

Tâches professionnelles :	T7.2	Réaliser une maquette, un prototype logiciel/matériel
Compétences du référentiel	C4.4	Développer un module logiciel.
Savoirs - Savoir faire	S4.7	Langages de programmation C++

*Table des matières*

1	Introduction.....	2
2	Identifiants.....	2
3	Types de données fondamentaux .....	3
4	Adresse de : l'opérateur &.....	6
5	Un peu d'anglais... ..	6
6	Exercices .....	7

{ C / C++ }

# 1 INTRODUCTION

L'utilité des programmes « Hello World » figurant dans le 1<sup>er</sup> chapitre précédent est plutôt discutable. Nous avons dû écrire plusieurs lignes de code, les compiler, puis exécuter le programme résultant, pour obtenir une simple phrase écrite sur l'écran. Il aurait certainement été beaucoup plus rapide de taper nous-même la phrase de sortie.

Cependant, la programmation ne se limite pas à l'impression de textes simples sur l'écran. Pour aller un peu plus loin et devenir capable d'écrire des programmes qui exécutent des tâches utiles, nous avons besoin d'introduire le concept de **variables**.

Imaginons que je vous demande de mémoriser le nombre 5, et que je vous demande de mémoriser également le numéro 2 en même temps. Vous venez de stocker deux valeurs différentes dans votre mémoire (5 et 2). Maintenant, si je vous demande d'ajouter 1 au premier nombre que j'ai dit, vous devriez retenir les numéros 6 (soit 5 + 1) et 2 dans votre mémoire. Ensuite, nous pourrions, par exemple, soustraire ces valeurs et donc obtenir 4 comme résultat.

L'ensemble du processus décrit ci-dessus est similaire à ce qu'un ordinateur peut faire avec deux variables. Le même processus peut être exprimé en C++ avec l'ensemble des instructions suivantes :

```
1  a = 5;  
2  b = 2;  
3  a = a + 1;  
4  result = a - b;
```

Evidemment, ceci est un exemple très simple, puisque nous avons seulement utilisé deux petites valeurs entières, mais considérez que votre ordinateur peut stocker des millions de nombres comme ceux-ci en même temps et effectuer des opérations mathématiques sophistiquées avec eux.

Nous pouvons définir la **variable** comme étant une partie de la mémoire qui permet de stocker une valeur.

Chaque variable a besoin d'un nom qui l'identifie et la distingue des autres. Par exemple, dans le code précédent, les noms de variables étaient **a**, **b**, et **result**, mais nous aurions pu appeler ces variables avec des noms différents, tant qu'ils sont des identifiants C++ valides.

## 2 IDENTIFIANTS

Un **identifiant valide** est une séquence d'une ou plusieurs lettres, de chiffres ou de caractères de soulignement (underscore `_`), et qui commence toujours par une lettre (**jamais par un chiffre**).

Ils peuvent également commencer par un caractère de soulignement (`_`), mais ces identifiants sont plus souvent considérés comme réservés pour des mots clés spécifiques au compilateur ou identifiants externes, ainsi que les identifiants contenant deux caractères de soulignement successifs.

Les espaces, signes de ponctuation et les symboles ne peuvent pas faire partie d'un identifiant.

Exemples :

**toto** : OK                      **toto2** : OK                      **to2to** : OK                      **2toto** : pas OK                      **to to** : pas OK

**maVariable** : OK                      **ma\_Variable** : OK

**\_maVariable** : pas recommandé, sauf cas particulier

**\_\_maVariable** : pas recommandé, sauf cas particulier

**\$maVariable** : pas recommandé, sauf cas particulier

C++ utilise un certain nombre de mots-clés pour identifier des opérations et des descriptions de données.

Par conséquent, les identifiants créés par un programmeur ne peuvent pas correspondre à ces mots-clés.

Les mots-clés standards réservés qui ne peuvent pas être utilisés pour un programmeur pour créer des identifiants sont:

alignas, alignof, and, and\_eq, asm, auto, bitand, bitor, bool, break, case, catch, char, char16\_t, char32\_t, class, compl, const, constexpr, const\_cast, continue, decltype, default, delete, do, double, dynamic\_cast, else, enum, explicit, export, extern, false, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, noexcept, not, not\_eq, nullptr, operator, or, or\_eq, private, protected, public, register, reinterpret\_cast, return, short, signed, sizeof, static, static\_assert, static\_cast, struct, switch, template, this, thread\_local, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, wchar\_t, while, xor, xor\_eq

Des compilateurs spécifiques peuvent également avoir des mots clés réservés supplémentaires.

**Très important :** Le C++ est « **sensible à la casse** ».

Cela signifie qu'un identificateur écrit en lettres majuscules ne correspond pas à un autre avec le même nom mais écrit en lettres minuscules. Ainsi, par exemple, la variable *RESULT* n'est pas la même que la variable *result* ou la variable *Result*. Ce sont là trois identifiants différents qui désignent trois variables différentes.

### 3 TYPES DE DONNEES FONDAMENTAUX

Les valeurs des variables sont stockées (quelque part dans un endroit non précisé) dans la mémoire de l'ordinateur sous forme de zéros et de uns.

Notre programme n'a pas besoin de connaître l'emplacement exact où une variable est stockée ; il peut simplement se référer à elle par son nom, son identifiant.

Ce que le programme doit savoir par contre est le type de données stockées dans la variable.

Il n'est pas le même pour stocker un entier simple, pour stocker une lettre, ou un grand nombre à virgule flottante.

Bien qu'ils soient tous représentés par des zéros et des uns, ils ne sont pas interprétés de la même manière et dans de nombreux cas, ils ne remplissent pas la même quantité de mémoire.

Les types de données fondamentales sont les types de base mis en œuvre directement par le langage ; ils représentent les unités de stockage de base pris en charge nativement par la plupart des systèmes.

Ils peuvent principalement être classés en 4 groupes :

- **Types caractères** : Ils peuvent représenter un caractère unique, comme 'A' ou '\$'. Le type le plus fondamental est le **char**, qui est un caractère d'un octet. D'autres types sont également prévus pour les caractères plus larges.
- **Types entiers numériques** : Ils peuvent stocker une valeur de nombre entier, tels que 7 ou 1024. Ils existent dans une grande variété de tailles et peuvent être soit **signé** ou **non signé**, selon qu'ils prennent en charge les valeurs négatives ou non. Les types **int**, **short** ou **long** sont les plus fréquents.
- **Types à virgule flottante** : Ils peuvent représenter des valeurs réelles, telles que 3.14 ou 0.01, avec différents niveaux de précision, le type **float** ou **double** sont les plus fréquents.
- **Type booléen** : Le type booléen, connu en C++ comme **bool**, ne peut représenter que deux états : vrai ou faux (true ou false).

Voici la liste complète des types fondamentaux en C++ :

Groupe	Nom du Type (*)	Remarques sur la taille / précision
types caractères	<b>char</b>	Exactement un octet de taille. Au moins 8 bits.
	<b>char16_t</b>	Pas plus petit que le char. Au moins 16 bits.
	<b>char32_t</b>	Pas plus petit que char16_t. Au moins 32 bits.
	<b>wchar_t</b>	Peut représenter le plus grand jeu de caractères pris en charge.
types entiers (signé)	<b>signed char</b>	Même taille que le char. Au moins 8 bits.
	<i>signed short int</i>	Pas plus petit que le char. Au moins 16 bits.
	<i>signed int</i>	Pas plus petit que short. Au moins 16 bits.
	<i>signed long int</i>	Pas plus petit que int. Au moins 32 bits.
	<i>signed long long int</i>	Pas plus petite que long. Au moins 64 bits.
types entiers (non signé)	<b>unsigned char</b>	(Même taille que leurs homologues signés)
	<b>unsigned short int</b>	
	<b>unsigned int</b>	
	<b>unsigned long int</b>	
	<b>unsigned long long int</b>	
types à virgule flottante	<b>float</b>	32 bits
	<b>double</b>	64bits - Précision non inférieure à float
	<b>long double</b>	80 bits - Précision au moins double
type booléen	<b>bool</b>	1 bit (mais en fait 1 octet donc 8bits)

(\*) Les noms de certains types entiers peuvent être abrégés sans leurs composantes signed et int - la partie non en italique est nécessaire pour identifier le type, la partie en italique est facultative.

Le **signed short int** peut être abrégé **signed short**, **short int**, ou tout simplement **short** ; ils identifient tous le même type fondamental.

Dans chacun des groupes ci-dessus, la différence entre les types est leur taille (combien ils occupent dans la mémoire) : le premier type dans chaque groupe est le plus petit, et le dernier est le plus grand, chaque type étant au moins aussi grand que celui qui le précède dans le même groupe. Mis à part cela, les types d'un groupe ont les mêmes propriétés.

Remarquez que dans le panneau ci-dessus, mis à part le char (qui a une taille d'exactly un octet), aucun des types fondamentaux n'a une taille standard spécifiée (mais une taille minimale, tout au plus).

Cela ne signifie pas que ces types sont d'une taille indéterminée, mais qu'il n'y a pas de taille standard sur tous les compilateurs et les machines ; chaque implémentation du compilateur peut spécifier pour ces types les tailles qui correspondent au mieux à l'architecture où le programme va exécuter.

Les tailles de types dans le tableau ci-dessus sont exprimées en bits ;

Plus un type a de bits et plus il peut représenter des nombres grands ou plus précis, mais en même temps, il consomme plus de place en mémoire :

Taille	Valeurs représentables	Plages de valeurs en signé , non signé
8 bits	$2^8 = 256$	Signé : <b>-128 à 127</b> Non signé : <b>0 à 255</b>
16 bits	$2^{16} = 65\,536$	Signé : <b>-32768 à 32767</b> Non signé : <b>0 to 65535</b>
32 bits	$2^{32} = 4\,294\,967\,296$ (4 milliards)	Signé : <b>-2 147 483 648 to 2 147 483 647</b> Non signé : <b>0 to 4 294 967 295</b> Float : <b>+/- 3.4 <math>10^{+/-38}</math> (~7 chiffres)</b>
64 bits	$2^{64} = 18\,446\,744\,073\,709\,551\,616$ (18 milliards de milliards)	Signé : <b>-9 à +9 milliards de milliards</b> Non signé : <b>0 à 18 milliards de milliards</b> Double : <b>+/- 1.7 <math>10^{+/-308}</math> (~15 chiffres)</b>

Notez que la gamme de des valeurs positives est réduite d'environ moitié dans les types signés par rapport aux types non signés, en raison du fait que l'un des 16 bits est utilisé pour le signe ; ceci est une différence relativement modeste en proportion, et justifie rarement l'utilisation de types non signés.

Pour les types à virgule flottante, la taille affecte leur précision, en ayant plus ou moins de bits pour leur mantisse et pour leur exposant.

Float mini :  $\pm 1.175,494,3 \cdot 10^{-38}$

Float maxi :  $\pm 3.402,823,4 \cdot 10^{38}$

Double mini :  $\pm 2.225,073,858,507,201,4 \cdot 10^{-308}$

Double maxi :  $\pm 1.797,693,134,862,315,7 \cdot 10^{308}$

Si la taille ou la précision du type ne sont pas une préoccupation, alors **char**, **int**, et **double** sont généralement choisis pour représenter des caractères, des entiers, et les valeurs à virgule flottante, respectivement. Les autres types dans leurs groupes respectifs ne sont utilisés que dans des cas très particuliers.

Mais deux types fondamentaux supplémentaires existent : **void**, qui identifie le manque de types ; et le type **nullptr**, qui est un type spécial de pointeur. Ces deux types seront étudiés plus loin dans un chapitre à venir sur les pointeurs.

C++ prend en charge une grande variété de types basés sur les **types fondamentaux** décrits ci-dessus ; ces autres types sont connus comme les **types de données composés**, et sont l'une des principales forces du langage C++. Nous allons également les voir plus en détail dans les prochains chapitres.

**Exercice :** Saisissez et testez le programme suivant, et dites pourquoi le résultat est incohérent.

Résultat : .....

Explication : .....

.....

.....

.....

```
#include <iostream>
#include <string>

int main()
{
    short a = 42365;
    short b = 20000;
    short c;

    c = a/b;

    std::cout << "c vaut: " << c << "!\n";
}
```

**Exercice :** Saisissez et testez le programme suivant, et dites pourquoi le résultat est incohérent.

Résultat : .....

Explication : .....

.....

.....

.....

```
#include <iostream>
#include <string>

int main()
{
    short a = 40000;
    short b = 20000;
    short c;

    c = a+b;

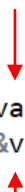
    std::cout << "c vaut: " << c << "!\n";
}
```

## 4 ADRESSE DE : L'OPERATEUR &

L'adresse physique d'une variable en mémoire peut être obtenue en faisant précéder le nom de la variable avec un signe **&** qui signifie **adresse de** (à ne pas confondre avec le & des opérations logiques que nous verrons plus tard, mais qui n'est pas placé au même endroit).

Par exemple :

```
1 // Opérateur & : adresse de
2 #include <iostream>
3
4
5 int main()
6 {
7     int var = 64;
8
9     std::cout << "La variable var vaut: " << var << std::endl;
10    std::cout << "L'adresse de var est: " << &var << std::endl;
11 }
```



Et le résultat est :

```
La variable var vaut: 64
L'adresse de var est: 0x7fbbec33fb8c
```

C'est le compilateur qui choisit les emplacements mémoire des variables, en fonction de la machine, du système d'exploitation, etc.

**Exercice :** Saisissez et testez ce programme.

## 5 UN PEU D'ANGLAIS...

Sensible à la casse	Case sensitive (qui différencie majuscules et minuscules)
Octet	Byte
Bit	Bit
Signé	Signed
Non signé	Unsigned
3 142,26	3142.26
esperluette ( & )	ampersand

## 6 EXERCICES

### 1. Vous devez saisir ce programme et voir si le résultat obtenu est conforme à celui-ci-dessous :

**Pas de panique** les lignes se ressemblent beaucoup et vous pourrez faire du copier/coller.

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

int main()
{
    cout << "Taille de char : \t" << sizeof(char) << " octet \t" << sizeof(char)*8 << " bits \n";
    cout << "Taille de wchar_t : \t" << sizeof(wchar_t) << " octets \t" << sizeof(wchar_t)*8 << " bits \n\n";

    cout << "Taille de short int : \t" << sizeof(short int) << " octets \t" << sizeof(short int)*8 << " bits \n";
    cout << "Taille de int : \t" << sizeof(int) << " octets \t" << sizeof(int)*8 << " bits \n";
    cout << "Taille de long int : \t" << sizeof(long int) << " octets \t" << sizeof(long int)*8 << " bits \n";
    cout << "Taille de long long int:" << sizeof(long long int) << " octets \t" << sizeof(long long int)*8 << " bits \n\n";

    cout << "Taille de float : \t" << sizeof(float) << " octets \t" << sizeof(float)*8 << " bits \n";
    cout << "Taille de double : \t" << sizeof(double) << " octets \t" << sizeof(double)*8 << " bits \n";
    cout << "Taille de long double : " << sizeof(long double) << " octets \t" << sizeof(long double)*8 << " bits \n\n";

    cout << "Taille de bool : \t" << sizeof(bool) << " octet \t" << sizeof(bool)*8 << " bits \n";

    return 0;
}
```

Résultats du programme :

Taille de char :	1 octet	8 bits
Taille de wchar_t :	4 octets	32 bits
Taille de short int :	2 octets	16 bits
Taille de int :	4 octets	32 bits
Taille de long int :	8 octets	64 bits
Taille de long long int:	8 octets	64 bits
Taille de float :	4 octets	32 bits
Taille de double :	8 octets	64 bits
Taille de long double :	16 octets	128 bits
Taille de bool :	1 octet	8 bits

Quelques explications :

- Le but est de voir quelle est la taille exacte en octets et en bits de chacun des types, sur votre ordinateur puisque la taille dépend du processeur, et du compilateur.
- L'instruction qui permet de connaître la taille d'un char est *sizeof(char)* : le résultat est en octet (byte)
- On multiplie par 8 pour avoir le résultat en bit (un octet = 8 bits)
- Dans les chaînes de caractères entre guillemets, il y a des caractères spéciaux : \t = tab et \n=fin de ligne

### 2. Vous devez concevoir un logiciel d'astronomie :

Quel type de variable pour stocker la distance Terre-Lune (en mètres) ?

.....

Quel type de variable pour stocker la distance Terre – Soleil (en mètres) ?

.....

La gravité à la surface de la terre en m/s<sup>2</sup>

.....

Le fait de savoir si un corps est une étoile ou pas ?

.....

Nom : .....	Cours C++ - Variables et Types.docx	Date : .....	7
-------------	-------------------------------------	--------------	---