PROGRAMMATION PAR OBJETS

LANGAGE C++

Notes de Cours

N. Castagné, M. Desvignes, F. Portet

Version 2013.02

PLAN DU COURS

1. C++

==> **C++ = C** + typage fort + programmation objet <==

➢ Naissance du langage C++ : 1979/1983

Bjarne Stroustrup, ATT Bell Labs.

Inspirés de SIMULA, ALGOL...
Initialement : extension du langage C, traducteur « C++ vers C »
Ajout de concepts objet (classification, encapsulation, héritage, abstraction...)

➢ Versions

1983 : première version (à l'origine *via* un traducteur de C++ vers C)
1998 : premier standard ISO, C++98
2011 : dernier standard en date (C++11)

➢ Objectifs et intérêt du langage C++ :

L'un des langages les plus utilisés aujourd'hui.

Communs aux approches objets :

Ecrire facilement de bons programmes de grande taille.
- Protection et robustesse
- Structuration, Réutilisabilité
- Lisibilité et évolutivité
- ...

Propres au C++ :

Haute performance : compilé, possibilité d'écrire « proche de la machine »
Du bit (bas-niveau) au concept (haut-niveau)
Possibilité de mixer paradigmes objet et impératif (« comme en C »).

2. LES APPORTS DU C++ AU C

OU LE C++ SANS LES OBJETS

➢ Le C++ comme une extension du C

Tout programme écrit en C compile avec un compilateur C++. Ajout au C :

- nombreux compléments : *dans cette partie*
  Rq : Certains ont été ajoutés ensuite au standard C récents.

- notions propres à l'approche objets : *partie 3 du cours*

➢ Compilateur et environnement de travail

A l'origine (1979-83), le C++ était implanté avec un traducteur « C++ vers C ».
Désormais : compilateur C++.
Nous utiliserons le compilateur **g++** de la GNU Compiler Collection.

Fichier header main.cpp pour « hello world »

```
#include <string>
#include <iostream>
main() {
    std ::string str = "Hello World" ; // un objet string !
    std::cout << str << std::endl ; // <=> printf(...)
}
```

Compilation dans le Terminal avec :

```
% g++ main.cpp -o main
```

Exécution avec :

```
% ./main
```

A part l'utilisation de g++, rien ne change dans l'environnement de travail, eg :

- Compilation en deux phases : compilation puis édition des liens
- Utilisation de fichier Makefile et de la commande make
- Debug avec le debugger gdb
- Possibilité d'utiliser la libc
- ...

COMMENTAIRES, ARGUMENT PAR DEFAUT, DECLARATION DE VARIABLES

➢ Commentaire de fin de ligne avec //

```
// Ceci est un commentaire
```

➢ Arguments par défaut

```
int f(int a, float x=3.14159)
// si x est omis lors de l'appel, x vaudra 3.14159
```

Rq 1 : les valeurs par défaut ne sont possibles que si les paramètres les plus à droite ont une valeur par défaut

```
int f(int a=9, float x) // Erreur
```

Rq 2 : les arguments de fonctions sont empilés (Stack) de la droite vers la gauche (C et Fortran) après évaluation et dépilés par la fonction appelée.

➢ Déclaration libre des variables et variables de boucle

Une variable peut être déclarée n'importe où dans le programme.

Portée d'une variable : une variable ne peut être utilisée qu'entre la ligne où elle est déclarée et la fin du bloc (ou du fichier) dans lequel elle est déclarée.

Variables de boucle: une variable peut être déclarée dans une boucle. Elle existe alors jusqu'à la fin de la boucle.

```
main() {
    int i;      // Création de i
    cin >> i;  // Lecture de i;
    j = 9;      // ERREUR : j n'existe pas encore
    int j = 12;    // Création de j
    for (int k=0; k<i; k++) {  // Création de k
      int localVar = k*10*j ;// Création de localVar.
      cout << localVar;
    } // fin de k et de localVar
    k = 8;          // ERREUR ! k n'existe plus
    localVar= 9;    // ERREUR ! localVar n'existe plus
} // fin de i,j
```

PORTEE DES VARIABLES

** 5 types de portées : locale, fonction, boucle, fichier, classe.

** **locale** : un nom déclaré dans un bloc (`{..}`)est local à ce bloc. Il ne peut être utilisé qu'entre la déclaration et la fin du bloc.

** **fonction** : les étiquettes peuvent être utilisées uniquement dans la fonction où elles sont déclarées.

** **variable de boucle** : du début à la fin de la boucle

** **fichier** : un nom déclaré hors de toute fonction, classe ou bloc a une portée de fichier. Il ne peut être utilisé qu'entre la déclaration et la fin du fichier. Attention à la directive `extern`.

** **classe** : un nom déclaré dans une classe est utilisable dans cette classe (fonction membres et amies).

** **Opérateur de résolution de portée : `::`**

accès à un membre de la classe nom_classe ou à la variable globale

Syntaxe : `<nom_classe>::membre`
`::nom`

Exemple:

```
int i;
void f(int j)
{
  int i;
  i=5;        // variable i locale
  ::i=10;     // variable i globale
}
```

Ceci est un exemple, cliquez sur le lien de téléchargement pour obtenir le cours complet.