Lab of Object-Oriented Programming:

Operator Overloading & Midterm Review

黃威、陳岳紘、邱彥翔

2022

課程規定

- 寄 E-mail 問問題的格式
  - 標題 : [OOP111] + 問題
  - 內文必須包含 系級學號姓名
  - 請附上 有問題的部分 的程式碼或截圖
- 作業
  - 作業可以討論但是不能抄襲 作業 相似度過高者一律 0 分
  - 修改資料夾權限防止作業被 偷看

重點整理

- Operator Overloading
- 動態記憶體配置
- Function
- Class

Operator Overloading

多載 (Overload)

- 讓接收不同參數的函式可以共用名稱
  - 編譯器自動判斷呼叫的函式
- 三種多載
  - 建構式多載 (Constructor overloading)
  - 函式多載 (Function overloading)
  - 運算子多載 (Operator overloading)

運算子多載

- 定義 class 使用運算子時的動作
  - 加減乘除、大於小於等

運算子多載範例

```
class Point2D {
  private:
   int _x;
   int _y;

  public:
   Point2D();
   Point2D(int, int);
   int x() { return _x; };
   int y() { return _y; };
   Point2D operator+(const Point2D&);  //定義加號
   Point2D operator-(const Point2D&);  //定義減號
   Point2D& operator++();              //定義 ++（前置） 如++p
   Point2D operator++(int);            //定義 ++（後置） 如--p
   Point2D& operator--();
   Point2D operator--(int);
   Point2D& operator+=(Point2D&);      //定義 +=
   Point2D& operator-=(Point2D&);
   bool operator==(const Point2D&);                      //定義 ==
   friend ostream& operator<<(ostream&, const Point2D&);  //定義 cout
};
```

運算子多載實作

```
Point2D Point2D::operator+(const Point2D& other) {
    Point2D result(this->_x + other._x, this->_y + other._y);
    return result;
}
```

運算子多載實作

```
Point2D& Point2D::operator++() {
    this->_x++;
    this->_y++;
    return *this;
}

Point2D Point2D::operator++(int) {
    Point2D result = (*this);
    this->operator++();
    return result;
}
```

運算子多載實作

```
Point2D& Point2D::operator+=(Point2D& other) {
    this->_x += other._x;
    this->_y += other._y;
    return *this;
}
```

運算子多載實作

```
ostream& operator<<(ostream& os, const Point2D& point2D) {
    os << "(" << point2D._x << ", " << point2D._y << ")";
    return os;
}
```

運算子多載實作

```
bool Point2D::operator==(const Point2D& other) {
   return (this->_x == other._x && this->_y == other._y) ? true : false;
}
```

- https://onlinegdb.com/4rkys9P2F

測試執行結果

```
int main() {
    Point2D p1(2, 3);
    Point2D p2(4, 5);
    Point2D p3(3, 3);
    cout << p1 << " " << p2 << endl;
    cout << p1 + p2 << endl;
    cout << p1-- << endl;
    cout << p1 << endl;
    p1 += p3;
    cout << p1 << ", " << (p1 == p2 ? "p1 == p2" : "p1 != p2") << endl;
}
```

```
(2, 3) (4, 5)
(6, 8)
(2, 3)
(1, 2)
(4, 5), p1 == p2
```

動態記憶體配置

動態記憶體配置

- 宣告指標
- 讓指標指向動態要求的記憶體空間
- 使用 new
  - 類似 c 語言的 malloc
- 使用 delete 手動釋放記憶體

new

- `int* var1 = new int;` //宣告一個 int
- `int* var2 = new int(5);` //宣告一個 int 並設定初始值
- `int* arr1 = new int[5];` //宣告一個 int 陣列
- `int* arr2 = new int[5]{1, 2, 3, 4, 5};` //宣告一個有初始值的 int 陣列

delete

- 釋放變數
  - `delete var1;`
- 釋放陣列
  - `delete[] arr1;`

Shallow Copy & Deep Copy

Shallow Copy & Deep Copy

- Shallow Copy
  - 所有變數 無論是不是指標 全部都複製到目的物件
  - c++ 通常預設使用 Shallow Copy
- Deep Copy
  - 如果是指標 就將指標中的物件複製一份
  - 通常需要自己實作

三大法則

- 只要實作其中一個 另外兩個缺一不可
  - 解構式 (Destuctor)
  - 複製建構式 (Copy constructor)
  - 複製運算子多載 (Copy assignment overloading)

三大法則

```
class Point2D {
  private:
   int _x;
   int _y;

  public:
   Point2D();
   Point2D(int, int);
   ~Point2D();                              //解構子
   Point2D(const Point2D&);                 //複製建構子
   Point2D& operator=(const Point2D&);  //複製運算子多載
```

三大法則

```
Point2D::Point2D(const Point2D& other) {
    this->_x = other._x;
    this->_y = other._y;
}

Point2D& Point2D::operator=(const Point2D& other) {
    this->_x = other._x;
    this->_y = other._y;
    return *this;
}
```

函式

函式

- Inline function
  - 編譯器會視情況在編譯時將 function 展開
  - 加快執行速度
  - 和 macro 的比較

const

- Const 參數
  - `void fun0(const int* a);`
    - 讓指標指向的位置的內容不能被修改
  - `void fun1(const int& a);`
    - 讓 reference 參照的位置的內容不能被修改

const

- class 中的 const 成員函式
  - `void func() const;`
  - 讓成員函式不能修改物件中的變數
  - const 函式只能呼叫 const 函式

const

- const 指標
  - `int* const p1;`
    - 指標本身是 const，指向的變數可以被修改
  - `const int* p2;`
    - 指標指向的變數是 const
  - `const int* const p3;`
    - 指標和指向的變數都是 const

static 成員

- 在 class 被宣告的當下就會存在
  - 不論有沒有建立物件
- 存在於 class 中而不是物件中
  - 從任何物件中存取都會是一樣的 內容
- static 函式只能存取 static 變數和函式

`friend` 類別

- 讓其他類別的 object 可以存取這個類別的 private 成員
- friend 是單向的 = =

Assign0 ~ 2 解答

/usr/local/class/oop/sample

Assignment 4

```
Testing constructor
  default constructor - Vector::Vector();
        Vector v1 = (0, 0)
        length = 0
  constructor - Vector::Vector(int size);
        Vector v2(5) = (0, 0, 0, 0, 0)
        length = 0
  constructor - Vector::Vector(int size, double elem[]);
        Vector v3(3, {1.1,2.2,3.3}) = (1.1, 2.2, 3.3)
        length = 4.11582
  copy constructor - Vector::Vector(const Vector& v);
        Vector v4 = v3, v4 = (1.1, 2.2, 3.3)

Testing get/set Dimension
  v4.getDimension() = 7
  after v4.setDimension(7) , v4.getDimension() = 7
        v4 = (1.1, 2.2, 3.3, 0, 0, 0, 0)
  after v4.setDimension(2) , v4.getDimension() = 2
        v4 = (1.1, 2.2)

Testing Assignment operator
        Vector v5(2, {6.6, 7.7}) = (6.6, 7.7)
        Vector v6; v6 = v5; v6 = (6.6, 7.7)
```

```
Testing Arithmetic operators
  -(negate):    -v3 = (-1.1, -2.2, -3.3)
  +:      v4 + v5 = (7.7, 9.9)
  -:      v4 - v5 = (-5.5, -5.5)
  *:      v4 * 2.1 = (2.31, 4.62) length = 5.16532
  *:      3.4 * v4 = (3.74, 7.48) length = 8.36289
  +=:     v6 += v5; v6 = (13.2, 15.4)
  -=:     v5 -= v4; v5 = (5.5, 5.5)
  *=:     v4 *= 7.6; v4 = (8.36, 16.72)

Testing Equality Operators
  v4 = (8.36, 16.72)
  v5 = (5.5, 5.5)
        ==: v4 == v5 :false
        !=: v4 != v5 :true
  after v4 = v5
        ==: v4 == v5 :true
        !=: v4 != v5 :false

General Testing
  Vector v7; v6 *= (v6 -= (v4 + v6[0] * (v7 = -v5)))[0];
        v6 = (6448.09, 6624.75) length = 9244.74
```

Assignment 4 評分標準

有交(含屍體)	20
可以make	10
有學號	10
每個section (10%) constructor, get dimension, operator arithmetic operation, equality operation general testing	60

Q&A