Université Sétif 1
Faculté des Sciences
Département d'informatique
Filière : Licence Académique
Module : Algorithmique et structure de données
Année universitaire : 2015-2016

Chapitre 1 : Rappel

I - NOTION D'ALGORITHME

1°) Exemple :
On veut calculer la moyenne des notes d'un élève dans une matière donnée.
On suppose que le nombre de notes est égal à 3.

```
Var
        Nom, Matière : chaîne
        Moyenne, Note1, Note2, Note3 : réel
Début
        1 : Saisir Nom, Matière, Note1, Note2, Note3
        2 : Calculer la moyenne : Moyenne ←——(Note1+Note2+Note3)/3
        3 : Afficher Nom, Matière, Moyenne
Fin
```

Cette suite d'opérations qui permet de passer des données de base aux résultats correspond à un algorithme.

2°) Définition de la notion d'algorithme.
C'est une suite finie d'opérations élémentaires constituant un schéma de calcul ou de résolution d'un problème.
Il sert à décrire sous une forme quelconque (schéma ou langage naturel) un ensemble de règles opératoires propres à un traitement de données.

Tout algorithme est caractérisé par :

- Un ensemble d'actions ou d'opérations à exécuter.
- Un ordre d'exécution de ces différentes opérations déterminé par la logique d'enchaînement et conditionné par les structures mises en œuvre.
- Un début et une fin.

3°) Représentation d'un algorithme : Programmer.
Pour un ordinateur, l'algorithme est décrit par un programme informatique. C'est à dire une suite d'instructions exprimées dans un langage de programmation.

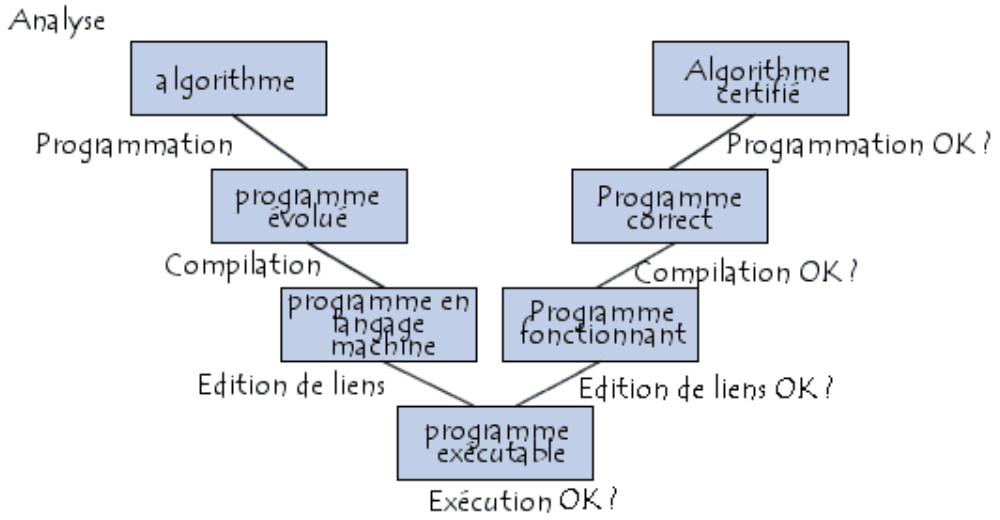
Ce langage n'est pas très adapté à la communication entre gestionnaires et informaticiens.
C'est pourquoi on utilise au préalable le langage algorithmique (proche du langage naturel), afin de décrire pas à pas une solution au problème posé.

4°) mise au point
La mise au point d'un programme informatique se fait en plusieurs étapes :
Il s'agit de fournir la solution à un problème, la première étape consiste donc à analyser le problème, c'est-à-dire en cerner les limites et le mettre en forme dans un langage descriptif, on parle généralement d'**analyse** pour décrire le processus par lequel le problème est formalisé. Le langage de description utilisé pour écrire le

résultat de l'analyse est appelé **algorithme**. L'étape suivante consiste à traduire l'algorithme dans un **langage de programmation** spécifique, il s'agit de la phase de **programmation**.

Le *langage de programmation* est l'intermédiaire entre l'humain et la machine, il permet d'écrire dans un langage proche de la machine mais intelligible par l'humain les opérations que l'ordinateur doit effectuer. Ainsi, étant donné que le langage de programmation est destiné à l'ordinateur, il doit donc respecter une syntaxe stricte. Un algorithme peut toutefois aboutir à plusieurs programmes.

Le programme est ensuite transformé en **langage machine** lors d'une étape appelée **compilation**. La *compilation* est une phase réalisée par l'ordinateur lui-même grâce à un autre programme appelé **compilateur**. La phase suivante s'appelle l'**édition de liens**, elle consiste à lier le programme avec tous les éléments externes (généralement des librairies auxquelles il fait référence).



5°) 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

II - LES DONNEES ELEMENTAIRES.

Tout algorithme utilise des objets ou données élémentaires comme par exemple des littéraux, des constantes ou des variables.

1°) Littéral.

C'est une valeur de type numérique ou alphanumérique.
Exemple : 11 ; 20.6 ; "Bonjour"

2°) Constante et variable.
Une constante est un objet qui ne peut pas être modifié par l'algorithme.

Une variable est un objet appelé à subir des transformations au sein de l'algorithme.
Constante et variable se caractérisent par :

- Un identificateur : nom de l'objet ou de la donnée qui ne doit pas contenir d 'espace.
- Une valeur : contenu de l'objet.
- Un type : domaine ou l'objet puise sa valeur.

Exemples :

```
Const
        Pi=3.1416
Var
        Diamètre, circonférence : réel
```

3°) Types d'objets existants.

Types d'objets	Ensemble de valeurs possibles	opérations
BOOLEEN	Vrai,Faux	Comparaison(=,<,>,..),NON, ET,OU.
CARACTERE	"A"	Comparaison, conversion
ENTIER	45 123	+,-,/,*,DIV, Comparaison.
REEL	12,345	Comparaison,arithmétique.
CHAÎNE	"Bonjour monsieur"	Comparaison, longueur, extraction, concaténation.

III - LES STRUCTURES DE BASE

1°) L'instruction d'affectation.
Elle permet d'affecter ou d'initialiser une variable à partir du contenu d'une autre variable, d'une constante, d'un littéral, d'une expression arithmétique ou logique.
Le symbole utilisé est : ← ou :=
Exemples d'affectation :

```
Total :=826          ou Total ←  226
PRIX ← Total
Somme ← Total+8
Somme ←Somme+50
```

2°) Instructions d'entrée-sortie.
L'instruction d'entrée autorise la saisie de l'information à partir du clavier.
Les instructions de sortie autorisent :

- L'affichage des informations à l'écran.
- L'impression des informations sur papier.

Instruction d'entrée :

Saisir Nom_variable

Consiste à affecter une valeur saisie à partir du clavier à une variable.

Instruction de sortie :

Afficher Nom_variable

Permet d'écrire la valeur d'une variable sur un support externe.

Exemples :

```
Afficher " Taper deux nombres : "
Saisir A,B
Somme ← A+B
Afficher " la somme est de : ",Somme
```

Remarque :

Les messages à afficher sont définies entre « ».
On peut saisir plusieurs variables en une seule fois, séparées par une virgule.
A la rencontre d'une instruction saisir, le programme est interrompu.
L'utilisateur doit alors entrer la donnée.
La saisie est terminée par l'appui sur la touche entrée.
Le déroulement du programme se poursuit.

IV - LES STRUCTURES CONDITIONNELLES OU ALTERNATIVES.

La structure conditionnelle permet un aiguillage des traitements.
Selon la valeur d'une expression booléenne, on pourra exécuter une suite d'actions I ou une suite d'actions II.

1°) La structure alternative.

Syntaxe

SI <expression logique>
ALORS action1
SINON action2
Fin Si

Le résultat de l'expression logique (ou condition) est un booléen.
Quand l'expression logique est vraie alors la suite d'actions située après le mot **ALORS** (action1) est exécutée.
Si le résultat est faux, on exécute la suite d'actions située après le mot **SINON** (action2).

Exemple : une remise de 5 % est accordée si la somme des achats dépasse 100 Euros.

```
SI  Montant>100
        ALORS  Rem :=Montant*0,05
        SINON  Rem :=0
Fin Si
```

Remarque :
- L'expression logique peut effectuer une comparaison entre plusieurs grandeurs. Elle utilise alors les opérateurs de comparaison :>,>=,<,<=,<>.

- L'expression logique peut être complexe et peut faire intervenir les opérateurs logiques : ET, OU.
Exemple : (a>b) ET (a>c).

Parfois la structure alternative peut être simple, et ne comporte pas la clause SINON.

SI <expression logique>
ALORS action
Fin Si

Si l'expression logique est vraie alors on exécute la suite d'actions sinon on poursuit la suite du traitement.

- On peut emboîter plusieurs structures alternatives dans certains cas de figure.

```
SI exp_log1
    ALORS  SI  exp_log2
                ALORS action1
                SINON action2
           Fin Si
    SINON  action 3
Fin Si
```

Le mot clé FSI permet de lever toute ambiguïté.

Exemple : Afficher le plus grand de deux nombres.

```
Algo PlusGrand
Var
    a,b :entier
Début
    AFFICHER "Entrez deux nombres : "
    SAISIR a,b
    Si a>b
        Alors AFFICHER " Le plus grand des deux est : ", a
        Sinon Si a=b
                Alors AFFICHER " Les nombres ",a," et ",b," sont égaux"
                Sinon AFFICHER " Le plus grand des deux est : ",b
              Fin Si
    Fin Si
Fin
```

2°) Structure à choix multiples.

Un nombre important de choix est à envisager selon les valeurs prises par une variable ou expression.
Cette structure permet une présentation plus claire d'un ensemble d'alternatives imbriquées.

Selon variable ou exp
Cas Valeur1 : action1
Cas Valeur2 : action2
Cas Valeur3 :action3
Cas Sinon action par défaut
Fin Selon

Exemple :

```
Selon mois
    Cas 1 :AFFICHER(" JANVIER")
    Cas 2 :AFFICHER(" FEVRIER")
    Cas 3 :AFFICHER(" MARS")
    …
Fin selon
```

V - LES STRUCTURES ITERATIVES.

Un ensemble d'actions qui se répète toujours dans un ordre précis, un nombre déterminé de fois constitue un traitement itératif.
Un tel traitement est défini par une structure itérative qui définit la suite des opérations à effectuer ainsi que le nombre de répétitions.
Dans la plupart des cas, on ne connaît pas le nombre de répétitions, on précise alors la nature de l'événement qui doit mettre un terme à l'itération.

1°) Structure Tant que ….Faire..

Syntaxe :

Tant que <expression logique> **faire**
Actions
Fin Tant Que

L'ensemble d'actions doit être exécuté tant que l'expression logique est vraie. Lorsque l'expression logique est fausse, le processus itératif s'arrête.

Phases d'exécution :

-1. Evaluation de l'expression logique.
-2. Résultat vrai : Exécuter les actions.
Reprise de l'étape précédente 1.
-3. Résultat faux :Arrêt de l'itération et le programme poursuit son exécution après FTQ.

Remarque :

La condition d'arrêt doit être réalisable : sa valeur doit passer à faux après un nombre fini de tours de boucle. Cette condition est **composée d'une variable** dont la valeur change à chaque tour de boucle. Cette variable est appelée **variable de contrôle ou d'itération**. Elle doit être modifiée par une action dans la boucle
Exemple : Calculer la somme des N premiers nombres entiers.

```
Algo  Somme
Var
        S,I,N : Entier
Début
        Afficher "Entrer la valeur de N"
        Saisir  N
        S ←— 0
        I ←— 1
        Tant que I<=N faire
                S ←— S+I
                I ←— I+1
        Fin Tant Que

        Afficher  "La somme des ",N," premiers entiers est : ", S
Fin
```

La variable S est utilisée pour additionner les différentes valeurs. Elle est initialisée à 0.

La variable I est appelée variable d'itération. Elle passe en revue les nombres

Cette instruction permet de faire évoluer I à la valeur suivante.

2°) La structure Répéter…jusqu'à..

Syntaxe :

Répéter
…
Actions
…
Jusqu'à <expression logique>

-1. Le bloc d'actions est exécuté.
-2. L'expression logique est testée
-3. Dans le cas où elle est égale à faux, on recommence au point 1.
-4. Dans le cas où elle est égale à vrai. Le programme poursuit son exécution après l'instruction "jusqu'à".

Dans ce cas la suite d'actions est exécutée au moins une fois, car le test de l'expression logique est effectué après exécution de l'ensemble d'actions.

Exemple : Traduire l'algorithme précédent en utilisant la structure : Répéter…Jusqu'à. Expression logique(I>N).

Comparaison des deux structures :

-Dans la **structure répéter** .., le test est placé en fin d'itération, par conséquent les actions répétitives sont exécutées au moins une fois.

-Dans la **structure Tant que**.., le test est placé avant le corps de la boucle. Dans certains cas, il est possible de ne pas exécuter une seule fois les opérations répétitives.

3°) La structure Pour …Fin pour.

Cette structure permet de répéter un ensemble d'actions un nombre connu de fois.

Syntaxe :

Pour Nom_var **de** valeur-initiale **à** valeur-finale [pas de incrémentation]
Actions
Fin pour

Nom_var est la variable **d'énumération des répétitions**. Elle sera initialisée à la valeur de début, l'ensemble des actions sera exécuté, elle passera automatiquement à la valeur suivante jusqu'à la valeur finale.

Exemple :

```
S ← 0
Pour I de1 à 10
S ← S+I
Fin pour
```