LO43 - TP1

1 Introduction

1.1 Quelques informations sur le langage C++

Fichier d'entête (header files)

- Extension : *.h, *.hpp
- Déclaration : des class (attributes, méthodes), des noms de fonctions, noms de variables, noms de constantes
- Exemple de fichier d'entête (calculatrice.h) :

```
class calculatrice {
  public:
    calculatrice() {};
    ~calculatrice() {};
    int addition(int a, int b);
}
```

- Le fichier peut également contenir des définitions de méthodes et de fonctions (recommandé seulement pour petite fonction/méthodes)
- Les méthodes de la classe définies dans le corps de la classe sont les méthodes *inline*.

Fichier source (source files)

- Extension : *.c *.cc, *.cpp
- Définir les methodes, fonctions dont le nom est défini dans le fichier d'entête
- Contiennent des références aux fichiers d'entête :
  `#include "..."`
- Exemple de fichier source (calculator.cc) :

```
#include "calculatrice.h"

int calculatrice::addition(int a, int b) {
  return a+b;
}
```

- Contiennent une fonction main(...), dont le programme débute son exploitation (un fichier pour le programme).

La compilation et le linkage

- La compilation : le compilateur compile chaque fichier source (.c, .cc .cpp), c'est-à-dire qu'il crée un fichier binaire (.o) par fichier source. Exemple :

`g++ -c fichier.cpp -I/chemin`

- fichier.cpp - fichier source fichier.cc, qui sera compilé (crée fichier fichier.o)
- -I/chemin - ajouter le rpertoire /chemin la liste des rpertoires o aller chercher les fishier d'entête

- Le linkage : Cette phase, constitue la phase finale pour obtenir un exécutable. Le compilateur agrge chaque fichier .o avec les éventuels fichiers binaires des librairies qui sont utilisées (fichiers .a et .so sous linux, fichiers .dll et .lib sous windows). Exemple :
  `g++ -o programme programme.o binaires1.o . . . binairesN.o -L/chemin -lsunmath`
  - programme.o - fichier binaire de fichier source programme.cc contiennent fonction main(...)
  - binaires1.o - fichier binaire de fichier source binaires1.cc
  - -L/chemin - ajouter le rpertoire /chemin la liste des rpertoires o aller chercher les librairies
  - -lsunmath - librairie sunmath

Makefile

Makefile est un fichier pour le programme make. Le programme make analyse un fichier Makefile et des règles sur cette base que les fichiers source doivent compiler. Il permettre de gagner de temps dans la création du programme, car à la suite de modifications dans le fichier source uniquement les fichiers qui sont dépendant de ce fichier sont compilé.

Makefile se compose d'un ensemble de règles. Règles consistent généralement de cible, des règles de dépendance et des commandes. On peux aussi définir de variable utilises dans le règles. Pour pour exécuter Makefile tape "make".

Exemple :

```
# C'est un commentaire

# Le variable qui indique le compilateur
CXX=g++

# Le variable avec les rpertoires ou ce trouve fichiers d'entetes
INCLUDE=-I/usr/include

# Le variable avec les rpertoires ou ce trouve librairies
LIBRARY_PATH=-L/usr/lib

# Le variable avec les options de compilation
CXXFLAGS= $(INCLUDE) -c
```

```
# Le variable avec les options de linkage
LDFLAGS= $(LIBRARY_PATH)

# Le variable qui contient les librairies avec lesquelles on va
# effectue l'edition de liens
LIBS=-lm

# Le cible:
# <nome_de_cible>: <dependance>
# [tab]<commandes>
# [tab] - tabulation  !!!!!!!!!!!!!!!!

# Le cible qui va cree le executable
executable: programme.o func1.o func2.o
[tab]$(CXX) $(LIB_DIRS) -o executable programme.o func1.o func2.o $(LIBS)

# Le cible qui supprime tout les fichiers interm\'ediaires.
clean:
  rm -f func1.o func2.o programme.o executable


# Les regles de dependance
# <nome_de_regle>: <dependance>
# [tab]<commandes>
# [tab] - tabulation  !!!!!!!!!!!!!!!!

# Les regles de dependance :
programme.o: programme.c func3.h
  [tab]$(CXX) $(CXXFLAGS) programme.c

  func1.o : func1.c
  [tab]$(CXX) $(CXXFLAGS) func1.c

  func2.o : func2.c
  [tab]$(CXX) $(CXXFLAGS) func2.c
```

1.2 Les Resources

- C++ FAQ LITE : http ://jlecomte.ifrance.com/c++/c++-faq-lite/index-fr.html
- Introduction à Makefile : http ://gl.developpez.com/tutoriel/outil/makefile/
- gnu 'make' : http ://www.gnu.org/software/autoconf/manual/make/index.html
- GCC online documentation : http ://gcc.gnu.org/onlinedocs/

2 Excercises

2.1 Helloworld

1) Écrire un programme qui lance une méthode *affiche de classe HelloW affichant sur un écran "HelloW - hello world !".*

HelloW.hpp : déclaration de la classe HelloW
HelloW.cpp : implémentation de la classe HelloW
ClientHW.cpp : programme de test (contient la méthode main)

Compilation et linkage avec g++

2) Ajoute les classes HelloA, HelloB avec les methodes affiche qui affiche sur un ecran "HelloA - hello world !" et "HelloB - hello world !".

Pour chaque classe une fichier d'entête et fichier source.
Compilation avec un fichier makefile.

2.2 Type Abstrait Vecteur

L'objet du TP est de construire le type vecteur et de le tester avec un programme client.

1) Compilation : rapatrier dans votre répertoire de travail une copie du dossier cpp_tp1/vecteur qui contient un squelette de programme et un makefile :

Vecteur.h : interface du TAD
Vecteur.cc : implémentation
Client.cc : programme de test du TAD (contient le main)
Makefile : permet la compilation et l'édition de liens du programme
Après avoir copier les fichiers, tester le compilation en utilisant Makefile.

2) Construction du TAD

Compléter les fichiers Vecteur.h et Vecteur.cc avec les opérateurs demandés.
Procéder par étapes en vérifiant la syntaxe (compilation) au fur et a mesure.
Pour les différents constructeurs/destructeur on définira des compteurs statiques pour calculer le nombre de passages dans chacun d'eux.

3) Programme client
Compléter le programme client avec des exemples d'utilisation du TAD