedure** nom_de_procédure ; Déclaration de la procédure
**begin**
    ...
    instructions
    ...
**end ;**
**begin ;** Début du programme principal
    ...
    nom de procédure ; appel de la procédure
    ...
**end.**

**Exemple :**

```
Program test ;
    Procedure AfficheDate ;
    begin
        writeln(DateTimeToStr(Now)) ;
    end ;
begin ;
    write('Début : ') ;AfficheDate ;
    writeln('Suite des instuctions ') ;
    write(' Fin : ') ;AfficheDate ;
end.
```

On peut déclarer des variables locales à une procédure ou une fonction. La portée de ces variables sera locale à la procédure ou á la fonction [2, 3, 6, 10, 15].

1. https ://www.bonneville.nom.fr/enseignement/pascal/cours/procedures-et-fonctions-1-16.htm

**Exemple**

```
Program Dessine_Carre_Chiffres ;
    Procedure Ligne ; {dessine une ligne de 5 chiffres}
    var i : integer ;  la variable i est locale à la procédure Ligne}
    begin
        for i :=1 to 5 do write(i) ;
    end ;
var i : integer ;
begin
    for i :=1 to 5 do {dessine 5 lignes de 5 chiffres}
    begin
        Ligne ; writeln ;
    end ;
end.
```

## 4.2 Fonctions

Une fonction est une section d'algorithme qui a un objectif bien défini et un nom. En général, elle communique avec l'extérieur par le biais de paramètres typés. Elle possède des variables locales qui ne sont pas visibles à l'extérieur de la fonction. Ces variables peuvent être des fonctions [2, 3, 6, 10, 17]. Une fonction retourne une valeur par l'instruction simple retourne(Expression) [4, 5, 10, 13, 16]. L'expression peut être

o vide, tout s'est bien passé mais il n'y a pas de résultat à retourner : retourne()

o sans résultat, il est impossible de retourner un résultat suite à un cas de figure de l'instance : retourne(NUL)

**Syntaxe**

- Ecriture de la fonction

**fonction***NomDeFonction*(*ListeParametres*) : **TypeResultat**;

// déclarations des variables ou fonctions locales autres que les paramètres
**début**
// partie instruction qui contient l'appel à retourne
**fin**

**- liste des paramètres**

Les paramètres sont passés

o par **référence** ref, on écrit

**ref***ListeVariable* : **NomDeType**;

la fonction travaille directement dans la variable passée en paramètre,

o par **valeur** val, on écrit

**val***ListeVariable* : **NomDeType**;

la fonction travaille sur une copie de la variable passée en paramètre.

Le type du résultat est vide si la fonction ne renvoie pas de résultat.

**Utilisation**

Une fonction s'utilise en écrivant

**NomDeFonction**( *ListeInstanceParamtres* )

* dans le calcul d'une expression si la fonction retourne une valeur,
* comme une instruction simple si elle ne retourne pas de valeur.

## 4.3 Exemple

**fonction** *exemple*(**val** $n$ : **entier**; **ref** $m$ : **entier**) : **vide**;

**début**

n :=5 ;

m :=7 ;

**fin**

**fin fonction**

Supposons que l'on ait la séquence suivante :

**var** $p, q$ : **entier**;

**début**

p=1 ;

q=2 ;

exemple(p,q) ;

**fin**

Après exécution p contiendra 1 et q contiendra 7 (Animation ici)

# Conclusion & Perspectives

Cette section clôt ce polycopié de cours d'initiation à l'algorithmique et à la programmation. Tout au long de ce polycopié, nous avons présenté les quatre chapitres qui le composent.

Le chapitre 1 est constitué des concepts de données, de variables, des différents types de variables, d'initialisation, d'affectation, d'incrémentation, de décrémentation d'une valeur ou d'une expression à une variable. Ensuite, nous sommes passé à l'évaluation des différents types d'expressions en respectant l'ordre de priorité des opérateurs. Tout cela est suivi par les opérations d'entrée / sortie.

Le chapitre 2 représente les structures de contrôle en détaillant toute sorte d'action, à commecer par les actions alternatives, avec les actions alternatives simples (**Si Alors Finsi**) et complexes (**Si Alors Sinon ... Finsi**), suivies par les actions itératives avec la boucle **Tant Que**, la boucle **Pour** et la boucle **Répéter**.

Le chapitre 3 est constitué de l'ensemble de structures de données telles que tableau à une dimension (tableau) et tableau à deux dimensions (matrice). Le chapitre 4 est constitué de l'ensemble des procédures et de Fonctions .

Quant aux perspectives, nous comptons introduire tous les cours que j'ai assuté tout au long de ma carriÃ¨re d'enseignant-chercheur au niveau de l'université algérienne, à savoir les langages de **Pascal**, de **Matlab**, de **SPSS**, de **Java** avec **Swing**, de **Python**, de **Java Script**, de **HTML et CSS**, de **PSP**, ...

# Références


[1] Rifflet Jean-Marie, Yunès Jean-Baptiste, (2014), Fondements de la programmation (concepts et techniques), édition Ellipses, 2014

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein, (2001), Introduction to Algorithms. MIT Press et McGraw-Hill edition. ISBN : 978-0-262-03293-3, 2001.

[3] Thomas Cormen, Charles Leiserson, Ronald Rivest, Clifford Stein, (2010), Algorithmique, Cours avec 957 exercices et 158 problèmes, $3^e$ édition, 23 juin 2010, Édition : Dunod

[4] Mc Belaid, (2008), Algorithmique et structures de données. Edition les pages bleus. ISBN : 978-9961-734-93-2, 2008.

[5] Mc Belaid, (2006), Algorithme et Programmation en Pascal. Edition les pages bleus. ISBN : 978-9961-734-68-8, 2006.

[6] Thomas H.Cormen, Charles E.Leiserson, Ronald L. Rivest et Clifford Stein, (2002), Introduction à l'algorithmique, $2^e$ cycle - Ecoles d'ingénieurs, $2^e$ édition, $1^{er}$ octobre 2002, Édition : Dunod

[7] Laurent Debrauwer, (2008), Algorithmique - Travaux Pratiques, Entraînez-vous et améliorez votre pratique de la programmation, $1^{re}$ édition, 1er octobre 2008, Édition : ENI

[8] Sébastien Rohaut, (2007), Algorithmique, Techniques fondamentales de programmation, $1^{re}$ édition, $1^{er}$ Juillet 2007, Édition : Eyrolles

[9] Claude Bauer Pierre Vicenti, (2005), Le langage Pascal appliqué à l'algorithmique - Cours et exercices corrigés, juillet 2005 Scolaire / Universitaire, édition Ellipses 2005.

[10] Michel Divay, (2004), Algorithmes et structures de données génériques. Edition Dunod, 2004.

[11] Pascal Lienhardt, (2022), Bases en algorithmique et en programmation : cours et exercices corrigés, References Sciences, 13 Septembre 2022, Sciences & Techniques, édition Ellipses

[12] Djelloul Bouchiha, (2019), Notions Avancées sur l'Algorithmique et la Programmation en Pascal, Editions Universitaires Européennes

[13] Jacques Courtin, (1998), Initiation à l'algorithmique et aux structures de données. Edition Dunod, 1998.

[14] Patrick Bosc, Marc Guyomard et Laurent Miclet, (2019), Conception d'algorithmes, Principes et 150 exercices corrigés, $2^e$ édition, 10 janvier 2019, Édition : Eyrolles

[15] Patrick Bosc, Marc Guyomard et Laurent Miclet, (2021), Conception d'algorithmes, 150 exercices corrigés, $3^e$ édition, 7 janvier 2021, Édition : Eyrolles

[16] Djelloul Bouchiha, (2020), Algorithmique et programmation en Pascal : Cours avec 190 exercices corrigés, Editions Universitaires Européennes

[17] Patrick Cousot, (1992), Algorithmique et programmation en Pascal, Editeur : Ellipses