Chapitre 1: Introduction

L'informatique, contraction d'*information* et *automatique*, est la science du traitement de l'information. Apparue au milieu du 20ème siècle, elle a connu une évolution extrêmement rapide. A sa motivation initiale, qui était de faciliter et d'accélérer le calcul, se sont ajoutées de nombreuses fonctionnalités, comme l'automatisation, le contrôle et la commande de processus, la communication ou le partage de l'information. L'ordinateur est, pour l'instant, la principale icône de l'informatique.
Depuis l'arrivée de l'ordinateur, on a essayé, sans cesse, d'exploiter au maximum cette machine et dans tous les domaines : industrie, gestion, calculs scientifiques et techniques, enseignement, etc.

1- Description d'un ordinateur

Un ordinateur est une **machine automatique** commandée par des **programmes** enregistrés dans sa mémoire. Il est capable d'effectuer des opérations variées, sur les données proposées, à une grande vitesse, sans risque d'erreur (à condition que les programmes soient corrects). L'utilisateur fournit des données, l'ordinateur effectue sur ces données les traitements pour rendre des résultats.
La partie matérielle (communément nommée Hardware) est composée de pièces internes (carte mère, processeur, carte graphique, etc…..) et d'autres externes (dits périphériques) alors que le software est l'ensemble des programmes qui restent immatériels même s'ils sont stockés physiquement sur des supports mémoires.

1.1 De quoi est composée une machine ?

Les éléments de base composant un ordinateur sont :

1. un écran pour que votre ordinateur vous parle ;
2. un clavier, pour que vous puissiez écrire à votre ordinateur ;

Figure 1. Un *Ordinateur et ses différents périphériques*

3. une souris, pour déplacer le curseur à l'écran pour parler à votre ordinateur un langage de signe ;
4. des enceintes pour le son (ce n'est pas obligatoire mais tout de même mieux) pour que vous entendiez ce que vous dit votre ordinateur ;
5. et surtout : une unité centrale[1] qui est le cœur et le cerveau de l'ordinateur.

A cette composition de base, il est possible d'y ajouter divers appareils électroniques qui assurent diverses fonctionnalités (Voir figure 1). C'est ce que l'on appelle les périphériques ; tels que :

1. Imprimante pour que votre ordinateur vous écrive des choses ;
2. Scanner pour que vous puissiez envoyer des photos à votre ordinateur ;
3. Webcam pour que votre ordinateur puisse vous voir ;
4. Disque dur externe, clé USB, carte mémoire... pour que votre ordinateur puisse garder en mémoire beaucoup de choses ;
5. Manette de jeu pour que vous puissiez jouer avec votre ordinateur
6. … etc.…

1.2 Brancher son ordinateur

Chaque branchement a une forme et une couleur bien définie, ce qui fait qu'il est presque impossible de se tromper. De plus à l'heure actuelle, la plupart des périphériques d'un ordinateur se branchent tous via un branchement universel : le port USB[2].

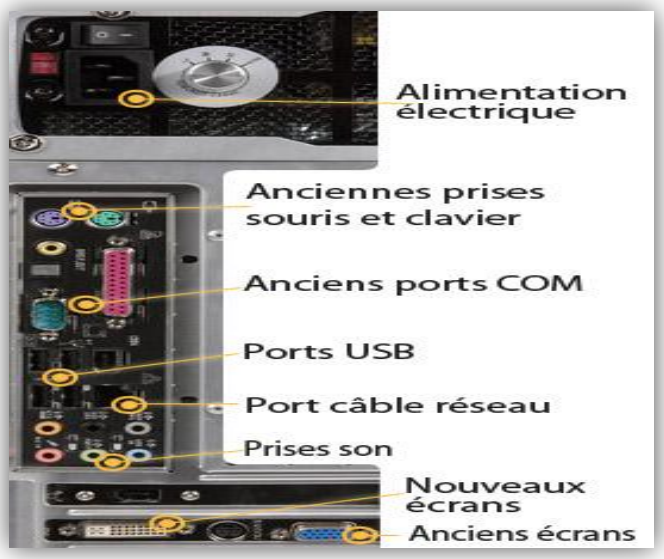


Figure 2. L'*arrière d'une unité centrale*

[1] L'unité centrale est le boitier contenant tout le matériel électronique permettant à l'ordinateur de fonctionner. Les périphériques y sont reliés (C'est dans l'unité centrale que l'on insère un disque par exemple.)

[2] USB est l'acronyme de « Universal Serial Bus » en anglais. C'est un branchement rectangulaire qui se veut universel : presque tout le matériel actuel se branche via USB à votre ordinateur. Les ordinateurs possèdent maintenant des ports USB à l'arrière comme à l'avant de l'unité centrale, mais aussi parfois sur votre écran.

Dans l'unité centrale, chaque branchement est indiqué par une couleur (voir figure 2) :

- L'alimentation électrique, qui est reliée directement à une prise secteur. Un bouton 0 - 1 permet de couper l'arriver du courant ;
- Les anciennes prises pour clavier et souris, rondes et vertes ou violettes ;
- Les anciens ports COM et parallèles, qui ne sont plus utilisés de nos jours ;
- Les ports USB, au nombre de 4 sur la photo, permettant de brancher divers périphériques. Ce sont actuellement les ports les plus utilisés !
- Le port pour brancher l'ordinateur à Internet ou sur un réseau ;
- Les prises son : pour brancher enceintes, caisson de basses, micro ;
- Un branchement DVI (blanc rectangulaire) pour brancher les nouveaux écrans et un VGA (bleu rectangulaire) pour les anciens écrans.

Ces branchements peuvent varier d'un ordinateur à l'autre selon son ancienneté. On retrouvera presque toujours par contre les ports USB.

1.3 Principe de fonctionnement d'un ordinateur

Les deux principaux constituants d'un ordinateur sont la mémoire principale et le processeur. Alors que la mémoire principale permet de stocker des informations (programmes et données), le processeur exécute pas à pas les instructions qui composent les programmes.

a. Les programmes

Un programme est une suite d'instructions élémentaires qui vont être exécutée dans l'ordre. Ces instructions correspondent à des actions très simples, comme additionner deux nombres,

Un ordinateur a obligatoirement un programme (appelé système d'exploitation) qui permet de gérer son fonctionnement depuis sa mise sous tension (alimentation en électricité) jusqu'à son extinction. D'autres programmes (dits aussi logiciels) répondent aux différents besoins des utilisateurs que ce soient des besoins de calculs (calculer la paie des employés), d'impression (saisir, mettre en page et imprimer un livre), de divertissement (lecteur vidéo, jeux), de navigation, etc.

b. La mémoire principale

Elle contient les instructions du ou des programmes en cours d'exécution et les données associées à ce(s) programme(s). Physiquement, elle se décompose souvent en :

- une mémoire morte (ROM = Read Only Memory) chargée de stocker le programme. C'est une mémoire à lecture seule.

- une mémoire vive (RAM = Random Access Memory) chargée de stocker les données intermédiaires ou les résultats de calculs. On peut lire ou écrire des données dedans, ces données sont perdues à la mise hors tension.

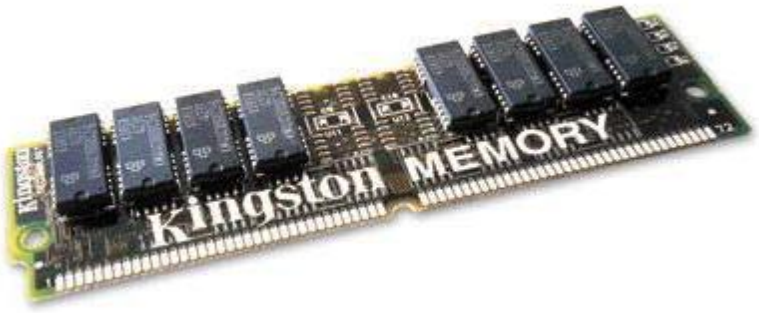


Figure 3. Un aperçu d'une RAM

Note : Les disques durs, clés USB, CDROM, etc… sont des périphériques de stockage et sont considérés comme des mémoires secondaires.

Une mémoire peut être représentée comme une armoire de rangement constituée de différents tiroirs. Chaque tiroir représente alors une case mémoire qui peut contenir un seul élément : des données. Le nombre de cases mémoires pouvant être très élevé, il est alors nécessaire de pouvoir les identifier par un numéro. Ce numéro est appelé adresse. Chaque donnée devient alors accessible grâce à son adresse.

c. Le processeur central

Le processeur est parfois appelé CPU (de l'anglais Central Processing Unit) ou encore MPU (Micro Processing Unit) pour les microprocesseurs. Un microprocesseur n'est rien d'autre qu'un processeur dont tous les constituants sont réunis sur la même puce électronique (pastille de silicium), afin de réduire les coûts de fabrication et d'augmenter la vitesse de traitement. Les microordinateurs sont tous équipés de microprocesseurs. L'architecture de base des processeurs équipant les gros ordinateurs est la même que celle des microprocesseurs.

2. Instructions de base d'un ordinateur

Un ordinateur contient un circuit, le processeur, qui permet d'effectuer de petits traitements de base qu'on appelle instructions et qui sont la base de tout ce qu'on trouve sur un ordinateur.

En effet, contrairement à une calculatrice, dont le rôle se limite à réaliser des opérations de calcul (le plus souvent arithmétiques), un ordinateur assure des opérations de traitement de l'information, c'est-à-dire qu'il exécute successivement des opérations en suivant les directives d'un algorithme. Ce traitement est mené à l'aide d'instructions plus ou moins sophistiquées, et plus ou moins proches du microprocesseur.

Une instruction informatique est une commande unique, représentée par un symbole (numérique ou alpha-numérique), et dont la finalité est prédéfinie: de la plus simple (déplacer l'index du microprocesseur dans la mémoire, additionner deux nombres) à la plus sophistiquée et abstraite (par exemple les instructions de gestion de classes du langage Java).
Une instruction est l'opération élémentaire que le processeur peut accomplir. Les instructions sont stockées dans la mémoire principale, en vue d'être traitée par le processeur. Une instruction est composée de deux champs :

- le code opération, représentant l'action que le processeur doit accomplir ;
- le code opérande, définissant les paramètres de l'action. Le code opérande dépend de l'opération. Il peut s'agir d'une donnée ou bien d'une adresse mémoire.

Les instructions peuvent être classées en catégories dont les principales sont :

- Accès à la mémoire : des accès à la mémoire ou transferts de données entre registres.
- Opérations arithmétiques : opérations telles que les additions, soustractions, divisions ou multiplication.
- Opérations logiques : opérations ET, OU, NON, NON exclusif, etc.
- Contrôle : contrôles de séquence, branchements conditionnels, etc.

On appelle jeu d'instructions l'ensemble des opérations élémentaires qu'un processeur peut accomplir. Le jeu d'instruction d'un processeur détermine ainsi son architecture, sachant qu'une même architecture peut aboutir à des implémentations différentes selon les constructeurs. Le processeur travaille effectivement grâce à un nombre limité de fonctions, directement câblées sur les circuits électroniques. La plupart des opérations peuvent être réalisées à l'aide de fonctions basiques. Certaines architectures incluent néanmoins des fonctions évoluées courantes dans le processeur.
Le module architecture des ordinateurs vous donnera plus d'information sur les instructions et leur exécution.

3. Différentes phases de résolution d'un problème par ordinateur

Un programme informatique n'est pas un élément de résolution en lui-même. Ce n'est qu'un automate matérialisant un schéma de résolution pour produire les grandeurs cherchées quand on lui injecte les grandeurs connues. Le travail de résolution du problème est donc totalement à la charge du programmeur qui doit transformer une méthode en un composant opérationnel qui est **le programme**.

Parlons plus clairement ! La résolution informatique d'un problème comporte plusieurs phases préliminaires à l'exécution du programme, lesquelles sont présentées ci-dessous.

1- Phase d'étude

Cette phase sert à inventorier ce qui est connu ou observable et ce qui est à connaître. On identifie ensuite les relations entre les grandeurs connues et les grandeurs à connaître, ce qui détermine le **modèle**.

2- Phase de réalisation du modèle

Le fait d'avoir un modèle n'implique pas forcément qu'on dispose d'une méthode pour le réaliser. Le travail de réalisation du modèle consiste à déterminer un enchaînement d'opérations produisant les grandeurs cherchées à partir des grandeurs connues, en respectant le modèle. Cet enchaînement d'actions constitue en fait le **schéma de résolution**.

En l'absence du schéma de résolution pour un modèle donné, on essaie de simplifier ce dernier en le remplaçant par un modèle approché pour lequel on dispose d'une méthode de résolution.

Note : les deux phases précédentes constituent la **phase d'Analyse**.

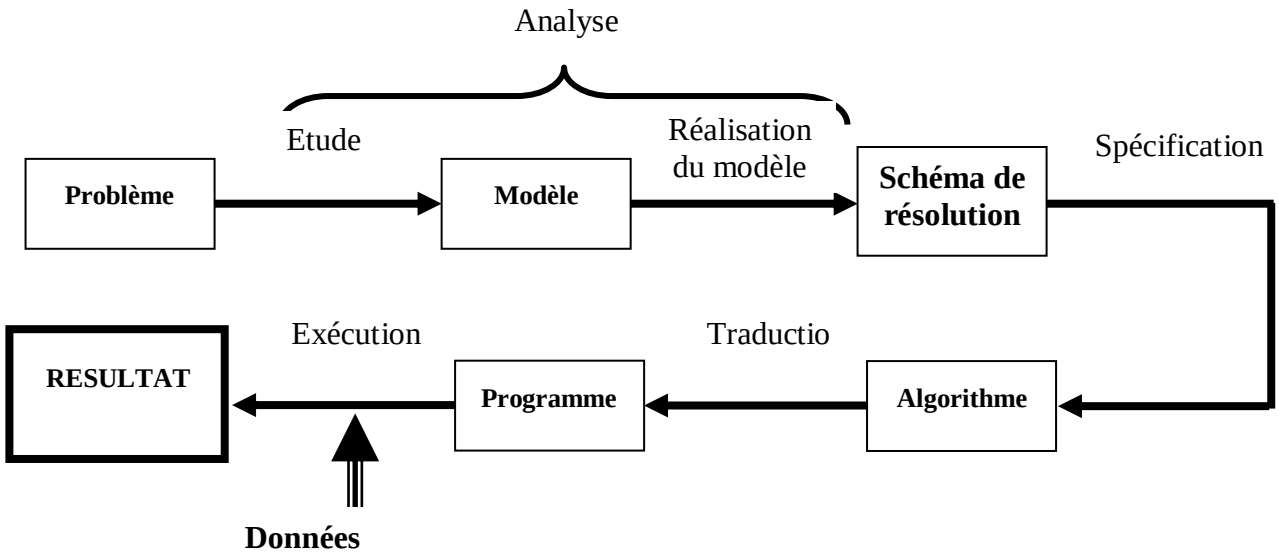


Figure 4. Les phases de résolution d'un problème.

3- Phase de spécification

Le schéma de résolution est souvent exprimé d'une manière pas assez claire ou il est d'un niveau sémantique trop élevé par rapport aux opérations informatiques. Il convient alors de l'exprimer d'une manière précise et complète en termes informatiques. Cette phase de spécification produit les **algorithmes** (descriptions des traitements) et les **descriptions de données**. A ce niveau, on dispose d'un processus informatique de résolution sans lien avec une machine particulière ; ce processus n'est donc pas opérationnel.

4- Phase de traduction

Pour la mise en œuvre effective du processus informatique de résolution sur une machine particulière, on doit traduire les algorithmes et les descriptions de données dans un langage de programmation disponible sur cette machine. On obtient alors un programme directement interprétable ou compilable, puis exécutable sur cette machine.

Conclusion

Ce chapitre bien que comportant beaucoup de « blabla » est nécessaire pour la bonne compréhension de l'algorithmique et de ses bases qui seront détaillés dans les chapitres suivants.

Chapitre 2: Algorithme

I- Définition

Le mot « Algorithme » خوارزم est un dérivé du nom de l'illustre savant musulman Muhammad Ibn Musa Al Khawarizmi qui vécut au 9ème siècle (2ème Hidjri), sous le règne du calife abbasside Al-Mamun.

Al Khwarizmi a exposé les méthodes de base pour l'addition, la multiplication, la division, l'extraction de racines carrées ainsi que le calcul des décimales de π. Ces méthodes sont précises, sans ambiguïté, mécaniques, efficaces, correctes. **Ces méthodes sont des algorithmes!**

L'algorithmique est l'art de construire des algorithmes !

Rappelons que les étapes de résolution d'un problème quelconque sont :

1. Comprendre l'énoncé du problème
2. Décomposer le problème en sous-problèmes plus simples à résoudre
3. Associer à chaque sous problème, une spécification :

- Les données nécessaires ;
- Les données résultantes ;
- La démarche à suivre pour arriver au résultat en partant d'un ensemble de données.

4. Elaboration d'un plan d'actions (algorithme)

Informatiser une application, facturation de la consommation d'eau par exemple, c'est faire réaliser par un ordinateur, une tâche qui était réalisée par l'Homme. Pour cela il faut tout d'abord, détailler suffisamment les étapes de résolution du problème, pour qu'elles soient exécutables par l'homme. Ensuite, transférer la résolution en une suite d'étapes élémentaires et simples à exécuter. Toute suite d'étapes, si élémentaires et simples à exécuter, s'appelle un ALGORITHME.

II- Caractéristiques d'un algorithme

L'algorithme est un moyen pour le programmeur de présenter son approche du problème à d'autres personnes. En effet, un algorithme est l'énoncé dans un langage bien défini d'une suite d'opérations permettant de répondre au problème. Un algorithme doit donc être :

- **Lisible** : l'algorithme doit être compréhensible même par un non-informaticien
- **De haut niveau** : l'algorithme doit pouvoir être traduit en n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un programme particulier ou bien à un système d'exploitation donné

- **Précis** : chaque élément de l'algorithme ne doit pas porter à confusion, il est donc important de lever toute ambiguïté
- **Concis** : un algorithme ne doit pas dépasser une page. Si c'est le cas, il faut décomposer le problème en plusieurs sous-problèmes
- **Structuré** : un algorithme doit être composé de différentes parties facilement identifiables à savoir : une entête où apparaissent le nom de l'algorithme ainsi que les déclarations des objets manipulés puis le corps de l'algorithme qui commence par "Début" et se termine par "Fin. Entre ces deux mots clés se trouvent les actions ordonnées à exécuter par la machine.

```
Algorithme Nom_de_l'algorithme ;
Variables                          } Entête
    Liste des variables ;
Début
    Action 1 ;
    Action 2 ;
    Action 3 ;                     } Corps
    ….
    Action n ;
Fin.
```

Les données dont on dispose au début, les résultats que l'on doit obtenir à la fin ainsi que les objets internes utilisés pour arriver à ces résultats sont ceux déclarés dans l'entête alors que les actions du corps lorsqu'elles sont appliquées dans l'ordre permettent d'accéder au résultat attendu.

Notes :

! Chaque mot clé est souligné.
! Une marque de terminaison (;) est utilisée après chaque action suivie d'une autre action.

III- Définition d'une variable et de ses caractéristiques

Une variable est une entité qui contient une information. Elle possède :

- Un nom, on parle d'identifiant ;
- Un type qui caractérise l'ensemble des valeurs que peut prendre la variable :
    - Nombre Entier
    - Nombre flottant (Réel)
    - Booléen (avec pou seules valeurs *Vrai* et *Faux*)
    - Caractère (alphabétique, numérique)
    - Chaine de caractères (mot ou phrase)

A un type donné, correspond un ensemble d'opérations définies pour ce type.

1- Opérateur, opérande et expression

- Un opérateur est un symbole d'opération qui permet d'agir sur des variables ou de faire des "calculs"
- Un opérande est une entité (variable, constante ou expression) utilisée par un opérateur
- Une expression est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type.

2- Opérateurs numériques

- On retrouve tout naturellement les opérations usuelles : +, -, *, /
- Avec en plus pour les entiers div et mod, qui permettent respectivement de calculer une division entière et le reste de cette division.
- L'opérateur d'égalité, que l'on retrouve chez tous les types simples, permet de savoir si les deux opérandes sont égaux. Il est représenté par le caractère = alors que l'opérateur d'inégalité est représenté par ≠. Et pour les types avec des valeurs ordonnées, il existe des opérateurs de comparaison <, ≤, >, ≥

3- Opérateurs booléens

- Les opérateurs sont essentiellement : Non, Et, Ou et OuExclusif
- Associativité, Commutativité et Distributivité sont les propriétés des opérateurs "Et" et "Ou"

Notes :

- Tout comme en arithmétique les opérateurs ont des priorités. Par exemple
  * et / sont prioritaires sur + et -
  Pour les booléens, la priorité des opérateurs est : Non, Et, ouExclusif et ou
- Pour clarifier les choses (ou pour supprimer toute ambiguïté) on peut utiliser des parenthèses.

IV- Actions de base

Afin de résoudre un problème quelconque il est impératif de (1) préciser les données ; (2) réaliser le traitement ; (3) Afficher les résultats.

En effet, pour réaliser une opération d'addition « A+ B » il est nécessaire de connaitre, au départ, les valeurs des données A et B ; puis réaliser le calcul et à la fin donner le résultat de la somme.

Trois actions sont donc essentielles afin d'écrire des algorithmes de traitement :

- Deux actions qui permettent d'établir la communication entre l'être humain et la machine : Lire / Ecrire
- Une action qui permet à la machine de réaliser des manipulations et des calculs : Affectation.

1- Action de Lecture

Pourquoi ?

Pour transmettre **les valeurs des données** de chez l'utilisateur (l'être humain) vers la machine.

Lecture : Utilisateur → Machine

Syntaxe

Lecture d'une seule variable : **Lire (Nom-variable) ;**

Lecture de plusieurs (n) variables : **Lire (Nom-variable-1, Nom-variable-2, …, Nom-variable-n) ;**

Exemple

Afin de résoudre une équation du second degré de la forme $a\ X^2 + b\ X + C = 0$ les valeurs des coefficients a, b et c connues par l'utilisateur doivent être transmises à l'ordinateur avant que la machine ne commence le calcul ; à savoir le calcul du discriminant delta.

Le seul moyen d'établir cette communication est le groupe d'instructions :

```
Lire (a) ;
Lire (b) ;
Lire (c) ;
```

qui peut être regroupé dans une seule instruction :

```
Lire (a, b, c) ;
```

2- Action d'écriture (affichage)

Pourquoi ?

Pour transmettre **les valeurs des résultats** calculés par la machine vers l'utilisateur (l'être-humain) ; ou encore transmettre un message textuel.

Ecriture : Machine → Utilisateur

Syntaxe

Affichage d'une seule variable : **Ecrire (Nom-variable) ;**

Affichage de plusieurs (n) variables : **Ecrire (Nom-variable-1, … , Nom-variable-n) ;**

Affichage d'un message : **Ecrire ('Texte') ;**

Exemple

La résolution d'une équation de second degré dans les réels donne lieu soit à des valeurs de X_1 et X_2 ou à un texte dans le cas où aucune valeur ne peut être calculée :

Dans le cas ou delta est supérieur ou égal à zéro : Ecrire (X_1, X_2) ;

Dans le cas ou delta est inférieur à zéro et donc pas de solution dans $\mathbb{R}$:

```
Ecrire ('pas de solution dans R') ;
```

3- Action d'affectation

Pourquoi ?

Pour donner une nouvelle valeur (contenu) à une variable ; cette valeur peut être un résultat de calcul.

Affectation : Machine → Machine

Syntaxe

Affectation d'une valeur à une variable : **Nom-variable ← valeur ;**

Affectation d'une variable à une variable : **Nom-variable1 ← Nom-variable2 ;**

Affectation du résultat de calcul à une variable :

Nom-variable ← nom-variable1 opérateur nom-variable2 ;

Ou encore l'affectation du résultat de plusieurs calcul à une variable :

Nom-variable ← nom-variable1 opérateur1 nom-variable2 … opérateur-n nom-variable n ;

Exemple

La résolution d'une équation de second degré se fait par calcul du discriminant delta :

delta ← b*b – 4*a*c ;

Dans le cas de delta > 0 la machine réalise le calcul des deux solutions X_1 et X_2 en exécutant les instructions :

X_1 ← -b + √delta / 2*a ;

X_2 ← -b - √delta / 2*a ;

4- Exercices

1- Ecrire un algorithme qui permet de multiplier un nombre par 2, lui ajouter 7, soustrait le nombre de départ puis afficher le résultat.

```
Algorithme Exo1 ;
Declaration
        x, y : entier ;
Debut
        Lire (x) ;   /* il faut bien connaître la valeur de départ sinon comment faire les calculs demandés !*/
        y ← x*2 ;
        y ← y + 7 ;
        y ← y – x ;
        Ecrire (y) ;  /* si on n'affiche pas le résultat sur écran on ne le connaîtra jamais*/
Fin.
```

Exécution de cet algorithme pour la valeur de x = 2 :

x	Y	Ecran
2		
	4	
	11	
	9	
		9

Exécution de cet algorithme pour la valeur de x = 10 :

X	Y	Ecran
10		
	20	
	27	
	17	
		17

2- Soit l'algorithme suivant :

```
Algorithme Exo2 ;
Declaration
        x, y, z : réel ;
Debut
        Lire (x, y) ;
        z ← x ;
        x ← y ;
        y ←z ;
        Ecrire (x, y) ;
Fin.
```

Quel est le rôle de cet algorithme ?

Pour répondre à cette question il est nécessaire d'exécuter l'algorithme.

Exécution de cet algorithme pour les valeurs x = 2.5 et y = 6.0

X	y	Z	Ecran
2.5	6.0		
		2.5	
6.0			
	2.5		
			6.0 2.5

Le résultat est x = 6.0 et y = 2.5

Il parait que cet algorithme permet d'échanger les valeurs de deux réels en utilisant une variable intermédiaire z.

Pour vérifier ce rôle il est impératif de faire au moins une autre exécution avec d'autres valeurs.

Prenons x = 101.67 et y = -18.3

X	y	Z	Ecran
101.67	-18.3		
		101.67	
-18.3			
	101.67		
			-18.3 101.67

Le résultat est x = -18.3 et y = 101.67

3- Ecrire un algorithme qui calcul le prix à payer pour une marchandise à partir d'un prix initial et d'une remise sur ce prix. La formule à appliquer est PaP = PI – PI*R/100

Exemple : Prix Initial (PI) = 100 da, Remise (R) = 25 → le Prix à payer (PaP) = 75 da

```
Algorithme Exo3 ;
Declaration
        PI, R, PaP : reel ;
Debut
        Ecrire ('Donnez SVP le prix initial et la remise') ;
        Lire (PI, R) ;
        PaP ← PI – PI* R / 100;
        Ecrire ('Le prix à payer =', PaP) ;
Fin.
```

V- Tests

Toutes les actions ne sont pas réalisables à chaque fois. En effet, s'il pleut l'action prendre le parapluie a un sens sinon elle est annulée ou remplacée par une autre comme prendre les lunettes de soleil.

Les actions de l'algorithmique peuvent, elles aussi, dépendre de certaines conditions et il existe plusieurs formes de tests en algorithmique.

1- Conditions

Une condition est une comparaison sous la forme

Valeur <opérateur-comparaison> Valeur

Les opérateurs de comparaison sont :

- Egal (=)
- Différent (≠)
- Supérieur (>)
- Inférieur (<)
- Supérieur ou égal (≥)
- Inférieur ou égal (≤)

Notes :

! La valeur peut être contenue dans une variable déclarée ou le résultat d'un calcul.

! Attention aux raccourcis du langage naturel qui peuvent regrouper dans une même phrase deux ou plusieurs opérateurs ce qui est valide en mathématique mais non en algorithmique.

Exemple âge compris entre 15 et 18 en mathématique il est correct d'écrire 15 ≤ âge ≤ 18 mais comme il y a deux (2) opérateurs (deux ≤) cela est interdit dans une même condition algorithmique.

2- Actions conditionnelles

Quand ?

Il s'agit dans ce cas de conditionner un groupe d'actions (qui peut contenir de une à plusieurs instructions) selon un certain test. Si la condition est vérifiée alors le groupe d'action est déclenché puis les actions suivantes sont exécutées. Dans le cas où la condition est fausse (non vérifiée) un saut est effectué vers les actions qui suivent le groupe conditionné.

Syntaxe

```
Si Condition Alors
        Instruction 1 ;
        Instruction 2 ;
        ¦
        ¦
        Instruction n ;
FinSi ;
```

Exemple

Ecrire un algorithme qui calcul et affiche la valeur absolue d'une valeur entière M.

```
Algorithme Condition ;
Declaration
        M : entier ;
Debut
        Lire (M) ;
        Si M < 0 Alors
                M ← - M ;
        FinSi ;
        Ecrire (M) ;
Fin.
```

L'exécution de cet algorithme pour la valeur x = 4 fait que les instructions **Lire (x)** et **Ecrire (x)** sont les seules à être exécutées car l'évaluation de la condition x < 0 rend **faux.**

L'exécution de cet algorithme pour la valeur x = -4 fait que toutes les instructions de l'algorithme sont exécutées car la condition x < 0 est évaluée à **vrai.**

3- Actions alternatives

Quand ?

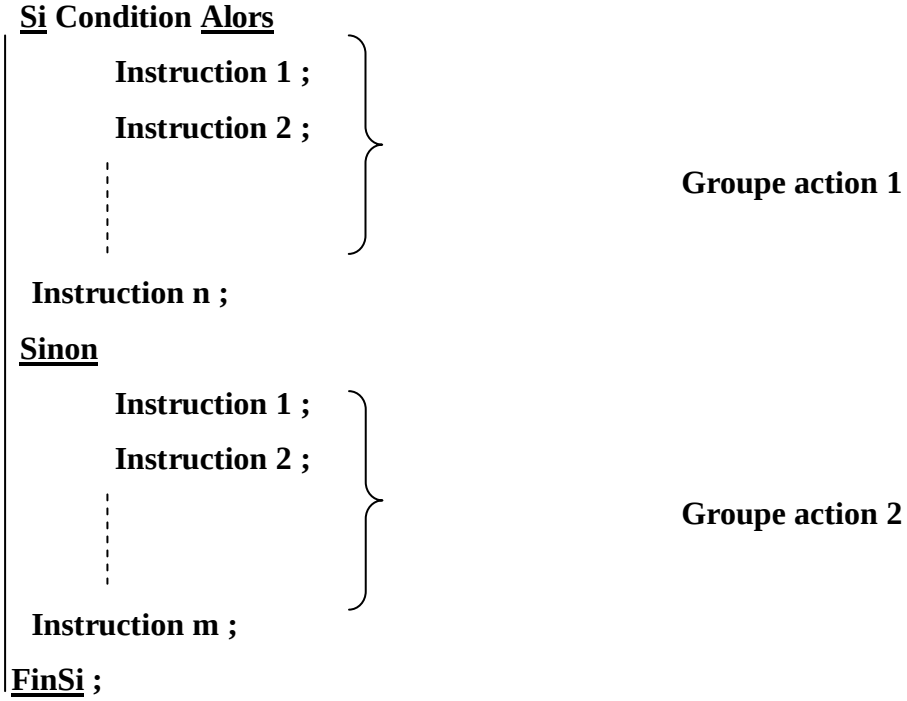
Il s'agit dans ce cas de conditionner deux groupes d'actions (qui peuvent contenir chacun de une à plusieurs instructions) selon un certain test. Si la condition est vérifiée alors le groupe d'action 1 est déclenché puis les actions qui suivent le FinSi sont exécutées. Dans le cas où la condition est fausse (non vérifiée) un saut est effectué vers le groupe actions 2 qui sera exécuté puis suivront les instructions qui suivent le FinSi.

Il y a donc obligatoirement une exécution de l'un ou l'autre des groupes d'action mais que si l'un est exécuté l'autre ne l'est pas.

Note :

! L'alternative n'admet que deux chemins (vrai ou faux)

Syntaxe



Exemple

Ecrire un algorithme qui calcul et affiche la nature (positive ou négative) d'un nombre entier. Le zéro est considéré comme valeur positive.

```
Algorithme Alternative ;
Declaration
        X : entier ;
Debut
        Lire (x) ;
        Si x < 0 Alors
                Ecrire ('Le nombre est négatif')
        Sinon
                Ecrire ('Le nombre est positif')
        FinSi ;
Fin.
```

L'exécution de cet algorithme pour la valeur x = 6 nous conduit à suivre le chemin avec les instructions Lire (x) et Ecrire ('Le nombre est positif') de même que dans le cas de x = 0 ; alors que l'exécution avec x = -4 le chemin comprend les instructions Lire (x) et Ecrire ('Le nombre est négatif')

4- Tests imbriqués

Quand ?

Dans certains cas il existe plus d'une alternative à une condition. Il s'avère alors que le test sur une valeur peut avoir plus de deux chemins possibles et des tests sont inclus dans d'autres. Cette forme est celle des tests imbriqués.

Syntaxe :

```
Si Condition 1 Alors
        Instruction 1 ;
        Instruction 2 ;
        .
        .
        Instruction n ;                    Groupe action 1
Sinon
    Si Condition 2 Alors
        Instruction 1 ;
        .
        .
        Instruction m ;                    Groupe action 2
    Sinon
        .
        .                                  Groupe action k
        Instruction p ;
    FinSi ;
FinSi ;
```

Prenons le cas de l'algorithme de la nature d'un nombre entier (section précédente) et que l'on considère que le zéro est une valeur qui n'est ni positive ni négative mais qu'il est déclaré nul. Ce changement implique trois alternatives et non pas deux ce qui oblige l'informaticien à la solution avec tests imbriqués comme suit :

```
Algorithme Imbriqués ;
Declaration
        X : entier ;
Debut
        Lire (x) ;
        Si x < 0 Alors
                Ecrire ('Le nombre est négatif')
        Sinon
                Si x > 0 Alors
                        Ecrire ('Le nombre est positif')
                Sinon
                        Ecrire ('Le nombre est une valeur nulle')
                FinSi
        FinSi ;
Fin.
```

A l'exécution de cet algorithme un et un seul des trois chemins (groupe d'actions) est suivi selon que la première condition est vraie :

Lire (x)
Ecrire ('Le nombre est négatif')

Ou si la première condition est fausse et que la deuxième est vraie :

Lire (x)
Ecrire ('Le nombre est positif')

Ou encore si les deux premières conditions sont fausses :

Lire (x)
Ecrire ('Le nombre est une valeur nulle')

Les tests imbriqués remplacent un ensemble d'actions conditionnelles particulièrement lorsque les conditions portent sur la (ou les) même(s) variable(s).

Note :

! Dans le cas où les conditions des tests imbriqués portent sur des égalités par rapport à des valeurs précises ou des intervalles, ces tests peuvent être modélisés avec le constructeur **Selon** qui modélise le choix multiple en suivant la syntaxe suivante :

Selon variable
Valeur1 : groupe instructions 1 ;
Valeur2 : groupe instructions 2 ;
⋮
Valeur$_n$: groupe instructions n ;
FinSelon ;

Dans le cas ou la variable est égale à la valeur1 alors le groupe d'instruction 1 est le seul qui est exécuté ; si elle est égale à la valeur2 c'est le groupe 2 et ainsi de suite.

La syntaxe peut aussi suivre la forme suivante s'il s'agit de tester l'appartenance de la variable à des intervalles :

Selon variable
Intervalle 1 : groupe instructions 1 ;
Intervalle 2 : groupe instructions 2 ;
⋮
Intervalle 3 : groupe instruction n ;
FinSelon ;

Note :

! Dans le cas ou il existe un groupe d'instructions à exécuter en dernier recours (la variable n'est égale a aucun(e) intervalle (ou valeur) cité(e)) une ligne est rajoutée en fin pour le cas autre selon la syntaxe :

Selon variable
Valeur 1 : groupe instructions 1 ;
Valeur 2 : groupe instructions 2 ;
⋮
Valeur n : groupe instructions n ;
Autre : groupe instructions n+1 ;
FinSelon ;

Prenons l'exemple d'un algorithme qui affiche l'action équivalente à la lumière active dans des feux de circulations.

```
Algorithme choix ;
Declaration
        lumiere : chaine de caractère ;
Debut
        Lire (lumiere) ;
        Selon lumiere
        'orange' : Ecrire ('Attention !')
        'vert' : Ecrire ('voiture départ')
        'rouge' : Ecrire ('Voiture arrêt')
        Finselon ;
Fin.
```

5- Conditions composées

Quand ?

Comme déclaré précédemment (section 1) certains problèmes exigent parfois de formuler des conditions qui ne peuvent pas être exprimées sous la forme simple désignée ci-dessus. Les opérateurs Logiques (essentiellement **et**, **ou**, **non**) servent alors à composer les conditions.

L'exemple du test $15 \leq$ âge ≤ 18 est alors formulé par la condition :

Si (Age ≥ 15) et (Age ≤ 18) Alors

Syntaxe

Si (Condition 1) **opérateur logique 1** (Condition 2) **opérateur logique 2** … (Condition n) **Alors**

Pour l'opérateur « **et** » il est nécessaire que toutes les conditions soient vraies pour que le test soit vrai.
Dans le test ci-dessous il est nécessaire que l'Age appartient à [15, 18] pour que le test soit vrai.

Pour l'opérateur « **ou** » il suffit que l'une des conditions soit vraie pour que le test soit vrai
Dans le test **Si (Age ≥ 15) ou (Age ≤ 18) Alors** il suffit que l'Age soit supérieur ou égal à 15 ou qu'il soit inférieur ou égal à 18 pour que le test soit vrai.

Pour l'opérateur « **non** » si la condition est vraie le test devient faux et si elle est fausse alors le test sera évalué à vrai.
Dans le test **Si Non (Age ≥ 15) Alors**. Ce test est vrai si l'Age est inférieur à 15 et faux dans le cas contraire.

Exemple

Ecrire un algorithme qui affiche si le produit de deux nombres est positif ou non sans calculer le produit.

```
Algorithme Composition1 ;
Declaration
        a, b : entier ;
Debut
        Lire (a, b) ;
        Si  ((a ≥ 0) et (b ≥ 0)) ou ((a < 0) et (b < 0)) Alors
                Ecrire ('Le produit est positif')
        Sinon
                Ecrire ('Le produit est négatif')
        FinSi ;
Fin.
```

Une autre solution peut être la suivante :

```
Algorithme Composition2 ;
Declaration
        a, b : entier ;
Debut
        Lire (a, b) ;
        Si ((a < 0) et (b > 0)) ou ((a > 0) et (b < 0)) Alors
                Ecrire ('Le produit est négatif')
        Sinon
                Ecrire ('Le produit est positif')
        FinSi ;
Fin.
```

VI- Boucles

Dans un algorithme, certaines instructions peuvent être répétées et il ne serait pas très intelligeant de les réécrire. Les boucles sont alors apparues afin de modéliser la répétition. Trois formes se présentent aux informaticiens, l'utilisation de l'une ou l'autre est en rapport avec le test de répétition bien que la plus utilisée soit celle du Tantque.

1- Boucle Tantque

Une boucle **Tantque** permet de répéter plusieurs fois le même bloc d'instructions tant qu'une certaine condition reste vraie. La syntaxe est la suivante :

```
Tantque Condition Faire
    Instruction 1 ;
    Instruction 2 ;
    ….
    Instruction n
FinTantque ;
```

Si la condition est vraie le groupe d'instructions est exécuté puis le FinTantque remet l'exécution à la condition et ainsi de suite jusqu'à ce que la condition soit évaluée à faux.

Notes :

! Le groupe d'instruction peut ne jamais être exécuté si dès le départ la condition de la boucle n'est pas satisfaite. Attention donc aux boucles qui ne sont jamais exécutées.

! A l'intérieur de la boucle il est impératif de faire des changements de manière à ce que le test soit erroné à un moment ou l'autre sinon si la condition reste toujours vraie on se retrouve dans une situation de boucle infinie qui est formellement interdite en algorithmique.

Exemple 1

Ecrire un algorithme qui calcul le nombre de valeurs entières saisies pour aboutir à une somme ne dépassant pas 500.

```
Algorithme Somme ;
Declaration
        x, som, co : entier ;
Debut
        sim ← 0 ;
        co ← 0 ;
        Tant que  som < 500 Faire
                Lire (x) ;
                som ← som + x ;
                co ← co + 1 ;
        FinTantque ;
        Si som > 500 Alors
                co ← co - 1 ;
        FinSi ;
        Ecrire (co)
Fin.
```

Exemple 2

Ecrire un algorithme qui affiche la division de 15 sur x avec obligation de ne jamais diviser sur le zéro.

```
Algorithme boucle1 ;
Declaration
        x : entier ;
        y : reel ;
Debut
        Lire (x) ;
        Tantque (x < 0) Faire
                Ecrire ('Donnez un nombre positif') ;
                Lire (x) ;
        Fintantque ;
        y ← 15 / x ;
        Ecrire (y) ;
Fin.
```

2- Boucle Pour

Lorsque le nombre de répétitions est connu, la boucle **Pour** utilise un compteur pour calculer le nombre de fois qu'est répété le bloc d'instructions de la boucle. La syntaxe est la suivante :

Pour compteur ← valeur-initiale à valeur-finale Pas = nombre Faire
    Instruction 1 ;
    Instruction 2 ;
    ….
    Instruction n
FinPour ;

Le compteur est initialisé à la valeur-initiale puis le groupe d'instruction est exécuté. Une fois arrivée à la FinPour, celle-ci remet l'exécution à la ligne Pour en rajoutant, **automatiquement**, la valeur du pas au compteur puis sa valeur est comparée à la valeur-finale. Dans le cas où il n'a pas atteint cette valeur la boucle est encore une fois exécutée mais si l valeur est atteinte la boucle Pour signale que c'est la dernière fois qu'elle peut être exécutée et donc arrivé à FinPour le retour n'est plus permis et l'exécution continue avec les instructions qui suivent la boucle.

Notes :

! Le pas, dont la valeur est entière, peut être positif (et le compteur est incrémenté par cette valeur à chaque itération) ou négatif et c'est une décrémentation.

! Si le pas = 1 il est généralement omis.

! La valeur du compteur étant automatiquement modifiée par la boucle Pour, il est strictement interdit d'écrire à l'intérieur de la boucle des instructions changeant la valeur de ce compteur.

Exemple

Ecrire un algorithme qui calcul la factorielle d'un nombre entier positif selon la formule :

Fact (x) = 1*2*3*….*x

```
Algorithme Factorielle ;
Declaration
        x, fact, co: entier ;
Debut
        Lire (x) ;
        fact ← 1 ;
        Pour co← 2 à x Faire
                fact ← fact * co ;
        FinPour ;
        Ecrire (fact) ;
Fin.
```

3- Boucle Repeter

La répétition en utilisant Repeter répond aux mêmes conditions que celles du Tantque sauf que la condition se trouve en fin de la boucle et non pas au début.

La différence entre les deux formes est qu'une boucle avec Tantque peut ne jamais être exécutée car la condition peut être fausse au départ alors que dans la boucle Repeter le groupe d'instructions est exécuté au moins une fois pour arriver à la condition.

La condition du Repeter est donc une condition de fin (la répétition est terminée quand la condition est vraie) alors que la condition du Tantque est une condition de répétition (la répétition est terminée quand la condition est évaluée à faux).

Repeter
    Instruction 1 ;
    Instruction 2 ;
    ….
    Instruction n
Jusqu'à Condition ;

Note :

! La boucle Repeter est particulièrement recommandée pour les saisies avec vérification.

Exemple

Ecrire un algorithme qui calcul la division d'un nombre réel par un entier selon la formule : z = x/y.

Solution avec Repeter :

```
Algorithme Division ;
Declaration
        x, z : reel ;
        y : entier ;
Debut
        Lire (x) ;
        Repeter
                Lire (y)
        Jusqu'à y ≠ 0 ;
        z ← x / y ;
        Ecrire (z) ;
Fin.
```

Lors de l'exécution de cet algorithme le passage à l'instruction de calcul de z ne se fait que lorsque la saisie de la valeur de y est différente du zéro.
Une autre solution en utilisant la boucle Tantque est la suivante :

```
Algorithme Division2 ;
Declaration
        x, z : reel ;
        y : entier ;
Debut
        Lire (x, y) ;
        Tantque y = 0 Faire
                Lire (y)
        FinTantque ;
        z ← x / y ;
        Ecrire (z) ;
Fin.
```

L'instruction de lecture est alors écrite deux fois : la première pour tester la condition et la deuxième pour répéter la lecture jusqu'à saisie d'une valeur non nulle.

Résumons maintenant avec un exemple pour lequel on donne 3 solutions pour les 3 types de boucles. Cet algorithme calcul et affiche 100 fois la somme de deux nombres entiers lus à chaque itération.

Version Tantque :

```
Algorithme boucle2 ;
Declaration
        x, y, co, som : entier ;
Debut
        co ← 0 ;
        Tantque (co ≠ 100) Faire
                Lire (x, y) ;
                co ← co + 1 ;
                Som ← x + y ;
                Ecrire ('la somme numéro', co, ' = ', som) ;
        Fintantque ;
Fin.
```

Version Répeter :

```
Algorithme boucle3 ;
Declaration
        x, y, co, som : entier ;
Debut
        co ← 0 ;
        Répeter
                Lire (x, y) ;
                co ← co + 1 ;
                Som ← x + y ;
                Ecrire ('la somme numéro', co, ' = ', som) ;
        Jusqu'à co = 100 ;
Fin.
```

Version Pour :

```
Algorithme boucle3 ;
Declaration
        x, y, co, som : entier ;
Debut
        Pour co ← 1 a 100 Faire
                Lire (x, y) ;
                Som ← x + y ;
                Ecrire ('la somme numéro', co, ' = ', som) ;
        FinPour ;
Fin.
```