# IFT159
# Analyse et programmation
## Chapitre 2 — Éléments de C++

Gabriel Girard

Département d'informatique
UNIVERSITÉ DE SHERBROOKE

2 septembre 2013

# Éléments de C++

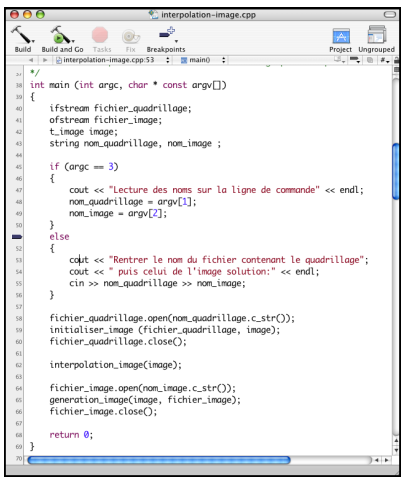


UNIVERSITÉ DE SHERBROOKE

# Qu'est-ce qu'un programme C++

## un programme est composé

- Votre code
- Du code existant (Bibliothèques)




```cpp
*/
int main (int argc, char * const argv[])
{
    ifstream fichier_quadrillage;
    ofstream fichier_image;
    t_image image;
    string nom_quadrillage, nom_image ;

    if (argc == 3)
    {
        cout << "Lecture des noms sur la ligne de commande" << endl;
        nom_quadrillage = argv[1];
        nom_image = argv[2];
    }
    else
    {
        cout << "Rentrer le nom du fichier contenant le quadrillage";
        cout << " puis celui de l'image solution:" << endl;
        cin >> nom_quadrillage >> nom_image;
    }

    fichier_quadrillage.open(nom_quadrillage.c_str());
    initialiser_image (fichier_quadrillage, image);
    fichier_quadrillage.close();

    interpolation_image(image);

    fichier_image.open(nom_image.c_str());
    generation_image(image, fichier_image);
    fichier_image.close();

    return 0;
}
```

```cpp
/*******************************************************************************
*                                                                              *
*     File:  math.h                                                            *
*                                                                              *
*     Contains: typedefs, prototypes, and macros germane to C99 floating point.*
*                                                                              *
*******************************************************************************/
#ifndef __MATH__
#define __MATH__

#include "sys/cdefs.h" /* For definition of __DARWIN_UNIX03 et al */

#if (!defined(__WANT_LONG_DOUBLE_FORMAT__))
#if defined(__APPLE_CC__) && defined(__LONG_DOUBLE_128__)
#define __WANT_LONG_DOUBLE_FORMAT__ 128
#else
#define __WANT_LONG_DOUBLE_FORMAT__ 64
#endif
#endif

#if ( __WANT_LONG_DOUBLE_FORMAT__ - 0L == 128L )
#define __LIBMLDBL_COMPAT(sym) __asm("_" __STRING(sym) "$LDBL128")
#elif ( __WANT_LONG_DOUBLE_FORMAT__ - 0L == 64L )
#define __LIBMLDBL_COMPAT(sym) /* NOTHING */
#else
#define __LIBMLDBL_COMPAT(sym) /* NOTHING */
#endif

#ifdef __cplusplus
extern "C" {
#endif
```

UNIVERSITÉ DE SHERBROOKE

# Entête

```
                /*                                  */
(1) ⟶           /* Identification du programme      */
                /*                                  */
(2) ⟶           Directive #include
(3) ⟶           using namespace std;
```

# La fonction `main`

**(4)** ⟶ `int main()`
`{`
**(5)** ⟶ *définitions/déclarations des constantes*
**(6)** ⟶ *définitions/déclarations des variables*

**(7)** ⟶ *énoncés exécutables*

**(8)** ⟶ `return 0;`
`}`

# (1) Commentaires

- Deux formes
  - //
  - /* ... */
- C'est de la documentation (sert à la maintenance)
- Utilisés éventuellement par d'autres outils
- Ignorés par le compilateur
- Normes départementales à respecter
- Documentation et normes : 30 % de la note.

```
/**
  Fichier : consommation.cpp

  Calcul de consommation d'essence d'une voiture.

  Auteur : Gerard Houdeville
  Date de derniere modification :  19 aout 2006 (derniere version)
  Date de creation :  8 janvier 1996 (creation)
  Version 1.4 : 19 aout 2006, mise aux normes, Benoit Fraikin
  Version 1.3 : 3 jan. 2004, ajout documentation, Benoit Fraikin
  Version 1.2 : 16 juil. 2003, modification legere, Gabriel Girard
  Version 1.1 : 8 jan. 1996, modification legere, Gerard Houdeville

  Entrees :
    (clavier) distance parcourue (reel positif non nul)
    (clavier) volume utilise (reel positif non nul)
  Sorties
    (ecran) consommation (reel positif)

  .... description detaillee.............
*/
```

```
/**
   \file consommation.cpp

   \brief Calcul de consommation d'essence d'une voiture.

   \author Gerard Houdeville
   \date 19 aout 2006 (derniere version)
   \date 8 janvier 1996 (creation)
   \version v1.4 : 19 aout 2006, mise a jour aux normes, Benoit Fraiki
   \version v1.3 : 3 janvier 2004, ajout de documentation, Benoit Frai
   \version v1.2 : 16 juillet 2003, modification legere, Gabriel Girar
   \version v1.1 : 8 janvier 1996, modification legere, Gerard Houdevi

   \b Entrees :
   \li (clavier) distance parcourue (reel positif non nul)
   \li (clavier) volume utilise (reel positif non nul)

   \b Sorties
   \li (ecran) consommation (reel positif)

*/
```

# (2) Directives au compilateur

- Directive `#include`

```
#include <nom_du_fichier>
#include "nom_du_fichier"
```

- Indique au compilateur d'insérer du code existant
- Exemple :

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

# (4) Fonctions

- Toute manipulation utilise une fonction en C++
- Une fonction réalise une tâche précise
- Pour IFT159, cette tâche sera décrite dans un module lors de la conception
- Tout programme possède une fonction `main`

# (5-6) Définitions (constantes et variables)

- Assignent un nom aux données
- Définissent l'ensemble des valeurs que peut recevoir une variable ou une constante (**type**)
- Noms significatifs
- Noms composés de lettres, chiffres et «_»
- ATTENTION aux noms réservés
- Normes
  - Mots séparés par «_» ou majuscules ;
  - Une variable débute par une minuscule ;
  - Constantes en majuscules ;
  - Pas de lettres accentuées.

# (5) Définition des constantes

**const** *type NOM_CONSTANTE = valeur*;

- `const float PI = 3.1416;`
- `const int NB_MOIS = 12;`

# (6) Définition des variables

*type liste_d'identificateurs*;

- `int jour, mois, annee;`
- `char initiale_prenom;`
- `float abcisse,ordonnee;`
- `int nbJour, nbHeure, nbSeconde;`

```
int main()
{
  // Declaration des constantes
  const float LITRE_A_GALLON = 1/4.5;     //constante de conversion
  const float KILOMETRE_A_MILLE = 1/1.6; //constante de conversion

  // Declaration des variables locales
  float distance_kms ;                    //distance parcourue en kms
  float volume_litres ;                   //quantite d'essence en litres
  float distance_milles ;                 //distance parcourue en milles
  float volume_gallons ;                  //quantite d'essence en gallons
  float consommation ;                    //consommation d'essence

  ...

}
```

# (7) Énoncés exécutables

- Énoncé d'affectation
- Énoncé de lecture
- Énoncé d'écriture

# Énoncé d'affectation

Permet de modifier la valeur d'une variable et d'entreprendre certaines actions :

*identificateur_variable* = *expression_mathématique*;

- `compteur = 0;`
- `total = total + (prix * taux_taxe);`
- `somme = somme + 1;`

# Énoncé de lecture

- `cin » note;`
- `cin » mois » jour » annee;`
- Opérateur : »
- Médium : `cin`

# Énoncé d'écriture

- `cout << note;`
- ```
  cout << "Le nombre d'etudiants est "
       << nbr_etu << endl;
  ```
- ```
  cout << "Bienvenue " << endl
       << endl << "au cours";
  ```
- Opérateur : «
- Médium : `cout`

```
// 1. lecture distance et quantite sur entree (clavier)
cout << endl;
cout << "Donner la distance parcourue en kilometres: " ;
cin >> distance_kms ;
cout << "Donner la quantite d'essence en litres: " ;
cin >> volume_litres ;

// 2. Conversions et calcul de la consommation
//   2.1 Conversion de la distance en milles
distance_milles = distance_kms * KILOMETRE_A_MILLE ;
//   2.2 Conversion du volume en gallons
volume_gallons = volume_litres * LITRE_A_GALLON ;
//   2.3 Calcul de la consommation en milles/gallons
consommation = distance_milles / volume_gallons ;

// 3. Affichage de la consommation
cout << endl << "La consommation en milles/gallon est : " ;
cout << consommation << endl ;

// Valeur de retour
return 0;
}
```

# Abstraction

- Simplification ou généralisation d'un concept ou d'un objet
- Abstraction procédurale
- Abstraction de données (types abstraits)
  - Types de données de base
  - Types de données pré-définies (bibliothèques)
  - Vos types de données (classes)

# Types de données de base : **float**

- Abstraction de l'ensemble $\mathbb{Q}$ (rationnels – réels par abus de langage) compris entre deux bornes MIN et MAX
- Opérations : $+, -, *$ et $/$.
- `const float PI = 3.1416;`
- `const float TEST = 2.1e-3;`
- `float result;`
- `result = PI * TEST / 0.04;`

# ATTENTION aux float

```
Valeur initiale du float = 100000000
Valeur incrémentée de 1 = 100000000
Valeur incrémentée de 2 = 100000000
Valeur incrémentée de 3 = 100000000
Valeur incrémentée de 4 = 100000000
Valeur incrémentée de 5 = 100000008
Valeur incrémentée de 6 = 100000008
Valeur incrémentée de 7 = 100000008
Valeur incrémentée de 8 = 100000008
Valeur incrémentée de 9 = 100000008
```

# Les `float` double précision : `double`

```
Valeur initiale du double= 100000000
Valeur incrémentée de 1 = 100000001
Valeur incrémentée de 2 = 100000002
Valeur incrémentée de 3 = 100000003
Valeur incrémentée de 4 = 100000004
Valeur incrémentée de 5 = 100000005
Valeur incrémentée de 6 = 100000006
Valeur incrémentée de 7 = 100000007
Valeur incrémentée de 8 = 100000008
Valeur incrémentée de 9 = 100000009
Valeur incrémentée de 10 = 100000010
```

# ATTENTION aux `double`

```
(double) 3.0 * .2 ---> 0.60000000000000008882
(double) 2.0 * .3 ---> 0.59999999999999999778
```

# Types de données de base : **int**

- Abstraction de l'ensemble $\mathbb{Z}$ (entiers) entre MIN et MAX
- Opérations : $+, -, *, /$ et %
- `const int TEST1 = 4;`
- `int test2, test3, test4;`
- `test2 = TEST1 * 2 + 5 / 6 % 2`

# Types de données de base : **char**

- Ensemble de symboles incluant les lettres minuscules et majuscules, les chiffres et des symboles spéciaux.
- Opérations : ? ? ?
- `const char INITIALE = 'p';`
- `char car1;`
- `cin >> car1;`
- ```
  cout << "La lettre initiale est "
       << INITIALE;
  ```
  ```

# Types de données de base : **bool**

- Valeurs de vérité : vrai (*true*) ou faux (*false*)
- Opérations : `&&`, `||` et !
- Le nom de la donnée doit contenir un verbe
- `bool est_un_echec, continuer;`
- `est_un_echec = true;`
- `continuer =!est_un_echec && (x > 0);`

# Types de données ajoutés : **string**

- Fourni dans une bibliothèque
- `#include <string>`
- Opérations : ??
- `const string PRENOM = "Arthur";`
- `string nom;`
- `cin >> nom;`
- `cout << PRENOM << nom;`

# Types d'erreurs

- Erreurs de syntaxe : la plus simple à corriger
- Erreurs à l'édition des liens
- Erreurs à l'exécution
- Erreurs de logique : la plus compliquée car peut remonter très haut dans le développement (conception ou analyse)

# Exemple

```
#include <iostream>

using namespace std;

int main()
{
  int valeur;

  cout << "Entrer un entier:" << endl;
  cin >> valeur;
  cout << "La valeur entree est "
       << valeur << endl;
  return 0;
}
```

# Exemples d'erreurs

- Mariner 1 (1962) - s'écrase dans l'Atlantique au lieu d'aller sur mars dû à un opérateur de négation manquant dans le système de guidage écrite en Fortran.
  Coût : 10 millions
- F-18 - écrasement causé par un condition d'exception manquée (pas de else dans un if) car la condition n'était jamais supposée être fausse.
- F-16 : avion vire sur le dos lorsqu'il traverse l'équateur à cause d'un opérateur «-» manquant pour indiquer la latitude sud.

# Exemples d'erreurs

- Sonde vers Vénus : mise en orbite raté à cause d'une mauvaise syntaxe dans la boucle («.» au lieu de «,») : `DO 100 I=1.10` (plutôt que "1,10") Cela provoquait une affectation (la variable DO100I = 1.10 plutôt qu'une boucle)
- Une dame de 104 ans a reçu une invitation pour entrer en maternelle
- Un homme de 101 ans a vu son assurance tripler car il a été classé comme adolescent (moins de 20 ans).

# Exemples d'erreurs

- Ariane 5 :
  Débordement de capacité lors d'une conversion d'un float vers un int.
  Le coût du développement de Ariane 5 était de $ 7 milliards et la perte lors de l'explosion était de $ 500 millions.

# Conclusion

- Bjarne Stroustrup
  "There are only two kinds of programming languages : those people always bitch about and those nobody uses."
- Loi de Mosher sur le développement de logiciel
  "Don't worry if it doesn't work right. If everything did, you'd be out of a job."
- Bob Gray
  "Writing in C or C++ is like running a chain saw with all the safety guards removed."