Université Mohammed Seddik Ben yahia- Jijel
Faculté des Sciences Exactes et Informatique
Département Informatique

2018

Support de Cours

Introduction à la Programmation

Spécialité

Première Année et Deuxième Année

Licence en Informatique

Auteur

Dr. KHALFAOUI KHALED

Ce cours est destiné aux étudiants de licence en informatique. Il expose les concepts de base de la programmation structurée et donne une introduction à la programmation orientée objets. Afin de mettre en pratique les différentes notions abordées, une série d'exercices variée est proposée.

Département d'informatique, université de Jijel BP 98 Ouled Aissa. 18000, JIJEL, ALGERIE

Sommaire

Chapitre 4 : Programmation structurée

Chapitre 5 : Programmation orientée objets

Travaux dirigés

Chapitre 01 :

Notions de base

1.1 Définitions :

Informatique :

C'est la science du traitement de l'information. A sa motivation initiale qui était de faciliter et d'accélérer les calculs scientifiques, se sont ajoutées de nombreuses fonctionnalités telles que la bureautique, la télécommunication, la gestion des entreprises,.....Etc.

On définit aussi l'informatique comme étant l'ensemble des applications mettant en œuvre deux éléments distincts mais indissociables :

- **Le hardware** : qui est l'ensemble du matériel constitutif de l'ordinateur et de ses périphériques.
- **Le software** : qui comprend l'ensemble des programmes et des logiciels.

L'ensemble des moyens logiciels (software) et matériels (hardware) nécessaires pour satisfaire les besoins informatiques des utilisateurs constitue ce qu'on appelle système informatique [1].

Programme :

Un programme informatique est une liste d'ordres indiquant à un ordinateur ce qu'il devrait faire. Il se présente sous la forme de plusieurs séquences d'instructions, devant être exécutées dans un certain ordre par un processeur .

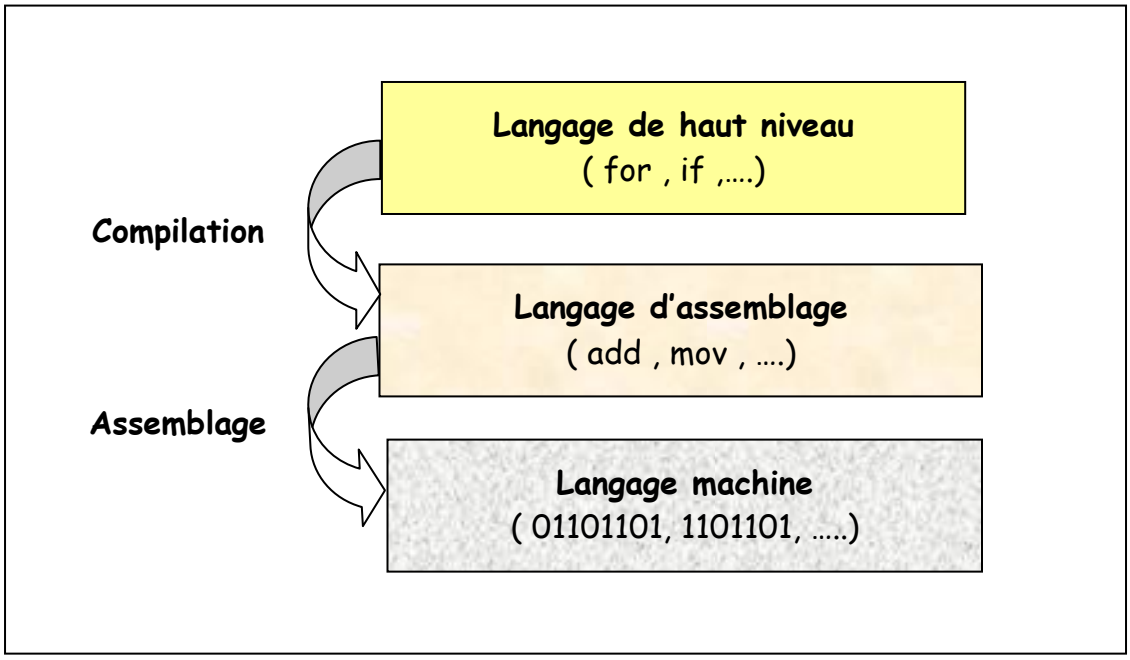
A l'origine d'un programme, il y a un code source écrit par un programmeur dans un langage de programmation. Ce code est ensuite traduit avec un jeu d'instructions spécifique au microprocesseur (langage machine) par un compilateur. Le programme obtenu peut alors être exécuté directement par l'ordinateur [2].

L'ensemble des programmes informatiques peut être reparti en deux sous ensembles :

- Les programmes de base : Les programmes de base sont des programmes conçus pour pouvoir gérer les éléments physiques de l'ordinateur. Ils constituent la couche logicielle de base qui s'intercale toujours entre l'utilisateur et le matériel. C'est les systèmes d'exploitation.
- Les programmes d'application : Pour ses besoins spécifiques, l'utilisateur utilise des programmes particuliers dits programmes d'application. Prenons comme exemple :
    - Les logiciels de la bureautique : Traitement de texte, …etc.
    - Les logiciels de messagerie et communication via un réseau, Internet,… etc.

1.2 Langages d'assemblage et langages évolués :

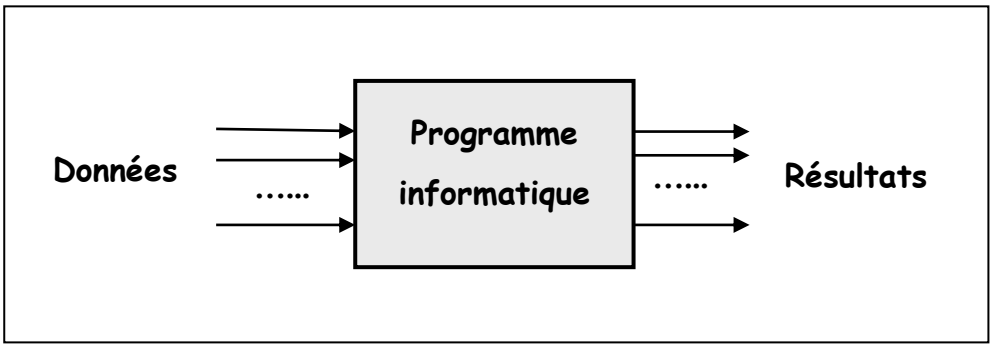
Au début de l'informatique, on a développé des langages de «bas niveau» appelés «langages d'assemblage». Pour exécuter un programme écrit dans un langage d'assemblage, il doit être traduit en un programme équivalent écrit en langage machine. Cette traduction s'effectue grâce à un programme appelé «Assembleur». Mais programmer en Assembleur n'est ni très simple ni très agréable car ça qui nécessite la connaissance de l'architecture matérielle de l'ordinateur sur lequel il s'exécutera ; il y a ainsi presque autant de langages d'assemblage différents que de microprocesseurs. Le problème est donc de définir une langue traduisible en Assembleur mais plus lisible et plus compréhensible par l'homme. De nouveaux langages de programmation ont peu à peu été définis. La traduction est réalisée par ce qu'on appelle les *compilateurs*. Un compilateur est un logiciel capable de transformer un programme écrit dans un langage de programmation donné en un programme réalisant le même traitement mais écrit dans un langage d'assemblage [1].



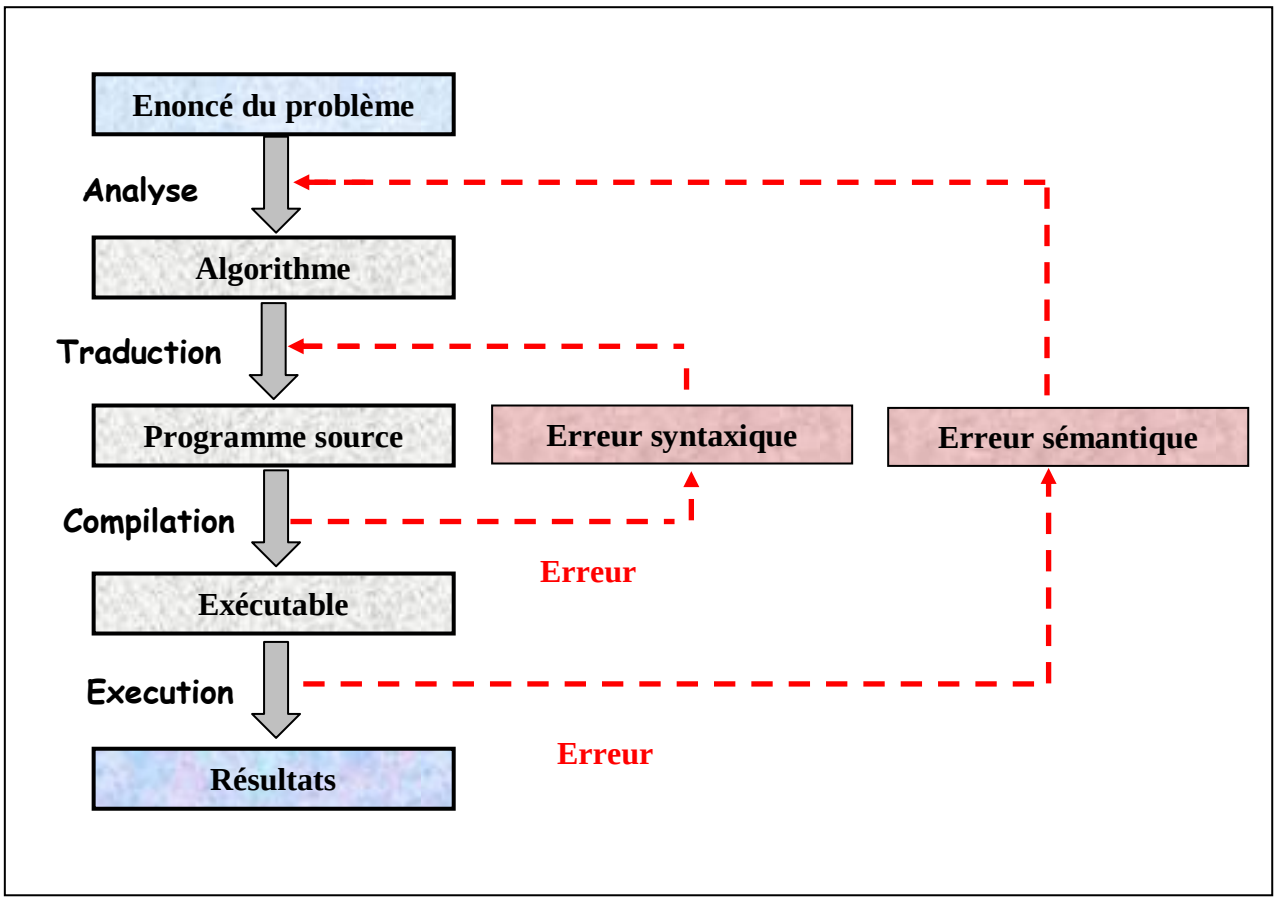
L'utilisation de compilateurs a permis la définition de langages de programmation de «haut niveau» ou «évolués». Dans de tels langages, on peut s'abstraire de la connaissance matérielle de l'ordinateur sur lequel s'exécutera le programme pour se concentrer sur sa seule logique. Pascal, C, et Java sont des langages de ce type.

1.3 Démarche de Conception d'un programme:

Le traitement de l'information par ordinateur consiste à faire élaborer, par cette machine, des informations appelées résultats, à partir d'informations appelées données.



Pour ce faire, la démarche de conception d'un programme est la suivante :



La première étape de développement consiste à bien analyser le problème. Le résultat de cette analyse est appelé algorithme. Il s'agit d'une description des étapes à suivre pour résoudre un problème bien particulier. Il permet d'expliciter clairement les idées de la solution d'un problème indépendamment d'un langage de programmation.

Dans ce schéma, les phases d'Analyse et de Traduction sont des opérations intellectuelles humaines. Par contre, les phases de Compilation et d'Exécution sont réalisées automatiquement par la machine.

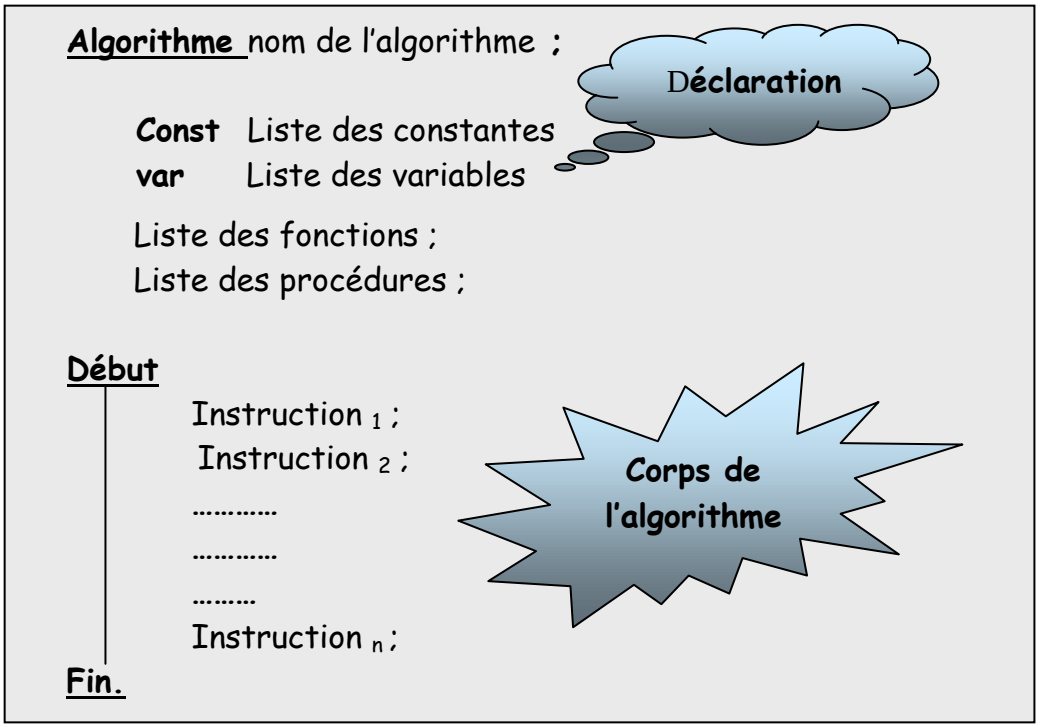
Remarques :

- Un programme source est la traduction d'un algorithme dans un langage accepté par la machine (langage de programmation).
- Une erreur syntaxique est une faute d'écriture (l'équivalent d'une faute d'orthographe ou de grammaire en français) ; elle est signalée par le compilateur quand celui-ci ne sait pas comment interpréter certaines instructions du programme source afin de les traduire dans le langage cible (Assembleur).
- Une erreur sémantique, beaucoup plus grave, provient d'une mauvaise conception de la méthode suivie pour résoudre le problème. C'est alors l'algorithme lui-même qui doit être modifié, ce qui nous oblige à refaire la phase d'Analyse.

1.4 langage algorithmique

Comme vu précédemment, L'utilisateur d'un algorithme n'aura qu'à suivre toutes les instructions, dans l'ordre (en séquence) pour arriver au résultat recherché.

Le "langage algorithmique" est un pseudo langage qui tient compte des caractéristiques de la machine, tout en étant plus souple qu'un langage de programmation. C'est donc un compromis entre un langage naturel et un langage de programmation. La structure générale d'un algorithme est la suivante :



- **Les déclarations :** c'est une liste de tous les objets utilisés et manipulés dans le corps de l'algorithme.
- **Le corps :** dans cette partie sont placées les opérations et les instructions à exécuter.

Chapitre 02 :

Types et instructions élémentaires

2.1 Variables et constantes :

Une **variable** est un espace mémoire identifié par un nom et destiné à stocker une valeur qui peut être modifiée durant les traitements. Les variables d'un algorithme contiennent les informations nécessaires à son déroulement. La variable diffère de la notion de **constante** qui, comme son nom l'indique, ne prend qu'une seule valeur au cours d'exécution de l'algorithme [3] .

Chaque variable est caractérisée par un nom (identifiant), un type et une valeur.

- **L'identificateur :** c'est un nom qui respecte une syntaxe particulière. Il est constitué d'une suite de lettres de l'alphabet et de chiffres commençant par une lettre, il peut aussi contenir un trait d'union"_". Exemple : x, y, PI, rayon,…….
  - Pour faciliter la lisibilité d'un algorithme, il est préférable d'utiliser des noms significatifs.
  - L'identificateur doit être différent de tous les mots clés.
- **Le type :** Le type correspond au genre d'information que l'on souhaite utiliser. Il peut être entier, réel, booléen, caractère, chaîne,…….ect.
  - Il faut noter qu'à un type donné, correspond un ensemble d'opérations définies pour ce type.

Remarques :

- Pour déclarer une variable on définit son nom et son type.
- Pour déclarer une constante on définit son nom et sa valeur.
- Une variable peut être simple ou structurée selon qu'elle représente une valeur unique manipulée globalement comme les valeurs entières (réelles) ou qu'elle regroupe plusieurs valeurs pouvant être manipulées séparément comme un nombre complexe qui est représenté par deux valeurs : partie réelle et partie imaginaire.

2.2 Types scalaires:

Un type scalaire encore appelé élémentaire définit un ensemble de valeurs simples et ordonnées. Tout langage de programmation offre un certain nombre de types standard qui sont des types préalablement définis.

Le type entier : Il est utilisé pour manipuler des valeurs entières : 1, 100, -5,……. Le type entier est muni des opérateurs suivants :

- Les opérateurs arithmétiques classiques : + (addition), - (soustraction), * (produit).
- la division entière, notée div , telle que : n div p donne la partie entière du quotient de la division entière de n par p .
- le modulo, noté mod, telle que : n mod p donne le reste de la division entière de n par p.
- les opérateurs de comparaison classiques : <, >, =, ...

D'autres opérations sur le type entier sont prédéfinies par des fonctions :

- sqr (n) : fonction qui fournit le carré d'un entier n.
- abs (n) : fonction qui fournit la valeur absolue d'un entier n.
- succ (n) : fonction qui fournit l'entier de valeur n+1.
- Pred (n) : fonction qui fournit l'entier de valeur n-1.

Le type réel : Il est utilisé pour manipuler des nombres réels : 10.2, 15.7, -4.8,……. Les opérations valides sur les réels sont :

- les opérations arithmétiques classiques : + (addition), - (soustraction), * (produit) et / (division).
- les opérateurs de comparaison classiques : <, >, =, ...

D'autres opérations sur le type entier sont prédéfinies par des fonctions :

- Les fonctions trigonométriques : sin, cos, arctg,…..
- La fonction carrée (sqr).
- La fonction qui fournit la racine carrée (sqrt).
- La fonction qui fournit la valeur absolue (abs).
- La fonction qui fournit la partie entière d'un réel (trunc).
- La fonction qui renvoie l'entier le plus proche d'un réel (round).

Remarques :

- On peut affecter à une variable de type réel une autre variable de type entier. C'est la machine qui prend en charge la conversion de la valeur entière en valeur réelle.
- Les variables de type entier et les variables de type réel peuvent être combinées dans des opérations définies sur les réels. Les entiers sont alors automatiquement convertis en réels.

Le type caractère : l'ensemble des valeurs de type caractère est un jeu fini et totalement ordonné de caractères. Il comporte :

- Les lettres de l'alphabet latin.
- Les chiffres de 0 à 9.
- Les symboles utilisés en tant qu'opérateurs : +,-,<,….
- Les caractères spéciaux : @,%,…..
- Et d'autres.

Une constante caractère est représentée par un et un seul caractère encadré par deux apostrophes. exp : 'a','Z',…….
Les opérations prédéfinies sur les caractères sont :

- Ord(x) : elle renvoie un entier positif correspond au rang du caractère c, dans le jeu totalement ordonné de caractères. C'est le code du caractère c.
- Chr (i) : c'est la fonction inverse de ord. pour un argument entier positif, elle renvoie le caractère de rang i.
- Succ(c) : elle fournit le caractère qui suit immédiatement le caractère c dans le jeu de caractères.
- Pred(c) : fournit le caractère qui précède immédiatement le caractère c dans le jeu de caractère.

Remarques :

- La relation d'ordre est définie sur le type caractère telle que :
  - ✓ Si x et y sont deux variables de type caractère alors :

    'x' > 'y' si ord ('x') > ord ('y').
- Il ne faut pas confondre entre la constante numérique 5 et la constante caractère '5'.
- Tous les caractères respectent les principes suivants :
  - ✓ Les caractères alphabétiques (majuscules et minuscules) se suivent et sont ordonnés dans l'ordre alphabétique.
  - ✓ Les caractères numériques se suivent et sont ordonnés dans l'ordre croissant.

Le type chaîne : Une chaîne est une suite de caractères. Une chaîne est encadrée par deux apostrophes. exp : "informatique", " cour", "Jijel",…..

- On appelle longueur d'une chaîne le nombre de caractères de cette chaîne. "Informatique" est une chaîne de longueur 12.

Les fonctions prédéfinies sur les chaînes sont :

- Length(c) : elle fournit la longueur de la chaîne c.
- Concat (c1, c2) : elle fournit la chaîne obtenue par concaténation des deux chaînes c1 et c2.

Les opérateurs de relation sont définis sur le type chaîne tel que :

- Deux chaînes sont égales si elles sont identiques.
- Les chaînes sont ordonnées selon l'ordre lexicographique.

Le type booléen : Il s'agit du domaine dont les seules valeurs sont *vrai* ou *faux*. Les opérateurs logiques sont définis par les tables suivantes appelées tables de vérité.

Et	*vrai*	*faux*
vrai	vrai	faux
faux	faux	faux

Ou	*vrai*	*faux*
vrai	vrai	vrai
faux	vrai	faux

Non	
vrai	faux
faux	vrai

2.3 Types structurés :

Une variable de type structuré regroupe sous un même nom plusieurs valeurs appartenant à des types constituants déjà définis. Ces types constituants peuvent à leur tour être structurés [4].

Les tableaux: Un tableau est un ensemble de données qui sont toutes du même type et qui possèdent un identificateur unique (nom du tableau) et se différencient les unes des autres dans ce tableau par leur numéro d'indice.

Exemple : tableau de notes, tableau de températures d'un mois.

- **Déclaration :** Pour déclarer un tableau il faut lui donner un nom, une valeur d'indice minimale et une valeur d'indice maximale. Il faut déclarer également l'indice qui permet d'adresser les différentes cases. L'indice doit être du type entier.

  Nom du tableau : **tableau [**indice minimal.. indice maximal**] de** type de données.

  Exemple : Notes : tableau [1..10] de réel ;

  Dans cet exemple, on a déclaré un tableau appelé Notes d'indice minimal 1 et d'indice maximal 10 pouvant donc contenir 10 valeurs de type réel.

  D'un point de vue pratique, lors de l'étape d'analyse, on représente un tableau par un ensemble de cases repérées par leurs indices (leurs positions dans le tableau).

  Notes

12.5	18.0	10.0	9.5						14.5
1	2	3	4						10

- **Accès aux éléments :** On accède à un élément d'une variable Tab de type tableau de la manière suivante : Tab [Exp]

  Tel que : Exp est une expression qui détermine le rang de l'élément sélectionné dans le tableau.

> **Exemple :** Notes [5] : désigne l'élément de rang 5 dans le tableau Notes.

Remarque : Le nom d'un tableau n'est jamais utilisé seul. Dans toutes les instructions, il est toujours suivi d'un indice entre crochés.

Les Matrices : Lorsqu'un traitement utilise plusieurs tableaux à une dimension ayant le même nombre d'éléments et subissent le même traitement, on utilise un seul tableau à deux dimension appelé matrice. Cette nouvelle forme de tableau possède un identifiant unique. Chaque élément est identifié par deux indices de type entier. Le premier spécifie la ligne, alors que le deuxième la colonne.

- **Déclaration :** Un tableau à deux dimensions est déclaré comme suit :

  Nom du tableau : **tableau [**indice ligne minimal.. indice ligne maximal,
  indice colonne minimal.. indice colonne maximal**] de** type de données.

> **Exemple :** Notes : tableau [1..100,1..10] de réel.
> Dans cet exemple, on a déclaré un tableau appelé Notes de 100 lignes et de 10 colonnes contenant des valeurs réelles.

Notes

	1	2	3				10
1							
2							
100							

- **Accès aux éléments :** On accède à un élément d'une variable Tab de type matrice de la manière suivante : Tab [Exp1,Exp2]
  - Exp1 : expression qui démine le numéro de ligne l'élément sélectionné.
  - Exp2 : expression qui démine le numéro de colonne l'élément sélectionné.

> **Exemple :** Notes [5,3] désigne l'élément qui se trouve dans la 5 ème ligne et la 3 ème colonne dans le tableau Notes.

Remarques :

- Le nom d'une matrice n'est jamais utilisé seul. Dans toutes les instructions, il est toujours suivi de deux indices entre crochés séparés par une virgule.
- les tableaux les plus fréquemment utilisés sont les tableaux à une et à deux dimensions, mais il est tout a fait possible de définir des tableaux à trois ou quatre dimensions, voir plus.

Les Enregistrements: Un enregistrement est une structure de données permet de regrouper dans un même type un ensemble de données de types différents associées à un même et seul objet. Il comporte des composants appelés champs. Chacun des champs est identifié par un nom qui permet d'y accéder directement et un type. Le type d'un champs peut être simple ou structuré.

- **Déclaration :** La déclaration d'une structure de type enregistrement se fait de la manière suivante :

```
Type
    Nom de l'enregistrement = enregistrement
                              Nom_champs 1 : type1 ;
                              Nom_champs 2 : type2 ;
                              ……
                              Nom_champs N : typeN ;
    Fin
```

- Nom_champs $_1$, Nom_champs$_2$,…, Nom_champs $_N$: Les identificateurs des champs de l'enregistrement.
- Type$_1$, Type$_2$,…,Type$_N$: types associés aux champs.

Une fois le type défini, on peut déclarer des variables de ce type :

Nom de la variable : Nom de l'enregistrement ;

Remarque: L'ordre de déclaration des champs est quelconque.

Exemple :

```
Type
Etudiant  = enregistrement
                    Nom : chaine ;
                    Prénom : chaine ;
                    Spécialité : chaine ;
                    Année : entier ;
                    Moyenne : réel ;
Fin

Var   Etudiant_01 : Etudiant.
```

- **Accès aux champs d'un enregistrement :** On accède à chaque information en précisant le nom de la variable de type enregistrement suivie de l'identificateur du champ séparé par un point :

  Nom de la variable. Nom du champ.

> **Exemple :** Etudiant_01.Nom désigne le champs Nom de l'enregistrement Etudiant_01.

2.4 Expressions :

Une expression désigne une valeur représentée par une combinaison d'opérandes et d'opérations effectuées sur ces opérandes. A toute expression est associé un type qui est le type de la valeur de cette expression.

> **Exemple :** Nbr + 5 est une expression qui représente une opération d'addition portant sur les opérandes Nbr et 5.

Les opérateurs sont :

- Les opérateurs algébriques : +,-,*, /, div, mod.
- Les opérateurs logiques : et, ou non.
- Les opérateurs relationnels :<,<=,>,>=,=,<>.

Un opérateur qui s'applique sur deux opérandes est dit binaire alors qu'un opérateur qui s'applique sur une seule opérande est dit unaire.

Règles d'évaluation d'une expression :

En absence de parenthèses, pour éviter toute ambiguïté, des règles d'évaluation ont été établies. Il existe un ordre de priorité entre les opérateurs de façon décroissante comme suit :

- Opérateurs unaire : +,-, non.
- Opérateurs multiplicatifs : *, /, div, mod, et.
- Opérateurs additifs : +,-, ou.
- Opérateurs relationnels : <, <=,>,>=, <>.

Remarques :

- Si une expression comporte des opérateurs de même priorité, alors les opérateurs de même priorité sont associatifs à gauche.
- Dans le cas où on souhaite modifier la sémantique définie par ces règles, il faut introduire des parenthèses.
- Pour éviter toute ambiguïté, il faut toujours introduire les parenthèses.

2.5 Les instructions élémentaires :

L'affectation : L'affectation est une instruction qui stocke la valeur d'une expression dans une variable.

Syntaxe :

X← exp ; x reçoit exp.
X : variable, ← l'opérateur d'affectation, exp : expression.

Ce qui revient à remplacer la valeur initiale de x par la valeur de l'expression exp.
Exemples : x← 5 ; y← x*3 ; x← y-2 ;
Remarque :

- Une opération de contrôle de type est effectuée avant l'affectation. C'est une erreur si la valeur de l'expression n'appartient pas au type de la variable, à moins que ce soit une valeur entière et que la variable est de type réel. Dans ce cas la valeur entière est convertie en réel.

Les instructions d'appel des procédures d'entrée /sortie :

Un programme effectue un traitement sur les données et fournit des résultats. Donc, pendant l'exécution une interaction doit être établie entre la machine et l'utilisateur afin de permettre l'échange de toutes ces informations. Pour ce faire, on dispose de deux procédures standard de communication :

- **L'instruction de lecture :** Cette instruction permet la lecture de la valeur d'une variable à partir du clavier.

Syntaxe :

Lire (variable) .

On peut regrouper plusieurs ordres de lecture en un seul. Par exemple, la suite d'instructions :

Lire (A) ; Lire (B) ; Lire (C) ; Peut s'écrire :
Lire (A, B, C) ;

Lorsque le processeur reçoit un ordre Lire (variable) il interrompe l'exécution du programme et se met en état d'attente d'une valeur. L'utilisateur doit alors saisir une valeur par l'intermédiaire du clavier. Dès la validation de la valeur saisie, l'exécution du programme se poursuit. Cette valeur est affectée à la variable en écrasant la valeur précédente.

Remarque : L'instruction Lire (variable) provoque une erreur si la valeur saisie n'appartient pas au type de la variable, à moins que se soit une valeur entière et que la variable est de type réel. Auquel cas la valeur entière est convertie en une valeur réelle.

- **L'instruction d'écriture :** Cette instruction permet l'affichage sur écran. Il y a deux types d'affichage :
  - Soit on affiche la valeur d'une variable :

Syntaxe : Ecrire (variable)

  - Soit on affiche du texte :

Syntaxe : Ecrire (" texte")

On peut regrouper plusieurs ordres d'écriture en un seul. Par exemple, la suite d'instructions :

Ecrire (" texte") ; Ecrire (Y) ; Ecrire (Z) ;

Peut s'écrire :

Ecrire (" texte", Y, Z) ;

Exemples :

- Ecrire (2*x+5) : permet d'afficher la valeur de l'expression 2*x+5. Si x vaut 10 alors cette instruction affiche 25.
- Ecrire (" Le résultat est : ") : permet d'afficher cette chaîne de caractères.
- Ecrire (" Le résultat est : ", X) : permet d'afficher la chaîne Le résultat est : suivie par la valeur de x. si x vaut 17 cette instruction affiche : Le résultat est : 17.

La communication par messages est très utile en programmation. Elle offre la possibilité :

- ✓ D'orienter l'utilisateur en lui indiquant ce que la machine attend de lui suite à un ordre de lecture.
- ✓ d'expliquer les résultats d'un traitement.

D'une manière générale, pour construire un algorithme on doit suivre trois étapes :

- Introduire les données.
- Résoudre le problème : c'est manipuler ces données pour obtenir la solution au problème donné. Cette manipulation se diversifie selon le problème posé.
- Affichage des résultats.

Exemple :

```
Algorithme Permutation ;

Var X, Y, Z : réel ;

Début
        Lire (A) ;
        Lire (B) ;
        Z ←X ;   X← Y ;   Y ← Z ;
        Ecrire (A) ;
        Ecrire (B) ;
Fin.
```

Chapitre 03 :

Structures de contrôle

Les **structures de contrôle** encore appelées **instructions structurées** permettent d'exprimer la façon dont s'enchaînent les instructions d'un algorithme. Trois structures fondamentales : La séquence, la sélection et l'itération sont suffisantes pour exprimer tous les enchaînement possibles dans un algorithme.

3.1 La séquence :

La séquence exprime l'enchaînement le plus élémentaire qui est l'enchaînement séquentiel des instructions. Elle se présente sous forme d'une suite ordonnée d'instructions regroupée en un bloc.

Syntaxe :

```
Intruction1 ;
Intruction2 ;
Intruction3 ;
……
……
IntructionN ;
```

Sémantique: L'exécution d'une séquence provoque l'exécution successive de toutes les instructions dans l'ordre spécifié.

3.2 La sélection:

La sélection exprime un enchaînement conditionnel sélectif. Elle offre la possibilité de sélectionner la prochaine séquence à exécuter pendant l'exécution. La sélection se fait sur la base d'une condition. Deux schémas permettent de formaliser cette structure.

L'alternative complète : L'alternative complète permet de sélectionner sur la base d'une condition une séquence d'instructions à exécuter parmi deux séquences.

Syntaxe :

```
Si   (Condition)   alors
|          Bloc1
Sinon
|          Bloc 2
Fsi
```

- La condition est formulée par une expression booléenne.

- **Sémantique:** L'exécution de ce schéma commence par l'évaluation de la condition. Si elle est vérifiée seulement le bloc1 est exécuté sinon c'est le bloc2 qui est exécuté.

Exemple : Ecrire un programme qui permet de lire deux entiers et afficher le minimum des deux.

```
Algorithme Minimum ;

Var A, B, Min : entier ;

Début
|          Lire (A) ; Lire (B) ;
|          Si   ( A < B )   alors
|          |          Min = A ;
|          Sinon
|          |          Min = B ;
|          Fsi
|          Ecrire (Min) ;
Fin.
```

L'alternative réduite : L'alternative réduite exprime le cas ou l'exécution d'une séquence d'instructions est conditionnée par la réalisation d'une condition.

Syntaxe :

```
Si   (Condition)   alors
|          Bloc
Fsi
```

- La condition est formulée par une expression booléenne.

- **Sémantique:** L'expression booléenne représentant la condition est évaluée. Le bloc d'instructions n'est exécuté que si cette expression a pour valeur vraie, sinon il sera ignoré.

Exemple : Soit N un entier donné. Afficher la valeur absolue de N.

```
Algorithme Valeur_Absolue ;

Var  N, Val_Abs : entier ;

Début
|          Lire (N) ;
|          Val_Abs = N ;
|          Si   ( Val_Abs  < 0 )  alors
|          |        Val_Abs = - Val_Abs ;
|          Fsi
|          Ecrire (Val_Abs ) ;
Fin.
```

3.3 L'itération :

On appelle itération toute répétition de l'exécution d'un traitement décrit par une séquence d'instructions. A la notion d'itération est associée la notion de boucle. On appelle corps de la boucle le bloc d'instructions à répéter. Le nombre d'itérations doit bien sur être fini. Il est contrôlé soit par une condition soit par un compteur.

Itération contrôlée par une condition : Il existe deux schémas itératifs contrôlés par une condition :

- **Le schéma répéter :** Ce schéma permet de répéter l'exécution d'un bloc d'instructions jusqu'à ce qu'une condition soit vérifiée.

Syntaxe :

```
Répéter
    Bloc
Jusqu'à (Condition)
```

- La condition est formulée par une expression booléenne.

- **Sémantique :** La réalisation de ce schéma provoque successivement :
  - ✓ L'exécution du corps de la boucle.
  - ✓ Evaluation de la condition.

  De manière répétitive jusqu'à ce que la condition soit vérifiée.

Remarques :

- ❖ Avec ce schéma, le bloc d'instructions qui forme le corps de la boucle est exécuté au moins une fois. La première exécution n'est soumise à aucune condition.
- ❖ L'expression booléenne qui définie la condition d'arrêt de l'itération est appelée expression de contrôle. La formulation de cette expression doit permettre l'arrêt de l'itération. Dans le cas contraire, l'algorithme boucle infiniment à l'exécution.

Exemple : Afficher tous les multiples de N qui sont inférieurs à un nombre m donné.

```
Algorithme Multiples ;

Var  N, M , Multiple : entier ;

Début
        Lire (N) ; Lire (M);
        Multiple = N ;
        Répéter
            Ecrire (Multiple) ;
            Multiple = Multiple + N ;
        Jusqu'à ( Multiple > M )
Fin.
```

- **Le schéma tant que :** Avec le schéma répéter toute exécution du corps de la boucle sauf la première est soumise à condition. Le schéma tant que est un schéma plus général dans lequel toute exécution du corps de la boucle même la première est soumise à condition.

Syntaxe :

```
Tq (Condition) faire
|     Bloc
FTq
```

- La condition est formulée par une expression booléenne.

- **Sémantique :** La réalisation de ce schéma provoque successivement et de manière répétitive :
  - ✓ L'évaluation de la condition.
  - ✓ L'exécution éventuelle (si la condition est satisfaite) du corps de la boule (séquence d'instructions).

  Jusqu'à ce que la condition soit fausse.

Remarques :

- ❖ Avec ce schéma le bloc d'instructions qui forme le corps de la boucle peut ne jamais être exécuté (cas ou la condition est initialement fausse).
- ❖ Comme pour le schéma répéter la formulation de l'expression de contrôle (booléenne) doit permettre l'arrêt de l'itération. Dans le cas contraire l'algorithme boucle infiniment.

Exemple :

```
Algorithme Multiples ;

Var  N, M , Multiple : entier ;

Début
|          Lire (N) ; Lire (M);
|          Multiple = N ;
|          Tq  ( Multiple < M ) Faire
|          |     Ecrire (Multiple) ;
|          |     Multiple = Multiple + N ;
|          FTq
Fin.
```

Itération contrôlée par un compteur :

Le schéma de répétition pour utilise une variable appelée variable de contrôle, ou encore compteur d'itérations pour contrôler le nombre d'itérations de la séquence d'instructions.

Syntaxe :

Pour i = ValInit à ValFin **pas** = n **faire**

Bloc

FPour

- **Sémantique** : Dans ce schéma le nombre d'itérations du bloc d'instructions qui forme le corps de la boucle est déminé par :
  - ✓ la valeur de l'expression ValInit : valeur initiale du compteur d'itérations i.
  - ✓ la valeur de l'expression ValFin : valeur finale du compteur d'itérations i.
  - ✓ la valeur et le signe de la constante n (pas d'incrémentation ou de décrémentation du compteur).

    n>0 : le compteur d'itération i est :
    - ❖ initialisé par la valeur de l'expression ValInit.
    - ❖ incrémenté de n à chaque itération.

    La répétition s'arrête des que le compteur atteint une valeur supérieure à celle de l'expression ValFin.

    n<0 : le compteur d'itération i est :
    - ❖ initialisé par la valeur de l'expression ValInit.
    - ❖ décrémenté de n à chaque itération.

    La répétition s'arrête des que le compteur atteint une valeur inférieure à celle de l'expression ValFin.

Remarques :

- ❖ La variable i est un compteur donc être de type entier.
- ❖ Le pas n est optionnel s'il est égale à 1.
- ❖ Avec le schéma pour, l'initialisation du compteur i, l'incrémentation (décrémentation) du compteur i et le test d'arrêt de l'itération sont à la charge du processeur.

Exemple : Ecrire un algorithme permet de calculer le produit de deux entiers A et B en effectuant que des additions.

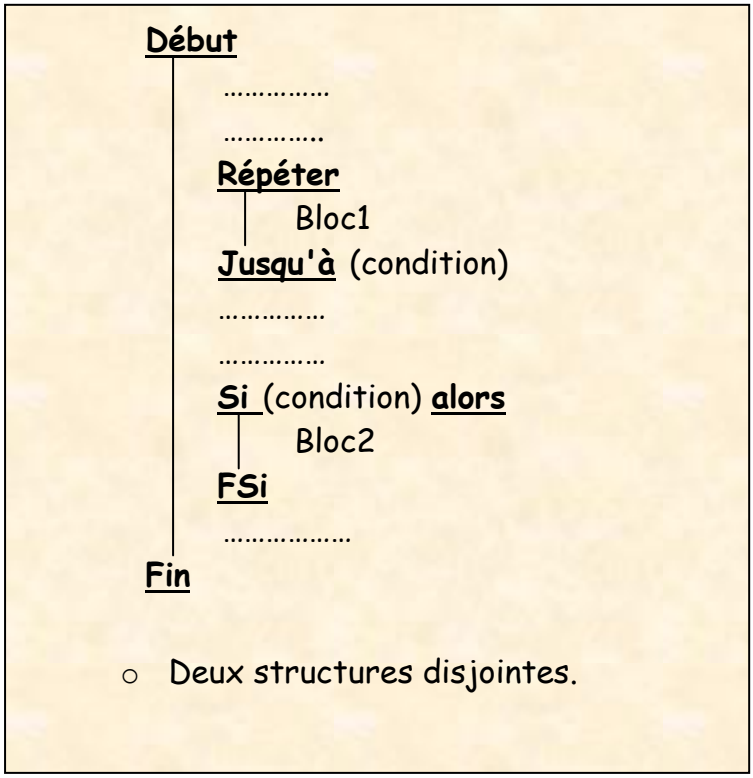
```
Exemple :

Algorithme Produit ;

Var  A, B, Prod, i : entier ;

Début
        Lire (A) ; Lire (B);
        Prod = 0 ;
        Pour i = 1  à  B faire
                Prod = Prod + A ;
        FPour
        Ecrire ( Prod ) ;
Fin.
```

Conclusion :

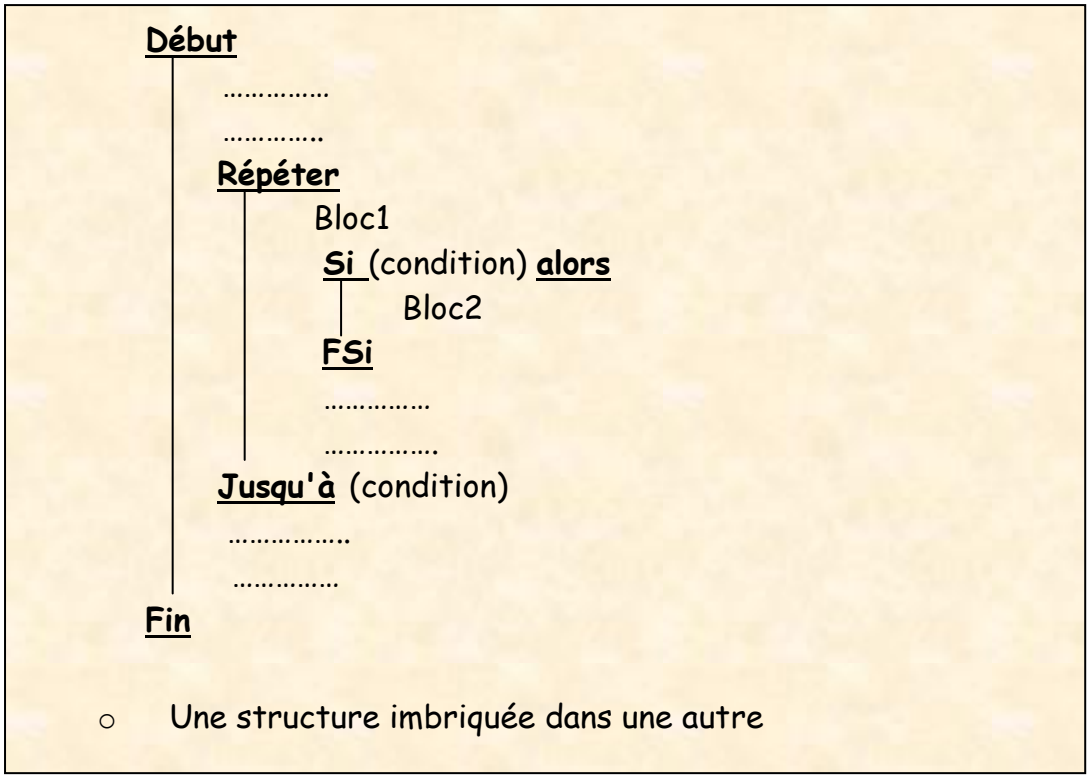
- Le schéma pour permet d'exprimer uniquement les itérations contrôlées par un compteur.
- Le schéma répéter permet d'exprimer les itérations dans lesquelles la première exécution du corps de la boucle n'est pas soumise à une condition.
- Le schéma Tant que permet d'exprimer toute itération.

3.4 Règles de base de construction d'un algorithme :

- Définir clairement et sans ambiguïtés le problème posé.
- un algorithme doit être bien structuré.
- le résultat doit être atteint en un nombre fini d'étapes. Il ne faut donc pas de boucles infinies.
- il faut étudier tous les cas possibles de données, ...
- le résultat doit répondre au problème demandé. Attention, un jeu d'essais ne prouve JAMAIS qu'un programme soit correct. Il peut seulement prouver qu'il est faux [4].
- Un algorithme doit être commenté. Le commentaire {commentaire} est une phrase que l'on peut insérer à n'importe quel niveau de l'algorithme (programme). Il sert uniquement à introduire des explications destinées au lecteur de l'algorithme. il est totalement ignoré par l'exécutant [4].
- Les instructions structurées peuvent être imbriquées ou disjointes mais ne peuvent en aucun cas se chevaucher.



- Deux structures disjointes.



- Une structure imbriquée dans une autre

- L'indentation est très utile pour mettre en évidence la structure des algorithmes et en faciliter la lecture.

Chapitre 04 :

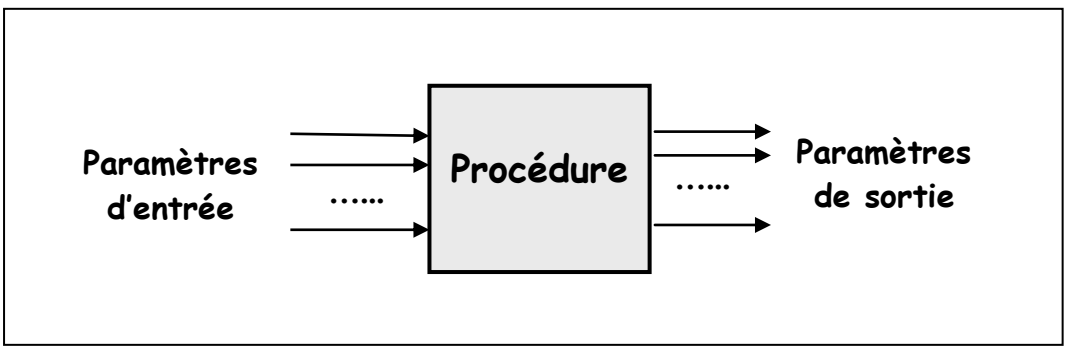
Programmation structurée

La résolution d'un grand problème par un seul code texte engendre souvent une solution comportant trop de détails, difficile à développer et à lire. Pour faciliter la tache, la programmation structurée offre des outils permettant de le décomposer en sous problèmes plus facilement maîtrisables. Chaque sous problème est traité séparément indépendamment des autres. Ainsi la solution sera décrite par plusieurs algorithmes : un algorithme principal qui défini la méthode générale de la résolution du problème et des sous algorithmes appelés fonctions ou procédures [4] .

4.1 Les procédures:

Une procédure décrit une suite d'actions qui s'exercent sur un ou plusieurs données appelés arguments pour calculer plusieurs résultats. Du point de vue algorithmique, on distingue :

- **Les paramètres d'entrée** : ils sont transmis en tant que données à la procédure.
- **Les paramètres de sortie** : par leurs biais, la procédure transmet les résultats des traitements. Dans le contexte de ce cours, ils seront précédés par le mot clé var.
- **Les paramètres d'entrée /sortie** : ils constituent à la fois les données en entrée et les résultats en sortie de la procédure.



Déclaration : Une procédure est définie par un entête et un corps.

- **L'entête :** il est décrit par la construction syntaxique suivante :

Procédure identificateur_ procédure **(**liste des paramètres formels**) ;**

- Identificateur_procédure : nom de la procédure, généralement un verbe d'ordre.
- Liste des paramètres formels : c'est une liste ordonnée de déclaration des paramètres utilisés pour définir la procédure. Les paramètres formels sont des variables. Chaque paramètre est décrit par :
  - Son nom,
  - Son type,
  - Son mode de transmission : paramètre d'entrée, paramètre de sortie, ou bien paramètre d'entrée / sortie.

- **Corps :** Il sert à décrire le traitement qu'elle réalise. Il utilisera les arguments de la procédure.

Exemple :

```
Procédure Permuter (var x réel, var y : réel) ;
        Var z: réel ;
        Début
                z ←x ;
                x← y ;
                y ← z ;
         Fin
```

On pourra également y trouver des variables locales déclarées à l'intérieur du corps de la procédure et qui par conséquent ne seront connues que de cette procédure.

Appel d'une procédure : Dans un algorithme, une procédure joue le rôle d'une instruction. Pour être exécutée, elle doit être appelée. L'algorithme qui comporte l'instruction d'appel est appelé appelant.

Syntaxe :

Identificateur_ Procédure **(**liste des paramètres effectifs**) ;**

Cette instruction d'appel de la procédure provoque l'exécution de la procédure appelée avec les paramètres effectifs définis dans l'appel. La liste des paramètres effectifs doit respecter les contraintes suivantes :

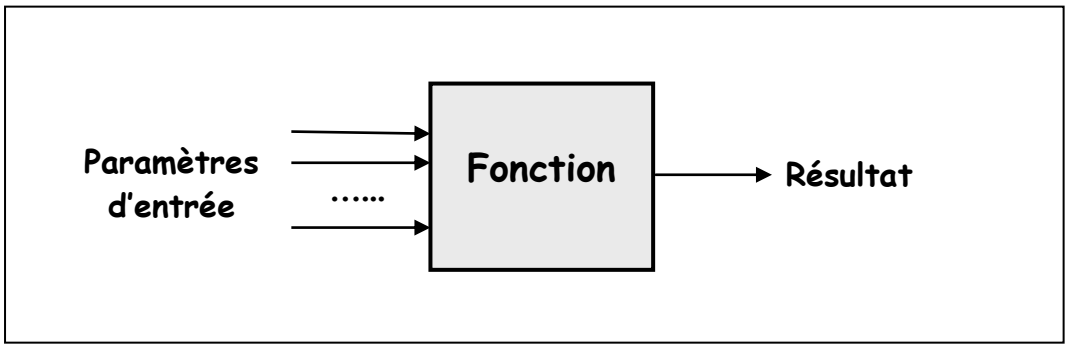
- Le nombre des paramètres effectifs doit être égale au nombre des paramètres formels définis dans l'entête de la procédure.
- La liste des paramètres effectifs doit être ordonnée dans le même ordre que la liste des paramètres formels.
- Chaque paramètre effectif doit être de type compatible avec le type du paramètre formel auquel il est associé.

Exemple :

```
Algorithme Test_Procédure ;

        Procédure Permuter (var x réel, var y : réel) ;
        Var z: réel ;
        Début
            z ←x ;
            x← y ;
            y ← z ;
        Fin

Var A, B , C : entier ;
Début
    Lire (A, B, C ) ;
    Permuter (A, B) ;
    Permuter (B , C ) ;
    Ecrire (A, B, C) ;
Fin.
```

4.2 Les fonctions:

Une fonction peut être considérée comme un opérateur non primitif crée à l'initiative du programmeur. Elle décrit une suite d'actions qui s'exercent sur un ou plusieurs paramètres et réalise le calcul d'un seul résultat.



Déclaration : Une fonction est définie par un entête et un corps.

- **L'entête :** il est décrit par la construction syntaxique suivante :

Fonction identificateur_ fonction **(**liste des paramètres formels**) :** type_ résultat ;

- Identificateur_ fonction : nom de la fonction, généralement un mot désignant une opération.
- Liste des paramètres formels : c'est une liste ordonnée de déclaration des paramètres utilisés pour définir la fonction. Les paramètres formels sont des variables. Chaque paramètre est décrit par :
  - Son nom,
  - Son type,
- Type_ résultat : Type de la valeur fournie par la fonction.

- **Corps :** L'ensemble des instructions utilisées pour le calcul du resultat de la fonction. Comme pour les procédures, on pourra également utiliser des variables locales déclarées à l'intérieur du corps de la fonction.
  - Toute fonction fournit un résultat unique. Ce résultat doit être affecté à l'identificateur de la fonction.

Exemple :

```
fonction Puissance (A : entier , B : entier) :  entier.
        Var Prod , i : entier ;
        Début
            Prod = 1  ;
            Pour i = 1  à  B faire
                    Prod ← Prod * A ;
            FPour

              Puissance ← Prod ;
        Fin
```

Appel d'une fonction : Une fonction est utilisée (appelée) dans une expression. Son appel provoque son exécution avec les paramètres effectifs définis dans l'appel et renvoie un résultat. La liste des paramètres effectifs d'une fonction doit respecter les mêmes contraintes que la liste des paramètres effectifs d'une procédure.

Exemple : Calcul de la somme : $1^5 + 2^5 + 3^5 + \ldots\ldots\ldots + 20^5$.

```
Algorithme Test_Fonction ;

        fonction Puissance (A : entier , B : entier) :  entier.
        Var Prod , j : entier ;
        Début
            Prod = 1 ;
            Pour j = 1 à B faire
                    Prod ← Prod * A ;
            FPour
            Puissance ← Prod ;
        Fin
Var  i , Som : entier ;

Début
      Som = 0 ;
      Pour i = 1 à 20 faire
              Som ← Som + Puissance( i , 5 ) ;
      FPour

      Ecrire (Som) ;
Fin
```

Remarques :

- Une procédure est utilisée pour abstraire et nommer une suite d'actions, elle peut être considérée comme une instruction.
- Une fonction est utilisée pour abstraire et nommer le calcul d'une expression, elle peut être considérée comme un opérateur.
- Il est utile de définir une procédure ou une fonction lorsqu'une suite d'actions intervient plusieurs fois dans un algorithme afin d'éviter la répétition du code.
- tous les arguments d'une fonction sont des arguments d'entrée.
- Le type du résultat d'une fonction est un type scalaire.
- Les paramètres formels et les variables locales n'appartiennent qu'à une procédure (ou une fonction). Leur valeur ne peut en aucun cas être récupérée par le programme principal.
- Rien n'empêche qu'un identificateur attribué à une variable locale d'une procédure (ou une fonction) ou à paramètre formel soit utilisé pour toute autre variable dans le programme principal ou dans une autre fonction. Mais il est préférable de bien différencier les variables afin de rendre le programme plus lisible.

4.3 La récursivité:

Une procédure ou une fonction récursive possède la propriété de s'appeler elle-même dans sa définition. La récursivité possède deux propriétés :

- Il doit exister des critères pour lesquels les appels cessent.
- Chaque fois que la procédure ou la fonction s'appelle, elle doit être plus proche de ses critères d'arrêt.

Syntaxe :

L'entête du sous_programme

Si (cas simple) **alors**

solution immédiate

Sinon

solution récursive, impliquant un cas plus simple que le problème original

Fsi

Ce type de programmation permet de réaliser par exemple des fonctions définies à partir de relations de récurrence.

Exemple : Calcul de la factorielle d'un entier N donné .

```
fonction Fact (Nbr : entier) :  entier.
Var Prod , j : entier ;
Début
        Si   (Nbr = 1 )  alors
                Fact ← 1 ;
        Sinon
                Fact ←  Nbr *  Fact(Nbr – 1 ) ;
        Fsi
Fin
```

Chapitre 05 :

Programmation orientée objets

En programmation classique le développement d'une application se décompose en deux étapes bien distinctes :

- La définition des structures de données capables de sauvegarder l'information que l'on veut gérer.
- La conception de fonctions et de procédures qui travaillent sur les structures de données conçues à l'étape précédente et qui permettent de mettre en œuvre tous nos besoins.

Cette forme de programmation, qui sépare les données des traitements, rend difficile la mise en place de contrôles, destinés à garantir la cohérence et l'intégrité des données. L'utilisateur peut en effet accéder directement aux données, sans utiliser les fonctions mises à sa disposition [3].
De plus, toute modification de l'implémentation des données à des conséquences sur l'ensemble des fonctions travaillant sur ces données, et donc sur l'ensemble des traitements utilisant ces fonctions.

La programmation orientée objets a été introduite afin de remédier à ces inconvénients.

5.1 Classe et objets :

Une classe est une description abstraite d'une entité particuliere. Elle est constituée de structure de données et de tous les traitements associées. A partir d'une classe, tout langage objet permet (par l'intermédiaire de l'opérateur new) de créer des objets qui seront des emplacements en mémoire centrale destinés à contenir effectivement l'information manipulée. Ces objets sont appelés instance et l'opération de création : instanciation. Generalement, les auteurs définissent une classe comme un "moule" à partir duquel on peut obtenir autant d'objets que nécessaires [5].

Exemple : La classe Point.

Class Point

{

}

Class Application

{

P1 : Point ;

……

P1= New Point(……) ;

…….

Manipulation des objets de type Point

…….

…….

}

Le principe de base de la programmation orientée objets consiste :

- d'une part à regrouper (ou à "encapsuler") au sein d'une même unité appelée classe les données et les traitements associés.
- d'autre part à contrôler l'accès aux données en les déclarant privées ce qui obligera l'utilisateur de la classe à employer les traitements encapsulés pour agir sur celles-ci.
- permettre la modification des classes sans avoir d'incidence sur les programmes des utilisateurs.

5.2 Les attributs d'une classe :

Toute classe comporte une partie données, dans laquelle on décrit les attributs qui seront utiles à la mémorisation de l'information que l'on veut gérer. Ces emplacements memoire sont en fait alloués à chaque instanciation et sont caractérisés par un nom et un type qui peut être simple ou structuré.

- **Spécificateur d'accès :** représente le niveau de protection de la donnée, Generalement on utilise l'un des dfeux niveaux suivants :
  - **public** : l'accès direct à la donnée est possible depuis tout programme d'utilisation de la classe par la référence :

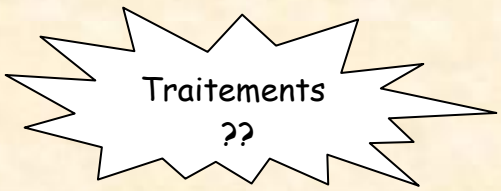
    Nom_Objet.Nom_Attribut .

  - **private** : l'accès direct à la donnée n'est pas possible depuis un programme d'utilisation de la classe. Le developpeur devra utiliser des traitements encapsulés pour lire ou mettre à jour ces données.

Exemple : Pour manipuler des points dans un plan, on utilise une classe Point avec deux attributs de type entier : x et y . A chaque instanciation, il y aura création de ces deux variables.

Exemple : La classe Point.

```
Class Point
    {
        public x : entier ;
        public y : entier ;
```




```
    }
```

5.3 Les méthodes d'une classe :

Les traitements d'une classe sont appellés méthodes. leur déclaration respecte la syntaxe suivante :

Syntaxe :

```
Spécificateur_d'accès Type_Résultat Nom_Méthode (Liste Paramètres formels)
        {
                Déclaration de variables locales
                    …
                Instructions

         }
```

Le type du resultat, le nom de la methode et la liste de ses paramètres formels constituent la **signature** de la méthode.

Remarques:

- Dans le cas où la fonction renvoie un résultat, ce dernier devra être retourné par l'instruction return (Résultat).
- Dans le cas où la fonction ne renvoie aucun résultat, on doit spécifier le mot clé **void** pour le TypeRésultat .

Exemple : La classe Point.

```
Class Point
    {
          public x : entier ;
          public y : entier ;
           ……
           ……
           Void déplacer ( dx :entier, dy :entier)
                {
                   x = x + dx ;
                   Y = y + dy ;
                }
          entier distCarrée_Origine (  )
                {
                   dist : entier ;
                   dist = ( x * x ) + ( y * y )  ;
                   Return ( dist ) ;
                }
          ……
          ……
    }
```

De meme que les attributs, les methodes d'une classe se répartissent generalement en deux catégories :

- Les méthodes **publiques** qui correspondent à tous les traitements que l'utilisateur d'une classe pourra appliquer à ses objet. Certains auteurs qualifient ces méthodes publiques de messages que l'on peut envoyer aux objets.
- Les méthodes **privées** qui sont des traitements introduits pour simplifier la conception de la classe, mais qui ne pourront pas être utilisés depuis un programme externe.

5.4 Les constructeurs :

Un constructeur est une méthode qui porte le nom de la classe. Elle est appellée à l'instanciation des objets. Son code est chargé d'initialiser les attributs de l'objet au moment de sa création. Généralement, le compilateur définit à chaque classe un constructeur par défaut qui affectera des valeurs dépendantes du type de l'attribut si aucune valeur d'initialisation n'est précisée dans la déclaration [6]. Le tableau ci-dessous récapitule, pour chaque type, les valeurs affectées par défaut.

Type	Valeur
Entier	0
Réel	0.0
Chaîne de caractères	Null

Si on se limitait à l'existence du constructeur par défaut, toute instanciation devrait être suivie d'une phase d'affectation pour valoriser les attributs de l'objet. Pour remédier à ce problème, la programmation orientée objet propose au concepteur de la classe de construire plusieurs constructeurs avec des signatures différentes. La définition d'un nouveau constructeur devra respecter la syntaxe suivante :

```
public Nom_Classe (Liste des paramètres formels)
        {
                Code d'initialisation des attributs
        }
```

Remarques :

- On remarque qu'un constructeur n'est pas typé.
- Une fois le constructeur défini, celui-ci est appelé par l'opérateur **new** afin d'instancier des objets.
- Lorsqu'une classe comporte plusieurs constructeurs on dit que le constructeur est surchargé.

Exemple : La classe Point.

```
Class Point
    {
        public x : entier ;
        public y : entier ;
        ……
        ……
        Point ( x_p : entier , y_p : entier)
            {
                x = x_p ;
                y = y_p ;
            }
        Point ( pt : Point )
            {
                x = pt.x ;
                y = pt.y ;
            }
        ……
        ……
    }
```

```
Class Application
  {
        p1 : Point ;
        p2 : Point ;
            ……
            …….
        p1= New Point(5,20) ;
        p1.déplacer(40,70);
            …….
            …….
        p2 = new Point(p1) ;
            …….
            …….
  }
```

5.5 Les attributs et méthodes de classe:

Un attribut de classe est une variable commune à tous les objets de la classe . Elle existe avant même qu'un objet soit instancié. Elle est déclarée au moyen du modificateur **static** , et peut être privée ou publique. Avec le statut Private , elle pourra être référencée uniquement par les méthodes de la classe dans laquelle elle est définie, alors qu'avec le statut Public, elle pourra être référencée depuis une autre classe. Une variable de classe est referencée par:

NomClasse.Nomvariable

Pour permettre à utilisateur de modifier une variable de classe , generalement on doit définir une méthode de classe. Celle-ci pourra être appelée, sans qu'aucun objet existe, en utilisant la référence:

NomClasse.NomMethode(parametres).

Exemple : La classe Point.

```
Class Point
   {
      static nbr_Points = 0 ;

      public x : entier ;
      public y : entier ;
          ……….
          ……….
      Point ( x_p : entier , y_p : entier)
          {
              x = x_p ; y = y_p ;
              nbr_Points = nbr_Points + 1 ;
          }
      Point ( pt : Point )
          {
               x = pt.x ;  Y = pt.y ;
              nbr_Points = nbr_Points + 1 ;
          }
          ……….
          ……….
   }
```

```
Class Application
  {
        p1 : Point ;
        p2 : Point ;
            ……….
       Ecrire ( Point.nbr_Points) ;
            ……..
       p1= New Point(5,20) ;
        p1.deplacer(40,70);
            ……..
            ……..
       p2 = new Point(p1) ;
            ……..
            ……..
       Ecrire ( Point.nbr_Points) ;
  }
```

5.6 L'héritage :

Compte tenu des connaissances acquises jusqu'à présent, pour implémenter une relation "est un" entre deux entités A et B , la seule solution envisageable consiste à intégrer l'entité A dans l'entité B. Celle technique appelée **composition**.

Exemple : La classe Point_Coloré.

```
Class Point_Coloré
    {
        public  pt : Point ;
        public couleur : chaine ;
            ……….
       Point_Coloré ( x_p : entier , y_p : entier, couleur_p :chaine)
          {
              pt = new Point( x_p ,  y_p ) ;
               couleur = couleur_p ;
          }
           ……….
    }
```

Cette solution qui consiste à dire qu' un Point_Coloré contient un Point. Avec une telle implémentation, pour deplacer le PointColoré ptc, on devra écrire l'instruction:

Ptc.pt.Deplacer(dx,dy);

Cette maniere de programmer est tres lourde compte tenu du fait qu 'un PointColoré est un Point. La maniere la plus naturelle est la suivante :

Ptc. Deplacer (dx,dy) ;

C'est le concept d'**héritage** qui va permettre de représenter le lien **est un** de manière plus pertinente.

Définition : Pour exprimer le fait que tout objet de la classe B est un objet de la classe A on devra écrire la classe B de la manière suivante:

```
class B extends A
      {
                  Attributs et
                  méthodes
                  spécifiques
      }
```

La définition de cette classe peut être interprétée ainsi:

- la clause **extends** spécifie qu'un objet de la classe B hérite de toutes les fonctionnalités de la classe A. Cela signifie qu'un objet de la classe B pourra utiliser directement les attributs et les methodes definis dans la classe A.
- le code de la classe B décrit uniquement les attributs et les methodes spécifiques à la classe B. L'ajout de ces nouvelles fonctionnalités pour la classe B est appelée **spécialisation**.
-

D'un point de vue terminologie, on dit que :

- B est une **sous-classe** de A ou encore une classe **dérivée** de A.
- A est une **super-classe** ou encore une **classe de base**.

L'héritage permet donc de réutiliser très facilement le code déjà écrit. En plus, il garantit que toute modification au niveau de la classe A sera automatiquement répercutée au niveau de la classe dérivée.

5.7 Caractéristiques des classes dérivées :

Constructeur d'une classe dérivée : Pour créer un objet de la classe dérivée, on doit obligatoirement créer un objet de la classe de base par un appel au constructeur de celle-ci en utilisant le mot clé super. Cette obligation garantit la relation "est un" entre l'objet de la classe dérivée et l'objet de la classe de base [3]. Ainsi, le constructeur de la classe PointColoré doit avoir la forme suivante:

Exemple : La classe Point_Coloré.

```
Class Point_Coloré extends Point
    {
        public couleur : chaine ;
          ……….
          ……….
        Point_Coloré ( x_p : entier , y_p : entier, couleur_p :chaine)
           {
               Super ( x_p , y_p ) ;
               couleur = couleur_p ;
           }
          ……….
          ……….
    }
```

Remarques:

- L'appel au constructeur de la classe mère doit être la première instruction du constructeur de la méthode dérivée.
- Si l'appel au constructeur de la classe mère n'est pas explicitement cité dans le constructeur de la classe dérivée, le compilateur recherche un constructeur par défaut (constructeur sans paramètre) dans la classe mère. Si celui-ci n'existe pas, le compilateur signale un message d'erreur .

Les méthodes d'une classe dérivée :

Avant de parler des méthodes qui seront explicitement définies dans la classe dérivée, il faut se rappeler, que toute méthode de la classe de base peut être appliquée à un objet de la classe dérivée. Outre ces traitements communs, la définition d'une classe dérivée intègre des méthodes dues à sa spécialisation. Celles-ci peuvent être réparties en 3 catégories distinctes:

- La première catégorie regroupe toutes les méthodes qui n'existent pas dans la classe mère. Ces méthodes ne peuvent s'appliquer que sur des objets de la classe dérivée.
- La seconde catégorie englobe les méthodes dites **surchargées**. Ces traitements ont le même nom que des méthodes de la classe mère mais avec des signatures différentes.
- La troisième et dernière catégorie regroupe les méthodes **redéfinies**. Ces méthodes qui ont la même signature que dans la classe mère sont dites **polymorphes**. C'est au niveau de l'exécution et en fonction de l'objet sur lequel la méthode est appliquée que la forme de la méthode est choisie.

Le modificateur "**final**" dans une méthode de la classe de base empêche la redéfinition de cette méthode dans les classes filles.

Manipulations d'objets issus de classe mère et fille

L'affectation d'un objet de la classe dérivée dans une variable objet de la classe mère est possible sans avoir à spécifier un transtypage. Ainsi tout Point_Coloré étant un Point on peut affecter dans une variable de la classe Point un objet de la classe Point_Coloré.

Exemple : La classe Point_Coloré.

```
Class Application
   {
          ……….
          Point Pt;
          Point_Coloré  Ptc;
          Ptc = new Point_Coloré( 50 ,  25 ,  "rouge ");
          Pt=Ptc;
          ……….
   }
```

Par contre l'affectation réciproque (objet de la classe mère dans une variable objet de la classe fille) nécessite un transtypage explicite. Ainsi un point n'étant pas forcément un point coloré, on devra spécifier un cast afin d'indiquer au compilateur que l'objet pointé par une variable de la classe Point est effectivement un Point_Coloré.

Remarque :

- Cette technique est aussi nécessaire dès lors que l'on veut utiliser des attributs ou des méthodes de la classe dérivée d'un objet pointé par une variable de la classe mère.

Exemple : La classe Point_Coloré.

```
Class Application
    {
                ……
                ……
            Point Pt;
            Point_Coloré  Ptc_01;
            Point_Coloré  Ptc_02;

            Pt = new Point ( 50 ,  25 ');
            Ptc_01 = new Point_Coloré( 50 ,  25 ,  "rouge ") ;
                 ……
            Pt = Ptc_01 ;
             Ptc_02 = ( Point_Coloré ) Pt ;
                ……
                ……
             Ecrire ( ( Point_Coloré )  Pt.couleur  ) ;
                ……
                ……

    }
```

Travaux dirigés

Partie_01 : Algorithmes simples

Exercice n°1-1:

Ecrire un algorithme qui permet de classer trois nombres entiers quelconques X, Y, et Z saisis par l'utilisateur dans l'ordre croissant et de les afficher.

Exercice n°1-2:

Soient deux nombres X et Y classés par ordre croissant et N un nombre entier quelconque. Ecrire un algorithme qui permet d'afficher ces trois nombres dans l'ordre croissant.

Exercice n°1-3:

Un photocopieur assure ses services au prix de 2 DA la photocopie (la page). Si le nombre de pages à photocopier dépasse 30, il applique une réduction de 15% au prix total et si le nombre de pages à photocopier dépasse 100, il applique une réduction de 25 % au prix total.

1) Ecrire un algorithme qui permet de calculer le montant à payer pour photocopier NBR pages.
2) Modifier cet algorithme pour qu'il fait ce calcul et affiche le montant à payer pour N clients.

Exercice n°1-4:

Ecrire un algorithme permettant de calculer la factorielle d'un entier positif N.

Exercice n°1-5:

Ecrire un algorithme permet de lire une suite de N entiers naturels et d'afficher la valeur maximale, la somme des nombres pairs et la somme des nombres impairs.

Exercice n°1-6:

Soit la suite de Fibonacchi définie par :

- $F_0 = 0$, $F_1 = 1$.
- $F_{n+2} = F_n + F_{n+1}$, $n \geq 0$.

Ecrire un algorithme qui permet de calculer le $n^{ème}$ terme de cette suite.

Exercice n°1-7:

Soit la suite S définie par :

$$\begin{cases} S_0 = 2, S_1 = 3 \\ S_n = S_{n-2} + ((-1)^n . S_{n-1}) \end{cases}$$

Ecrire l'algorithme qui détermine et affiche le $n^{ième}$ terme Sn.

Exercice n°1-8:

Ecrire un algorithme qui permet de calculer l'exponentielle d'un réel x notée Exp en utilisant la formule :

$$Exp(x) = 1 + X/1! + X^2/2! + X^3/3! + \ldots\ldots\ldots\ldots + X^n/n!$$

Avec : $|X^{n+1}/(n+1)!| < Epsilon$.

Exercice n°1-9:

Ecrire l'algorithme qui permet de :

- Lire deux nombres entiers Nbr1 et Nbr2.
- Déterminer et afficher le quotient et le reste de la division entière de Nbr1 par Nbr2 en utilisant des soustractions successives.

Exercice n°1-10:

L'une des méthodes de calcul du Plus Petit Commun Multiple de deux entiers positifs a et b, tel que a>b est de trouver le plus petit multiple de a qui est aussi multiple de b.

- Ecrivez un algorithme qui permet de saisir deux entiers positifs non nuls x et y. Puis, détermine et affiche leur Plus Petit Commun Multiple.

Exercice n°1-11:

Un nombre parfait est nombre égal à la somme de ses diviseurs propres y compris 1 mais lui-même exclu.

1) Ecrire un algorithme qui lit un entier Nbr et affiche s'il est parfait ou pas.

2) Modifier cet algorithme de telle sorte qu'il affiche les N premiers nombres parfaits.

Exercice n°1-12:

Deux nombres naturels Nbr1 et Nbr2 sont dits nombres amis si la somme des diviseurs de Nbr1 est égale à Nbr2 et la somme des diviseurs de Nbr2 est égale à Nbr1.

Ecrire un algorithme qui lit deux nombres naturels Nbr1 et Nbr2 et affiche s'ils sont amis ou pas.

Exercice n°1-13:

1) Ecrire un algorithme qui permet de lire un nombre entier Nbr et affiche s'il est premier ou pas.
2) Modifier cet algorithme de telle sorte qu'il affiche les N premiers nombres premiers.

3,5,7,11,13,……,

N Nombres premiers

Exercice n°1-14:

Le calcul du plus grand commun diviseur (pgcd) de deux entiers a et b positifs peut se faire en suivant l'algorithme d'Euclide. Cet algorithme prend pour base le théorème d'Euclide.

> Si a > b et si on effectue la division de a par b :
>
> a = q b + r avec 0 <= r < b
>
> alors : le pgcd de a et b **est égal** au pgcd de b et r.

Pour calculer pgcd (a, b) il suffit d'itérer cette technique, jusqu'à l'obtention d'un reste nul. Le dernier reste non nul est alors le pgcd de a et b.

- En se basant sur ce théorème, écrire un algorithme permettant de calculer et d'afficher le plus grand commun diviseur de deux entiers Nbr1 et Nbr2.

Exercice n°1-15:

1. Ecrire un algorithme qui permet à l'utilisateur de saisir deux nombres entiers N et p, puis affiche le chiffre qui se trouve dans N à la $p^{ième}$ position.

2. Ecrire un algorithme qui prend de l'utilisateur un entier Nbr puis affiche la somme de ses chiffres.
   - Nbr est un entier quelconque.

Partie_02 : Tableaux et matrices

Exercice n°2-1:

Soit Vect un vecteur de N valeurs entières. Donner un algorithme permettant d'extraire deux tableaux PAIR et IMPAIR et de les afficher. Le premier contiendra les éléments pairs de Vect alors que le second les éléments impairs.

Exercice n°2-2:

Nous disposons d'un tableau de N entiers naturels.

- Ecrire un algorithme qui permet de ranger les nombres pairs au début et les nombres impairs à la fin de ce tableau.

Exercice n°2-3:

Un palindrome est un tableau de caractères qui se lit de la même façon dans les deux sens (ex : « elle », « radar », « laval »).

- Écrire un algorithme qui dit si un tableau de caractères noté Tab est un palindrome ou non.

Exercice n°2-4:

Ecrire un algorithme qui construit un tableau FUS trié par ordre croissant avec les éléments de deux tableaux A et B déjà triés par ordre croissant.

- Ajouter les instructions qui permettent de saisir un nouvel élément NBR, et l'insérer dans FUS tout en respectant l'ordre croissant.

Exercice n°2-5:

Soit TabEntier un tableau de N valeurs entières.

- Ecrire un algorithme qui permet de vérifier et d'afficher si c'est un tableau trié ou pas.

Exercice n°2-6:

On considère deux tableaux Tab1 et Tab2 contenant des éléments de type entier chacun. On dit que Tab1 est contenu dans Tab2 si chaque élément de Tab1 figure au moins une fois dans Tab2. Ecrire un algorithme qui permet de :

- Saisir les éléments des deux tableaux Tab1 et Tab2.
- Vérifier et afficher si Tab1 est contenu dans Tab2 ou non.

Exercice n°2-7:

Soit TabEntier un tableau de N valeurs entières, et Val une variable de type entier.

- Ecrire un algorithme qui permet de déterminer et d'afficher la longueur de la plus grande succession de val dans le tableau TabEntier .

Exercice n°2-8:

Soit TabEntier un tableau de n valeurs entières ou chaque nombre peut apparaître plusieurs fois.

- Ecrire un algorithme qui permet de compresseur le vecteur par l'élimination des valeurs répétées en utilisant le même vecteur.

Exercice n°2-9:

Soit TabEntier un tableau de N valeurs entières.

Ecrire un algorithme qui permet de :

- Saisir les éléments du tableau TabEntier.
- Saisir un entier NBR, ensuite calculer et afficher le nombre d'occurrences de NBR dans le tableau TabEntier.
- Calculer et afficher le nombre des éléments pairs.
- Permuter le plus grand élément et le plus petit élément du tableau TabEntier.
- Chercher et afficher l'élément médian. (Pour simplifier on suppose que les éléments du tableau sont tous différents et que N est un nombre impair).
- Calculer et afficher le nombre des éléments premiers.

Exercice n°2-10:

1) Ecrire un algorithme permettant de créer un tableau nommé TabEntier contenant N entiers et initialisé par les valeurs 1, 2,3,……,N.

- la première case contient la valeur 1, la deuxième la valeur 2,……, et la dernière N.

2) Ajouter les instructions permettant de décaler l'ensemble des valeurs d'une case vers la droite (la dernière valeur devra se retrouver dans la première case !).

3) Ajouter une boucle permettant de faire une rotation circulaire des éléments du tableau.

Exercice n°2-11:

Soit Mat une matrice carrée de taille n*n. Ecrire un algorithme qui permet de :

- Tester si la matrice Mat est symétrique.
- Permuter les deux diagonales de cette matrice
- Calculer et afficher la trace de cette matrice (la somme des éléments de la diagonale principale).
- Calculer et afficher les éléments de la transposée de la matrice Mat.
- Permuter La i ^{ième} ligne et la k ^{ième} ligne de la matrice Mat.

Exercice n°2-12:

Etant donnée une matrice Mat de taille n*n. Ecrire un algorithme qui permet de vérifier s'il s'agit d'un carré magique ou non. Un carré magique d'ordre n est une matrice carrée n x n telle que la somme des éléments de chaque ligne, chaque colonne et des deux diagonales sont identiques.

Exercice n°2-13:

Soit Mat une matrice carrée de taille n*n. Ecrire un algorithme qui permet d'extraire deux vecteurs Vmin et Vmax tel que :

- Vmax : contient les maximums des lignes.
- Vmin : contient les minimums des colonnes.

Exercice n°2-14:

Étant données deux matrice Mat_1 de taille n*m et Mat_2 de taille m*l. Ecrire un algorithme qui calcule le produit de ces deux matrices et affiche les éléments de la matrice résultante.

Exercice n°2-15:

Ecrire un algorithme qui cherche et affiche les points selles d'une matrice.

- Points selles : ce sont les éléments qui sont maximum sur leurs colonnes et minimum sur leurs lignes.

Exercice n°2-17:

Soit Mat une matrice carrée de tailles n * n contenant des éléments de type entier.

Ecrire un algorithme permettant, à partir du paramètre n saisi par l'utilisateur, de remplir la matrice Mat selon le principe suivant :

- Chaque élément situé sur et au dessus de la diagonale soit égal au produit du numéro de ligne et du numéro de colonne.
- Tous les autres éléments étant égaux à zéro (matrice triangulaire supérieure).

Exercice n°2-18:

On souhaite écrire un algorithme qui construit et affiche un triangle de Pascal de degré N. Voici quelques exemples pour différentes valeurs :

N = 7 :

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
```

N = 8 :

```
1
1   1
1   2   1
1   3   3   1
1   4   6   4   1
1   5   10  10  5   1
1   6   15  20  15  6   1
1   7   21  35  35  21  7   1
```

Partie_03 : Programmation structurée

Exercice n°3-1:

- Écrire fonction Puissance qui permet de calculer A^B .
- Ecrire une fonction Fact qui permet de calculer N !.
- Ecrire un algorithme qui permet de calculer Exp (X) en utilisant la formule :

$$Exp(X) = 1 + X/1! + X^2/2! + X^3/3! + \ldots\ldots\ldots\ldots\ldots\ldots + X^n/n!$$

Avec : $|X^{n+1} / (n+1)!| <$ Epsilon.

Exercice n°3-2:

Un nombre parfait est un nombre qui est égal à la somme de ses diviseurs propres y compris 1 mais lui-même exclu.

- Écrire fonction booléenne, appelée Parfait, prenant comme paramètre un nombre entier, et retournant un booléen de valeur :
  - Vrai : si le nombre est parfait.
  - faux : sinon.
- Ecrire un algorithme Parfaits_Max affichant tous les nombres parfaits inférieurs à N_Max.
- Ecrire un autre algorithme Parfaits_N affichant les N premiers nombres parfaits.

Exercice n°3-3:

Soit la suite de Fibonnacci, définie par :

$$\begin{cases} U_0 = 1 \ , \ U_1 = 1 \\ U_{n+1} = U_n + U_{n-1} \end{cases}$$

- Utiliser une fonction récursive FiboRec qui détermine le $n^{ième}$ terme de la suite.
- Écrire un algorithme calculant le nombre d'Or. La suite (U_{n+1} / U_n) converge vers ce nombre .

Exercice n°3-4:

Le calcul du plus grand commun diviseur (pgcd) de deux entiers a et b positifs peut se faire en suivant l'algorithme d'Euclide. Cet algorithme prend pour base le théorème d'Euclide.

- Si a > b et si on effectue la division entière de a par b alors :

$$a = q\,b + r \quad \text{avec} \quad 0 <= r < b$$

alors, le pgcd de a et b est égal au pgcd de b et r.

- Pour calculer pgcd (a, b) il suffit d'itérer cette technique, jusqu'à l'obtention d'un reste nul.
- Le dernier reste non nul est alors le pgcd de a et b.

En se basant sur le théorème d'Euclide :

- Ecrire une fonction itérative permettant de calculer le plus grand commun diviseur de deux entiers Nbr1 et Nbr2.
- Donner la version récursive équivalente.

Exercice n°3-5:

Soit TabEntier un tableau de N valeurs entières, et Val une variable de type entier.

- Ecrire un sous programme Lecture qui permet de saisir les éléments du tableau TabEntier.
- Ecrire un sous programme Nbr_Succ qui permet de calculer le nombre de succession de val dans le tableau TabEntier à partir d'une position pos.
- Ecrire un algorithme Plus_Gr_Succ qui permet de :
  - Saisir les éléments du tableau TabEntier.
  - Saisir la valeur de Val.
  - Déterminer et afficher la longueur de la plus grande succession de val dans le tableau TabEntier .

Exercice n°3-6:

- Ecrire un sous programme Permuter qui permet de permuter l'élément d'indice ind1 et l'élément d'indice ind2 d'un tableau Tab contenant des valeurs de type entier.
- Soit TabEntier un tableau de N valeurs entières. Ecrire un algorithme qui permet de :
  - Saisir les éléments du tableau TabEntier.
  - Faire une rotation circulaire des éléments du tableau TabEntier en utilisant le sous-programme permuter.
    - Le premier doit se retrouver en dernier, et le dernier en premier.
  - Trier le des éléments du tableau TabEntier en utilisant le sous-programme permuter.

Exercice n°3-7:

- Ecrire un algorithme qui cherche et affiche les points selles de la matrice Mat en utilisant les sous programmes suivants :
  - Est_Max_Col (M,Nbr,ind_c): fonction retournant un booléen de valeur :
    - Vrai : si l'entier Nbr est maximum sur la colonne ind_c .
    - faux : sinon.
  - Est_Min_Lig (M,Nbr,ind_l): fonction retournant un booléen de valeur :
    - Vrai : si l'entier Nbr est minimum sur la ligne ind_l.
    - Faux : sinon.

Exercice n°3-8:

- Ecrire un sous programme Lecture (M,Nbr_Lig,Nbr_Col) qui permet de saisir les éléments d'une matrice M de Nbr_Lig lignes et Nbr_Col colonnes contenant des entiers.
- Ecrire un autre sous programme Premier(Nbr) prenant comme paramètre un nombre entier Nbr, et retournant un booléen de valeur :
  - Vrai : si le nombre Nbr est premier.
  - faux : sinon.
- Etant donnée une matrice Mat de dimension n*n contenant des valeurs de type entier. Ecrire un algorithme qui permet de :
  - Saisir les éléments de la matrice Mat.
  - Afficher tous les nombres premiers contenus dans la matrice Mat.

Exercice n°3-9:

Au départ, on dispose de 3 piquets. Le premier porte N disques de tailles toutes différentes et empilés du plus grand (en bas) au plus petit (en haut). Le problème consiste à déplacer tous ces disques jusqu'au troisième piquet sachant qu'on doit respecte les règles suivantes :

- On ne déplace qu'un disque à la fois;
- On ne peut déplacer qu'un disque se trouvant en haut d'une pile ;
- On peut poser un disque sur un piquet vide, sur un disque de diamètre supérieur, mais aucun disque ne doit être empilé sur un disque de diamètre inférieur.

Ecrire un algorithme permettant de résoudre ce problème.

Partie_04 : Programmation orientée objets

Exercice n°4-1:

Vous allez construire une classe qui représente un point. Chaque instance de cette classe possède 2 attributs **x** et **y** qui représentent les coordonnées du point.

1) A la création du point il est possible de fixer les coordonnées ou pas.

- Si on fixe les coordonnées, il faut que se soit à l'aide d'un point déjà existant ou alors en donnant les valeurs séparées de x et de y. Les valeurs possibles pour x et pour y doivent être comprises entre 0 et 1000. Si on dépasse ces valeurs, le point résultant deviendra le point origine (0,0).
- Dans le cas où on ne fixe pas de coordonnées, cela correspond au point se situant à l'origine des axes (0 , 0).

❖ Donner les trois constructeurs de la classe Point.

2) Ecrire deux méthodes déplacer qui permettent de modifier les coordonnées du point, soit en donnant les nouvelles valeurs de x et de y, soit en donnant un autre point.

3) Ecrire une méthode similaire qui détermine si les coordonnées d'un point passé en argument sont égaux aux coordonnées du point qui exécute la méthode.

4) On considère la classe Cercle qui dispose de 2 attributs : d'une part, l'attribut centre (de type Point) qui fixe les coordonnées du centre du cercle, d'autre part, l'attribut rayon qui aura toujours une valeur fixe de 50.

Plusieurs types de création seront possibles, soit par rapport à un point, soit par rapport aux coordonnées séparées, soit par rapport à un autre cercle ou soit sans aucune précision en sachant que le cercle se trouvera alors à l'origine.

❖ Construire la classe Cercle en donnant les quatre constructeurs possibles.

5) Ecrire une méthode getCentre qui renvoie le centre du cercle.

6) Ecrire 2 méthodes déplacer qui permettent de modifier les coordonnées du centre du cercle.

7) De même écrire une méthode similaire qui détermine si un cercle passé en argument est le même que le cercle qui exécute la méthode.

8) Ecrire dans la classe Cercle la méthode surface permettant de calculer la surface d'un cercle.

Exercice n°4-2:

On considère la classe Compte destinée à gérer des comptes bancaires. Chaque objet de cette classe sera décrit par les informations suivantes: son Numéro, le Nom de son titulaire et son Solde. Les numéros de compte prennent des valeurs entières successives et sont attribués automatiquement à la création des comptes en commençant par 800001.

1) Utiliser la notion de surcharge de constructeur pour construire un objet de la classe Compte des deux façons suivantes :

- Les valeurs initiales Numéro et Nom d'un compte sont passées en paramètre, le solde est mis à 0.
- Les trois valeurs sont passées en paramètre.

2) Agir sur le solde d'un compte c'est pouvoir y faire des dépôts et des retraits.

- ❖ Ecrire les méthodes associées: déposer, retirer.

3) Créer une classe TestCompte possédant une méthode main dans laquelle on réalise les traitements suivants:

- Création des trois comptes suivants :

  Compte C_1 : Nom :"benzerafa ".
  Compte C_2 : Nom :"bellala" , solde : 3000.
  Compte C_3 : Nom :"boussouf", solde :5600.

- Afficher le numéro et le solde du compte C_2.
- Dépôt de 500 dans le compte C_1 .
- Retrait de 200 du compte C_2.

4) On souhaite enrichir la classe Compte par la méthode publique **EstSuperieur** dont la signature et l'objectif sont donnés ci-dessous:

```
public boolean EstSuperieur (Compte C)
        {
            Renvoie : vrai si le solde de l'objet est supérieur au solde de C
                      faux sinon
                                }
```

- Ecrire la méthode EstSupérieur.

5) Ecrire une méthode transfert permettant de transférer un montant d'un compte à un autre.

6) Un compte bancaire est soit un compte chèque, soit un compte épargne. Donc un compte d'épargne est un compte bancaire qui a, en plus, un taux d'intérêt (TauxR).

- Proposer une classe Compte-Epargne avec une méthode AjouterInterets qui ajoute au solde les intérêts annuels.

Exercice n°4-3:

Une Ville est caractérisée par son Nom, son Code_postal et sa Population.

- Le Nom d'une Ville et son Code_postal sont invariables ; ils doivent être connu dès l'instanciation de l'objet.

1) Ecrire une classe Ville avec les attributs et les méthodes nécessaires.

2) Ajouter à la classe Ville la méthode décrisToi permettant de renvoyer toutes les informations d'une ville encapsulées sous la forme d'une chaîne de caractères.

3) Ajouter également une variable Nbr_Villes de classe permettant de compter le nombre de villes qui seront instanciées.

4) Créer une classe TestVille possédant une méthode main dans laquelle on réalise les traitements suivants:

- Création des trois instances de la classe Ville suivantes :

  Ville V_1 : Nom :"Jijel ",Code_postal : 18000, Population : 450000.
  Ville V_2 : Nom :"Sétif", Code_postal : 19000.
  Ville V_3 : Nom :"Oran", Code_postal : 31000, Population : 700000.

- Afficher les informations de la ville V_3.
- Ajouter 2000 habitants à la Population de la ville V_1.

5) Une grande ville est une ville ayant au moins 500 000 habitants.

- Ajouter à la classe Ville une méthode estGrandVille permettant de vérifier s'il s'agit d'une grande ville ou non.

6) Une Capitale est une grande ville particulière contenant en plus le pays où elle est située.

- Créez une classe Capitale héritant de la classe Ville avec les attributs et les méthodes nécessaires.

7) On souhaite maintenant mémoriser un ensemble de villes.

- Ecrire la classe Tableau-Villes qui implémente une structure de tableau pour le stockage des villes. Cette classe doit proposer les deux méthodes suivantes:

  - insère : ajouter une nouvelle Ville.
  - rechercher : qui restitue l'objet Ville dont le nom est passé en paramètre.

8) Un Pays est constitué d'un ensemble de villes et d'une capitale.

- Proposer une classe Pays avec les attributs et les méthodes nécessaires.
- Ecrire la méthode population qui renvoie le nombre d'habitants du pays.
- Ecrire la méthode descript_villes permettant de décrire toutes les villes du pays.

Exercice n°4-4:

Une pile est une structure de données où les données sont retirées dans l'ordre inverse où elles ont été ajoutées. Le dernier élément ajouté est le premier élément retiré (Last In, First out, LIFO en Anglais). Les éléments y sont rajoutés après les éléments déjà présents.

Par contre, une file d'attente est une structure de données où les données sont retirées dans l'ordre où elles ont été ajoutées. Le premier élément ajouté est le premier élément retiré (First In, First out, FIFO en Anglais). De même, les éléments y sont rajoutés après les éléments déjà présents.

Ecrire deux classes *PileEntiers* et FileEntiers en donnant pour chacune :

1) Un constructeur.

2) les méthodes estVide, estPleine : pour tester respectivement si cette structure est vide, ou si elle est pleine.

3) les méthodes ajouter et retirer qui ajoute un élément dans cette structure et en retire un élément.

Exercice n°4-5:

Un couple de lapins a sa première portée au bout de 2 mois, ensuite une portée tous les mois. Chaque portée est un couple de lapins. Tous les couples ainsi obtenus se reproduisent de la même manière. La durée de vie moyenne d'un lapin est de 8 mois.

Donner une implémentation calculant:

- le nombre de lapins au bout de cinq ans.
- au bout de combien de temps le nombre de lapins dépassera dix millions ?

Références bibliographiques

[1] MC. Belaid, "*Architecture des ordinateurs*", Edition les Pages Bleues Internationales, 2004.

[2] M. Gautier, "*Compilation des langages de programmation : Ce que fait un compilateur, comment le réaliser*", Edition Ellipses Marketing, 2006.

[3] HM. Deitel, PJ. Deitel," *Comment programmer en Java*", Edition Reynald Goulet, 2002.

[4] TH. Cormen, "*Algorithmique*", Edition Dunod, 2010.

[5] H. Bersini, "*La programmation orientée objet*", Edition Eyrolles, 2017.

[6] L. Gervais, "*Apprendre la Programmation Orientée Objet avec le langage Java*", Edition Eni, 2014.