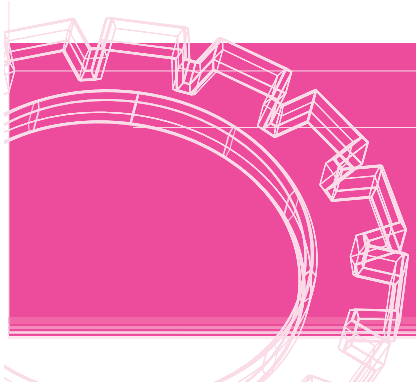
Chapitre 1

Généralités, types de base, opérateurs et expressions



Rappels

Généralités

Le canevas minimal à utiliser pour réaliser un programme C++ se présente ainsi :

```
#include <iostream>
using namespace std ;
main()          // en-tête
{ .....         // corps du programme
}
```

Toute variable utilisée dans un programme doit avoir fait l'objet d'une déclaration en précisant le type et, éventuellement, la valeur initiale. Voici des exemples de déclarations :

```
int i ;   // i est une variable de type int nommée i
float x = 5.25 ;   // x est une variable de type float nommée x
                   // initialisée avec la valeur 5.25
const int NFOIS = 5 ;  // NFOIS est une variable de type int dont la
                       // valeur, fixée à 5, ne peut plus être modifiée
```

L'affichage d'informations à l'écran est réalisé en envoyant des valeurs sur le « flot *cout* », comme dans :

```
cout << n << 2*p ;  // affiche les valeurs de n et de 2*p sur l'écran
```

La lecture d'informations au clavier est réalisée en extrayant des valeurs du « flot *cin* », comme dans :

```
cin >> x >> y ;  // lit deux valeurs au clavier et les affecte à x et à y
```

Types de base

Les types de base sont ceux à partir desquels seront construits tous les autres, dits dérivés (il s'agira des types structurés comme les tableaux, les structures, les unions et les classes, ou d'autres types simples comme les pointeurs ou les énumérations).

Il existe trois types entiers : `short int` (ou `short`), `int` et `long int` (ou `long`). Les limitations correspondantes dépendent de l'implémentation. On peut également définir des types entiers non signés : `unsigned short int` (ou `unsigned short`), `unsigned int` et `unsigned long int` (ou `unsigned long`). Ces derniers sont essentiellement destinés à la manipulation de motifs binaires.

Les constantes entières peuvent être écrites en notation hexadécimale (comme 0xF54B) ou octale (comme 014). On peut ajouter le « suffixe » `u` pour un entier non signé et le suffixe `l` pour un entier de type `long`.

Il existe trois types flottants : `float`, `double` et `long double`. La précision et le « domaine représentable » dépendent de l'implémentation.

Le type « caractère » permet de manipuler des caractères codés sur un octet. Le code utilisé dépend de l'implémentation. Il existe trois types caractère : `signed char`, `unsigned char` et `char` (la norme ne précise pas s'il correspond à `signed char` ou `unsigned char`).

Les constantes de type caractère, lorsqu'elles correspondent à des « caractères imprimables », se notent en plaçant le caractère correspondant entre apostrophes.

Certains caractères disposent d'une représentation conventionnelle utilisant le caractère « \ » notamment `'\n'` qui désigne un saut de ligne. De même, `'\''` représente le caractère ' et `'\"'` désigne le caractère ". On peut également utiliser la notation hexadécimale (comme dans `'\x41'`) ou octale (comme dans `'\07'`).

Le type `bool` permet de manipuler des « booléens ». Il dispose de deux constantes notées `true` et `false`.

Les opérateurs de C++

Voici un tableau présentant l'ensemble des opérateurs de C++ (certains ne seront exploités que dans des chapitres ultérieurs) :

Catégorie	Opérateurs	Associativité
Résolution de portée	:: (portée globale - unaire) :: (portée de classe - binaire)	<-- -->
Référence	() [] -> .	-->
Unaire	+ - ++ -- ! ~ * & sizeof cast dynamic_cast static_cast reinterpret_cast const_cast new new[] delete delete[]	<---
Sélection	->* .*	<--
Arithmétique	* / %	--->
Arithmétique	+ -	--->
Décalage	<< >>	--->
Relationnels	< <= > >=	--->
Relationnels	== !=	--->
Manipulation de bits	&	--->
Manipulation de bits	^	--->
Manipulation de bits	|	--->
Logique	&&	--->
Logique	||	--->
Conditionnel (ternaire)	? :	--->
Affectation	= += -= *= /= %= &= ^= |= <<= >>=	<---
Séquentiel	,	--->

Les opérateurs arithmétiques et les opérateurs relationnels

Les opérateurs arithmétiques binaires (+, -, * et /) et les opérateurs relationnels ne sont définis que pour des opérandes d'un même type parmi : `int`, `long int` (et leurs variantes non signées), `float`, `double` et `long double`. Mais on peut constituer des expressions mixtes

(opérandes de types différents) ou contenant des opérandes d'autres types (`bool`, `char` et `short`), grâce à l'existence de deux sortes de conversions implicites :

- les conversions d'ajustement de type, selon l'une des hiérarchies :

```
int -> long -> float -> double -> long double
unsinged int -> unsigned long -> float -> double -> long double
```

- les promotions numériques, à savoir des conversions systématiques de `char` (avec ou sans attribut de signe), `bool` et `short` en `int`.

Les opérateurs logiques

Les opérateurs logiques && (et), || (ou) et (non) acceptent n'importe quel opérande **numérique** (entier ou flottant) ou pointeur, en considérant que tout opérande de valeur non nulle correspond à « faux » :

Opérande 1	Opérateur	Opérande 2	Résultat
0	&&	0	faux
0	&&	non nul	faux
non nul	&&	0	faux
non nul	&&	non nul	vrai
0	||	0	faux
0	||	non nul	vrai
non nul	||	0	vrai
non nul	||	non nul	vrai
	!	0	vrai
	!	non nul	faux

Les deux opérateurs && et || sont « à court-circuit » : le second opérande n'est évalué que si la connaissance de sa valeur est indispensable.

Opérateurs d'affectation

L'opérande de gauche d'un opérateur d'affectation doit être une *lvalue*, c'est-à-dire la référence à quelque chose de modifiable.

Les opérateurs d'affectation (=, -=, += ...), appliqués à des valeurs de type numérique, provoquent la conversion de leur opérande de droite dans le type de leur opérande de gauche. Cette conversion « forcée » peut être « dégradante ».

Opérateurs d'incrémentation et de décrémentation

Les opérateurs unaires d'incrémentation (++) et de décrémentation (--) agissent sur la valeur de leur unique opérande (qui doit être une *lvalue*) et fournissent la valeur après modification lorsqu'ils sont placés à gauche (comme dans ++n) ou avant modification lorsqu'ils sont placés à droite (comme dans n--).

Opérateur de cast

Il est possible de forcer la conversion d'une expression quelconque dans un type de son choix, grâce à l'opérateur dit de « cast ». Par exemple, si n et p sont des variables entières, l'expression :

```
(double) n / p        // ou :   static_cast<double> (n/p)
```

aura comme valeur celle de l'expression entière n/p convertie en double.

Opérateur conditionnel

Cet opérateur ternaire fournit comme résultat la valeur de son deuxième opérande si la condition mentionnée en premier opérande est non nulle (vraie pour une expression booléenne), et la valeur de son troisième opérande dans le cas contraire. Par exemple, avec cette affectation :

```
max = a>b ? a : b ;
```

on obtiendra dans la variable max la valeur de a si la condition a>b est vraie, la valeur de b dans le cas contraire. Avec :

```
valeur = 2*n-1 ? a : b ;
```

on obtiendra dans la variable valeur la valeur de a si l'expression 2*n-1 est non nulle, la valeur de b dans le cas contraire.

Exercice 1

Énoncé

Éliminer les parenthèses superflues dans les expressions suivantes :

```
a = (x+5)           /* expression 1 */
a = (x=y) + 2       /* expression 2 */
a = (x==y)          /* expression 3 */
(a<b) && (c<d)      /* expression 4 */
(i++) * (n+p)       /* expression 5 */
```

Solution

```
a = x+5             /* expression 1 */
```

L'opérateur + est prioritaire sur l'opérateur d'affectation =.

```
a = (x=y) + 2        /* expression 2 */
```

Ici, l'opérateur + étant prioritaire sur =, les parenthèses sont indispensables.

```
a = x==y             /* expression 3 */
```

L'opérateur == est prioritaire sur =.

```
a<b && c<d           /* expression 4 */
```

L'opérateur && est prioritaire sur l'opérateur <.

```
i++ * (n+p)          /* expression 5 */
```

L'opérateur ++ est prioritaire sur * ; en revanche, * est prioritaire sur +, de sorte qu'on ne peut éliminer les dernières parenthèses.

Exercice 2

Énoncé

Soient les déclarations :

```
char c = '\x01' ;
short int p = 10 ;
```

Quels sont le type et la valeur de chacune des expressions suivantes :

```
p + 3            /* 1 */
c + 1            /* 2 */
p + c            /* 3 */
3 * p + 5 * c    /* 4 */
```

Solution

1. `p` est d'abord soumis à la conversion « systématique » `short -> int`, avant d'être ajouté à la valeur `3` (`int`). Le résultat `13` est de type `int`.
2. `c` est d'abord soumis à la conversion « systématique » `char -> int` (ce qui aboutit à la valeur `1`), avant d'être ajouté à la valeur `1` (`int`). Le résultat `2` est de type `int`.
3. `p` est d'abord soumis à la conversion systématique `short -> int`, tandis que `c` est soumis à la conversion systématique `char -> int` ; les résultats sont alors additionnés pour aboutir à la valeur `11` de type `int`.
4. `p` et `c` sont d'abord soumis aux mêmes conversions systématiques que ci-dessus ; le résultat `35` est de type `int`.

Exercice 3

Énoncé

Soient les déclarations :

```
char c = '\x05' ;
int n = 5 ;
long p = 1000 ;
float x = 1.25 ;
double z = 5.5 ;
```

Quels sont le type et la valeur de chacune des expressions suivantes :

```
n + c + p            /* 1 */
2 * x + c            /* 2 */
(char) n + c         /* 3 */
(float) z + n / 2    /* 4 */
```

Solution

1. `c` est tout d'abord converti en `int`, avant d'être ajouté à `n`. Le résultat (`10`), de type `int`, est alors converti en `long`, avant d'être ajouté à `p`. On obtient finalement la valeur `1010`, de type `long`.
2. On évalue d'abord la valeur de `2*x`, en convertissant `2` (`int`) en `float`, ce qui fournit la valeur `2.5` (de type `float`). Par ailleurs, `c` est converti en `int` (conversion systématique). On évalue ensuite la valeur de `2*x`, en convertissant `2` (`int`) en `float`, ce qui fournit la valeur `2.5` (de type `float`). Pour effectuer l'addition, on convertit alors la valeur entière `5` (`c`) en `float`, avant de l'ajouter au résultat précédent. On obtient finalement la valeur `7.75`, de type `float`.
3. `n` est tout d'abord converti en `char` (à cause de l'opérateur de « cast »), tandis que `c` est converti (conversion systématique) en `int`. Puis, pour procéder à l'addition, il est nécessaire de reconvertir la valeur de `(char) n` en `int`. Finalement, on obtient la valeur `10`, de type `int`.
4. `z` est d'abord converti en `float`, ce qui fournit la valeur `5.5` (approximative, car, en fait, on obtient une valeur un peu moins précise que ne le serait `5.5` exprimé en `double`). Par ailleurs, on procède à la division entière de `n` par `2`, ce qui fournit la valeur entière `2`. Cette dernière est ensuite convertie en `float`, avant d'être ajoutée à `5.5`, ce qui fournit le résultat `7.5`, de type `float`.

Exercice 4

Énoncé

Soient les déclarations suivantes :

```
int n = 5, p = 9 ;
int q ;
float x ;
```

Quelle est la valeur affectée aux différentes variables concernées par chacune des instructions suivantes ?

```
q = n < p ;                 /* 1 */
q = n == p ;                /* 2 */
q = p % n + p > n ;         /* 3 */
x = p / n ;                 /* 4 */
x = (float) p / n ;         /* 5 */
x = (p + 0.5) / n ;         /* 6 */
x = (int) (p + 0.5) / n ;   /* 7 */
q = n * (p > n ? n : p) ;   /* 8 */
q = n * (p < n ? n : p) ;   /* 9 */
```

Solution

1. `1`
2. `0`
3. `5` (`p%n` vaut 4, tandis que `p>n` vaut `1`).
4. `1` (`p/n` est d'abord évalué en `int`, ce qui fournit `1` ; puis le résultat est converti en `float`, avant d'être affecté à `x`).
5. `1.8` (`p` est converti en `float`, avant d'être divisé par le résultat de la conversion de `n` en `float`).
6. `1.9` (`p` est converti en `float`, avant d'être ajouté à `0.5` ; le résultat est divisé par le résultat de la conversion de `n` en `float`).
7. `1` (`p` est converti en `float`, avant d'être ajouté à `0.5` ; le résultat (`5.5`) est alors converti en `int` avant d'être divisé par `n`).
8. `25`
9. `45`

Exercice 5

Énoncé

Quels résultats fournit le programme suivant :

```
#include <iostream>
using namespace std ;
main ()
{
   int i, j, n ;
   i = 0 ; n = i++ ;
   cout << "A : i = " << i << " n = " << n << "\n" ;

   i = 10 ; n = ++ i ;
   cout << "B : i = " << i << " n = " << n << "\n" ;
   i = 20 ; j = 5 ; n = i++ * ++ j ;
   cout << "C : i = " << i << " j = " << j << " n = " << n << "\n" ;
   i = 15 ; n = i += 3 ;
   cout << "D : i = " << i << " n = " << n << "\n" ;

   i = 3 ; j = 5 ; n = i *= --j ;
   cout << "E : i = " << i << " j = " << j << " n = " << n << "\n" ;
}
```

Solution

```
A : i = 1 n = 0
B : i = 11 n = 11
C : i = 21 j = 6 n = 120
D : i = 18 n = 18
E : i = 12 j = 4 n = 12
```

Exercice 6

Énoncé

Quels résultats fournira ce programme :

```
#include <iostream>
using namespace std ;
main()
{
   int n=10, p=5, q=10, r ;

   r = n == (p = q) ;
   cout << "A : n = " << n << "  p = " << p << " q = " << q
                      << " r = " << r << "\n" ;

   n = p = q = 5 ;
   n += p += q ;
   cout << "B : n = " << n << " p = " << p << " q = " << q << "\n" ;

   q = n < p ? n++ : p++ ;
   cout << "C : n = " << n << " p = " << p << " q = " << q << "\n" ;

   q = n > p ? n++ : p++ ;
   cout << "D : n = " << n << " p = " << p << " q = " << q << "\n" ;
}
```

Solution

```
A : n = 10  p = 10  q = 10  r = 1
B : n = 15  p = 10  q = 5
C : n = 15  p = 11  q = 10
D : n = 16  p = 11  q = 15
```

Exercice 7

Énoncé

Quels résultats fournira ce programme :

```
#include <iostream>
using namespace std ;
main()
{  int n, p, q ;

   n = 5 ; p = 2 ;                                  /* cas 1 */
   q = n++ >p || p++ != 3 ;
   cout << "A : n = " << n << " p = " << p << " q = " << q << "\n" ;

   n = 5 ; p = 2 ;                                  /* cas 2 */
   q = n++<p || p++ != 3 ;
   cout << "B : n = " << n << " p = " << p << " q = " << q << "\n" ;

   n = 5 ; p = 2 ;                                  /* cas 3 */
   q = ++n == 3 && ++p == 3 ;
   cout << "C : n = " << n << " p = " << p << " q = " << q << "\n" ;

   n = 5 ; p = 2 ;                                  /* cas 4 */
   q = ++n == 6 && ++p == 3 ;
   cout << "D : n = " << n << " p = " << p << " q = " << q << "\n" ;
}
```

Solution

Il ne faut pas oublier que les opérateurs && et || n'évaluent leur second opérande que lorsque cela est nécessaire. Ainsi, ici, il n'est pas évalué dans les cas 1 et 3. Voici les résultats fournis par ce programme :

```
A : n = 6  p = 2  q = 1
B : n = 6  p = 3  q = 1
C : n = 6  p = 2  q = 0
D : n = 6  p = 3  q = 1
```