Université 8 Mai 45, Guelma
Faculté des sciences économiques, commerciales et sciences de gestion
Département des sciences de gestion

Informatique 2

Cours et exercices en algorithmique

Polycopié destiné aux étudiants de la deuxième année licence en science économiques, commerciales et sciences de gestion

Dr. AS Kelaiaia

Présentation

Ce polycopié est destiné principalement aux étudiants de la deuxième année licence en sciences économiques, commerciales et science de gestion, mais aussi peut être très utile aux étudiants de la deuxième année tronc-commun informatique.

L'auteur a essayé de simplifier le plus possible le contenu, par la simplification de la rédaction, d'une part, et d'autres par des exemples simples et intuitifs. Un ensemble consistant d'exercices a été aussi fourni à travers cinq séries de travaux dirigés et ce afin de faciliter la tâche au maximum aux enseignants assurant l'enseignement de la matière Informatique 2.

Sommaire

Chapitre I

Généralités et concepts fondamentaux

Introduction, les premiers pas en algorithmique…

L'ordinateur par lui-même ne peut rien faire, il lui faut des programmes lui indiquant quoi faire à tout moment. Ces programmes sont conçus par des programmeurs dans des langages de programmation tels que C, Java, Visual Basic, etc…[1]

Chacun des langages cités a sa propre syntaxe et ses propres règles. Le point commun entre deux programmes écrits dans deux langages différents est la forme initiale de conception qui est la forme algorithmique. Ceci nous amène à la question qui s'est toujours posée:

Pourquoi apprendre l'algorithmique alors que nous pouvons directement programmer?

La réponse à cette question est que l'algorithmique exprime les instructions résolvant un problème donné **indépendamment des particularités de tel ou tel langage,** ce qui permet la concentration sur la résolution du problème à traiter et laisser, ainsi, à coté les difficultés posées par les langages de programmation (problèmes de syntaxe, types d'instructions, etc.).

I/ Définitions

Dans ce qui suit, nous allons donner une définition à un ensemble de notions nécessaires pour la bonne compréhension de la suite du cours.

1. Donnée

Toute entité qui a un sens est une donnée (information[2]). Elle est destinée subir un ou plusieurs traitements.

Exemple :

Les nombres 2 et 3 sont des données. Mohamed et Ali sont aussi des données.

2. Instruction

On appelle une instruction toute manipulation d'une ou de plusieurs données.

Exemple :

A prend pour valeur 2, l'opération qui consiste à mettre la valeur 2 dans la '**variable**' A est l'instruction d'affectation 'prend pour valeur'[3] .

3. Traitement

On appelle un traitement une ou plusieurs instructions successives.

[1] Chaque langage a une utilité propre selon l'usage que l'on veut en faire.
[2] Ici nous ne sommes pas encore au stade de faire la différence entre une donnée et une information.
[3] L'expression 'Prend pour valeur' sera remplacée dans la suite du cours par l'expression 'ppv'.

Exemple :

A ppv 2

Ecrire (A)

Ce traitement permet d'affecter la valeur 2 à la variable A puis l'affichage du contenu de cette dernière.

4. Langage de programmation

Un langage de programmation est un langage destiné à décrire l'ensemble des instructions consécutives qu'un ordinateur doit exécuter. Un langage de programmation informatique est ainsi une façon pratique pour nous permettre de donner des instructions à l'ordinateur.

5. Algorithme

Un algorithme est une suite finie d'instructions (ou encore actions) écrites dans le but de résoudre un problème donné et ce indépendamment des particularités de tel ou tel langage et qui, une fois exécuté correctement, conduit à un résultat donné**.** Le langage algorithmique a été conçu depuis 1958, et le mot algorithme vient du nom du mathématicien musulman ALKHAWARIZMI.

Un algorithme doit posséder les qualités suivantes :

La clarté ou la lisibilité: lecture facile des instructions;

La structuration : entête, déclaration, traitement et présentation des résultats ;

L'optimisation : chercher la solution la moins coûteuse en temps et en mémoire (solution optimale);

La documentation : utiliser des commentaires pour faciliter sa compréhension.

Nous allons revenir plus loin en détail sur la notion algorithme.

6. Programme

Le langage dans lequel est rédigé un algorithme n'est pas compréhensible par un ordinateur, nous devons donc le traduire dans un langage qui lui est connu. Le résultat de cette traduction s'appelle «programme».

7. Résolution de problèmes

A partir de l'énoncé en langage naturel du problème à résoudre, nous devons parcourir quatre grandes étapes pour parvenir à sa résolution algorithmique.

- **Analyse :** Rassembler le maximum d'informations sur le problème en question et savoir ce qui est donné et ce qui est demandé. Décortiquer ensuite le problème en sous problèmes si nécessaire ;

- **Etablissement de la solution :** Expression de la solution sous une forme algorithmique en ce limitant dans ce qui est demandé ;
- **Déroulement de la solution (algorithme)**: Cette étape contient la vérification logique de l'algorithme en utilisant des exemples par l'affectation des valeurs en entrées et la vérification des valeurs en sorties ;
- **Traduction :** Traduction de la solution algorithmique en une solution dans un langage de programmation.

Exemple:

Soit à résoudre l'équation du second degré $AX^2+BX+C=0$ avec A, B, C, X des entiers et A<>0. En appliquant les quatre étapes de la résolution d'un problème nous aurons :

- **Analyse** : L'équation à résoudre est une équation de second degré. Sa résolution revient à trouver les solutions des sous problèmes suivant :

  Fixer les valeurs de A, B et C
  Calculer Delta qui est égale à B^2- 4*A*C
  Discussion de l'existence des racines selon les valeurs de Delta
    Si Delta ≥ 0 Alors il existe deux racines R1 et R2
    Sinon il n'existe pas de racine dans l'espace des réels
  Calculer les racines dans le cas où elles existent
    R1 = (-B- racine(Delta))/2*A
    R2 = (-B+ racine(Delta))/2*A
    Affichage de la solution trouvée

- **Solution sous la forme algorithmique :** La solution dégagée par la phase d'analyse peut être traduite sous la forme exprimée dans le plan suivant qui récapitule les différents phases de la résolution du problème :

```
Algorithme Solution_Equation_Second_Degré
    A, B, C, Delta sont des entiers
    R1, R2 sont des réels
    Lire (A,B,C)
    Delta ppv B^2- 4*A*C
    Si (delta < 0) Alors
        Ecrire ('Pas de solution dans l'espace réel')
    Sinon
```

R1 ppv (-B+ racine (Delta))/2*A

R2 ppv (-B- racine (Delta))/2*A

Ecrire (R1,R2)

FinSi

Fin

- **Déroulement de l'algorithme sur des exemples :**

Exemple 1 :

Lire (A B,C) : A=4, B=6, C=5.

Delta ppv B^2- 4*A*C : Delta = 36 – 80 = -44.

Delta < 0 Donc Pas de solutions dans l'espace réel.

Exemple 2 :

Lire (A,B,C) : A=1, B=6, C=5.

Delta ppv B^2 - 4*A*C : Delta = 36 – 20 = 16.

Delta > 0 Donc R1 = -1 et R2 = -5.

Exemple 3 :

Lire (A,B,C) : A=9, B=6, C=1.

Delta ppv B^2-4*A*C : Delta = 36 – 36 = 0.

Delta = 0 Donc R1 = R2 = 0,33 (Racine double)[1].

- **Reste maintenant à traduire cet algorithme dans l'un des langages connus.**

Remarque

Plusieurs concepts ont été utilisés dans l'exemple précédant, tels que les variables et les deux instructions Lire () et Ecrire (), ces concepts seront présentés en détail plus loin.

II/ Concepts algorithmiques fondamentaux

1. Structure générale d'un algorithme

Nous avons vu plus haut qu'est ce qu'un algorithme, voyons maintenant sa structure générale (ou syntaxe). Cette structure se présente globalement comme suit :

Algorithme <Nom_Algorithme>

<Partie_Déclarations>

Début

[1] Ce cas peut être traité à part dans l'algorithme.

<Partie_Actions>

Fin

Remarque :

Certains auteurs préfèrent inclure la partie déclaration dans le corps de l'algorithme c.-à-d. entre Début et Fin. D'autres insèrent, après chaque expression (de déclaration ou d'action), un point virgule «;» pour bien distinguer ces dernières entre elles.

1.1. Entête de l'algorithme : Permet simplement d'identifier un algorithme.

Exemple :

Algorithme Calcul_Moyenne

1.2. Partie Déclarations : Contient les expressions de désignation des données du problème à traiter.

Exemple :

A et B sont des entiers

C et D sont des réels

1.3. Partie Actions (corps de l'algorithme): Comporte les instructions (ou expressions d'actions) destinées à décrire les traitements à appliquer aux données pour aboutir à des résultats. Généralement nous retrouvons trois types d'actions : préparation des données, traitement proprement dit et enfin présentation des résultats.

Exemple :

Obtention des valeurs de A et de B	*Préparation (acquisition) des données pour le traitement*
C ppv A / B D ppv C * B	*Traitement proprement dit*
Ecrire (C,D)	*Présentation (affichage) des résultats*

1.4. Commentaires : Les commentaires permettent de documenter l'algorithme et faciliter sa compréhension, les commentaires sont vivement conseillés. Les caractères ***C et FC*** *mentionnent* le début et la fin d'un commentaire.

Exemple :

Algorithme Calcul_Moyenne ***C*** *Algorithme qui calcule la moyenne* ***FC***

Remarque :

Dans la littérature d'autres caractères peuvent être retrouvés pour marquer les commentaires. Nous citons, à titre d'exemple, le double slash «//» pour écrire un commentaire sur une seule ligne et le slash et l'astérisque « /* ...*/ » pour un commentaire multi-lignes.

1.5. Exemple récapitulatif

L'exemple suivant récapitule les concepts vus dans l'algorithme Calcul :

Algorithme Calcul

C Partie de déclarations FC

A et B sont des entiers

C et D sont des réels

Début

C Partie d'actions FC

A ppv 3

B ppv 4

C ppv A / B

D ppv C * B

Ecrire (C,D)

Fin

2. Trace d'un algorithme

La trace d'un algorithme (déroulement) permet de suivre pas à pas l'exécution de celui-ci. On numérote les actions de l'algorithme et, dans un tableau, on suit l'évolution des variables[1] intéressantes.

Exemple :

Soit l'algorithme suivant :

Algorithme Trace

A et B sont des entiers

Début

A ppv 4

B ppv 11

A ppv B − A

B ppv B−A

A ppv A + B

Fin

→

Instructions	A	B
1	4	
2		11
3	7	
4		4
5	11	

[1] Cette notion sera sujette du deuxième chapitre.

III/ Les variables

Au cours de son évolution (exécution) l'algorithme (programme) a besoin de stocker les données avec lesquels il travaille, il lui faut donc des boites ou des endroits dans lesquels il emmagasine ses données. Ces endroits ou ces boites sont appelées variables. Les variables possèdent plusieurs propriétés et obéissent à plusieurs règles que nous allons voir ci-après.

1. Le concept Variable

1.1. Définition, une variable c'est quoi ?

Une variable est un emplacement mémoire capable de stocker une donnée (une valeur). Le nom de cette variable obéit à des impératifs changeant selon les langages, mais en général ce nom peut contenir des lettres, des chiffres et le caractère souligné (_). Le nom des variables doit toujours commencer par une lettre.

Exemple :

A et B sont des entiers, A et B peuvent, donc à tout moment, prendre des valeurs entières.

1.2. Type de variables

Avant de pouvoir utiliser les variables dans un algorithme, on doit définir leur type. Ce type est appelé **type de données**. Un type de données est un ensemble d'objets sur lesquels on peut appliquer des opérations bien déterminées.

Exemple :

L'ensemble des nombres entiers est le type entier. Les opérations qui lui sont associées sont les opérations arithmétiques.

En algorithmique, on distingue plusieurs types de données:

a. Les Types standards

Ce sont des types prédéfinis c.-à-d. qu'ils sont reconnaissables par la plus part des langages de programmation. Ces types sont :

- **Le type numérique**

Les variables de type numérique utilisées en algorithmique ont comme domaines usuels : le réel (**Réel**) ou l'entier (**Entier**). Les opérateurs arithmétiques utilisables dans ces domaines sont : l'addition (+), la soustraction (-), le produit (*), la division (/) et la puissance (^). Les opérateurs : division entière (div) et reste de la division entière (mod) concernent les entiers. On peut aussi appliquer sur les éléments de type entier ou réel, les opérateurs de comparaison classique : >, <, ≠, =, ≥ et ≤.

Exemple :

La variable entière Age peut prendre la valeur 20.

La variable réelle Note peut prendre la valeur 16,50.

- **Type Caractère et Chaine de caractères**

Il s'agit d'un domaine constitué des caractères alphabétiques, numériques, de ponctuation et les autres symboles &, #, …etc[1]. Les opérations élémentaires pour les éléments de type caractère (**Caractère**) ou chaine de caractères (**Chaine de caractères**) sont les opérations de comparaison : >, <, ≠, =, ≥, ≤ et le + pour la concaténation.

Exemple :

La variable Lettre de type Caractère peut comporter la valeur 'K'.

La variable Nom de type Chaine de caractère peut comporter la valeur 'Hamed'.

La variable Prénom de type Chaine de caractère peut comporter la valeur 'Abdel wahab'.

- **Type Booléen (logique)**

Le domaine des booléens (**Booléen**) est l'ensemble formé des deux seules valeurs {vrai, faux}. Les opérations admissibles sur les éléments de ce domaine sont réalisées à l'aide de tous les connecteurs logiques notés : Et (Et logique), Ou (Ou logique), Ouex (Ou exclusif) et Non (négation logique), ainsi que les deux opérateurs de comparaison =, ≠.

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

De plus, les opérateurs >, <, ≠, =, ≥, ≤ appliqués sur des opérandes de type Entier, Réel, Caractère ou Chaine de caractères produisent comme résultat une valeur booléenne.

Exemple :

La variable Admis est de type Booléen prend la valeur Vrai si l'étudiant est admis, elle prend la valeur Faux sinon.

La variable qui qualifie le résultat de la comparaison 5 > 2 prend la valeur Vrai, tandis que celle qui qualifie la comparaison 0 < -1 prend la valeur Faux.

[1] Les valeurs des deux types Caractères et Chaines de caractères sont mises entre deux apostrophes (**').**

Remarque :

Pour des raisons d'optimisation des ressources de la machine, on retrouve dans certains langages de programmation des types tels que Byte, Entier simple et Entier long pour représenter l'ensemble des entiers. On retrouve également les types Réel simple et Réel double pour représenter l'ensemble des réels.

b. Le type énuméré

Les valeurs de ce type sont ordonnées et citées explicitement. L'expression générale pour représenter est la suivante :

Type <Nom_Type> = (V1, V2, ……, Vn)

Exemple :

Type Jour = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)

Type Mois = (Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre, Octobre, Novembre, Décembre)

c. Le type intervalle

Les valeurs de ce type sont finies est comprises entre une borne inférieure et une borne supérieure. Ce type est exprimé généralement comme suit :

Type <Nom_Type> =<Inf> .. <Sup>

Exemple :

Type Jours_Mois = 1..31

Remarque :

Les opérations applicables aux valeurs de type énumérés et intervalles sont : $>$, $<$, $\neq$, $=$, $\geq$, $\leq$. De plus, on peut utiliser les fonctions suivantes :

Fonction	***Description***	***Exemple***
Succ(x)	*Donne la valeur qui suit x (successeur)*	*Succ(Lundi) = Mardi*
Pred(x)	*Donne la valeur qui précède x (prédécesseur)*	*Pred(Lundi) = Dimanche*
Ord(x)	*Donne l'ordre de x parmi les valeurs du type.*	*Ord(Lundi) = 2*

2. Les constantes, des variables à valeurs figées

Une constante est une valeur à laquelle on donne un nom, c'est une variable à valeur figée. Elle ne peut donc pas être changée par des actions.

Exemple :

Pi peut désigner la valeur réelle « **3.14** », **Un** peut désigner la valeur entière « **1** ».

Travaux dirigés 1

Exercice 1 :

1. Peut on faire de l'algorithmique sans utiliser un ordinateur ?
2. Quel est le type d'une variable qui va contenir un email comme 'Licence@mail.com' ?
3. Quelle est la différence entre les deux valeurs 20 et '20'**.**
4. Quelle est la différence entre une variable et une constante ?

Exercice 2 :

Soit les deux instructions Lire () et Ecrire ().

- L'instruction lire permet de stoker une valeur saisie dans une variable. Elle a la syntaxe suivante : Lire (x); où x est une variable.
- L'instruction Ecrire permet de visualiser une valeur stockée dans une variable. Elle a la syntaxe suivante : Ecrire (x); où x est une variable.

Questions :

1. Ecrire un algorithme qui calcule la somme et la soustraction de deux nombres entiers, cet algorithme doit à la fin afficher les résultats demandés.

2. Ecrire un algorithme qui donne l'ordre, le successeur et le prédécesseur de la valeur d'une variable dans le type jour vu dans le cours.

Exercice 3 :

Quelles seront les valeurs des variables A, B et C après exécution des deux algorithmes suivants?

Algorithme Calcul_1

A, B sont des Entier

Début

A ppv 1

B ppv A + 3

A ppv 3

Fin

Algorithme Calcul_2

A, B, C sont des Entiers

Début

A ppv 5

B ppv 3

C ppv A + B

A ppv 2

C ppv B – A

Fin

Exercice 4 :

1. Quelles seront les valeurs des variables A et B après l'exécution de l'algorithme Calcul_3 ?

Algorithme Calcul_3

A, B sont des Entiers

Début

A ppv 5

B ppv 2

A ppv B

B ppv A

Fin

2. Les deux dernières instructions permettent-elles d'échanger les deux valeurs de A et B ? Pourquoi ?

3. Si on inverse les deux dernières instructions, cela change-t-il quelque chose ?

4. Ecrire un algorithme permettant d'échanger les valeurs des deux variables A et B, et ce quel que soit leurs contenus préalables.

Exercice 5 :

Que produit l'algorithme suivant ?

Algorithme Calcul_4

A, B, C sont des chaines de caractères

Début

A ppv '423'

B ppv '12'

C ppv A + B

Fin

Remarque :

Dans les quatre derniers exercices, utiliser la trace de l'algorithme pour vérifier les résultats.

Chapitre II

Déclaration et manipulation des données

Introduction, apprendre à la machine comment travailler…

Dans le présent chapitre nous allons voir la description détaillée des syntaxes des deux parties de l'algorithme vues dans le premier chapitre.

I/ Déclaration des données (Partie Déclarations)

Toute ressource (constante, type, variable, …) doit être déclarée avant son utilisation dans un algorithme. Dans ce qui suit nous allons donner la syntaxe des expressions de déclarations dans l'ordre de leurs apparitions dans l'algorithme.

1. Déclaration des constantes

La déclaration des constantes se fait selon la syntaxe suivante :

Syntaxe :

Const < Nom_Constante> **Fixée à** <Valeur>

Exemple :

Const Douzaine **Fixée à** 12

Const Matière **Fixée à** 'Algorithmique'

2. Déclaration des types

Les types non standards peuvent être déclarés comme déjà vu dans le chapitre I, ainsi nous aurons la syntaxe suivante :

Syntaxe :

Type <Nom_Type> = <Définition>

3. Déclaration des variables

La déclaration des variables se fait selon la syntaxe suivante :

Syntaxe :

Var <Nom_Variable> : <Nom_Type>

Exemple :

Var A,B : **Entier**

Nom : **Chaine de caractères**

Jr : Jour

4. Structure générale détaillée d'un algorithme

La structure générale d'algorithme vue dans le premier chapitre devient :

Syntaxe :

Algorithme <Nom_Algorithme>

C** Partie Déclarations **FC

Const ……..... ***C** Déclaration des constantes **FC***

Type ……. ***C** Déclaration des types de données non standards **FC***

Var …….. ***C** Déclaration des variables **FC***

Début

C** Partie Actions **FC

<Action_1>

<Action_2>

…

<Action_N>

Fin

Exemple :

Algorithme Exemple

Const Matière **Fixée à** 'Algorithmique'

Type Jour = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)

Mois = (Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre, Octobre, Novembre, Décembre)

Var i,j : **Entier**

Jr : Jour

M : Mois

Début

<Partie_Actions>

Fin

Remarque :

D'autres déclarations (comme la déclaration des procédures) peuvent êtres retrouvées dans la partie déclarations.

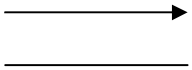
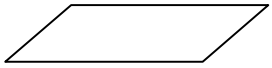
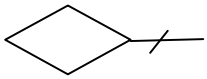
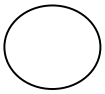
II/ Manipulation des données (Partie Actions)

Cette partie comporte les différentes formes d'expressions d'actions ou instructions qui constituées le traitement dans l'algorithme. Ces formes peuvent être organisées selon les trois familles

de structures algorithmiques suivantes :

- Structures séquentielles ou linéaires (lecture/écriture, affectation, etc.) ;
- Structures conditionnelles ou alternatives (Si …Alors, Si… Alors… Sinon, etc.);
- Structures itératives ou répétitive (Tant que … Faire, etc.).

Pour la bonne compréhension du fonctionnement de ces expressions nous allons utiliser, en plus des syntaxes, des logigrammes[1]. Ces logigrammes disposent de plusieurs symboles normalisés représentés ci-dessous[2].

	Représente le début, la fin ou l'interruption de l'exécution de l'algorithme.
	Représentent la liaison et l'enchainement de l'exécution des instructions.
	Représente la lecture (entrée) ou l'écriture (sortie) des données.
	Représente le traitement.
	Représente le test d'une condition (Vrai, Faux).
	Connecteur utilisé à la fin et en début de ligne pour en assurer la continuité.

1. Expressions de Lecture/Ecriture ou Entrée/Sortie

1.1. Expression de lecture (entrée ou saisie)

L'instruction **Lire (…)** permet de stoker des valeurs saisies par l'utilisateur dans des variables.

Syntaxe :

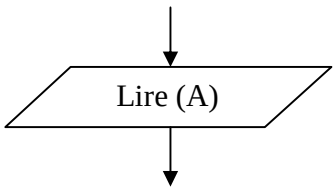
Lire (Var1,Var2,…..,VarN)

Exemple :

Lire (A)

Lire (B,C)

Lire (Nom)



[1] Dans certains ouvrages les logigrammes sont aussi appelés organigrammes de programmation, algorigrammes, ou encore ordinogrammes.

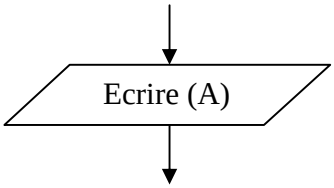
[2] D'autres symboles peuvent êtres retrouvés dans la littérature.

1.2. Expression d'écriture (sortie ou affichage)

L'instruction **Ecrire (…)** permet d'afficher le contenu des variables.

Syntaxe :

Ecrire (Var1,Var2,…..,VarN)



Exemple :

Ecrire (A)

Ecrire (B,C)

Ecrire (Nom)

Remarque :

L'instruction d'écriture peut aussi afficher des messages.

Exemple :

Ecrire ('Bonjour')

Ecrire ('Veuillez saisir le nom')

2. Expression d'affectation

2.1. Les opérateurs

En algorithmique, les opérateurs permettent de rattacher les variables dans des expressions qui seront évaluer lors de l'exécution de l'algorithme. Ces opérateurs sont regroupés dans trois familles : opérateurs arithmétiques, opérateurs logiques, et opérateurs relationnels (de comparaison).

Famille	Opérateur	Signification
Opérateurs arithmétiques	+	Addition ou Unaire (exemple : +2)
	-	Soustraction ou Unaire (exemple : -2)
	*	Multiplication
	/	Division
	^	Puissance
	Div	Division entière (donne la partie entière de la division)
	Mod	Modulo ou reste de division entière
Opérateurs logiques	Et	Et logique
	Ou	Ou logique
	Non	Non logique (unaire)
Opérateurs relationnels	=	Egal
	≠	Différent
	>	Supérieur
	<	Inferieur
	≥	Supérieur ou égal
	≤	Inférieur ou égal

2.2. Types d'expressions

Deux types d'expressions sont à distinguer :

- Les expressions arithmétiques sont constituées d'opérandes numériques reliés par des opérateurs arithmétiques.

Exemple :

(A+2)/3 où A est une variable numérique.

- Les expressions booléennes (logiques) sont des expressions dont le résultat est de type booléen. Elles peuvent comporter des opérateurs arithmétiques, des opérateurs booléens et des opérateurs relation.

Exemple :

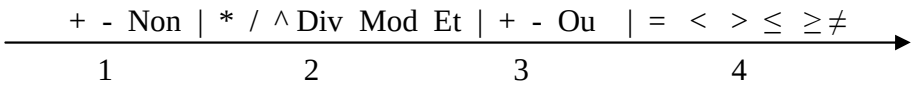
((A < 5) **Et Non** Fin_Calcul) où A est une variable numérique et Fin_Calcul est une variable booléenne.

Remarque :

Les expressions qui comportent des opérations sur les chaines de caractères (comme l'opération de concaténation avec l'opérateur « + »), peuvent être considérées comme un type à part. Une expression booléenne peut comporter une expression de ce type.

2.3. Règles d'évaluation des expressions

Il existe une certaine priorité à respecter entre les opérateurs cités plus haut :

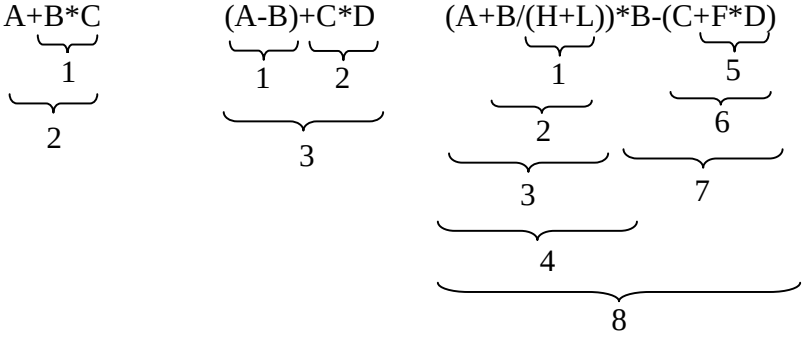


Pour évaluer une expression on procède comme suit :

- Appliquer les opérateur selon les priorités 1, 2, 3 et 4. Dans le cas des opérateurs de même priorité, on commence par le plus à gauche.
- Si l'expression comporte des parenthèses, on évalue d'abord leurs contenus en commençant par les parenthèses les plus internes.

Exemple :

Les expressions A+B*C, (A-B)+C*D et (A+B/(H+L))*B-(C+F*D) sont évaluées comme suit :

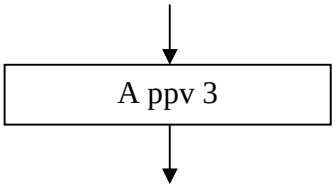


2.4. Expression d'affectation

Une expression d'affectation est une instruction qui permet de donner (affecter) une valeur ou un résultat de l'évaluation d'une autre expression à une variable.

Syntaxe :

<Nom_Variable> **ppv** <Expression>



Exemple :

A **ppv** 3

B **ppv** (A+2)/3

Nom **ppv** 'Mohamed'

3. Structures de contrôle

Les structures de contrôle visent à modifier le chemin d'exécution des instructions de l'algorithme, habituellement séquentiel, dans le but de couvrir toutes les solutions possibles du problème à traiter.

3.1. Structures de contrôle conditionnelles ou alternatives

Ces structures sont utilisées lorsque l'exécution d'une ou de plusieurs actions est conditionnée. Nous avons deux types de structures conditionnelles :

a. Structure conditionnelle simple ou à une branche

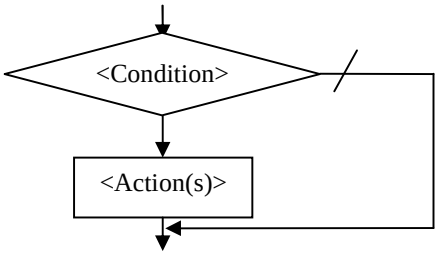
Une structure conditionnelle simple ne permet l'exécution d'une ou de plusieurs actions que si une condition donnée est vérifiée.

Syntaxe :

Si <Condition> **Alors**

<Action(s)>

FinSi



Où <Condition> est une expression Booléenne et <Action(s)> est une expression d'action, qui n'est exécutée que si <Condition> est vérifiée (vraie).

Exemple :

Si (Moyenne_Semestrielle ≥ 10) **Alors**

Ecrire ('Semestre acquis')

FinSi

Ici « Moyenne_Semestrielle ≥ 10 » représente la condition, «Ecrire ('Semestre acquis')» représente l'action et elle n'est exécutée que si la condition est vérifiée c.-à-d., dans notre exemple, la moyenne semestrielle est supérieure ou égale à 10.

b. Structure conditionnelle alternée ou à deux branches

La structure conditionnelle alternée est une structure conditionnelle simple avec son alternance dans le cas où la condition est fausse. Sa syntaxe est la suivante :

Syntaxe :

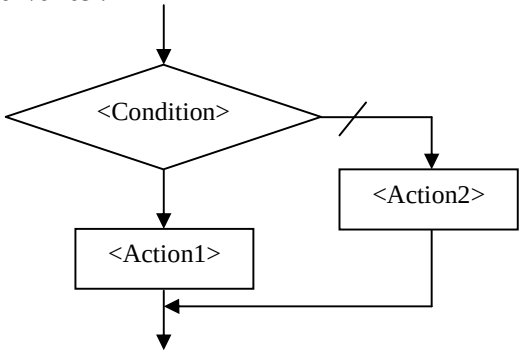
Si <Condition> **Alors**

<Action1>

Sinon

<Action2>

FinSi



Si la condition <Condition> est vraie <Action1> est exécutée, si cette condition est fausse <Action2> est exécutée.

Exemple :

Si (Moyenne_Semestrielle ≥ 10) **Alors**

Ecrire ('Semestre acquis')

Sinon

Ecrire ('Cas à étudier')

FinSi

Remarque :

1. <Action1> et <Action2> peuvent représenter une ou plusieurs actions.

2. Une condition <Condition> peut être constituée de plusieurs conditions élémentaires rattachées par les opérateurs logiques (Et , Ou et Non)

Exemple 1:

Si Non** (Moyenne_semestrielle ≥ 10) **Alors

***Ecrire** ('Semestre acquis')*

FinSi

Exemple 2:

Si** (Moyenne_Semestrielle < 10) **Ou** (Etudiant_Exclu) **Alors

***Ecrire** ('Semestre non acquis')*

FinSi

Etudiant_Exclu est une variable booléenne qui a la valeur vrai si l'étudiant est exclu d'une matière, faux sinon.

c. Structure conditionnelle imbriquée

La structure conditionnelle (simple ou alternative) imbriquée est une structure utilisée lorsqu'on a plus de deux branches possibles.

Syntaxe :

Si <Condition1> **Alors**
    Si <Condition2> **Alors**
        <Action1>
        …
    Sinon
        …
        <Action2>
    FinSi
Sinon
    Si <Condition3> **Alors**
        …
        <Action3>
    Sinon
        …
        <Action4>
    FinSi
FinSi

Remarque :

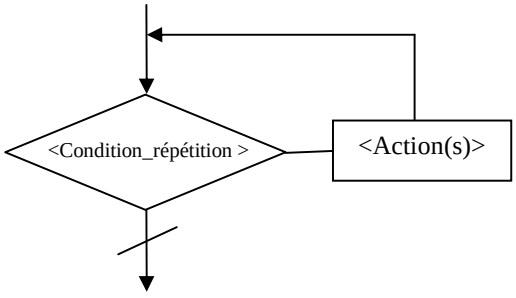
Les trois points alignés «…» expriment la possibilité d'avoir d'autres structures conditionnelles.

3.2. Structure itérative ou répétitive, la boucle « Tant que … Faire »

La structure itérative (ou boucle) « **Tant que … Faire »** permet la répétition d'une ou de plusieurs actions tant que la condition de répétition est vérifiée. Si la condition est fausse dès le départ, le traitement n'est **jamais** exécuté.

Syntaxe :

<Action_amorçage>
Tant que <Condition_répétition> **Faire**
    <Action(s)>



<Action_ relance>

FinTantque

Où <Action_d'amorçage> est l'action d'initialisation de la (des) variable(s) de la condition de répétition,

<Condition_répétition> est une expression booléenne qui est vraie quand l'événement attendu ne s'est pas encore produit, on l'appelle donc, condition de répétition,

<Action(s)> est le traitement à exécuter (répétition) tant que <Condition_répétition> est vraie,

<Action_ relance> sert à la réaffectation de la (des) variable(s) de la condition de répétition.

Exemple :

Somme **ppv** 0
Ecrire ('Donnez une autre valeur ou 0 pour finir.') } <Action_amorçage>
Lire (Valeur)

Tant que (Valeur < > 0) **Faire**

Somme **ppv** Somme + Valeur
Ecrire ('Donnez une autre valeur ou 0 pour finir.') } <Action_ relance>
Lire (Valeur)

FinTantque

Remarque :

Il faut toujours s'assurer que la condition de répétition sera rompue après un nombre fini de répétitions afin de ne pas tomber dans le cas d'une boucle infinie, c.-à-d., que l'action(s) s'exécute indéfiniment.

4. Exemple général

L'algorithme Calcul_Moyenne calcule la moyenne semestrielle de N matières d'un étudiant.

Algorithme Calcul_Moyenne

Const N **Fixée à** 10
Type Coefficient =1..5
Var i, Somme_coef : **Entier**
Coef : Coefficient
Moyenne, Somme_Moyenne: **Réel**
Non_Exclu_Matière : **Booléen**

Début

i **ppv** 1

```
        Non_Exclu_Matière ppv Vrai
        Somme_Moyenne ppv 0
        Somme_coef ppv 0
        Tant que  ((i ≤ N) Et Non_Exclu_Matière)  Faire
            Ecrire ('Etudiant exclu de la matière ',i,' Oui/Non ?')
            Lire (Réponse_Exclu)
            Si (Réponse_Exclu='Non') Alors
                Ecrire ('Donner le coefficient de la matière ',i)
                Lire (Coef)
                C Calcul de la somme des I premiers  coefficients FC
                Somme_coef ppv Somme_coef + Coef
                Ecrire ('Donner la moyenne de la matière ',i)
                Lire (Moyenne)
                C Calcul de la somme des moyennes * aux coefficients des i premières
                matières FC
                Somme_moyenne ppv Somme_moyenne + Coef * Moyenne
                C Passage à la (i +1) ème matière FC
                i ppv i + 1
            Sinon
                Ecrire ('Calcul impossible de la moyenne semestrielle, étudiant exclu')
                Non_Exclu_Matière ppv Faux
            FinSi
        FinTantque
        Si Non_Exclu_Matière Alors
            Ecrire ('La moyenne semestrielle est égale à :')
            Ecrire (Somme_moyenne/Somme_coef)
        FinSi
    Fin
```

Travaux dirigés 2

Exercice 1 :

1. Reprendre l'algorithme de résolution de l'équation du deuxième degré vu dans le premier chapitre du cours en respectant les syntaxes des concepts vus dans le deuxième chapitre.
2. Généraliser cet algorithme pour résoudre N équations où N est un entier naturel ≥ 1.

Exercice 2 :

Soit la fonction f exprimée mathématiquement comme suit :

$$\forall\ x,y \in \mathbb{R},\ f(X,Y) = \begin{cases} X^{\wedge 2}+Y^{\wedge 2} \text{ Si } X \geq 0 \text{ et } Y \geq 0 \\ 1 \text{ Si } X<0 \text{ et } Y<0 \\ X \text{ ailleurs} \end{cases}$$

Donner un algorithme qui calcule cette fonction. Dérouler cet algorithme sur des exemples.

Exercice 3 :

1. Donner un algorithme réalisant la division entière par la soustraction successive d'un entier positif A par un entier positif B.
2. Utiliser cet algorithme pour démontrer que la division par 0 est indéfinie.

Exercice 4 :

Définition : On appelle PGCD (plus grand commun diviseur) de deux entiers A et B positifs, un entier calculé comme suit :

Si A = B alors PGCD(A, B) = A = B
Si A < B alors PGCD(A, B)=PGCD (A, B - A)
Si A > B alors PGCD(A, B)=PGCD(A - B, B).

Donner un algorithme qui réalise cette fonction.

Remarque :

Pour les quatre exercices, représenter les résultats par des logigrammes.

Chapitre III

Les tableaux

Introduction, les tableaux un moyen d'optimisation

Dans ce chapitre nous allons voir un type primitif en algorithmique qui est les Tableaux. Supposons que dans un algorithme, nous ayons besoin simultanément de 12 valeurs (par exemple, des notes pour calculer une moyenne). Evidemment, la seule solution dont nous disposons, jusqu'à ce moment, consiste à déclarer douze variables, appelées par exemple N1, N2,…, N12, destinées à contenir les 12 notes. Ensuite arrivé au calcul, et après une succession de 12 instructions Lire ( ) distinctes, nous serons obligatoirement amenés au calcul suivant:

Moyenne **ppv** (N1+N2+N3+N4+N5+N6+N7+N8+N9+N10+N11+N12)/12

Cela étant pour un traitement comportant 12 variables, avec l'augmentation du volume des traitements le nombre de variables augmente considérablement, ce qui rend les traitements difficilement réalisables. En algorithmique (programmation) le rassemblement de toutes ces variables en une seule, appelée tableau, au sein de laquelle chaque valeur sera désignée par un numéro, permet de résoudre ce problème, cela changerait la désignation de ces variables et donnerait donc quelque chose du genre « la note numéro 1 », « la note numéro 2 », etc.

On appelle, généralement, **Tableau** un ensemble **fini** d'éléments de même type, qui ont le même sens et subissent souvent, les mêmes traitements. Le nombre des éléments d'un tableau définit sa cardinalité, le type de ses éléments définit son type. Un tableau est représenté par un identificateur.

I/ Tableau à une dimension, monodimensionnel ou vecteur

1. Déclaration

Pour déclarer un type ou une variable de type tableau monodimensionnel, il faut fixer sa cardinalité ainsi que le type de ses éléments. La cardinalité peut être exprimée, en général, par un entier ou comme un nombre d'éléments d'un type intervalle ou énuméré.

Syntaxe :

a. Type<Identif_type_tableau> = **Tableau** [<Cardinalité>] **de** <Type_élém_tableau>

Var <Identif_tableau > : <Identif_type_tableau >

ou

b. Var <Identif_tableau > : **Tableau** [<Cardinalité>] **de** <Type_élém_tableau>

Les deux syntaxes de déclaration a et b sont équivalentes.

Exemple:

Type TDouzaine = **Tableau** [12] **de Réel**

Var TNote : TDouzaine

ou

Var TNote : **Tableau** [12] **de Réel**

Remarque :

Dans certains ouvrages la déclaration des tableaux monodimensionnels mentionne la cardinalité minimale et cardinalité maximale. Les syntaxes a et b deviennent alors :

a. Type *<Identif_type_tableau≥* ***Tableau*** *[<Card_Min>.. <Card_Max>]* ***de*** *<Type_élém_tableau>*
Var *<Identif_tableau > : <Identif_type_tableau >*

b. Var *<Identif_tableau > :* ***Tableau*** *[<Card_Min>..<Card_Max>]* ***de*** *<Type_élém_tableau>*

2. Accès aux éléments

Les éléments d'un tableau monodimensionnel, sont rangés les uns derrière les autres. On a donc un premier élément, un deuxième élément, … etc. Un élément peut donc être désigné par le nom du tableau et un numéro qui représente son rang dans ce tableau.

On désigne un élément d'un tableau par l'expression :

<Nom_du_tableau> [<indice>]

où <indice> est une expression dont la valeur donne le rang de l'élément désigné. La valeur de cette expression doit être d'un type ordinal c.-à-d. entier, intervalle ou énuméré.

Pour désigner un élément du tableau, il est donc évident que la valeur de l'indice soit comprise entre sa valeur minimale et sa valeur maximale. Une valeur d'indice, hors de cet intervalle, ne peut alors désigner un élément de ce tableau. Elle entraîne une erreur fatale.

Exemple 1:

Var TNote : **Tableau** [12] **de Réel**

TNote [1] : Désigne le premier élément de TNote (ou borne inférieure)

TNote [12] : Désigne le douzième et le dernier élément de TNote (ou borne supérieure)

TNote [i] : La valeur de la variable i désigne le rang d'un élément de TNote

TNote [i*2+j] : La valeur de l'expression i*2+j désigne le rang d'un élément de TNote

Pour bien comprendre cet exemple il convient de représenter TNote comme suit :

Indice →	1	2	3	4	5	6	7	8	9	10	11	12	
TNote	13	20	9.5	14	16	12	18	15	10	3.5	11.5	2	← Valeurs

TNote [2] égale à 20

Exemple 2:

Type Etat =(Libre, Occupé)

Jour = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)

Var TJour : **Tableau**[Jour] de Etat

TJour est donc un tableau de 7 éléments où TJour[samedi], par exemple, désigne le premier élément de TJour. L'indice est de type énuméré.

Schématiquement TJour se représente comme suit :

Indice →	Samedi	Dimanche	Lundi	Mardi	Mercredi	Jeudi	Vendredi	
TJour	Occupé	Occupé	Libre	Occupé	Occupé	Libre	Libre	← Valeurs

TJour[samedi] égale à Occupé.

II/ Tableaux à deux dimensions ou tableaux bidimensionnels

Soient les déclarations suivantes:

Type Ligne = **Tableau**[20] **de Entier**

Table = **Tableau**[10] **de** Ligne

Var Tab : Table

Tab est donc, un ensemble de 10 lignes, une ligne est un ensemble de 20 entiers. Un élément de Tab est une ligne. C'est donc un tableau. Il peut être désigné par Tab[i], où « i » est le numéro de cette ligne. Un tableau de type ligne est un tableau de 20 entiers, un entier de ce tableau sera donc désigné par Tab[i][j] où « j » est le rang de cet entier dans la ligne « i ». Il convient, donc, de désigner cet entier par Tab[i, j] au lieu de Tab[i][j].

Un élément de Tab étant désigné par deux indices, Tab est appelé tableau à deux dimensions ou tableau bidimensionnel.

Syntaxe :

a. Type<Identif_type_tableau2d > = **Tableau** [<Card1>,<Card2>] **de** <Type_élém_tableau2d>

Var <Identif_tableau2d > : <Identif_type_tableau2d >

ou

b. Var <Identif_tableau2d > : **Tableau** [<Card1>,<Card2>] **de** <Type_élém_tableau2d>

Les deux syntaxes de déclaration dans a et b sont équivalentes.

Exemple:

Var Tab: **Tableau** [2,3] **de Entier**

Lignes ↓	1	2	3	← Colonnes
1	10	20	1	
2	04	12	12	

Tab [1,2] égale à 20.

Remarques :

1. De même que pour un élément d'un tableau à deux dimensions désigné par deux indices, un élément d'un tableau à N dimensions doit être désigné par N indices.

2. Les règles appliquées à la cardinalité d'un tableau monodimensionnel sont les mêmes appliquées aux cardinalités d'un tableau à N dimensions.

3. De même que pour la cardinalité d'un tableau monodimensionnel, dans certains ouvrages, les cardinalités des tableaux bidimensionnels sont écrits comme suit :

***a. Type**<Identif_type_tableau2d > = **Tableau** [<Card1_Min>..<Card1_Max>,<Card2_Min>..<Card2_Max>] **de** <Type_élém_tableau2d>*

***Var** <Identif_tableau2d > : <Identif_type_tableau2d >*

***b. Var** <Identif_tableau2d > : **Tableau** [<Card1_Min>..<Card1_Max>,<Card2_Min>..<Card2_Max>] **de** <Type_élém_tableau2d>*

III/ Schémas généraux de traitement de tableaux

Selon la dimension du tableau en question, les schémas généraux de traitement de ce dernier sont les suivants :

1. Tableau à une dimension

```
Algorithme Traitement_Général_Tab1D
    Var  T : Tableau [<Cardinalité>] de <Type_élém_tableau>
         i : Entier
    Début
        i ppv 1    C Obtention du premier élément du tableau FC
        Tant que (i ≤ <Cardinalité>) Faire
            Traiter T[i]    C Traitement du ième élément du tableau FC
            i ppv i+1       C Obtention du ième +1 élément du tableau FC
        FinTantque
    Fin
```

Exemple :

Remplissage des éléments d'un tableau des notes :

```
Algorithme Remplissage_Notes
    Var  TNote : Tableau [12] de Réel
         i : Entier
    Début
    i ppv 1
    Tant que (i ≤ 12) Faire
         Lire (TNote[i])
         i ppv  i+1
    FinTantque
    Fin
```

2. Tableau à deux dimensions

```
Algorithme Traitement_Général_Tab2D
    Var Tab : Tableau [<Card1>,<Card2>] de <Type_élém_tableau2d>
        i, j  : Entier
    Début
     i ppv 1  C Obtention de la première ligne du tableau  FC
    Tant que (i ≤ <Card1>) Faire
         j ppv 1  C Obtention de la première colonne FC
         Tant que (j ≤ <Card2>) Faire
              Traiter Tab[i, j] C Traitement de l'élément de la ième ligne et jème colonne FC
              j ppv j+1  C Obtention de la  jème +1 colonne FC
         FinTantque
         i ppv  i+1   C Obtention de la ième +1 ligne FC
    FinTantque
    Fin
```

Remarque :

Dans la syntaxe précédente nous avons utilisé deux boucles imbriquées pour traiter tous les éléments du tableau à deux dimensions. Dans un tableau à un N dimensions nous aurons besoins de N boucles pour effectuer ce traitement et donc N-1 imbrications.

Exemple :

Remplissage des éléments d'un tableau des 12 notes de 10 étudiants:

```
Algorithme Remplissage_Notes2
```

```
Var TNote2 : Tableau [40,12] de Réel
    i, j  : Entier
Début
    C Obtention de la première ligne du tableau qui correspond aux notes du premier étudiant FC
    i ppv 1
    Tant que (i ≤ 40) Faire
        C Obtention de la première colonne qui correspond à la première note du ième
        étudiant FC
        j ppv 1
        Tant que (j ≤ 12) Faire
            Lire (TNotes2[i, j])   C Lecture de la  jème note du l'ième étudiant FC
            j ppv j+1   C Passage à la  jème +1 note FC
        FinTantque
        i ppv  i+1   C Passage au ième +1 étudiant FC
    FinTantque
Fin
```

Travaux dirigés 3

Exercice 1 :

Etant donné un tableau T1D de cardinalité N et de type entier :

1. Ecrire un algorithme qui remplit ce tableau avec des éléments entiers.
2. Ecrire un algorithme qui compte le nombre d'éléments positifs, négatifs et nuls.
3. Ecrire un algorithme qui compte le nombre d'occurrences (apparitions) d'un élément X dans le tableau T.
4. Ecrire un algorithme qui cherche l'élément minimal et l'élément maximal dans le tableau T.

Exercice 2 :

Etant donné, un tableau T2D1 à deux dimensions de cardinalité N et M :

1. Ecrire un algorithme qui remplit ce tableau avec des éléments entiers.
2. Ecrire un algorithme qui compte le nombre d'éléments positifs, négatifs et nuls.
3. Ecrire un algorithme qui compte le nombre d'apparitions (occurrences) d'un élément dans le tableau.
4. Ecrire un algorithme qui cherche l'élément minimal et l'élément maximal dans ce tableau.
5. Ecrire un algorithme qui calcule la somme des éléments de ce tableau.
6. Etant donné un autre tableau T2D2 de même dimension et de même type que T2D1, écrire un algorithme qui produit un tableau T2D3 somme des deux tableaux T2D1 et T2D2.

Exercice 3 :

Soit à trier un tableau T de 12 entiers dans l'ordre croissant. Généralement on utilise quatre techniques de tri :

1. Le tri par sélection : Consiste à mettre en bonne position l'élément numéro 1, c.-à-d. le plus petit. Puis mettre en bonne position l'élément suivant. Et ainsi de suite jusqu'au dernier. Ceci se traduit par :

- On commence par le 1er élément du tableau et on le parcourt afin de sélectionner l'indice du plus petit élément.
- On permute le plus petit élément trouvé avec le premier élément du tableau.
- Refaire les deux étapes précédentes dès le deuxième élément jusqu'à l'avant dernier élément du tableau.

2. Le tri à bulles : Consiste à se dire qu'un tableau trié en ordre croissant, est un tableau dans lequel tout élément est plus petit que celui qui le suit. Donc on compare chaque élément du

tableau T avec l'élément qui le suit. Si l'ordre n'est pas bon, on permute ces deux éléments. Et on recommence jusqu'à ce que l'on n'ait plus aucune permutation à effectuer. Les éléments les plus grands « remontent » ainsi peu à peu vers les dernières places, ce qui explique la dénomination de «tri à bulle».

3. Le tri par insertion : Commence par le deuxième élément. Il compare l'élément choisi avec tous ses précédents dans le tableau et l'insert dans sa bonne position. Puis il répète cette opération pour l'élément suivant jusqu'à arriver au dernier.

4. Le tri par comptage : L'algorithme de tri consiste à construire un tableau intermédiaire appelé Indice dans lequel on indique la place de chaque élément du tableau à trier (pour trouver l'emplacement if faut compter le nombre des éléments inférieurs à chaque élément).

Ecrire les algorithmes qui implémentent ces quatre techniques de tri.

Exercice 4 :

Pour rechercher une valeur dans un tableau T de cardinalité N, généralement, on utilise l'une des deux techniques suivantes :

1. Recherche séquentielle ou linéaire : Consiste à comparer la valeur recherchée aux différents éléments du tableau jusqu'à trouver cette valeur ou atteindre la fin du tableau.

2. Recherche dichotomique : Cette technique s'effectue sur des tableaux triés, elle consiste à diviser chaque fois en deux l'intervalle de recherche (deux sous tableaux) et à rechercher la valeur souhaitée dans la moitié de l'intervalle dans laquelle elle peut être incluse. Pour cela, on procède comme suit :

Soient BorneInf et BorneSup les deux extrémités de l'intervalle de recherche, X la valeur recherchée et T le tableau dans lequel se fait la recherche. On calcule le milieu de cet intervalle Milieu = (BorneInf + BorneSup) Div 2. Trois cas sont alors à traiter :

- X = T[Milieu]: La valeur X est trouvée et la recherche est terminée,
- X < T[Milieu]: La valeur X, si elle existe, se trouve dans l'intervalle [BorneInf , Milieu-1],
- X > T[Milieu]: La valeur X, si elle existe, se trouve dans l'intervalle [Milieu+1,BorneSup].

La recherche dichotomique consiste à itérer ce processus jusqu'à ce que la valeur X et retrouvée ou que l'intervalle de recherche soit vide. Au départ de la recherche BorneInf = 1 et BorneSup = N.

Ecrire deux algorithmes qui implémentent ces deux techniques de recherche pour N = 12.

Chapitre IV

Compléments d'expressions

Introduction, un peu plus d'expressions

Nous avons vu jusqu'ici, un ensemble d'expressions de déclarations et d'actions pour concevoir un algorithme. Nous allant poursuivre avec d'autres expressions constituant un raffinement des précédentes. Les expressions de déclaration déjà données sont applicables aux types simples. Elles seront généralisées aux types composés.

I/ Expressions d'actions

A partir des expressions algorithmiques de base, nous déduirons d'autres expressions plus adaptées à des solutions particulières.

1. La structure conditionnelle « Choix … Dans »

Cette structure permet le choix des actions à effectuer selon la valeur d'une expression donnée.

Syntaxe :

Choix <Expression> **Dans**

Début

<v1> : <Action1>

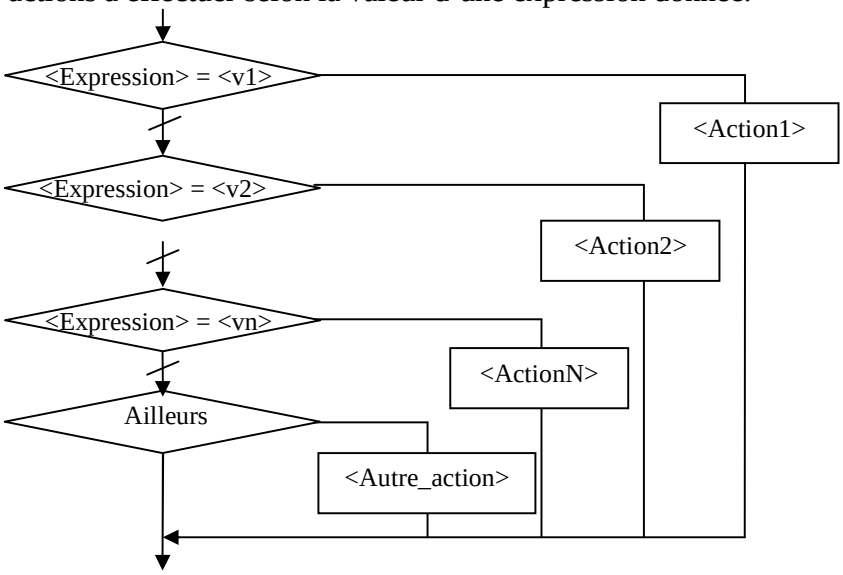
<v2> : <Action2>

….

<vn> : <ActionN>

Ailleurs : <Autre_ action>

FinChoix



Exemple :

Reprenons l'algorithme de la résolution d'une équation de second ordre, en utilisant l'expression **« Choix … Dans »** on aura :

Choix Delta **Dans**

Début

> 0 : <Traitement_calcul_deux_racines>

= 0 : <Traitement_calcul_racine_double>

< 0 : <Traitement_pas_de_solution>

FinChoix

2. La structure répétitive avec compteur « Pour … Faire »

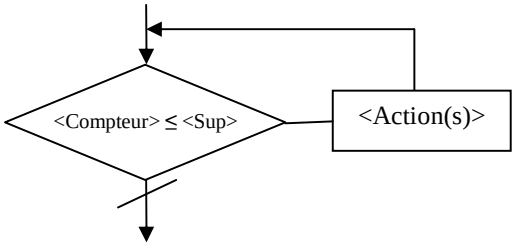
Cette structure permet de répéter une ou plusieurs actions tant que la valeur d'une variable entière appelée <Compteur > appartient à un intervalle fermé [<Inf>, <Sup>].

Syntaxe :

Pour <Compteur> **de** <Inf> **à** <Sup> **Pas** <P> **Faire**

<Action(s)>

FinPour



où : <Compteur> est un entier qui varie d'une valeur inférieure (valeur de départ) <Inf> vers une valeur supérieure (valeur d'arrêt) <Sup> avec une valeur d'incrémentation (Pas**)** qui est égale à <p>.

Exemple :

Soit un tableau T de cardinalité 10. L'instruction suivante permet de remplir le tableau T :

Pour i **de** 1 **à** 10 **Faire**

Lire (T[i])

FinPour

Remarque :

Si la valeur du pas <P> =1, on peut ne pas le mentionner, ce qui est le cas de l'exemple précèdent.

3. L'expression répétitive s'exécutant au moins une fois

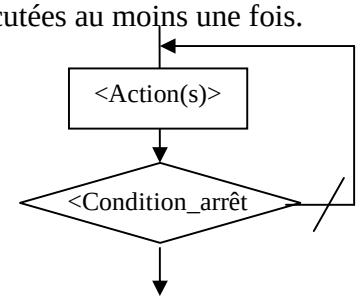
Dans une structure répétitive de type « **Tant que … Faire »**, on teste d'abord la condition de répétition et on exécute ensuite les actions de la structure si cette condition est vraie. Dans plusieurs cas, on désire avoir la possibilité d'exécuter d'abord les actions d'une structure répétitive et tester ensuite la condition de répétition. Les actions d'une telle structure sont exécutées au moins une fois.

Syntaxe :

Répéter

<Action(s)>

Jusqu'à <Condition_arrêt>



<Action(s)> est exécutée jusqu'à ce que <Condition_arrêt> soit vraie.

Exemple :

Répéter

Lire (i)

Jusqu'à i=3

II/ Expressions de déclaration

Jusqu'ici, nous avons étudié des types simples : entier, caractère, tableau, etc. Un élément de l'un de ces types contient une seule information appelée valeur. En pratique cette notion est insuffisante

pour le traitement de certains problèmes. En effet, nous avons surtout besoin d'avoir des types dont un élément est composé de plusieurs valeurs de types différents. On dit alors que ces types sont composés.

1. Déclaration des types composés

La méthode la plus générale pour obtenir des types composés est de regrouper des éléments de types arbitraires, éventuellement composés, par une expression de composition. Cette expression de composition a la syntaxe suivante :

Syntaxe :

Type <Nom_de_type> = **Ensvar** ***C*** *ensemble de variables* ***FC***

<Déclaration_de_variables>

FinEnsvar

Exemple :

Type Personne = **Ensvar**

Nom : **Chaine de caractères**

Prénoms : **Chaine de caractères**

Adresse : **Chaine de caractères**

Date_Nais : **Date**

FinEnsvar

Remarques:

1. Dans certains ouvrages le terme Ensvar est remplacé par le terme Enregistrement.

2. Les variables déclarées à l'intérieur d'un ensemble de variables sont aussi appelées champs.

2. Accès aux éléments d'un type composé

Un élément d'un type composé contient plusieurs informations, l'accès à l'une d'elles se fait en utilisant une fonction d'accès appelée **Selecteur** qui a le formalisme suivant:

<Nom_de_variable_composée>**.**<Nom_du_composant>

Exemple1 :

Si la variable p est une variable du type Personne on a :

p**.**Nom : désigne le composant nom de la variable p.

p**.**Prénom : désigne le composant prénom de la variable p.

p**.**Date_Nais : désigne le composant Date_Nais de la variable p.

Exemple 2 :

L'exemple suivant rempli un tableau appelé Etud de cardinalité 100 avec les informations des étudiants :

```
Algorithme Informations_Etudiants
    Type Jours_Mois = 1..31
        Mois = (Janvier, Février, Mars, Avril, Mai, Juin, Juillet, Aout, Septembre,
                Octobre, Novembre, Décembre)
        SexeE  = (M,F)
        Date = Ensvar
                J : Jour _Mois
                M : Mois
                Année : Entier
            FinEnsvar
        Information  = Ensvar
                        Nom, Prénoms : Chaines de caractères
                        Date_nais   : Date
                        Sexe        : SexeE
                        Adresse     : Chaines de caractères
                    FinEnsvar
        Etudiant = Tableau [100] de Information
      Var Etud : Etudiant
          i : Entier
Début
    i ppv 1
    Tant que (i ≤ 100) Faire
        Ecrire ('Donnez le nom de l'étudiant  ')
        Ecrire (i)
        Lire (Etud[i].Nom)
        Ecrire ('Donnez le prénom de l'étudiant ')
        Ecrire (i)
        Lire (Etud[i].Prénom)
        Ecrire ('Donnez le jour de naissance de l'étudiant ')
        Ecrire (i)
        Lire (Etud[i].Date_nais.J)
```

Ecrire ('Donnez le mois de naissance de l'étudiant ')

Ecrire (i)

Lire (Etud[i].Date_nais.M)

Ecrire ('Donnez l'année de naissance de l'étudiant ')

Ecrire (i)

Lire (Etud[i].Date_nais.Année)

Ecrire ('Donnez le sexe de l'étudiant ')

Ecrire (i)

Lire (Etud[i].Sexe)

Ecrire ('Donnez l'adresse de l'étudiant ')

Ecrire (i)

Lire (Etud[i].Adresse)

i **ppv** i +1

FinTantque

Fin

Travaux dirigés 4

Exercice 1 :

Dresser un tableau qui compare les caractéristiques des structures répétitives vues dans le cours.

Exercice 2 :

Reprendre les questions 1, 2 et 3 de l'exercice 2 de la série de TD 3 en utilisant la structure répétitive « **Pour … Faire**» puis la structure «**Répéter … Jusqu'à**».

Exercice 3 :

Un algorithme pose une question à laquelle l'utilisateur doit répondre par O (Oui) ou N (Non) (exemple : 'Voulez vous sortir ?'). Il se peut que l'utilisateur tape autre chose que la réponse attendue. Dès lors, l'algorithme peut planter.

1. Écrire un algorithme (qui repose la même question jusqu'à l'une des deux réponses attendues soit saisie au clavier par l'utilisateur. Utiliser la structure «**Tant que … Faire**» puis la structure «**Répéter … Jusqu'à**».
2. A votre avis quelle est la structure répétitive qui convienne le mieux pour ce genre de problème ? Justifier ?
3. Dire pourquoi la structure «**Pour … Faire**» ne convient pas à ce genre de problème ?

Exercice 4 :

Soit le menu suivant :

1 : Exécuter traitement 1
2 : Exécuter traitement 2
3 : Exécuter traitement 3
0 : Quitter"

1. En utilisant la structure «**Si … Alors**» comme structure de choix, écrire un algorithme qui contenu à afficher ce menu jusqu'à ce que l'utilisateur entre l'une des valeurs affichées (1,2,3,0). Une fois la valeur entrée l'algorithme doit afficher :

   'Le traitement 1 va être exécuté' pour la valeur 1
   'Le traitement 2 va être exécuté' pour la valeur 2
   'Le traitement 3 va être exécuté' pour la valeur 3
   'Vous voulez sortir ?O/N '

2. Réecrire l'algorithme en utilisant la structure **«Choix** … **Dans»** à la place de la structure «**Si … Alors**».

Exercice 5 :

On désir garder les informations relatives aux salles de classes d'une école. Ces informations sont : la désignation, le type (cours, TD ou TP), le nombre de place, la localisation et le nom de l'agent responsable.

1. Proposer une structure pour contenir les informations citées.
2. Ecrire un algorithme qui lit les informations relatives à 10 salles de classes dans un tableau appelé Salle.

Chapitre V

Les sous programmes

(Procédures et fonctions)

Introduction, une autre forme de structuration des traitements

Il arrive qu'une application (un grand algorithme) ait besoin de répéter le même traitement plusieurs fois dans des différents endroits au cours de son déroulement. La manière la plus évidente d'implémenter ce genre d'algorithme, est de répéter le code correspondant autant de fois que nécessaire. Cette répétition cause, en général, deux grands inconvénients :

1. Une lourdeur dans la lisibilité de l'algorithme en général, c.-à-d. qu'avec la répétition du code du traitement en question, l'algorithme devient plus grand.

2. Une difficulté dans la maintenance de l'algorithme, car en cas de modification du code en question, il faut traquer toutes ces apparitions pour faire la modification convenablement.

Une solution pour remédier à ces deux inconvénients consiste à séparer le traitement répétitif du corps de l'algorithme principal et à regrouper les instructions qui le composent en un module séparé. Il ne restera alors plus qu'à appeler ce groupe d'instructions (qui n'existe donc désormais qu'en un exemplaire unique) à chaque fois qu'on en a besoin. Ainsi, la lisibilité est assurée ; l'algorithme devient **modulaire**, et il suffit de faire une seule modification au bon endroit, pour que cette modification prenne effet dans la totalité de l'application.

On appelle ces groupes d'instructions, auxquels on a recours, les sous programmes. Deux types de sous programmes se présentent, les **fonctions** et les **procédures** (nous verrons un peu plus loin la différence entre ces deux termes).

Remarques :

*Dans certains ouvrages, un algorithme, solution du problème entier, s'appelle **Procédure principale**. Une procédure principale peut être exprimée comme combinaison de procédures et de fonctions, solutions de sous problèmes.*

I/ Procédure

1. Définition

Une procédure est un ensemble d'instructions regroupées sous un nom, qui réalise un traitement particulier dans un algorithme lorsqu'on l'appelle.

Comme un algorithme, une procédure possède un nom, des variables, des instructions, un début et une fin.

Syntaxe :

Procédure <Nom_Procédure> (Paramètres)

C** Partie Déclarations **FC

Const ……... ***C** Déclaration des constantes **FC***

Type ……. ***C** Déclaration des types **FC***

Var …….. ***C** Déclaration des variables **FC***

Début

C** Partie Actions **FC

<Action1>

<Action2>

…

<ActionN>

Fin

2. Déclaration d'une procédure

La déclaration d'une procédure se fait dans la partie déclaration de l'algorithme propriétaire de cette procédure. L'exécution ou l'appel de cette procédure se fait à l'intérieur de la partie des actions.

Syntaxe :

Algorithme <Nom_Algorithme>

<Instructions_de_Déclaration>

Procédure <Nom_Procédure> (Paramètres)

<Instructions_de_Déclaration_de_la_procédure>

Début

<Instructions_d'Action_de_la_procédure>

Fin

<Instructions_de_Déclaration>

Début

<Action1>

<Action2>

…

<Nom_Procédure> (Paramètres)

…

<ActionN>

Fin

3. Exécution (appel) d'une procédure

Une procédure peut être appelée soit par son algorithme propriétaire, soit par une autre procédure

(qui elle même a été appelée). L'hors de l'appel d'une procédure les évènements suivants se produisent :

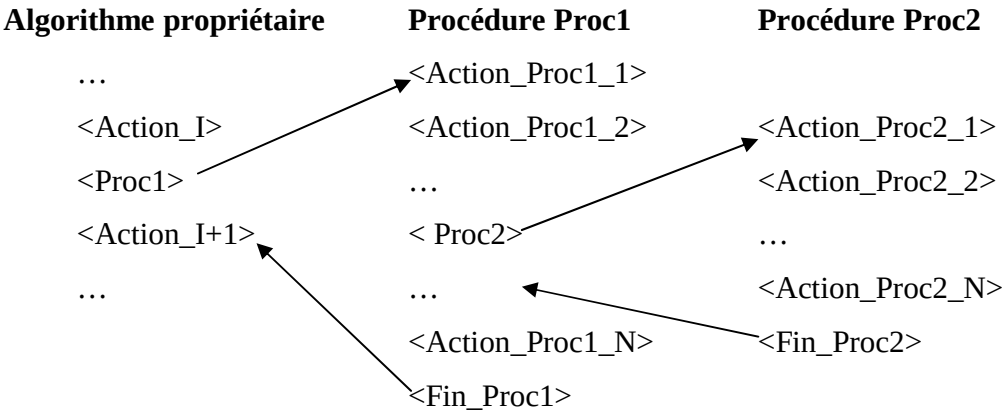
a. Arrêt momentané de l'exécution de l'appelant (en général, l'algorithme propriétaire) ;

b. Exécution des instructions de la procédure ;

c. Reprise de l'exécution de l'appelant là où il s'était arrêté (instruction suivant l'appel).

Les appels de procédures peuvent s'imbriquer autant de fois que cela est utile. Le schéma suivant synthétise ce qui a été dit.

Exemple :



Dans cet exemple l'algorithme principal appelle une procédure appelée **Proc1** qui à son tour appelle une autre procédure appelée **<Proc2>**. Ici nous supposons que <Proc1> et <Proc2> ne possèdent pas de paramètres (arguments) chose quand va voir dans ce qui suit.

4. Notions de paramètres (arguments)

Un paramètre est une variable particulière qui sert, d'une part, à la communication entre procédure appelante et une procédure appelée, et d'autre part, l'exécution de la procédure appelée avec des valeurs différentes. Comme toute variable un paramètre a un nom et un type.

4.1. Déclaration

La déclaration d'un ou de plusieurs paramètres, qui s'appellent **paramètres formels**, à l'entête d'une procédure **<Proc>** décrit les données (leurs nombres, leurs types et leurs ordres) attendues par **<Proc>** pour s'exécuter. Ces données seront passées à **<Proc>** au moment de son appel. Elles s'appellent alors, **paramètres effectifs** (valeurs des paramètres formels au moment de l'exécution de <Proc>). Les paramètres effectifs doivent correspondre alors, en nombre, en type et en ordre aux paramètres formels.

Syntaxe :

Procédure <Nom_Procédure> (<P1> : <Type P1> ; <P2> : <Type P2> ; …)

<Instructions_de_Déclaration_de_la_procédure>

Début

<Instructions_d'Action_de_la_procédure>

Fin

Exemple :

Soit la procédure suivante :

Procédure Addition1 (A,B : **Entier**)

Var C : **Entier**

Début

C **ppv** A + B

Ecrire (C)

Fin

L'appel de cette procédure dans un algorithme ce fait par l'instruction suivante :

Début

…

Addition1 (X,Y)

…

Fin

4.2. Fonctionnement

Lors de l'appel d'une procédure paramétrée,

1. La valeur du paramètre effectif passée en argument est copiée dans le paramètre formel (qui est une variable locale) ;
2. La procédure effectue alors le traitement avec cette variable ;
3. La procédure n'utilise pas directement la variable mise en paramètre effectif : elle utilise sa valeur, qu'elle a recopiée dans sa propre variable locale (le paramètre formel).

4.3. Type de passage des paramètres

Il existe deux manières de passer les paramètres, donc deux manières de substituer les paramètres effectifs aux paramètres formels : le **passage par valeur** et le **passage par variable (ou par référence)**. Pour distinguer les paramètres passés par valeur de ceux passés par variable ces derniers sont précédés par le mot clé **Var**.

Syntaxe :

Procédure <Nom_Procédure> (<Pv> : <Type_Pv> ; … ; Var <Pr1> : <Type_Pr1> ; …)

<Instructions_de_Déclaration_de_la_procédure>

Début

<Instructions_d'Action_de_la_procédure>

Fin

Exemple :

La procédure Addition1 citée plus haut peut être modifiée pour contenir les deux types de passage des paramètres cités:

Procédure Addition2 (A,B : **Entier** ; **Var** Résultat : **Entier**)

Début

Résultat **ppv** A + B

Fin

a. Passage par valeur

Dans un passage par valeur, un paramètre effectif est une valeur utilisée par la procédure mais non modifiée. Ici, la valeur du paramètre effectif est copiée dans le paramètre formel correspondant.

Exemple :

L'algorithme suivant utilise la procédure Doubler ( ):

Algorithme Doubler_Valeur

Procédure Doubler (A : **Entier**)

Début

A **ppv** A * A

Ecrire (A)

Fin

Var X,Y : **Entier**

Début

X **ppv** 3

Y **ppv** 4

Doubler (X)

Ecrire (X)

Doubler (Y)

Ecrire (Y)

Fin

L'exécution de l'algorithme Doubler_Valeur donne :

Instructions	X	Y	A
1	3		
2		4	
3 : Appel de la procédure Doubler			3
3 : 1			9
3 : 2			Affichage de : 9
Retour			
4	Affichage de : 3		
5 : Appel de la procédure Doubler			4
5 : 1			16
5 : 2			Affichage de : 16
Retour			
6		Affichage de : 4	

Dans l'exemple précédant aucune modification de la variable a, dans la procédure Doubler () appelée, ne modifie les variables X et Y passées en paramètre, parce que les modifications ne s'appliquent qu'à une copie de ces dernières (c.-à-d. X et Y).

b. Passage par variable ou par référence

Dans un passage par variable, un paramètre effectif est une variable, dont la valeur est utilisée et modifiée par la procédure. Ici, l'adresse du paramètre effectif est copiée dans le paramètre formel correspondant. Ce dernier devient un « **pointeur »** sur la variable du paramètre effectif.

Exemple :

L'algorithme Additionner_Valeur utilise la procédure Addition2 ( ) citée plus haut:

```
Algorithme Additionner_Valeur
    Procédure Addition2 (A,B : Entier ; Var Résultat : Entier)
    Début
        Résultat ppv A + B
    Fin
    Var X,Y,Z : Entier
Début
    X ppv 3
```

Y **ppv** 4

Addition (X,Y,Z)

Ecrire (Z)

Fin

L'exécution de l'algorithme Additionner_Valeur donne :

Instructions	X	Y	Z	A	B	Résultat
1	3					
2		4				
3 : Appel de la procédure Addition				3	4	
3 : 1						7
Retour			7			
4			Affichage de : 7			

II/ Fonction

1. Définition

Une fonction est une procédure particulière, qui ne génère qu'un seul et un seul résultat de type entier, réel, chaine caractère ou booléen. Ce résultat est contenu dans une variable qui a pour nom le nom de cette fonction. Avec cette particularité, une fonction ne peut donc être utiliser qu'à l'intérieur d'une expression (affectation, condition, ...).

Syntaxe :

Fonction <Nom_Fonction> (<Pv> :<Type_Pv> ;… ; Var <Pr1> :<Type_Pr1> ; …) : <Type_Fonction>

<Instructions_de_Déclaration_de_la_fonction>

Début

<Instructions_d'Action_de_la_fonction>

Fin

2. Exécution (appel) d'une fonction

Comme il a été mentionné dans la définition et à l'inverse de l'appel d'une procédure (qui représente une instruction dans le corps de l'appelant), l'appel d'une fonction doit être dans une expression.

Exemple :

Soit la fonction suivante :

Fonction Soustrac (A,B : **Entier**) : **Entier**

Début

Soustrac **ppv** A - B

Fin

L'appel de cette fonction dans un algorithme se fait comme suit:

Début

…

Résultat **ppv** Soustrac (X,Y)

…

Fin

Remarques:

1. Une expression d'affection d'une valeur à la fonction doit toujours terminer la partie des actions dans une fonction.

2. Tout ce qui a été dit pour les paramètres et leurs passages dans les procédures est valide pour les fonctions.

III/ Notion de variable locale et de variable globale

1. Portée d'une variable

La portée d'une variable est l'ensemble des procédures et des fonctions où cette variable est connue (les instructions de ces procédures et ces fonctions peuvent utiliser cette variable).

Exemple :

Soit l'algorithme suivant :

Algorithme Calcul

Procédure Addition (A,B : **Entier** ; **Var** Résultat : **Entier**)

Début

Résultat **ppv** C

Fin

Fonction Soustraction (A,B : **Entier**) : **Entier**

Var C : **Entier**

Début

C **ppv** A - B

Ecrire (C)

Soustraction **ppv** A - B

Fin

Var X,Y : **Entier**

Début

C** Partie Actions de Calcul **FC

Fin

La portée des variables X et Y est l'algorithme Calcul, la procédure Addition et la fonction Soustraction.

La portée de la variable C est la fonction Soustraction.

2. Variable globale (publique) et variable locale (privée)

Une variable définie au niveau de l'algorithme principal (procédure principale) est appelée variable globale. Sa portée est totale : toute procédure ou fonction de l'algorithme principal peut utiliser cette variable.

Une variable déclarée au sein d'une procédure ou d'une fonction est créée avec cette procédure, et disparaît avec elle. Durant tout le temps de son existence, une telle variable n'est visible que par la procédure qui l'a vu naître. C'est pourquoi ces variables par défaut sont dites **privées**, ou **locales**.

Remarques :

Lorsque le nom d'une variable locale, dans une procédure ou dans une fonction, est identique à une variable globale, cette dernière est localement masquée (c.-à-d. à l'intérieur de la procédure ou de la fonction propriétaire) et devient inaccessible.

Travaux dirigés 5

Exercice 1 :

Soit l'algorithme Enigme suivant :

```
Algorithme Enigme
    Procédure Somme_Carrées(X1, X2 : Entier; Var S : Entier)
    Début
        X1 ppv X1 * X1
        X2 ppv X2 * X2
        S ppv X1 + X2
    Fin
    Var X, Y, Z : Entier
Début
    X ppv 3
    Y ppv 4
    Z ppv 0
    Somme_Carrées (X, Y, Z)
    Ecrire (X,Y, Z)
Fin
```

1. Quel est le résultat fourni par l'algorithme Enigme.
2. Reprendre cet algorithme en remplaçant la procédure par une fonction.

Exercice 2 :

Ecrire une fonction qui calcule la partie entière d'un nombre réel positif.

Exercice 3 :

Écrire une procédure qui renvoie la somme de cinq nombres entiers pairs fournis en argument. Cette procédure doit appeler à chaque fois une fonction pour vérifier si un nombre est pair ou non. Si l'un des cinq nombres est impair le calcul est arrêté et un message d'erreur est affiché.

Exercice 4 :

Ecrire un algorithme qui rassemble les quatre techniques de tri de tableaux vues dans l'exercice 3 de la troisième série de TD (chacune dans une fonction). Cet algorithme doit donner à l'utilisateur la main pour choisir quelle est la technique à utiliser.

Solutions types pour quelques exercices

Chapitre I

Exercice 2 :

1. L'algorithme Somme_Soustraction calcule la somme et la soustraction de deux nombres entiers, et affiche à la fin les résultats demandés.

Algorithme Somme_Soustraction

Var A, B, Som, Sous : **Entier**

Début

Lire (A)

Lire (B)

Som **ppv** A + B

Sous **ppv** A - B

Ecrire (Som)

Ecrire (Sous)

Fin

- Déroulement de l'algorithme Somme_Soustraction sur l'exemple A= 10 et B= 7

Instructions	A	B	Som	Sous
1	10			
2		7		
3			17	
4				3
5			Affichage de : 17	
6				Affichage de : 3

2. L'algorithme Ord_Succ_Pred donne l'ordre, le successeur et le prédécesseur de la valeur d'une variable dans le type jour vu dans le cours.

Algorithme Ord_Succ_Pred

Type Jour = (Samedi, Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi)

Var J : **Jour**

Début

Lire (J)

Ecrire (Ord(J))

Ecrire (Succ(J))

Ecrire (Pred(J))

Fin

- Déroulement de l'algorithme Ord_Succ_Pred sur l'exemple J= Mardi

Instructions	**J**
1	Mardi
2	Affichage de : 3
3	Affichage de : Mercredi
4	Affichage de : Dimanche

Exercice 5 :

L'algorithme Calcul_4 concatène les valeurs des deux variables A et B ('423'+'12'='42312'). Le déroulement de cet algorithme est le suivant :

Instructions	**A**	**B**	**C**
1	'423'		
2		'12'	
3			'42312'

Chapitre II

Exercice 1 :

1. Reprise de l'algorithme de résolution de l'équation du deuxième degré vu dans le premier chapitre du cours en respectant les syntaxes des concepts vus dans le deuxième chapitre.

```
Algorithme Solution__Equation_Second_Degré
    Var  A, B, C, Delta : Entier
         R1, R2 : Réel
Début
    Ecrire ('Introduire A B C')
    Lire (A,B,C)
    Delta ppv B^2- 4*A*C
    Si (Delta <0) Alors
        Ecrire ('Pas de solution dans l'espace réel')
    Sinon
        R1 ppv (-B+ Racine (Delta))/2*A
        R2 ppv (-B- Racine (Delta))/2*A
        Ecrire ('Les deux solutions sont: ')
        Ecrire (R1,R2)
    FinSi
Fin
```

2. Généralisation de l'algorithme pour résoudre N équations où N est un entier naturel ≥ 1.

```
Algorithme Solution_N_Equation_Second_Degré
    Var  A, B, C, Delta, N, i : Entier
         R1, R2 : Réel
Début
    Lire (N)
    i ppv 1
    Tant que (i≤N) Faire
        Ecrire ('Solution équation ', i)
        Ecrire ('Introduire A B C')
        Lire (A,B,C)
```

```
            Delta ppv B^2- 4*A*C
            Si (Delta <0) Alors
                Ecrire ('Pas de solution dans l'espace réel')
            Sinon
                R1 ppv (-B+ Racine (Delta))/2*A
                R2 ppv (-B- Racine (Delta))/2*A
                Ecrire ('Les deux solutions sont: ')
                Ecrire (R1,R2)
            FinSi
            i ppv i+1
        FinTantque
    Fin
```

Exercice 4 :

L'algorithme PGCD calcule le plus grand diviseur commun de deux entiers A et B positifs :

```
Algorithme PGCD
    Var   A, B, PGCD : Entier
Début
    Ecrire ('Entrer A et B ')
    Lire (A,B)
    Tant que (A*B < > 0) Faire
        Si (A > B) Alors
            A ppv A-B
        Sinon
            B ppv B-A
        FinSi
    FinTantque
    Si (A = 0) Alors
        Ecrire ('PGCD = ', B)
    Sinon
        Ecrire ('PGCD = ', A)
    FinSi
Fin
```

Chapitre III

Exercice 2:

Etant donné, un tableau T2D1 à deux dimensions de cardinalité N et M :

1. L'algorithme Remplissage_Tableau_2D remplit T2D1 avec des éléments entiers.

```
Algorithme Remplissage_Tableau_2D
    Var  T2D1 : Tableau [N,M] de Entier
         i, j  : Entier
Début
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
            Lire (T2D1[i, j])
            j ppv j+1
        FinTantque
        i ppv  i+1
    FinTantque
Fin
```

2. L'algorithme Comptage_Positif_Negatif_Nul compte le nombre d'éléments positifs, négatifs et nuls dans le tableau.

```
Algorithme Comptage_Positif_Negatif_Nul
    Var  T2D1 : Tableau [N,M] de Entier
         i, j, CptPos,CptNeg,CptNul : Entier
Début
    CptPos ppv 0
    CptNeg ppv 0
    CptNul ppv 0
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
```

```
                Si (T2D1[i, j] > 0) Alors
                    CptPos ppv CptPos + 1
                Sinon
                    Si (T2D1[i, j] < 0) Alors
                        CptNeg ppv CptNeg + 1
                    Sinon
                        CptNul ppv CptNul + 1
                    FinSi
                FinSi
                j ppv j+1
            FinTantque
            i ppv  i+1
        FinTantque
        Ecrire ('Le nombre des éléments positifs est : ', CptPos)
        Ecrire ('Le nombre des éléments négatifs est : ', CptNeg)
        Ecrire ('Le nombre des éléments nuls est : ', CptNul)
Fin
```

3. L'algorithme Comptage_Apparition_Element compte le nombre d'apparitions d'un élément dans le tableau.

```
Algorithme Comptage_Apparition_Element
    Var   T2D1 : Tableau [N,M] de Entier
          i, j, NbrApp, Elem : Entier
Début
    NbrApp ppv 0
    Ecrire ('Entrer l'élément recherché')
    Lire  (Elem)
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
            Si (T2D1[i, j] =Elem) Alors
                NbrApp ppv NbrApp + 1
```

```
                FinSi
                j ppv j+1
            FinTantque
            i ppv  i+1
        FinTantque
        Ecrire ('Le nombre d'apparitions de ', Elem, ' est : ', NbrApp)
Fin
```

4. L'algorithme Min_Max cherche l'élément minimal et l'élément maximal dans le tableau.

```
Algorithme Min_Max
    Var   T2D1 : Tableau [N,M] de Entier
          i, j,Min,Max : Entier
Début
    Min ppv T2D1[1,1]
    Max ppv T2D1[1,1]
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
            Si (Min > T2D1[i, j]) Alors
                Min ppv T2D1[i, j]
            FinSi
            Si (Max < T2D1[i, j]) Alors
                Max ppv T2D1[i, j]
            FinSi
            j ppv j+1
        FinTantque
        i ppv  i+1
    FinTantque
    Ecrire ('L'élément minimal est : ', Min)
    Ecrire ('L'élément maximal est : ', Max)
Fin
```

5. L'algorithme Somme_Elements calcule la somme des éléments du tableau.

```
Algorithme Somme_Elements
    Var  T2D1 : Tableau [N,M] de Entier
         i, j,Somme : Entier
Début
    Somme ppv 0
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
            Somme ppv Somme + T2D1[i, j]
            j ppv j+1
        FinTantque
        i ppv  i+1
    FinTantque
    Ecrire ('La somme des éléments du tableau est égale à : ', Somme)
Fin
```

6. L'algorithme Somme_2_Tableaux_2D produit la somme des deux tableaux T2D1 et T2D2 dans un troisième tableau T2D3.

```
Algorithme Somme_2_Tableaux_2D
    Var  T2D1, T2D2, T2D3 : Tableau [N,M] de Entier
         i, j : Entier
Début
    i ppv 1
    Tant que (i ≤ N) Faire
        j ppv 1
        Tant que (j ≤ M) Faire
            T2D3[i, j] ppv Somme T2D1[i, j]+ T2D2[i, j]
            j ppv j+1
        FinTantque
        i ppv  i+1
    FinTantque
```

```
        C Affichage des éléments du tableau T2D3 ligne par ligne FC
        i ppv 1
        Tant que (i ≤ N) Faire
            j ppv 1
            Ecrire ('Les éléments de la ', i, ' ligne :')
            Tant que (j ≤ M) Faire
                Ecrire (T2D3[i, j])
                j ppv j+1
            FinTantque
            i ppv  i+1
        FinTantque

    Fin
```

Exercice 3 (1ère et 2ème technique de tri):

Soit à trier un tableau T de 12 entiers dans l'ordre croissant.

1. L'algorithme Tri_Tab_Sélection trie le tableau T en utilisant la technique de tri par sélection :

```
Algorithme Tri_Tab_Sélection
    Var T : Tableau [12] de Entier
        i, j, Indice, Temp : Entier
Début
    i ppv 1
    Tant que (i < 12) Faire
        indice ppv  i
        j ppv i+1
        Tant que (j ≤ 12) Faire
            Si (T[Indice] > T[j]) Alors
                Indice ppv j
            FinSi
        FinTantque
        Si (indice <> i) Alors
            Temp ppv T[i]
            T[i] ppv T[Indice]
            T[indice] ppv Temp
```

```
            FinSi
            i ppv i+1
        FinTantque
    Fin
```

2. L'algorithme Tri_Tab_Bulles trie le tableau T en utilisant la technique de tri à bulles:

```
Algorithme Tri_Tab_Bulles
    Var  T : Tableau [12] de Entier
         i,Temp : Entier
         Permut : Booléen
    Début
        Permut ppv Vrai
        Tant que (Permut = Vrai) Faire
            i ppv 1
            Permut ppv Faux
            Tant que (i < 12) Faire
                Si (T[i] > T[i+1]) Alors
                    Temp ppv T[i]
                    T[i] ppv  T[i+1]
                    T[i+1] ppv Temp
                    Permut ppv Vrai
                FinSi
                i ppv i+1
            FinTantque
        FinTantque
    Fin
```

Exercice 4 (1ère technique):

1. Recherche séquentielle ou linéaire : Consiste à comparer la valeur recherchée aux différents éléments du tableau jusqu'à trouver cette valeur ou atteindre la fin du tableau.

L'algorithme Recherche_Seq_Tab recherche une valeur donnée dans le tableau T

```
Algorithme Recherche_Seq_Tab
    Var  T : Tableau [12] de Entier
         i,X : Entier
```

```
        Permut : Booléen
Début
    i ppv 1
    Trouv ppv Faux
    Ecrire ('Entrez la valeur recherchée : ')
    Lire (X)
    Tant que (i =< 12) Et (Trouv = Faux) Faire
        Si (T[i] > X) Alors
            Trouv ppv Vrai
        Sinon
            i ppv i+1
        FinSi
    FinTantque
    Si (Trouv = Vrai) Alors
        Ecrire ('Valeur recherchée trouvée dans la case:' ; i)
    Sinon
        Ecrire ('Valeur recherchée non trouvée)
    FinSi
Fin
```

Chapitre IV

Exercice 2 :

Reprenant les questions 1,2 et 3 de l'exercice 2 de la série de TD 3 en utilisant la structure répétitive « **Pour … Faire** » puis la structure «**Répéter … Jusqu'à**».

Etant donné, un tableau T2D1 à deux dimensions de cardinalité N et M :

1. L'algorithme Remplissage_Tableau_2D_Pour remplit T2D1 avec des éléments entiers en utilisant la boucle « **Pour … Faire** ».

```
Algorithme Remplissage_Tableau_2D_Pour
    Var  T2D1 : Tableau [N,M] de Entier
         i, j  : Entier
Début
    Pour i de 1 à N Faire
        Pour j de 1 à M Faire
            Lire (T2D1[i, j])
        FinPour
    FinPour
Fin
```

L'algorithme Remplissage_Tableau_2D_Répéter remplit T2D1 avec des éléments entiers en utilisant la boucle «**Répéter … Jusqu'à**».

```
Algorithme Remplissage_Tableau_2D_Répéter
    Var  T2D1 : Tableau [N,M] de Entier
         i, j  : Entier
Début
    i ppv 1
    Répéter
        j ppv 1
        Répéter
            Lire (T2D1[i, j])
            j ppv j+1
        Jusqu'à j > M
        i ppv i+1
    Jusqu'à i > N
Fin
```

2. L'algorithme Comptage_Positif_Negatif_Nul_Pour compte le nombre d'éléments positifs, négatifs et nuls dans le tableau en utilisant la boucle « **Pour … Faire** ».

```
Algorithme Comptage_Positif_Negatif_Nul_Pour
    Var  T2D1 : Tableau [N,M] de Entier
         i, j, CptPos,CptNeg,CptNul : Entier
Début
    CptPos ppv 0
    CptNeg ppv 0
    CptNul ppv 0
    Pour i de 1 à N Faire
        Pour j de 1 à M Faire
            Si (T2D1[i, j] > 0) Alors
                CptPos ppv CptPos + 1
            Sinon
                Si (T2D1[i, j] < 0) Alors
                    CptNeg ppv CptNeg + 1
                Sinon
                    CptNul ppv CptNul + 1
                FinSi
            FinSi
        FinPour
    FinPour
    Ecrire ('Le nombre des éléments positifs est : ', CptPos)
    Ecrire ('Le nombre des éléments négatifs est : ', CptNeg)
    Ecrire ('Le nombre des éléments nuls est : ', CptNul)
Fin
```

L'algorithme Comptage_Positif_Negatif_Nul_Répéter compte le nombre d'éléments positifs, négatifs et nuls dans le tableau en utilisant la boucle «**Répéter … Jusqu'à**».

```
Algorithme Comptage_Positif_Negatif_Nul_Pour
    Var  T2D1 : Tableau [N,M] de Entier
         i, j, CptPos,CptNeg,CptNul : Entier
Début
    CptPos ppv 0
```

```
        CptNeg ppv 0
        CptNul ppv 0
        i ppv 1
        Répéter
            j ppv 1
            Répéter
                Si (T2D1[i, j] > 0) Alors
                    CptPos ppv CptPos + 1
                Sinon
                    Si (T2D1[i, j] < 0) Alors
                        CptNeg ppv CptNeg + 1
                    Sinon
                        CptNul ppv CptNul + 1
                    FinSi
                FinSi
                j ppv j+1
            Jusqu'à j > M
        Jusqu'à i > N
        Ecrire ('Le nombre des éléments positifs est : ', CptPos)
        Ecrire ('Le nombre des éléments négatifs est : ', CptNeg)
        Ecrire ('Le nombre des éléments nuls est : ', CptNul)
Fin
```

3. L'algorithme Comptage_Apparition_Element_Pour compte le nombre d'apparitions d'un élément dans le tableau en utilisant la boucle « **Pour … Faire** ».

```
Algorithme Comptage_Apparition_Element_Pour
    Var  T2D1 : Tableau [N,M] de Entier
         i, j, NbrApp, Elem : Entier
Début
    NbrApp ppv 0
    Ecrire ('Entrer l'élément recherché')
    Lire  (Elem)
    Pour i de 1 à N Faire
```

```
        Pour j de 1 à M Faire
            Si (T2D1[i, j] =Elem) Alors
                NbrApp ppv NbrApp + 1
            FinSi
        FinPour
    FinPour
    Ecrire ('Le nombre d'apparitions de ', Elem, ' est : ', NbrApp)
Fin
```

L'algorithme Comptage_Apparition_Element_Répéter compte le nombre d'apparitions d'un élément dans le tableau en utilisant la boucle «**Répéter … Jusqu'à**».

```
Algorithme Comptage_Apparition_Element_Répéter
    Var  T2D1 : Tableau [N,M] de Entier
         i, j, NbrApp, Elem : Entier
Début
    NbrApp ppv 0
    Ecrire ('Entrer l'élément recherché')
    Lire  (Elem)
    i ppv 1
    Répéter
        j ppv 1
        Répéter
            Si (T2D1[i, j] =Elem) Alors
                NbrApp ppv NbrApp + 1
            FinSi
            j ppv j+1
        Jusqu'à j > M
        i ppv  i+1
    Jusqu'à i  > N
    Ecrire ('Le nombre d'apparitions de ', Elem, ' est : ', NbrApp)
Fin
```

Exercice 3 :

1. L'algorithme Réponse_Question_Tantque repose la même question ('Voulez vous sortir ? O/N'), en utilisant la structure «**Tant que … Faire**», jusqu'à ce que l'une des deux réponses attendues soit saisie au clavier par l'utilisateur.

```
Algorithme Réponse_Question_Tantque
    Var  Réponse : Chaine de caractères
Début
    Réponse ppv 'X'
    Tant que (Réponse <> 'O') Et (Réponse <> 'N') Faire
        Ecrire ('Voulez vous sortir ? O/N')
        Lire (Réponse)
    FinTantque
Fin
```

L'algorithme Réponse_Question_Repeter repose la même question ('Voulez vous sortir ? O/N'), en utilisant la structure «**Répéter … Jusqu'à**», jusqu'à ce que l'une des deux réponses attendues soit saisie au clavier par l'utilisateur.

```
Algorithme Réponse_Question_Repeter
    Var  Réponse : Chaine de caractères
Début
    Répéter
        Ecrire ('Voulez vous sortir ? O/N')
        Lire (Réponse)
    Jusqu'à (Réponse = 'O') Ou (Réponse = 'N')
Fin
```

2. L'instruction «**Répéter … Jusqu'à**» est la mieux adaptée à ce type de problème car elle permet d'éviter l'instruction d'initialisation de la variable « Réponse » (Réponse **ppv** 'X') dans le premier algorithme, écrite dans but de permettre l'exécution de l'instruction «**Tant que … Faire**».

3. La structure «**Pour … Faire**» ne convient pas à ce genre de problème parce que :

- Cette structure est destinée à répéter une suite d'instructions un certain nombre de fois;
- La variable de répétition varie dans un intervalle défini et ne doit pas être modifiée par le traitement.

Exercice 4 :

1. L'algorithme Afficher_Menu affiche le menu demandé jusqu'à ce que l'utilisateur entre l'une des valeurs affichées (1,2,3,0).

```
Algorithme Afficher_Menu
    Var  Réponse : Entier
         Confirmation : Chaine de caractères
Début
    Répéter
        Ecrire ('1 : Exécuter traitement 1')
        Ecrire ('2 : Exécuter traitement 2')
        Ecrire ('3 : Exécuter traitement 3')
        Ecrire ('0 : Quitter')
        Lire (Réponse)
        Si (Réponse = 1) Alors
            Ecrire ('Le traitement 1 va être exécuté')
        FinSi
        Si (Réponse = 2) Alors
            Ecrire ('Le traitement 2 va être exécuté')
        FinSi
        Si (Réponse = 3) Alors
            Ecrire ('Le traitement 3 va être exécuté')
        FinSi
        Si (Réponse = 0) Alors
            Répéter
                Ecrire ('Voulez vous sortir ? O/N')
                Lire (Confirmation)
            Jusqu'à (Confirmation = 'O') Ou (Confirmation = 'N')
            Si (Confirmation = 'N') Alors
                Réponse ppv 4
            FinSi
        FinSi
    Jusqu'à (Réponse  = 0)
Fin
```

2. L'algorithme Afficher_Menu_Choix utilise la structure **« Choix…Dans »** à la place de la structure **« Si…Alors »**.

```
Algorithme Afficher_Menu _Choix
    Var  Réponse : Entier
         Confirmation : Chaine de caractères
Début
    Répéter
        Ecrire ('1 : Exécuter traitement 1')
        Ecrire ('2 : Exécuter traitement 2')
        Ecrire ('3 : Exécuter traitement 3')
        Ecrire ('0 : Quitter')
        Lire (Réponse)
        Choix Réponse Dans
        Début
            1 : Ecrire ('Le traitement 1 va être exécuté')
            2 : Ecrire ('Le traitement 2 va être exécuté')
            3 : Ecrire ('Le traitement 3 va être exécuté')
            0 : Répéter
                    Ecrire ('Voulez vous sortir ? O/N')
                    Lire (Confirmation)
                Jusqu'à (Confirmation = 'O') Ou (Confirmation = 'N')
                Si (Confirmation = 'N') Alors
                    Réponse ppv 4
                FinSi
        FinChoix
    Jusqu'à (Réponse  = 0)
Fin
```

Chapitre V

Exercice 1 :

1. Le résultat fourni par l'algorithme Enigme est le suivant :

Instructions	X	Y	Z	X1	X2	S
1	3					
2		4				
3			0			
4 : Appel de la procédure Addition				3	4	0
4 : 1				9		
4 : 2					16	
4 : 3						25
Retour			25			
5	Affichage de : 3	Affichage de : 4	Affichage de : 25			

2. Reprenant l'algorithme Enigme en remplaçant la procédure Somme_Carrées par une fonction.

```
Algorithme Enigme
    Fonction Somme_Carrées(X1, X2 : Entier) : Entier
    Début
        X1 ppv X1 * X1
        X2 ppv X2 * X2
        Somme_Carrées ppv X1 + X2
    Fin
    Var X, Y, Z : Entier
Début
    X ppv 3
    Y ppv 4
    Z ppv 0
    Z ppv Somme_Carrées (X, Y)
    Ecrire (X,Y, Z)
Fin
```

Exercice 3 :

La procédure Somme_5_Pair renvoie la somme de cinq nombres entiers pairs fournis en argument. Cette procédure appelle à chaque fois la fonction Verification_Pair pour vérifier si un nombre est pair ou non.

```
Fonction Verification_Pair (A : Entier) : Booléen
Début
    Si (A Mod 2 = 0) Alors
        Verification_Pair ppv Vrai
    Sinon
        Verification_Pair ppv Faux
    FinSi
Fin

Procédure Somme_5_Pair (X1,X2,X3,X4,X5 : Entier ; Var Somme : Entier)
    Var EstImpair : Booléen
Début
    EstImpair ppv Faux
    Si (NonVerification_Pair(X1)) Alors
        Ecrire (X1, ' est un nombre impair')
        EstImpair ppv Vrai
    Sinon Si (NonVerification_Pair(X2)) Alors
            Ecrire (X2, ' est un nombre impair')
            EstImpair ppv Vrai
        Sinon Si (NonVerification_Pair(X3)) Alors
                Ecrire (X3, ' est un nombre impair')
                EstImpair ppv Vrai
            Sinon Si (NonVerification_Pair(X4)) Alors
                    Ecrire (X4, ' est un nombre impair')
                    EstImpair ppv Vrai
                Sinon Si (NonVerification_Pair(X5)) Alors
                    Ecrire (X5, ' est un nombre impair')
                    EstImpair ppv Vrai
                FinSi
```

```
                FinSi
            FinSi
        FinSi
        Si (Non EstImpair) Alors
            Somme ppv X1 + X2 +X3 +X4 + X5
        FinSi
    Fin
```

Références bibliographiques

- **« TEC 611, Structures de données algorithmiques et preuves de programmes »**, *Pr. Said Ghoul*, Université Badji Mokhtar, Annaba, 1991-1992;
- **« Algorithmique et programmation pour non-matheux»**, *Pr Christophe Darmangeat*, Université Paris 6, France ; http://pise.info/algo/index.htm
- **« Informatique 1 »**, *Dr. Mohamed Mahdi Benmoussa*, École Nationale Polytechnique de Constantine, Algérie ;
- **« Cours d'algorithmique »**, *Pr. Ismail El haddioui*, Ecole Supérieure de Technologie de Casablanca, Maroc.
- **« Cours algorithmique et programmation informatique »**, *Dr. Marie-Agnès peraldi-frati*, UNS/IUT de Nice côte d'azur, France.