Un premier exemple

christophe.labourdette(at)cmla.ens-cachan.fr

Septembre 2016

conventions

- Structurer son travail est comme ranger son bureau, indispensable !
- Travailler dans un répertoire *td1*
- Utiliser des suffixes classiques et explicites
    - pour les fichiers sources .cpp, .cc, .cxx, .C
    - pour les fichiers d'entêtes .hpp, .hxx, .hh
- Utiliser un véritable éditeur pour programmer : emacs, vim
- Ou bien un environnement intégré : codeblock

exemple1.cpp

```
#include "exemple1.hpp"
int main() {
    // declaration de i,j et k
    int i,j=1,k=2;
    int res;
    std::cout << "Petit exemple en C++\n";
    std::cout << "Pour j = "<<j<<", k = "<<k<<";
    std::cout <<" , et nbmax = "<<nbmax<<std::endl;
    for (i=j;i<nbmax;i++)
    {
        res = (3 * j++) + 5 * ++k;
        std::cout << "res("<<i<<") = " << res;
        std::cout << std::endl;
    } // fin de la boucle i<nbmax
} // fin du programme
```

exemple1.hpp

```
#include <iostream>
#include <ostream>
#include <cstdlib>
/* declaration des constantes */
/* nbmax represente le nombre d'iteration */

const int nbmax = 15;
```

Des notions

- Entêtes
- Includes
- Portée et Blocks
- Main
- STL et stream
- Variables et initialisation
- Commentaires
- Constantes

exécution

résultat

```
$ ./exemple1
Petit exemple en C++
Pour j = 1, k = 2, et nbmax = 10
res(1) = 18
res(2) = 26
res(3) = 34
res(4) = 42
res(5) = 50
res(6) = 58
res(7) = 66
res(8) = 74
res(9) = 82
```

le fichier d'entêtes

Il contient en général :

- des "includes"
  différence entre " " et <>
- des macros
  #define
- des constantes
  const
- des prototypes de fonctions
- des définitions

Le main

La fonction main :

- est obligatoire
- est unique
- est le coeur de l'exécutable
- renvoie un entier
- peut prendre des arguments
- se comporte comme une fonction classique

compilation

compilation simple

```
g++   exemple1.cpp
```

produit un exécutable a.out

compilation avec options

```
g++ -O2 -o exemple1   exemple1.cpp
```

produit exemple1 avec une optimisation de niveau 2

compilation debug

```
g++ -g -o exemple1_dbg exemple1.cpp
```

produit exemple1_dbg prêt pour le debugger

les différentes étapes

La compilation d'un code source C++ se déroule en **quatre** étapes.

1. Le **preprocesseur** inclue les fichiers d'entêtes dans le code source, exécute les macros commandes et remplace les variables symboliques des #define par leurs valeurs,
2. Le code souce étendu précedemment est ensuite compilé en un code assembleur adapté à la plateforme visée,
3. Le code assembleur généré par le compilateur est assemblé en un code objet pour la plateforme visée,
4. Le code objet est enfin lié avec les codes objets de toutes les fonctions de librairies nécessaires à la production d'un exécutable.

Options utiles

Le compilateur comprend en général de très nombreuses options, il en existe notamment pour préciser ce que l'on veut de la part du compilateur.

- ne produire que le résultat du preprocesseur :

```
g++ -E exemple1.cpp
```

- s'arrêter après l'étape 2 (assembleur), on obtient un fichier *exemple1.s* :

```
g++ -S exemple1.cpp
```

- sarrêter après l'étape 3 (compilateur), on obtient un fichier *exemple1.o* :

```
g++ -c exemple1.cpp
```

syntaxe et bonnes pratiques

- Les instructions pour le préprocesseur se placent au début du fichier,
- Il est essentiel d'indenter son programme,
- **Toutes** les variables doivent être déclarées,
- Les instructions se terminent toujours par un " ;",
- Un bloc d'instructions est plaçé entre accolades { },
- Eviter de déclarer des variables au fil de l'eau, le faire au début,
- Utiliser des noms de variables explicites,
- Commenter n'est jamais inutile,

Que permet de faire le C++

Le **C++** est un langage, générique, procédural, orienté objet, il permet de :

- déclarer et manipuler des variables typées
- effectuer des opérations avec des variables
- effectuer des boucles
- effectuer des opérations conditionnelles
- construire des fonctions
- construire et manipuler des objets
- manipuler les adresses des, variables, fonctions ou objets (pointeurs)

La portée

- Les variables ont une durée de vie limitée au bloc, ainsi qu'aux sous-blocs, ou elles sont déclarées.
- Une variable définie dans une fonction *truc* n'est donc pas visible dans la fonction depuis laquelle truc est appelée.
- Les variables dites globales sont donc déclarées au dessus de toutes les fonctions, y compris main.
- Quand cela est possible, il ne faut pas utiliser de variables globales.

espace de noms

- Il est possible en **C++** de definir des espaces de noms (namespaces).
- Dans un espace de noms sont définis des noms de fonctions ou de variables.
- Cela permet de résoudre des ambiguïtés lorsque l'on est en présence du même nom provenant de plusieurs espace.
- On utilise l'opérateur de résolution de portée pour indiquer d'ou provient la variable.

```
std::cout;
```

provient de l'espace std correspondant à la librairie standard.

Directive using

La directive using permet de s'affranchir de l'opérateur de résolution de portée pour des fonctions ou des espaces de noms.

```
using std::cout;
cout << "plus besoin de std:: \n";
```

On peut également indiquer que l'on va utiliser toutes les fonctions de l'espace de noms.

```
using namespace std;
```

indique que l'on peut utiliser toutes les fonctions de la librairie standard sans spécifier std ::
ATTENTION cet usage est déconseillé !