© *Jean-Baptiste APOUNG KAMGA <jean-baptiste.apoung@math.u-psud.fr>*

Fiche de TP2 : Programmation orientée objet en C++

Le but de ce TP est d'aborder les outils fournis par le C++ , pour la programmation orientée objet. Pour gagner du temps dans l'écriture des bouts de programmes fournis dans les exercices, nous rappelons que les fichiers sources de ces programmes sont téléchargeables à l'adresse **`/doc/apoung/M2_IM_STAT`**

Thème - 1 *Abstraction - classes*

Exercice-1 **:** Concevoir une class **`Q`** modélisant modélisant l'ensemble des nombres rationnels.

▲ On fournira
- un constructeur et (ou) destructeur,
- une fonction membre **`val()`** retournant la valeur de type double du rationnel.
- deux fonctions membres **`num()`, `denom()`** permettant d'accéder au dénominateur et au numérateur du rationnel.
- une fonction membre statique **`int pgcd(int a, int b)`** qui retourne le plus grand diviseur commun entre deux entier.

▲ Modifier la classe pour
- permettre de modifier la valeur d'un rationnel,
- pour que tout rationnel puisse être représenté sous forme irréductible. On fournira une fonction membre **`affiche()`** qui affichera la forme irréductible sous la forme **`a/b`**.

Exercice-2 **:** On désire approcher les dérivées première et seconde de fonction réelles à valeur réelles. Un choix potentiel pour une approximation de la dérivée f' de la fonction f est donnée en chaque point x, pour h assez petit (proche de 0) par :

$$f'(x) \simeq \frac{f(x+h) - f(x-h)}{2h}.$$

Apporter une solution à ce problème en créant une classe **`CFunction`** de sorte que les appels se fassent comme dans le **Listing 1**.

Listing 1 – fichier : derivFunc.cpp

```
int main(int argc, char** argv){
 CFunction Cf;          //< instantiation d'un objet
 Cf.setStep(1e-4);      //< valeur du pas  \a h
 Cf.setFunc(std::cos);//< chargement de la function \a f
 double x;
```

```
  std::cout<<"Entrez un réelle: ";
  std::cin>> x;
  std::cout<<"f  ( "<<x<<" )= "<< Cf.f(x) <<"\n"  //< f(x)
           <<"f\'( "<<x<<" )= "<< Cf.dxf(x) <<"\n" //< approx f'(x)
           <<"f\"( "<<x<<" )= "<< Cf.dxxf(x) <<"\n";//<approx f"(x)
  return 0;
}
```

Exercice-3 : Dans le même esprit que précédemment, On souhaite approcher la valeur d'une intégrale de fonction réelles à valeur réelles sur une intervalle $[a, b]$. On fait fait le choix de subdiviser l'intervalle $[a, b]$ en n sous intervalle et d'applique une formule d'intégration numérique sur chaque intervalle : un exemple est la formule des trapèzes :

$$\int_{x_i}^{x_{i+1}} f(x)\,dx \approx (x_{i+1} - x_i)\frac{f(x_i) + f(x_{i+1})}{2}$$

Q-3-1 : En vous inspirant de la structure de la classe **`CFunction`**, concevoir une classe **`Integrateur`** telle que l'utilisation puisse se faire comme dans le **Listing 2**

Listing 2 – fichier : integrator.cpp

```
int main(int argc, char** argv){
double a = 0, b = 1;
 Integrateur integ(a,b,std::cos);   //< instantiation d'un objet
 std::cout<<"Integrale de cosinus  de "
          << integ.borne(0)
          <<"  à " <<
          << integ.borne(1)
          <<" avec  100 pas est"
          << integ.execute()
          <<"\n";

 integ.setBornes(1,2); //< changement de bornes d'intégration
 integ.setFunc(std::sin); //< changement de function

 std::cout<<"Integrale de sinus  de "
          << integ.borne(0)
          <<"  à " <<
          << integ.borne(1)
          <<" avec  200 pas est"
          << integ.execute(200)
          <<"\n";
 return 0;
}
```

Q-3-2 : Pouvez-vous modifier la classe **`Integrateur`** de sorte à lui fournir la formule d'intégration approchée ?

Thème - 2 *Abstraction - classes - Opérateurs - Opérations*

Exercice-1 **:**

- Créer une classe qui contient un **`int`** (un entier).
- Implémenter l' **`operator<<`** pour l'afficher.
- Tester la solution.

Exercice-2 **:**

- Créer une classe contenant un **`int`**.
- Ajouter une méthode pour l'afficher.
- Implémenter les opérateurs **++** préfixe et suffixe incrémentant cet entier.
- Prouver que la solution fonctionne.

Exercice-3 **:** On considère le programme suivant :

Listing 3 – fichier : temporaryobjects.cpp

```
#include <iostream>
using namespace std;
struct C{};
C f(){return C();}
void g(C& c){}
int main(int argc, char** argv)
{
  g(f());
return 0;
}
```

- Ce programme compile-t-il ? Justifier.

Exercice-4 **:** On considère le programme suivant :

Listing 4 – fichier : assignation.cpp

```
#include <iostream>
using namespace std;
class C{
  int id;
public:
  C(int i): id(i) {cout<<" C::C(int) : "<<id <<endl;}
  C& operator=(const C& c){
    id =c.id;
    cout<<"C::operator =(const C&) : "<<id<<endl;
  return *this;
```

```
  }
};

int main(int argc, char** argv)
{
  C c1(1);
  C c2 = c1;
return 0;
}
```

- Qu'affiche-t-il ? Justifier.
- Il y a une erreur grave. Corriger.

Exercice-5 : Écrire la classe **R2**.

▲ Utiliser un tableau de 2 **doubles** pour stocker x et y

▲ Coder les fonctions d'accès à x et y de deux manières :

- créer les fonctions

  double& x()
  double& y()
  const double& x() const
  const double& y() const

- ainsi qu'un accès par numéro de composante en surchargeant l'opérateur [].

▲ Programmer les opérations standards (somme, différence, ajout, retranchement, produit scalaire, produit vectoriel, produit par un scalaire, . . .).

Démontrer le fonctionnement des fonctions écrites dans le programme principal.

Thème - 3 *Encapsulation - données et comportements*

Exercice-1 : Répondre aux commentaires et remarques donnés dans le programme suivant :

Listing 5 – fichier : methodeprivees.cpp

```
#include <iostream>
//=========================================================
// CLASS
//=========================================================
class ObjC
{
 int* ptr;
 public:
   ObjC(int& a):ptr(&a){;}
   int& refData()const {return *ptr;}
   int*& refPtrData(){   return ptr;}
   int* ptrData()const{return ptr;}
};
//=========================================================
// EVALUATION
```

```
//=========================================================
using std::cout;
int main()
{
  int a = 2;
  int b = 3;
  ObjC o(a);
  cout<<" avant a "<< a<<"\n";
  o.refData() = b;                  // Expliquer pourquoi ceci marche
  cout<<" apres a "<< a <<"\n";
  o.refPtrData()= &b;               // Expliquer pourquoi ceci marche
  cout<<" apres a "<< o.refData() <<"\n";

  const ObjC oo(b);
  //oo = o;                         //Expliquer pourquoi ceci ne ↩
     marchera pas
  cout<<" avant b "<< b<<"\n";
  * oo.ptrData() = 2;               //Expliquer pourquoi ceci marche
  // oo.ptrData() = &b;             //Expliquer pourquoi ceci ne ↩
     marchera pas
  cout<<" apres b "<< b <<"\n";
   return 0;
}
```

Exercice-2 : On donne le programme suivant

Listing 6 – fichier : methodeprivees.cpp

```
#include <iostream>
using namespace std;
//-----------------------------------------------------------
class Foo{
 private:
  Foo(){}
  ~Foo(){}
};
//-----------------------------------------------------------
int main(int argc, char** argv)
{
  Foo C;
return 0;
}
```

- Expliquer pourquoi ce programme ne compile pas.
- Proposer deux manières de créer un objet de la classe **`Foo`**, sans changer de modificateur.

Thème - 4 *Heritage -*

Exercice-1 : On considère le programme suivant :

Listing 7 – fichier : objectsplicing.cpp

```
#include <iostream>
using namespace std;
//-------------------------------------------------------------
class C{
public:
  virtual void f()const{cout <<"C::f()" << endl;}
};
//-------------------------------------------------------------
class D: public C{
public:
  virtual void f()const{cout <<"D::f()" << endl;}
};
//-------------------------------------------------------------
void f(C c){
  c.f();
  D* d = reinterpret_cast<D *> (&const_cast<C&>(c)); //cast le plus↩
      violent; a eviter autant que possible
  (*d).f();
}
//-------------------------------------------------------------
int main(int argc, char** argv)
{
  D d; f(d);
return 0;
}
```

- Qu'affiche-t-il ? Expliquer.

Exercice-2 : On considère le programme suivant :

Listing 8 – fichier : incompleteConstruct.cpp

```
class C{
private:
  int id;
public:
  C(int i){id = i;}
};
//-----------------------------------
class D:public C{
public:
  D(){}
};
//-----------------------------------
int main(){ }
```

- Ce programme ne compile pas. Expliquer.
- Proposer 3 corrections possibles.

Exercice-3 : On considère le programme suivant :

Listing 9 – fichier : incompleteConstruct.cpp

```cpp
#include <iostream>
using namespace std;
class C{
public:
  virtual void f(){cout<< "C::f()"<<std::endl;}
};

class D:private C{
public:
 void f(){C::f(); cout<< "D::f()"<<std::endl;}
 void g(){ C* c = this;}
};

int main(){
  D d;
  d.f();
  d.g();
  C* c = &d;
  return 0;
  }
```

- Expliquer ce qui fonctionne et ce qui ne fonctionne pas.

Exercice-4 : On considère le programme suivant :

Listing 10 – fichier : heritageDiamant.cpp

```cpp
#include <iostream>
class C{
public:
  C(){std::cout << "C::C() - ";}
};
//----------------------------------------
class C1:public C{
public:
  C1(){std::cout << "C1::C1() - ";}
};
//----------------------------------------
class C2:public C{
public:
  C2(){std::cout << "C2::C2() - ";}
};
//----------------------------------------
class D:public C1,public C2{
public:
  D(){std::cout << "D::D()"<<std::endl;}
};
//----------------------------------------
int main(int argc, char** argv){
 D d;
}
```

- Qu'affiche-t-il ?

- Est-ce correct ? Justifiez.
- Corriger si nécessaire.

Thème - 5 *Polymorphisme -*

Exercice-1 : On considère le programme suivant :

Listing 11 – fichier : virtualImpl.cpp

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

class C{
public:
  virtual void f()=0{cout << "C::f()"<< endl;}
};

class D:public C{
public:
  void f(){ C::f(); cout<< "D::f()" << endl;}
};

int main(int argc, char** argv){
 D d;
 d.f();
}
```

- Compile-t-il ? Justifier.
- Corriger si nécessaire.

Exercice-2 : On considère le programme suivant :

Listing 12 – fichier :virtualInConstruct.cpp

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

class C{
public:
C(){f();}
virtual ~C(){f();}
virtual void f(){cout << "C::f()"<< endl;}
};

class D:public C{
public:
  void f(){ cout<< "D::f()" << endl;}
};
```

```
int main(int argc, char** argv){
 C* c = new D();
 c->f();
 delete c;
}
```

- Qu'affiche-t-il ? Justifier.

Exercice-3 : On considère le programme suivant :

Listing 13 – fichier :virtualPseudoConstruct.cpp

```
#include <iostream>
class C{
public:
virtual  C* generate(){return new C();}
virtual void print(){std::cout << "C::print()"<< std::endl;}
};

class D:public C{
public:
   D* generate(){return new D();}
  void print(){ std::cout<< "D::print()" << std::endl;}
};

int main(int argc, char** argv){
 C c1;
 D d1;
 C* t[] = {&c1,&d1};
 C* pc1 = t[0]->generate();
 C* pc2 = t[1]->generate();
 pc1->print();
 pc2->print();
 delete pc1;
 delete pc2;
}
```

- Compile-t-il ?
- Respecte-t-il le principe du polymorphisme ? justifier.
- S'il compile, qu'affiche-t-il ?