Chapitre 1 : syntaxe, compilation

Master 2 IFMA – Programmation en C++

Université Pierre et Marie Curie

Automne 2017

Qu'est-ce que le C++ ?

- langage de programmation ;
  - compilé (comme le C) ;
  - de niveau intermédiaire (comme le C) ;
  - programmation orientée objet ;
  - programmation générique.
- libre ;
- nombreux compilateurs (pour nous, g++).

- 1972 : conception du langage C, "ancêtre" du C++ ;
- 1983 : conception du langage C++ ;
- 1998 : première normalisation du C++ ;
- 2011 : dernière modification majeure du standard du C++ ;
- 2017 : prochaine modification du standard.

Plan

Fonctions

Programme en C++ $\simeq$ enchaînement de *fonctions*.

```
type_retour F(args)
{
   // Code de la fonction
}
```

- `F` : nom de la fonction
- `args` : arguments
- `type_retour` : type de la valeur renvoyée par `F`.

Fonctions

Les fonctions

- Effectuent des instructions, et renvoient une valeur ;
- Ne peuvent pas être imbriquées ;
- Mot-clé `return` : on quitte la fonction, en précisant la valeur renvoyée.

Déclaration d'une fonction :

```
type_retour F(args);
```

Cette ligne appelée le *prototype* de la fonction `F`.

Programme exécutable : présence d'une fonction `main`, la seule à être exécutée.

Types et variables

- *Variable* : zone réservée en mémoire pour stocker une valeur ;
- La mémoire se compose de suites (finies !) de 0 ou de 1 (*bits*) ;
- *Type* : façon dont la valeur doit être interprétée.
- *Déclaration* d'une variable `nom_var` de type `type_var` :

```
type_var nom_var ;
```

- Une variable est *locale* au bloc où elle a été déclarée.

Types : entiers

Entiers

- peuvent être représentés exactement :
  n bits $= \{0,1\}^n \simeq \{0,\ldots,2^n-1\}$ (ou $\{-2^{n-1},\ldots,2^{n-1}-1\}$ (écriture binaire) ;
- arithmétique modulo 2^n ;
- différents types (par nombre de bits croissant) : `char`, `short`, `int`, `long` et `long long` ;
- existent tous en version `signed` (entiers positifs ou négatifs) et `unsigned` (tous positifs) ;
- `signed` par défaut ;
- `char` correspond aussi aux caractères (code ASCII).

Types : nombres à virgule flottante

- Impossible de représenter exactement tous les réels ;
- Écriture scientifique en base 2 : $\pm(1+x)\times 2^n$, $x \in [0,1[$ et $n \in \mathbb{Z}$;
- Par ex., $5 = 1.25 \times 2^2$, $0.1 = 1.6 \times 2^{-4}$, $\pi = 1.570796... \times 2$.
  En binaire, $1.25 = 1.01$, $1.6 = 1.100110011...$,
  $1.570796 = 1.1001001000011...$
- On définit un flottant par :
  - son signe (1 bit) ;
  - la valeur (entière) de n ;
  - le (début du) développement binaire de x.
- Différents types flottants (du moins au plus précis) `float`, `double`, `long double`
- Erreurs d'arrondi (pour les `double`, $\simeq 10^{-16}$ en valeur relative).

Types : booléens

- Type `bool` : deux valeurs “vrai” et “faux”, notées `true` et `false` ;
- Par convention, toute valeur nulle peut jouer le rôle de `false` et toute valeur non-nulle peut jouer le rôle de `true`.

Boucles if

La boucle if utilise la syntaxe suivante :

```
if (condition)
{
// Lu si condition est vrai/different de 0
}
else
{
// Lu si condition est faux/vaut 0
}
```

Boucles `while`

Répeter des instructions tant que `condition` est vraie :

```
while (condition)
{
// Instructions
}
```

Si l'on veut s'assurer que les instructions sont effectuées au moins une fois, on utilise la syntaxe :

```
do
{
// Instructions
}
while (condition)
```

Boucles `for`

```
for (I1 ; I2 ; I3)
{
// Instructions
}
```

- `I1`, `I2` et `I3` sont trois instructions ;
- `I1` effectuée une seule fois, au début de la boucle ;
- `I2` est évaluée ; si elle vaut `true`, on effectue le contenu de la boucle ;
- fin de la boucle : `I3` est évalué, puis on recommence : si `I2` est vraie, on repasse dans la boucle, puis on effectue `I3`, etc.
- on pourrait écrire des boucles `for` avec le même effet que des boucles `while` ou `if` (déconseillé pour la lisibilité).

Entrées/sorties

- Bilitothèque `iostream` (io pour “in/out”, stream pour “flux”) ;
- Flux d'entrée standard `std::cin` et flux de sortie standard `std::cout` ;
- Opérateurs d'*injection* `<<` (pour un flux de sortie) et d'*extraction* `>>` (pour un flux d'entrée).
- Entrées/sorties depuis/vers un fichier : bibliothèque `fstream`.

Exemple

```
std::cin >> x ;
std::cout << x << "\n";
```

Plan

Lisibilité du code : identation

Caractères blancs (espaces, sauts de lignes) *facultatifs* : à utiliser pour rendre son code *lisible*. Exemple :

```
                                                    void
       f(
void){for(int
    i       =0;i<
10;i                                          ++)std::cout<<
                                    *
i <<"\n"                                        ;}
```

```
void f(void)
{
   for (int i=0; i<10; i++)
      std::cout << i*i << "\n";
}
```

Lisibilité du code : noms des variables/fonctions

Choisir des noms de variables/fonctions *parlants*.

```
int OOOO(int OOOO)
{
   int OOOO = 0;
   for (int OOOO=0; OOOO<OOOO; OOOO++)
      OOOO += OOOO*OOOO;
   return OOOO;
}
```

```
int somme_carres(int N)
{
   int somme = 0;
   for (int n=0; n<N; n++)
      somme += n*n;
   return somme;
}
```

Lisibilité du code : commentaires

Commentaires :

```
int f(void)
{
   // Ceci est un commentaire

   /* Ceci
   aussi */
}
```

Toujours commenter ses programmes (économie de temps lors d'une consultation ultérieure)

Plan

Compilation

Compilation : conversion d'un code source en fichier exécutable.

- *précompilation* : modifier le code source en suivant les instructions commençant par # (quasi-exclusivement pour inclure des bibliothèques avec `#include`) ;
- *compilation* à proprement parler : traduction du code C++ en langage assembleur (version lisible du langage machine).
- *assemblage* : transformation du code assembleur en langage machine. On obtient *fichier objet* d'extension `.o`.
- *édition de liens* : on rassemble les fichiers objets en un seul fichier executable.

Compilation

Deux situations possibles :

- Unique fichier `prog.cpp` : toutes les étapes d'un seul coup avec

```
g++ prog.cpp
```

Crée un ficher exécutable `a.out`, à exécuter avec

```
./a.out
```

Changer le nom par défaut :

```
g++ prog.cpp -o prog
./prog
```

Compilation

Deux situations possibles :

- Plusieurs fichiers, p. ex. `fichier1.cpp` et `fichier2.cpp`.

```
g++ -c fichier1.cpp
g++ -c fichier2.cpp
```

Crée deux fichiers objets `fichier1.o` et `fichier2.o`. On crée l'exécutable `prog` avec

```
g++ fichier1.o fichier2.o -o prog
```

Messages d'erreur

- Inévitable : on n'écrit jamais un programme correct du premier coup ;
- Le compilateur renvoie des messages d'erreur (“error”), ou des avertissement (“warning”) ;
- Erreurs : empêchent de créer un exécutable ;
- Avertissements : signalent une syntaxe correcte, mais suspecte, pouvant avoir un effet différent de celui escompté.
- Forcer l'affichage des avertissements : option `-Wall` (`W` pour “warning”, et `all` pour “tous”) de g++.
- g++ signale le numéro de la ligne où une erreur a été repérée.

Toujours lire les messages d'erreur !