R6.A.05 / C++

TP2

Classes, objets, destructeur, copy-constructeur

2024 - 2025

Pour ce TP2, on créera sous *VSCode* un nouvel espace de développement « ProjetTP2 » avec l'arborescence habituelle :

/ProjetTP2

/	|	\
/src	**/ws**	**/bin**
*.cpp	Makefile	*.bin
*.h		*.o

Exercice N°1

Ecrire une classe *CPoint* qui contient :

- Deux membres privés *m_abs* et *m_ord* de type *int*.
- Des accesseurs.
- Un constructeur qui prend en arguments deux entiers pour l'initialisation des membres et qui affiche une trace de passage (*cout << "construction de l'objet CPoint d'adresse : " << this << endl*).
- Un constructeur qui ne prend aucun argument et qui initialise les membres à zéro (indispensable pour la création d'un tableau de *CPoint*). Affiche également une trace de passage (*cout << "…" << endl*).
- Un destructeur dans lequel on affichera simplement une trace de passage (*cout << "destruction de l'objet CPoint d'adresse : " << this << endl*).
- Une méthode *presentation()* qui affiche les attributs de l'objet *CPoint* à l'écran.

Dans une première étape, écrire un programme de test de la classe *CPoint* (fichier *testCPoint.cpp*) mettant en œuvre succinctement le test de la classe (**cas normaux uniquement et en faisant appel aux fonctions de tests unitaires des méthodes : *void testConstructeur1EtAccesseurs()*, *void testConstructeur2EtAccesseurs()*, *void testPresentation()***.

Sous *VSCode*, on devra trouver les fichiers *.cpp, .h, .o et .bin* dans l'arborescence suivante une fois le projet compilé :

/ProjetTP2

/	|	\
/src	**/ws**	**/bin**
CPoint.cpp	Makefile	CPoint.o
CPoint.h		testCPoint.bin
AllIncludes.h		
testCPoint.cpp		

Le fichier *AllIncludes.h* vous est fourni et contient l'importation des bibliothèques standards C++ nécessaires aux TPs.

Le fichier *CPoint.h* doit être créé et incluera le fichier *AllIncludes.h* :

```
#ifndef CPOINT_H
#define CPOINT_H

#include "AllIncludes.h"

class CPoint {
     // attributs
     private :
          …

     // méthodes
     public :
          …
};
#endif
```

Le fichier *testCPoint.cpp* contiendra le lanceur *main()* + les fonctions de test :

```
#include "CPoint.h"
void testConstructeur1EtAccesseurs() ;
void testConstructeur2EtAccesseurs() ;
void testPresentation() ;

int main() {
     …
}

void testConstructeur1EtAccesseurs() {…}
void testConstructeur2EtAccesseurs() {…}
void testPresentation() {…}
```

La compilation se fera en 2 étapes :

1. Compiler d'abord la classe *CPoint.cpp* pour obtenir la version binaire *CPoint.o*.

2. Compiler ensuite le lanceur *testCPoint.cpp* pour obtenir l'exécutable *testCPoint.bin* qui intègrera le binaire de la classe *CPoint*.

Le fichier Makefile s'écrira de la façon suivante (attention aux tabulations !!) :

```
GPP = g++ -Wall
SRC = ./src
BIN = ./bin

compilCPoint :
	@echo Compilation CPoint
	$(GPP)	$(SRC)/CPoint.cpp	-o	$(BIN)/CPoint.o	-c

# pour compiler testCPoint on commence d'abord par compiler CPoint
compilTestCPoint :  compilCPoint
	@echo Compilation testCPoint
	$(GPP) $(SRC)/testCPoint.cpp $(BIN)/CPoint.o -o $(BIN)/testCPoint.bin
```

Pour l'exécution, sous *VSCode* cliquer sur "Makefile" dans le menu de la barre de gauche puis cliquer sur l'icône "Run the selected binary target in the terminal".

Dans une deuxième étape, créer un fichier *Lanceur.cpp* et écrire dans le *main ()* l'appel à 4 fonctions décrites ci-dessous. Chacune de ces fonctions utilise en paramètre soit un objet *CPoint* (avec passage par valeur, par référence et par pointeur), soit un tableau d'objets *CPoint* qui devra être préalablement construit et initialisé avec des objets *CPoint*. Le type de paramètre passé à la fonction sera :

1. pour la fonction1 : un objet *CPoint theP (passage par valeur)*

2. pour la fonction2 : un objet *CPoint& theP (passage par référence)*

3. pour la fonction3 : un pointeur sur un objet *CPoint*, c'est-à-dire *CPoint* pTheP*

4. pour la fonction4 : un tableau d'objets *CPoint,* c'est-à-dire *CPoint* pTabPt,* de même que sa taille (*int tailleTab*)

Les points (ou tableau de points de taille donnée et complètement rempli) sont chaque fois créés dans le lanceur *main()*, puis ils sont passés en paramètre à la fonction. Chaque fonction a simplement pour rôle d'afficher les attributs de l'objet *CPoint* (ou du tableau d'objets) à l'écran en appelant la méthode *presentation()*.

Appeler chaque fonction séparément dans *Lanceur.cpp* et analyser les affichages (les traces dans le constructeur et le destructeur).

La déclaration des signatures des 4 fonctions se fera en en-tête dans le fichier *Lanceur.cpp* :

```
#include "CPoint.h"

void fonction1 ( CPoint theP );
void fonction2 ( CPoint& theP );
void fonction3 ( CPoint* pTheP );
void fonction4 ( CPoint* pTabPt, int tailleTab );

int main() {…}

void fonction1 ( CPoint theP ) {…}

etc.
```

Exercice N°2

Soit une classe (décrite dans un fichier *CVoiture.h*) :

```
class CVoiture {

      private:
            int   m_type;
            char* m_pNom;
            char* m_pCouleur;

      public:
            ~CVoiture();                        // éventuellement
            CVoiture ( int typeVt, char* nomVt, char* coulVt );
            CVoiture ( const CVoiture& vt );    // éventuellement
            void setVoiture (int typeVt, char* nomVt, char* coulVt);
            void presentation ();
};
```

La fonction membre *presentation()* permet d'afficher les données membres (attributs) d'un objet *CVoiture*.

A. Version1

Dans cette version (la + simple), les chaînes de caractères passées en paramètres au constructeur *CVoiture(...)* et à la méthode *setVoiture(...)* ne sont PAS DUPLIQUEES (il y aura juste une recopie de pointeurs). Dans le lanceur *testCVoiture.cpp*, après la construction d'un objet *CVoiture*, modifiez, sans la re-créer, une des chaînes de caractères passée au constructeur et affichez ensuite les attributs de *CVoiture* : que se passe-t-il ? Tentez l'écriture d'un destructeur des chaînes de caractères : que se passe-t-il à l'exécution ?

B. Version2

Dans cette version, on veut que tous les objets *CVoiture* créés possèdent leur propre copie des chaînes de caractères. Les chaînes de caractères passées en paramètres au constructeur *CVoiture(...)* et à la méthode *setVoiture(...)* sont alors DUPLIQUEES (tous les objets *CVoiture* créés possèdent leur propre copie des chaînes de caractères). Ecrivez également un copy-constructeur qui fera cette duplication. Faut-il écrire un destructeur ?

Important : utiliser les fonctions *strlen* et *strcpy* pour dupliquer les chaînes facilement.

Pour chacune de ces 2 versions :

Ecrire le fichier *CVoiture1.cpp* (version 1) et *CVoiture2.cpp* (version 2).

Ecrire un programme de test de chaque classe *CVoiture* (fichiers *testCVoiture1.cpp* et *testCVoiture2.cpp* contenant un *main()*) mettant en oeuvre le test complet de la classe par appel de fonctions de tests unitaires (comme pour le test de la classe *CPoint*).

Compilation : écrire à la suite, dans le même fichier *Makefile* que pour *CPoint*, les cibles *compilCVoiture1*, *compilCVoiture2*, *compilTestCVoiture1*, *compilTestCVoiture2*.

Références nécessaires

- Les flux d'entrées/sorties « cin » et « cout » instances respectives des classes de la bibliothèque standard C++, « istream » et « ostream ».
- La définition de classes en C++.
- Le constructeur de recopie.
- Le cours C++.

De l'aide sur C++ est disponible sur les sites suivants :

http://www.cplusplus.com/ref

http://www.cppreference.com