1
Variables

Une des notions les plus importantes en programmation est la notion de variable.

C'est quoi une variable et à quoi sert-elle?

En programmation, on utilise des variables pour stocker les données à manipuler (données du problème). En fait, pour qu'un ordinateur puisse effectuer un calcul ou résoudre un problème, il doit tout d'abord stocker toutes les données de ce problème dans sa mémoire. Par exemple, pour pouvoir résoudre les équations du second degré ($ax^2 + bx + c = 0$), en plus des coefficients a, b et c, l'ordinateur a besoin de stocker la valeur de Δ.

1. Déclaration des variables

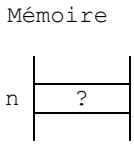
Une variable représente un espace mémoire qui sert à stocker une seule donnée. On peut utiliser une variable pour stocker soit un nombre, une lettre (caractère), un mot, une phrase (chaîne de caractères), ...

La déclaration d'une variable en C/C++ suit la syntaxe suivante:

```
type identifier;
```

Exemple 1

```
int n;
```



Cette ligne de code veut dire réserver un espace mémoire qui

- peut contenir seulement des nombres entiers[3].
- a pour nom 'n'.
- contient pour le moment une valeur quelconque (?).

En général, une variable possède trois caractéristiques: type (genre), identificateur (nom) et valeur (donnée).

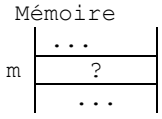
- *Type* **:** permet de déterminer si la donnée stockée dans la variable est un nombre (entier ou décimal), un caractère (lettre, symbole, ...), un ensemble de caractères (mot, phrase, ...), etc. Et ce afin de pouvoir déterminer (1) l'ensemble des valeurs possibles que la variable peut stocker et (2) l'ensemble des traitements (opérations) qu'on peut réaliser sur la variable.
- *Identificateur* : sert à référencer la donnée pour pouvoir l'utiliser à plusieurs endroits dans le programme.
- *Valeur* : représente la donnée stockée dans la variable. Par exemple, si la variable est de type nombre sa valeur pourra être 19 ou 3.14.

[3] int veut dire entier (en anglais integer); i.e., la variable n ne peut stocker que des nombres entiers.

Exemple 2 : si on déclare une variable `'m'` comme suit :

```
int m;
```

Alors, l'ordinateur va réserver un espace (de type `int`) dans sa mémoire et l'appeler `'m'`. Pour le moment, cet espace contient une valeur quelconque (`?`).



Identificateur : m
Type : int
Valeur: quelconque (indéfinie).

Types de variables

Les principaux types de base en C/C++ sont :

1. `int`
   Nombres entiers : `-(max) ... -75 -8 0 5 71 ... (max - 1)`

   On trouve aussi, `unsigned int`
   pour les nombres entiers positifs : `0 1 2 3 4 5 6 ...`

2. `double`
   Nombres à virgule : `... -10.5 -5.13 0 12.0 15.38 ...`

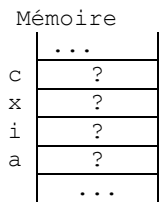
3. `char`
   Tous les caractères (touches) du clavier : `A .. Z a ..z 0..9 # é ^ + - / * _ . : , ; [ ] { } ( ) ...`

4. `Bool`
   Type booléen (logique): `true false`
   (Ce type sera détaillé dans une section ultérieure).

Exemple 3

```
#include<iostream>
using namespace std;
int main(){
       int a;
       unsigned int i;
       double x;
       char c;

       return 0;
}
```



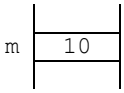
2. Initialisation des variables

Au moment de la déclaration d'une variable, on peut lui affecter une valeur initiale comme suit:

```
type identifier = initial_value;
```

Exemple 1

```
int m = 10;
```



Identificateur : m
Type : int
Valeur: 10

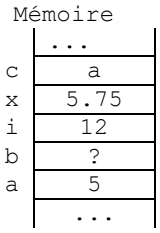
Cette fois-ci, on a donné une valeur initiale à la variable `m`. En d'autres termes, on a réservé un espace mémoire et on y a mis la valeur 10.

Exemple 2

```
#include<iostream>
using namespace std;

int main(){
	int a = 5;
	int b;
	unsigned int i = 12;
	double x = 5.75;
	char c = 'a';

	return 0;
}
```



Remarque : Si une variable n'est pas initialisée alors, elle peut contenir n'importe quelle valeur. Donc, il faut toujours initialiser les variables avant de les utiliser.

Autres exemples de déclaration

On peut déclarer les variables séparément comme suit:

```
int a;
int b = 2;
int c;

double x = 2.23;
double y = 12.18;
double z;
```

Comme on peut déclarer plusieurs variables de même type sur la même ligne en les séparant par des virgules :

```
int a, b = 2, c;
double x = 2.23, y = 12.18, z;
```

3. Nommage des variables

On n'est pas obligé de nommer les variables `a, b, x, y, n,` ... On peut donner des noms significatifs à nos variables comme par exemple :

```
double temperature;
double pression;
double speed;

double grade1, grade2, average;
int numStudents;
```

En effet, les programmes informatiques sont écrits pour être lus et compris par les être-humains (le programmeur lui-même ou autres programmeurs) et non pas par les machines. C'est pour cette raison, il est très conseillé de donner des noms significatifs qui reflètent le rôle des différentes variables et permettent de connaître ce qu'elles stockent. Par exemple, le nom `temperature` signifie que la variable stocke la température.

Cependant, pas tous les noms sont acceptés, il existe certaines règles à respecter lors du choix des noms de variables.

1. Le nom de la variable doit être constitué seulement des caractères suivants :
   - `A..Z, a..z` : 26 lettres de l'alphabet (minuscules et majuscules).
   - `0..9 :` chiffres.
   - `_ :` caractère souligné (underscore), appelé aussi tiret du 8.

   Sauf les caractères cités ci-dessus sont autorisés pour nommer une variable. Ce qui veut dire ni les espaces ni tout autre symbole n'est accepté.
   `# . ? ! : " * $ ( ) & [ ] - { } , :` etc.

   Même chose pour les lettres accentuées, elles ne sont pas acceptées : `é è ê à â ù û ...`[4]

   Exemple de déclarations valides

```
int sum;
int x1, x2, x3, y1, z31;
int num_students;
int numColors;
```

   Exemple de déclarations invalides

```
int $prog;                    // invalide: contient le symbole $
int u.t.c.;                   // invalide: contient le point
int A!8;                      // invalide: contient '!'
int num students;             // invalide: contient un espace
int num-students;             // invalide: contient le symbole moins '-'
int nombre_étudiants;         // invalide: contient 'é'
```

2. Le premier caractère du nom de la variable doit être obligatoirement une lettre (`A..Z, a..z`). Le caractère souligné (_) est considéré comme une lettre. Donc, le nom d'une variable peut commencer par ce caractère.

   Exemple de déclarations invalides

```
int 1element;                       // invalide: commence par un chiffre
int 1stNumber, 2ndNumber;           // invalide: commence par un chiffre
```

   Exemple de déclarations valides

```
int element1;
int number1, number2;
int _number;
```

[4] En cas d'obligation, on peut utiliser tout simplement la lettre 'e' au lieu de 'é', et ainsi de suite.

3. Le nom de la variable ne doit pas être un mot-clé réservé par le langage C++, comme par exemple: `return, using, int, double, char, ...`

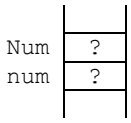
Exemple de déclarations invalides

```
int int;         // invalide: int est un mot-clé
int double;      // invalide: double est un mot-clé
int using;       // invalide: using est un mot-clé
int return;      // invalide: return est un mot-clé
```

4. Les majuscules et les minuscules sont autorisées mais ne sont pas équivalentes. Le langage C/C++ est dit langage sensible à la casse (case-sensitive), i.e, il fait la différence entre majuscule et minuscule.

Exemple

```
int num;
int Num;
```

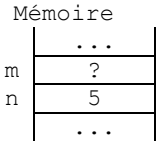


Les variables `num` et `Num` sont deux variables différentes. De même, les variables `x` et `X` sont deux variables différentes.

4. Affectation des variables

Maintenant, nous savons comment déclarer, nommer et initialiser une variable. Par exemple:

```
int m;
int n = 5;
```



`m` : variable (case mémoire) contenant une valeur quelconque (`?`).
`n` : variable contenant la valeur `5`.
Comme son nom l'indique une variable "varie" ou change de valeurs. A titre d'exemple, si une variable stocke la température alors il est bien évident que cette variable pourrait changer sa valeur parce que la température change.

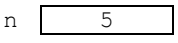
Comment peut-on changer la valeur d'une variable? Pour changer la valeur d'une variable, on utilise le symbole `'='`.

Exemple 1

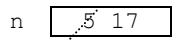
```
int n = 5;
n = 17;
```

Exécution pas à pas

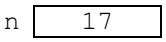
```
int n = 5;
```




```
n = 17;
```



`n = 17;` veut dire mettre la valeur `17` dans la variable `n`. L'ancienne valeur de la variable `n` (c'est-à-dire `5`) est écrasée et remplacée par la nouvelle valeur (`17`). Maintenant `n` contient `17` au lieu de `5`.



Notez bien qu'une variable ne peut contenir qu'une seule valeur à la fois.

Exemple 2

```
int m;
m = 8;
```

Exécution pas à pas

```
int m;
```


```
m = 8;
```

Cette dernière instruction (i.e., `m = 8`) veut dire mettre `8` dans la variable `m`.

L'opération qui permet de changer la valeur d'un variable est appelé **affectation**.

Exemple 3

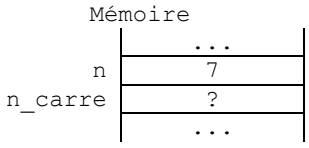
```
int n = 7;
int n_carre;
n_carre = n * n;
```

L'affectation se fait en deux étapes:

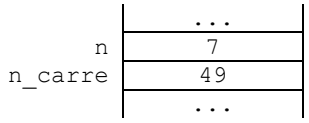
1. évaluer l'expression à droite du signe `'='`. Dans l'exemple précédent, évaluer l'expression :
   `n * n`
   `n * n = 7 * 7 = 49`
2. stocker le résultat dans la variable à gauche. Dans cet exemple, stocker la valeur `49` dans la variable `n_carre`.

Exécution pas à pas

```
int n = 7;
int n_carre;
```




```
n_carre = n * n;
```



Remarque : Soit le code suivant:

```
int a = 12, b = 25;
int c = a * b;
```

Le symbole de la multiplication est `'*'.` Pour multiplier deux variables `a` et `b`, on écrit :

```
int c = a * b;
```

Les opérations suivantes ne sont pas acceptées :

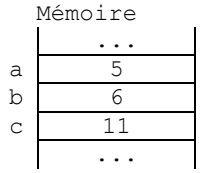
```
int c = a x b;
int c = a.b;
int c = ab;
int c = a b;
```

Exemple 4

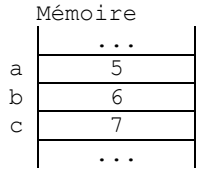
```
int main(){
        int a = 5;
        int b = 6;
        int c = a + b;
        c = a + 2;
        return 0;
}
```

Exécution pas à pas

```
int a = 5;
int b = 6;
int c = a + b;
```




```
c = a + 2;
```



On remarque bien que l'affectation écrase l'ancienne valeur de la variable et la remplace par la nouvelle valeur.

De façon générale, l'affectation suit le schéma suivant:

```
nom_variable = expression;
```

Exemple d'expressions

```
4
n
n * n
n * (n - 1) + 3 * n - 2
```

Exemples d'affectation

```
z = 1;
resultat = 2 * 3 + 5;
x = -b / a;
nombre1 = nombre2 + 1;
a = b;
a = 3 * 5;
b = a + 5;
```

Nous allons discuter les expressions avec plus de détails dans la section suivante (Expressions et opérateurs).

Remarque 1 : Soit le morceau de code suivant:

```
int x = 60, y = 23;
int z;
x + y = z;
```

L'instruction (`x + y = z;`) n'est pas valide parce que à gauche du symbole d'affectation `'='`, on doit toujours trouver une variable, c.-à-d., une case mémoire dans laquelle on peut mettre le résultat de l'expression qui se trouve à droite du symbole `'='`. On devait écrire:

```
z = x + y;
```

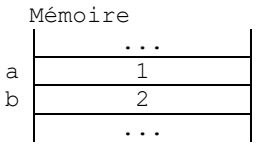
Remarque 2 : Ne confondez pas l'affectation avec l'égalité mathématique. Toutes les deux utilisent le signe égal '=', mais elles ne signifient pas la même chose.

Exemple 1 : En mathématique, `a = b` et `b = a` sont identiques. Mais en programmation, les deux instructions d'affectation suivantes: `a = b;` et `b = a;` ne sont pas identiques.

`a = b;` copie la valeur de `b` dans `a`.
`b = a;` copie la valeur de `a` dans `b`.

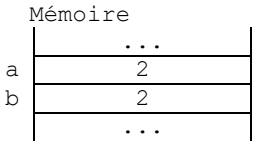
Soit le morceau de code suivant:

```
int a = 1;
int b = 2;
```



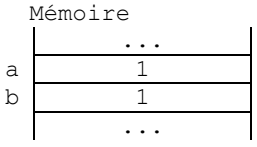
Après exécution de `a = b;` on obtient : `a = 2` et `b = 2`.

```
int a = 1;
int b = 2;
a = b;
```



Après exécution de `b = a;` on obtient : `a = 1` et `b = 1`.

```
int a = 1;
int b = 2;
b = a;
```



Exemple 2 : Soit le morceau de code suivant :

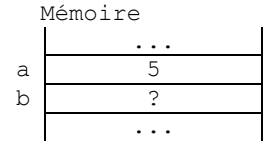
```
int a = 5, b;
b = a + 1;
a = 2;
```

En mathématique, `b = a + 1` veut dire que quelle que soit la valeur de `a`, `b` égale toujours `a + 1`. Mais, l'affectation n'est pas une égalité mathématique. En programmation, l'instruction

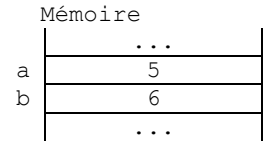
`b = a + 1` veut dire mettre le résultat de l'expression (`a + 1`) dans la variable `b`.

Exécutons le code précédent pas à pas.

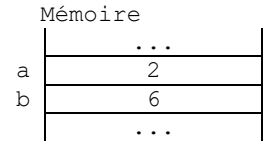
```
int a = 5, b;
```




```
b = a + 1;
```




```
a = 2;
```



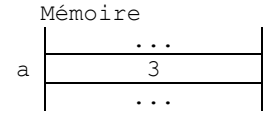
On remarque bien qu'après exécution de `a = 2;` que `b ≠ a + 1`.

Exemple 3 : Soit le code suivant :

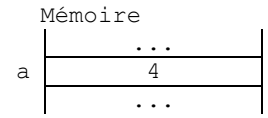
```
int a = 3;
a = a + 1;
```

Exécutons les instructions pas à pas :

```
int a = 3;
```



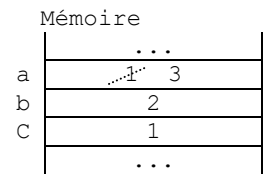

```
a = a + 1;
```



`a = a + 1;` cette instruction veut dire évaluer l'expression `a + 1` et ranger le résultat dans la variable `a`. Ce qui revient à augmenter la valeur de `a` de 1.

Exemple 4 : Soit le morceau de code suivant:

```
int a = 1;
int b = 2;
int c = a;
a = 3;
```



L'instruction `int c = a;` ne veut pas dire que la variable `c` égale la variable `a` dans tout le programme. En d'autres termes, elle ne veut pas dire que si la variable `a` change sa valeur alors la variable `c` aussi change sa valeur. En fait, l'instruction `int c = a;` veut dire copie la valeur de la variable `a` dans la variable `c`.

Remarquez bien que l'instruction `int c = a;` ne vide pas la variable `a` de sa valeur, elle copie juste la valeur de `a` dans la variable `c`.

Remarque : Contrairement à la valeur stockée dans une variable, le type de la variable, une fois défini, ne peut pas changer durant l'exécution du programme. Par exemple, une variable de type `int` reste de type `int` durant l'exécution du programme.

Déclaration des constantes

Soit les déclarations suivantes:

```
double vitesse_lumiere = 299792.458;
double PI = 3.14159;
```

Comme nous avons déjà appris, on peut à tout moment dans un programme changer la valeur stockée dans une variable en lui affectant une nouvelle valeur. Cela veut dire qu'on peut affecter les deux variables `vitesse_lumiere` et `PI` comme suit:

```
vitesse_lumiere = 500.17;
PI = 5.3;
```

Mais, on sait bien que la vitesse de la lumière est toujours `299792.458` et π égale toujours `3.14159`. Par conséquent, si nous avons une variable et on ne veut pas qu'elle change de valeur dans le programme, on doit la déclarer comme constante selon la syntaxe suivante:

```
const type identificateur = valeur;
```

Exemple

```
const double VITESSE_LUMIERE = 299792.458;
const double PI = 3.14159;
```

Une fois déclarée, la valeur d'une constante ne peut pas être changée et toute tentative de modification sera considérée comme erreur.

```
const double PI = 3.14159;
PI = 1.2;     // erreur de compilation
```

`PI` est une variable à lecture seule (read-only variable) ou constante, elle ne peut pas être changée.

Remarque : Ce n'est pas une obligation, mais par convention on écrit les noms des constantes en majuscules.

```
const double VITESSE_LUMIERE = 299792.458;
const double PI = 3.14159;
const int MAX = 10;
```