*Cours Privés*

*Algorithme  
et   
Structures de Données*

*Gnénéssio Robert*

*Assistant en Informatique*

*A l'Inp Hb*

*Bp 1093 Yamoussoukro*

## SOMMAIRE

**I. INTRODUCTION**

1.1. Environnement matériel

1.2. Environnement logiciel

1.3. La programmation

1.4. Différentes approches dans la résolution d’un problème

1.5. Les étapes de résolution d’un problème

1.6. La Notion d’algorithme

1.7 Les langages de programmation

1.8. Cycle de vie d’un logiciel

**II. ACTIONS ELEMENTAIRES**

2;1; Lecture d'une donnée

2.2. Exécution d'une procédure prédéfinie

2.3. Ecriture d'un résultat

2.4. Affectation d'un nom à un objet

2.5. Déclaration d'un nom

**III. TYPES ELEMENTAIRES**

3.1. Le type entier

3.2. Le type réel

3.3. Le type booléen

3.4. Le type caractère

3.5. Constructeurs de types simples

3.6. Types article

3.7. Variable pointée ou sélection par notion ''pointée''

3.5. Structures d'articles avec variantes

3.8. Le type ensemble

**IV. EXPRESSIONS**

4.1. Evaluation

4.2. Type d'une expression

4.3. Conversion de type

**V. ENONCES STRUCTURES**

5.1. Enoncé composé

5.2. Enoncé conditionnel

**VI. PROCEDURES ET FONCTIONS**

6.1. Intérêt

6.2. Déclaration d'un sous–programme

6.3. Appel d'un sous-programme

6.4. Transmission des paramètres

6.5. Retour d'un sous-programme

6.6. Localisation

6.7. Programmation par objet

**VII. ENONCES ITERATIFS**

7.1. Forme générale

7.2. Enoncé Tant Que

7.3. L'énoncé Répéter

**VIII. LES TABLEAUX**

8.1. Déclaration d'un tableau

8.2. Dénotation d'un composant de tableau

8.3. Modification sélective

8.4. Opérations sur les tableaux

8.5. Chaîne de caractères

**IX. L'ENONCE ITERATIF Pour**

9.1. Forme générale

9.2. Forme restreinte

9.3. Complexité des algorithmes

**X. LES TABLEAUX A PLUSIEURS DIMENSIONS**

10.1. Déclaration d'un tableau à plusieurs dimensions

10.2. Dénotation d'un composant de tableau

**XI. LES FICHIERS SEQUENTIELS**

11.1. Déclaration de fichier

11.2. Manipulation des fichiers

11.3. Les fichiers de texte

**XII. RECURSIVITE**

12.1. Récursivité des actions

12.2. Récursivité des objets

**XIII. ALGORITHME DE TRI**

17.1. Introduction

17.2. Tri interne

17.3. Tris externes

I. INTRODUCTION

**1.1. Environnement matériel**

Un automate est un ensemble fini de comportements physiques **pouvant prendre des états identiques** et reproductibles en nombre fini, auquel est associé un ensemble de **changements** **d’états** **non** **instantanés** qu’il est possible de **commander** **et** **d’enchaîner** **sans** **intervention** **humaine**.

Un ordinateur est un automate déterministe à composants électroniques. Un ordinateur est muni :

- **D’une mémoire centrale** (ou principale) qui contient deux sortes d’information :

- Les informations traitantes (Instructions)

- Les informations traitées (Données).

La mémoire centrale est formée d’un ensemble de cellules ou **mots** ayant chacun une adresse unique. La représentation de l’information est faite grâce à une codification binaire (0, 1).

On appelle **Longueur** **d’un** **mot**, le nombre d’éléments binaires appelés **bits** groupés dans une simple cellule. Les longueurs de mots usuelles sont 8, 16, 32, 64 bits.

Cette mémoire possède une capacité **Finie**, souvent exprimée en Mégaoctets (Mo). Un octet est un ensemble de 8 bits, un Kilo-octet (Ko) est égal à **1024** octets, 1 Mo est égale à **1024** Ko, un Giga-octet (Go) est égale à **1024** Mo.. Les mémoires habituelles actuelles vont de **32** Mo à **512** Mo.

- **D’une unité de traitement (Unité de commande et l'Unité Arithmétique et Logique).** L’unité de commande extrait de la mémoire centrale les instructions et les données sur lesquelles portent les instructions. Elle déclenche ensuite le traitement de ces données dans l’Unité arithmétique et logique et éventuellement range le résultat dans la mémoire centrale.

- **De registres** qui sont des unités de mémorisation. Ils permettent de ranger temporairement les données pendant que l’unité centrale de traitement effectue des calculs. L’accès à ces registres est beaucoup plus rapide que l’accès à la mémoire centrale.

- **D’unités d’échanges** reliées à des périphériques pour échanger l’information avec le monde extérieur.

- **De mémoire cache**, introduite entre le processeur et les mémoires centrales pour accélérer l’accès aux données.

- **D’équipements externes** qui relient l’ordinateur au monde extérieur ; ce sont :.

- Les dispositifs de **communication (Clavier, souris, écran, imprimante, micros, hauts parleurs, scanners, etc.).**

- **Les mémoires secondaires** qui permettent de conserver les informations même quand l’ordinateur n’est plus en marche. Ce sont les disquettes, les disques durs, les CD ROM, les bandes magnétiques, les puces USB, etc.

- **Les dispositifs qui permettent** d’échanger les informations sur le réseau.

Mémoire Centrale

*Instructions*

*Données*

Registre

Unité de commande

Données

Unité arithmétique et logique

Résultats

Unité d’échanges

Périphériques

***Structure schématique d'un système informatique***

**1.2. Environnement logiciel**

Pour utiliser un ordinateur, il faut qu’il soit muni d’un système d’exploitation, c'est-à-dire, un programme ou un ensemble de programmes qui assure la gestion des ressources matérielles employées par les utilisateurs. Un système d’exploitation a pour tâches essentielles :

- La gestion et la conservation de l’information (Gestion des processus et de la mémoire centrale, système de gestion de fichiers).

- De créer l’environnement nécessaire à l’exécution d’un travail.

- De repartir les ressources entre les usagers.

Le traitement de l’information est l’exécution par l’ordinateur d’une série de commandes préparées à l’avance, le programme, et qui vise à calculer et rendre des résultats généralement en fonction des données entrées au début et au cours de l’exécution.

Chaque ordinateur est doté d’un langage appelé langage machine qui est un ensemble de commandes élémentaires représenté en code binaire.

**1.3. La programmation**

La programmation est reconnue ces dernières années comme une discipline dont la maîtrise est indispensable à la réussite de nombreux projets d’ingénierie. Les programmes peuvent être soumis à une analyse exacte basée sur un raisonnement mathématique.

Il faut reconnaître que la programmation est une activité constructive, inventive. Enseigner une telle activité peut paraître Insensé dans la mesure où le champ d’étude est vaste et varié. Mais, on peut adopter le principe de compositions élémentaires à partir de nombreux cas, puis les énoncer de façon systématique.

Avec les quelques principes élémentaires et les différents cas complexes types présentés, il revient au programmeur de s’inspirer de ces cas traités pour affronter tout autre problème qui lui sera posé.

**1.4. Différentes approches dans la résolution d’un problème**

Il faut tout d’abord faire remarquer que le but de tout programme est de calculer et de retourner des résultats valides et fiables. La conception d’un programme exige donc des méthodes rigoureuses.

Lorsque le choix des actions précède celui des objets, le problème à résoudre est décomposé en termes d’actions, en sous problèmes plus simples, eux-mêmes décomposés en d’autres sous problèmes encore plus simples. Cette méthode de construction appelée programmation descendante par raffinements successifs retarde le plus possible la représentation particulière des objets sur lesquelles portent les actions.

En revanche, lorsque le choix des objets précède celui des actions, la structure du programme est fondée sur les objets et sur les interactions. Le problème à résoudre est vu comme une modélisation (Opérationnelle) d’un aspect du monde réel constitué d’objets. Les objets sont des composantes qui contiennent des attributs (Données) et des méthodes (Actions) qui décrivent le comportement des objets.

Ces deux approches présentés reposent sur la notion d’algorithme (Nom donné à cause du mathématicien **Persan du 9ème siècle, AL – KHOWÄRIZMI).** L’algorithme décrit de façon non ambiguë, l’ordonnancement des actions à effectuer dans le temps pour spécifier une fonctionnalité à traiter de façon automatique. Il utilise un langage universel appelé le langage de description algorithmique (LDA).

Le Cahier des charges est le document qui présente le fonctionnement du système à étudier, les problèmes et les solutions souhaitées. Les algorithmes découlent donc du cahier des charges.

**1.5. Les étapes de résolution d’un problème**

Pour écrire un programme en vue de résoudre un problème quelconque, il faudra :

**- Analyser** le problème posé pour comprendre:

- L’approche à adopter (programmation descendante par raffinements successifs ou modélisation opérationnelle) ;

- Les moyens à mettre en jeu ou le cheminement à suivre pour résoudre le problème ;

- Les données qui sont consommées (Utilisées) par ce problème

- Les différentes contraintes posées (Salaire >= 250.000 ; 8 Heures par Jour)

- Traduire le résultat de l’analyse en une **synthèse d’actions** exprimées sous forme de « prédicats puis d’un algorithme »

- **Sélectionner un langage de programmation** parmi les langages présents dans l’environnement de travail ou selon les besoins exigés par les contraintes financières et, techniques (nécessité d’un langage spécialisé, contraintes financières).

Par convention toute description pragmatique (pragmat) qui est une description informelle délimité par **Pr- - - - Pr** est une forme algorithmique valable

**1.6. La Notion d’algorithme**

L'algorithme est la description statique et rigoureuse du processus de transformation de données en résultat. Il est indépendant de la machine utilisée. L’algorithme est en quelque sorte la partie raisonnement d’un problème quand le programme est lui la partie exécution (L’ordinateur abstrait) de la résolution du problème. Ceci explique l’indépendance de l’algorithme par rapport à la machine, ce qui n’est pas le cas du programme.

Une foi le programme écrit, le système d'exploitation de la machine se charge d'effectuer le séquencement automatique des traitements, c'est à dire, passer d'un enchaînement à l'enchaînement suivant. Ce qui emmène la machine à se munir de branchements conditionnels

Si condition aller à étiquette

Est réalisée ( adresse)

**Exemple** : Le calcul de moyenne

Machine

Traduit

Données

Résultat

Résultat

Données

**1.7. Les langages de programmation**

Chaque ordinateur possède un langage qui lui est propre, **le langage machine**, **qui est en général incompatible avec celui d’un autre ordinateur**. Si un programme est écrit dans le langage d’un ordinateur, il ne peut être utilisé sur un autre. Le langage le plus proche du langage machine est l’assembleur. C’est un codage alphanumérique du langage machine et est lui aussi dépendant de l’ordinateur. C’est un langage d’un niveau élémentaire (Une suite linéaire de commandes et sans structure) et dont la lisibilité est rudimentaire.

Ces défauts précités ont conduit les concepteurs à la conception de langages de programmation de hauts niveaux qui offrent aux programmeurs des moyens d’expression structurés, proches des problèmes à résoudre et améliorent la fiabilité des programmes.

La définition d’un langage de programmation recouvre trois notions fondamentales :

- La notion de **lexique** qui définit les symboles (Caractères) et mots utilisés qui servent à la rédaction des programmes ;

- La notion de **syntaxe** qui est l’ensemble des règles grammaticales qui organisent les mots en phrases ;

- La notion de **sémantique** qui étudie la signification des phrases et qui définit les règles qui donnent un sens aux phrases.

L’ensemble des règles lexicales, syntaxiques et sémantiques définit un langage de programmation. La vérification de la conformité lexicale, syntaxique et sémantique est assurée par des analyseurs appelés compilateurs et interpréteurs.

Pour exécuter un programme écrit dans un langage de haut niveau sur un ordinateur qui ne comprend que le langage machine, il faut d’abord faire :

- La traduction du programme source (Ecrit dans le langage de haut niveau) en programme sémantiquement équivalent écrit dans le langage machine de l’ordinateur appelé **compilateur**. Ce dernier exécute la traduction en deux phases :

- La vérification du lexique, l’analyse de la syntaxe et de la sémantique

- Si l’analyse ne décèle aucune anomalie alors il produit le code machine.

- L’interprétation des instructions du langage de haut niveau en stimulant le fonctionnement d’un ordinateur fictif qu’est le langage sur un ordinateur.

Compilateur

Données

Interpréteur

Résultats

Voici quelques langages de programmation de haut niveau regroupés par catégorie :

- L’Assembleur

- Les Langages machine

- Pour les applications numériques :

- FORTRAN (1954) pour des calculs spécifiquement scientifiques.

- FORTRAN II (58)

- FORTRAN IV (1962)

- FORTRAN 77 (1977)

- BASIC (1965)

- APL (1967) fait par Mr IVERSON et utilisé par IBM

- Pour des applications de gestion

- COBOL (1960)

- . Intelligence Artificielle

- LISP (1962)

- PROLOGUE (1972) (Programmation logiques) (Pas d'affectation)

. Universels

- PL1 (1964) Fortran + Cobol

- Simula (1967)

- Algol (1968) (Algorithmique Langage)

- Pascal (70/72)

- ADA (1981) créé par la volonté du Ministère de la défense des USA pour éviter d'avoir des milliers de programmes

**1.8. Cycle de vie d’un logiciel**

###### 1.8.1. Introduction

Lorsqu’il y a un problème quelconque qui est posé et que nous souhaitons une solution informatique, nous passons par les étapes suivantes : Conception ‑ Réalisation ‑ Validation ‑ Exploitation ‑ Maintenance

###### 1.8.2. Description du cycle de vie

*- . Spécifications du logiciel*

*- (description fonctionnelle, les cahiers de charges)*

*- . Conception préliminaire*

*- (décomposition en tâches des modules, construction des jeux d'essai).*

*- Conception, détaillée*

*- (Analyse de chaque module → Algorithme)*

*- choix du L.d.p*

***Conception***

***Réalisation***

***Production***

*- Codage et tests unitaire*

*- Dossier de programmation de tests*

*- . Intégration*

*- Tests de l'ensemble*

*- Dossier de tests*

***Validation***

*- Traitement d’erreurs et solutions du problème*

*-* ***Manuel d'exploitation***

*- Manuel d'utilisation*

***Exploitation***

*- Maintenance corrective*

*- Maintenance évolutive*

*- Maintenance adaptative*

***Maintenance***

###### 1.8.3. Validation = Certification:

Quelque soit le logiciel produit, on doit avoir les conditions suivantes :

- L'exploitation est sans incidents

- Le programme admet la variété des données

- Les résultats sont exacts

- Les performances sont admissibles

Pour arriver à une certification, les actions à prendre sont les suivantes :

- On prépare un jeu d'essai test

- On exécute ce programme

- On analyse les résultats

- Exécution avec sans incidents

- Performances {non} admissibles

- Résultats {non} vraisemblable

Les tests sont coûteux et peu surs. Seul l'échec est probant. En effet tant que ça marche, on a rien prouvé .Il faut un échec pour voir la versatilité d'un programme.

###### 1.8.4. Consolidation de la phase codage

Quelques conseils à respecter :

- Ne coder que les algorithmes mathématiques prouvés (par assertion)

- Utiliser les langages de hauts niveaux. Nombres d'erreurs = (nombre de lignes du programme)

- Utiliser des langages fortement typés

- Pas de déclaration par défaut

- Pas d'utilisations implicites de paramètres

II. ACTIONS ELEMENTAIRES

Un programme est une suite de commandes qui sont installées sur un ordinateur et qui effectuent des **actions** sur des données appelées objets. Il existe trois actions élémentaires ou énoncés simples :

- La lecture d'une donnée

- L'exécution d'une procédure ou d'une fonction prédéfinie

- L'écriture d'un résultat

**2.1. Lecture d'une donnée**

La lecture d'une donnée consiste à faire entrer un objet en mémoire centrale à partir d'un équipement externe. L'action de lecture sera **Lire**. Pour être reconnu sans ambiguïté, tout objet placé en mémoire centrale doit avoir un nom.

La lecture de l'objet de nom x se fera par : **Lire(x)**

Plusieurs commandes de lecture peuvent être écrites les unes après les autres. Si le même nom d'objet est utilisé, l'objet désignera la dernière valeur lue.

**2.2. Exécution d'une procédure prédéfinie**

L'exécution d'une ou d'une fonction est action élémentaire appelé appel de fonction ou de procédure. Ecrire Sinus(x) revient à faire un appel de la fonction qui calcule le sinus de l'objet x en mémoire.

Soit une fonction f, la notation f(x) sert à la fois à commander l'appel et à nommer le résultat. Une fonction peut être appelée avec plusieurs paramètres donnés. On peut avoir f(x, y, z).

Si l'on veut donner un nom au résultat pour une procédure lors de l'appel sous la forme d'un autre paramètre appelé *paramètre effectif résultat* à l'opposé du *paramètre effectif donnée.*

**2.3. Ecriture d'un résultat**

Une fois le résultat d'une procédure ou d'une fonction calculé, il est souhaitable de le récupérer sur un équipement externe. L'action élémentaire permettant cela est l'action d'écriture : **Ecrire**

L'écriture d'un objet de nom x se fera par : **Ecrire(x)**

**2.4. Affectation d'un nom à un objet**

On peut donner un nom à un objet particulier soit par une action de lecture, soit par l'intermédiaire d'un paramètre, résultat d'une procédure**. Un nom qui sert à désigner un ou plusieurs objets s'appelle une variable.**

Lorsqu'un nom, pour des raisons de fiabilité du programme désigne toujours le même objet en tout point du programme, **on l'appelle une constante**.

**Exemples d'objets** :2, 4, "Ami", "Homme", "A", et.

Une instruction d'affectation est donc une injonction de mettre à égalité. En langage algorithmique, la syntaxe est la suivante

:

Soit deux objets X et Y, affecter Y à X s'écrit **X ← Y** . Quelque soit ce que désignait X avant cette instruction, une fois exécutée, ceci signifie désormais que X désigne le même objet (Le contenu) que celui désigné par Y.

Exemple

X **← 20** signifie que X désigne l'objet 20.

**2.5. Déclaration d'un nom**

Le nom donné à une variable ou à une constante est appelé **identificateur** de ce paramètre Variable ou constante. Pour des raisons de sécurité des programmes (Pouvoir disposer des ressources nécessaires avant le démarrage de l'exécution d'un programme), la plupart des langages de programmation exigent que les noms des objets soient déclarés, tout en indiquant l'ensemble des objets désignés par ce nom. Ceci se fait à travers des commandes de déclaration.

P L'action d'associer un nom à un objet est appelée **affectation**. Pour une question de lisibilité, il faut donner des noms suffisamment longs et expressifs (Désignant généralement l'utilisation qu'on veut en faire).

###### 2.5.1. Déclaration de constante

La déclaration se fait de la manière suivante :

**Constante NomConstante = ValeurConstante**

**Exemple**:

**Constante**

**Pi = 3,14**

**Tva = 0,18**

###### 2.5.2. Déclaration de variable

La déclaration se fait de la manière suivante :

**Variable NomVariable : NomType**

ou

**Variable NomVariable Type NomType**

Le mot type introduit le domaine de valeurs de NomVariable.

**Exemple**

**Variable**

**Reponse Type Booléen**

**Racine1, Racine2 Type Réels**

Reponse est une variable qui peut désigner l'ensemble des objets booléens (Vrai ou Faux) alors que Racine1 et Racine2 désignent l'ensemble des objets réels.

III. TYPES ELEMENTAIRES

Une façon de distinguer les objets est de les classer en fonction des actions qu'on peut leur appliquer. Les classes obtenues en répertoriant les différentes actions possibles, et en mettant dans la même classe les objets qui peuvent être soumis aux mêmes actions s'appellent des types.

Il existe deux catégories de types, le type élémentaire ou simple et le type structuré. Les types élémentaires sont :

- Entier,

- Réel,

- Caractère,

- Booléen.

Alors que les types simples sont prédéfinis et connus des langages, le type structuré lui est construit par le programmeur.

Un langage est dit fortement typé si les variables sont associées à un type lors de leur déclaration.

**3.1. Le type entier**

Le type entier représente partiellement l'ensemble de entiers relatifs Z des mathématiciens. Cependant, alors que l'ensemble Z est infini, l'ensemble des valeurs défini par le type entier est fini et limité par les possibilités de chaque ordinateur. Cette limite dépend du nombre de bits utilisés pour sa représentation.

Les nombres entiers négatifs sont représentés sous forme complémentaire, soit **en complément à un**, soit **en complément à deux**. En complément à un, la représentation binaire d'un entier positif est inversée. Chaque bit est inversé.

**Exemple**

- 11 sera écrit [**01011**]

- -11 sera écrit [**10100**]

On obtient le complément à deux en ajoutant 1 au complément à un, soit [**10101**]

L’ensemble des valeurs sera défini de la façon suivante :

**- 2n-1 ≤ x ≤ 2n-1 –1 en complément à deux**

**et - 2n-1 - 1 ≤ x ≤ 2n-1 –1 en complément à un**

**n** représente le nombre de bits utilisés pour représenter un entier

Opérations possibles  **+, -, \*, Div, Mod**

- Div = Division entière

- Mod = Reste de la division entière

10 Mod 3 = 1

Le **type cardinal** représente les valeurs positives du type entier. Si x est de type cardinal, on aura

0 <= x <=2n n représentant le nombre de bits utilisés par un ordinateur pour représenter un entier.

**3.2. Le type réel**

Le type réel sert à définir partiellement l'ensemble R des mathématiciens. Les réels ne peuvent être représentés sur l'ordinateur que par des approximations plus ou moins fidèles.

**Le type entier n'est pas inclus dans le type réel**. Ce sont deux ensembles disjoints. L'arithmétique sur les réels est inexacte à cause des résultats approchés obtenus. Ces résultats dépendent de la représentation du réel sur l'ordinateur, de la précision souhaitée ainsi que de la méthode utilisée pour le calculer.

Parmi les différentes représentations du réel qui existent, la représentation en **virgule flottante** est la plus utilisée. Les caractéristiques de la représentation qui sont l'exposant **e**, la base **b** et la mantisse **m** dépendent de l'ordinateur. On écrira le réel sous la forme :

**X = + m \* be**

**X = - m \* be**

b désigne la base (2, 8, ou 16), m désigne la mantisse et e l'exposant. Ainsi, 125.32 peut s'écrire:

0.12532 \* 103

1.2532 \* 102

12.532 \* 10

On constate dans ces écritures que la base est 10 au lieu de 2, 8 ou 16 qui sont les bases utilisées pour la représentation. Ceci est une source d'erreurs.

En virgule flottante, le réel est codé sur 32 bits pour les réels simples et sur 64 bits pour les réels doubles.

Les opérateurs d'arithmétiques réelle et de relation sont applicables ( **+ - \* / +** et certaines fonctions mathématiques usuelles), **mais le test d'égalité entre deux réels devrait être évité.**

**3.3. Le type booléen**

Le type booléen est un type fini qui représente un ensemble composé de deux valeurs logiques, **Vrai** et **Faux**.

Opérations possibles

- **NON** Négation

- **ET** Conjonction

- **OU** Disjonction

- **X OU** La disjonction exclusive

et toutes les combinaisons possibles

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **P** | **Q** | **NON P** | **P ET Q** | **P OU Q** | **P X OU Q** |
| V | V | F | V | V | F |
| V | F | F | F | V | V |
| F | V | V | F | V | V |
| F | F | V | F | F | F |

**P X OU Q Le ou exclusif** Si P est vrai alors Q est faux

NON ( P OU Q) = NON P ET NON Q

NON ( P ET Q) = NON P OU NON Q

Pour représenter un booléen, un seul bit est suffisant mais on utilise un octet avec comme par convention 1 pour Vrai et 0 pour Faux. Toute comparaison renvoie un booléen.

**3.4. Le type caractère**

Chaque ordinateur possède son jeu de caractères normalisés pour représenter des lettres, des symboles et des chiffres. La représentation des caractères étaient assurée par deux normes, la norme **ASCII** codée sur 7 bits (128 caractères au maximum) et la norme **EBCDI** d'IBM codée sur 8 bits et utilisant 8 bits (256) caractères). Aujourd'hui, la norme **ISO – 8859** propose plusieurs jeux de 256 caractères codés sur 8 bits.

Le type caractère est de type ordonné, d’où l’utilisation des fonctions :

- Rang ( ),

- Caractère ( )

- Précédent()

- Suivant()

Code de a < Code de b

**3.5. Constructeurs de types simples**

Les types énumérés et intervalles permettent de construire des ensembles de valeurs particulières. L'intérêt de ces types est de pouvoir spécifier précisément le domaine de définition des variables utilisées dans le programme.

###### 3.5.1. Le type intervalle

Un type intervalle définit un intervalle de valeurs sur un type de base. Les types de base possibles sont les types élémentaires et la déclaration doit indiquer les bornes inférieures et supérieures de l'intervalle.

Ceci empêche de commettre des erreurs à l’exécution, car ces erreurs sont découvertes à la compilation.

###### Exemple

Type T = Min**..**Max

Type An = 1900**..**2003

Déclarer Année: An

###### 3.5.2. Le type énuméré

Une façon simple de construire un type est d'énumérer les éléments qui le composent. On indique le nom du type suivi entre parenthèses des valeurs de l'ensemble à construire. Ces valeurs sont des noms de constantes prises dans un même type d'éléments.

TYPE JOUR = (Lundi, Mardi, Mercredi, Jeudi, Vendredi, Samedi, Dimanche)

TYPE Couleur de carte = (Pique. Cœur, Carreau, Trèfle)

Ces Types sont ordonnés et l ordre des éléments est implicite. Ce qui donne les fonctions suivantes :

- Sucesseur( )

- Prédecesseur ( )

Soit C, 'C, couleur de cartes

C ← pique

**C' ← c + 1** impossible mais plutôt **C'←succ (C)**

**3.6. Types article**

La façon la plus générale d'obtenir des types structurés est de juxtaposer des éléments de types arbitraires, qui peuvent eux-mêmes être structurés. Ceci est nécessaire parce qu'en informatique, on a souvent besoins de décrire un individu à l’aide de certaines caractéristiques comme son nom, prénom, date de naissance, sexe, statut etc.

L’utilisation des articles vient apporter des solutions à ces problèmes non résolus par l’utilisation des types habituels.

Les types composés, comme ceux correspondants à la description de personnes ou d’objets, se trouvent souvent dans les fichiers ou les banques de données et permettent d’enregistrer les caractéristiques d’une personne ou d’un objet. C’est pourquoi le terme anglais **record** (enregistrement) s’est imposé plutôt que produit cartésien. En français on utilise le mot article. En général, un type article T dont les composants sont des types T1, T2, T3, …, Tn est défini par :

n types Ti →T= T1 \* T2 --\*Tη

**Type** complexe = **Article**

Partie réelle : réel

Partie imaginaire : réel

###### Fin article

**Type** carte à jouer = **article**

Couleur : (pique, cœur, trèfle, carreau)

Hauteur : (As, Roi, Dame)

**Fin** **article**

###### 3.6.1. Champ et sélection de champ

###### Définition : Un article est composé d'un ensemble de champs qui eux même sont composés d'un sélecteur

###### Exemple

**Type** T = Article

**Sélecteur** Ch 1: **Type** Ch 1

**Sélecteur** Ch 2: **Type** Ch 2

.

###### Fin article

**Type** Date = **Article**

Jour : Entier

Mois : (Janvier ..Décembre)

Année : Entier

###### Fin article

Comment se comporte un article ?

Si on a :

Hier, Demain, Aujourd'hui : date

Hier ← Demain (Est une instruction valable car variable de même nature)

**Si** Hier = Demain **Alors**

Instructions

###### Fin si

Est aussi une instruction valable

**3.7. Variable pointée ou sélection par notion ''pointée''**

###### Syntaxe

Variable '**.**' sélecteur

Hier**.**jour Puisque hier contient tout le type de date, nous pointons sur le jour qui nous intéresse

###### Exemple

**Ttype** individu = **article**

##### Nom : chaîne de caractère

##### Date de naissance : Date

Adresse : **Article**

Numéro : Entier

Voie : (vue, impasse, Avenue)

Nom\_de\_la\_voie : Chaîne de caractère

Code\_postal : Entier

Commune : Chaîne de caractère

**FIN**

Situation\_de\_famille : (célibataire, marié, divorcé, veuf)

###### Fin article

Si je parle d'un individu particulier (Ou instance d'un individu) x, on écrira

X : Individu

Pour parler du code postale de l'adresse de cet individu, on écrira,

X.adresse.code\_postale

**3.8. Structures d'articles avec variantes**

Il est souvent pratique de considérer que deux types ne sont au fond que deux variantes d'un même type. On peut par exemple voir un type individu pour lequel certaines informations dépendent du sexe. Pour un homme, on s'intéressera à sa barbe et à son poids. Pour une femme, le teint nous intéresse.

Soit le type sexe définit de la façon suivante :

Type Sexe = (Homme, Femme)

**Type** Individu = **Article**

Nom, Prénom : Chaîne de caractères

Naissance : Date

Sexe : (Homme, Femme)

Statut marital : (Célibataire, Marié, Veuf)

**Fin article**

Pour pouvoir inclure toutes les informations sur les hommes et les femmes, on aura :

**Type** Individu = **Article**

Nom, Prénom : Chaîne de caractères

Naissance : Date

Statut marital : (Célibataire, Marié, Veuf)

**Selon** S : Sexe **Parmi**)

Homme :

Poids : Réel

Barbu : Booléen

Femme :

Teint : (Claire, Noire)

**Fin selon**

**Fin article**

Certains langages de programmation comme Pascal et C n'admettent pas ce type de construction de structures.

**3.9. Le type ensemble**

Les valeurs possibles d'une variable de type ensemble sont des ensembles d'éléments d'un type. La syntaxe est donnée à travers les exemples suivants :

Type Véhicule = (Train, Car, Voiture, Bateau)

Type T = Ensemble de T0

Type EnsBit = Ensemble de [0..15]

Type Locomotion = Ensemble de Véhicule

IV. EXPRESSIONS

Les langages de programmation permettent de composer entre eux des opérandes et des opérateurs pour former des expressions. Les opérandes sont des identificateurs de constantes ou de variables, des appels de fonctions. Les opérateurs correspondent à des opérateurs qui portent sur un ou plusieurs opérandes.

Dans la plupart des langages, la notation utilisée suit la notation algébrique classique. Elle utilise aussi les parenthèses puisque les opérateurs habituels utilisés par ordre de priorité sont :

- La division (/ ou Div)

- La multiplication (\*)

- L'addition et la soustraction (+ et -)

**Exemple**

X \* y + z ≠ \* (y + z)

X \* y / z = x \* (y / z)

**Tableau récapitulatif des opérateurs**

|  |  |
| --- | --- |
| **Opérateurs** | **Fonctions** |
| ← | Affectation |
| = | Egalité |
| Ou | Disjonction |
| Et | Conjonction |
| <> | Différent de |
| > | Supérieur à |
| < | Inférieur à |
| <= | Inférieur ou égal |
| >= | Supérieur ou égal |
| + | Addition |
| - | Soustraction |
| \* | Multiplication |
| / | Division |
| Non | Négation |

**4.1. Evaluation**

Le but de toute expression est de calculer lors de son évaluation, un résultat. En général, l'évaluation d'une expression produit toujours le même résultat. En général, l'évaluation d'une expression produit toujours le même résultat. Le résultat d'une expression est déterminé par l'ordre d'évaluation des formules simples qui la composent. Par exemple, dans l'instruction **x ← y + z**, l'expression **y + z** est évaluée, puis le résultat est affecté à x.

Les expressions peuvent avoir un ou plusieurs opérateurs. Il est cependant judicieux de n'avoir que deux ou trois opérandes dans une expression.

**4.2. Type d'une expression**

Une expression calcule un résultat, ce dernier est typé et définit par voie de conséquence, le type de l'expression. Par exemple si P et Q sont deux booléens, l'expression P OU Q est une expression booléenne puisqu'elle produit un résultat booléen.

Grâce aux parenthèses, on peut composer des expressions manipulant des objets de types différents.

**Exemple**

Soient deux réels x et y et un booléen p

**P ← (x < y)**

**4.3. Conversion de type**

Certains objets peuvent être convertis d'un type vers un autre. Généralement, on distingue des types de conversion implicite pour lesquelles l'opérateur décide de la conversion à faire en fonction de la nature de l'opérande (Entier vers réel), et des types de conversion explicites qui sont l'œuvre des programmeurs.

V. ENONCES STRUCTURES

Les actions étudiées jusque là sont des actions élémentaires. Une action structurée est formée à partir d'autres actions qui peuvent être élémentaires ou structurées.

**5.1. Enoncé composé**

Certains langages de programmation permettent de grouper une suite d'actions comme une seule action composée. Pour cela, ces actions peuvent commencer par deux parenthéseurs **Début** et **Fin**.

**Début**

E1

E2

E3

**Fin**

Ces actions sont exécutées de façon séquentielle.

**5.2. Enoncé conditionnel**

Les langages de programmation proposent des énoncés conditionnels qui permettent d'exécuter une action ou non selon une décision prise en fonction d'un choix.

###### 5.2.1. Enoncé Choix

Lorsque l'exécution d'une action dépend d'un choix qui doit être fait parmi un ensemble d'actions possibles selon des valeurs possibles d'une expression, on utilise une instruction dite choix multiple. Elle précise l'expression dont l'évaluation fournira la valeur de l'objet discriminatoire, puis la liste des actions possibles, chacune étant précédée de la valeur correspondante. La syntaxe est la suivante:

**Choix** Expr **parmi**

Val1:

Action1

Val2:

Action2

Val3:

Action3

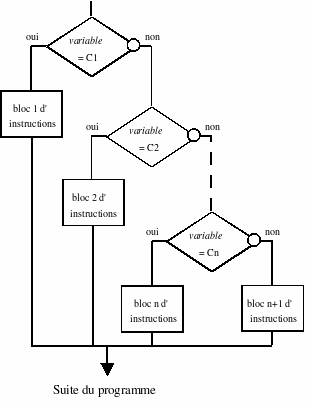
.

.

Valn:

ActionN

**FinChoix**



**Exemple**

**Choix** Couleur du feu **Parmi**

Rouge:

Pr S'arrêter pr

Vert:

Pr Passer pr

Orange:

**Si** Possible s'arrêter **alors**

Pr S'arrêter pr

**Sinon**

Pr Passer pr

**FinSi**

**finChoix**

###### 5.2.2. Enoncé SI

La syntaxe est la suivante:

**1. Si** Condition 1 **Alors**

2. suite instructions1

3. [**sinon**

**4. si** Condition2 **alors**

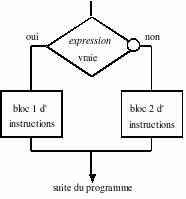
5. suite d'instructions2

**6. sinon**

7. suite d'instruction3. ] Facultatif

#### **8. Fin si**

###### 9. Fin si



**Remarque**

Le bloc facultatif, s'il est omis, on est dans une condition absolue. Dans le cas contraire, on est dans une condition alternative.

Si la condition 1 est vraie, alors suite d'instruction1 est exécutée. Dans le cas contraire, on se branche immédiatement à la ligne 4.

Si la condition 2 est vraie, on exécute la ligne 5. Dans le cas contraire, on se branche à la ligne 7.

**Exemples**

**Si** Mon salaire augmente **Alors**

Pr Je reste pr

**Sinon**

Pr je pars pr

**FinSI**

Lire 3 nombres a, b, c et afficher le min, c'est à dire, le plus petit des 3.

###### Corrigé 1

**Si** (a <= b) et (a <= c) **alors**

min ← a

**sinon**

**si** (b < c) **alors**

min ← b

**sinon**

min ← c

**fin** **si**

**fin si**

###### Corrigé 2

**Si** (a < b) **alors**

**si** (a < c) **alors**

min ← a

**sinon**

min ← c

**fin si**

**sinon**

min ← b

###### Fin si

**Corrigé 3**

min ← a

**si** min > b **alors**

min ← b

**fin si**

**si** min > c **alors**

min ← c

**fin si**

VI. PROCEDURES ET FONCTIONS

**6.1. Intérêt**

Une procédure ou une fonction permet d'associer un nom à une suite d'énoncés, et d'utiliser ce nom comme abréviation chaque fois que la suite apparaît dans le programme.

L'association d'un nom à la suite d'énoncés se fait par une déclaration de sous-programme. L'utilisation du nom à la place du sous-programme s'appelle un appel de sous-programme.

Les sous-programmes ont un rôle important dans la structuration et la localisation, le paramétrage et la lisibilité des programmes. Les principaux rôles sont :

- Ce sont des outils de structuration qui abrègent la codification et rend les programmes très lisibles;

- Ce sont des unités textuelles permettant de définir des objets locaux dont le domaine de validité sera bien défini;

- Ils permettent un paramétrage qui permet à des sous-programmes de faire plusieurs fois le même travail sur plusieurs données différentes;

- Ils améliorent la documentation des programmes.

**6.2. Déclaration d'un sous-programme**

Le rôle de la déclaration d'un sous-programme est de lier un nom unique à une suite d'énoncé sur des objets formels ne prenant des valeurs effectives qu'au moment de l'appel de ce sous-programme. Cette déclaration est toujours formée d'un en-tête et d'un corps.

**L'en-tête comprend :**

- Le nom du sous-programme

- Les paramètres formels et leur type

- Le type du résultat dans le cas d'une fonction

Exemple

**Procédure** NomProc(["Liste paramètres formels"])

**Fonction** NomFonc(["Liste paramètres formels"])

**Le corps :**

Il contient la suite d'énoncés. Il commencera par **Début** et se termine par **Fin** *(FinProc ou FinFonc)*

***PROCEDURE*** *NomProcédure(Liste de paramètres)*

*[Déclarations internes]*

###### Début

*Corps de la procédure*

###### Fin

**6.3. Appel d'un sous-programme**

L'appel d'un sous-programme est une action élémentaire qui permet l'exécution de la suite d'énoncés associés à ce nom. Il consiste simplement à indiquer le nom de la procédure ou de la fonction avec ses *paramètres effectifs.*

**NomProc**(["Liste de paramètres effectifs])

**6.4. Transmission des paramètres**

Un paramètre formel est un nom sous lequel un paramètre d'un sous-programme est connu à l'intérieur de celui-ci.

Un paramètre effectif est l'entité fournie au moment de l'appel du sous-programme.

Il existe trois types de paramètres formels :

- ***Les paramètres données*** qui fournissent des valeurs à partir desquelles les énoncés du corps du sous-programme effectuent les calculs.

**Procédure** NomProc(Param1 : TypeParam E)

Ou

**Procédure** NomProc(Données Param1 : TypeParam)

- ***Les paramètres résultats*** qui donnent les valeurs calculées par la procédure.

**Procédure** NomProc(Param1 : TypeParam S)

Ou

**Procédure** NomProc(Résultat Param1 : TypeParam)

Une procédure peut avoir un nombre quelconque de paramètres données et résultats. Une fonction, parce qu'elle renvoie un résultat unique n'aura que des paramètres données.

La correspondance entre le paramètre formel et le paramètre effectif se fait par la position. Le nombre de paramètres formels doit être égal aux paramètres effectifs

**Procédure** NomProc(Données Param1 : TypeParam1, Param2 : TypeParam2, Param3 : TypeParam3)

Appel par *NomProc(x , y , z*)

Les paramètres effectifs x, y, z correspondent respectivement aux paramètres Param1, Param2, Param3 et doivent être de même types.

###### 6.4.1. Transmission de données par valeur

Elle est utilisée pour les paramètres données et a pour effet d'affecter au paramètre formel, la valeur du résultat de l'évaluation du paramètre effectif.

**Procédure** P(Données Param : TypeParam)

Dans l'appel P(x) on effectue d'abord ***Param ← x*** avant d'exécuter la procédure

###### 6.4.2. Transmission de données par résultat

Elle est utilisée pour les paramètres résultats et a pour effet d'affecter au paramètre formel, la valeur du résultat du calcul effectué par la procédure.

**Procédure** P(Résultat Param : TypeParam)

Dans l'appel P(x) on effectue ***Param ← x*** après l'exécution de la procédure, x étant le résultat du calcul effectué par la procédure.

###### 6.4.3. Transmission de données par référence

Elle est utilisée pour les paramètres à la fois données et résultats et a pour effet d'affecter au paramètre formel, la valeur du paramètre transmis par référence et d'affecter le résultat du calcul effectué par la procédure à ce même paramètre.

**Procédure** P(Param : TypeParam E/S)

**6.5. Retour d'un sous-programme**

L'endroit où se fait l'appel d'une fonction s'appelle le contexte d'appel. Il peut être soit le programme principal, soit un autre sous-programme. Après l'exécution d'un sous-programme, l'énoncé qui suit immédiatement l'appel d'un sous-programme est exécuté.

Pour une fonction qui doit rendre un résultat, elle se termine par l'énoncé ***Rendre*** Expr ou ***Retour*** Expr ou ***NomFonc*** ***←*** Expr

***Fonction*** *NomFonc(Liste de paramètres): Type*

*[Déclarations internes]*

###### Début

*Corps de la fonction*

***Retour*** *Expr*

###### Fin

**6.6. Localisation**

Chaque fois que des objets n'ont de signification qu'à l'intérieur d'un sous-programme, ils devront être déclarés comme des ***paramètres locaux*** à l'intérieur de ce sous‑programme.

Les objets locaux démarrent leur existence au moment de l'appel du sous-programme et sont détruits au moment lors de l'achèvement du sous-programme.

Les objets déclarés à l'intérieur d'un programme principal sont dits ***globaux*** et sont significatifs pour tous les sous-programmes.

Des objets peuvent avoir les mêmes noms dans des sous-programmes différents car leur visibilité s'arrête dans le sous-programme où ils ont été déclarés.

**Exemple**

**Procédure** P1

**Variable** X Type T

**Début**

Corps procédure

**finProc**

**Procédure** P2

**Constante** X = 3,14

**Début**

Corps procédure

**finProc**

**6.7. Programmation par objet**

La programmation par objet est une méthodologie qui fonde la structure des programmes autour des objets. C'est un système d'interaction d'une collection d'objets dynamiques considérés comme des fournisseurs de services à d'autres objets qui sont les clients.

Les services offerts par un objet sont des données appelées ***attributs*** et des actions appelées ***méthodes***. Les objets possédant les mêmes propriétés sont appelées ***classe***. Une classe possède la déclaration des données et des méthodes.

**Exemple**

***Classe*** *Rectangle*

***Public*** *Périmètre, Surface, ChangerLargeur, ChangerLongueur*

*Largeur, Longueur Type Réel*

***Fonction*** *Périmètre() : Réel*

***Début***

*Retour 2 X (Largeur + Longueur)*

***Fin***

***Fonction*** *Surface() : Réel*

***Début***

*Retour Largeur X Longueur*

***Fin***

***Procédure*** *ChangerLargeur(lg : Réel E)*

***Début***

*Largeur ← lg*

***Fin***

***Procédure*** *ChangerLongueur(lg : Réel E)*

***Début***

*Longueur ← lg*

***Fin***

***FinClasse***

VII. ENONCES ITERATIFS

**7.1. Forme générale**

Les instructions itératives souvent appelées boucles sont des instructions dans lesquelles les expressions sont exécutées plusieurs fois.

E1

E2

Ei

B

Pa

On entre dans la boucle en **Pa** et les expressions **Ei** sont évaluées autant de fois que nécessaire jusqu'à ce que le prédicat d'achèvement **B** soit vérifié. Dans ce cas, on passe immédiatement à l'expression Esuite. La syntaxe générale est la suivante :

**Itérer**

Expression

**Si** B **Alors**

Pr Arrêter pr

**Finsi**

Expression

**FinItérer**

**7.2. Enoncé Tant Que**

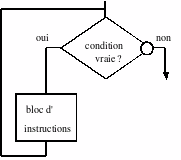
###### Syntaxe

**Tant que** Expression booléenne **Faire**

(On peut l'évaluer)

#### Instructions

**Fin Tant que**



L'instruction suivant le **FinTantQue** est exécutée si expression booléenne est évaluée à faux.

**Problème**: Est – ce que si on entre dans la boucle on en sort?

###### Généralement, avec la boucle tant que, on commence par une initialisation d'une variable.

**Tant que** Pas fini **Faire**

Traitement

**Fin** **Tant** **que**

Pr Edition résultat pr

**Fin**

**En résumé, avec la forme tant que on a toujours :**

**Tant que** expression **faire**

suite d'instructions

**fin Tant que**

**Remarque**

Il est possible de sortir de la boucle Tant Que même si Expression booléenne est vraie. Il suffit pour cela qu'une autre expression à l'intérieur de la boucle soit vérifiée pour sortir avec l'expression **Pr Sortie de boucle pr**.

**7.3. L'énoncé Répéter**

Syntaxe

###### Répéter

#### Suite d'instruction

J**usqu'à** expression

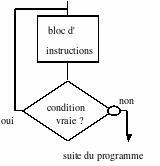
On exécute la suite d'instruction autant de fois que expression reste Faux.

###### Répéter

Lire (caract)

**Jusqu'à** caract. dans {"o", "N"}

**7.4. L'énoncé Répéter tant que (condition vraie)**



VIII. LES TABLEAUX

Un tableau est un agrégat de composants, objets élémentaires ou non, de même type et dont l'accès à ses composants se fait par un indice calculé.

**8.1. Déclaration d'un tableau**

La définition d'un tableau doit faire apparaître :

- Le type des composants

- Le nombre des indices

La déclaration d'un tableau t est de la forme suivante :

***TYPE t = Tableau[T1] de T2***

***Variable t11 : t***

- **T1** est le nombre de composants. La (ou les ) valeur indiqué (es) pour le nombre déterminent le type des indices. Ceux – ci doivent être de type ordinal (Dénombrables, positifs, ordonnés et limités). En Java, les tableaux n'ont de nombre figé. Le nombre de composants est dynamique.

- **T2** est le type des composants

Exemples

Type T1 = Tableau [Vrai, Faux] de Caractères

Variable x : T1

**TYPE** Jeu\_de\_carte = **Tableau**[32] Cartes

**TYPE** NomT = **Tableau**[80] Caractères

**8.2. Dénotation d'un composant de tableau**

On désigne un composant d'un tableau **t** d'indice i par **t[i]**.

**8.3. Modification sélective**

Le fait que les indices de tableau doivent être d’un type (Scalaire) défini a une conséquence très importante. **On peut calculer les indices**. Cependant, l’indice calculé peut être supérieur à l’intervalle des valeurs du type d’indices de tableau.

*t[vrai] ← 234;*

*t[faux] ← 0;*

*t[i + 4] ← 'a'*

**8.4. Chaîne de caractères**

Une constante chaîne de caractères se dénote entre deux guillemets pour certains langages comme le **Java** ou le **VB**, ou entre quotes pour le **C** ou **Pascal.**

*Java, VB : "Bonjour"*

*C, Pascal : 'Bonjour'*

Pour manipuler des chaînes de caractères, le C et Pascal utilisent des tableaux. Le type chaîne de caractères est un tableau de caractères.

***TYPE*** *Chaîne =* ***Tableau****[80] de Caractères*

Remarque

Le type tableau étant très utilisé dans la plupart des langages, la déclaration d'un tableau se fait comme des variables simples.

***Variable*** *Chaîne :* ***Tableau****[80] de Caractères*

IX. L'ENONCE ITERATIF Pour

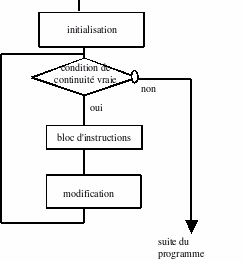
**9.1. Forme générale**

Il arrive qu'on veuille faire le même traitement un nombre fixe de fois. Pour ce faire, beaucoup de langages de programmation proposent une construction utilisant l'énoncé itératif "***Pour***". La syntaxe générale est la suivante :

***Pour*** *x* ***de*** *T* ***Faire***

*Expressions*

***FinPour***



x représente une variable de boucle qui prendra successivement toutes les valeurs du type T, et exécutera pour chacune d'entre elles, l'énoncé de l'expression (le traitement).

**9.2. Forme restreinte**

***Pour*** *x* ***allant de*** *Min à Max Par PAsDe* ***Faire***

*Expressions*

***FinPour***

T comprend les éléments suivants :

- ***Min*** : La valeur initiale de x lors du premier passage dans la boucle

- ***Max*** : La valeur finale de x

- ***PAsDe :*** L'incrémentation (ou pas) si différente de 1

Exemple

Ordonner dans l'ordre croissant, une liste d'éléments saisis dans un tableau. Par exemple ***53 914 827 302 631 785 230 11 567 350*** doivent être triés pour devenir ***11 53 230 567 631 785 827 914***

***Algorithme*** *TriParSélection*

*/\* Tri par sélection dans l'ordre croissant des n valeurs d'un tableau Tb \*/*

***Constant*** *n = Val1*

***Variable*** *Min, i, j : Entier*

***Début***

***Pour*** *i allant de 1 à n-1* ***Faire***

*Min ← 1*

***Pour*** *J allant de i + 1 à n* ***Faire***

***Si*** *Tb[j] < Tb[Min]* ***Alors***

*Min ← j*

***FinSi***

***FinPour***

*Tb[j] ← Tb[Min]*

***FinPour***

***Fin***

X. LES TABLEAUX A PLUSIEURS DIMENSIONS

Les composants d’un tableau peuvent être de type quelconque ou de type tableau. Les tableaux de tableaux sont souvent appelés *tableaux à plusieurs dimensions*. Bien que le nombre de la dimension ne soit pas limité, mais dans la pratique il ne dépasse guerre deux dans les programmes.

**10.1. Déclaration d'un tableau à plusieurs dimensions**

La déclaration d'un tableau à deux dimensions possède la forme suivante :

***TYPE*** *NomTableau1 =* ***Tableau****[T1, T2] de Type*

***TYPE*** *NomTableau2 =* ***Tableau****[1..T1, 1..T2] de Type*

*Moyenne[1,5] ← 14,25*

*Moyenne[1,5] désigne la moyenne de la 5ème matière pour le 1er étudiant.*

**10.2. Dénotation d'un composant de tableau**

*Moyenne[1,5] ← 14,25*

*Moyenne[1,5] désigne la moyenne de la 5ème matière pour le 1er étudiant.*