Chapitre 2 : Eléments de base

Exercice 2.1 Identificateurs C++

Pour chacun des cas ci-dessous, indiquez s'il s'agit d'un identificateur C++ légal ou non. Justifiez votre réponse si celle-ci est "Non".

		Oui / Non	**Justification**
1)	007	……	……...
2)	james_bond_007	……	……...
3)	james_bond__007	……	……...
4)	james bond	……	……...
5)	sOs	……	……...
6)	SOS	……	……...
7)	_007	……	……...
8)	__007	……	……...
9)	_007_	……	……...
10)	bond-007	……	……...
11)	tom&jerry	……	……...
12)	int	……	……...
13)	INT	……	……...
14)	André	……	……...
15)	_	……	……...

Solution exercice 2.1

		Oui / Non	Justification
1)	007	Non	Un identificateur ne peut pas commencer par un chiffre
2)	james_bond_007	Oui	
3)	james_bond__007	Oui	Plusieurs _ peuvent se suivre
4)	james bond	Non	Pas d'espace dans un identificateur
5)	sOs	Oui	
6)	SOS	Oui	Vu que C++ tient compte de la casse, 5) est un identificateur différent de 6)
7)	_007	Oui	Un identificateur peut commencer par _
8)	__007	Oui	
9)	_007_	Oui	
10)	bond-007	Non	Le caractère '-' n'est pas autorisé
11)	tom&jerry	Non	Le caractère '&' n'est pas autorisé
12)	int	Non	Mot réservé
13)	INT	Oui	Déconseillé toutefois !
14)	André	Non	Les lettres accentuées ne sont pas autorisées
15)	_	Oui	… mais pas des plus parlants (!)

Exercice 2.2 Déclarations de variables

a) Que vaut la variable *n* au terme de la séquence d'instructions suivante :

```
int n = 1;
n = 1 - 2 * n;
n = n + 1;
```

b) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n = 1;
n = n + 1;
int n = 1 - 2 * n;
```

c) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n = 1, p = 2;
n = (n + 1) * (n - P);
```

d) Expliquez pourquoi la séquence d'instructions suivante n'est pas correcte :

```
int n, m = 0;
n = 2 * n - 1;
m = n + 1;
```

Solution exercice 2.2

a) 0

b) n est déclaré deux fois

c) P n'est pas déclaré
(p et P sont 2 variables distinctes car C++ est sensible à la casse)

d) n n'est pas initialisé (donc indéfini)
m est initialisé inutilement

Exercice 2.3 Types de base

1) Dressez la liste exhaustive de tous les types entiers supportés par C++
2) Idem pour les réels
3) Idem pour les caractères (à code non étendu)
4) Quel type permet en C++ de représenter des grandeurs booléennes ?
5) Quel type permet en C++ de représenter une absence de type ou un type neutre ?
6) Le type `int` est-il signé ou non signé par défaut ?
7) Le type `char` est-il signé ou non signé par défaut ?
8) Le domaine de définition des entiers est-il fixé par la norme ou dépend-il de l'environnement utilisé ?
9) Qu'ont de particulier les identificateurs des types de base en C++ ?

Solution exercice 2.3

1) [signed | unsigned] short [int]
   [signed | unsigned] int
   [signed | unsigned] long [int]
   [signed | unsigned] long long [int]
2) float
   [long] double
3) [signed | unsigned] char
4) bool
5) void
6) Signé
7) Dépend du compilateur utilisé
8) Dépend de l'environnement utilisé
9) Ce sont tous des mots réservés

Exercice 2.4 Taille et domaine de définition des types entiers

Ecrire un programme C++ qui détermine / affiche à l'écran la taille en bits ainsi que l'intervalle des valeurs possibles des types *signed char*, *short*, *int*, *long* et *long long*.

Les résultats sont à présenter comme suit (ici pour le type *signed char*) :

```
signed char (8 bits) : -128 .. 127
```

Solution exercice 2.4

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

int main() {

   cout << "signed char (" << numeric_limits<signed char>::digits + 1 << " bits) : "
        << (int) numeric_limits<signed char>::lowest() << " .. "
        << (int) numeric_limits<signed char>::max() << endl;

   cout << "short (" << numeric_limits<short>::digits + 1 << " bits) : "
        << numeric_limits<short>::lowest() << " .. "
        << numeric_limits<short>::max() << endl;

   cout << "int (" << numeric_limits<int>::digits + 1 << " bits) : "
        << numeric_limits<int>::lowest() << " .. "
        << numeric_limits<int>::max() << endl;

   cout << "long (" << numeric_limits<long>::digits + 1 << " bits) : "
        << numeric_limits<long>::lowest() << " .. "
        << numeric_limits<long>::max() << endl;

   cout << "long long (" << numeric_limits<long long>::digits + 1 << " bits) : "
        << numeric_limits<long long>::lowest() << " .. "
        << numeric_limits<long long>::max() << endl;

   return EXIT_SUCCESS;
}

// signed char (8 bits) : -128 .. 127
// short (16 bits) : -32768 .. 32767
// int (32 bits) : -2147483648 .. 2147483647
// long (32 bits) : -2147483648 .. 2147483647
// long long (64 bits) : -9223372036854775808 .. 9223372036854775807
```

Exercice 2.5 Littéraux constants

Indiquez si les littéraux constants suivants sont corrects ou non. Dans le cas où le littéral constant est correct, indiquez son type; dans le cas contraire, expliquez pourquoi il est faux.

1) 1.5 ……...

2) 1E3 ……...

3) 12u ……...

4) 12.0u ……...

5) 1L ……...

6) 1.0L ……...

7) .5 ……...

8) 5. ……...

9) 1000000000 ……...

10) 0x33 ……...

11) 0xefg ……...

12) 0xef ……...

13) 0xEF ……...

14) 0x0.2 ……...

15) 08 ……...

16) 07 ……...

Solution exercice 2.5

1)	1.5	Juste. De type double.
2)	1E3	Juste. De type double.
3)	12u	Juste. De type unsigned int.
4)	12.0u	Faux. Un réel ne peut pas être unsigned.
5)	1L	Juste. De type long int.
6)	1.0L	Juste. De type long double.
7)	.5	Juste. De type double.
8)	5.	Juste. De type double.
9)	1000000000	Juste. Du premier des types entiers permettant de le contenir selon la hiérarchie suivante : int -> long -> long long
10)	0x33	Juste. De type int.
11)	0xefg	Faux. La lettre g ne fait pas partie des chiffres hexadécimaux.
12)	0xef	Juste. De type int.
13)	0xEF	Juste. De type int.
14)	0x0.2	Faux. Une constante réelle peut s'écrire (depuis C++11) en hexadécimal … mais pas sous la forme proposée. Doit s'écrire 0x0.2p0 (nécessite un exposant). cout << 0x0.2p0; // affiche 0.125
15)	08	Faux. Une constante octale ne peut pas comporter le chiffre 8.
16)	07	Juste. De type int.

Exercice 2.6 Evaluations d'expressions (1)

Que va afficher le programme C++ suivant ?

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

int main() {

   int i, j, k;

   i = 0; k = i++;
   cout << "A : i = " << i << " k = " << k << endl;

   i = 1; k = ++i;
   cout << "B : i = " << i << " k = " << k << endl;

   i = 2; j = 3;
   k = i++ * ++j;
   cout << "C : i = " << i << " j = " << j << " k = " << k << endl;

   i = 3; j = 4;
   k = i *= --j;
   cout << "D : i = " << i << " j = " << j << " k = " << k << endl;

   return EXIT_SUCCESS;
}
```

Solution exercice 2.6

```
A : i = 1 k = 0
B : i = 2 k = 2
C : i = 3 j = 4 k = 8
D : i = 9 j = 3 k = 9
```

Exercice 2.7 Traduction d'expressions mathématiques

Ecrire l'équivalent C++ de chacune des expressions mathématiques ci-dessous :
(r, x et y sont supposés de type double; i, j et k de type int)

a) $\frac{4\pi r^3}{3}$

b) $\sqrt{|x-y|}$

c) $\sqrt{x^2+y^2}$

d) $\cos 45°$

e) e^{-x^2}

f) $\frac{i+j+k}{3}$

Solution exercice 2.7

a)
```
4 * PI * pow(r, 3) / 3
(4 / 3.) * PI * pow(r, 3)
4 * PI * r * r * r / 3
(4 / 3) * PI * pow(r, 3)        /!\ FAUX (car 4/3 = division entière)
```

b)
```
sqrt(fabs(x - y))
sqrt(abs(x - y))                /!\ Ok si #include <cmath>, sinon faux
                                car abs de cstdlib travaille sur des int
```

c)
```
hypot(x, y)
sqrt(x * x + y * y)
```

d)
```
cos(PI / 4)                     /!\ L'angle doit être donné en radians
```

e)
```
exp(-x * x)
```

f)
```
((double) i + j + k) / 3        /!\ Division réelle et pas entière
                                Valable pour tout i, j, k (pas de débordement)
i / 3. + j / 3. + k / 3.        Ok! Pas de débordement possible
(i + j + k) / 3.                /!\ Débordement possible
```

Exercice 2.8 Evaluations d'expressions (2)

Que va afficher le programme ci-dessous ? Expliquer les résultats obtenus.

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

int main() {

   cout << fixed << setprecision(0);

   cout << "1)" << 3 * 1000 * 1000 * 1000 << endl;
   cout << "2)" << 3.0 * 1000 * 1000 * 1000 << endl;
   cout << "3)" << 100000 * 100000 * 100000.0 << endl;
   cout << "4)" << 100000.0 * 100000 * 100000 << endl;
   cout << "5)" << 1E7 + 1.0 << endl;
   cout << "6)" << 1E7f + 1.f << endl;
   cout << "7)" << 1E8 + 1.0 << endl;
   cout << "8)" << 1E8f + 1.f << endl;

   return EXIT_SUCCESS;
}
```

Solution exercice 2.8

```
1)-1294967296
2)3000000000
3)141006540800000
4)1000000000000000
5)10000001
6)10000001
7)100000001
8)100000000
```

Explications

1) Résultat faux car multiplication entière induisant un débordement
2) Résultat juste car multiplications successives réelles
3) Résultat faux car la première multiplication s'effectue sur les entiers => débordement
4) Résultat juste car multiplications successives réelles
5) Résultat juste.
6) Résultat juste.
7) Résultat juste.
8) Résultat faux dû à la précision insuffisante des calculs avec le type float

Exercice 2.9 Conversions implicites

Soient les déclarations suivantes :

```
char c = 'A';
int n = 7;
int a = -2;
unsigned b = 1;
long p = 10;
float x = 1.25f;
double z = 5.5;
```

Pour chacune des expressions suivantes, indiquez :

- combien de conversions implicites sont mises en œuvre et lesquelles
- ce que vaut l'expression et quel est le type du résultat

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

Solution exercice 2.9

Rappel Les promotions numériques : bool → int, char → int et short → int

1) 2 conversions implicites :
   c est tout d'abord converti en int, avant d'être ajouté à n.
   Le résultat (72), de type int, est ensuite converti en long, avant d'être ajouté à p.
   Au final, on obtient la valeur 82 de type long.

2) 3 conversions implicites :
   c (de type char) est tout d'abord converti en int (promotion numérique), ce qui donne 65.
   On évalue ensuite 2 * x en convertissant 2 (de type int) en float, ce qui donne 2.5 de type float.
   Pour effectuer l'addition, on convertit la valeur entière 65 en float, avant de l'ajouter au résultat précédent (2.5).
   Au final, on obtient la valeur 67.5 de type float.

3) 2 conversions implicites :
   c est tout d'abord converti en int (promotion numérique).
   n est ensuite converti explicitement en char (cast) avant d'être reconverti, implicitement cette fois, en int.
   Au final, on obtient la valeur 72 de type int.

4) 1 conversion implicite :
   z est tout d'abord converti en float (cast), ce qui donne 5.5.
   La division entière n / 2 est ensuite effectuée; on obtient la valeur 3.
   Cette valeur (3) est ensuite convertie en float, avant d'être ajoutée à 5.5.
   Au final, on obtient la valeur 8.5 de type float.

5) 1 conversion implicite (de type ajustement de type) de int en unsigned int.
   Au final, on obtient UINT_MAX (soit 4'294'967'295).

Exercice 2.10 Evaluations d'expressions (3)

Soient les déclarations suivantes :

```
int i = 5, j = 11;
double x;
```

Que vaut la variable x dans chacun des cas ci-dessous ?

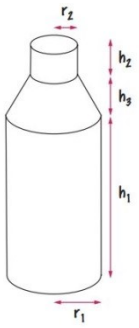
1) x = (double) j / i; ……...
2) x = double (j / i); ……...
3) x = j / i + .5; ……...
4) x = (double) j / i + .5; ……...
5) x = (int) (j + .5) / i; ……...

Solution exercice 2.10

1) x = (double) j / i; 2.2 (division réelle)
2) x = double (j / i); 2.0 (division entière)
3) x = j / i + .5; 2.5
4) x = (double) j / i + .5; 2.7 (division réelle)
5) x = (int) (j + .5) / i; 2.0 (division entière)

Exercice 2.11 Volume d'une bouteille

Comme illustré ci-dessous, la forme d'une bouteille peut s'approximer par deux cylindres, de rayons r_1 et r_2 et de hauteurs h_1 et h_2, joints par un cône tronqué de hauteur h_3.



Ecrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir (en centimètres) les cinq paramètres ci-dessus, calcule puis affiche (avec un chiffre après la virgule) la contenance en litres de la bouteille.

Solution exercice 2.11

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

int main() {

   const double PI = 3.14159,
                CM3_EN_LITRE = 1E-3; // 1 litre = 1000 cm3

   double r1, h1, // rayon [cm] et hauteur [cm] du cylindre 1
          r2, h2, // rayon [cm] et hauteur [cm] du cylindre 2
          h3;     // hauteur [cm] du tronc de cône

   // Saisies utilisateur (supposées correctes)
   cout << "Entrez le rayon du cylindre 1 [cm]      : ";
   cin >> r1;
   cout << "Entrez le rayon du cylindre 2 [cm]      : ";
   cin >> r2;
   cout << "Entrez la hauteur du cylindre 1 [cm]    : ";
   cin >> h1;
   cout << "Entrez la hauteur du cylindre 2 [cm]    : ";
   cin >> h2;
   cout << "Entrez la hauteur du tronc de cone [cm] : ";
   cin >> h3;

   // Calculs des divers volumes [cm3] et du volume total [litre]
   const double
      VOLUME_CYLINDRE_1 = PI * r1 * r1 * h1,
      VOLUME_CYLINDRE_2 = PI * r2 * r2 * h2,
      VOLUME_CONE       = PI * (r1 * r1 + r1 * r2 + r2 * r2) * h3 / 3,
      VOLUME_TOTAL      = (VOLUME_CYLINDRE_1 + VOLUME_CYLINDRE_2 + VOLUME_CONE)
                          * CM3_EN_LITRE;

   // Affichage du résultat
   cout << "\nLa contenance de la bouteille est de "
        << fixed << setprecision(1)
        << VOLUME_TOTAL << " litre"
        << (VOLUME_TOTAL < 2 ? "" : "s") << "." << endl;

   return EXIT_SUCCESS;
}

// Entrez le rayon du cylindre 1 [cm]      : 4
// Entrez le rayon du cylindre 2 [cm]      : 1
// Entrez la hauteur du cylindre 1 [cm]    : 12
// Entrez la hauteur du cylindre 2 [cm]    : 1
// Entrez la hauteur du tronc de cone [cm] : 3
//
// La contenance de la bouteille est de 0.7 litre.
```

Exercice 2.12 Conversion de mètres en miles, pieds et pouces

Ecrire un programme C++ permettant de réaliser les trois conversions d'unités suivantes :

- mètres en miles
- mètres en pieds (*feet* en anglais)
- mètres en pouces (*inches* en anglais).

Le nombre de mètres est saisi par l'utilisateur sous la forme d'un entier (de type *unsigned int*) > 0. On suppose ladite saisie correcte.

Faire en sorte que l'affichage des résultats soit absolument identique à l'exemple d'exécution donné ci-dessous (alignement vertical des "=", résultats des conversions avec 2 chiffres après la virgule).

Exemple d'exécution

```
Entrez le nombre de metres a convertir (entier > 0) : 1000

1000 [m] = 0.62 [mile]
         = 3280.84 [ft]
         = 39370.08 [inch]
```

Solution exercice 2.12

```
#include <cmath>
#include <cstdlib>
#include <iomanip>
#include <iostream>
using namespace std;

int main() {

   const double METRES_EN_MILES = 6.213711922e-4,
                METRES_EN_FT = 3.280839895,
                METRES_EN_INCH = 39.37007874;

   // Saisie utilisateur
   unsigned nb_metres;
   cout << "Entrez le nombre de metres a convertir (entier > 0) : ";
   cin >> nb_metres;

   // Affichage du résultat des diverses conversions
   const int W = (int) log10(nb_metres) + 8;

   cout << fixed << setprecision(2) << endl
      << nb_metres << " [m] = " << nb_metres * METRES_EN_MILES << " [mile]" << endl
      << setw(W) << "= " << nb_metres * METRES_EN_FT << " [ft]" << endl
      << setw(W) << "= " << nb_metres * METRES_EN_INCH << " [inch]" << endl;

   return EXIT_SUCCESS;
}
```

Rappels

- 1 pouce (inch) ≈ 2.54 cm
- 1 pied (foot) ≈ 30.48 cm
- 1 mile ≈ 1609.344 m
- 1 mile = 5'280 feet = 63'360 inches

Autre façon de faire, plus souple, mais pas encore à la portée des étudiants :

```
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <string>
using namespace std;

int main() {

   const double METRES_EN_MILES = 6.213711922e-4,
                METRES_EN_FT = 3.280839895,
                METRES_EN_INCH = 39.37007874;

   // Saisie utilisateur
   unsigned nb_metres;
   cout << "Entrez le nombre de metres a convertir (entier > 0) : ";
   cin >> nb_metres;

   // Affichage du résultat des diverses conversions
   const string TEXTE = to_string(nb_metres) + " [m] = ";
   const int W = (int) TEXTE.size();

   cout << fixed << setprecision(2) << endl
        << TEXTE << nb_metres * METRES_EN_MILES << " [mile]" << endl
        << setw(W) << "= " << nb_metres * METRES_EN_FT << " [ft]" << endl
        << setw(W) << "= " << nb_metres * METRES_EN_INCH << " [inch]" << endl;

   return EXIT_SUCCESS;
}
```

Exercice 2.13 Analyse d'un nombre entier (1)

Sans utiliser la classe *string* ni le concept de boucle (car pas encore vus en cours) mais en utilisant exclusivement les services offerts par la librairie *cmath*, écrire un programme C++ qui, après avoir demandé à l'utilisateur de saisir un nombre entier (de type *unsigned int*) > 0, affiche à l'écran combien de chiffres contient ce nombre et ce que valent le premier chiffre et le dernier chiffre du nombre.

NB On supposera ici la saisie utilisateur correcte.

Exemple d'exécution

```
Entrez un nombre entier > 0 : 148

Nombre saisi       : 148
Nombre de chiffres : 3
Premier chiffre    : 1
Dernier chiffre    : 8
```

Solution exercice 2.13

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

int main() {

   unsigned n; // nombre entier saisi par l'utilisateur

   cout << "Entrez un nombre entier > 0 : ";
   cin >> n;

   const unsigned NB_CHIFFRES = (unsigned) log10(n) + 1,
                  PREMIER_CHIFFRE = n / (unsigned) pow(10, NB_CHIFFRES - 1),
                  DERNIER_CHIFFRE = n % 10;

   cout << endl;
   cout << "Nombre saisi       : " << n << endl;
   cout << "Nombre de chiffres : " << NB_CHIFFRES << endl;
   cout << "Premier chiffre    : " << PREMIER_CHIFFRE << endl;
   cout << "Dernier chiffre    : " << DERNIER_CHIFFRE << endl;

   return EXIT_SUCCESS;
}
```