Licence Professionnelle Modélisation et Gestion de l'environnement

Cours d'Algorithmique

Pr. Lamia ZIAD
Ecole Superieure de Technologie d'Essaouira

Contents

Chapter 1

Introduction a l'Algorithmique

Definition

Un **algorithme** est une suite finie et non ambigue d'instructions permettant de resoudre un probleme ou d'atteindre un objectif.

1.1 Qu'est-ce qu'un Algorithme?

- **Sequence d'operations** : Etapes ordonnees
- **Determinisme** : Memes entrees = memes resultats
- **Finitude** : Nombre fini d'etapes
- **Effectivite** : Chaque operation est executable

1.2 Proprietes d'un Bon Algorithme

Important

Un bon algorithme doit etre : Correct, Clair, Efficace, Robust et General.

Exemple

Exemple d'algorithme simple : Calcul de la moyenne de deux nombres

```
debut
    lire a
    lire b
    moyenne $\Leftarrow$ (a + b) / 2
    ecrire "La moyenne est : ", moyenne
fin
```

Listing 1.1: Algorithme de calcul de moyenne

1.3 Representation des Algorithmes

- **Pseudo-code** : Langage intermediaire comprehensible
- **Organigrammes** : Representation graphique
- **Langages de programmation** : Implementation concrete

Chapter 2

Variables et Types de Donnees

2.1 Concept de Variable

Definition

Une **variable** est une zone memoire nommee servant a stocker une valeur qui peut etre modifiee pendant l'execution.

Exemple

```
debut
    // Declaration de variables
    entier age
    reel salaire
    chaine nom
    booleen estActif

    // Affectation de valeurs
    age $\Leftarrow$ 25
    salaire $\Leftarrow$ 2500.50
    nom $\Leftarrow$ "Jean Dupont"
    estActif $\Leftarrow$ vrai

    // Affichage
    ecrire "Nom: ", nom
    ecrire "Age: ", age
    ecrire "Salaire: ", salaire
fin
```

Listing 2.1: Declaration et utilisation de variables

2.2 Types de Donnees Fondamentaux

Type	Description	Exemple
entier	Nombres entiers	42, -15, 0
reel	Nombres a virgule	3.14, -2.5, 0.0
caractere	Un seul caractere	'A', '5', '$'
chaine	Suite de caracteres	"Bonjour", "ABC123"
booleen	Valeur logique	vrai, faux

2.3 Operateurs

Important

Les operateurs permettent de manipuler les valeurs des variables.

2.3.1 Operateurs Arithmetiques

Operateur	Operation	Exemple
+	Addition	a + b
-	Soustraction	a - b
*	Multiplication	a * b
/	Division	a / b
mod	Modulo	a mod b

2.3.2 Operateurs de Comparaison

Operateur	Signification	Exemple
=	Egal a	a = b
<> ou !=	Different de	a <> b
<	Inferieur a	a < b
>	Superieur a	a > b
<=	Inferieur ou egal	a <= b
>=	Superieur ou egal	a >= b

2.3.3 Operateurs Logiques

Operateur	**Signification**	**Exemple**
et	ET logique	(a $>$ 0) et (b $<$ 10)
ou	OU logique	(a = 0) ou (b = 0)
non	Negation	non (a = b)

Chapter 3

Structures de Controle

Definition

Les **structures de controle** determinent l'ordre d'execution des instructions dans un algorithme.

3.1 Structure Sequence

Important

Les instructions sont executees dans l'ordre ou elles sont ecrites.

Exemple

```
debut
    // Instructions executees dans l'ordre
    entier a, b, resultat
    a $\Leftarrow$ 10
    b $\Leftarrow$ 5
    resultat $\Leftarrow$ a + b
    ecrire "Resultat: ", resultat
fin
```

Listing 3.1: Structure sequentielle

3.2 Structure Conditionnelle SI

Exemple

```
debut
    entier age
    ecrire "Entrez votre age: "
    lire age

    si age >= 18 alors
        ecrire "Vous etes majeur"
    sinon
        ecrire "Vous etes mineur"
    finsi
fin
```

Listing 3.2: Structure conditionnelle simple

3.3 Structure Conditionnelle Multiple

Exemple

```
debut
    entier note
    ecrire "Entrez la note: "
    lire note

    si note >= 16 alors
        ecrire "Tres bien"
    sinon si note >= 14 alors
        ecrire "Bien"
    sinon si note >= 12 alors
        ecrire "Assez bien"
    sinon
        ecrire "Insuffisant"
    finsi
fin
```

Listing 3.3: Structure conditionnelle multiple

3.4 Structure SELON (Switch)

Exemple

```
debut
    entier choix
    ecrire "Choisissez une option (1-3): "
    lire choix

    selon choix
        cas 1:
            ecrire "Option 1 selectionnee"
        cas 2:
            ecrire "Option 2 selectionnee"
        cas 3:
            ecrire "Option 3 selectionnee"
        autre:
            ecrire "Option invalide"
    finselon
fin
```

Listing 3.4: Structure de selection multiple

Chapter 4

Structures Repetitives (Boucles)

4.1 Boucle POUR

Important

La boucle POUR execute un bloc d'instructions un nombre determine de fois.

Exemple

```
debut
    entier i, somme
    somme $\Leftarrow$ 0

    pour i de 1 a 10 faire
        somme $\Leftarrow$ somme + i
        ecrire "i = ", i, ", somme = ", somme
    finpour

    ecrire "Somme totale: ", somme
fin
```

Listing 4.1: Boucle POUR simple

4.2 Boucle TANT QUE

Exemple

```
debut
    entier nombre, somme
    somme $\Leftarrow$ 0

    ecrire "Entrez un nombre (0 pour terminer): "
    lire nombre

    tant que nombre <> 0 faire
        somme $\Leftarrow$ somme + nombre
        ecrire "Entrez un nombre (0 pour terminer): "
        lire nombre
    fin tant que

    ecrire "Somme des nombres: ", somme
fin
```

Listing 4.2: Boucle TANT QUE

4.3 Boucle REPETER

Important

La boucle REPETER execute au moins une fois le bloc d'instructions.

Exemple

```
debut
    entier nombre

    repeter
        ecrire "Entrez un nombre positif: "
        lire nombre
    jusqua nombre > 0

    ecrire "Nombre valide: ", nombre
fin
```

Listing 4.3: Boucle REPETER

4.4 Instructions de Controle de Boucle

Exemple

```
debut
    entier i

    pour i de 1 a 20 faire
        // Saute les nombres pairs
        si i mod 2 = 0 alors
            continuer
        finsi

        // Arrete si on depasse 15
        si i > 15 alors
            arreter
        finsi

        ecrire "Nombre impair: ", i
    finpour
fin
```

Listing 4.4: Utilisation de CONTINUER et ARRETER

Chapter 5

Structures de Donnees : Tableaux

Definition

Un **tableau** est une structure de donnees contenant plusieurs elements de meme type, accessibles par un indice.

5.1 Declaration et Utilisation

Exemple

```
debut
    // Declaration d'un tableau de 5 entiers
    entier nombres[5]
    entier i, somme

    // Saisie des elements
    pour i de 0 a 4 faire
        ecrire "Entrez le nombre ", i+1, ": "
        lire nombres[i]
    finpour

    // Calcul de la somme
    somme $\Leftarrow$ 0
    pour i de 0 a 4 faire
        somme $\Leftarrow$ somme + nombres[i]
    finpour

    // Affichage du resultat
    ecrire "Somme: ", somme
    ecrire "Moyenne: ", somme / 5
fin
```

Listing 5.1: Manipulation de tableaux

5.2 Algorithmes Classiques sur Tableaux

Exemple

```
debut
    entier tableau[10]
    entier i, maximum

    // Initialisation du tableau
    pour i de 0 a 9 faire
        tableau[i] $\Leftarrow$ aleatoire(1, 100)
    finpour

    // Recherche du maximum
    maximum $\Leftarrow$ tableau[0]
    pour i de 1 a 9 faire
        si tableau[i] > maximum alors
            maximum $\Leftarrow$ tableau[i]
        finsi
    finpour

    ecrire "Le maximum est: ", maximum
fin
```

Listing 5.2: Recherche du maximum dans un tableau

5.3 Tableaux Multidimensionnels

Exemple

```
debut
    // Declaration d'une matrice 3x3
    entier matrice[3][3]
    entier i, j, somme

    // Remplissage de la matrice
    pour i de 0 a 2 faire
        pour j de 0 a 2 faire
            matrice[i][j] $\Leftarrow$ (i + 1) * (j + 1)
        finpour
    finpour

    // Calcul de la somme des elements
    somme $\Leftarrow$ 0
    pour i de 0 a 2 faire
        pour j de 0 a 2 faire
            somme $\Leftarrow$ somme + matrice[i][j]
        finpour
    finpour

    ecrire "Somme des elements: ", somme
fin
```

Listing 5.3: Manipulation de matrice

Chapter 6

Modularisation : Procedures et Fonctions

Definition

La **modularisation** consiste a decomposer un probleme complexe en sous-problemes plus simples, implementes sous forme de procedures ou fonctions.

6.1 Procedures

Important

Une procedure est un sous-programme qui effectue une tache sans retourner de valeur.

Exemple

```
// Definition de la procedure
procedure afficherMenu()
debut
    ecrire "=== MENU PRINCIPAL ==="
    ecrire "1. Ajouter un element"
    ecrire "2. Supprimer un element"
    ecrire "3. Afficher les elements"
    ecrire "4. Quitter"
    ecrire "======================"
fin procedure

// Programme principal
debut
    entier choix

    repeter
        afficherMenu()  // Appel de la procedure
        ecrire "Votre choix: "
        lire choix

        // Traitement du choix...
    jusqua choix = 4
fin
```

Listing 6.1: Definition et utilisation d'une procedure

6.2 Fonctions

Important

Une fonction est un sous-programme qui retourne une valeur.

Exemple

```
// Fonction qui calcule la factorielle
fonction factorielle(n: entier): entier
debut
    entier i, resultat
    resultat $\Leftarrow$ 1

    pour i de 1 a n faire
        resultat $\Leftarrow$ resultat * i
    finpour

    retourner resultat
fin fonction

// Fonction qui verifie si un nombre est premier
fonction estPremier(n: entier): booleen
debut
    entier i

    si n < 2 alors
        retourner faux
    finsi

    pour i de 2 a racine(n) faire
        si n mod i = 0 alors
            retourner faux
        finsi
    finpour

    retourner vrai
fin fonction

// Programme principal
debut
    entier nombre

    ecrire "Entrez un nombre: "
    lire nombre

    ecrire nombre, "! = ", factorielle(nombre)

    si estPremier(nombre) alors
        ecrire nombre, " est premier"
    sinon
        ecrire nombre, " n'est pas premier"
    finsi
fin
```

6.3 Passage de Parametres

Exemple

```
// Passage par valeur (copie)
procedure incrementerParValeur(x: entier)
debut
    x $\Leftarrow$ x + 1
    ecrire "Dans la procedure: x = ", x
fin procedure

// Passage par reference (original)
procedure incrementerParReference(ref x: entier)
debut
    x $\Leftarrow$ x + 1
    ecrire "Dans la procedure: x = ", x
fin procedure

// Programme principal
debut
    entier a $\Leftarrow$ 5

    ecrire "Avant appel: a = ", a
    incrementerParValeur(a)
    ecrire "Apres appel par valeur: a = ", a

    incrementerParReference(a)
    ecrire "Apres appel par reference: a = ", a
fin
```

Listing 6.3: Passage de parametres par valeur et par reference

Chapter 7

Complexite Algorithmique

Definition

La **complexite algorithmique** mesure l'efficacite d'un algorithme en termes de temps d'execution et d'espace memoire utilises.

7.1 Notation Grand O

Important

La notation O (grand O) decrit le comportement asymptotique d'un algorithme dans le pire cas.

Complexite	**Notation**	**Exemple**
Constante	O(1)	Acces a un element de tableau
Logarithmique	O(log n)	Recherche binaire
Lineaire	O(n)	Parcours de tableau
Linearithmique	O(n log n)	Tri fusion
Quadratique	O(n^2)	Tri bulle, deux boucles imbriquees
Cubique	O(n^3)	Trois boucles imbriquees
Exponentielle	O(2^n)	Probleme du voyageur de commerce

7.2 Analyse de Complexite

Exemple

```
// Complexite O(1) - Constant
fonction accesElement(t: tableau, i: entier): entier
debut
    retourner t[i]
fin fonction

// Complexite O(n) - Lineaire
fonction rechercheLineaire(t: tableau, n: entier, x: entier): entier
debut
    entier i
    pour i de 0 a n-1 faire
        si t[i] = x alors
            retourner i
        finsi
    finpour
    retourner -1
fin fonction

// Complexite O($n^2$) - Quadratique
procedure triBulles(ref t: tableau, n: entier)
debut
    entier i, j, temp
    pour i de 0 a n-2 faire
        pour j de 0 a n-i-2 faire
            si t[j] > t[j+1] alors
                temp $\Leftarrow$ t[j]
                t[j] $\Leftarrow$ t[j+1]
                t[j+1] $\Leftarrow$ temp
            finsi
        finpour
    finpour
fin procedure
```

Listing 7.1: Algorithmes avec differentes complexites

7.3 Regles de Calcul de Complexite

- **Sequences** : O(max(f(n), g(n)))

- **Boucles** : O(nombre d'iterations $\times$ complexite du corps)
- **Instructions conditionnelles** : O(max(bloc alors, bloc sinon))
- **Appels de fonctions** : O(complexite de la fonction)

Chapter 8

Algorithmes de Recherche

8.1 Recherche Lineaire

Important

La recherche lineaire parcourt sequentiellement tous les elements jusqu'a trouver la valeur cherchee.

Exemple

```
fonction rechercheLineaire(t: tableau, n: entier, x: entier): entier
debut
    entier i
    pour i de 0 a n-1 faire
        si t[i] = x alors
            retourner i  // Element trouve
        finsi
    finpour
    retourner -1  // Element non trouve
fin fonction

// Version avec indicateur booleen
fonction contientElement(t: tableau, n: entier, x: entier): booleen
debut
    entier i
    pour i de 0 a n-1 faire
        si t[i] = x alors
            retourner vrai
        finsi
    finpour
    retourner faux
fin fonction
```

Listing 8.1: Recherche lineaire

8.2 Recherche Binaire

Important

La recherche binaire necessite un tableau trie et divise l'espace de recherche par deux a chaque iteration.

Exemple

```
fonction rechercheBinaire(t: tableau, n: entier, x: entier): entier
debut
    entier gauche, droite, milieu
    gauche $\Leftarrow$ 0
    droite $\Leftarrow$ n - 1

    tant que gauche <= droite faire
        milieu $\Leftarrow$ (gauche + droite) div 2

        si t[milieu] = x alors
            retourner milieu
        sinon si t[milieu] < x alors
            gauche $\Leftarrow$ milieu + 1
        sinon
            droite $\Leftarrow$ milieu - 1
        finsi
    fin tant que

    retourner -1
fin fonction
```

Listing 8.2: Recherche binaire iterative

Exemple

```
fonction rechercheBinaireRecursive(t: tableau, x: entier,
                                   gauche: entier, droite: entier):
   entier
debut
    si gauche > droite alors
        retourner -1
    finsi

    entier milieu $\Leftarrow$ (gauche + droite) div 2

    si t[milieu] = x alors
        retourner milieu
    sinon si t[milieu] < x alors
        retourner rechercheBinaireRecursive(t, x, milieu + 1, droite)
    sinon
        retourner rechercheBinaireRecursive(t, x, gauche, milieu - 1)
    finsi
fin fonction
```

Listing 8.3: Recherche binaire recursive

Chapter 9

Algorithmes de Tri

9.1 Tri par Selection

Important

Le tri par selection recherche le minimum et le place a sa position finale.

Exemple

```
procedure triSelection(ref t: tableau, n: entier)
debut
    entier i, j, minIndex, temp

    pour i de 0 a n-2 faire
        minIndex $\Leftarrow$ i

        // Recherche du minimum dans la partie non triee
        pour j de i+1 a n-1 faire
            si t[j] < t[minIndex] alors
                minIndex $\Leftarrow$ j
            finsi
        finpour

        // Echange des elements
        si minIndex <> i alors
            temp $\Leftarrow$ t[i]
            t[i] $\Leftarrow$ t[minIndex]
            t[minIndex] $\Leftarrow$ temp
        finsi
    finpour
fin procedure
```

Listing 9.1: Tri par selection

9.2 Tri par Insertion

Exemple

```
procedure triInsertion(ref t: tableau, n: entier)
debut
    entier i, j, elementCourant

    pour i de 1 a n-1 faire
        elementCourant $\Leftarrow$ t[i]
        j $\Leftarrow$ i - 1

        // Decalage des elements plus grands
        tant que j >= 0 et t[j] > elementCourant faire
            t[j + 1] $\Leftarrow$ t[j]
            j $\Leftarrow$ j - 1
        fin tant que

        // Insertion de l'element courant
        t[j + 1] $\Leftarrow$ elementCourant
    finpour
fin procedure
```

Listing 9.2: Tri par insertion

9.3 Tri a Bulles

Exemple

```
procedure triBulles(ref t: tableau, n: entier)
debut
    entier i, j, temp
    booleen echange

    pour i de 0 a n-2 faire
        echange $\Leftarrow$ faux

        pour j de 0 a n-i-2 faire
            si t[j] > t[j+1] alors
                // Echange des elements adjacents
                temp $\Leftarrow$ t[j]
                t[j] $\Leftarrow$ t[j+1]
                t[j+1] $\Leftarrow$ temp
                echange $\Leftarrow$ vrai
            finsi
        finpour

        // Si aucun echange, le tableau est trie
        si non echange alors
            arreter
        finsi
    finpour
fin procedure
```

Listing 9.3: Tri a bulles

9.4 Comparaison des Algorithmes de Tri

Algorithme	**Complexite temporelle**	**Complexite spatiale**	**Stable**
Selection	$O(n^2)$	O(1)	Non
Insertion	$O(n^2)$	O(1)	Oui
Bulles	$O(n^2)$	O(1)	Oui
Fusion	O(n log n)	O(n)	Oui
Rapide	O(n log n)	O(log n)	Non

Chapter 10

Structures de Donnees Avancees

10.1 Piles (LIFO)

Definition

Une **pile** est une structure LIFO (Last In, First Out) ou le dernier element ajoute est le premier a etre retire.

Exemple

```
constante TAILLE_MAX = 100

structure Pile
    tableau elements[TAILLE_MAX]
    entier sommet
fin structure

procedure initialiserPile(ref p: Pile)
debut
    p.sommet $\Leftarrow$ -1
fin procedure

fonction pileVide(p: Pile): booleen
debut
    retourner p.sommet = -1
fin fonction

fonction pilePleine(p: Pile): booleen
debut
    retourner p.sommet = TAILLE_MAX - 1
fin fonction

procedure empiler(ref p: Pile, x: entier)
debut
    si pilePleine(p) alors
        ecrire "Erreur: Pile pleine"
    sinon
        p.sommet $\Leftarrow$ p.sommet + 1
        p.elements[p.sommet] $\Leftarrow$ x
    finsi
fin procedure

fonction depiler(ref p: Pile): entier
debut
    si pileVide(p) alors
        ecrire "Erreur: Pile vide"
        retourner -1
    sinon
        p.sommet $\Leftarrow$ p.sommet - 1
        retourner p.elements[p.sommet + 1]
    finsi
fin fonction
```

Listing 10.1: Implementation d'une pile

10.2 Files (FIFO)

Definition

Une **file** est une structure FIFO (First In, First Out) ou le premier element ajoute est le premier a etre retire.

Exemple

```
structure File
    tableau elements[TAILLE_MAX]
    entier tete, queue, taille
fin structure

procedure initialiserFile(ref f: File)
debut
    f.tete $\Leftarrow$ 0
    f.queue $\Leftarrow$ -1
    f.taille $\Leftarrow$ 0
fin procedure

fonction fileVide(f: File): booleen
debut
    retourner f.taille = 0
fin fonction

fonction filePleine(f: File): booleen
debut
    retourner f.taille = TAILLE_MAX
fin fonction

procedure enfiler(ref f: File, x: entier)
debut
    si filePleine(f) alors
        ecrire "Erreur: File pleine"
    sinon
        f.queue $\Leftarrow$ (f.queue + 1) mod TAILLE_MAX
        f.elements[f.queue] $\Leftarrow$ x
        f.taille $\Leftarrow$ f.taille + 1
    finsi
fin procedure

fonction defiler(ref f: File): entier
debut
    si fileVide(f) alors
        ecrire "Erreur: File vide"
        retourner -1
    sinon
        entier valeur $\Leftarrow$ f.elements[f.tete]
        f.tete $\Leftarrow$ (f.tete + 1) mod TAILLE_MAX
        f.taille $\Leftarrow$ f.taille - 1
        retourner valeur
    finsi
fin fonction
```

Listing 10.2: Implementation d'une file

10.3 Listes Chainees

Exemple

```
structure Noeud
    entier valeur
    Noeud suivant
fin structure

procedure ajouterDebut(ref tete: Noeud, valeur: entier)
debut
    Noeud nouveau $\Leftarrow$ new Noeud
    nouveau.valeur $\Leftarrow$ valeur
    nouveau.suivant $\Leftarrow$ tete
    tete $\Leftarrow$ nouveau
fin procedure

procedure afficherListe(tete: Noeud)
debut
    Noeud courant $\Leftarrow$ tete

    tant que courant <> NULL faire
        ecrire courant.valeur, " -> "
        courant $\Leftarrow$ courant.suivant
    fin tant que

    ecrire "NULL"
fin procedure

fonction rechercherListe(tete: Noeud, x: entier): booleen
debut
    Noeud courant $\Leftarrow$ tete

    tant que courant <> NULL faire
        si courant.valeur = x alors
            retourner vrai
        finsi
        courant $\Leftarrow$ courant.suivant
    fin tant que

    retourner faux
fin fonction
```

Listing 10.3: Structure de liste chainee

Chapter 11

Algorithmes sur les Chaines de Caracteres

11.1 Recherche de Sous-chaine

Exemple

```
fonction rechercheSousChaine(texte: chaine, motif: chaine): entier
debut
    entier n $\Leftarrow$ longueur(texte)
    entier m $\Leftarrow$ longueur(motif)
    entier i, j

    pour i de 0 a n-m faire
        j $\Leftarrow$ 0
        tant que j < m et texte[i+j] = motif[j] faire
            j $\Leftarrow$ j + 1
        fin tant que

        si j = m alors
            retourner i  // Motif trouve a la position i
        finsi
    finpour

    retourner -1  // Motif non trouve
fin fonction
```

Listing 11.1: Algorithme naif de recherche de sous-chaine

11.2 Inversion de Chaine

Exemple

```
fonction inverserChaine(s: chaine): chaine
debut
    entier n $\Leftarrow$ longueur(s)
    chaine resultat
    entier i

    pour i de n-1 a 0 pas -1 faire
        resultat $\Leftarrow$ resultat + s[i]
    finpour

    retourner resultat
fin fonction

// Version avec echange en place
procedure inverserChaineEnPlace(ref s: chaine)
debut
    entier gauche $\Leftarrow$ 0
    entier droite $\Leftarrow$ longueur(s) - 1
    caractere temp

    tant que gauche < droite faire
        temp $\Leftarrow$ s[gauche]
        s[gauche] $\Leftarrow$ s[droite]
        s[droite] $\Leftarrow$ temp
        gauche $\Leftarrow$ gauche + 1
        droite $\Leftarrow$ droite - 1
    fin tant que
fin procedure
```

Listing 11.2: Algorithme d'inversion de chaine

11.3 Validation de Chaine

Exemple

```
fonction estEmailValide(email: chaine): booleen
debut
    entier i, longueur
    booleen aArobase, aPoint
    longueur $\Leftarrow$ longueur(email)

    si longueur < 5 alors  // a@b.c
        retourner faux
    finsi

    aArobase $\Leftarrow$ faux
    aPoint $\Leftarrow$ faux

    pour i de 0 a longueur-1 faire
        si email[i] = '@' alors
            si aArobase alors  // Plus d'un @
                retourner faux
            finsi
            aArobase $\Leftarrow$  vrai
        sinon si email[i] = '.' alors
            si i > 0 et i < longueur-1 alors
                aPoint $\Leftarrow$ vrai
            finsi
        finsi
    finpour

    retourner aArobase et aPoint
fin fonction
```

Listing 11.3: Validation d'une adresse email