| | | | |
|---|---|---|---|
| CY UNIVERSITÉ | **ING2 MF : PROGRAMMATION C++**<br>**EXAMEN SESSION Normale**<br>**DUREE : 1:30**<br>**Aucun document n'est autorisé** | | |
| Nom: | Prénom: | Groupe: | Créé le 06/12/2024 |

**Exercice 1 :** Soit le programme principal suivant:

```
void main() {
        int a = 1; int b = -2; int c = 0;
        ajouter(a,b,c);
        cout << "apres appel de ajouter" << endl;
        cout << "c = " << c << endl;
}
```

**Questions** :

1. Donner la sortie du programme dans les 3 cas suivants:

   a. **void** ajouter (**int** a, **int** b, **int** c) { c = a+b; }

*apres appel de ajouter*
*c = 0*

   b. **void** ajouter (**int** a, **int** b, **int** * c) { *c = a+b; }

*invalid conversion from int to *int*

   c. **void** ajouter (**int** a, **int** b, **int** & c) { c = a+b; }

*apres appel de ajouter*
*c = -1*

2. Comment on appelle chacun de ces 3 passages de paramètres ?

*pour le paramètre **c**, l'appel se fait par : cas a. Par copie de valeur,*
*cas b. par copie de pointeur,*
*cas c. par copie de référence*
*pour les paramètres **a** et **b**, l'appel se fait par copie de valeur dans les 3 cas*

**Exercice 2 :** Soit la classe Forme est contenue dans le fichier nommée forme.h

```
#ifndef  FORME_H_
#define  FORME_H
#include<string>
#include <iostream.h>
using namespace std;

class Forme {

public:
  virtual void description( const string s) {
    cout << "Ceci est une forme." << endl;
  }
  virtual const double aire(){};

private:
virtual double perimetre() const =0;
};

#endif
```

**Questions :**

1. Cette classe ne peut pas possèder de constructeur, pourquoi ?
2. Que signifie le mot clé **virtual** dans chacune des 3 méthodes
3. Que signifie le mot clé **const** dans chacune des 3 méthodes
4. Est-ce que ce code va compiler ou pas ? Expliquer

**Réponses :**

1. Cette classe ne peux pas posséder de constructeur, pourquoi ?
   *Parce qu'elle possède une **méthode virtuelle pure**, la classe est dite est abstraite*

2. Que signifie le mot clé **virtual** dans les 3 méthodes
   *Méthodes polymorphes*
   *Les 2 premières méthodes déjà définies, **peuvent être redéfinies** dans les classes fille*
   *La dernière qui est **virtuelle pure doit être définie** dans la classe fille*

3. Que signifie le mot clé **const** dans les 3 méthodes

   *Dans la méthode description(), cette méthode ne peut modifier **la valeur string s***
   *Dans la méthode aire(), **la valeur retournée** par cette méthode **ne peut être modifiée***
   *Dans la méthode perimetre(), l'appel à cette méthode **ne peut modifier les propriétés de de l'instance appellante***

4. Pourquoi ce code ne compilera-t-il pas ? Expliquer
   *On ne peut pas avoir une **méthode virtuelle pure en mode private***

**Exercice 3 :** Soit la classe Point suivante définie dans le fichier point.h

```
////////////////////////////////////// fichier point.h ////////////////////////////////////////////
#include <iostream.h>
using namespace std;

class Point {
private : short x, y;
public :
        Point() { cout << " ++ normal \n"; }
        ~Point() { cout << " -- normal  \n"  ; }
} ;
```

Soit le fichier main.h suivant, donner la sortie du programme C++ main suivant et la commenter très brièvement

```
////////////////////////////////////// fichier main.h ////////////////////////////////////////////
#include "point.h"

a.  void main() {
        Point  p;
}
```

*++ normal  (on appelle le constructeur, pas de destructeur donc  destructeur par défaut)*

**b.** Même question avec.

```
void main() {
        Point * p = new Point();
}
```

*++ normal (on appelle le constructeur, le destructeur est appelé à la fin du programme donc non visible)*

**c.** Même question avec.

```
void main() {
        Point * p = new Point();
        delete p ;
}
```

*++ normal (on appelle le constructeur)*
*-- normal (on appelle le destructeur explicitement avant la fin du programme)*

**Exercice 4 :** On reprend la classe Point et on la modifie ainsi (voir ci-dessous).

```
////////////////////////////////////////////fichier point.h ////////////////////////////////
#ifndef  POINT_H_
#define  POINT_H_
#include <iostream.h>
using namespace std;

class Point {
private : short x, y;
public :
        Point(short abs, short ord) : x(abs) , y(ord) {}
        virtual void decrire(){
          cout << "je suis un point \n";
          cout <<"mes coordonnees  : " << x << " " << y << "\n";
        }
        ~Point() {};
} ;
#endif

////////////////////////////////////////////////////fichier pointcol.h ////////////////////
#ifndef  POINTCOL_H_
#define  POINTCOL_H_
#include "point.h"

class PointCol : public Point{
  private :
       unsigned int color;
  public :
    PointCol(short abs, short ord, unsigned int cl): Point(abs, ord){ color = cl;}
    void decrire (){
       cout << "je suis un point colore \n";
       cout <<"mes coordonnees  : " << x << " " << y << " et ma couleur : " << color <<"\n";
     }
};
#endif
```

```
////////////////////////////////////////// fichier  main.h ////////////////////////////////////////
#include "pointcol.h"

void main(){
    Point  *p1 = new Point(3,5);                p1-> decrire();
    PointCol  *pc1=new PointCol(8,6,2);         pc1-> decrire ();
    Point *p2=new PointCol(8,6,5);              p2-> decrire();
    PointCol  *pc2=new Point(7,6);              pc2-> decrire ();
}
```

***Question :***

1. Comment s'est faite l'initialisation des propriétés dans le constructeur? Et pourquoi?
2. Les propriétés sont définies avec le type **short** dans la classe Point et **unsigned int dans la classe PoinCol.** Quelle est la différence?
3. Cette classe Point est-elle une classe canonique ou pas commenter?
4. A quoi sert cette instruction : "**using namespace** std"; Si on le supprime, qu'est ce qu'il faut ajouter pour que le programme compile
5. Exliquer l'interet de la directive suivante "#ifndef  POINT_H_  #define  POINT_H_"
6. Dans la déclaration de la classe fille pointCol, expliquer ce que signifie **public** devant la classe Point
7. Dans la classe fille pointCol nous avons un problème d'accès à ses  propriétés, montrer ce problème? Comment corriger?
8. Comment appelle-t-on ce typage dans la construction de nos points colorés dans le main
9. Dans la fonction main une ligne est fausse, laquelle et pourquoi?
10. Supprimer cette ligne et afficher la sortie

***Réponses :***

1. *L'initialisation s'est faite **à la C++ et non à la C**. L'intérêt est **le contrôle du type***
2. *Une variable de type **unsigned int** est une variable qui ne peut prendre que des valeurs positives ou nulles (**valeurs naturelles**). La taille de ce type dépend de la machine (de l'implémentation des types)*
   *Une variable de **type short** peut prendre des valeurs signées (positives ou négatives) mais de **taille réduite**, en général un octet.*
3. *Cette classe Point est-elle une classe canonique ou pas commenter? Non car il manque le constructeur de recopie ainsi que la surcharge de l'opérateur d'affectation*
4. A quoi sert "**using namespace** std";
   *Namespace, veut dire **espace de nommage ou espace de nom sert à regrouper des variables et des fonctions, des classes**, tout ce que vous voulez dans un même ensemble.*
   *Les fonctions d'entrée-sortie standard du C++ (cin, cout) sont déjà situées dans un espace de noms : std. Si on le supprime, qu'est ce qu'il faut ajouter pour que le programme compile*
   *Il faut précéder la méthode cout de std (**std::cout**)*
5. Expliquer la directive suivante "#ifndef  POINT_H_  #define  Point_H_  ….#endif"
   Afin d'éviter des conflits, charger en mémoire (dans le projet) le fichier point.h s'il n'a pas été encore chargé sinon ne rien faire. **Charger en mémoire une seule copie du fichier point.h dans le programme ou dans le projet**
6. *La classe fille PointCol dérive de la classe Point ne change pas les modes (**public**, protected, private). Avec le spécificateur public, la classe fille va garder les mêmes spécificateurs d'accès que la classe mère (voir le cours pour **protected** et pour **private**).*
7. *Dans la classe fille PointCol, on essaie d'accéder aux propriétés x y de la classe mère Point qui sont déclarées **en private**. Ce n'est pas possible.*
   *Solution 1 : implémenter des getteurs (getX et getY) dans la classe mère Point*
   *Solution 2 : les déclarer en mode d'accès Protected dans la classe mère Point*

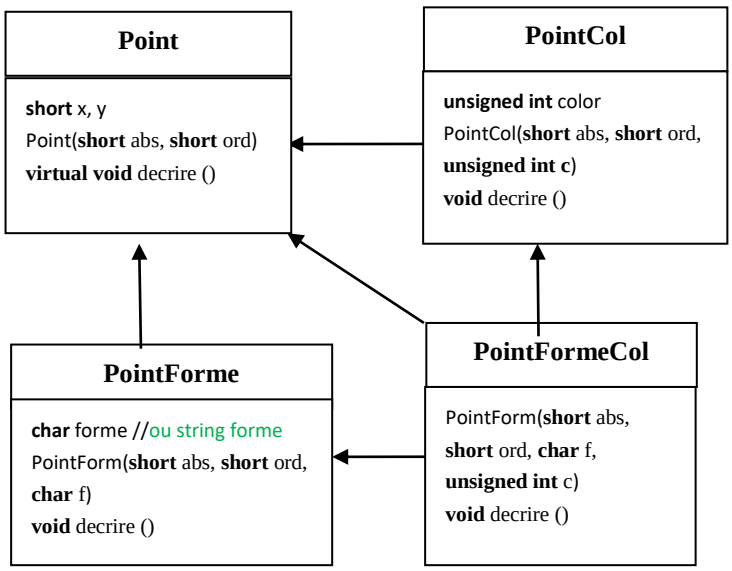
8. *On fait du **typage dynamique à l'exécution. Le type de l'objet à créer est choisi à l'exécution et non à la compilation (Polymorphisme). L'objet est déclaré comme un point mais créé comme un point coloré***
9. C'est la **4ième ligne PointCol *pc2=new Point(7,6); pc2->decrire();**
   On ne peut pas créer une instance de la classe mère Point en utilisant le constructeur de la classe fille PointCol
10. *Si on supprime cette ligne le résultat de l'affichage est le suivant :*

*je suis un point*
*mes coordonnees sont : 3 5*
*je suis un point colore*
*mes coordonnees sont : 8 6*
*ma couleur est : 2*
*je suis un point colore*
*mes coordonnees sont : 8 6*
*ma couleur est : 5*

**Exercice 5** : On reprend les deux classes précédentes, Point et PointCol en corrigeant les erreurs, et on complète par deux autres classes PointForme qui contient un attribut de type chaine, pour dire comment le point sera dessiné (un point, un carré, un cercle, etc…), et PointFormeCol.

1. Donner le schema complet de votre programme avec toutes les classes, leurs attributs, leurs méthodes et leurs dépendances. (avec des boites comme UML)
2. Donner le fichier.h de chacune de ces 2 nouvelles classes ajoutées.
3. Dans la nouvelle méthode principale main (), définissez le template Vector<T> (un conteneur d'objets), ajouter un objet créé de chaque type à ce conteneur. Pour finir parcourir ce template et decrire chacun des objets contenus dans ce dernier en utilisant la method decrire() implémentée dans chacune des classes (utiliser un ierateur pour parcourir votre conteneur).

1. Diagramme de classes



```
//////////////////////////////////////////////////fichier pointforme.h ////////////////////
#ifndef  POINTFORME_H_
#define  POINTFORME_H_

#include "point.h"
#include<string>
class PointForme : virtual public Point{
  protected :
        char forme; // ou string forme
  public :
    PointForme(short abs, short ord, char f): Point(abs, ord), forme(f){}
    void decrire (){
       cout << "je suis un point avec une forme \n";
       cout <<"mes coordonnees  : " << x << " " << y << " et ma forme : " << forme <<"\n";
     }
};
#endif


//////////////////////////////////////////////////fichier pointformecol.h ////////////////////
#ifndef  POINTFORMECOL_H_
#define  POINTFORMECOL_H_

#include "pointcol.h"
#include "pointforme.h"

class PointFormeCol : virtual public Point,
                                    public PoinrForme,
                                    public PointCol
{
  public :
      PointFormeCol(short abs, short ord, char f, unsigned int cl):   Point(abs, ord),
                                                                                  PointFome(abs, ord, f),
                                                                                 PointCol(abs, ord, c){ }
     void decrire (){
       cout << "je suis un point colore avec une forme \n";
       cout <<"mes coordonnees  : " << x << " " << y << ", "
                << " ma couleur : "<< color << ", "
                <<" ma forme : " << forme <<  "\n";
     }
};
#endif
```

```
//////////////////// fichier main.cpp ////////////////////////////
#include "pointformecol.h"
#include<vector>

int main() {

vector<Point*> LPs;
vector<Point*>::iterator it;

Point *p   = new Point(3,5);
Point *pc  = new PointCol(3,5,3);
Point *pf  = new PointForme(5,6,'o');
Point *pfc = new PointFormeCol (5,6,9,'x');

LPs.push_back(p);
LPs.push_back(pc);
LPs.push_back(pf);
LPs.push_back(pfc);

for(it=LPs.begin(); it!=LPs.end(); ++it)
    (*it)->decrire();

return 0;
}
```

## Annexe : méthodes de la classe template <typename ValueType> class Vector

**Member functions**
(constructor) Construct vector (public member function)
(destructor) Vector destructor (public member function)
operator= Assign content (public member function)

**Iterators**:
begin Return iterator to beginning (public member function)
end Return iterator to end (public member function)

**Capacity**:
size Return size (public member function)
max_size Return maximum size (public member function)
resize Change size (public member function)
capacity Return size of allocated storage capacity (public member function)
empty Test whether vector is empty (public member function)

**Element access**:
operator[] Access element (public member function)
front Access first element (public member function)
back Access last element (public member function)
data Access data (public member function)

**Modifiers**:
push_back Add element at the end (public member function)
pop_back Delete last element (public member function)
insert Insert elements (public member function)
erase Erase elements (public member function)
swap Swap content (public member function)
clear Clear content (public member function)