TP C++

PR. Houda Mouttalib

Plan

1. Comparaison langage c et c++
2. Classes et objets
3. Constructeurs et destructeurs
4. Operateur New et Delete
5. Surcharge des fonctions
6. Surcharge des operateurs
7. Fonction amies et Classe amies
8. Héritage simple et Héritage multiple
9. Polymorphisme

Comparaison langage c et c++

Classes et objets

```
class Point {
private:
    double x;
    double y;
public:
    Point() {
        this->x = 0;
        this->y = 0;
    }
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
};
int main() {
    Point p1, p2(2,-4);

    return 0;
}
```

Constructeurs et destructeurs

```
public:
    Point() {
        this->x = 0;
        this->y = 0;
    }
    Point(double x, double y) {
        this->x = x;
        this->y = y;
    }
    Point(const Point& p) {
        this->x = p.x;
        this->y = p.y;
    }
};
```

```
int main() {
    Point p1, p2(2,-4);
    Point p3(p2);

    return 0;
}
```

```
~Point(){}
```

Operateur New et Delete

```
PileChar() {
    max = 50;
    sommet = 0;
    pile = new char[max];
}
```

```
~PileChar() {
    delete[] this->pile;
}
```

Surcharge des fonctions

Élément de déclaration de fonction	Utilisé pour surcharger ?
Type de retour de fonction	**Non**
Nom de la fonction	Oui
Nombre d’arguments	Oui
Type d’arguments	Oui

Surcharge des operateurs

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

struct Complex {
   Complex( double r, double i ) : re(r), im(i) {}
   Complex operator+( Complex &other );
   void Display( ) {   cout << re << ", " << im << endl; }
private:
   double re, im;
};

// Operator overloaded using a member function
Complex Complex::operator+( Complex &other ) {
   return Complex( re + other.re, im + other.im );
}

int main() {
   Complex a = Complex( 1.2, 3.4 );
   Complex b = Complex( 5.6, 7.8 );
   Complex c = Complex( 0.0, 0.0 );

   c = a + b;
   c.Display();
}
```

Fonction amies

```
#include <iostream>

using namespace std;
class Point
{
    friend void ChangePrivate( Point & );
public:
    Point( void ) : m_i(0) {}
    void PrintPrivate( void ){cout << m_i << endl; }

private:
    int m_i;
};

void ChangePrivate ( Point &i ) { i.m_i++; }

int main()
{
   Point sPoint;
   sPoint.PrintPrivate();
   ChangePrivate(sPoint);
   sPoint.PrintPrivate();
// Output: 0
           1
}
```

Classe amies

```cpp
#include <iostream>

using namespace std;
class YourClass {
friend class YourOtherClass;  // Declare a friend class
public:
   YourClass() : topSecret(0){}
   void printMember() { cout << topSecret << endl; }
private:
   int topSecret;
};

class YourOtherClass {
public:
   void change( YourClass& yc, int x ){yc.topSecret = x;}
};

int main() {
   YourClass yc1;
   YourOtherClass yoc1;
   yc1.printMember();
   yoc1.change( yc1, 5 );
   yc1.printMember();
}
```

Héritage simple

```
class Document {
public:
   char *Name;   // Document name.
   void PrintNameOf();   // Print name.
};

// Implementation of PrintNameOf function from class Document.
void Document::PrintNameOf() {
   cout << Name << endl;
}

class Book : public Document {
public:
   Book( char *name, long pagecount );
private:
   long  PageCount;
};

// Constructor from class Book.
Book::Book( char *name, long pagecount ) {
   Name = new char[ strlen( name ) + 1 ];
   strcpy_s( Name, strlen(Name), name );
   PageCount = pagecount;
};
```

Héritage multiple

```
class ordinateurportable: public ordinateur, public bagage {
    float poids1;
    float epaisseur;
public:
    ordinateurportable(float pd1 = 0.0, float epaiss = 0.0, int rm = 0, int
        tp1 = "", float pd = 0.0): ordinateur(rm, hd), bagage(tp1, pd) {
        poids1 = pd1;
        epaisseur = epaiss;
    }
    void affiche() {
        ordinateur::affiche();
        bagage::affiche();
        cout << "ordinateur portable poids: " << poids1 << endl;
        cout << "epaisseur: " << epaisseur << endl;
    }
};
```

Polymorphisme