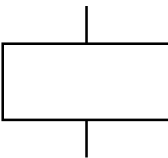

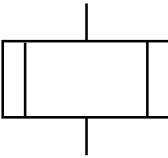
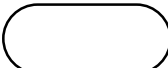
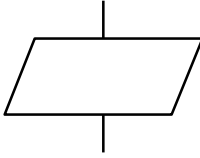

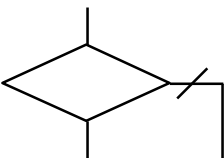


**1. DEFINITIONS**

**Algorithme** : Description en langage naturel de la suite des actions effectuées par un programme structuré. Un algorithme est écrit en utilisant un langage de description d'algorithme (LDA). L'algorithme ne doit pas être confondu avec le programme proprement dit (tel que Pascal, C, ..)

**Algorigramme** : Traduction graphique de l'algorithme. Aussi appelé Organigramme.

Quelques symboles utilisés pour construire un algorigramme

SYMBOLE	DESIGNATION	SYMBOLE	DESIGNATION
Symboles de traitement		Symboles auxiliaires	
	<b>Symbole général</b> Opération sur des données, instructions, ...		<b>Renvoi</b> Connecteur utilisé à la fin et en début de ligne pour en assurer la continuité
	<b>Sous-programme</b> Portion de programme		<b>Début, fin ou interruption</b> d'un algorithme
	<b>Entrée-Sortie</b> Mise à disposition ou enregistrement d'une information		<b>Liaison</b> Les différents symboles sont reliés entre eux par des lignes de liaison. Le cheminement va de haut en bas et de gauche à droite. Un cheminement différent est indiqué à l'aide d'une flèche.
Symbole de test			
	<b>Branchement</b> Décision d'un choix parmi d'autres en fonction des conditions		

**Syntaxe** : Règles d'écriture d'un langage donné.

**Type de données :**

Un programme peut-être amené à manipuler différents types de données :

- **booléen** : valeur pouvant être soit Vraie, soit Fausse.
- **entiers** : valeur numériques entières pouvant être signées ou non signées (codées sur un ou plusieurs octets).
- **réels** : valeurs numériques codées avec une mantisse et un exposant.
- **caractère** : octet correspondant à un code ASCII.
- **chaîne de caractères** : ensemble de caractères.
- **tableau de données** : ensemble de données de même type (exemple : tableau d'entiers, tableau de réels).

Toutes ces données sont codées sous forme d'octets en mémoire.

**Constante** : donnée manipulée par un programme et ne pouvant être modifiée.

Exemple : Constante Pi = 3.141559

**Variable** : donnée manipulée par un programme et pouvant être modifiée.

Ce peut être :

- une donnée d'entrée ;
- le résultat final d'un calcul ;
- un résultat intermédiaire de calcul.

**Identificateur** : nom explicite d'une constante, d'une variable ou d'une fonction.

Exemples : Conversion\_BCD, Résultat, Lettre...

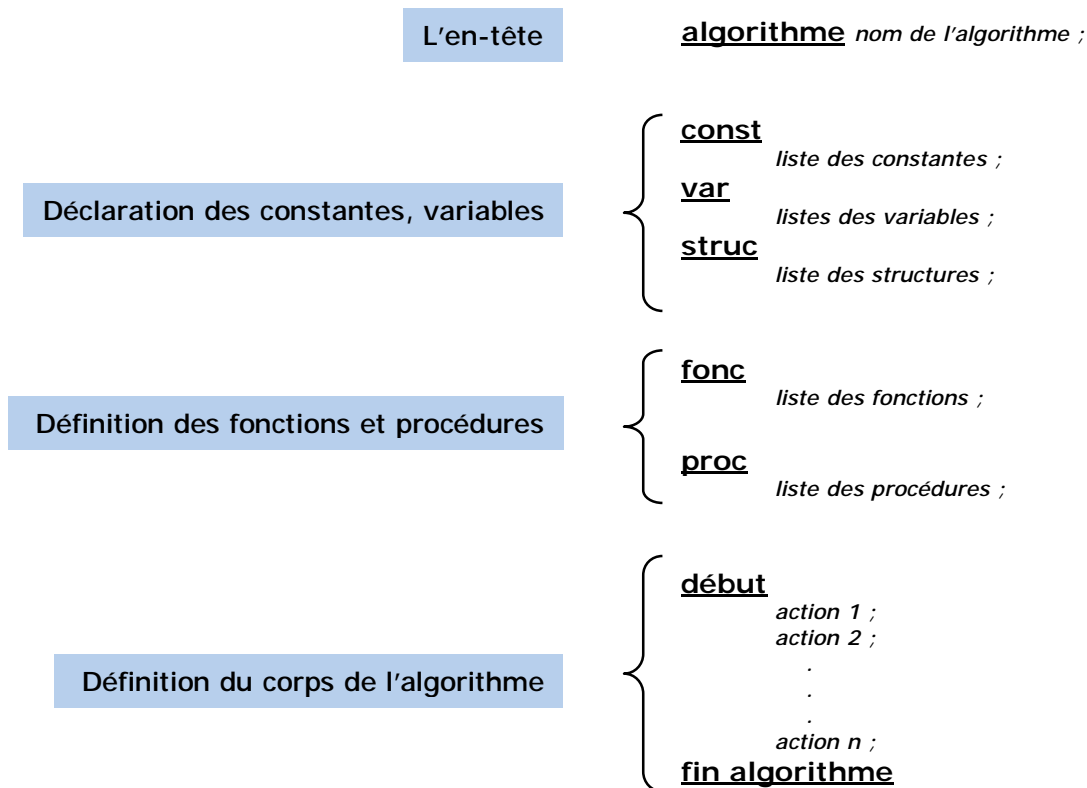
**Procédures et fonctions** : Une procédure ou une fonction effectuent une suite d'actions élémentaires constituant un tout.

Une fonction se différencie d'une procédure par le fait qu'elle fournit un résultat.

## 2. ORGANISATION D'UN PROGRAMME

L'algorithme d'un programme est organisé en plusieurs parties :

- déclaration des constantes
- déclaration des variables
- définition des fonctions et procédures
- définition du programme principal



### 2.1 Déclaration des constantes

Syntaxe : **Constante** NomConstante : [Type] = Valeur

Exemples : **Constante** Pi : Reel = 3.141559  
**Constante** NombreLettres : Entier = 10

### 2.2 Déclaration des variables

Syntaxe : **Variable** NomVariable : [Type]

Exemples : **Variable** Rayon : Reel  
**Variable** Compteur : Entier  
**Variable** Lettre : Caractere

### 2.3 Définition des fonctions et procédures

Les procédures et fonctions peuvent nécessiter éventuellement un ou plusieurs paramètres d'entrée ou de sortie.

Un paramètre d'entrée est la référence à une variable manipulée par la procédure ou la fonction.

Un paramètre de sortie est une valeur renvoyée par une fonction.

Une fonction ou une procédure peut elle-même appeler une ou plusieurs fonctions et procédures.

*Syntaxe de la déclaration d'une fonction :*

**Fonction** NomFonction (NomEntrée1 : [Type], NomEntrée2 : [Type],...) : [TypeDuRésultat]

**Constante** ~ déclaration des constantes locales ~

**Variable** ~ déclaration des variables locales ~

**Début**

~ description des actions effectuées par la fonction ~

**Fin**

*Syntaxe de l'appel d'une fonction :*

Variable ← NomFonction (NomEntrée1, NomEntrée2...)

Exemple de déclaration de fonction :

**Fonction** Moyenne (Note1 : Reel, Note2 : Reel) : Reel

**Variable** Intermediaire : Reel

**Début**

Intermediaire ← Note1 + Note2

Intermediaire ← Intermediaire / 2

Moyenne ← Intermediaire

**Fin**

Exemples d'utilisation de la fonction :

Afficher (Moyenne(10.5,15)) ou NouvelleNote ← Moyenne (10,5.5)

**Nota : la déclaration et l'appel d'une procédure ne sont pas développés ici.**

### 2.4 Définition du programme principal

Le programme principal consiste en une suite d'opérations élémentaires faisant souvent appel à des fonctions ou procédures. Ces différentes opérations sont mentionnées en utilisant les structures algorithmiques décrites au paragraphe 5.

Le programme principal est délimité par les mots-clefs **Début** et **Fin**

## 3. AFFECTATION

Une affectation consiste à attribuer une valeur à une variable.

La syntaxe générale est la suivante : NomVariable ← Expression

« Expression » peut être :

- une constante. ....Ex : surface ← 40
- une autre variable. ....Ex : Donnee ← ValeurMemorisee
- le résultat d'une fonction. .... Ex : resultat ← racine (nombre)
- un calcul portant sur ces différents éléments. .... Ex : surface ← (PI \* Carre (Diametre)) / 4

**Nota : Un abus d'écriture nous amène parfois à voir le signe = à la place de la flèche. C'est valable normalement pour l'écriture en langage de programmation (Basic compilé, Pascal, ...) mais pas dans un algorithme.**

**4. OPERATEURS – CONDITIONS**

**4.1 Opérateurs**

Les opérateurs permettent d’élaborer une expression en vue d’effectuer un calcul ou une comparaison.

L’usage des parenthèses est vivement conseillé dans le cas d’expressions complexes.

Nature	Variables utilisées	Notation	Signification
Opérateurs arithmétiques	Entier Réel	+	Addition
		-	Soustraction
		*	Multiplication
		/	Division (réelle)
		DIV	Division entière
		MOD	Reste de la division entière
Opérateurs logiques	Booléen Entier	et	Fonction ET
		ou	Fonction OU
		ouex	Fonction OU EXCLUSIF
		non	Fonction NON
Opérateur de concaténation	Chaîne de caractères	+	Concaténation
Opérateurs de comparaison	Booléen Entier Réel Caractère Chaîne de caractères	=	Egal
		≠	Différent
		<	Inférieur
		>	Supérieur
		≤	Inférieur ou égal
		≥	Supérieur ou égal

**4.2 Conditions**

Dans les structures algorithmiques qui vont suivre, le terme « Condition » peut représenter :

- une condition simple : Ex :  $x \geq 10$  Indice  $3 \leq 80$
- une condition complexe : Ex :  $(x > 0) \text{ ET } ((y > 0) \text{ OU } (z > 0))$   
 $(\text{Indice} \geq 1) \text{ ET } (\text{Indice} \leq 10) \sim \text{pour } 1 \leq \text{Indice} \leq 10 \sim$

**5. LES STRUCTURES ALGORITHMIQUES**

Les structures algorithmiques sont réparties en 3 catégories :

- structures linéaire d'opérations;
- structures alternatives (ou conditionnelles) ou de choix : en fonction d'une condition, le programme exécute des opérations différentes;
- structures itératives ou répétitives: sous contrôle d'une condition, une séquence d'opérations est exécutée répétitivement.

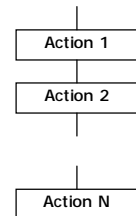
**5.1 Structure linéaire**

Les actions successives sont mentionnées les unes après les autres.

*Syntaxe*

Action1  
Action2  
...  
ActionN

*Algorithme*



*Remarque* : dans la suite, la notation « Actions » ou « ActionsN » représentera une succession d’actions comme ci-dessus.

*Exemple* : Calcul d’un produit de 2 nombres

**Variable**

a,b : réel ~ opérandes ~  
p : réel ~ résultat du produit ~

**Début**

Afficher ('Saisir le nombre a ')  
 Saisir (a)  
 Afficher ('Saisir le nombre b ')  
 Saisir (b)  
 $p \leftarrow a * b$   
 afficher (p)

**Fin**

**5.2 Structures alternatives**

**5.2.1 Structure SI ... ALORS ...**

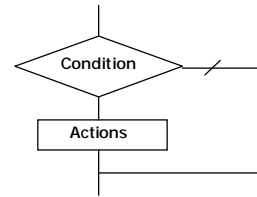
Une condition est testée pour déterminer si l'action ou le groupe d'actions suivant doit être exécuté.

*Syntaxe*

**Si** Condition **Alors**  
 Actions  
**FinSi**

*Remarque* : au lieu de **FinSi** on écrit aussi **fsi**

*Algorithme*



*Exemple* : Calcul d'une racine carrée

**Variable**

x: réel ~ opérande ~  
 r: réel ~ résultat de la racine carrée ~

**Début**

Afficher ('Saisir le nombre x')  
 Saisir (x)  
**Si**  $x > 0$  **Alors**  
      $r \leftarrow \text{racine}(x)$   
     afficher (r)

**FinSi**

**Fin**

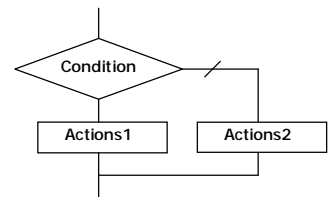
**5.2.2 Structure SI ... ALORS ...SINON ...**

Une condition est testée pour déterminer quelle action ou quel groupe d'actions doit être exécuté.

*Syntaxe*

**Si** Condition **Alors**  
 Actions1  
**Sinon**  
 Actions2  
**FinSi**

*Algorithme*



*Exemple* : Calcul d'une racine carrée

**Variables** :

x: réel ~ opérande ~  
 r: réel ~ résultat de la racine carrée ~

**Début**

Afficher ('Saisir le nombre x')  
 Saisir (x)  
**Si**  $x < 0$  **Alors**  
     afficher ('x est négatif')  
**Sinon**  
      $r \leftarrow \text{racine}(x)$   
     afficher (r)

**FinSi**

**Fin**

### 5.2.3 Structure de choix multiple

Une donnée est comparée successivement à des valeurs constantes :

*Syntaxe*

**Suivant** Donnée **faire**

Valeur1 : Actions1

Valeur2 : Actions2

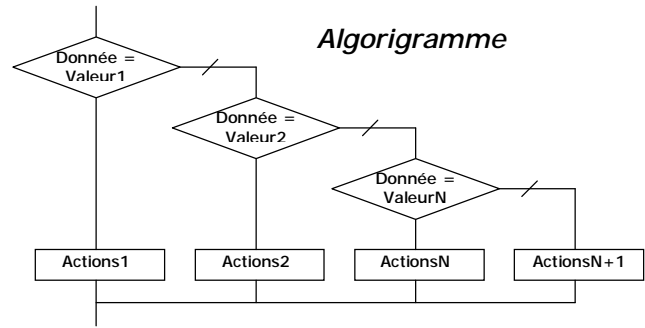
...

ValeurN : ActionsN

**Sinon**

ActionsN+1

**FinSuivant**



*Remarques* : la partie « ActionsN+1 » peut ne pas exister.

Plusieurs valeurs différentes peuvent être regroupées sur une même ligne si les actions correspondantes sont identiques.

*Exemple* : Affichage de la nature d'un caractère

**Variable**

c: caractère ~ caractère saisi au clavier ~

**Début**

Afficher ('Taper un caractère')

Saisir (c)

**Suivant** c **Faire**

'A'..'Z' : afficher ('Lettre majuscule')

'a'..'z' : afficher ('Lettre minuscule')

'0'..'9' : afficher ('Chiffre')

**Sinon**

afficher ('Ni Lettre Ni Chiffre')

**FinSuivant**

**Fin**

## 5.3 Structures itératives (ou répétitives)

### 5.3.1 Structure REPETER ... JUSQUA ...

Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une condition soit vérifiée.

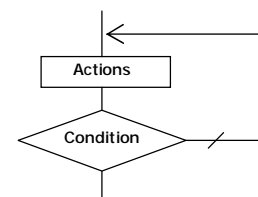
*Syntaxe*

**Répéter**

Actions

**Jusqu'à** Condition

*Algorithme*



*Remarque* : la vérification de la condition s'effectue **après** les actions. Celles-ci sont donc exécutées au moins une fois.

*Exemple* : exécution répétitive d'un programme

**Variables** :

a,b : réel ~ opérandes ~

p : réel ~ résultat du produit ~

c : caractère ~ réponse de l'utilisateur ~

DébutRépéter

Afficher ('Saisir le nombre a')

Saisir (a)

Afficher ('Saisir le nombre b')

Saisir (b)

 $p \leftarrow a * b$ 

afficher (p)

afficher ('encore un calcul ? Non touche N ; Oui autre touche')

saisir (c)

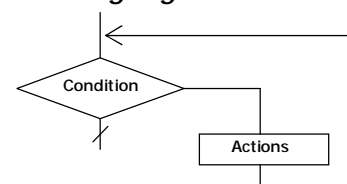
Jusqu'à c = 'N'Fin

### 5.3.2 Structure TANT QUE ... FAIRE ...

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est vraie.

*Syntaxe*TantQue ConditionFaire

Actions

FinFaire*Algorithme*

*Remarque* : la vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.

*Exemple* : Exécution répétitive d'une action

DébutTant Que Non (ToucheAppuyée)Faire

Afficher ('Attente')

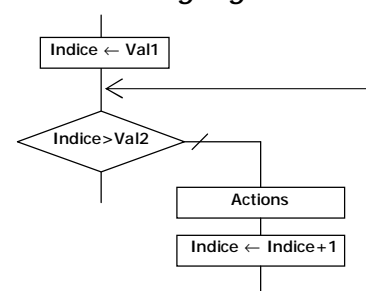
FinFaireFin

### 5.3.3 Structure POUR Indice DE ... A .... FAIRE ...

Une action ou un groupe d'actions est exécuté répétitivement un certain nombre de fois : le nombre dépend des valeurs initiale et finale données à la variable « Indice ».

*Syntaxe*Pour Indice De Val1 A Val2Faire

Actions

FinFaire*Algorithme*

*Remarque* : les valeurs initiale (Val1) et finale (Val2) sont incluses dans le comptage. Il est éventuellement possible de spécifier un autre pas d'incréméntation (+2, +10, -1....)

*Exemple* : Affichage d'une ligne d'étoiles

Variable

i : entier ~ compteur de boucles ~

Début

Pour i de 1 à 80

Faire

Afficher ('\*')

FinFaire

Fin

*Remarque* : cette structure algorithmique peut en fait être remplacée par une structure TANT QUE ... FAIRE ...

Variable

i : entier ~ compteur de boucles ~

Début

i ← 1

Tant Que i ≥ 80

Faire

Afficher ('\*')

i ← i + 1

FinFaire

Fin

**Par la suite, nous nous intéresserons plus particulièrement aux structures algorithmiques du programme principal.**