Algorithmique et Structures de Données

Objectifs du cours :

Ce cours permettra aux étudiants de première année 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, comme ça était demandé dans le programme du LMD.

De plus, l'étudiant sera capable de définir la structure de données adéquate au problème à résoudre et par conséquent celle qui permettra d'optimiser l'algorithme.

Nous attirons l'attention que ce support de cours est inspiré de plusieurs références bibliographiques (livres et ressources disponibles sur le web).

Pré-requis : Aucun

Bibliographie :

FABER. (2009). FABER F. INTRODUCTION A LA PROGRAMMATION EN ANSI-C, http://www.ltam.lu/cours-c/, Date de consultation : 28 Juin 2009 .

HARDOUIN. (2009). HARDOUIN L. *http://www.istia.univ-angers.fr/~hardouin/cours_c.pdf, Date de constation : 19 Mai 2009* .

La Page de l'algorithmique. (2008). La Page de l'algorithmique. *http://algor.chez.com/index.html,* .

LAPORTE. (2009). LAPORTE S. *http://btsig972.free.fr/cours/Algorithme/1e_annee/07.pdf, Date de consultation : 09 Mai 2009* .

REBAINE. (1, 2009). REBAINE D. *http://wwwens.uqac.ca/~rebaine/8INF805/courslistespilesetfiles.pdf, Date de consultation : 20 Mai 2009* .

REBAINE. (2,2009). REBAINE D. *http://wwwens.uqac.ca/~rebaine/8SIF109/Les%20arbres.pdf, Date de consultation : 10 Septembre 2009* .

Shaffer. (2001). Shaffer C A. *Practical Introduction to Data Structures and Algorithm Analysis (C++ Edition) (2nd Edition) (chapitre2)* .

TThomas et al. (2002). 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 dition, Paris 2002.*

Table des matières

Table des figures

Table des tableaux

Chapitre 1

Introduction à l'algorithmique

1.1. Démarche pour la résolution d'un problème

Pour résoudre un problème, en informatique, plusieurs étapes sont nécessaires :

1. Environnement : la connaissance précise de l'environnement (machine utilisée, processeur,...)
2. Problème : la définition précise du problème.
3. Analyse : l'analyse du problème et sa décomposition en sous-problèmes simples et distinctes.
4. Algorithme : la formulation de la solution sous une forme textuelle. Description des opérations à mettre en œuvre expliquant comment obtenir un résultat à partir de données. Il s'agit d'une description compréhensible par un être humain de la suite des opérations à effectuer pour résoudre le problème.

1.2. De l'algorithmique à la programmation

Pour automatiser la résolution d'un problème, nous devons traduire un algorithme en un programme. En effet, un programme ce n'est que la traduction d'un algorithme dans un langage de programmation compréhensible par la machine utilisée. Ainsi, pour obtenir un programme exécutable, nous devons passer par les étapes suivantes :

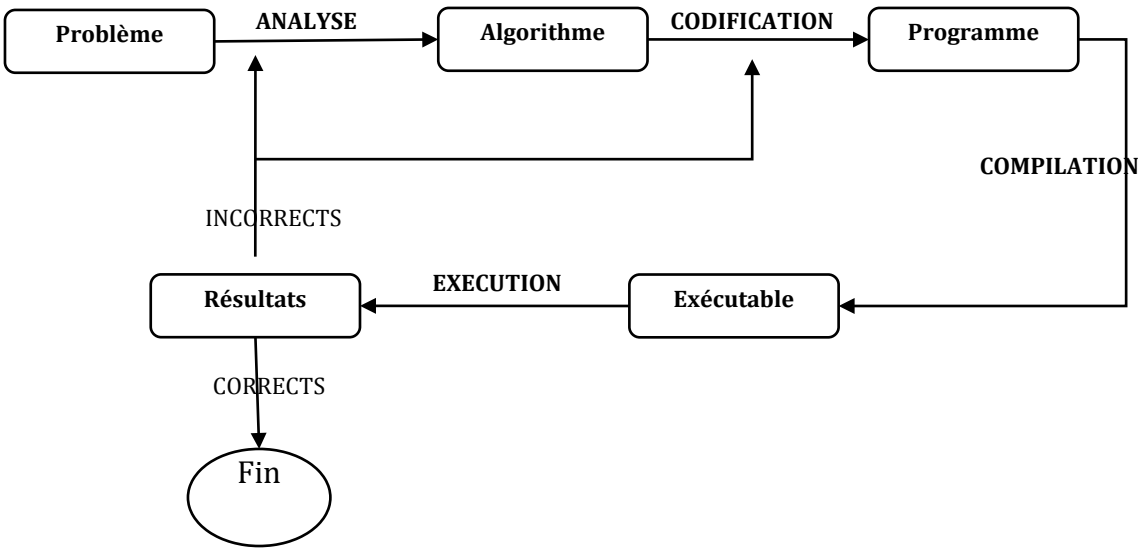

Figure 1.1-Démarche de résolution d'un problème

1.3. Définition d'un Algorithme

Un algorithme peut être définit comme suit :

« Un algorithme est une suite finie d'opérations ou règles à appliquer dans un ordre déterminé, à un nombre fini de données, pour arriver à en un nombre fini d'étapes, à un certain résultat et cela indépendamment des données (La Page de l'algorithmique, 2008)».

A noter que le mot algorithme vient du nom du mathématicien arabe El Khawarizmi, originaire de la ville de Khawarizmi (La Page de l'algorithmique, 2008)

1.4. Exemple

Pour déterminer le PGCD (Plus Grand Commun Diviseur) de deux entiers naturels a et b nous suivons la démarche suivante :

- Début
- Données de a et b
- si (a,b) premiers entre eux alors pgcd(a,b) = 1
  - sinon
    - décomposer a en facteurs premiers
    - décomposer b en facteurs premiers
    - pgcd(a,b) = produit des facteurs premiers communs à a et b, avec les exposants les moins forts.
- Fin

A partir de cette démarche, nous pourrons donner la structure générale d'un algorithme et nous détaillerons dans le chapitre suivant ses différents composants.

1.5. Structure d'un algorithme

Pour construire un algorithme, nous devons mettre en évidence les 3 parties suivantes :

Nom de l'algorithme	Algorithme Nom_algorithme
Déclarations	Déclarations de types Déclarations de variables Déclarations de constantes
	Début

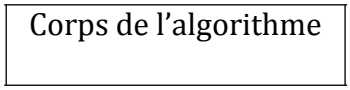

Figure 1.2- Les trois parties d'un algorithme

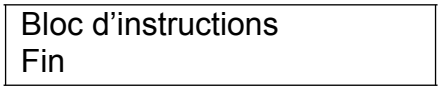

Figure 1.3- Structured'un algorithme

Où :

- **Nom de l'Algorithme** : c'est l'identification ou le nom.
- **Déclarations** : description de tous les objets utilisés dans l'algorithme, définition de variables, de constantes et de types.
- **Instructions**: séquence d'actions à exécuter sur l'environnement afin de résoudre le problème. Cette partie est également appelée le corps de l'algorithme.

1.6. Conclusion

Nous venons d'introduire la notion d'algorithme et les différentes étapes pour l'obtention d'un programme exécutable. Pour tout algorithme, nous devons définir une structure de données adéquate pour utiliser les objets traités qui seront détaillés dans le chapitre suivant.

Chapitre 2

Introduction aux Structures de Données

2.1. Introduction

Un ordinateur manipule des objets. Chaque objet a besoin de trois qualificatifs pour sa définition.

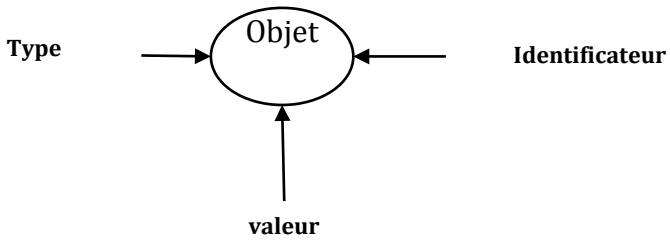

Figure 2.1- Les trois qualificatifs d'un objet

Un objet est caractérisé par son nom (identificateur), sa valeur, son type (figure 2.1). La valeur de l'objet peut être constante ou variable. Le type de l'objet peut être simple ou composite. Un type simple comme : entier, réel, caractère, etc. Un type composite peut être : tableau, enregistrement, etc. Toutes ces nouvelles notions seront détaillées dans les sections suivantes.

2.2. Constante et variable

Une constante est un objet dont la valeur ne varie pas tout au long du programme. Un objet dont la valeur change est nommé variable.

2.3. Identificateur

Un identificateur est dénomination d'un objet qui doit respecter les règles suivantes :

Un identificateur est soit :

Une suite alphanumérique qui ne doit pas commencer par un chiffre. En algorithmique et en programmation le symbole '_' est aussi considéré comme une lettre.

2.4. Type

Un type de donnée définit un ensemble dans lequel les variables prennent leur valeur. Chacune des variables d'un algorithme doit être associée à un type de donnée et un seul. A chaque type de donnée est associé un ensemble d'opérateurs qui sont possibles sur les variables et les constantes de ce type. Il y a cinq types élémentaires standards :

- Entier
- Réel
- Booléen
- Caractère
- Chaîne de caractère

2.4.1. Le type entier

Une variable de type entier prend ses valeurs dans l'ensemble des nombres entiers. Les opérateurs qui sont appliqués à des entiers, donnent un résultat entier sont :

- Addition (+)
- Soustraction (-)
- Multiplication (*)
- Division entière (DIV)
- Reste de la division entière (MOD).

2.4.2. Le type réel

Une variable de type réel fait partie des nombres réels.

2.4.2.1. Les opérateurs réels

Le résultat des opérateurs suivants est réel, si au moins l'un des opérandes est réel (l'autre pouvant être entier) :

- Addition (x+y)
- Soustraction (x-y)
- Multiplication (x*y)
- Division (x/y)

2.4.2.2. Les fonctions sur les réels

Les fonctions suivantes donnent un résultat réel, pour un argument entier ou réel :

sin(x), cos(x), exp(x), ln(x), sqrt(x), arctan(x), etc.

L'argument des fonctions suivantes est réel, et le résultat est entier :

- trunc(x) : partie entière de x,
- round(x) : entier le plus proche de x.

2.4.3. Le type booléen

Une variable booléenne ou de type logique doit prendre pour valeurs la constante TRUE (pour VRAI) ou la constante FALSE (pour FAUX). Ces valeurs pouvant être obtenues par l'évaluation d'une expression logique.

2.4.3.1. Les opérateurs booléens

Les opérateurs booléens (ou logiques) sont :

- Non (NOT: Négation)
- ET (AND logique)
- OU (OR logique)

a	NOT(a)
FAUX	VRAI
VRAI	FAUX

Tableau 2.1 – Table de vérité de l'opérateur NOT

a	b	a OR b	a AND b
FAUX	FAUX	FAUX	FAUX
FAUX	VRAI	VRAI	FAUX
VRAI	FAUX	VRAI	FAUX
VRAI	VRAI	VRAI	VRAI

Tableau 2.2 – Table de vérité de l'opérateur OR et de l'opérateur AND

2.4.3.2. Propriétés des opérateurs logiques

Quelques propriétés des opérateurs logiques sont importantes, notons les plus utilisées :

- a ET (b OU c) = (a ET b) OU (a ET c)
- a OU (b ET c) = (a OU b) ET (a OU c)
- NON(a ET b) = NON(a) OU NON(b)
- NON(a OU b) = NON(a) ET NON(b)

2.4.3.3. Les opérateurs de comparaison

Une expression logique contient un ou plusieurs opérateurs comparaison et/ou un ou plusieurs opérateurs logiques. Les opérateurs relationnels sont :

- = : égal à
- < > : différent de
- < : inférieur à
- > : supérieur à
- <= : inférieur ou égal à
- >= : supérieur ou égal à

La priorité des opérateurs logiques est : NON– ET – OU, cet ordre peut être modifié par l'utilisation de parenthèses.

2.4.4. Le type caractère

Le type caractère sont les lettres minuscules, lettres majuscules, chiffres et signes spéciaux. En règle générale, cet ensemble correspond à celui des caractères qui sont représentés sur le clavier d'un ordinateur.

Un caractère est représenté sur 8 bits, ce qui donne 256 combinaisons possibles. Le code, couramment utilisé, est le code ASCII (American Standard Code for Information Interchange).

Les opérateurs de comparaison définis précédemment s'appliquent également sur les caractères avec les conventions :

- 'A' < 'B' < … < 'Z'
- '0' < '1' < ... < '9'

Les fonctions suivantes sont utiles pour les opérations sur les caractères :

- SUCC : pour successeur
- PRED : pour prédécesseur
- ORD : pour rang
- CHR : la fonction CHR n'admet pour paramètre qu'un nombre entier. Elle fournit comme résultat le caractère dont le numéro d'ordre est spécifié comme paramètre.

Les fonctions CHR et ORD sont des fonctions inverses.

Exemples :

- ORD('A') = 65
- CHR(65) = 'A'
- SUCC('X') = 'Y'
- PRED("G") = 'F'

2.4.5. Le type chaîne de caractères

Le type chaîne de caractères est l'ensemble des séquences de caractères que l'on peut former. Une chaîne vide est une chaîne qui ne contient aucun caractère. Une chaîne peut contenir au maximum 256 caractères.

2.5. Les expressions

La formation des expressions est définie par récurrence : Les constantes et les variables sont des expressions. Les expressions peuvent être combinées entre elles par des opérateurs et former ainsi des expressions plus complexes (FABER, 2009)

Une expression arithmétique a pour valeur un nombre entier ou réel. Une expression booléenne a pour valeur le type de données booléen.

Pour composer un algorithme, nous aurons souvent à exprimer des expressions qui seront par la suite évalués au moment de l'exécution. Pour écrire une expression, nous écrirons très souvent une comparaison entre deux valeurs de même type. Une telle comparaison est appelée expression booléenne simple. Lorsque l'expression simple ne suffit pas pour exprimer une situation, nous utilisons des expressions composées à partir de d'expressions simples par l'emploi d'opérateurs logiques.

Exemples :

Si X et Y sont deux variables numériques de valeurs respectives 5 et 10, nous aurons :

Expression booléenne	valeur
x=y	Faux
x<y	Vrai
x>2	Vrai

Tableau 2.3 – Exemple d'évaluation d'expressions booléennes

Evaluez les deux expressions suivantes lorsque x, y et z ont pour valeurs respectives 1, 3 et 2 :

- (x = 1 OU y = 2) ET z > 3
- x = 1 OU (y = 2 ET z > 3)

Il est parfois nécessaire d'utiliser des parenthèses dans l'expression d'une expression composée. Les expressions entre parenthèses sont alors évaluées en premier.

2.6. Hiérarchie entre les opérateurs

Nous présentons dans le tableau ci-dessous la hiérarchie entre les différents opérateurs. En effet, c'est la priorité d'évaluation dans l'ordre décroissant (du plus prioritaire au moins prioritaire) :

Opérateur	Intitulé
()	les parenthèses
NON	la négation
/, *	la multiplication et la division
+, -	l'addition et la soustraction
=,>,<,<=,>=,<>	les opérateurs relationnels
ET	ET logique
OU	OU logique

Tableau 2.4 – Hiérarchie entre les opérateurs

Dans l'évaluation d'une expression sans parenthèses, les opérations sont effectuées par ordre de priorité. Les opérations de même priorité sont effectuées de la gauche vers la droite. Ainsi, pour effectuer B + C/D – E*F, nous calculons d'abord C/D puis E*F. Nous pouvons évidemment modifier les priorités en utilisant des parenthèses. Dans l'expression B + C/(D-E), nous calculerons d'abord D-E.

Chapitre 3

Instructions élémentaires et structure d'un algorithme

3.1. Structure d'un algorithme

Comme nous l'avons mentionné dans le chapitre 1, un algorithme est composé de trois parties :

Le nom de l'algorithme, la partie déclarative et le corps de l'algorithme. Dans ce qui suit, nous détaillons ces différentes composantes.

3.2. Déclaration de variables

- a, b, c : entier : a : entier
  B : entier
  C : entier
- x, y, z : réel
- n, v : booléen
- mot : chaîne de caractère
- c : caractère

3.3. Déclaration de constantes

- Pi = 3,14159
- ch. = 'calcul'
- C = 'R'
- test = TRUE

Dans la partie déclarative de l'algorithme, nous devons énumérer :

1. La liste des variables, tout en précisant leurs types respectifs.
2. La liste des constantes, en leur affectant leurs valeurs.
3. Les types qui ne sont pas standards.

3.4. L'affectation

Dans un environnement donné, pour affecter (attribuer) à une variable (V) une valeur (e), nous utilisons la notation suivante : V ← e

Où :

- V : le nom de la variable à laquelle on doit attribuer la valeur,
- ← : symbole, caractérisant l'affectation,
- e : représente la valeur à affecter et peut être :
  - Une constante.
  - Le nom d'une autre variable qui contient la valeur.
  - Une expression de même type que V qui peut être une expression logique ou une expression arithmétique.

Exemples :

a) X ← 1

b) NOM ← "mohamed"

c) TOTAL ← SOMME

d) NB ←A+B

- L'action (a) affecte à la variable numérique X la valeur 1.
- L'action (b) affecte à la variable nom la chaîne "mohamed"; pour que cette action soit correcte, il faut que NOM soit de type chaîne de caractères.
- dans ces 2 premières affectations, la valeur à affecter est indiquée à l'aide d'une constante.
- dans (c), la valeur à affecter est une variable. TOTAL et SOMME doivent être de même type.
- Dans (d), nous affectons à NB, le résultat d'un calcul numérique. NB, A et B doivent être de même type. L'action (d) s'exécute en deux temps :
  - calcul de la valeur de l'expression arithmétique (A+B)
  - affectation de cette valeur à la variable NB.

Dans une affectation, seule la variable dont le nom apparaît à gauche du signe ← change de valeur.

3.5. Fonctions d'entrées/sorties

La fonction d'entrée lire() permet d'affecter, à une variable, une valeur saisie au clavier. Cette fonction est validée par la frappe de la touche (entrée).

lire(v) : saisit une valeur au clavier et la mettre dans la variable v.

La fonction de sortie ecrire() permet d'afficher la valeur d'une constante, d'une variable ou d'une expression.

Exemples :

- ecrire ('Calcul de la somme des n premiers entiers')
- ecrire (i)
- ecrire ('Résultat du calcul = ', S+i)

3.6. Les commentaires

Pour qu'un algorithme ou un programme soit clair et compréhensible, il est conseillé de le commenter. Les commentaires sont des textes rédigés lors de l'élaboration de l'algorithme (ou programme). Le commentaire est ignoré par le processeur lors de l'exécution du programme. Généralement ce texte est inséré entre { et } ou bien (* et *) ou encore entre /* et*/.

3.7. Les instructions

Le corps d'un algorithme se compose d'instructions. Nous distinguons les instructions simples (une seule action) et les instructions composées (un bloc contenant des séquences d'instructions et comptant pour une seule instruction).

3.8. Exemple d'algorithme

Ecrire un algorithme intitulé « calculs » qui permet de saisir trois réels quelconques, de calculer ensuite leur somme, leur produit et leur moyenne et d'afficher les résultats obtenus.

Méthode (ou analyse) : nous distinguerons trois étapes :

1. Obtention des trois nombres : il nous faudra trois objets de type réel pour recevoir ces nombres.
2. Calcul des différents résultats : il nous faudra déclarer trois objets de type réel qui reçoivent le résultat de chacune des opérations : somme, produit et moyenne.
3. affichage des résultats.

Solution :

```
Algorithme calculs
        Variable
        Somme, Produit, Moyenne, nb1, nb2, nb3 : réel
Début
```

```
        (* saisie des trois nombres *)
        ecrire ("entrez les trois nombres")
        lire (nb1, nb2, nb3)
        (* réalisation des différentes opérations *)
        Somme ← nb1 + nb2 + nb3
        Produit ← nb1 * nb2 * nb3
        Moyenne ← somme / 3
        (* édition des résultats *)
        ecrire ("la somme des trois nombres est : ", Somme)
        ecrire ("le produit des trois nombres est : ", Produit)
        ecrire ("la moyenne des trois nombres est : ", Moyenne)
Fin
```

Remarque :

Nous pouvons afficher directement le résultat des opérations sans passer par les variables Somme, Produit et Moyenne. Ce qui donne une seconde version de l'algorithme (calculs_v2) plus compacte et plus optimisée :

```
Algorithme calculs_v2
        Variable
        nb1, nb2, nb3 : réel
Début
        (* saisie des trois nombres *)
        ecrire ("entrez les trois nombres")
        lire (nb1, nb2, nb3)
        (* réalisation des différentes opérations et édition des résultats *)
        ecrire ("la somme des trois nombres est : ", nb1+nb2+nb3)
        ecrire ("le produit des trois nombres est : ", nb1 * nb2 * nb3)
        ecrire ("la moyenne des trois nombres est : ", (nb1 + nb2 + nb3) / 3)
Fin
```

Chapitre 4

Les instructions conditionnelles

4.1.1. L'instruction conditionnelle simple

Une instruction conditionnelle simple à la syntaxe suivante :

```
si condition alors
        bloc d'instructions
finsi
```

Où :

- condition : expression logique ayant pour valeur Vrai ou Faux.

Si la condition est vraie, le bloc d'instructions est effectué, si elle est fausse on ne fait rien.

4.1.2. Structures alternatives complètes

Une instruction conditionnelle d'alternative à la forme (ou la syntaxe) suivante :

```
si condition alors
        bloc d'instructions1
sinon
        bloc d'instructions2
finsi
```

Si la condition est vraie, le bloc d'instructions 1 est effectué, si elle est fausse c'est le bloc d'instructions 2 qui est effectué.

Notons que les deux blocs d'instructions peuvent comporter à leur tour des structures conditionnelles. Nous disons qu'il s'agit d'imbrication de structures conditionnelles (voir l'exemple suivant).

Remarque :

Les deux règles suivantes doivent être toujours respectées dans un schéma conditionnel :

1. A chaque si doit correspondre un finsi
2. Un finsi revient au dernier si qui n'a pas de finsi

Exemple :

Ecrire un algorithme qui permet de résoudre une équation de second degré.

Solution :

```
Algorithme Exemple1
        Variable
        a, b, c : réel (* les coefficients de l'équation *)
        x1, x2 : réel (* les racines si elles existent *)
        d : réel (* le discriminant *)
Début
        ecrire(''entrez les 3 coefficients : '')
        lire(a,b,c)
        si a <> 0 alors
                d ← -b – 4*a*c
                si d >= 0 alors
                        x1 ←(-b + sqrt(d) ) / (2*a)
                        x2 ←(-b - sqrt(d) ) / (2*a)
                        ecrire(''deux solutions réelles : x1 = '', x1, ''et x2 = '', x2)
                sinon
                        ecrire(''pas de solutions réelles'')
                finsi
        sinon
                si b <> 0 alors
                        x1 ← -c/b
                        ecrire(''une solution : '', x1)
                sinon
                        si c <> 0 alors
                                ecrire(''équation impossible'')
                        sinon
                                ecrire(''tout réel est solution'')
                        finsi
                finsi
        finsi
Fin
```

2) Ecrire un algorithme qui permet d'afficher à l'écran le classement, dans l'ordre croissant, de deux réels quelconques, saisis au clavier par l'utilisateur. Indication : les deux nombres sont lus dans les variables Inf et Sup. Ensuite, si Inf n'est pas la plus petite valeur, on échange les deux valeurs pour avoir Inf < Sup.

```
Algorithme Exemple2
        Variable
        Inf, Sup : réel (* les 2 nombres à classer *)
        aux : réel (* une variable auxiliaire *)
Début
        ecrire(''entrez un premier nombre : '')
        lire(Inf)
        ecrire(''entrez un deuxième nombre : '')
        lire(Sup)
        si (Inf > Sup alors) (* échange des 2 valeurs *)
                aux ←Inf
                Inf ←Sup
                Sup ← aux
        finsi
```

écrire(''Le classement est : '', Inf, ''et'', Sup)
Fin

4.1.3. Structure conditionnelle à choix multiple

Une généralisation supplémentaire de la structure alternative est la structure à choix multiple. Cette dernière est utilisée quand nous avons à effectuer plusieurs choix possibles.

La structure conditionnelle à choix multiple à la forme suivante :

```
selon (variable) faire
        valeur 1 : bloc d'instructions 1
        valeur 2 : bloc d'instructions 2
        .....
        valeur n, valeur n+1 : bloc d'instructions n
        .....
        autre : bloc d'instructions 0
finselon
```

Si variable est égale à la valeur i, nous exécutons le bloc d'instruction i, et nous passons à la suite de l'algorithme, sinon on exécute le bloc d'instructions 0 et nous passons à la suite de l'algorithme.

Exemple :

Ecrire un algorithme permettant d'afficher le jour d'inscription d'un étudiant suivant le niveau d'étude saisit au clavier :

- 1ère année →03/09/2004
- 2ème année → 04/09/2004
- 3ème année →05/09/2004
- 4ème et 5ème année →06/09/2004

Chapitre 5

Les structures itératives

Nous appelons itération toute répétition de l'exécution d'une action ou d'une séquence d'actions. Nous distinguons trois types de boucles.

5.1. La boucle Pour

La boucle « POUR » est une boucle fixe, c'est à dire, nous connaissons d'avance le nombre d'itérations que nous allons effectuer à l'aide de cette boucle. Sa syntaxe est la suivante :

Pour v **de** vinitiale **à** vfinale [**pas de** p] **faire**
Bloc d'instructions
finpour

Où :

- v : variable de contrôle, ou compteur,
- p : valeur de variation de v à chaque itération, càd (v←v+p). si le pas est non défini sa valeur par défaut est égale à 1.

Cette structure permet de répéter le bloc d'instructions un nombre de fois défini à l'avance.

Exemple 1:

Ecrire un algorithme permettant de calculer la somme des dix premier chiffres.

Solution :

```
Algorithme Exemple1
        Variable
         i , S: entier
Début
        S←0
        pour i de 0 à 9 faire
                S←S+i
        finpour
        ecrire (" La somme = ",S)
Fin
```

Exemple 2:

Ecrire un algorithme permettant d'afficher les nombres pairs dans les 100 premiers entiers naturels.

Solution :

```
Algorithme Exemple2
        Variable
         i : entier
Début
        pour i de 0 à 100 pas de 2 faire
                ecrire(i, " est pair")
        finpour
Fin
```

5.2. La boucle Répéter

Contrairement à la boucle « Pour », le nombre d'itérations dans la boucle « Répéter » n'est pas prédéterminé. Sa syntaxe est la suivante :

Répéter
Bloc d'instructions
Jusqu'à (Cd**)**

Où :

- Bloc d'instructions : action ou séquence d'actions à répéter.
- Cd : expression logique, servant de contrôle de l'itération.

Cette structure permet de répéter A jusqu'à ce que la condition Cd soit vraie. Cd est appelé condition d'arrêt de la boucle. La condition n'est testée qu'après une première exécution du Bloc d'instructions. Le bloc d'instructions est donc exécuté au moins une fois.

Exemple :

```
lire (a)
répéter
        b ←a DIV 3
        a ←a DIV 2
jusqu'à b < 2
écrire (a, b)
```

Soit a = 18; le tableau ci-dessous reprend les différentes valeurs de b, a, et b<2 :

a	b	b<2
18	-	-
9	6	Faux
4	3	Faux
2	1	**Vrai**

Et la boucle se termine. Donc, a = 2 et b = 1.

5.3. La boucle Tant que

La syntaxe de la boucle « tant que » est la suivante :

Tantque (Cd**) faire**
Bloc d'instructions
Fintantque

Où :

- Bloc d'instructions : action ou séquence d'actions à répéter.
- Cd : expression logique, servant de contrôle de l'itération.

Cette structure permet de répéter le bloc d'instructions tant que la condition Cd est satisfaite. La condition Cd est testée avant l'exécution du bloc d'instructions.

Exemple :

lire (a)
b←5
Tantque (b>=2) faire
b ←a DIV 3
a ←a DIV 2
Fintanque
ecrire (a, b)

Soit a = 18; le tableau ci-dessous reprend les différentes valeurs de b, a, et b<2 :

a	b	b>=2
18	5	Vrai
9	6	Vrai
4	3	Vrai
2	1	**Faux**

Et la boucle se termine. Donc, a = 2 et b = 1.

Exercices :

Ecrire un algorithme permettant de lire un entier strictement inférieur à 100.

Solution :

```
Algorithme Exercice1
        Variable
         n : entier
Début
        Répéter
                lire(n)
        Jusqu'à (n>0 et n<=100)
Fin
```

Ecrire un algorithme permettant de calculer la somme avec une précision epsilon =:

Solution :

```
Algorithme Exercice2
Variable
         i : entier
        S : réel
        e : réel
Début
        S←0
        Tantque(e>) faire
                S←S+
        Fintanque
Fin
```

Chapitre 6

Les tableaux

Dans ce chapitre, nous allons introduire une nouvelle structure de donnée complexe qui permet le stockage de plus d'une valeur de même type. Nous distinguons deux types de tableaux. Tableaux simples (unidimensionnels) ou appelés aussi vecteurs en mathématique et une structure de tableaux à deux dimensions (bidimensionnels) ou matrice.

6.1. Tableaux simples

6.1.1. Définition

Un tableau à une dimension, appelé aussi vecteur, est une structure de données formée de données de même type pouvant être accédé avec un indice. Il nous permet de manipuler plusieurs valeurs en utilisant un seul nom de variable.

Pour définir un tableau, il faut préciser :

- Son identificateur,
- Sa taille (nombre de cases)
- Le type de ses éléments (composants).

6.1.2. Notations et déclarations

Un tableau T est déclaré de la façon suivante :

nom **: tableau [**min**..**max**] de** type_ éléments

Avec :

- nom : nom du tableau.
- type_ éléments : le type des éléments du tableau.

Nous prendrons souvent min = 1 et max désignera alors le nombre des éléments du tableau, que nous appelons également longueur ou taille du tableau. Nous notons que le premier élément est d'indice 1. Ici il s'agit d'une convention et dépend des langages de programmation. Par exemple en C, le premier élément est d'indice 0.

Exemple :

Moyenne : tableau [1..30] de réel

Absence : tableau [1..30] d'entier

Dans un tableau :

- Tous les composants sont de même type.
- Les indices des éléments d'un tableau peuvent varier dans n'importe quel intervalle d'entiers naturels.
- Le type des indices est entier.
- Le nombre des composants est défini à la déclaration du tableau. Il est donc statique.

6.1.3. Accès à un élément du tableau

D'une façon générale, T[i] identifie l'élément de position i dans le tableau T. Cela traduit l'accès direct aux éléments du tableau. Il faut que : min< i <max.

Exemple :

Moyenne[13] ← 13.40

Absence[27] ← 4

6.1.4. Opérations sur les tableaux

6.1.4.1. Affectation

T1←T2 ⇔ pour i de min à max faire

T1[i]←T2[i]

Finpour

A condition que les deux tableaux sont de même type et de même taille. Autrement dit, l'affectation se traduit par la recopie d'un tableau dans un autre.

6.1.4.2. Comparaison

T1 =T2 est vrai si et seulement si pour tout i ∈[min..max] nous avons T1[i]=T2[i]

T1 ≠T2 est vrai si et seulement si il existe i ∈[min..max] tel que T1[i]=T2[i]

6.2. Lecture d'un tableau :

Pour lire les éléments d'un tableau T, de taille n, nous procédons de la façon suivante :

```
Début
    ……..
    lire(n) /* lire la taille du tableau */
    pour i de 1 à n faire /* lire les éléments du tableau */
            lire ( T[i] )
    finpour
    ……..
```

Fin

6.3. Ecriture d'un tableau :

Pour écrire (c'est à dire afficher) les éléments d'un tableau T, de taille n, nous procédons de la façon suivante :

```
Début
        ……..
        lire(n) /* lire la taille du tableau */
        ……..
        pour i de 1 à n faire /* afficher les éléments du tableau */
                ecrire ( T[i] )
        finpour
        ……..
Fin
```

6.4. Tableaux à deux dimensions

6.4.1. Définition

Si un traitement utilise plusieurs tableaux à une dimension, subissant le même traitement, on utilise souvent un seul tableau à deux dimensions (ou matrice). Chaque élément du tableau est alors identifié par deux indices : l'un désignant la ligne et l'autre la colonne.

6.4.2. Déclaration

Un tableau T à deux dimensions est déclaré de la façon suivante :

Nom **: tableau [**min1**..**max1, min2**..**max2**] de** type_éléments

Avec :

- max1 – min1 + 1 = nombre d'éléments d'une ligne.
- max2 – min2 + 1 = nombre d'éléments d'une colonne.

6.4.3. Accès à un élément du tableau à deux dimensions

T[L,C] identifie l'élément situé sur la ligne L et la colonne C du tableau T.

Exemple

tab : tableau [1..2 , 1..5] de caractère

a	c	?	f	1
T	&		?	n

Le caractère '&' est situé à ligne 2 et à la colonne 3, nous notons : tab[2, 3] = '&'.

Exercice

Ecrire un algorithme qui permet de

- Lire le nombre de lignes (l) et le nombre de colonnes (c) d'un tableau à deux dimensions A
- Lire les éléments de A.
- Calculer et afficher tous les éléments de A

Solution :

```
Algorithme Matrice
Variable
        A : tableau[1..300,1..300] de réél
        S : réel
        l,c,i,j : entier
        e : réel
Début
        ecrire("Donnez l ")
        lire(l)
        ecrire("Donnez c ")
        lire(c)
        pour i de 1 à l faire
                pour j de 1 à c faire
                        lire(A[i,j])
                Finpour
        Finpour

        pour i de 1 à l faire
                pour j de 1 à c faire
                        S←A[i,j]
                Finpour
        Finpour
        ecrire (s)
Fin
```

6.5. Définition d'un nouveau type

On peut définir un type en utilisant la syntaxe suivante :

Type Nouveau_Type =Ancien_Type

Par la suite Nouveau_Type est utilisé pour déclaré les variables.

Exemple

Type vecteur = tableau[1...1000] d'entiers

Type matrice = tableau[1...1000, 1...4000] d'entiers

Chapitre 7

Les sous-programmes : les procédures et les fonctions

7.1. Les sous-programmes

Pour être lisible et efficace, un algorithme ne doit pas être excessivement long. D'autre part, nous rappelons qu'une action au sein d'un algorithme peut-être elle-même un algorithme. Sa description doit être faite en dehors des limites de l'algorithme qui l'utilise, en respectant les règles syntaxiques habituelles.

D'une façon générale, la réalisation d'un algorithme peut amener à distinguer, au moment de la description du traitement, différentes parties dont l'assemblage constituera l'algorithme principal, comme le montre le schéma ci-dessous :

Algorithme **Principal**	Algorithme **A**	Algorithme **B**	Algorithme **C**	Algorithme **D**
Début appel de A appel de B appel de C Fin	Début …….. …….. …….. Fin	Début …….. appel de A …….. Fin	Début …….. appel de B …….. Fin	Début …….. appel de B appel de C …….. Fin

Nous pouvons décrire un algorithme sous une forme modulaire, afin de structurer son architecture d'une manière claire. On peut également vouloir définir un sous-programme (ou un module) lorsque le même type de traitement doit être répété plusieurs fois, à différents endroits de l'algorithme.

Le module (ou le sous-programme), est décrit qu'une seule fois et nous effectuerons un appel pour chaque demande d'exécution dans l'algorithme principal.

Nous parlerons alors d'algorithme appelant et d'algorithme appelé. L'appelant est celui qui contient l'appel à un module défini séparément, qui est lui même l'appelé.

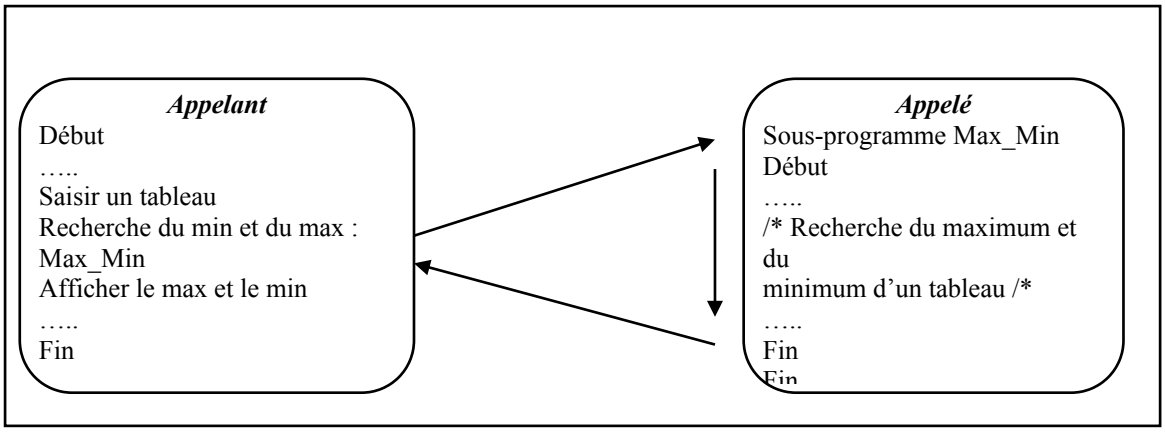

Figure 7.1Algorithme appelant et d'algorithme appelé

Les variables déclarées dans l'algorithme appelant sont appelées des variables globales. L'appelé peut déclarer ses propres variables qu'il est le seul à utiliser. Dans ce cas, nous parlerons de variables locales.

Il y a deux parties dans la définition d'un module :

- La spécification (QUOI ou nom du sous-programme).
- La réalisation (COMMENT ou regroupement d'actions).

Syntaxiquement, la partie spécification constitue l'en-tête du sous-programme et la partie réalisation constitue le corps du sous-programme.

7.2. Les procédures

7.2.1. Définition

Un module auquel sont associés un nom et une liste de paramètres formels est appelé action paramétrée ou procédure. Pour utiliser cette procédure, il faut lui fournir une liste de paramètres réels (ou effectifs).

7.2.2. Déclaration d'une procédure

Une procédure est déclarée de la façon suivante :

```
PROCEDURE nom_procedure (liste des paramètres)
(* Les déclarations *)

Début
(* Les instructions de la procédure *)

Fin
```

7.2.3. Les paramètres formels et effectifs

1. Un **paramètre formel** est une variable choisie comme paramètre à la définition d'un algorithme (ou d'une procédure).

2. Un **paramètre effectif** (ou réel) est une valeur utilisée dans un appel d'une procédure à la place d'un paramètre formel.
3. A l'exécution d'un appel, la correspondance entre un paramètre effectif et un paramètre formel est définie par la position : le premier paramètre effectif correspond au premier paramètre formel, le deuxième au deuxième, etc.
4. Chaque appel doit comprendre autant de paramètres effectifs que la définition de la procédure comporte de paramètres formels.
5. Chaque paramètre effectif doit être du même type que le paramètre formel correspondant.
6. Lors de l'exécution d'un appel, l'algorithme appelant travaille directement sur les paramètres effectifs à la place des paramètres formels.

Exemple

Ecrire un algorithme « classer », permettant d'effectuer le classement, dans l'ordre croissant de 3 nombres réels quelconques n1, n2 et n3 saisis au clavier par l'utilisateur. Cet algorithme doit effectuer des classements 2 par 2 des 3 nombres, pour déduire le classement souhaité.

Indications et plan de l'algorithme :

- Lecture des 3 nombres (n1, n2, n3).
- Classement de n1 et de n2 (n1 < n2, par exemple).
- Classement de n2 et de n3 (n2 < n3 et n1 < n3 : mais il se peut qu'on ait n1 > n3, dans le cas où n3 est le plus petit des trois réels ; ce qui exige le reclassement de n1 et de n2).
- Classement de n1 et de n2 (n1 < n2 < n3).
- Affichage des résultats.

Nous constatons que l'action de classement de 2 nombres sera répétée 3 fois dans l'algorithme demandé. Pour cela, il est évident que l'écriture d'un module indépendant, permettant d'effectuer le classement de 2 nombres et l'appel de ce module dans un programme principal serait bien plus efficace. Ce module est la procédure Classer_2réels ci-dessous :

```
PROCEDURE Classer_2réels (x, y : réel)
        Variables
        aux : réel
Début
        Si (x > y )alors (* échange des 2 variables par l'intermédiaire de aux *)
                aux ← x
                x← y
                y ← aux
        Finsi
Fin
```

Par la notation Classer_2réels(x, y), nous indiquons que les objets x et y ne sont pas des variables, mais des paramètres qui représentent des objets ou valeurs sur lesquels

nous pouvons souhaiter faire travailler l'algorithme classer_2réels. L'algorithme effectuant le classement de 3 nombres s'écrira alors, en utilisant classer_2réels comme suit :

```
Algorithme Classer_3réels
        variable
        n1, n2, n3 : réel
Début
        ecrire (''Saisir les 3 réels à classer'')
        lire(n1, n2, n3)
        Classer_2réels(n1, n2)
        Classer_2réels(n2, n3)
        Classer_2réels(n1, n2)
        ecrire (''Le classement est : '', n1 , n2 , n3)
Fin
```

Dans l'algorithme principal, l'écriture Classer_2réels(n1, n2) est un appel de la procédure Classer_2réels par lequel on indique au processeur d'exécuter cet appel en faisant jouant à n1, le rôle de x, et à n2, le rôle de y.

Les variables x et y sont les paramètres formels de la procédure Classer_2réels : ils servent de modèle pour désigner les objets qui leurs seront substitués lors d'un appel de Classer_2réels.

Les variables n1 et n2 sont les paramètres effectifs d'un appel. Ce sont des variables sur lesquelles s'exécute un appel qui emploi ces objets en lieu et place des paramètres formels.

7.3. Les fonctions

7.3.1. Définition

Une fonction est une procédure particulière. En effet, une fonction présente la particularité de générer (retourner) une et une seule valeur d'un type déterminé, appelée résultat. Ce résultat peut être exploité directement dans une instruction au sein de l'algorithme principal.

7.3.2. Déclaration d'une fonction

Une fonction est déclarée de la façon suivante :

```
FONCTION nom_fonction (liste des paramètres) : Type de résultat
(* Les déclarations *)

Début
(* Les instructions de la fonction *)
nom_fonction←…
Fin
```

- Le type de la valeur que retourne une fonction est déclaré dans la première ligne de la fonction, à la suite de la liste des paramètres formels.
- La valeur que retourne la fonction (le résultat) est déterminée par une ou plusieurs instructions retour (affectation d'une valeur au nom de la fonction).

7.3.3. Appel d'une fonction

L'appel d'une fonction est contenu dans une instruction :

- variable ← nom_fonction (< paramètres effectifs >)
- si (nom_fonction (< paramètres effectifs >) > valeur)alors

  Action

  Finsi
- tantque (nom_fonction (< paramètres effectifs >) = valeur) faire

  < action >

  fintantque

Lors de l'utilisation d'une fonction, le système vérifie la comptabilité des types de variables traitées, ainsi que le type du résultat retourné par la fonction.

Exemples

Ecrire une fonction, qui permet de déterminer si une personne est majeure à partir de son âge. On suppose que l'âge de majorité est 18 ans.

Solution :

```
FONCTION majeur (âge : entier) : booléen
Début
        si âge >= 18 alors
                majeur ← vrai
        sinon
                majeur ←faux
        finsi
Fin
```

Une autre version, beaucoup plus simple, et tout aussi correcte, strictement équivalente à la précédente serait :

```
FONCTION majeur_2 (âge : entier) : booléen
début
        majeur_2 ← (âge >= 18)
fin
```

Ecrire une fonction qui retourne vrai, si et seulement si les 3 entiers a, b et c (qui sont les paramètres de cette fonction) peuvent être les mesures des côtés d'un triangle rectangle.

```
FONCTION triangle_rectangle (a, b, c : entier) : booléen
début
```

triangle_rectangle ←((a*a = b*b + c*c) OU (b*b = a*a + c*c) OU (c*c = a*a + b*b))
fin

7.4. Les variables globales et locales

7.4.1. Variables Globales

Une variable globale est une variable déclarée en entête de l'algorithme principal. Dans ce cas, elle est accessible dans tout l'algorithme, y compris les procédures et les fonctions.

7.4.2. Variables locales

Par contre, une variable locale est une variable déclarée à l'intérieure d'une procédure ou d'une fonction. Dans ce cas, elle ne peut être utilisée (ou visible) que par la fonction ou la procédure où elle a été déclarée.

7.5. Les paramètres d'appel

La définition d'une procédure peut se faire avec ou sans paramètres d'appel.

7.5.1. Procédure sans paramètres d'appel

Certaines procédures sans paramètres d'appel effectuent un certain travail indépendamment du reste du programme, c'est à dire sans aucun échange d'informations. C'est le cas pour beaucoup de procédures qui servent à l'affichage de message, ou de menus.

7.5.2. Procédure avec paramètres d'appel

Une procédure a souvent besoin d'avoir des informations pour exécuter sa tâche. Lors de la définition de la procédure, il est possible de choisir entre deux modes de passage (ou la transmission) des paramètres : le passage par valeur et le passage par adresse.

7.5.2.1. Passage par valeur

Lors de l'appel d'une procédure, un emplacement dans la PILE est réservé pour chaque paramètre formel. De même, un emplacement mémoire (SEGMENT DE DONNEES) est également réservé pour chaque paramètre réel lors de la déclaration. Pendant l'appel de la procédure, les valeurs des paramètres réels sont copiées dans les paramètres formels. Ainsi, l'exécution des instructions de la procédure se fait avec les valeurs des paramètres formels. Ainsi toute modification des paramètres formels ne peut affecter en aucun cas celles des paramètres réels.

Les valeurs des paramètres réels sont connues avant le début de l'exécution de la procédure et jouent le rôle uniquement d'entrées de la procédure.

Pour spécifier, dans une procédure, qu'il s'agit du mode « passage par valeur », il suffit d'écrire simplement les noms des paramètres formels.

Exemple :

```
Algorithme passage_par_valeur
        variable
        r : entier
PROCEDURE ajouter ( k : entier ) (* k : paramètre formel *)
Début
        écrire(''valeur de k = '', k)
        k ← k+1
        écrire(''résultat dans la procédure = '', k )
Fin
Début (* programme principal *)
        r ← 20 (* initialisation de r *)
        ajouter ( r ) (* appel de la procédure "ajouter" par passage de la valeur 20 *)
        écrire(''résultat dans l'algorithme = '', r )
Fin
```

Cet algorithme affichera à l'écran les résultats suivants :

```
valeur de k =20
 résultat dans la procédure =21
résultat dans l'algorithme =20
```

7.5.2.2. Passage par adresse (ou variable)

La différence principale entre le passage de paramètres par valeur et le passage de paramètres par adresse est que pour ce dernier mode un emplacement mémoire est réservé pour le paramètre formel et le paramètre réel correspondant. Dans ce cas, le paramètre formel utilise directement l'emplacement mémoire du paramètre réel. Par conséquent, toute modification du paramètre formel entraîne la même modification du paramètre réel correspondant.

Pour spécifier, dans une procédure, qu'il s'agit du mode « passage par adresse », il suffit de mettre devant le nom du paramètre formel la mention « variable ».

Exemple :

```
Algorithme passage_par_variable
        variable
        r : entier
PROCEDURE ajouter (variable  k : entier ) (* k : paramètre formel *)
Début
        écrire(''valeur de k = '', k)
        k ← k+1
        écrire(''résultat dans la procédure = '', k )
Fin
Début (* programme principal *)
        r ← 20 (* initialisation de r *)
        ajouter ( r ) (* appel de la procédure "ajouter" par passage de la valeur 20 *)
```

écrire(''résultat dans l'algorithme = '', r)
Fin

Cet algorithme afficherait à l'écran les résultats suivants :

valeur de k =20
résultat dans la procédure =21
résultat dans l'algorithme =21

La variable globale r est modifiée au même titre que le paramètre formel k.

Remarques :

1. Une fonction peut retourner une seule valeur à l'algorithme principal, alors qu'une procédure ne retourne rien au programme principal. Par contre, une procédure avec passage d'adresse peut retourner aucune ou plusieurs valeurs.
2. En algorithmique, toute fonction peut être alors convertie en procédure, mais l'inverse n'est pas vrai.

Chapitre 8

Les enregistrements

8.1. Introduction

Contrairement aux tableaux qui sont des structures de données dont tous les éléments sont de même type, les enregistrements sont des structures de données dont les éléments peuvent être de types différents et qui se rapportent à la même entité

Les éléments qui composent un enregistrement sont appelés champs.

Avant de déclarer une variable enregistrement, il faut avoir au préalable défini son type, c'est à dire le nom et le type des champs qui le compose. Le type d'un enregistrement est appelé type structuré.

8.2. Déclaration d'un type structuré

Jusqu'à présent, nous n'avons utilisé que des types primitifs (caractères, entiers, réels, chaînes) et des tableaux de types primitifs. Mais nous pouvons créer nos propres types puis déclarer des variables ou des tableaux d'éléments de ce type.

Pour créer des enregistrements, il faut déclarer un nouveau type, basé sur d'autres types existants, qu'on appelle type structuré Après avoir défini un type structuré, on peut l'utiliser comme un type normal en déclarant une ou plusieurs variables de ce type. Les variables de type structuré sont appelées enregistrements.

La déclaration des types structurés se fait dans une section spéciale des algorithmes appelée Type, qui précède la section des variables (et succède à la section des constantes).

Syntaxe (notation inspirée du Pascal)

```
Type
    nom_type= enregistrement
        nom_champ1: type_champ1
        …
        nom_champn: type_champn
    finenreg
```

Exemple

```
Type
    tpersonne = enregistrement
```

nom : chaîne
prénom : chaine
âge : entier
finenreg

8.3. Déclaration des variables à partir d'un type structuré

Une fois qu'on a défini un type structuré, on peut déclarer des variables enregistrements exactement de la même façon que l'on déclare des variables d'un type primitif.

Syntaxe

nom_var : nom_type

Exemple:

pers1, pers2, pers3 : tpersonne

8.4. Manipulation d'un enregistrement

La manipulation d'un enregistrement se fait au travers de ses champs. Comme pour les tableaux, il n'est pas possible de manipuler un enregistrement globalement, sauf pour affecter un enregistrement à un autre de même type. Par exemple, pour afficher un enregistrement il faut afficher tous ses champs uns par uns.

8.4.1. Accès aux champs d'un enregistrement

Alors que les éléments d'un tableau sont accessibles au travers de leur indice, les champs d'un enregistrement sont accessibles à travers leur nom, grâce à l'opérateur '.'

nom_enregistrement.nom_champ

Représente la valeur mémorisée dans le champ de l'enregistrement

Par exemple, pour accéder à l'âge de la variable pers2, on utilise l'expression: pers2.âge

Remarques :

1. le nom d'un champ est toujours précédé du nom de l'enregistrement auquel il appartient. On ne peut pas trouver un nom de champ tout seul, sans indication de l'enregistrement.
2. Les champs d'un enregistrement, tout comme les éléments d'un tableau, sont des variables à qui on peut faire subir les mêmes opérations (affectation, saisie, affichage,…).

Exemple :

Ecrire un algorithme qui saisie des données concernant les personnes pers1 et pers2, puis affiche la différence d'âge entre ces deux personnes

```
Algorithme Exemple
Type
        tpersonne =enregistremen
                nom : chaîne
                prénom : chaîne
                âge : entier
        finenregistrement
        Variable
        pers1, pers2 : tpersonne
Début
        ecrire ("Entrez le nom puis l'age de la personne 1")
        lire( pers1.nom, pers1.age)
         ecrire ("Entrez le nom puis l'âge de la personne 2")
        lire (pers2.nom, pers2.age)
        ecrire( "La différence d'âge entre ", pers1.nom, " et ", pers2.nom, " est de ")
        Si pers1.age > pers2.age Alors
                ecrire( pers1.age – pers2.age, " ans ")
        Sinon
                ecrire ( pers2.age – pers1.age, " ans ")
        FinSi
Fin
```

8.4.2. Passage d'un enregistrement en paramètre d'une fonction ou d'une procédure

Il est possible de passer tout un enregistrement en paramètre d'une fonction ou d'une procédure (on n'est pas obligé de passer tous les champs un à un, ce qui permet de diminuer le nombre de paramètres à passer), exactement comme pour les tableaux.

Exemple 1 :

Ecrire une fonction qui renvoie la différence d'âge entre deux personnes

```
Fonction différence (p1 : tpersonne, p2 : tpersonne) : entier
Début
        Si pers1.age > pers2.age Alors
                différence ←( pers1.age – pers2.age )
        Sinon
                différence ←( pers2.age – pers1.age )
        FinSi
Fin
```

Exemple 2 :

Ecrire une procédure qui permet de modifier l'âge d'une personne passée en paramètre..

```
Procédure majage (var : p : tpersonne, nouveauage : entier)
Début
        p.age← nouveauage
FinProc
```

8.4.3. L'imbrication d'enregistrements

Supposons que dans le type personne, nous ne voulions plus l'âge de la personne, mais sa date de naissance. Une date est composée de trois variables (jour, mois, année) indissociables. Une date correspond donc à une entité du monde réel qu'on doit représenter par un type enregistrement à 3 champs.

Si on déclare le type date au préalable, on peut l'utiliser dans la déclaration du type personne pour le type de la date de naissance.

```
TYPE
    date =enregistrement
            jour: entier
            mois: chaîne
            année: entier
    finenregistrement

    tpersonne =enregistrement
            nom : chaîne
            prénom : chaîne
            ddn : date
    finenregistrement
```

Pour accéder à l'année de naissance d'une personne, il faut utiliser deux fois l'opérateur '.'

```
pers1.ddn.année
```

Il faut lire une telle variable de droite à gauche : l'année de la date de naissance de la pers1.

Exemple :

Une adresse est composée d'un numéro, d'une rue, d'un code postal et d'une ville.

Un fournisseur est caractérisé par son code, sa raison social, son numéro de téléphone et de son adresse.

Un produit est décrit par son identifiant, son prix d'achat (Pa), son prix de vente (Pv), sa taxe et le code de son fournisseur.

Un produit est livré par un seul fournisseur.

```
Type
    adresse = enregistrement
            num : entier
            rue: chaîne
```

```
        cp: chaîne
        ville: chaîne
finenregistrement

fournisseur = enregistrement
code_frs : chaine
raison_sociale: chaine
ad_frs: adresse
tel: chaine
finenregistrement

Produit= enregistrement
        code: chaîne
        lib: chaîne
        paht: réel
        pvht: réel
        txtva: réel
        frs: fournisseur
finenregistrement
```

p.frs.tel : permet d'accéder au numéro de téléphone du fournisseur du produit p

8.5. Les tableaux d'enregistrement

Il arrive souvent que l'on veuille traiter non pas un seul enregistrement mais plusieurs. Par exemple, on veut pouvoir traiter un groupe de personne. On ne va donc pas créer autant de variables du type personne qu'il y a de personnes. On va créer un tableau regroupant toutes les personnes du groupe. Il s'agit alors d'un tableau d'enregistrements.

Exemple :

groupe: tableau [1..300] de tpersonne

Chaque élément du tableau est un enregistrement, contenant plusieurs variables de type différent. On accède à un enregistrement par son indice dans le tableau.

groupe[2] représente la deuxième personne du groupe

groupe[2].nom représente le nom de la deuxième personne du groupe

Chapitre 9

Les fichiers

9.1. Généralités sur les fichiers

Généralement, avant l'utilisation de la notion de fichier, nous saisissons les données à partir du clavier. Ces données sont volatiles puisqu'elles sont mémorisées en mémoire centrale.

Or, dans la pratique, nous avons besoin de conserver ces données pour des traitements ultérieures. Pour cette raison, ces données sont alors stockées sur un support (disques, flash USB, etc..

Dans ce dernier cas, les données peuvent être lues à partir d'un fichier afin d'être traitées, comme elles peuvent être écrites dans le fichier pour qu'elles deviennent conservées.

9.2. Définition

Un fichier est un regroupement d'informations sur un support non volatile tel que le disque (LAPORTE, 2009).

9.2.1. Les types de fichiers

Il existe de nombreux types de fichiers, qui différent en fonction des langages. Nous distinguerons les fichiers de type « texte » et les fichiers structurés.

1. **Les fichiers structurés** permettent d'enregistrer des données de même nature. Ils sont composés d'enregistrements (ou articles) contenant les mêmes champs (ou rubrique). Généralement, chaque enregistrement correspond à une ligne, et les rubriques sont séparées par un séparateur qui peut être par exemple un point-virgule (format csv).
2. **Les fichiers non structurés** contiennent des données hétérogènes. Ils ne sont pas obligatoirement structurés en enregistrements, mais ils ne peuvent être lus que de manière séquentielle. Ex : un fichier word, où on peut trouver les données comme (image, texte, graphique, etc.).

9.2.2. Mode d'accès

On appelle mode d'accès à un fichier la façon de retrouver une donnée dans ce fichier. Il existe deux modes d'accès principaux :

1. **l'accès séquentiel** : possible sur tous les types de support et avec tous les types de fichiers Pour accéder à un article particulier, on doit avoir parcouru tous les articles précédents sur le support (LAPORTE, 2009).
2. **l'accès direct** : possible seulement sur support adressable avec des fichiers structurés On peut se positionner directement sur l'enregistrement voulu à partir de son emplacement sur le support (ou à partir d'une clé). Un fichier à accès direct peut être vu en quelque sorte comme un énorme tableau stocké sur support non volatile (LAPORTE, 2009).

9.3. Les fichiers à accès séquentiel

Un fichier séquentiel est un fichier dont les données ne sont accessibles que de manière consécutive sur le support. Pour accéder à un enregistrement particulier, il faut parcourir le fichier depuis le début jusqu'à trouver l'enregistrement recherché.

Si on assimile un fichier à un roman, La recherche d'un mot dans ce dernier exige une recherche séquentielle.

9.3.1. Déclaration

Pour déclarer un fichier séquentiel structuré, il faut d'abord déclarer les articles (enregistrements) qu'il va contenir. Les articles sont de type enregistrement.

Syntaxe

nom_fichier : **fichier séquentiel de** type_articles

9.3.2. Ouverture et fermeture

Afin d'accéder au fichier, il est nécessaire de l'ouvrir avant de pouvoir l'utiliser. L'ouverture permet la réservation d'une mémoire tampon en mémoire centrale pour les échanges entre le disque et la mémoire centrale.

Il existe deux modes principaux d'ouverture d'un fichier:

1. **en lecture** pour récupérer des données du fichier dans des variables de la mémoire centrale.
2. **en écriture** pour créer un fichier dans lequel enregistrer des résultats.

Syntaxe

Ouvrir nom_fichier **en** mode_ouverture

Exemple

Ouvrir personnel **en** lecture

Un fichier doit être fermé après son utilisation, afin de libérer la mémoire tampon allouée lors de l'ouverture et d'enregistrer les dernières données du tampon non encore transférées.

Syntaxe

Fermer nom_fichier

9.3.3. Lecture et Ecriture

La communication entre la mémoire centrale et le fichier peut se faire dans les deux sens:

1. de la mémoire au fichier : c'est l'écriture
2. ou du fichier vers la mémoire : c'est la lecture

Syntaxe

lire(nom_fichier,nomvariable)

ecrire(nom_fichier,nomvariable)

9.3.4. Fonction de test de fin de fichier

La fin de fichier est détectée par la fonction EOF (End Of File) qui renvoie vrai si la fin de fichier est sinon la valeur Faux.

Syntaxe

EOF(nom_fichier)

9.4. Exemple

```
Algorithme comptes_clients
Type
        tcompte = enregistrement
                num: entier
                etat: caractère /*N pour normal, I pour impayé, C pour contentieux*/
                solde: réel
        finenregistrement
Variable
        compte: tcompte
        cpteclt : fichier séquentiel de tcompte
        cltcont: fichier séquentiel d'entier
        rep: caractère
        n, i : entier
        tabcompte : tableau[1..100] de tcompte
```

```
        /*tableau d'enregistrement pour le chargement en mémoire*/
Début
/* création et remplissage du fichier */

        Ouvrir cpteclt en ecriture
        // on saisit les informations sur les clients et on les écrit dans le fichier
        Répéter
                // on saisit les champs de l'enregistrement
                ecrire( "Numéro?")
                Lire( compte.num)
                ecrire ("etat?")
                Lire(  compte.etat)
                ecrire ("solde?")
                Lire(  compte.solde)
                // on recopie l'enregistrement dans le premier article vide du fichier
                ecrire (cpteclt, compte)
                ecrire ("Autre compte? (O/N)")
                Lire(  rep)
        Jusqu'à (rep = 'O')
        Fermer cpteclt

/* lecture des articles du fichier */
        Ouvrir cpteclt en le cture
        // On lit les articles et les affiche tant qu'on n'a pas atteint la fin du fichier
        Tantque non eof(cpteclt) faire
                /*récupérer l'article courant dans l'enregistrement en mémoire centrale*/
                lire (cpteclt,compte)
                // Afficher les champs de l'enregistrement
                ecire( "Numéro: ", compte.num)
                ecire( "Solde: ", compte.solde)
                ecire( "Etat:")
                Selon compte.etat Faire
                        'N': ecire( "Normal")
                        'I':  ecire( "Impayé")
                        'C': ecire( "Contentieux")
                FinSelon
        FinTantque
        Fermer (cpteclt)
/* ajout d'un compte client à la fin du fichier */
        Ouvrir cpteclt en mode ajout
        // on saisit un enregistrement correspondant à l'article à ajouter
        ecire( "Numéro?")
        lire (compte.num
        ecire( "etat?")
        lire (compte.etat)
        ecire( "solde?")
        lire (compte.solde)
        // on recopie l'enregistrement sur le fichier
        ecrire (cpteclt, compte)
        Fermer (cpteclt)
```

```
/*modification d'un compte client, le 5689*/

        /* Chargement en mémoire centrale, dans le tableau d'enregistrements
        tabcompte et modification*/
        Ouvrir cpteclt en lecture
        n ← 0
        Tantque non eof(cpteclt) Faire
                n ← n + 1
                lire (cpteclt, tabcompte[n])
                Si (tabcompte[n].num = 5689 )Alors
                        tabcompte[n].solde ← tabcompte[n].solde – 500
                FinSi
        FinTantque
        Fermer (cpteclt)
        // Sauvegarde du tableau dans le fichier
        Ouvrir cpteclt en ecriture
        // n contient le nombre d'enregistrements
        Pour i de 1 jusqu'à n Faire
                ecrire (cpteclt, tabcompte[i])
        FinPour
        Fermer (cpteclt)
/*Suppression d'un compte, le 1268*/
        //chargement
        Ouvrir cpteclt en lecture
        n ← 0
        Tantque non eof(cpteclt ) Faire
                n ← n + 1
                lire (cpteclt, tabcompte[n])
        FinTantque
        Fermer (cpteclt)
        //sauvegarde (sauf le compte 1268)
        Ouvrir cpteclt en ecriture
        Pour i de 1 jusqu'à n faire
                Si (tabcompte[i].num ≠ 1268) alors
                        Ecrire (cpteclt, tabcompte[i])
                FinSi
        FinPour
        Fermer (cpteclt)
/* stockage dans un fichier séparé du numéro de tous les clients en contentieux.*/
// cltcont : fichier séquentiel d'entiers,  fichier contenant des numéros de clients
        Ouvrir cpteclt en lecture
        Ouvrir cltcont en écriture
        Tantque non eof ( cpteclt ) Faire
                Lire (cpteclt, compte)
                Si (compte.etat = 'C') Alors
                        ecrire (cltcont, compte.num)
                Finsi
        FinTantque
        Fermer (cpteclt)
        Fermer (cltcont)
Fin
```

Chapitre 10

Les algorithmes de recherche

Soit le type vecteur défini de la manière suivante :

Type vecteur= tableau [1..4000] de réel

Nous voulons écrire des fonctions qui permettent de vérifier si une valeur appelée **val** est présente dans un tableau (vecteur) T de taille n.

10.1. Définition d'un tableau trié

Un tableau est dit trié dans l'ordre croissant s'il contient 0 élément ou si
$\forall$ 1<= i<=n-1 on a T[i]<=T[i+1].

Un tableau est dit trié dans l'ordre décroissant s'il contient 0 élément ou si
$\forall$ 1<= i<=n-1 on a T[i]>=T[i+1].

10.2. Recherche séquentielle

L'algorithme le plus simple est la recherche séquentielle qui consiste à parcourir le vecteur t en partant de la première case jusqu'à la dernière case. A chaque fois on vérifie si l'élément en cours est égal à la valeur recherchée. Si c'est le cas on arrête le parcours si non on examine l'élément suivant si la taille du vecteur n n'est pas encore atteinte.

```
FONCTION Recherche_lineaire (T : vecteur, n : entier, val : réel ) : booléen
        Variable
        i : entier
        TROUVE : booléen
Début
        i ← 1
        TROUVE ←Faux
        Tantque(i<n+1 et TROUVE =Faux) faire
                si(T[i]=val) alors
                        TROUVE ←Vrai
                Finsi
                i←i+1
        fintantque
        Recherceh_lineaire← TROUVE
   Fin
```

10.3. Recherche dichotomique

Le principe de ce type de recherche exige que le tableau soit trié. En effet, pour rechercher un élément dans T, on le compare à l'élément qui se trouve au milieu du tableau T. En cas d'inégalité, on relance la recherche sur l'une des moitiés du tableau.

Principe détaillé :

Supposons que le tableau est trié d'une manière croissante. Le principe de recherche dichotomique consiste à diviser le tableau en deux parties sensiblement égales. On fait une comparaison de l'élément recherché avec l'élément médium (case du milieu du tableau). S'il est plus grand que le médium, on doit donc continuer la recherche dans la deuxième partie (qui correspond au indices supérieurs) et on ignore la première (car les éléments de cette partie sont tous inférieurs au médium). Dans le cas contraire la recherche s'effectue dans la première partie du tableau (indices inférieurs).

La partie retenue du tableau doit elle même être divisée en deux parties. Ainsi de suite jusqu'à réduire la partie retenue à un seul élément. L'indice du médium est calculé en faisant le quotient de la division euclidienne par 2 de la somme de l'indice le plus petit (Borne inférieure) et l'indice le plus grand (Borne supérieure) dans la partie retenue du tableau.

L'application de ce principe sur l'exemple ci-dessous, illustre les étapes de cet algorithme. Avec comme élément cherché =215.

Eta initial

Gauche			Milieu			Droite
3	12	45	122	200	210	215

Premier passage :

122< 215; Gauche←5 ; Milieu←6

				Gauche	Milieu	Droite
3	12	45	122	200	210	215

Deuxième passage :

210< 215; Gauche←7 ; Milieu←7

					Gauche Droite Milieu	
3	12	45	122	200	210	215

Troisième passage :

On arrête en effet T[Milieu]=215;

L'application de ce principe sur l'exemple ci-dessous, illustre les étapes de cet algorithme. Avec comme valeur cherchée val =45.

Eta initial

Gauche			Milieu			Droite
3	12	45	122	200	210	215

Premier passage :

122> 45; Droite ←3 ; Milieu ←2

Gauche	Milieu	Droite				
3	12	45	122	200	210	215

Deuxième passage :

12<45; Gauche←2 ; Milieu←2

Gauche Milieu	Droite					
3	12	45	122	200	210	215

Troisième passage :

On arrête en effet T[Milieu]=45;

L'application de ce principe sur l'exemple ci-dessous, illustre les étapes de cet algorithme. Avec comme valeur cherchée val =4.

Eta initial

Gauche			Milieu			Droite
3	12	45	122	200	210	215

Premier passage :

122> 4; Droite ←3 ; Milieu ←2

Gauche	Milieu	Droite				
3	12	45	122	200	210	215

Deuxième passage :

12>4; Droite ←1; Milieu ←1

Gauche

Droite
Milieu

3	12	45	122	200	210	215

Troisième passage :

On arrête en effet (Gauche < Droite) est fausse

```
FONCTION Recherceh_dichotomique (T : vecteur, n : entier, val : réel) : booléen
        Variable
        Gauche, Droite, Milieu : entier
Début
        Gauche ← 1
        Droite ←Faux
        Milieu  ←(Droite+Gauche) Div 2
        Tantque(Gauche < Droite T[Milieu] ≠val) faire
                si(T[Milieu]>val) alors
                        Droite← Milieu -1
                sinon
                        Gauche ← Milieu +1
                Finsi
                Milieu  ←(Droite+ Gauche) Div 2
        fintantque
        Recherceh_dichotomique ← (T[Milieu] =val )
Fin
```

Chapitre 11

Les algorithmes de tri

Lorsque les éléments du tableau à trier possèdent un ordre total, on peut donc les rangers en ordre croissant ou décroissant.

Il existe plusieurs méthodes de tri qui se différencient par leur temps d'exécution.

Dans ce support de cours, nous nous limitons aux tri à bulle, tri par sélection et tri par insertion.

11.1. Le tri à bulles

Principe :

Le principe du tri bulle est de comparer deux à deux les éléments e1 et e2 consécutifs d'un tableau et d'effecteur une permutation si e1 > e2. On continue de trier les éléments du tableau jusqu'à ce qu'il n'y ait plus de permutation.

Exemple :

143	12	122	3	21

Premier passage :

12	143	122	3	21

12	122	143	3	21

12	122	3	143	21

12	122	3	21	143

Deuxième passage :

12	122	3	21	143

12	3	122	21	143

12	3	21	122	143

12	3	21	122	143

Troisième passage :

12	3	21	122	143

3	12	21	122	143

3	12	21	122	143

3	12	21	122	143

Quatrième passage :

12	3	21	122	143

3	12	21	122	143

3	12	21	122	143

3	12	21	122	143

A ce passage aucune permutation n'est effectuée et le traitement est arrêté. Ainsi, le tableau est trié.

Algorithme :

```
PRODECURE Tri_bulle (T : vecteur, n : entier)
        VARIABLE
        permut : Booleen
Debut
        REPETER
                permut = FAUX
                pour i de 1 à n-1 FAIRE
                        SI T[i] > T[i+1] ALORS
                                permuter (a[i],a[i+1])
                                permut = VRAI
                        FIN SI
                FIN POUR
        Jusqu'à (permut = FAUX)
FIN
```

11.2. Le tri par sélection

Principe :

Le principe du tri par sélection/échange (ou ***tri par extraction***) est d'aller chercher le plus petit élément du vecteur pour le mettre en premier, puis de repartir du second élément et d'aller chercher le plus petit élément du vecteur pour le mettre en second, etc...

Exemple :

143	12	122	3	21

Premier passage : le plus petit élément est à la position 3, on permute la valeur T[3] et la valeur T[1]. Le tableau devient :

3	12	122	143	21

Deuxième passage : le plus petit élément est à la position 2, on permute la valeur T[2] et la valeur T[2]. Le tableau devient :

3	12	122	143	21

Troisième passage : le plus petit élément est à la position 5, on permute la valeur T[5] et la valeur T[3]. Le tableau devient :

3	12	21	143	122

Quatrième passage : le plus petit élément est à la position 5, on permute la valeur T[5] et la valeur T[4]. Le tableau devient :

3	12	21	122	143

Algorithme :

```
PRODECURE Tri_Selection (T : vecteur, n : entier)
        Variable
        indice_min, i,j : entier
Debut
        pour i de 1 à n-1 FAIRE
                indice_min ← i
                POUR j de i+1 à n FAIRE
                        si T[j]<max alors
                                indice_min ←j
                        finsi
                Finour
                Permuter( T[i], T[indice_min])
        Finour
FIN
```

11.3. Le tri par insertion

Principe :

Le principe du tri par insertion est d'insérer à la n-ième itération le n-ième élément à la bonne place dans le sous tableau de 1 à n-1 qui est déjà trié.

Exemple :

Dans cet exemple, nous voulons trier un tableau de 5 éléments dans l'ordre croissant. La technique du tri par sélection est la suivante : on met en bonne position l'élément numéro 2 dans la partie gauche du tableau qui est déjà trié. En passe ensuite à l'élément 3 et ainsi de suite jusqu'au dernier. Par exemple, si l'on part de :

143	12	122	3	21

Premier passage :

12	143	122	3	21

Deuxième passage :

12	122	143	3	21

Troisième passage :

3	12	122	143	21

Quatrième passage

3	12	21	122	143

Algorithme

```
PRODECURE Tri_Insertion (T : vecteur, n : entier)
        Variable
        v, i,j : entier
debut
        pour i de 2 à n faire
                v← T [i]
                j←i
                tantque(T [j-1]>v) faire
                        T [j]← T [j-1]
                        j←j-1
                fintantque
                T [j]←v
        finpour
fin
```

Chapitre 12

La récursivité

12.1. Définition

Une définition est dite récursive (récurrente) si dans l'énoncé de la définition intervient ce que l'on veut définir.

Exemple

- Un répertoire est un élément informatique qui peut contenir des fichiers et des répertoires.

En algorithmique, une procédure récursive est une procédure qui s'appelle elle-même.

12.2. Principe de la récursivité

Le principe est le même que celui de la démonstration par récurrence en mathématiques. On doit avoir :

- un certain nombre de cas dont la résolution est connue, ces « cas simples» formeront les cas d'arrêt de la récursion.
- un moyen de se ramener d'un cas « compliqué » à un cas « plus simple ».

12.3. Récursivité simple

Une procédure p est définie récursivement simple si p appelle lui-même une seule fois.

12.3.1. La fonction factorielle

La fonction factorielle est définie de la manière suivante :

Fact(n)=1*2*..n

Cette fonction est définie d'une manière récursive comme suit :

```
FONCTION Fact( n : entier)  : entier
Debut
        Si n=0 alors
```

```
        Fact←1
    Sinon
        Fact←Fact(n-1)*n
    finsi
fin
```

12.3.2. La fonction exponentielle

La fonction exponentielle est définie de la manière suivante :

exp(x,n)=x*x*..x, (n fois)

Cette fonction est définie d'une manière récursive comme suit :

```
FONCTION exp(x : réel n : entier)  : entier
Debut
    Si n=0 alors
        exp←1
    Sinon
        exp←exp(x,n-1)*x
    finsi
fin
```

12.4. Récursivité multiple

Une procédure récursive peut contenir plus d'un appel récursif.

12.4.1. La fonction combinaison

La fonction comb est définie de la manière suivante :

```
FONCTION comb(n: entier, m :entier) : entier
Debut
    Si p=0 ou p=n alors
        comb←1
    Sinon
```

```
            comb ← comb (n-1,p)+comb(n-1,p-1)
        finsi
fin
```

12.5. Récursivité mutuelle (ou indirecte)

Des définitions sont dites mutuellement récursives si elles dépendent les unes des autres.

12.5.1. La fonction paire et la fonction impaire

```
FONCTION paire(n : entier) : booléen
Debut
    si (n=0) alors
            paire←vrai
    sinon
            paire←impaire(n-1)
    finsi
Fin
FONCTION impaire(n : entier) : booléen
Debut
    si (n=0) alors
            impaire ←Faux
    sinon
            impaire ← paire (n-1)
    finsi
Fin
```

12.6. Récursivité imbriquée

On parle de la récursivité imbriquée quand la valeur de retour d'une fonction récursive est passée en paramètre à cette même fonction.

12.6.1. La fonction d'Ackermann

La fonction d'Ackermann est définie comme suit

A(m,n):

```
FONCTION A(m : entier ,n : entier) : entier
Debut
    si (n=0) alors
        A ←n+1
    sinon
        si (m>0 et n=0) alors
            A ← A(m-1,1)
        Sinon
            A ← A(m-1,A(m,n-1))
        finsi
    finsi
Fin
```

Chapitre 13

Notion de pointeur

13.1. Introduction

Toute variable manipulée dans un algorithme est stockée en mémoire centrale. Cette mémoire est constituée d'octets qui sont identifiés de manière univoque par un numéro qu'on appelle ***adresse***. Pour retrouver une variable, il suffit donc de connaître l'adresse de l'octet où elle est stockée (ou, s'il s'agit d'une variable qui recouvre plusieurs octets contigus, l'adresse du premier de ces octets). Pour des raisons évidentes de lisibilité, on désigne souvent les variables par des identificateurs, et non par leur adresse. C'est le compilateur qui fait alors le lien entre l'identificateur d'une variable et son adresse en mémoire. Toutefois, il est parfois très pratique de manipuler directement une variable par son adresse (HARDOUIN, 2009)

13.2. Définition

Un pointeur est une variable qui contient l'adresse d'une autre variable dans la mémoire.

Les pointeurs sont typés : si T dénote un type, alors ^T dénote le type "adresse d'un objet de type T".

13.3. Déclaration

La déclaration d'une variable de type pointeur se fait de la manière suivante :

Nom_variable : ^Type

Exemple

p,p1 :^entier

Nom_variable^ est la variable pointée par Nom_variable. La déclaration d'une variable de type pointeur ne réserve pas un espace mémoire pour la variable pointée.

13.4. Opérations sur les pointeurs

13.4.1. Affectation, addition, soustraction et différence

La valeur d'un pointeur est un entier, on peut lui appliquer un certain nombre d'opérateurs arithmétiques classiques. Les seules opérations arithmétiques valides sur les pointeurs sont :

- L'affectation d'un pointeur p à un pointeur p1 de même type que p.

```
p,p1 : ^Type
        p1←p
```

- l'addition d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ.

```
p,p1 : ^Type
n : entier
        p1←p+n
```

- la soustraction d'un entier à un pointeur. Le résultat est un pointeur de même type que le pointeur de départ.

```
p,p1 : ^Type
n : entier
        p1←p-n
```

- la différence de deux pointeurs pointant tous deux vers des objets de même type. Le résultat est un entier.

```
p,p1 : ^Type
n : entier
n←p1-p
```

Notons que la somme de deux pointeurs n'est pas autorisée.

13.4.2. Allocation

La fonction nouveau (^T), permet de réserver une zone mémoire de taille égale à la taille d'une variable de type T. la fonction renvoie l'adresse de cette zone mémoire qui sera sauvegarder dans une variable de type pointeur sur T.

Exemple

```
p←nouveau (^entier)
```

13.4.3. Destruction

La fonction liberer (p), permet de libérer la zone mémoire dans l'adresse se trouve dans p.

Exemple

```
liberer(p)
```

Série 1 : Enoncé

Exercice 1 :

Evaluer si c'est possible les expressions suivantes en tenant compte des priorités des opérateurs, et des valeurs suivantes des données :

(a = 15 : entier) , (b = '#') , (c = VRAI) , (d = 1025,2508)

(a > 20) ET c ((d – 15) = (d – a))	(d <> a) ET (c = VRAI)
((a + b) > a) <> c	a OU b
((-a-a-a-a-a-a-a) > - 7a)	3/a – 44 *a = (3 – 44(a ^ 2))/a
a>b+a	(NON C) OU (b <> 'q')
33 + 55 + 66 + 98 + 102 * 15 / a	(a <> a) (b <> b) (c <> FAUX)
45 / a – 1500 / 100	(d /d – 1) <> 0 ET 44 > a

Exercice 2 :

Dire, si les écritures suivantes sont correctes, sachant que : R est réel, B est booléen, C est caractère et E est entier :

R ← 58 + 0 + 0 + 10 +	B ← 5
R ← -E - R	B ← C
C ← 'R'	C ← 8
E ← - 45 * 2	E ← E + +
E –10 ← 478 * 2	R ← B – 45
B ← R > E OU C <> '1'	B ← (R + 10) – (45) > 0 ET C = #

Exercice 3 :

Ecrire un algorithme qui permet de calculer la moyenne générale d'un étudiant et ce à partir des moyennes de ses 4 matières (supposées toutes de même coefficients) et de nous informer si l'étudiant passe ou non.

Exercice 4 :

Ecrire un algorithme qui, à partir de date de naissance de 2 individus nous informe si le premier est plus âgé que le deuxième ou non. Sachant que la date de naissance est sous la forme : numéro-jours / numéro-mois / année.

Exercice 5 :

Ecrire l'algorithme qui, à partir des coordonnées de 3 points dans le plan nous informe si ces points forment ensemble un triangle.

Exercice 6 :

Ecrire un algorithme qui nous permet de permuter la valeur de 3 entiers sous forme de rotation : (exemple : si au départ : a = 10, b =20 et c = 30. Après l'exécution de l'algorithme on aura : a=30, b=10 et c=20)

Exercice 7 :

Ecrire un algorithme qui nous permet de dire si un point (défini par ces coordonnées dans le plan) appartient à la surface d'un disque défini par les coordonnées de son centre et par son rayon.

Exercice 8 :

Ecrire un algorithme qui nous permet de dire si un point (défini par ces coordonnées dans le plan) appartient à un cercle défini par les coordonnées de son centre et par son diamètre ou non.

Exercice 9 :

Ecrire un algorithme qui nous permet de nous informer si un point (définis par ces coordonnées dans le plan) appartient ou non à la surface d'un rectangle défini dans le plan par les coordonnées de son coin haut gauche et son coin bas droit.

Exercice 10 :

Ecrire un algorithme qui nous permet de convertir un volume horaire en secondes en son équivalent exprimé en heures, minutes et secondes.

Série 1 : Correction

Exercice 1 :

Evaluation impossible	VRAI
Evaluation impossible	Evaluation impossible
FAUX	FAUX
Evaluation impossible	VRAI
254	Evaluation impossible
12	FAUX

Exercice 2 :

Incorrecte	Incorrecte
Correcte	Incorrecte
Correcte	Incorrecte
Correcte	Correcte
Incorrecte	Incorrecte
Incorrecte	Incorrecte

Exercice 3 :

```
Algorithme Calcul_Moyenne
Variable
NM1, NM2, NM3, NM4, Moyenne : réel
Debut
    ecrire("donner note de la matière 1")
    lire(NM1)
    ecrire("donner note de la matière 2")
    lire(NM2)
    ecrire("donner note de la matière 3")
    lire(NM3)
    ecrire("donner note de la matière 4")
    lire(NM4)
    Moyenne←(NM1+NM2+NM3+NM4)/4

    Si(Moyenne>10)alors
        ecrire("Admis")
    sinon
        ecrire("Redouble")
    finsi
fin
```

Exercice 4 :

```
Algorithme Compare_Age
Variable
j1,j2, m1,m2,a1,a2 : entier
Debut
    ecrire("Jours de naissance de l'individu 1")
    lire(j1)
    ecrire("mois de naissance de l'individu 1")
    lire(m1)
    ecrire("année de naissance de l'individu 1")
    lire(a1)
    ecrire("Jours de naissance de l'individu 2")
    lire(j2)
    ecrire("mois de naissance de l'individu 2")
    lire(m2)
    ecrire("année de naissance de l'individu 2")
    lire(a2)
    Si(a1>a2)alors
        ecrire("l'individu 1 est plus âgé que l'individu 2")
    sinon
        Si(a1<a2)alors
            ecrire("l'individu 2 est plus âgé que l'individu 1")
        sinon
            Si(m1>m2)alors
                ecrire("l'individu 1 est plus âgé que l'individu 2")
            sinon
                Si(m1<m2)alors
                    ecrire("l'individu 2 est plus âgé que l'individu 1")
                sinon
                    Si(j1>j2)alors
                        ecrire("l'individu 1 est plus âgé que l'individu 2")
                    sinon
                        Si(j1<j2)alors
                            ecrire("l'individu 2 est plus âgé que l'individu 1")
                        sinon
                            ecrire("Même âge")
                        finsi
                    finsi
                finsi
            finsi
        finsi
    finsi
fin
```

Exercice 5 :

```
Algorithme triangle
Variable
x1, x2, x3, y1,y2,y3,a,b: réel
Debut
    ecrire("donner les coordonnées du point 1")
    lire(x1,y1)
    ecrire("donner les coordonnées du point 1")
    lire(x2,y2)
    ecrire("donner les coordonnées du point 1")
    lire(x3,y3)
    si((x1=x2) et (x2=x3) alors
        ecrire("Ces 3 points ne  forment pas un triangle ")
    sinon
        si(x1<>x2) alors
            a←(y2-y1)/(x2-x1)
        sinon
            si(x1<>x3) alors
                a←(y3-y1)/(x3-x1)
            sinon
                a←(y3-y2)/(x3-x2)
            finsi
        fin
        b←y1-a*x1
        si((ax1+b=0) et(ax2+b=0) et (ax3+b=0)) alors
            ecrire("Ces 3 points ne  forment  pas un triangle ")
        sinon
            ecrire("Ces 3 points forment  un triangle ")
        finsi
    finsi
fin
```

Exercice 6 :

```
Algorithme Permutation
    Variable
    a,b,c, aux: entier
Debut
    ecrire("donner trois entiers a, b et c")
    lire(a,b,c)
    aux←a
    a←c
    c←b
    b←aux
Fin
```

Exercice 7 :

```
Algorithme appartient_ surface_ disque
    Variable
    xm,ym, ,xc,yc,r : réel
Debut
    ecrire("donner les coordonnées du point M ")
    lire(xm,ym)
    ecrire("donner les coordonnées du centre ")
    lire(xc,yc)
    ecrire("donner le rayon ")
    lire(r)
    si(sqrt(sqr(xc-xm)+sqr(yc-ym))>r) alors
        ecrire("M n'appartient pas au disque ")
    sinon
        ecrire("M  appartient au disque ")
    finsi
Fin
```

Exercice 8 :

```
Algorithme appartient_ cercle
    Variable
    xm,ym, ,xc,yc,d, : réel
Debut
    ecrire("donner les coordonnées du point M ")
    lire(xm,ym)
    ecrire("donner les coordonnées du centre ")
    lire(xc,yc)
    ecrire("donner le diamètre ")
    lire(d)
    si(sqrt(sqr(xc-xm)+sqr(yc-ym))<>d/2) alors
        ecrire("M n'appartient pas au cercle ")
    sinon
        ecrire("M  appartient au cercle ")
    finsi
Fin
```

Exercice 9 :

```
Algorithme appartient_ rectangle
    Variable
    xm,ym, ,xhg,yhg,xbd,ybd : réel
Debut
    ecrire("donner les coordonnées du point M ")
    lire(xm,ym)
    ecrire("donner les coordonnées du coin haut gauche ")
```

```
        lire(xhg,yhg)
        ecrire("donner les coordonnées du coin bas droit ")
        lire(xbd,ybd)
        si( (xm>=xhg et xm<=xbd et ym>=ybd et ym<= yhg) alors
              ecrire("M n'appartient pas au rectangle ")
        sinon
              ecrire("n'appartient pas  au rectangle ")
        finsi
Fin
```

Exercice 10 :

```
Algorithme convertion_ secondes
      Variable
            h,m,s,v, : entier
Debut
      ecrire("donner un volume horaire ")
      lire(v)
      h←v div 3600
      m←(v-h*3600) div 60
      s←(v-h*3600-m*60)
      ecrire("heures =", h, " minutes =", m, " secondes =", s)
Fin
```

Série 2 : Enoncé

Exercice 1 :

Ecrire un algorithme permettant de savoir si une année est bissextile ou non. Une année est dite bissextile si elle est divisible par quatre sauf dans le cas où elle est divisible par 100 et non par 400.

Exercice 2 :

Ecrire un algorithme qui permet d'ajouter une seconde à une heure donnée représentée sous forme :

- heures
- minutes
- secondes.

Exercice 3 :

Facturation d'électricité : on désire calculer le montant que doit payer un abonnée, sachant que :

- Il à des frais fixes : 2,500 Dinars
- Les 100 premiers kwh son facturés à 50 millimes/kwh
- Les 150 kwh suivantes sont facturés à 35 millimes/kwh
- La consommation dépassant 250 kwh est facturée à 20 millimes/kwh.

Etant donnée, pour un abonné, les valeurs de l'ancien et du nouvel index, déterminer la somme à payer.

Exercice 4 :

Ecrire un algorithme qui permet étant donné le numéro du mois (1 à 12) de calculer le nombre de jours dans ce mois.

Exercice 5 :

Ecrire un algorithme qui demande deux nombres à l'utilisateur et l'informe ensuite si le produit est négatif, positif ou nul. On ne doit pas calculer le produit.

Exercice 6 :

Ecrire un algorithme qui permet de vérifier l'appartenance d'un nombre réel à un intervalle de IR

Série 2: Correction

Exercice 1 :

```
Algorithme année_bissextile
    Variable
        a : entier
Debut
    ecrire("donner une année ")
    lire(a)
    si( a mod 100) alors
        ecrire("année est bissextile ")
    sinon
        ecrire("année est non bissextile ")
    fin
Fin
```

Exercice 2 :

```
Algorithme ajout_ seconde
    Variable
        h,m,s,v, : entier
Debut
    ecrire("donner l'heure ")
    lire(h)
    ecrire("donner les minutes")
    lire(m)
    ecrire("donner  les secondes ")
    lire(m)
    si(s<>59) alors
        s←s+1
    sinon
          s←0
          si(m<>59) alors
                m←m+1
          sinon
                m←0
                si(m<>23) alors
                      h←h+1
                sinon
                      h←0
                finsi
          finsi
    finsi
    ecrire("heures =", h, " minutes =", m, " secondes =", s)
```

```
Fin
```

Exercice 3 :

```
Algorithme montant
    Variable
        nindex, aindex, m, c: entier
Debut
    ecrire("donner l'ancien index ")
    lire(aindex)
    ecrire("donner le nouvel index ")
    lire(nindex)
    c← nindex-aindex
    m←2500
    si (c<100) alors
        m← m+c*50
    sinon
        si(c<250) alors
                m←m+100*20+(c-100)*35
        sinon
                m←m+100*20+150*35+(c-250)*20
        finsi
    finsi
    ecrire("la somme à payer = ",m)
Fin
```

Exercice 4 :

```
Algorithme jours_dans_mois
    Variable
        m: entier
Debut
    ecrire("donner le numéro du mois ")
    lire(m)
    selon (m)
        1 : ecrire("le nombre de jours du mois = ",m, "est : 31")
        2 :  ecrire("le nombre de jours du mois = ",m, "est : 29")
        3 :  ecrire("le nombre de jours du mois = ",m, "est : 31")
        4 : ecrire("le nombre de jours du mois = ",m, "est : 30")
        5 : ecrire("le nombre de jours du mois = ",m, "est : 31")
        6 : ecrire("le nombre de jours du mois = ",m, "est : 30")
        7 : ecrire("le nombre de jours du mois = ",m, "est : 31")
        8 : ecrire("le nombre de jours du mois = ",m, "est : 31")
        9 :  ecrire("le nombre de jours du mois = ",m, "est : 30")
        10 : ecrire("le nombre de jours du mois = ",m, "est : 31")
        11: ecrire("le nombre de jours du mois = ",m, "est : 30")
        12: ecrire("le nombre de jours du mois = ",m, "est : 31")
```

```
            autre : ecrire("le nombre du mois  doit être entre 1 et 12"j)
        finselon
Fin
```

Exercice 5 :

```
Algorithme signe_ produit
        Variable
                x,y: réel
Debut
        ecrire("donner x ")
        lire(x)
        ecrire("donner y ")
        lire(y)
        si(x>0 et y<0) ou (x<0 et y>0) alors
                ecrire("le produit est négatif ")
        sinon
                ecrire("le produit est positif ou nul ")
        finsi
Fin
```

Exercice 6 :

```
Algorithme appartenance
        Variable
                x, a, b: réel
Debut
        ecrire("donner x ")
        lire(x)
        ecrire("donner a ")
        lire(a)
        ecrire("donner b ")
        lire(b)

        si(x>a et x<b) ou (x<a et x>b) alors
                ecrire("x appartient ")
        sinon
                ecrire("x n'appartient  pas")
        finsi
Fin
```

Série 3 : Enoncé

Exercice 1 :

Ecrire un algorithme qui permette de connaître la probabilité de gagner au tiercé, quarté et quinté.

Etant donnés le nombre de chevaux partants, et le nombre de chevaux joués. Il faut calculer la probabilité de gagner dans l'ordre, puis la probabilité de gagner dans le désordre.

X et Y qui sont respectivement les chances de gagner dans l'ordre et les chances dans le désordre, nous sont donnés par la formule suivante, si n est le nombre de chevaux partants et p le nombre de chevaux joués :

X = n ! / (n - p) !

Y = n ! / (p ! * (n – p) !)

Exercice 2 :

Ecrire l'algorithme qui permet de calculer la somme des N premiers entiers pairs.

Exercice 3 :

Ecrire l'algorithme qui permet de calculer le produit de tous les diviseurs d'un entiers N.

Exercice 4 :

Ecrire l'algorithme qui détermine si un entier N est parfait ou non. Un entier est dit parfait s'il est égal à la somme de ses diviseurs. Exemple 6 = 3 + 2 +1.

Exercice 5 :

Ecrire l'algorithme qui permet de calculer le produit de deux entiers en utilisant des additions successives.

Exercice 6 :

Ecrire l'algorithme qui permet de calculer la division de deux entiers en utilisant des soustractions successives.

Exercice 7 :

Ecrire l'algorithme qui permet de déterminer si un nombre n est premier ou non. Un nombre est dit premier s'il est divisible uniquement par 1 et par lui-même.

Exercice 9 :

On démontre en mathématique que le cosinus d'un angle exprimé en radian est donné par la somme infinie suivante :

COS(x) = 1 – X2 / 2! + X4/4! – X6/6! +…

Ecrire l'algorithme qui permet d'évaluer le cosinus d'une valeur x donnée, avec un développement à l'ordre n.

Ecrire l'algorithme qui permet d'évaluer le cosinus d'une valeur x donnée, avec une précision e donnée.

Exercice 10 :

Ecrire l'algorithme qui permet étant donné un entier positif N1 et qui calcule un deuxième nombre N2. Le nombre N2 doit obéir la condition suivante : s'il est interprétée dans la base 2, alors il sera égal à N1 interprété dans la base 10.

Exemple :

Si N1 = 7, alors N2 = 111 → (7)10 = (111)2

Exercice 11 :

Ecrire un algorithme qui permet d'entrer un entier et une base inférieure ou égale à 10 et de vérifier si ce nombre appartient à la base ou non.

Exercice 12 :

Ecrire un algorithme qui permet d'enter deux entiers et de vérifier si les chiffres du premier appartiennent à ceux du second nombre ou non.

Exercice 13 :

Ecrire un algorithme qui permet d'entrer deux entiers positifs et de déterminer leur plus grand commun diviseur (PGCD).

Le PGCD(A,B) = PGCD(A-B, B) si A est le plus grand et à PGCD(A,B) = PGCD(A, B-A) si B est le plus grand. Si A=B le PGCD(A,B) est A ou B.

Exercice 14 :

Ecrire l'algorithme qui permet d'entrer deux entiers et de déterminer leur plus petit commun multiple (PPCM).

Exercice 15 :

Ecrire un algorithme qui permet, étant donné un entier de calculer la partie entière de sa racine carrée.

Exercice 16 :

Ecrire un algorithme qui permet, étant donné un entier de calculer la somme des nombres premiers qui lui sont inférieur.

Série 3 : Correction

Exercice 1 :

```
Algorithme probabilité
    Variable
        n, p,fn,fp,fnp,i: entier
        x,y : réel
Debut
    ecrire("donner n ")
    lire(n)
    ecrire("donner p ")
    lire(p)
    fn←1
    pour i de 1 à n faire
        fn←fn*i
    finpour
    fp←1
    pour i de 1 à p faire
        fp←fp*i
    finpour
    fnp←1
    pour i de 1 à n-p  faire
        fnp←fnp*i
    finpour
    x←fn/fnp
    y←fn/(fp*fnp)
    ecrire("la probabilité de gagner dans l'ordre =",x)
    ecrire("la probabilité de gagner dans le désordre =",y)

Fin
```

Exercice 2 :

```
Algorithme somme_n_premiers_pairs
    Variable
        n, s,i: entier
Debut
    ecrire("donner n ")
    lire(n)
    s←0
    pour i de 2 à n pas de 2 faire
        s←s+i
    finpour
    ecrire("la somme des ",n, "est ",s)
```

```
Fin
```

Exercice 3 :

```
Algorithme produit_tous_diviseurs
    Variable
        n, p,i: entier
Debut
    ecrire("donner n ")
    lire(n)
    p←1
    pour i de 2 à (n div 2) faire
        si(n mod i=0) alors
          p←p*i
        finsi
    finpour
    p←p*n
    ecrire("le produit de tous les diviseurs de",n, "est ",p)
Fin
```

Exercice 4 :

```
Algorithme parfait
    Variable
        n, s,i: entier
Debut
    ecrire("donner n ")
    lire(n)
    s←0
    pour i de 1 à (n div 2) faire
        si(n mod i=0) alors
          s←s+i
        finsi
    finpour
    si(s =n) alors
        ecrire("le nombre ",n, "est parfait ")
    sinon
        ecrire("le nombre ",n, "n'est parfait ")

    finsi
Fin
```

Exercice 5 :

```
Algorithme somme
    Variable
        n,m, s,i: entier
Debut
```

```
        ecrire("donner n ")
        lire(n)
        ecrire("donner m ")
        lire(m)

        s←0
        pour i de 1 à n faire
              s←s+m
        finpour
        ecrire("la somme ",n, "+ ",m,"la somme ",s)
Fin
```

Exercice 6 :

```
Algorithme division
        Variable
              n,m, q,r: entier
Debut
        ecrire("donner n ")
        lire(n)
        ecrire("donner m ")
        lire(m)
        //on calcul n/m
        q←0
        tantque(n>=m) faire
              n←n-m
              q←q+1
        fintantque
        r←n
        ecrire("q=",q, "et r=", r)
Fin
```

Exercice 7 :

```
Algorithme division
        Variable
              n,m, q,r: entier
Debut
        ecrire("donner n ")
        lire(n)
        i←2
        //on cherche un diviseur de n
        tantque(n mod i<>0) faire
              i←i+1
        fintantque
        si(i =n  et n<>2) alors
              ecrire("le nombre ",n, "est premier ")
```

```
        sinon
            ecrire("le nombre ",n, " est non premier ")

        finsi
Fin
```

Exercice 8 :

```
Algorithme cosinus_ordre_n
    Variable
        i,n, f,signe: entier
        x,s,sx : réel
Debut
    ecrire("donner x ")
    lire(x)
    ecrire("donner n ")
    lire(n)
    s←1
    sx←1
    f←1
    pour i de 2 à  n pas de 2 faire
        sx←sx*x^2
        f←f*(i-1)*i
        signe←signe*(-1)
        s←s+signe*sx/f
    finpour
    ecrire("cosinus  de",x, "a l'ordre ",n, "= ",s)

Fin
Algorithme cosinus_ précision _donnée
    Variable
        i, f,signe: entier
        e,x,s,sx : réel
Debut
    ecrire("donner x ")
    lire(x)
    ecrire("donner e ")
    lire(e)
    s←1
    sx←1
    f←1
    i←2
    sx←sx*x^2
    f←f*(i-1)*i
    signe←signe*(-1)
    tantque(ABS(signe*sx/f)<e) faire
        s←s+signe*sx/f
```

```
            sx←sx*x^2
            f←f*(i-1)*i
            signe←signe*(-1)
            i←i+2
        fintantque
        ecrire("cosinus  de",x, "à une précision ",e, "= ",s)

Fin
```

Exercice 9 :

```
Algorithme conversion_B10_vers_B2
        Variable
                q,r,n1,n2: entier
Debut
        ecrire("donner n1 ")
        lire(n1)
        i←1
        q←n1
        répéter
                r←q mod 2
                q←q div 2
                n1=n1+*r*i
                i←i*10
        jusqu'à (q=0)
        ecrire(n,"en base 10=",n1, "en base 2")
Fin
```

Exercice 10 :

```
Algorithme base
        Variable
                q,r,n,b: entier
Debut
        ecrire("donner n ")
        lire(n)
        répéter
                ecrire("donner la base b ")
                lire(b)
        jusqu'à (b>0 et b<10)

        répéter
                r←q mod b
                q←q div b
        jusqu'à (q=0 ou r>b)
        si(r>b) alors
                ecrire("le nombre ",n, "n'est pas dans la base ",b)
```

```
        sinon
            ecrire("le nombre ",n, "est dans la base ",b)
        finsi
    Fin
```

Exercice 11 :

```
Algorithme chiffres_ appartiennent
    Variable
        n,m,qn,rn,qm,qn: entier
        ok : booléen
Debut
    ecrire("donner n ")
    lire(n)
    ecrire("donner m ")
    lire(m)

    répéter
        rn←n mod 10
        qn←qn div 10
        répéter
            rm←qm mod b
            qm←qm div b
          jusqu'à (qm=0 ou rn=rm)
        si(rn<>rm) alors
            ok←faux
        sinon
            ok←vrai
        finsi
    jusqu'à (qn=0 ou ok=faux)
    si(ok=vrai) alors
        ecrire("tous les chiffres du nombre ",n, "appartiennent ",m)
    sinon
        ecrire("au moins un chiffre du nombre ",n, "n' appartient pas ",m)
    finsi
Fin
```

Exercice 12 :

```
Algorithme PGCD
    Variable
        a,b: entier
Debut
    ecrire("donner a ")
    lire(a)
    ecrire("donner b ")
    lire(b)
```

```
        tantque(a<>b) faire
            si(a>b) alors
                a←a-b
            sinon
                b←b-a
            finsi
        fintantque
        ecrire("PGCD= ",a)
fin
```

Exercice 13 :

```
  Algorithme PPCM
        Variable
            m,n,a,b,aux: entier
  Debut
        ecrire("donner a ")
        lire(a)
        ecrire("donner b ")
        lire(b)
        n←1
        m←1
        si(a>b) alors
            aux←a
            a←b
            b←aux
        finsi
        tantque(n*a<>m*b) faire
              n←n+1
              m←1
              tantque(m*b<n*a) faire
                    m←m+1
              fintantque
        fintantque
        ecrire("PPCM = ",n*a)
fin
```

Exercice 14 :

```
  Algorithme partie_entière_racine_carrée
        Variable
            a :réel
            p: entier
  Debut
        ecrire("donner a ")
        lire(a)
        tantque(p^2<a) faire
```

```
            p←p+1
        fintantque
        ecrire("la partie entière de sa racine carrée de = ",a, " = ",p-1)
fin
```

Série 4 : Enoncé

Exercice 1 :

Ecrire un algorithme qui permet la détermination du plus petit élément d'un tableau d'entiers de taille 50 et qui nous fourni son indice dans le tableau.

Exercice 2 :

Ecrire un algorithme qui permet la détermination du plus petit élément et du plus petit élément d'un tableau de réels de taille 100.

Exercice 3 :

Ecrire un algorithme qui nous informe si le contenu d'un tableau de caractères de taille N forme un palindrome ou non. Sachant qu'une suite de caractères est dite palindrome lorsqu'elle se lit de gauche vers la droit comme droit vers la gauche tel que « ADA ».

Exercice 4 :

On suppose qu'on représente une courbe par l'ensemble des points à laquelle ils appartiennent. Dans un premier tableau on place les abscisses et dans un deuxième on place les ordonnés.

Ecrire un algorithme qui permet la transformation de la courbe formée de N points d'une échelle initial à une autre échelle.

Exercice 5 :

Ecrire un algorithme qui permet de compter le nombre de mots contenues dans un tableau de caractères de taille maximale 500. Sachant qu'on suppose que :

- la fin de la suite de caractères utile est marquée par le caractère '#',
- les séparateurs des mots sont les caractères : espace, ', ', ';' et '.',
- qu'on ne peut pas rencontrer deux séparateurs successives.

Exercice 6 :

Ecrire un algorithme qui permet la conversion d'un entier x du décimal au binaire. Sachant que le résultat doit être fourni dans un tableau et qu'on n'effectue pas la conversion si x est supérieur à 255. L'algorithme doit alors nous informé si il y avait eu de conversion ou non.

Exercice 7 :

Ecrire un algorithme qui permet de nous informé si une suite de caractères de taille maximale 150 et se terminant par '#' est l'image miroir d'une deuxième elle aussi de taille maximale 150 et se terminant par '#'.

Exercice 8 :

Ecrire un algorithme qui permet d'initialiser la première diagonale d'une matrice carrée 50 x 50 à zéro.

Exercice 9 :

Ecrire un algorithme qui permet d'initialiser les deux diagonales d'une matrice carrée N x N par le caractère '#'

Exercice 10 :

Ecrire un algorithme qui permet de copier les éléments d'une matrice d'entiers de taille 40 x 50 dans un tableau de taille 200.

Série 4 : Correction

Exercice 1 :

```
Algorithme indice_plus _petit_ élément
    Variable
        t: vecteur
        ipp,p,i: entier
Debut
    pour i de 1 à 50 faire
        lire(t[i])
    finpour
    ipp←1
    pour i de 2 à 50 faire
        si(t[i]<t[ipp]) alors
                ipp←i
        finsi
    finpour
    ecrire("le plus petit élément est à la position ",ipp)
fin
```

Exercice 2 :

```
Algorithme plus _petit_ élément
    Variable
        t: vecteur
        pp : réel
        i,p: entier
Debut
    pour i de 1 à 100 faire
        lire(t[i])
    finpour
    pp←t[1]
    pour i de 2 à 100 faire
        si(t[i]<pp) alors
          pp← t[i]
        finsi
    finpour
    ecrire("le plus petit élément est ",pp)
 fin
```

Exercice 3 :

```
Algorithme palindrome
    Variable
        t: vecteur
        n,i: entier
Debut
    ecrire("donner n")
    lire(n)
    pour i de 1 à n faire
        lire(t[i])
    finpour
    tantque(g<d et t[g]=t[d]) faire
        g←g+1
        d←d-1
    fintantque
    si g<d alors
        ecrire(("palindrome ")
    sinon
        ecrire(("non palindrome ")
    finsi
 fin
```

Exercice 4 :

```
Algorithme échelle
    Variable
        tx,ty: vecteur
        n,i: entier
        e : réel
Debut
    ecrire("donner n")
    lire(n)
    ecrire("donner e")
    lire(e)

    pour i de 1 à n faire
        lire(tx[i])
        lire(tx[i])
    finpour
    pour i de 1 à n faire
        tx[i]← tx[i]*e
        ty[i]← ty[i]*e finpour
 fin
```

Exercice 5 :

```
Algorithme nombre_mot
    Variable
```

```
        t: vecteur
        nbmot,i: entier
Debut
    i←1
    répéter
        lire(t[i])
    jusqu'à (t[i]='#')
    i←1
    tantque(t[i]<>'#')
        si(t[i]=' 'ou t[i]=' ;' ou t[i]=',' ou t[i]=' .') alors
        nbmot← nbmot+1
        finsi
    fintantque
    ecrire(("nombre des mots= ", nbmot)


 fin
```

Exercice 6 :

```
Algorithme conversion_B10_vers_B2
    Variable
        q,r,x,n: entier
        t : vecteur
Debut
    ecrire("donner x ")
    lire(x)
    si  x> 255 alors
        ecrire ("conversion non réalisée ")
    sinon
        n←0
        q←x
        répéter
            r←q mod 2
            q←q div 2
            n←n+1
            t[n]←r
          jusqu'à (q=0)

          pour i de n à 1 pas de -1 faire
                ecrire(t[n])
          finpour
        ecrire ("conversion réalisée ")
    finsi

Fin
```

Exercice 7 :

```
Algorithme miroir
    Variable
        i,n: entier
        t1,t2 :vecteur
Debut
    n1←1
    Répéter
        lire(t1[n1])
        n1←n1+1
    Jusqu'à (t1[n1]='# ')
    n2←1
    Répéter
        lire(t2[n2])
        n2←n2+1
    Jusqu'à (t2[n2]='# ')

    tantque(i<=n1 et n2-i>=1 et t1[i]=t2[n2-i) faire
        i←i+1
    fintantque

    si  i=n1+1 alors
        ecrire ("Miroir ")
    sinon
        ecrire ("non Miroir ")
    finsi

Fin
```

Exercice 8 :

```
Algorithme Matrice
    Variable
        l,c,i,j: entier
        m :matrice
Debut
    ecrire("donner l ")
    lire(l)
    ecrire("donner c ")
    lire(c)
    pour i de 1 à l faire
        pour i de 1 à c faire
            si(i=j) alors
                m[i,j]←0
            finsi
        finpour
```

```
        finpour
Fin
```

Exercice 9 :

```
Algorithme Matrice_DIAGONAL
        Variable
                l,c,i,j: entier
                m :matrice
Debut
        ecrire("donner l ")
        lire(l)
        ecrire("donner c ")
        lire(c)
        pour i de 1 à l faire
                pour i de 1 à c faire
                        si(i=j ou j=l-i-1) alors
                                m[i,j]← '#'
                        finsi
                finpour
        finpour
Fin
```

Exercice 10 :

```
Algorithme copier _Matrice_vecteur
        Variable
                l,c,i,j: entier
                m :matrice
                t : vecteur
Debut
        ecrire("donner l ")
        lire(l)
        ecrire("donner c ")
        lire(c)
        pour i de 1 à l faire
                pour i de 1 à c faire
                        lire(m[i,j])
                finpour
        finpour
        pour i de 1 à l faire
                pour i de 1 à c faire
                        t[(i-1)*c+j]←m[i,j]
                finpour
        finpour

Fin
```

Série 5 : Enoncé

Exercice 1 :

Ecrire une fonction qui permet de renvoyer le PGCD de deux entiers m et n donnés en paramètres

Exercice 2 :

Ecrire une fonction qui permet de renvoyer le PPCM de deux entiers m et n donnés en paramètres.

Exercice 3 :

Ecrire une fonction qui permet de renvoyer la factorielle d'un entier n donné en paramètre.

Exercice 4 :

Ecrire une fonction qui renvoie vrai si un point (défini par ces coordonnées dans le plan) appartient à un cercle défini par les coordonnées de son centre et par son diamètre ou non.

Exercice 5 :

Ecrire une fonction qui permet de renvoyer le nombre d'apparition d'un entier val dans un tableau t de taille n.

Exercice 6 :

Ecrire une procédure qui permet de calculer la somme de deux tableaux dans un troisième.

Exercice 7 :

Ecrire une fonction qui permet de renvoyer vrai si un entier val existe dans un tableau t de taille n, faux si non.

Exercice 8 :

Ecrire une procédure qui permet de calculer l'intersection de deux tableaux dans un troisième.

Série 5 : Correction

Exercice 1 :

```
FONCTION PGCD(a : entier, b : entier) : entier
Debut
        tantque(a<>b) faire
                si(a>b) alors
                        a←a-b
                sinon
                        b←b-a
                finsi
        fintantque
        PGCD←a
fin
```

Exercice 2 :

```
FONCTION PPCM (a : entier, b: entier): entier
        Variable
                m,n, aux: entier
Debut
        n←1
        m←1
        si(a>b) alors
                aux←a
                a←b
                b←aux
        finsi
        tantque(n*a<>m*b) faire
                  n←n+1
                  m←1
                  tantque(m*b<n*a) faire
                          m←m+1
                  fintantque
        fintantque
        PPCM ←n*a
fin
```

Exercice 3 :

```
FONCTION FACT (n): entier
        Variable
                i ,f: entier
```

```
  Debut
        f←1
        pour i de 1 à n faire
              f←f*i
        finpour
        FACT ←f
fin
```

Exercice 4 :

```
  FONCTION  appartient_ cercle(xm : réel,ym: réel ,xc: réel : réel,yc: réel, d : réel) :
        booléen
  Debut
        si(sqrt(sqr(xc-xm)+sqr(yc-ym))=d/2) alors
              appartient_ cercle ←vrai
        sinon
              appartient_ cercle ←faux
        finsi
  Fin
```

Exercice 5 :

```
  FONCTION  nbocc(t :vecteur,n: entire ,val: réel) : entier
        Variable
        i,nb : entier
  Debut
        nb←0
        pour i de 1 à n faire
              si(t[i]=val) alors
                 nb←nb*1
              finsi
        finpour
        nbocc ←nb
  Fin
```

Exercice 6 :

```
  PROCEDURE  SOMME(t1 :vecteur, t2 :vecteur, t3 :vecteur, n :  entier)
        Variable
        i : entier
  Debut
        pour i de 1 à n faire
              t3[i]← t1[i]+ t2[i]
        finpour
  Fin
```

Exercice 7 :

```
FONCTION  existe(t :vecteur, n: entier ,val: réel) : entier
        Variable
        i : entier
Debut
        i←1
        tantque(i<=n et t[i]<>val) alors
                i←i+1
        fin tantque
        existe ←(i<=n)
Fin
```

Exercice 8 :

```
PROCEDURE  intersection (t1 :vecteur, t2 :vecteur, t3 :vecteur, n1 :  entier, n2 :  entier,
        var n3 :  entier)
        Variable
        i : entier
Debut
        i←1
        n3←0
        pour i de 1 à n faire
                si((existe(t2,n2, t1[i]) et non(existe(t3,n3, t1[i]))    alors
                                n3←n3+1
                                t3[n3]← t1[i]
                finsi
        finpour
Fin
```

Série 6 : Enoncé

Exercice 1 :

Donner les déclarations des types suivants:

1. date : numéro du jour, nom du mois, année;
2. point : coordonnées (entières) d'un point;
3. cercle : un point et son rayon.

Exercice 2 :

Ecrire la procédure AfficheDate affichant la date passée en paramètre.

Exercice 3 :

Ecrire une procédure DemandeDate demandant une date au clavier et remplissant les champs de la structure passée.

Exercice 4 :

Ecrire une fonction CompareDates prenant deux dates d1 et d2 et retournant -1 si d1 est avant d2, 1 si d1 est après d2, ou 0 si les deux dates sont égales.

Exercice 5 :

Ecrire une procedure TriDates triant un tableau de N dates (définies comme dans l'exercice 1), en utilisant la fonction CompareDates

Exercice 6 :

Ecrire une fonction qui renvoie vrai si un point appartient à un cercle ou non.

Série 6 : correction

Exercice 1 :

1. type date= enregistrement
   j : entier
   m : entier
   a : entier
   finenregistrement
2. type point= enregistrement
   x : réel
   y : réel
   a : entier
   finenregistrement

3. type cercle= enregistrement
   c : point
   r : réel
   finenregistrement

Exercice 2 :

```
PROCEDURE  AfficheDate (d :date)
Debut
    ecrier(d.j,"/", d.m,"/", d.m,"/")
Fin
```

Exercice 3 :

```
PROCEDURE  DemandeDate (var d :date)
Debut
    lire(d.j, d.m, d.m)
Fin
```

Exercice 4 :

```
FONCTION CompareDates (d1 :date, d2 : date) : entier
    Variable
    s1,s2 : entier
Debut
    s1←d1.j*100+ d1.m*10000+ d1.m*100000000
    s1←d2.j*100+ d2.m*10000+ d2.m*100000000
    si(s1>s2) alors
        CompareDates←1
```

```
        Sinon
            si(s2>s1) alors
                    CompareDates←-1
            sinon
                    CompareDates←0
            Finsi
        Finsi
  Fin
```

Exercice 5 :

```
PRODECURE TriDates (T : vecteur, n : entier)
        VARIABLE
        permut : Booleen
Debut
        REPETER
            permut = FAUX
            pour i de 1 à n-1 FAIRE
                SI (CompareDates (T[i] , T[i+1])>1) ALORS
                    permuter (a[i],a[i+1])
                    permut = VRAI
                FIN SI
            FIN POUR
        Jusqu'à (permut = FAUX)
FIN
```

Exercice 6 :

```
  FONCTION  appartient_ cercle(p :point, c : cercle) : booléen
  Debut
        si(sqrt(sqr(p.x-c.c.x)+sqr(p.y-c.c.y))=r) alors
            appartient_ cercle ←vrai
        sinon
            appartient_ cercle ←faux
        finsi
  Fin
```

Série 7 : Enoncé

Exercice 1 :

Ecrire une fonction qui permet de réaliser une recherche séquentielle sur une matrice. Les éléments de la matrice sont organisés de la manière suivante :

1. Les lignes de ce tableau sont tous trié par ordre croissant.

2. le maximum de chaque ligne est inférieur au minimum de la ligne suivante.

Exercice 2 :

Ecrire une fonction qui permet de réaliser une recherche dichotomique sur une matrice. Les éléments de la matrice sont organisés de la manière suivante :

1. Les lignes de ce tableau sont tous trié par ordre croissant.

2. le maximum de chaque ligne est inférieur au minimum de la ligne suivante.

Série 7: correction

Exercice 1 :

```
FONCTION  Recherche_ séquentielle _Matrice (m : matrice, l : entier, c : entier, val :
      réel) : booléen
      Variable
            i,j: entier
Debut
      i←1
      j←1
      tantque(t[(i-1)*c+j]<v  et i<=l) faire
            si(j=c) alors
                  i← i+1
                  j←1
            sinon
                  j←j+1
            finsi
      fintantque
      Recherche_ séquentielle _Matrice ←(t[(i-1)*c+j]=v al )

Fin
```

Exercice 2 :

```
FONCTION  Recherche_ dichotomique _Matrice(m : matrice, l : entier, c : entier, val :
      réel) : booléen
      Variable
            bas  , haut , Gauche, Droite , MilieuLigne , MilieuColonne  : entier
           m :matrice
Debut
      pour i de 1 à l faire
            pour i de 1 à c faire
                  lire(m[i,j])
            finpour
      finpour
      bas ←1
      haut ← l
      MilieuLigne  ←( haut + bas ) Div 2
      Tantque(bas  < haut  et Non(m[MilieuLigne ,1]>= val et m[MilieuLigne ,c]<= val ))
      faire
            si(m[MilieuLigne ,1]>val) alors
                  haut ← MilieuLigne  -1
            sinon
                  bas  ← MilieuLigne  +1
            Finsi
```

```
        MilieuLigne ←( bas  + haut  ) Div 2
Fintantque

Gauche ← 1
Droite ←l
MilieuColonne  ←(Droite+Gauche) Div 2
Tantque(Gauche < Droite T[MilieuLigne , MilieuColonne ] ≠val) faire
        si(T[MilieuLigne  , MilieuColonne ]>val) alors
                Droite← MilieuColonne  -1
        sinon
                Gauche ← MilieuColonne  +1
        Finsi
        MilieuColonne  ←(Droite+ Gauche) Div 2
fintantque
Recherche_ dichotomique _Matrice ←(m[MilieuLigne, MilieuColonne ] =val )
fin
```

Série 8 : Enoncé

Exercice 1 :

Ecrire une fonction récursive qui permet de renvoyer le PGCD de deux entiers m et n donnés en paramètres

Exercice 2 :

Ecrire une fonction récursive qui permet de renvoyer la factorielle d'un entier n donné en paramètre.

Exercice 3 :

Ecrire une fonction récursive qui permet de renvoyer le nombre d'apparition d'un entier val dans un tableau t de taille n.

Exercice 4 :

Ecrire une procédure récursive qui permet de calculer la somme de deux tableaux dans un troisième.

Exercice 5 :

Ecrire une fonction récursive qui permet de renvoyer vrai si un réel val existe dans un tableau t de taille n, faux si non.

Série 8 : correction

Exercice 1 :

```
FONCTION  PGCD (a : entier, b : entire) : entier
Debut
        si(a=b) alors
                PGCD ←a
        Sinon
                si(a>b) alors
                                PGCD ← PGCD (a-b,b)
                Sinon
                                PGCD ← PGCD (a,b-a)

        finsi
Fin
```

Exercice 2 :

```
FONCTION  FACT (n : entier) : entier
Debut
        si(n=1 ou n=0) alors
                FACT ←1
        Sinon
                FACT ←n* FACT (n-1)
        finsi
Fin
```

Exercice 3 :

```
FONCTION nbocc (t : vecteur,n : entier, val : entier) : entier
Debut
        si(n=0) alors
                nbocc ←0
        Sinon
                si(t[n]=val)alors
                                nbocc ←1+nbocc(t,n-1,val)
                sinon
                                nbocc←nbocc(t,n-1,val)
                finsi
        finsi
Fin
```

Exercice 4 :

```
PROCEDURE somme (t1 : vecteur, t2 : vecteur, t3 : vecteur, n : entier)
Debut
        si(n<>0) alors
                t3[n]←t1[n]+t2[n]
                somme(t1,t2,t3,n-1)
        finsi
Fin
```

Exercice 5 :

```
FONCTION existe (t : vecteur n : entier, val : réel) : booléen
Debut
        si(n=0) alors
                existe←faux
        sinon
                si(t[n]=val) alors
                                existe←vrai
                sinon
                                existe←existe(t,n-1,val)
        finsi
Fin
```