1-Semaine 1 - Bases de programmation

Buts

Ce **chapitre** a pour but de se **familiariser** avec les toutes **premières notions** nécessaires à **l'écriture de programmes C++**, en offrant de pratiquer des exemples simples ne **mettant en œuvre** que des **variables** et des **opérateurs** pour **créer des expressions**.

Principaux rappels

Variables

En C++, une **valeur** à conserver est **stockée** dans une **variable**, **caractérisée** par un **type** et un **nom**.

Le type et le nom sont **définis** lors de la **déclaration**, et ne peuvent **pas** être **changés** par la suite ; la **valeur peut être définie** lors de **l'initialisation**, puis **éventuellement modifiée par la suite**.

Rappels de **syntaxe** :

Expression	Commentaire	Exemple
type nom;	déclaration d'une variable de nom nom et de type type	int i;
type nom(valeur);	déclaration et initialisation	double x(3.14);
nom = expression ;	affectation du résultat de l'expression à la variable nom	i = j+3;

Types élémentaires :

- **int** : pour les nombres entiers,
- **double** : pour les nombres réels,
- **char** : pour les caractères (un seul caractère, pas un mot entier !),
- **bool** : pour les valeurs booléennes (vrai/faux).

[C++11] Lorsque dans une initialisation le **compilateur** peut **déduire** le **type** de la **valeur** fournie, on peut **utiliser** le mot clé **`auto`** .
Exemple : `auto x(3.14);`

Nous **recommandons** cependant au programmeur de **toujours expliciter les types**.

Lors d'une déclaration, le mot clé **`const`** indique au compilateur qu'une **donnée ne sera pas modifiée** par le programme au travers de ce nom.
[C++11] Le mot clé **`constexpr`**, plus contraignant, indique au compilateur qu'une **donnée ne changera pas du tout** et que cette **valeur** peut être **calculée** au **moment** de la **compilation** (i.e. elle ne dépend pas de ce qu'il va se passer lors de l'exécution du programme).

Exemples :

```
constexpr double x(3.14);
const int i(j+3); /* "const" car le compilateur ne connaît pas forcément *
                   * la valeur de j.                    */
```

Opérateurs et expressions

Voici les **principaux opérateurs** de C++ (les opérateurs dont la description est suivie d'une étoile (*) n'ont qu'un seul opérande) :

Opérateurs arithmétiques		Opérateurs de comparaison		Opérateurs logiques	
*	multiplication	==	teste l'identité	&& ou and	et
/	division	!=	non égalité	|| ou or	ou
%	modulo	<	inférieur	! ou not	négation (*)
+	addition	>	supérieur		
-	soustraction	<=	inférieur ou égal		
++	incrément (*)	>=	supérieur ou égal		
--	décrément (*)				

Priorités des opérateurs (par ordre décroissant ; tous les opérateurs d'un même groupe sont de priorité égale) :

! ++ --	* / %	+ -	< <= > >=	== !=	&&	||

Une expression combine différentes valeurs et/ou variables à l'aide d'opérateurs.

Commentaires

Il existe en C++ **deux façons d'écrire des commentaires** dans le code :

```
// Commentaire jusqu'à la fin de la ligne
/* commentaire sur
   plusieurs lignes */
```

Pas à pas : mon premier programme C++

Le **but** de cet **exercice** est **d'apprendre** pas à pas à **éditer** puis **compiler** un **programme**, et ensuite **corriger** les erreurs de compilation.
Les **codes sources** des programmes C++ sont en fait de **simples fichiers textes**. **N'importe quel éditeur de texte** permet donc d'écrire un programme en C++. On peut aussi **utiliser** des environnement plus évolués (« **environnement de développement intégré** », EDI ou **IDE** en anglais pour « Integrated Development Environment ») tels que par **exemple Code::Blocks**, environnement de développement libre et gratuit disponible pour toutes les plates-formes (Linux, MacOs, etc.) ; mais n'importe quel logiciel permettant d'éditer un fichier peut convenir ici.

TP : Installation d'un environnement de développement intégré

1. Installation des outils GNU pour le développement en C++ : **`build-essential`**.
2. Installation de l'IDE eclipse
3. Installation de Visual Code Studio [https://code.visualstudio.com/]
    I. Suivre le tutorial jusqu'a <<RUN helloworld.cpp>> [https://code.visualstudio.com/docs/cpp/config-linux]
4. Installation de l'éditeur de texte Atom avec les les packages suivants :
    I. file-icons
    II. script
    III. terminal-tab
    IV. atom-beautify (le paquet uncrustify est nécessaire : `sudo apt install uncrustify`)
5. IDE en ligne : https://replit.com/ [https://replit.com/]

snippets pour atom

```
'.source.cpp':
    'Minimal C++ program':
        'prefix': 'cmin'
        'body': """
            #include <iostream>

            using namespace std;

            int main(int argc, char *argv[])
            {
                $1

                return 0;
            }
        """




    'En-tête C++ program':
        'prefix': 'hmin'
        'body': """
            #ifndef Maclasse_H
            #define Maclasse_H
            class Maclasse{

            public:
              Maclasse();
              $2


            private:

            };
            #endif
        """
```

Étape 1 : éditer un programme

À l'aide d'un éditeur de texte, ouvrir le **nouveau fichier** (c.-à-d. créer) **salut.cc** . Saisir ensuite le petit programme suivant :

```
#include <iostream>
using namespace std;
int main()
{
cout << "Bonjour tout le monde !" << endl;
return 0;
}
```

Penser à sauvegarder le programme.

Il est inutile pour l'instant de chercher à comprendre toutes les lignes écrites, cela viendra au fur et à mesure des progrès en C++ (de plus, une explication complémentaire est donnée en fin d'exercice). Le but ici est vraiment d'apprendre à créer des programmes, pas encore de tout comprendre… Patience !

Étape 2 : compiler

Avant d'être **exécuté**, un programme C++ doit être **compilé** : le fichier texte contenant le **code source** (que l'on vient d'écrire) doit être **traduit** en **langage machine**. On effectue cette opération à l'aide d'une **commande de compilation**, qui peut par exemple être tapée dans une fenêtre de commande.

Par exemple, pour une compilation dans une fenêtre de commande avec le compilateur Gnu, on aura :

```
g++ salut.cc -o salut
```

Ici, g++ est en fait un nom de compilateur C++ ; se rapporter à la documentation de son environnement pour savoir si l'on dispose d'un tel compilateur et comment il s'appelle.

Attendre jusqu'à ce qu'un message indiquant la fin de la compilation apparaisse.

Étape 3 : exécuter

Lorsque la compilation s'est déroulée sans problème, on peut **tester le programme** : en utilisant une fenêtre de commande (ou « terminal »), **se déplacer** si besoin dans le **répertoire** où se trouve le **programme** et l'**exécuter en tapant** :

```
$> ./salut
```

Le message « `Bonjour tout le monde !` » doit alors apparaître.

Étape 4 : erreurs de compilation

On va maintenant s'intéresser à la correction des erreurs d'un programme, du moins celles que le compilateur est capable de détecter.

Il **existe** en fait **divers types d'erreurs** :

- **Les erreurs de syntaxe** : le **programme** est **mal écrit**, avec comme conséquence que le **compilateur n'arrive pas à le comprendre**.

Ces erreurs sont **relativement faciles à trouver**, car le **compilateur signale toujours le problème**, en **indiquant** souvent **l'endroit de l'erreur**, parfois quelques lignes après.

- **Les erreurs d'exécution** : la **syntaxe** du programme est **correcte** (le compilateur le compile sans erreur), mais **ce que fait** finalement le **programme est erroné** (e.g. une division par zéro se produit, une variable n'a pas été initialisée correctement, . . .).

Ce type d'**erreurs** ne se **détecte** que lors des **tests** du programme, soit **par un arrêt intempestif** de celui-ci (cas de la division par zéro), soit par la **production de résultats erronés** (cas de la mauvaise initialisation). Ces **erreurs** sont **plus difficiles à trouver** car elles ne sont pas forcément toujours visibles !

- **Les erreurs de conception** : **l'algorithme** choisi (la « recette ») **ne fait pas** ce que l'on croit (**ce qu'il devrait**). **C'est** plus la méthode globale, voire même **l'approche du problème**, qui est **erronée**.

Nous ne nous **intéresserons** ici qu'aux **erreurs de syntaxe**. De telles erreurs sont **fréquemment commises** ; c'est normal ! Il faut donc **apprendre** à **connaître** leurs **origines** et **comprendre** le sens des **messages** du **compilateur**, qui peuvent être très utiles **pour corriger** rapidement un programme.

Pour commencer, **introduire** volontairement une **erreur de syntaxe** dans le programme (**supprimer** par exemple la dernière **accolade ou** le dernier **point-virgule**) et **relancer la compilation**.

On devrait **obtenir** un **message indiquant le fichier dans lequel se trouve l'erreur** (ici salut.cc), la **ligne concernée**, ainsi que **le type d'erreur**. Par exemple, **s'il manque la dernière accolade**, le **compilateur** devrait **indiquer** :

```
salut.cc:6: parse error at end of input
```

Corriger l'erreur, et relancer la compilation.

C'est typiquement **l'approche** qu'il faut avoir lorsque des erreurs de syntaxe sont signalées par le compilateur : **identifier l'erreur, la corriger puis recompiler**.

Un **conseil** : toujours **commencer par la première erreur** signalée, car une erreur de syntaxe dans le code empêche le plus souvent l'interprétation de bon nombre d'instructions qui la suivent, provoquant du même coup l'affichage d'erreurs qui n'en sont en fait pas !

Étape 5 : utilisation de variables

ICI AVEC LES CNAM 2023/2024

On est maintenant prêt pour **compliquer** un peu ce premier **programme** et **utiliser** une première **variable**. L'idée est d'**afficher** la **température** du jour après le message de salutation.
La première chose à faire est donc de **prévoir** une **place** pour **stocker** cette **température**.
Cela se fait en **déclarant** une **variable**. **De quel type ?** Nous ne connaissons pour l'instant que quatre types élémentaires : `int` pour les entiers, `double` pour les réels, `char` pour les caractères et bool pour les valeurs de vérité. Le type **double** semble donc le plus **approprié** pour stocker la température :

```
#include <iostream>
using namespace std;
int main()
{
double temperature;
cout << "Bonjour tout le monde !" << endl;
return 0;
}
```

Nous **déclarons** cette **variable** dans le **`main`** car **c'est là** qu'elle va être **utilisée**. Il est toujours recommandé de **déclarer** les **variables** au **plus près de leur utilisation**, jamais à un niveau plus haut (et surtout pas hors du main). Cela rend le code plus sûr et plus facilement maintenable.

La ligne de déclaration de la variable peut également servir à lui **donner une valeur initiale** (on parle d'initialisation), ce qui est **fortement recommandé** :

```
#include <iostream>
using namespace std;
int main()
{
double temperature(19.5);
cout << "Bonjour tout le monde !" << endl;
return 0;
}
```

Une alternative (**utilisée en C**), **moins recommandée (en C++)**, consiste à **utiliser** une **affectation** :

```
#include <iostream>
using namespace std;
int main()
{
double temperature = 19.5;
cout << "Bonjour tout le monde !" << endl;
return 0;
}
```

On peut ensuite **afficher** la **valeur** de cette **variable** :

```
#include <iostream>
using namespace std;
int main()
{
double temperature(19.5);
cout << "Bonjour tout le monde !" << endl;
cout << "Il fait " << temperature << " degrés." << endl;
return 0;
}
```

Voici pour finir une **version un peu plus évoluée**, laissée pour étude (essayer de la comprendre avant de lire la suite) :

```
#include <iostream>
using namespace std;
int main()
{
cout << "Bonjour tout le monde !" << endl;
cout << "Quelle temperature fait-il ?" << endl;
double temperature;
cin >> temperature;
cout << temperature << " !" << endl;
cout << "Et moi qui pensait qu'il faisait "
<< temperature + 4.5 << endl;
return 0;
}
```

Complément : structure du programme d'exemple

Plusieurs notions présentées ici seront détaillées, au fur et à mesure de l'apprentissage. Il ne faut donc pas s'acharner maintenant à essayer de comprendre en profondeur tous les détails, mais plutôt se focaliser sur l'idée globale de chaque instruction.

1. La **première ligne** (**`#include <iostream>`**) ordonne au compilateur de lire le catalogue de la **bibliothèque `iostream`** avant de compiler le programme. Ce catalogue **contient** les **commandes** (et variables) disponibles pour **réaliser** des **opérations** de **lecture/écriture**.
Dans le cas présent, il s'agit des définitions de **cout** (sortie écran) et **cin** (entrée clavier), de la constante **endl** (retour à la ligne) et du fonctionnement des **opérateurs « (affichage) et » (lecture)**. Ces notions devraient être explicitées lors de l'étude des fichiers et des « flots ».

2. La **seconde ligne** est nécessaire pour **indiquer** que l'on veut **utiliser** les fonctionnalités de l'« **espace de noms** » (namespace) **standard** (std). Cela **permet** par exemple **d'écrire directement** « **`cout`** », **au lieu** de « **`std::cout`** ».

3. La ligne de «**`main()`** » **indique** le **début** du **programme**. Les instructions comprises entre l'accolade ouvrante « { » et l'accolade fermante associée « } » **constituent** la **définition** du **programme principal** (la fonction main), c.-à-d. les **instructions** qui seront effectivement **exécutées** au **lancement** du **programme**. Ces instructions doivent se **terminer** par la commande « **`return 0;`** », indiquant par **convention** que le **programme** s'est **terminé correctement** (une **valeur différente** de **0** indique un **code d'erreur**).

4. Lorsqu'il s'agit d'écrire un programme C++, il est donc **conseillé** de **commencer** par ce que l'on peut appeler la « **coquille vide**[1)] », qui sont ces quelques **instructions** presque **systématiquement présentes** dans tous les **programmes C++** :

```
#include <iostream>
using namespace std;
int main()
{
return 0;
}
```

5. Pour en revenir à l'exercice, **la première ligne du « main() » affiche « Bonjour tout le monde ! » à l'écran**, représenté ici par `cout` , suivi d'un retour à la ligne (**`endl`** , **pour end of line**).

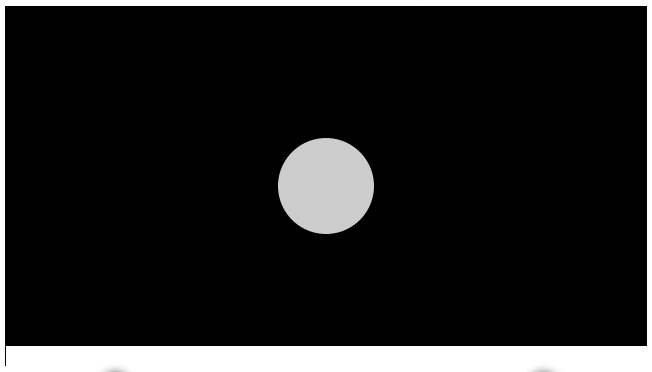
6. La **ligne suivante affiche « Quelle température fait-il ? »**.

7. On **déclare** ensuite une **variable**, de nom **temperature** et de **type double** ; puis on **lit au clavier la valeur** à **mettre** dans la **variable temperature** (il faudra donc que l'utilisateur du programme entre un nombre).

8. La ligne suivante **affiche la valeur stockée dans la variable temperature** , suivie d'un point d'exclamation.

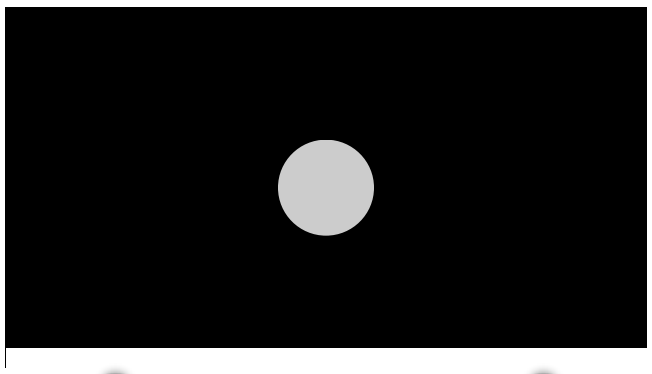
9. Les **dernières lignes affichent le message « Et moi qui pensait qu'il faisait »**, **suivi** de la **valeur** de la variable **temperature augmentée** de **4.5**. Nous avons voulu **illustrer** ici le fait qu'une même **instruction** C++ **peut s'écrire** sur **plusieurs lignes**.

Cours du MOOC



0:00 / 3:21

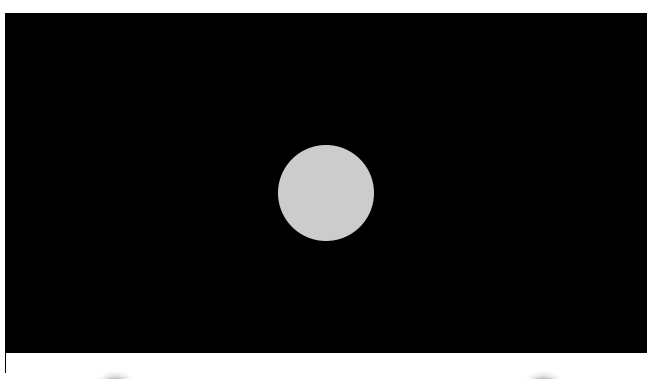
2 - 1 - Bienvenue [03-21]



0:00 / 9:56

2 - 2 - Introduction [09-55]

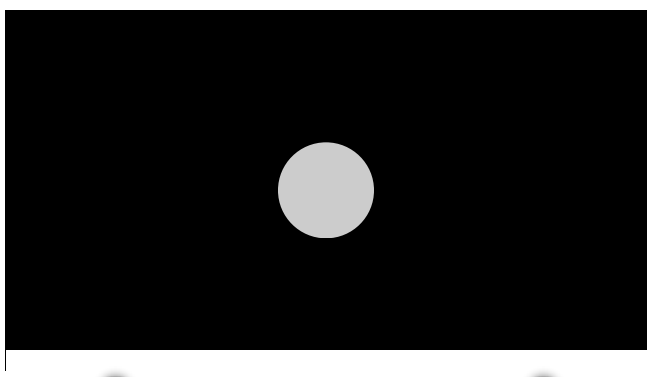
- Le cours en pdf



0:00 / 18:09

2 - 3 - Variables [18-08]

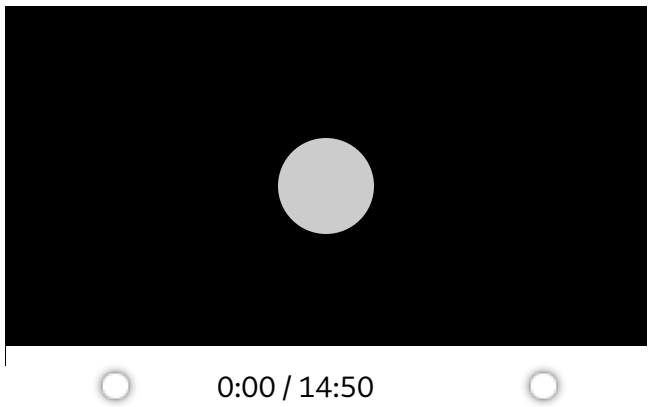
- Le cours en pdf



0:00 / 13:44

2 - 4 - Variables - lecture-écriture [13-43]

- Le cours en pdf



2 - 5 - Expressions [14-50]

- Le cours en pdf

tutoriels

- tuto

Exercices

- Énoncés

Devoirs

- Cours C++ - Devoir 1

---

1)
Voir snippets pour atom [https://wiki.simoko.eu/_export/code/btssn/savoirs/s4/s44/1-semaine_1_-_bases_de_programmation?codeblock=2]