ALGORITHMIQUE ET PROGRAMMATION INFORMATIQUE

Chapitre 1 : Généralités sur l'algorithmique et les langages de programmation

La recherche des algorithmes est une préoccupation ancienne, avant l'idée même d'ordinateur. Mais avec l'avènement de l'ordinateur, il fallait décrire de manière précise à l'ordinateur avec toutes les éventualités possibles, la suite d'actions à exécuter pour obtenir via les informations fournies les résultats espérer ou attendus.

Dans cette partie nous définirons certains concepts, aborderons les concepts de description de l'algorithme, sa qualité et sa conception.

I. Pourquoi faire de l'ALGO

La raison d'être d'un algorithme est de résoudre un problème tout en demandant par la suite à la «machine» d'effectuer un travail à notre place.

Le problème réside du faite que nous devons expliquer à la «machine» comment elle doit s'y prendre. Pour cela, Une attention particulière doit être portée à la compréhension du problème, sinon l'algorithme pourra être incorrect.

Pour que la machine puisse bien faire le travail demandé, il faut :

- pouvoir expliciter son raisonnement
- savoir formaliser son raisonnement
- concevoir (et écrire) des algorithmes.

II. Définitions

- **Algorithme** est un mot dérivé du nom du célèbre mathématicien arabe Muhammad Ibn Musa Abu Djefar Al Khwarizmi qui a vécu au 9ème siècle (780-850), et était membre d'une académie des sciences à Bagdad.
- Un **algorithme** est une séquence d'instructions qui décrit comment résoudre un problème particulier.

  C'est est une suite finie d'actions à entreprendre en respectant une chronologie imposée pour traiter des problèmes. Tout cela indépendamment du langage de programmation.

  Un algorithme prend des données en entrée, exprime un traitement particulier et fournit des données en sortie.
- **L'algorithmique** est une science qui cherche la manière la plus optimale pour calculer des solutions pratiques à un problème de calcul. Par extension, toute manipulation d'un ensemble de données pour le transformer ou en tirer des résultats est le sujet de l'algorithmique.

- Un **algorithme**, traduit dans un langage compréhensible par l'ordinateur (ou langage de programmation, ici le C++), donne un **programme**, qui peut ensuite être exécuté, pour effectuer le **traitement** souhaité.
- Un **programme** est donc une série d'instructions pouvant s'exécuter en séquence, ou en parallèle (parallélisme matériel) qui réalise (implémente) un algorithme

III. Exemple d'algorithmes:

- Recette de cuisine ;
- Notice de montage de meuble en kit ;
- Algorithme d'Euclide permettant de trouver le PGCD de deux nombres entiers a et b
  - diviser a par b : a = bq + r
  - si le reste r est nul alors le PGCD est b sinon continuer en (3)
  - prendre pour nouvelle valeur de a la valeur de b et pour valeur de b la valeur de r
  - recommencer le traitement depuis (1).
- Algorithme de résolution dans IR de l'équation $ax2 + bx + c = 0$.

IV. Règle d'un algorithme

Un algorithme doit suivre les règles suivantes :

Règle d'entrée : Un algorithme a zéro ou plusieurs entrées appelées données, prises dans des ensembles spécifiés d'objets.

Règle de sortie : Un algorithme a zéro ou plusieurs sorties appelées résultats, quantités qui sont en relation de façon spécifiée avec les entrées et qui appartiennent à des ensembles bien spécifiés.

Règle de précision : Chaque pas de l'algorithme doit être défini précisément sans ambiguïté.

Règle de finitude : Un algorithme doit toujours se terminer après un nombre fini d'étapes, quel que soit l'entrée.

Règle d'effectivité : Chaque opération doit être suffisamment basique pour pouvoir être effectuée en un temps fini par un homme utilisant les moyens humains.

V. Qualités d'un algorithme

Tout programme fourni à l'ordinateur n'est que la traduction dans un langage de programmation d'un algorithme mis au point pour résoudre un problème donné. Pour obtenir un bon programme, il faut donc partir d'un bon algorithme. Il doit, entre autres, posséder les qualités suivantes :

- **Etre lisible** : programmer, c'est communiquer avec l'ordinateur mais aussi avec soi-même et avec les autres. Un algorithme doit donc être compréhensible par tous ceux qui le lisent, même par des non informaticiens. On utilisera des désignations évocatrices pour les objets et sous-programmes, des commentaires, des indentations,…
- **Etre d'utilisation aisée** : même par ceux qui ne l'ont pas écrit et ce grâce aux messages affichés qui indiquent les données à fournir et leur forme ainsi que les actions attendues de l'utilisateur.
- **Etre de haut niveau** : l'algorithme doit pouvoir être traduit dans n'importe quel langage de programmation, il ne doit donc pas faire appel à des notions techniques relatives à un langage de programmation ou à un système donné.
- **Etre concis** : un algorithme ne doit pas dépasser une page ; si c'est le cas, il faut décomposer le problème qu'il résout en plusieurs sous-problèmes en utilisant la méthode d'analyse descendante**.**
- **Etre structuré** : un algorithme doit être composé de différentes parties facilement identifiables.
- **Etre le plus général possible** : pour répondre au plus grand nombre de cas possibles.
- **Etre efficace** : un algorithme doit être conçu de manière à limiter le temps d'exécution des opérations à effectuer et la place occupée en mémoire par les informations manipulées.

VI. Modes de représentation d'un algorithme

Il existe plusieurs manières de décrire les algorithmes.

a) Langage naturel

Français, anglais, ...

Exemple: Calculer l'intérêt et la valeur acquise par une somme placée pendant un an à intérêt simple.

1) Prendre connaissance de la somme initiale et du taux d'intérêt
2) multiplier la somme initiale par le taux
3) diviser le produit obtenu par 100, le quotient obtenu est l'intérêt
4) additionner ce montant et la somme initiale, cette somme est la valeur acquise
5) afficher les valeurs de l'intérêt et de la valeur acquise

Il est évident qu'une telle formalisation risque de produire un texte long difficile à comprendre et de ne pas mettre clairement en évidence les différentes étapes du traitement.

b) Langage de description algorithmique (LDA)

Un LDA est un ensemble de règles syntaxiques et de mots clé permettant de décrire, de manière complète, claire l'ensemble des actions à exécuter sur des données pour obtenir des résultats.

Les actions sont généralement décrites par un symbole ou un verbe à l'infinitif choisi pour éviter la confusion.

Exemple:

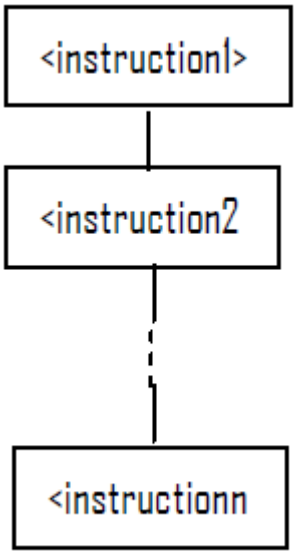
```
ecrire ("Entrer un premier nombre entier):");
lire (nombre1);
ecrire("Entrer un deuxieme nombre");
lire (nombre2);
somme ← nombre1 + nombre2;
moyenne ← somme / 2;
ecrire("La somme de vos deux nombres est :", somme);
ecrire("Leur moyenne est : ", moyenne);
```

L'ordre dans lequel les opérations sont écrites indique l'ordre dans lequel elles seront exécutées.

c) Algorigramme

L'algorigramme est une représentation graphique de l'algorithme. Il utilise des symboles normalisés par AFNOR (Association Française de la NORmalisation).

Syntaxe :

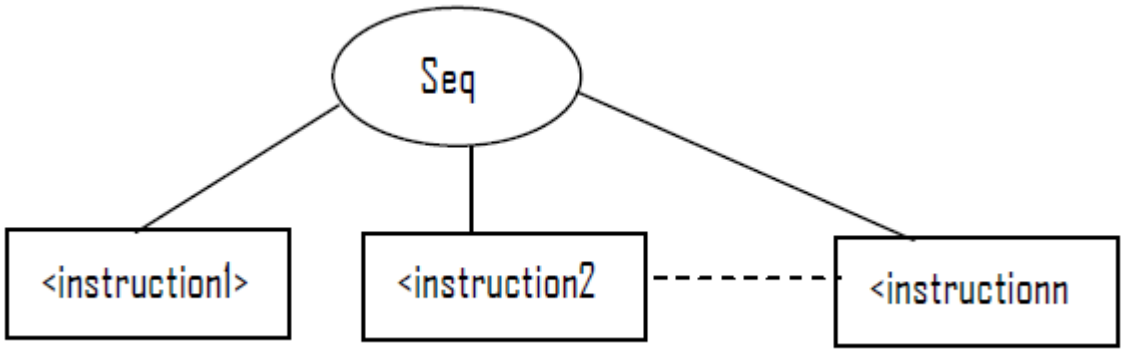


d) Arbre programmatique

Les arbres programmatiques permettent également de représenter graphiquement les algorithmes.

Un arbre programmatique est un diagramme donnant une vision spatio-temporelle de la structure d'un Algorithme.

Syntaxe :



VII. Procédure de conception d'algorithme

1) Comprendre la nature du problème posé et préciser les données fournies et les résultats que l'on désire obtenir ;
2) Déterminer les données intermédiaires ou de travail ;
3) Déterminer la logique de traitement c'est-à-dire le processus de transformation des données en résultats ;
4) Rédiger l'algorithme dans un des modes de représentation.

Chapitre 2 : Les éléments de Base d'un langage algorithmique

I. Notion d'objet

Un algorithme manipule des informations (données) pour produire d'autres informations (résultats).

Une information contenue dans un algorithme est appelée **objet**. L'ensemble des objets manipulés par un algorithme est appelé **environnement** de l'algorithme.

Généralement à chaque objet correspond un emplacement en mémoire. Un objet est caractérisé son adresse, son identificateur, sa valeur, son type.

1. adresse

C'est l'adresse de l'emplacement correspondant à l'objet.

2. identificateur d'objet

L'identificateur est une suite de caractères alphanumériques non accentués, sans espace, commençant par une lettre. C'est nom donné à l'objet par le programmeur, permettant de sa manipulation. Le nom doit être unique et en rapport avec le contenu de l'objet.

3. Valeur d'un objet

C'est le contenu de l'objet (c'est à dire une valeur de l'information) à un moment donné du déroulement de l'algorithme. Un objet ne peut prendre qu'une valeur à un instant donné.

Au cours de l'exécution de l'algorithme, certains objets peuvent changer de valeur: ce sont des **variables** ; d'autres objets au contraire ne seront pas modifiables. Ce sont des **constantes**.

La manipulation d'un objet concerne sa valeur.

4. Type d'un objet

Le type d'un objet est la nature, la catégorie de l'objet. Le type d'un objet détermine:

- L'ensemble des valeurs que peut prendre l'objet.
- La taille de l'objet c'est à dire de l'emplacement correspondant.
- Les opérations applicables à l'objet

On distingue généralement deux catégories de types :

- Les types élémentaires (ou simples ou de base): un type est dit élémentaire si les actions qui manipulent un objet de ce type ne peuvent l'accéder que dans sa totalité.

- Les types structurés (ou complexes) qui sont définis à partir d'autres types et dont les objets peuvent être accédés en partie.

II. Types de données de base

1. Type entier

Le type entier représente l'ensemble des entiers relatifs Z.

Une constante littérale entière c'est à dire un élément du type entier est notée de manière classique: suite chiffres en base 10, précédée ou non des signes + ou - .

Les opérations applicables:

Opérations arithmétiques: -, +, -, *, div, mod, ∧

Opérations de comparaison**: <, >, =, ≠, ≥, ≤**

2. Type réel

Le type réel représente l'ensemble IR des réels. Les constantes littérales réelles s'écrivent:

- En notation décimale avec le point décimal : 5, 3.5, -10.67,…
- En notation exponentielle : -5.698e -12, 25e7,…

Les opérations applicables:

- Opérations arithmétiques: -, +, -, *, /, ∧
- Opérations de comparaison**: <, >, =, ≠, ≥, ≤**

3. Type booléen

Le type booléen représente l'ensemble composé des deux constantes **.vrai.** et **.faux.**.

Les opérateurs applicables:

Opérations logiques**:** non, et, ou.

Opérations de comparaison **: <, >, =, ≠, ≥, ≤**

1. Type caractère

Le type caractère est l'ensemble formé des lettres (minuscules, majuscules), des chiffres et des symboles spéciaux (ponctuation, espace, ...).

Les constants caractères se notent entre simple quote (').

Chaque caractère est lié à une valeur entière qui le représente en mémoire.

Les opérateurs sur les caractères sont les opérateurs de comparaison

III. Les Expressions

1. Définition

Une expression est une suite bien formée d'opérateurs portant sur des objets (variables ou constantes), des constantes littérales, des appels de fonctions, et qui a une valeur.

Une expression représente en fait un calcul que l'ordinateur doit mener pour obtenir une valeur.

La valeur d'une expression est généralement utilisée: affichée, affectée, utilisée pour appeler un sous-programme. On peut utiliser des parenthèses dans une expression pour former l'ordre d'évaluation dans l'expression.

2. Type d'une expression

Le type d'une expression est le type de sa valeur. Ainsi on distingue des expressions:

- entières (type entier) ;
- réelles (type réel) ;
- numériques (type réel ou entier) ;
- booléennes ou logiques (type booléen) ;

IV. Structure de base

En matière de programmation, il n'y a pas grande chose d'obligatoire mais beaucoup de choses recommandées. En particulier, un programme a à peu près toujours la même organisation générale

```
1: Nom du programme
2: Déclaration de variables
3: Déclaration de fonctions
4: Début
5:     ...
6:     Liste des instructions
7:     ...
8: Fin
```

V. Instruction élémentaires

Une instruction est l'indication d'une action dont est capable l'ordinateur; il s'agit d'une instruction d'action c'est à dire d'ordre. Donc l'algorithme est en fait un ensemble d'instructions d'action sur des objets.

Chaque instruction donnera lieu à une action de la part de l'ordinateur.

1. Déclaration de variables

Définition de variable : Une variable est un espace mémoire nommée, de taille fixée prenant au cours du déroulement de l'algorithme un nombre indéfini de valeurs différentes. Ce changement de valeur se fait par l'opération d'**affectation** (notée dans notre langage algorithmique). La variable diffère de la notion de **constante** qui, comme son nom l'indique, ne prend qu'une unique valeur au cours de l'exécution de l'algorithme.

Utilité de la déclaration de variable : La partie *déclaration de variable* permet de spécifier quelle seront les variables utilisées au cours de l'algorithme ainsi que le type de valeur qu'elles doivent respectivement prendre. Il est bien évident que l'on ne peut mémoriser une chaine de caractères dans une variable de type ``Entier''. Le type des variables est utile à l'algorithmicien pour lui permettre de structurer ses idées. Il est très utile au programmeur car il permet de détecter des erreurs potentielles. Il est indispensable au compilateur car les différents types ne sont pas tous représentés de la même façon. La représentation d'un même type peut même varier d'un processeur à l'autre.

a) Syntaxe en LDA

i. Variable simple

Syntaxe

Variable <nom de donnée>: type

Exemple

- val, unNombre: entiers
- nom, prénom : chaînes de caractères

Fonctionnement

Instruction permettant de réserver de l'espace mémoire pour stocker des données de type <entier> et <chaine de caractère> dont les noms sont respectivement <val > et <unNombre>

ii. Variable constante

Syntaxe

Constante <nom de donnée>: type ← valeur ou Expression

Exemple

- MAX : entier ← 10

- DEUXFOISMAX : entier ← MAX x 2

Fonctionnement

Instruction permettant de réserver de l'espace mémoire pour stocker une constante dont la valeur ne varie plus. Ici les noms sont Max et DEUXFOISMAX avec les valeurs respectives 10 et 20.

Remarque:

Tout objet ou variable avant d'être manipulé doit être déclaré.

2. Affectation

L'affectation consiste à placer une valeur dans une variable. Le contenu précédent de la variable est définitivement perdu.

a) Syntaxe en LDA

<nom_var> ← <expression>;

```
1: { Exemple d'affectation }
2: Entier A
3: Début
4:    A← 1
5: Fin
6: {Cette séquence d'instructions donne la valeur entière "1" à la variable
   A.}
7: {Remarque : la valeur donnée est toujours du même type que la va-
   riable.}
```

b) Fonctionnement

L'instruction d'affectation s'exécute en deux étapes:

- Evaluation de l'expression <expression>
- Assignation du résultat dans la variable <nom_var>

L'expression et la variable doivent être de types compatibles.

3. LECTURE ÉCRITURE DE DONNÉES

a) Ecriture de données

L'écriture de données consiste à communiquer sur l'écran des résultats sous une forme directement compréhensible par l'utilisateur. Elle consiste en l'affichage d'un texte ou de la valeur d'une variable.

i) Syntaxe en LDA

Ecrire (<expression>);

ii) Fonctionnement

L'instruction va évaluer l'expression <expression> puis afficher le résultat à l'écran.

Exemple:

```
1. {Exemple d'utilisation de Ecrire}
2. A : Entier ←5
3. DEBUT
4.     Ecrire (A+10)
5.     Ecrire (''La Valeur de A est :'', A)
6.     Ecrire (''La Valeur de A+1 est :'', A+1)
7. FIN
```

Cette séquence d'instructions affiche les phrases suivantes à l'écran:

```
15
La Valeur de A est 5
La Valeur de A+1 est 6.
```

Remarque:

On peut généraliser la syntaxe afin de permettre l'écriture de plusieurs expressions:

Ecrire(<exp1>, ..., <exp n>);

b) Lecture de données

La lecture consiste à lire (à récupérer) une information saisie depuis le clavier de l'ordinateur, à la convertir en binaire et à la ranger en mémoire centrale dans une variable.

L'instruction de lecture de données permet donc de communiquer des données à l'algorithme. La saisie se fait par l'intermédiaire de la commande ***Lire***.

i) Syntaxe en LDA

Lire(<nom_var>);

ii) Fonctionnement

L'instruction de lecture suspend le déroulement de l'algorithme jusqu'à ce que l'utilisateur ait fini de taper au clavier une valeur et assigne la valeur entrée à la variable <nom_var>. Le type de la variable détermine ce que l'on entre au clavier (entier, réel, ...).

Exemple:

```
1. {Exemple d'utilisation de Lire}
2. A : Entier
```

```
3. DEBUT
4.     Lire (A)
5.     Ecrire (''La Valeur de 5*A est :'', 5 x A)
6. FIN
```

Si l'utilisateur saisit la valeur 10, nous aurons alors à l'écran:

```
La Valeur de 5*A est : 50
```

4. Commentaire

Le commentaire est l'indication d'un texte à ignorer lors de la traduction de l'algorithme.

Syntaxe en LDA

(* <texte a ignorer>*)

TD1

Exercice 1 Que fait la liste d'instructions suivantes ?

```
1: A ← 2
2: A ← A + 2
3: B ← A × 2 + A
4: C ← 4
5: C ← B − C
6: C ← C + A − B
7: A ← B − C × A
8: A ← (B − A) × C
9: B ← (A + C) × B
```

Exercice 2 Que fait la liste d'instructions suivantes ?

```
1: X ← −5
2: X ← X²
3: Y ← −X − 3
4: Z ← (−X − Y)²
5: X ← −(X + Y)² + Z
6: Y ← Z^X × Y
7: Y ← −(Z + Y)
8: X ← X + Y − Z
9: Y ← X + Z
10: X ← (Y − Z)²
11: Y ← X − Y
12: X ← (Y + Z)/(X/10)
13: Y ← ((X × Z)/Y) × 9
```

Comprendre les principes de l'affectation

Exercice 3 Comment inverser le contenu de deux variables ?

```
1: {Inversion de deux variables}
2: Entier A, B, X
3: Début
4:    Lire A
5:    Lire B
6:    ...
7:    ...
8:    ...
9:    Écrire ('La valeur de A est ', A)
10:   Écrire ('La valeur de B est ', B)
11: Fin
```

Exercice 4 Comment faire sans utiliser une variable supplémentaire?

```
1: {Inversion de deux variables}
2: Entier A, B, X
3: Début
4:     Lire A
5:     Lire B
6:     ...
7:     ...
8:     ...
9:     Écrire ('La valeur de A est ', A)
10:    Écrire ('La valeur de B est ', B)
11: Fin
```

Exercice 5 étant donnée X, comment calculer le plus rapidement possible X^{16} ?

```
1: {Exponentiation}
2: Entier X
3: Début
4:     Lire X
5:     ...
6:     ...
7:     ...
8:     ...
9:     Écrire ('La valeur de X^16 est ', X)
10: Fin
```

Résolution de problèmes simples

Exercice 6 : Ecrire un algorithme saisissant le prix "TTC" d'une marchandise et affichant le prix "Hors Taxe" sachant que cet article a une T.V.A. de 18,6%.

Exercice 7 : Ecrire un algorithme saisissant 2 variables entières qui calcule et affiche leur moyenne.

Exercice 8 : Ecrire un algorithme saisissant un temps en seconde que l'on transcrira en jours, heure, minutes, secondes.

Exercice 9 : En se basant sur l'exercice précédent, Ecrire un algorithme permettant de faire la différence entre deux horaires saisis en heure, minutes, secondes.

Exercice 10 : Ecrire un algorithme qui demande à l'utilisateur d'entrer la largeur et la longueur et afficher la surface d'un rectangle