CREATION D'UN COMPILATEUR DU LANGAGE ALGORITHMIQUE

Module : Compilation

Réalisé par :

LABIAD Salah Eddine

ZIRY Asmaa

OUASSINE Younes

HADDANI Abdelhakim

Introduction

Pourquoi apprendre l'algorithmique pour apprendre à programmer ? En quoi a-t-on besoin d'un langage spécial, distinct des langages de programmation compréhensibles par les ordinateurs ?

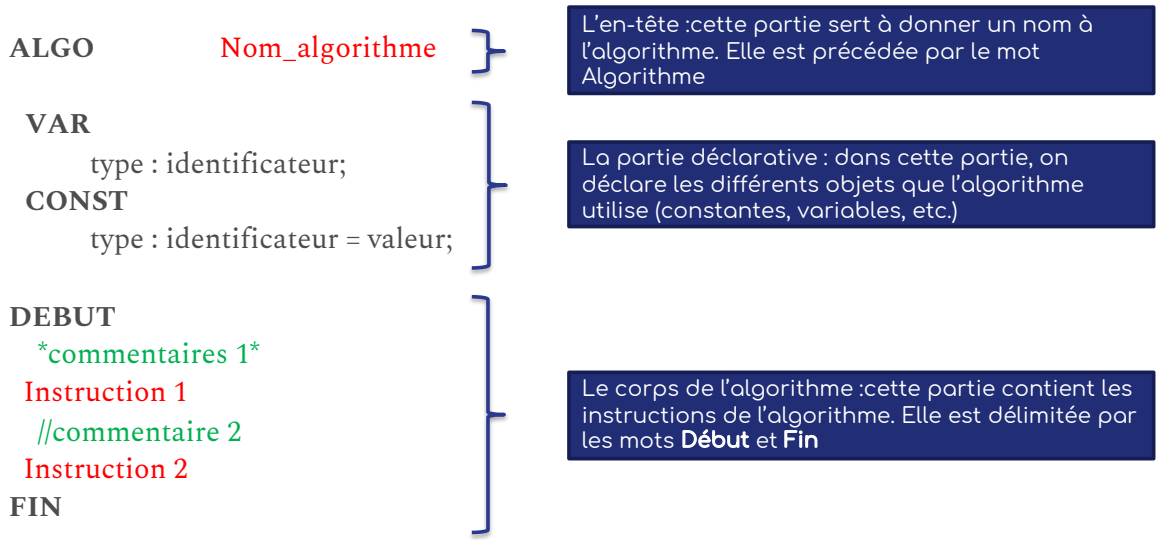
Parce que l'algorithmique exprime les instructions résolvant un problème donné indépendamment des particularités de tel ou tel langage. Pour prendre une image, si un programme était une dissertation, l'algorithmique serait le plan, une fois mis de côté la rédaction et l'orthographe. Or, vous savez qu'il vaut mieux faire d'abord le plan et rédiger ensuite que l'inverse...

Apprendre l'algorithmique, c'est apprendre à manier la structure logique d'un programme informatique. Cette dimension est présente quelle que soit le langage de programmation ; mais lorsqu'on programme dans un langage (en C, en Visual Basic, etc.) on doit en plus se colleter les problèmes de syntaxe, ou de types d'instructions, propres à ce langage. Apprendre l'algorithmique de manière séparée, c'est donc sérier les difficultés pour mieux les vaincre.

A cela, il faut ajouter que des générations de programmeurs, souvent autodidactes, ayant directement appris à programmer dans tel ou tel langage, ne font pas mentalement clairement la différence entre ce qui relève de la structure logique générale de toute programmation (les règles fondamentales de l'algorithmique) et ce qui relève du langage particulier qu'ils ont appris. Ces programmeurs, non seulement ont beaucoup plus de mal à passer ensuite à un

langage différent, mais encore écrivent bien souvent des programmes qui même s'ils sont justes, restent laborieux. Car on n'ignore pas impunément les règles fondamentales de l'algorithmique... Alors, autant l'apprendre en tant que telle!

1. Représentation générale



2. Généralités

2.1 Les variables

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs. Il peut s'agir de données sur le disque dur, fournies par l'utilisateur (frappées au clavier), ou que sais-je encore. Il peut aussi s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ces données peuvent être de plusieurs types (on en reparlera): elles peut être des nombres, du texte, etc. Toujours est-il que dès que l'on a besoin de stocker une informez-moi au cours d'un programme on utilise une variable.

La première chose à faire avant de pouvoir utiliser une variable est de créer la variable. Ceci se fait tout au début de l'algorithme, avant même les instructions proprement dites. C'est ce qu'on appelle la déclaration des variables.

Le nom de la variable (étiquette de la boîte) obéit à des impératifs changeant selon les langues. Toutefois, une règle absolue est qu'un nom de variable peut être porteur des lettres et des chiffres, mais qu'il exclut la plupart des signes de ponctuation, en particulier les

espaces. Un nom de variable bonne commence également impérativement par une lettre. Quant au nombre maximal de signes pour un nom de variable, il est limite à 20 caractères.

Lorsqu'on déclare une variable, il ne suffit pas de créer une boîte (réserver un emplacement mémoire); encore doit-on préciser ce que l'on voudra mettre dedans, car cela dépend de la taille de la boîte (de l'emplacement mémoire) et le type de codage utilisé.

Exemples :

G20, Nom, TTC_2012 **valides**

TTC 2012, TTC-2012, TTC+2012, read, end **non valides**.

- **Le type ENTIER (entiers)**

Le choix du nombre d'octets retenus pour écrire en mémoire un entier dépend du couple ordinateur plus logiciel qui l'équipe. Souvent, un octet est utilisé pour le codage d'un nombre entier. Un bit va servir pour coder le signe du nombre.

Exemples : Type entier tels que : 0, 405, -10,...

- **Le type REEL (réels)**

Un nombre réel ou flottant est caractérisé par : son signe, son exposant ou caractéristique et sa mantisse.

Exemples : Type réel tels que : 0.5, -3.67, 0.0001,..

- **Le type CAR (caractères)**

C'est le type qui permet de gérer les chaînes de caractères. Les chaînes de caractères sont écrites indifféremment entre guillemets ou entre apostrophes.

Exemples : Type caractère tels que : 'a', 'B', '9', ' ',

2.2 Les Constantes (CONST)

Une CONSTANTE, comme une variable, peut représenter un chiffre, un nombre, un caractère, une chaîne de caractères, un booléen. Toutefois, contrairement à une variable dont la valeur peut être modifiée au cours de l'exécution de l'algorithme, la valeur d'une constante ne varie pas.

2.3 Opérandes et operateurs

Il existe plusieurs types d'opérateurs :

- Les opérateurs arithmétiques qui permettent d'effectuer des opérations arithmétiques entre opérandes numériques :

  -Opérateurs élémentaires : « + », « - », « x », « ÷ »

  -Changement de signe : « - »

- Les opérateurs de comparaison (« = », « <> », « > », « < », « ≥ » et « ≤ ») qui permettent de comparer deux opérandes et produisent une valeur booléenne, en s'appuyant sur des relations d'ordre :
- Les opérateurs logiques qui combinent des opérandes booléennes pour former des expressions logiques plus complexes

  -Opérateur unaire : «non » (négation)

  -Opérateurs binaires : «et » (conjonction), «ou » (disjonction),

- L'opérateur d'affectation, représenté par le symbole « = », qui confère une valeur à une variable ou à une constante.

Type	Opération possibles	Symbole ou mot correspondant
Entier	Addition Soustraction Multiplication Division Comparaisons	+ - * / <, =, >, <=, >=, <>
Réel	Addition Soustraction Multiplication Division Comparaisons	+ - * / <, =, >, <=, >=, <>
Caractère	Comparaisons	<, =, >, <=, >=, <>
Chaîne	Comparaisons	<, =, >, <=, >=, <>
Booléen	Logiques	ET, OU, NON

- Opérateurs sur les entiers et les réels : addition, soustraction, multiplication, division, puissance, comparaisons

- Opérateurs sur les booléens : comparaisons, négation

- Opérateurs sur les caractères : comparaisons

- Opérateurs sur les chaînes de caractères : comparaisons

2.4 L'instruction d'affectation

L'affectation est l'opération qui consiste à stocker une valeur dans une variable.

Cette opération se fait à l'aide de la syntaxe suivante :

```
$Nom_variable  =  Valeur
```

Exemple : $x = 5 Signifie mettre la valeur 5 dans la case identifiée par x.

A l'exécution de cette instruction, la valeur 5 est rangée en x (nom de la variable).

```
$A = 15

$B = 2

$A = $B * 3

$C = 12 - 5

$D = $A + 9

$E = "Bonjour"
```

2.5 Les notions de lecture et d'écriture

L'ENTREE ou la lecture de données correspond à l'opération qui permet de saisir des valeurs pour qu'elles soient utilisées par le programme. Cette instruction est notée « LIRE ». La SORTIE ou l'écriture des données permet l'affichage des valeurs des variables après traitement. Cette instruction est notée « ECRIRE ».

```
ALGO     Nom_algorithme
VAR
     ENTIER : x;
DEBUT
     *afficher un message*
ECRIRE("Donner un entier: ");
*Saisir une variable*
LIRE( x );
ECRIRE("Vous avez tapé : ", x);
FIN
```

3. Structures de contrôle conditionnel

Ces structures sont utilisées pour décider de l'exécution d'un bloc d'instruction : est- ce-que ce bloc est exécuté ou non. Ou bien pour choisir entre l'exécution de deux blocs différents.

3.1 Test alternatif simple

Un test simple contient un seul bloc d'instructions. Selon une condition (expression logique), on décide est-ce-que le bloc d'instructions est exécuté ou non. Si la condition Est vraie, on exécute le bloc, sinon on ne l'exécute pas. La syntaxe d'un test alternatif simple est comme suit :

```
SI (expression logique) ALORS
Instructions ! Exécuté si la condition est vraie
FINSI
```

3.2 Test alternatif double

Un test double contient deux blocs d'instructions : on est amené à décider entre le premier bloc ou le second. Cette décision est réalisée selon une condition (expression logique ou booléenne) qui peut être vraie ou fausse. Si la condition est vraie on exécute le premier bloc, sinon on exécute le second. La syntaxe d'un test alternatif double est comme suit :

```
SI (expression logique) ALORS
Instructions 1
SINON
Instructions 2
FINSI
```

```
SI (expression logique) ALORS
 Instructions 1
SINON SI (expression logique) ALORS
Instructions n-1
SINON
 Instructions n
FINSI
```

3.3 choix multiples SELON-QUE

Lorsque l'ordinateur rencontre cette instruction, il vérifie la valeur de la variable de sélection (sélecteur) et il la compare aux différentes valeurs.

```
SELON sélecteur     FAIRE
          Valeur 1 : action 1
          Valeur 2 : action 2
          Valeur 3 : action 3
               ...
          Valeur n : action n

FINSELON
```

4. Les structures répétitives

Une boucle permet de parcourir une partie d'un programme un certain nombre de fois. Une itération est la répétition d'un même traitement plusieurs fois. Un indice de boucle varie alors de la valeur minimum (initiale) jusqu'à la valeur maximum (finale).

4.1 La boucle Pour

Cette structure est une BOUCLE ITERATIVE ; elle consiste à répéter un certain traitement un nombre de fois fixé à l'avance

Cette structure utilise une variable (indice) de contrôle d'itérations caractérisée par :

-sa valeur initiale,

-sa valeur finale,

-son pas de variation.

La syntaxe de la boucle pour est comme suit :

```
POUR I ALLANT 1 JUSQUA N PAS 1 FAIRE
      Instructions
FINPOUR
```

I : variable

1 : valeur initiale

N : valeur finale

Le pas de variation égale à 1

4.2 La boucle TANT QUE

Une action ou un groupe d'actions est exécuté répétitivement tout le temps où une condition est vraie.

La syntaxe de la boucle tant que est comme suit :

```
TANTQUE (Condition) FAIRE
 Instructions
FINTANTQUE
```

Remarque : la vérification de la condition s'effectue avant les actions. Celles-ci peuvent donc ne jamais être exécutées.

4.3 La boucle REPETER ... JUSQUA ...

Une action ou un groupe d'actions est exécuté répétitivement jusqu'à ce qu'une condition soit vérifiée.

La syntaxe de la boucle répéter est comme suit :

```
REPETER
     Instructions
JUSQUA(Condition)
```

Remarque : la vérification de la condition s'effectue après les actions. Celles-ci sont donc exécutées au moins une fois.

5. Les tableaux

Un tableau est un ensemble d'éléments de même type, repérés au moyen d'indices entiers. Les éléments d'un tableau sont rangés selon un ou plusieurs axes appelés dimensions du tableau. Dans les tableaux à une dimension (qui permettent de représenter par exemple des vecteurs au sens mathématique du terme), chaque élément est repéré par un seul .

Un tableau à une dimension est parfois appelé vecteur. Il peut être représenté sous la forme suivante :

L(1)	L(2)	L(3)	L(4)	.....	L(n)

-Dimension du tableau : 1

-Taille du tableau : n

-Les L(i), i=1, 2, ..., n doivent être de même type

5.1 Déclaration des tableaux

Comme pour les variables simples se pose le problème du type du tableau, c'est-à-dire de ses éléments. Tous les éléments du tableau sont de même type. Si le nom du tableau apparait dans un ordre type (REAL, INTEGER, ...)

Exemples:

ENTIER : TABLEAU A[10] *A est un tableau entier à 1 dimension, de 10 éléments*

REEL: TABLEAU B[15][5] *B est un tableau reel à 2 dimension, de 15 lignes et 5 colonnes.*

5.2 Manipulation d'un tableau

Une fois déclaré, un tableau peut être utilisé comme un ensemble de variables simples. Les trois manipulations de base sont l'affectation, la lecture et l'écriture.

- **L'affectation :**

Pour affecter une valeur à un élément i d'un tableau nommé par exemple A, on écrira : $$A[i] = valeur.

Par exemple, l'instruction : $$A[0] = 20 ; affecte au premier élément du tableau A la valeur 20. Pour affecter la même valeur à tous les éléments d'un tableau A de type numérique et de dimension 100, on utilise une boucle.

Exemple :

```
POUR i ALLANT 0 JUSQUA 100 PAS 1 FAIRE

$$A[i] = 0 ;

FINPOUR
```

Cette boucle permet de parcourir le tableau A élément par élément et affecter à chacun la valeur 0. La variable i est appelée indice.

- **La lecture :**

Comme les variables simples, il est possible aussi d'assigner des valeurs aux éléments d'un tableau lors de l'exécution c.-à-d. les valeurs sont saisies par l'utilisateur à la demande du programme.

Exemple :

```
ECRIRE("écrire une note :") ;

LIRE(A[6]) ;
```

Dans cet exemple, la valeur saisie est affectée au sixième (6ème) élément du tableau A.

- **L'écriture :**

De façon analogue à la lecture, l'écriture de la valeur d'un élément donné d'un tableau s'écrira comme suit : LIRE(A[i]) Cette instruction permet d'afficher la valeur de l'élément i du tableau A.

6. sous-algorithme

6.1 Procédure

Une procédure est un bloc d'instructions nommé et déclaré dans l'entête de l'algorithme et appelé dans son corps à chaque fois que le programmeur en a besoin.

```
PROCEDURE affiche(ENTIER:a1,ENTIER:a2)
    VAR
        ENTIER : s1;
    DEBUTP
    $s1 = $a1 + $b1;
    ECRIRE(s1);
FINPROCEDURE
```

6.2 Fonction

Une fonction est un bloc d'instructions qui retourne obligatoirement une et une seule valeur résultat à l'algorithme appelant. Une fonction n'affiche jamais la réponse à l'écran car elle la renvoie simplement à l'algorithme appelant

```
ENTIER : FONCTION somme(ENTIER:a1,ENTIER:b1)
    VAR
        ENTIER : s1;
    DEBUTF
    $s1 = $a1 + $b1;
    RETOURNE (s1);
FINFONCTION
```

7.Interface graphique

Compilateur en ligne pour l'algothme Il s'agit d'un serveur nodejs, expressjs pour exécuter le code algo entrant et renvoyer le résultat de la sortie. Le programme est exécuté sur le système serveur et le résultat est capturé et envoyé à nouveau au client comme réponse à la demande de publication.

ALgo Compiler v1.0

CODE *Run* **RESULT**

```
1  ALGO exemple
2      VAR
3          ENTIER : tmp;
4  DEBUT
5      POUR tmp ALLANT 1 JUSQUA 5 PAS 1 FAIRE
6          ECRIRE("HELLO WORLD");
7          RETOURE_LIGNE;
8      FPOUR
9  FIN
```

Input

Created by

LABIAD Salah Eddine *ZIRY Asmaa* *OUASSINE Younes* *HADDANI Abdelhakim*

ALgo Compiler v1.0

CODE *Run* **RESULT**

```
1  ALGO exemple
2      VAR
3          ENTIER : tmp;
4  DEBUT
5      POUR tmp ALLANT 1 JUSQUA 5 PAS 1 FAIRE
6          ECRIRE("HELLO WORLD");
7          RETOURE_LIGNE;
8      FPOUR
9  FIN
```

```
HELLO  WORLD
HELLO  WORLD
HELLO  WORLD
HELLO  WORLD
HELLO  WORLD
```

Input

Created by

LABIAD Salah Eddine *ZIRY Asmaa* *OUASSINE Younes* *HADDANI Abdelhakim*