Chap 2 : Les instructions élémentaires

Introduction

Nous allons voir de quoi est constitué un algorithme, puis comment en écrire. Un souci primordial lors de l'écriture d'un algorithme est sa lisibilité. Il sera lu et corrigé par différentes personnes, c'est pourquoi il est important de respecter la syntaxe et d'y ajouter le maximum de commentaires possibles.

Un algorithme est composé de données et d'instructions. Les données sont tous les objets que l'on utilise (nombre, texte...), les instructions permettent de les manipuler.

1) La structure d'un algorithme

Un algorithme (programme) sera constitué des trois parties suivantes, dans l'ordre : le nom du programme (identificateur), la déclaration des variables utilisées et enfin le traitement.
Pour que le compilateur puisse retrouver les instructions, on précède la première du mot-clé *début* et on fait suivre la dernière du mot *fin*.

Il faut aérer les programmes pour les rendre plus lisibles. On suivra donc les trois règles suivantes :

On saute des lignes entre les différentes parties du programme.
On ne met qu'une instruction par ligne.
On indente à bon escient.

Un programme aura donc la structure suivante :

```
programme identificateur

const
   // déclaration des constantes

var
   // déclaration des variables


début
   // première instruction
   // deuxième instruction
            …
   // dernière instruction
fin
```

Comment s'exécute un programme ? C'est très simple : il exécute la première instruction, puis la deuxième, la troisième et ainsi de suite jusqu'à la dernière. Le programme est alors terminé. C'est une exécution séquentielle.

2) Les données

a) les identificateurs

Chaque donnée d'un algorithme a un nom que l'on appelle un identificateur. Il faut respecter certaines règles pour ce nom soit valide :

- il est formé d'un seul mot
- il ne peut contenir que des chiffres, des lettres et le caractère souligné (_)
- le premier caractère ne peut être un chiffre
- les noms doivent être explicites, très utile pour l'humain (ex : prix_ht plutôt que k)
- il doit être unique
- l'espace est un séparateur de mot, il ne peut donc pas être utilisé dans identificateur. Le compilateur travaille mot après mot, il considère qu'un identificateur est terminé lorsqu'il rencontre un espace
- éviter les lettres accentuées car les compilateurs ne les acceptent pas (syntaxe anglo-saxonne).

b) le concept des variables

En programmation, il est nécessaire de garder temporairement des données afin de les utiliser pour effectuer des calculs ou des comparaisons.
Pour cela, on utilise une variable qui est une zone de stockage temporaire. Elle est désignée par un nom unique et caractérisée par un type de donnée (numérique, texte…). A un nom de variable correspond un espace réservé dans la mémoire vive de l'ordinateur.

On distingue :
Si la valeur d'une donnée peut être modifiée, on parle de variable.
Si la valeur ne peut pas être modifiée, on parle de constante.

Remarque:
Une variable ne peut avoir plusieurs valeurs en même temps.

c) les types de données élémentaires

A une variable, on associe un type de données. Cela permet au programme de connaître l'espace à réserver dans la mémoire vive.
Voici les principaux types :

Type	Valeurs
Entier	-58, 0, 328
Réel	-58, 3.1415, -1.1
Caractère	'c', '7', ' '
Chaîne	"oh la belle chaîne !", ""
Booléen	vrai, faux

Les entiers :
On utilise l'écriture décimale (exemple : 1324). On peut faire des opérations dessus : + , - , / , x , division entière et modulo (exemple : 7 DIV 2 → 3 ; 9 MOD 4 → 1).

Les réels :
On peut faire les mêmes opérations que sur les entiers (sauf DIV et MOD).

Les caractères et les chaînes :
Que sont *c*, *a*, *maison* ? D'après les règles précédentes, ce sont des identificateurs. De même, *Bonjour maman* sont deux identificateurs qui se suivent (ce qui ne veut rien dire, le compilateur provoquera une erreur). Pour distinguer :

- les caractères des identifiants, on les entoure d'apostrophes. Ainsi c est un identificateur, `'c'` un caractère ;
- les chaînes des identifiants, on les entoure de guillemets. `"Coucou"`, `"maison"` sont des chaînes.

L'opérateur & (ou +) sert à concaténer des chaînes de caractère, ce qui signifie transformer plusieurs chaînes en une seule en les ajoutant les unes à la suite des autres.
Exemple :
"Bon" & "jour" donne "Bonjour"

Les booléens :
Ils permettent de réaliser des tests. Un booléen ne peut prendre que deux valeurs : soit vrai, soit faux.

d) la déclaration des variables

Si le compilateur voit l'identifiant *age*, il ne saura pas quoi faire: qu'est-ce qu'« age » ? Un réel, une chaîne ? Il faut donc déclarer explicitement les variables avant de les utiliser. Or, une variable est caractérisée par son nom et son type (et pas par son contenu, puisque celui-ci varie). La déclaration d'une variable se fera de la façon suivante :

```
var
   variable : type
```

La variable aura pour nom *variable* (qui doit être un identificateur valide) et sera de type *type*.

Exemple : Définition de la variable *age*, de type entier

```
var
   age : entier
```

Pour déclarer plusieurs variables, il suffit d'écrire successivement les différentes déclarations (l'ordre de déclaration n'a aucune importance). Il n'est pas obligatoire de répéter le mot-clé *var* dans une suite de déclarations.
Dans une suite de déclarations, seul le premier *var* est obligatoire.
Par souci de simplification, on peut déclarer en même temps plusieurs variables de même type. Pour déclarer *n* variables de nom var_1 … var_n de type *type*, on écrira :

```
var
   var1, var2 … varn : type
```

Ainsi, pour déclarer prix_ttc, prix_ht, taux_tva et quantité_achetée, on écrira :

```
var
   prix_ttc, prix_ht, taux_tva : réel
   quantité_achetée            : entier
```

Remarque :
La déclaration d'une variable et son initialisation (le fait de lui donner une valeur) sont deux opérations distinctes. Après leur déclaration, les variables ont une valeur indéterminée (et surtout pas 0). En effet, dire que *prix* est un réel ne donne aucune information sur sa valeur.

Ainsi, toute variable déclarée doit donc être initialisée avant d'être utilisée.
Nous verrons cela par la suite lors des affectations d'une variable.

e) la déclaration d'une constante

Une constante a une valeur fixe pour l'ensemble de l'algorithme. Une constante doit être déclarée au début de l'algorithme en indiquant son identificateur et sa valeur (son type devient inutile, il est conditionné par la valeur).

```
const
   nom_constante : valeur
```

Exemple :

```
const
   taux_tva ← 0,196
```

f) la notion d'expression

Une expression est une combinaison d'opérateurs et d'opérandes dont le résultat est obligatoirement d'un des cinq types de base.

Voici quelques exemples d'expressions entières :

6
2+3
x+y-5 (si x et y sont des variables entières)
x (si x est un entier)

Sémantiquement, une expression d'un type donné est strictement équivalente à sa valeur (ce qui signifie que 2+3 est équivalent à 5).
Le cas de l'expression booléenne est important. Les deux valeurs booléennes (*vrai* et *faux*) ne sont jamais utilisées telles quelles car elles n'ont aucun intérêt. En revanche, on veut souvent savoir ce que vaut une expression booléenne, c'est-à-dire si elle est vraie ou fausse.
Tous les opérateurs de comparaison fournissent des résultats booléens (ils sont vrais ou faux).

Exemples :

2 > 3
2+1 > 3
2 = 2
x > y
x < x +1

Remarques :

1. Les opérateurs de comparaison habituels existent : >, <, >=, <=, =, <>.
2. De même, les opérateurs arithmétiques : +, -, *, /, *div* (division entière), *mod*. Attention aux types : un entier combiné avec un réel donnera un réel ; *div* et *mod* n'ont de sens qu'avec des entiers.

Série d'exercices

Exercice 1 :
Identifier dans la liste suivante les identificateurs non valides. Justifier la réponse.

- Prix toutes taxes F
- Adr2 V
- 1Prenom F
- Prix-ttc F
- ca V
- un_prenom V
- -titi F
- lm_12_kjf V

Exercice 2 :
Identifier dans la liste suivante les expressions non valides. Justifier la réponse.

- k MOD 4.5 F
- -19.6 V
- "mon caractère" V
- c = 'ma maison' F
- " toto" > 25 F
- " 16.52" V
- machine = lessive * linge V
- result > = 10 * coeff V

Exercice 3 :
Donner le type de chacune des variables ou expressions suivantes.

- -2.56 Réel
- 2 = 4 Bool
- ‘c’ Caract
- "Prix-ttc" Chaîne
- -456 Entier
- 47 > dep Bool
- 12 < 45 DIV j Bool
- "12.25" Chaîne

3) L’instruction « afficher »

Une instruction affichant des données à l’écran est nécessaire.

```
afficher (expr1, expr2… exprn)
```

si n vaut 1, cela donne :

```
afficher (expr)
```

Il faut bien noter que les paramètres d’*afficher* sont des expressions et non des variables ; ils doivent être d’un des types de base (entier, réel, booléen, caractère ou chaîne). Cette instruction affiche successivement à l’écran les *n* expressions.
En pratique, lors d’un affichage, le programme prend successivement chaque paramètre. Si c’est un nombre, il l’affiche ; si c’est une variable, il affiche sa valeur et si c’est une expression, il l’évalue (calcule) et affiche sa valeur.

Exemples d’affichages :

```
afficher ("c'est bon maintenant")
afficher (400*2+200, "fois", 'm', 'e', 'r', 'c', 'i')
```

4) L’instruction « saisir »

Le seul fait d’afficher n’est pas très intéressant. Il faut pouvoir lire des valeurs par le biais du clavier pour que l’utilisateur puisse lui-même donner des valeurs aux variables.
Le principe consiste à lire la valeur d’une variable au clavier. Une fois l’instruction exécutée, la variable aura pour valeur ce que l’utilisateur aura entré (on parlera d’affectation d’une valeur à une variable).

```
saisir (variable)
```

Il faut bien comprendre la sémantique de cette instruction : lorsqu’on l’exécute, il ne se passe rien (à l’écran). Le programme attend que l’utilisateur tape quelque chose. Une fois que cela est fait, la variable est affectée et l’exécution continue par l’instruction suivante.

Remarques :

1. *Afficher* est la seule instruction permettant de donner des informations à l’utilisateur.
2. Généralement, *saisir* est toujours précédé d’*afficher* précisant à l’utilisateur ce qu’il doit taper. C’est une règle d’ergonomie.
3. N’oublions pas que toute variable utilisée doit être déclarée

Ecrire l'algorithme permettant de calculer un montant de TVA

```
algo MontantTVA

var
    prixHT          : réel
const
    TauxTVA=0.196

début
   afficher ("Entrez un prix en Eur: ")
   saisir (prixHT)
   afficher ("Le montant de TVA correspondant en euros est : ", prixHT * 6.55957)

fin
```

5) L'affectation

L'instruction *saisir* permet de modifier la valeur d'une variable, mais elle nécessite l'entrée au clavier de la nouvelle valeur. L'instruction d'affectation permet au programme de modifier « tout seul » la valeur d'une variable. Cette instruction doit bien entendu avoir deux paramètres : une variable et une valeur.

```
variable ← expression
```

Remarques :
Bien noter la souplesse de l'affectation, qui autorise les expressions.
La variable doit impérativement être de même type que l'expression (c'est la même contrainte que *saisir*).
Avant d'utiliser une variable, il faut l'initialiser. L'affectation directe (←) ou par lecture au clavier (*saisir*) sont les deux seules instructions disponibles pour le faire.
L'affectation est différente de l'égalité mathématique. Quand on dit qu'une variable prend pour valeur une autre variable, ça ne veut pas dire qu'elles seront toujours égales ! Cela veut seulement dire qu'au moment de l'instruction, la première variable va prendre la valeur de la seconde.

Exemple :

```
Algo machin
Variables
a, b : entiers //a et b ont une valeur indéterminée
Début
   a ← 3 // a vaut 3
   // b n'a pas encore de valeur (ou plus exactement a une valeur indéterminée)
   b ← a + 2 // b vaut 5, a vaut toujours 3
   a ← b * 2 // a vaut 10, b vaut toujours 5
   b ← b + 1 // b vaut 6
Fin
```

Voici quelques exemples d'affectations qui se suivent (donner le type des variables).

```
i ← 0                  entier
i ← i+1                entier
a ← 'a'                caractère
j ← j + i              entier
n ← i > j              bool
k ← "coucou"           chaine
m ← 2.36 * j           réel
p ← 2 * m              réel
q  ← m + m             réel
reçu ← moy > = 10      bool
```

Exercice: la permutation
Comment échanger les valeurs de deux variables a et b ?
Que pensez-vous de l'algorithme suivant ? Permet-il de résoudre notre problème ?
Ecrire l'algorithme permettant d'échanger les valeurs de deux variables.

```
Algo bidule
Variables
a, b : Entier
Début
   a ← 3
   b ← 5
   a ← b
   b ← a
Fin
```

6) Les commentaires

Nous avons vu qu'un algorithme devait être le plus lisible et compréhensible possible pour les programmeurs. Il existe deux façons de respecter cette règle de bonne conduite : présenter son travail de façon soignée (indentation, identificateurs pertinents…) et expliquer ce que l'on fait.

Même bien présenté, un algorithme peut rester complexe, de par sa longueur, le nombre de variables manipulées ou tout simplement sa difficulté. Il est toujours agréable pour le lecteur d'avoir une brève description en début de programme (en français standard) du principe de l'algorithme et du résultat qu'il fournit.
C'est tout un art que de donner une explication concise et pertinente, les deux extrêmes étant une simple paraphrase du nom du programme ou une recopie de l'algorithme en français.
L'introduction de l'algorithme doit permettre de comprendre le principe général de l'algorithme.

Le principe général d'un algorithme ne renseigne pas sur sa validité, c.-à-d. sur l'exactitude de chaque instruction. Il ne faut pas hésiter à expliquer (brièvement mais correctement) tout ce qui est complexe, astucieux, optimisé. à chaque fois que l'on se pose des questions face à un programme, les commentaires doivent fournir la réponse.
Un commentaire débute par « /* » et se terminent par « */ ». Ils peuvent être écrits sur plusieurs lignes et leur contenu est libre (ils sont ignorés par le compilateur).

```
/* Ceci
        est un
            commentaire libre*/
```

Pour les petites indications brèves tenant sur une ligne, on utilise « // ». Tout ce qui suit « // » (sur la même ligne) est ignoré par le compilateur.

```
// Commentaire de ligne
```

Exercices d'application :
Ecrire un algorithme qui va calculer et afficher la moyenne de trois nombres que l'utilisateur aura au préalablement saisis. Cette moyenne devra être conservée dans une variable.

```
programme calcul
var
   nb1, nb2, nb3 : entier
   result        : réel
début
   afficher ("Entrez les 3 nombres : ")
   saisir (nb1, nb2, nb3)
   result ← (nb1+nb2+nb3)/3
   afficher ("La moyenne de ces trois nombres est ", result)
fin
```