Cpp

Gilles Maire <gilles@gillesmaire.com>

Corrigé des exercices

15/11/2018

IGNU

Contents

Exercice 1 : Classe vs Structure

Énoncé

- Le but de cet exercice est de constater la similitude entre classe et structure

1. Déclarer un programme C contenant une structure
2. Déclarer un pointeur sur la structure et allouer lui un espace mémoire
3. Déclarer et définir un champ entier dans la structure
4. Dans le programme passer le champ à la valeur 2. Afficher le champ
5. Déclarer une fonction *clear* dans la structure qui met le champ de la structure à zéro par `void clear(){i=0;}`
6. Appeler la fonction clear(). Afficher le champ.
7. Compiler en C et en C++ les différentes étapes. Que constatez-vous ?

Solution

```
#include "malloc.h"

struct MaStructure {
    int i;
    void clear() { i=0;}
};


int main()
{
    struct MaStructure *p ;
    p= ( MaStructure *) malloc (sizeof (struct MaStructure ));
    p->i=2;
    p->clear();
    printf ("%d\n",p->i);
}
```

Exercice 2 : Dix lignes cout

Énoncé

- Écrire un programme qui affiche 10 lignes à l'écran.
- Chaque ligne comprendra
  - un numéro incrémental de 1 à 10
  - suivi d'un message fixe

Solution

```
#include <iostream>

int main()
{

    for ( int i =0 ; i<10; i++)
    {
       std::cout << i << std::endl;
     }

    return 0;
}
```

Exercice 3 : Classement entiers

Énoncé

1 - Écrire une méthode qui prend deux paramètres entiers en entrée et renvoie ces deux paramètres entiers classés (le premier paramètre le plus petit)

2 - On écrira la même méthode deux fois l'une avec passage des arguments par pointeur l'autre avec passage par référence.

Solution

```
#include <iostream>
using namespace std;

void classe (int &a, int &b)
{
    if ( a >b )
   {
     int c=a;
     a=b;
     b=c ;
 }
}
void classeOldSchool (int *a, int *b)
{
    if ( *a >*b )
   {
     int c=*a;
     *a=*b;
     *b=c ;
 }
}
```

```
int main()
{
int i=2;
int j=1;
classe(i,j);
classeOldSchool(&i,&j);
cout<<i<<endl<<j<<endl;
}
```

Exercice 4 : Jeu

Énoncé

Ces paires de fonctions sont-elles surchargeables ?

```
1  void fonction ( int i , char *);
2  void fonction ( char * , int i);
3
4  void fonction ( int i , char *);
5  int fonction ( int i , char *);
6
7  void fonction ( int i, int j);
8  void fonction ( int nombre, int facteur);
9
10 void fonction ( int i , int j );
11 void fonction ( int i , int j = 0);
```

Solution

Exercice 5 : Classe entier

Énoncé

- Définir une classe dotée d'un membre entier et d'une méthode print() qui permet d'afficher la valeur du nombre membre.

Solution

```
#ifndef MACLASSE_H
#define MACLASSE_H

class MaClasse
{
public:
```

```
    MaClasse(const int &val);
    int m_i ;
    void print();
};

#endif // MACLASSE_H

#include "maclasse.h"
#include <iostream>
MaClasse::MaClasse(const int &i)
{
    m_i=i;
}

void MaClasse::print()
{
    std::cout<<m_i<<std::endl;
}
```

Exercice 6 : Constructeur Copie

Énoncé

- Il s'agit de mettre en évidence la faiblesse du constructeur de copie synthétisé par un cas d'école
- Écrire un programme contenant une classe
- Cette classe contient un attribut entier, un pointeur sur cet attribut ainsi qu'une méthode qui imprime le contenu du pointeur.
- Utilisez le constructeur de copie synthétisé pour copier un deuxième objet
- changer la valeur de l'attribut entier
- Sur quoi pointe le pointeur de l'objet pointé ?
- Corriger le problème avec un constructeur de copie convenable

Solution

```
#ifndef MACLASSE_H
#define MACLASSE_H

class MaClasse
{
public:
    MaClasse();
    MaClasse(const MaClasse &mc);
    int m_i;
    int  *m_ptr;
    void setI( int i );
    void setPtr ();
```

```
    void printI();
};

#endif // MACLASSE_H

#include "maclasse.h"
#include <iostream>

MaClasse::MaClasse()
{
}

MaClasse::MaClasse ( const MaClasse &mc)
{
   m_i=mc.m_i;
   m_ptr=&m_i;
}

void MaClasse::setI(int i)
{
    m_i=i;
}

void MaClasse::setPtr()
{
    m_ptr=&m_i;
}

void MaClasse::printI()
{
    std::cout<<*m_ptr<<std::endl;
}
```

Exercice 7 : Constructeur

Énoncé

Pour la classe QCombobox trouver le type de constructeur appelé dans chacun des cas

```
1 QCombobox *p;
2 QCombobox comb;
3 QCombobox *p = new QCombobox ( "titre");
4 QCombobox *p = new QCombobox ( "titre1","titre2");
5 QCombobox comb="titre";
6 Affiche (comb);
```

Solution

1. QCombobox *p : pas de constructeur appelé, il s'agit d'une déclaration de pointeur. 2 QCombobox comb : constructeur par défaut ou synthétisé
2. QCombobox *p = new QCombobox ("titre"); : constructeur par transtypage
3. QCombobox *p = new QCombobox ("titre1","titre2"); : constructeur à deux arguments
4. QCombobox comb="titre"; : constructeur par défaut avec transtypage avec conversion implicite
5. Affiche (comb) : constructeur par copie

Exercice 8 : Mise en oeuvre constructeur

Énoncé

- Mettre en évidence les différentes remarques évoquées précédemment
- nous définirons 4 attributs

Solution

ifndef MACLASSE_H

```
#define MACLASSE_H

class MaClasse
{
public:
    MaClasse();
    MaClasse(const MaClasse &mc);
    int m_i;
    int  *m_ptr;
    void setI( int i );
    void setPtr ();
    void printI();
};

#endif // MACLASSE_H

#include "maclasse.h"
#include <iostream>

MaClasse::MaClasse()
{
}
```

```
MaClasse::MaClasse ( const MaClasse &mc)
{
   m_i=mc.m_i;
   m_ptr=&m_i;
}

void MaClasse::setI(int i)
{
    m_i=i;
}

void MaClasse::setPtr()
{
    m_ptr=&m_i;
}

void MaClasse::printI()
{
    std::cout<<*m_ptr<<std::endl;
}
```

Exercice 9 : Methode et variable statiques

Énoncé

- Déclarer une classe, munie d'une méthode publique
- Dans cette méthode déclarer une variable statique, l'initialiser et l'incrémenter
- Appeler la méthode plusieurs fois depuis le main

Solution

- Machin.h ~cpp #ifndef MACHIN_H #define MACHIN_H

class Machin { public: static int i ; static void methode(); };

endif // MACHIN_H

~

- Machin.cpp

```
#include "machin.h"

int Machin::i =2;
```

```
void Machin::methode()
{
    i ++ ;
}
```

- main.cpp

```
#include <iostream>
#include "machin.h"
using namespace std;
int main()
{
  for ( int i = 0; i< 10; i++)
  {
      Machin::methode();
      std::cout<<Machin::i<<std::endl;
  }
}
```

Exercice 10 : Constructeur classe fille

Énoncé

Mettre en évidence par des affichages dans les constructeurs mère et fille que lorsqu'on instancie une classe fille, on appelle les constructeurs de tous les parents.

Solution

- Mere.h

```
#ifndef MERE_H
#define MERE_H


class Mere
{
public:
    Mere();
};

#endif // MERE_H
```

- Mere.cpp

```
#include "mere.h"
#include <iostream>

Mere::Mere()
{
   std::cout<<"mere"<<std::endl;
}
```

- Filles.h

```
#ifndef FILLES_H
#define FILLES_H

#include "mere.h"
class Filles : public Mere
{
public:
    Filles();
};

#endif // FILLES_H
```

- Filles.cpp

```
#include "filles.h"
#include <iostream>
Filles::Filles()
{
    std::cout<<"fille"<<std::endl;
}
```

- main.cpp

```
#include <iostream>
#include "filles.h"
using namespace std;

int main()
{
    Filles f;
    return 0;
}
```

- L'exécution

```
mere
fille
```

Exercice 11 : Héritage multiple

Énoncé

```
class Mere {
public: Mere();
    Mere( int i );
    void FonctionMere( int i );
};
class Fille: public Mere {
public: Fille();
    void FonctionFille ( int i );
};
```

Quelles sont les possibilités correctes et incorrectes ?

1. Mere a(10);
2. Mere b; b.FonctionFille();
3. Fille c(12);
4. Fille d; d.FonctionMere();

Solution

1. Mere a(10) : correct
2. Mere b; b.FonctionFille() : incorrect
3. Fille c(12) : incorrect
4. Fille d; d.FonctionMere() : incorrect

Exercice 12 : Jeu mère fille

Énoncé

```
class Mere {
public: int publique;
private: int privee;
protected : int protegee;
};
class Fille : public Mere{
private: int fille;
};
```

Départagez les affirmations vraies et fausses :

1. Dans main.cpp : Mere m ; m.privee=1;

2. Dans main.cpp : Fille f; f.protegee=1;
3. Dans main.cpp : Fille f; f.privee=3;
4. Dans la classe Fille : protegee=1;
5. Dans la classe Fille : privee=3;

Solution

1. Dans main.cpp : Mere m ; m.privee=1; : incorrect
2. Dans main.cpp : Fille f; f.protegee=1; : incorrect
3. Dans main.cpp : Fille f; f.privee=3; : incorrect
4. Dans la classe Fille : protegee=1; : correct
5. Dans la classe Fille : privee=3; : correct

Exercice 13 : Redéfinition de fonction

Énoncé

- Mettre en évidence un cas de redéfinition de fonction
- Accéder à la méthode redéfinie à partir de la classe fille.
  - avec sa définition mère
  - avec sa définition fille

Solution

- mere.h

```
#ifndef MERE_H
#define MERE_H
#include <iostream>

class Mere {
public:
   void print() { std::cout<<"mere"<<std::endl;}
    };



#endif // MERE_H
```

- fille.h

```
#ifndef FILLES_H
#define FILLES_H
#include <iostream>

#include "mere.h"
class Fille : public Mere
{
   public :
    void print() { std::cout<<"filles"<<std::endl;}

};
#endif // FILLES_H
```

- main.cpp

```
#include <iostream>
#include "filles.h"
using namespace std;

int main()
{
 Fille  f;
 f.print();
 f.Mere::print();

}
```

Exercice 14 : Polymorphisme

Énoncé

- Déclarer une classe Mere et une classe Fille
- Définir pour chacune d'entre elles des membres différents
- Déclarer une fonction de même signature dans chaque classe mais qui produit un traitement différent (cout<< différent)
- Déclarer dans la fonction main deux pointeurs sur la classe Mere
- Le premier pointera sur un élément Mère
- Le second sur un élément Fille
- Appeler la fonction de même signature depuis les deux pointeurs
- Que se passe-t-il à l'appel de la fonction de même signature ?

Solution

- mere.h

```
#ifndef MERE_H
#define MERE_H
```

```
#include <iostream>

class Mere {
public:
   void print() { std::cout<<"mere"<<std::endl;}
    };



#endif // MERE_H
```

- fille.h

```
#ifndef FILLES_H
#define FILLES_H
#include <iostream>

#include "mere.h"
class Fille : public Mere
{
   public :
    void print() { std::cout<<"filles"<<std::endl;}

};
#endif // FILLES_H
```

- main.cpp

```
#include <iostream>
#include "filles.h"
using namespace std;

int main()
{
 Mere m;
 Fille f;
 Mere *ptrm = &m;
 ptrm->print();
 Mere *ptrf = &f;
 ptrf->print();

}
```

- Sortie

```
mere
mere
```

Exercice 15 : Polymorphisme effectif

Énoncé

- Reprendre l'exercice précédent et déclarer les fonctions identiques virtuelles.

Solution

- mere.h

```
#ifndef MERE_H
#define MERE_H
#include <iostream>

class Mere {
public:
   virtual void print() { std::cout<<"mere"<<std::endl;}
    };



#endif // MERE_H
```

- fille.h

```
#ifndef FILLES_H
#define FILLES_H
#include <iostream>

#include "mere.h"
class Fille : public Mere
{
   public :
    void print() { std::cout<<"filles"<<std::endl;}

};
#endif // FILLES_H
```

- main.cpp

```
#include <iostream>
#include "filles.h"
using namespace std;

int main()
{
 Mere m;
```

```
 Fille f;
 Mere *ptrm = &m;
 ptrm->print();
 Mere *ptrf = &f;
 ptrf->print();

}
```

- Sortie

```
mere
fille
```

Exercice 16 : Heure et opération

Énoncé

- Écrire une classe Heure définissant trois membres Heure, Minute, Seconde
- Créer l'opérateur + ajoutant deux heures d'abord par une fonction membre, mettre en évidence un problème de symétrie.
- Corriger en définissant l'opérateur par une fonction friend

Solution

- Fonction membre
- heure.h

```
#ifndef HEURE_H
#define HEURE_H

#include <iostream>

class OperateurHeure;

class HEURE
{
public:
    HEURE( int h , int m, int s){ heure=h; minute=m ; seconde=s;}
    friend HEURE operator+( HEURE a,  HEURE b);
    void print () { std::cout << heure<<":"<<minute<<":"<<seconde<<std::endl;}

private :
    int heure;
    int minute;
    int seconde;
```

```
};


    HEURE operator+(  HEURE a,  HEURE b )
    {
        int h,m,s;
        int minuteretenue=0;
        int heureretenue=0;
        s=a.seconde+b.seconde;
        if  ( s > 60 ) { minuteretenue=1; s -= 60;}
        m=a.minute+b.minute+minuteretenue;
        if  ( m > 60 ) { heureretenue=1; m -= 60;}
        h=a.heure+b.heure+heureretenue;
        return HEURE(h,m,s);
    }


#endif // HEURE_H
```

- main.cpp

```
#include <iostream>
#include "heure.h"

using namespace std;

int main()
{
    HEURE a(8,0,0);
    HEURE b(2,0,10);
    a.print();
    b.print();
    b=a+b;
    a=a+b;
    b.print();
    a.print();
    return 0;
}
```

- fonction friend

```
#ifndef HEURE_H
#define HEURE_H

#include <iostream>

class OperateurHeure;

class HEURE
```

```
{
public:
    HEURE( int h , int m, int s){ heure=h; minute=m ; seconde=s;}
    friend HEURE operator+( HEURE a,  HEURE b);
    void print () { std::cout << heure<<":"<<minute<<":"<<seconde<<std::endl;}

private :
    int heure;
    int minute;
    int seconde;

};


    HEURE operator+(  HEURE a,  HEURE b )
    {
        int h,m,s;
        int minuteretenue=0;
        int heureretenue=0;
        s=a.seconde+b.seconde;
        if  ( s > 60 ) { minuteretenue=1; s -= 60;}
        m=a.minute+b.minute+minuteretenue;
        if  ( m > 60 ) { heureretenue=1; m -= 60;}
        h=a.heure+b.heure+heureretenue;
        return HEURE(h,m,s);
    }


#endif // HEURE_H
```

Exercice 17 : Exception

Énoncé

Écrire une classe *chiffre* qui comporte une exception permettant de tester si un entier est un chiffre

Solution

```
#include  <iostream>

using namespace std;
class exemple {};


class Analyse {
```

```
public:
void LectureChiffre(){
        int i;
        cout << "Entrez un entier :";
        cin >> i;
        if (i >9 || i < 0)
throw
        exemple();
        else
              cout << i << "accepte" <<"\n";
}

};


main(){
try
{                       // Bloc pouvant generer une exception
            Analyse a;
            a.LectureChiffre();
        }
catch
(exemple) {         // Capter et traiter l'erreur
            cerr << "Erreur, entier invalide\n";
        }
}
```

Exercice 18 : Swap en template

Énoncé

- Définir une fonction template swap(T i, T j) qui swappe deux objets
- L'appeler avec deux entiers
- L'appeler avec deux élements d'une classe définie

Solution

```
#include <iostream>

using namespace std;


template < class T>
void swap ( T *arg1 , T *arg2)
{
    T temp=*arg1;
```

```
    *arg1=*arg2;
    *arg2=temp;
}

class Exemple {
public:
    Exemple(int i){m_i=i;}
    int val(){return m_i;}
private:
   int m_i;
};

int main()
{
    int i=1 ; int j=2;
    cout<<i<<j<<endl;
    swap (i, j);
    cout<<i<<j<<endl;
    Exemple e1(1);
    Exemple e2(2);
    cout<<e1.val()<<e2.val()<<endl;
    swap(e1,e2);
    cout<<e1.val()<<e2.val()<<endl;;
    return 0;
}
```