



Expressions arithmétiques en C++

PLAN

- 1.1 Introduction
- 1.2 Les entiers
- 1.3 Les réels
- 1.4 Les règles de conversion implicites
- 1.5 Les règles de conversion explicites
- 1.6 Les constantes
- 1.7 Les opérateurs et la priorité des opérations
- 1.8 Récapitulatif

OBJECTIFS

- Connaître les différents types pris en compte par C++
- Utiliser les opérateurs
- Appliquer les bonnes règles de conversion

1.1 INTRODUCTION

Tout langage de programmation manipule des **variables**.

Une *variable* est un **identificateur** qui désigne un **type** d'information dans un programme. Elle est située dans un endroit précis de la mémoire de la machine et représente souvent une donnée élémentaire, c'est-à-dire une valeur numérique ou un caractère.

Pour associer une valeur à une *variable* on utilise le mécanisme d'**affection**. En C++, l'*opérateur d'affection* est le signe =.

Une *variable* possède obligatoirement un *type* qui définit l'encombrement mémoire que prendra la *variable*.

Les *types* de données de base sont les suivant :

- **int** : valeur entière
- **char** : caractère simple
- **float** : nombre réel en virgule flottante
- **double** : nombre réel en virgule flottante double précision

Des **qualificateurs** (ou **spécificateurs**) comme **short**, **signed**, **unsigned** peuvent enrichir les *types* de données.

En C++, le compilateur doit être informé des *types* de *variables* qui seront utilisés dans le programme, pour ce faire, le développeur va donc faire une **déclaration**.

Pour déclarer une *variable*, on précise son *type* suivi de son *identificateur* (son nom).

L'*identificateur* d'une *variable* est composé d'un ensemble de chiffres ou de lettres dans un ordre quelconque qui suit les règles suivantes :

- Le premier caractère doit être obligatoirement une lettre.
- Les minuscules ou les majuscules (la **casse**) sont autorisées et considérées comme différentes.
- Le caractère `_` (underscore ou blanc souligné) est autorisé.
- Il faut veiller à ne pas utiliser des **mots-clés** ou **mots réservés** du langage C++ (voir annexe 5) ou des **séquences d'échappement** (voir annexe 2)

Voici quelques exemples d'*identificateurs* admis :

X	x	X1a
somme_totale		X1a
taux	N5	_montant
Total	N5	PRODUIT

La *déclaration* d'une *variable* peut être assimilée à la création en mémoire d'un contenant dont le *type* serait la taille (ou dimension) et le contenu, la valeur.

Quand aucune valeur n'a encore été affectée à une *variable*, seule sa place est réservée. Son contenu n'étant pas encore défini, il viendra ultérieurement.

Au cours de la rédaction d'un programme, une *variable* peut être déclarée à tout moment, toutefois, les développeurs ont pour habitude de regrouper les déclarations, ce qui apporte une meilleure lisibilité et une compréhension accrue lors de la lecture.

Dans une *déclaration*, on peut mentionner le *type* une seule fois pour plusieurs *variables*. Il suffit de les séparer par une virgule.

L'endroit où est déclarée une *variable* définit son **accessibilité** ou sa **visibilité**.

Quand une *variable* est déclarée dans le code même, c'est-à-dire à l'extérieur d'un bloc d'instructions ou d'une fonction, elle est considérée comme une **variable globale** et reste accessible depuis n'importe quel endroit du code.

Une *variable* déclarée dans un bloc d'instruction voit sa portée limitée à l'intérieur de ce bloc, on parle de **variable locale**. En C++, un bloc d'instructions est délimité par des accolades.

Il existe un opérateur de résolution de **portée** qui offre l'accès à des *variables globales* plutôt que *locales*, il se note `::`.



PORTÉE DES VARIABLES

```
#include <iostream>
using namespace std;
//déclarations et initialisations des variables
globales e et i
double e=2.718 ;
int i=1;
//programme principal
int main(){
    //déclarations et initialisations des variables
    locales j et e
    int j=2;
    double e=1.282 ;
    //somme e:globale + e:locale
    e::e+e;
    //affichage des résultats
    cout<<"La somme e = <<<e<<<endl;
    cout<<"La variable locale j = "<<j<<endl;
    cout<<"La variable globale i = "<<i<<endl;
    cout<<"la variable globale e = "<<::e<<endl;
    cout<<"La somme i + e (avec i:globale et e:locale)
    est égale à : <<<i+e<<<endl;
}
```

Résultat après exécution :

```
La somme e = 4
La variable locale j = 2
La variable globale i = 1
```

```

la variable globale e = 2.718
La somme i + e (avec i:globale et e:locale) est égale à
: 5

```

1.2 LES ENTIERS

Ils s'expriment sous trois *types* différents : **short int** (entier court), **int** (entier simple) et **long int** (entier long).

Chaque *type* peut être **signed** (signé) ou **unsigned** (non signé).

Il faut rajouter au *type* entier le *type* **char** (caractère) qui est aussi un *type* entier. Il manipule des caractères comme des entiers à l'intérieur d'un programme, chacun est pris en compte sous sa valeur **ASCII** (voir annexe 3).

Tableau 1-1 ENSEMBLE DES COMBINAISONS POSSIBLES EN C++ POUR LES ENTIERS.
LES PLAGES PRÉCISÉES PEUVENT VARIER EN FONCTION DE LA MACHINE ET DU COMPILATEUR.

Type	Plage couverte		Taille en octets
	Limite inférieure	Limite supérieure	
char	-128	127	1
unsigned char	0	255	1
short int	-32 768	32767	2
unsigned short int	0	65535	2
int	-2 147 483 648	2 147 483 647	4
unsigned int	0	4 294 967 295	4
long int	-2 147 483 648	2 147 483 647	4
unsigned long int	0	4 294 967 295	4

Le fichier d'en-tête¹ `<climits>` définit l'étendue des nombres entiers.



MANIPULATION DES ENTIERS

```

#include <iostream>
using namespace std;
int main ()
{
    //initialisation des variables
    char char_std=65;

```

¹ Voir annexe 6.

```

unsigned char char_uns=65;
short int int_short=1;
int int_std=1;
long int int_long=1;
unsigned short int int_short_uns=1;
unsigned int int_std_uns=1;
unsigned long int int_long_uns=1;
//affichage des variables après initialisation
cout<<char_std<<" "<<char_uns<<" "<<endl;
cout<<int_short<<" "<<int_std<<" "<<int_long
<<" ";
    cout<<int_short_uns<<" "<<int_std_uns<<" "
<<int_long_uns << endl;
//affichage de la taille en octets prise par chaque
variable
    cout<<sizeof(int_short)<<" "<<sizeof(int_std)<<" "
<<sizeof(int_long)<<" ";
    cout<<sizeof(int_short_uns)<<" "
<<sizeof(int_std_uns)<<" "
<<sizeof(int_long_uns)<<endl;
    cout << endl;
//boucle de 1 à 32 affichant la valeur de chaque
variable suivant une suite croissante des puissances de
2
    for (int i=0; i<=32; ++i)
    {
        int_short=int_short*2;
        int_std=int_std*2;
        int_long=int_long*2;
        int_short_uns=int_short_uns*2;
        int_std_uns=int_std_uns*2;
        int_long_uns=int_long_uns*2;
        //affichage de l'indice de la boucle
        cout<<"ind : "<<i<<endl;
        //affichage des valeurs calculées pour
chaque variable
        cout<<"short int : "<<int_short<<endl;
        cout<<"int : "<<int_std<<endl;
        cout<<"long int : "<<int_long<<endl;
        cout<<"unsigned short int : "
<<int_short_uns<<endl;
        cout<<"unsigned int : "<<int_std_uns<<endl;
        cout<<"unsigned long int : "
<<int_long_uns<<endl;
        cout<<endl;
    }
return(0);
}

```

Résultat après exécution :

```

A A
1 1 1 1 1 1
2 4 4 2 4 4

ind : 1
short int : 2
int : 2
long int : 2
unsigned short int : 2
unsigned int : 2
unsigned long int : 2

ind : 2
short int : 4
int : 4
long int : 4
unsigned short int : 4
unsigned int : 4
unsigned long int : 4
.
.
.
ind : 31
short int : 0
int : -2147483648
long int : -2147483648
unsigned short int : 0
unsigned int : 2147483648
unsigned long int : 2147483648

ind : 32
short int : 0
int : 0
long int : 0
unsigned short int : 0
unsigned int : 0
unsigned long int : 0

```



Remarques. La manipulation des *variables* de type `char` comme des entiers au sein de C++ peut sembler bizarre, cependant de nombreuses applications manipuleront ainsi très facilement des données au format 8 bits ce qui est intéressant pour traiter des caractères de façon simple.

Il faut aussi noter que le type `char` peut être aussi `unsigned` (non signé), comme un `int`.