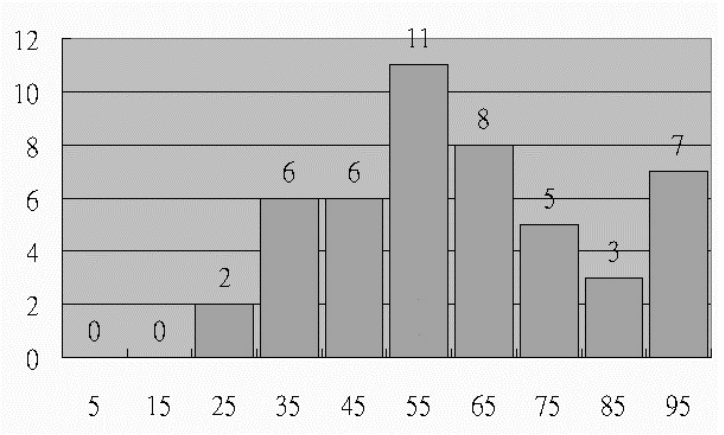
942 Midterm Exam Discussions

C++ Object Oriented Programming
Pei-yih Ting
95/05 NTOU CS

成績分佈



Problem #1

a. 分開來存放在 .h 和 .cpp 的檔案有什麼好處?
   1. 各類別分開存放：容易管理和修改，修改後也可以節省重新編譯的時間
   2. .h 和 .cpp 檔案分開：只有 .h 檔案的內容是給其它模組引入的，.cpp 檔案的內容則不是，所以必須分開存放在不同的檔案中

b. C++語言中, 類別層次, 封裝 (Encapsulation)" 特性
   1. 類別中需要製作介面函式，所有的資料在使用時都透過這一組介面函式，常常這一組介面函式是以服務為導向的
   2. 強制將資料放在 private 區段中，將介面函式放在 public 區段中，如此編譯器可以協助檢查出不透過介面函式的資料存取

Problem #1 (cont'd)

c. 撰寫程式時程式設計者在心裡面常常對資料內容有一些假設，例如完成配置記憶體的動作後你會假設有指定大小的記憶體可以使用；完成開啟檔案的動作後你會假設檔案已經可以存取；完成某一個資料排序的動作過後你會假設資料已經按順序排好；使用某一個類別時你會假設它的功能和它的說明是一致的；使用某一個變數時也許你假設變數存放的資料不是負數...

類似的假設非常多，很多都是隱藏起來，甚至不小心時都不會注意到，有些假設在設計完程式一段時間後你就會忘記了，這些都可以運用 assert() 敘述來記錄，如果是正常應該考慮的邏輯就應該用 if-then-else 來寫，如果是不常出現的就應該用 assert() 來寫

Problem #1 (cont'd)

d. inline 的意義?為什麼要用?

```
01 inline double square(double x)
02 {
03     return x * x;
04 }
```

inline 是告訴編譯器在翻譯這個程式中呼叫這個函式的敘述，例如 r = square(x+y);，時將這個敘述換成類似 z = x + y; r = z * z; 這樣子等效的敘述，不要做函式的呼叫，這樣子的作法很像是 C 裡面的 macro #define square(x) ((x)*(x));, 以增進程式執行時的效率，但是又不會有 C 裡面 macro 遇見的語法問題，注意 inline 函式的宣告只對於同一個檔案內呼叫 square 的敘述有作用

Problem #1 (cont'd)

e. const 的意義為何? (為什麼要用 const 呢? 如此宣告的成員函式在撰寫時有什麼要注意的?)

為一個 const function 的宣告，這樣子的函式裡面規定不可以修改物件的狀態 (物件所有的資料成員)，宣告 const function 是對將來使用這個函式的程式的一種保障，呼叫這樣的函式之後物件內的狀態絕對不會有變化，不會有任何的 side effect

```
01 class Quoter
02 {
03 public:
04     Quoter();
05         int lastQuote() const;
06 private:
07     int m_lastQuote;
08 };
```


Problem #2

a. 違背了物件導向程式設計的什麼原則? 該如何修改?

```
01 class Book
02 {
03 public:
04     Book(string title, string author);
05     string getTitle();
06     string getAuthor();
07 private:
08     string m_title;
09     string m_author;
10 };
11
12 ...
13 Book book("Harry Potter", "J. K. Rowling");
14 cout << "Title:" <<book.getTitle() << endl;
15 cout << "Author:" << book.getAuthor() << endl;
```


Problem #2 (cont'd)

基本上間接地違背了**封裝**的原則，雖然沒有直接地違背：客戶端直接存取物件內的資料，但是透過 Accessor 函式來把所有資料送出這個 Book 類別的物件，還是不恰當的，如果能夠避免的話應該盡量避免，以這個例子來說，應該替 Book 類別增加一個 print() 的介面如下：

```
void Book::print()
{
   cout << "Title:" << m_title <<endl;
   cout << "Author:" << m_author << endl;
}
```

14 及 15 列換成 book.print()，如此封裝得比較完全

Problem #2 (cont'd)

b. 運用初始化串列 (initialization list) 完成類別 Book 的建構元函式

```
Book::Book(string title, string author): m_title(title), m_author(author)
{
}
```


Problem #3

a. 說明程式最主要邏輯功能

```
01 #include <iostream>
02 #include <vector>
03 #include <string>
04 using namespace std;
05
06 int main()
07 {
08    ifstream inf("inputData.txt");
09    string line;
10    vector<string> lines;
11
12    while (getline(inf, line))
13       lines.push_back(line);
14
15    for (int i=0; i<lines.size(); i=i+2)
16       cout << i << ":" << lines[i] << endl;
17
18    return 0;
19 }
```

inputData.txt

```
January
February
March
April
May
June
July
August
September
October
November
December
```

```
0: January
2: March
4: May
6: July
8: September
10: November
```

開啟 inputData.txt 檔案 12 及 13 列由檔案中以列為單位依序將字串讀入程式中，存放在 lines 容器物件中 15 及 16 列將奇數列列印在螢幕上，印出資料如上

Problem #3 (cont'd)

b. 程式中哪些地方可能有呼叫建構元函式?

第 8 列呼叫 ifstream(char *) 或是 ifstream(string) 建構元函式 1 次

第 9 列呼叫 string() 建構元函式 1 次

第 10 列呼叫 vector<string>() 建構元函式 1 次

第 13 列呼叫 string(string &) 建構元函式 12 次

第 18 列之前會反序解構 12 次 string 類別的物件lines[i]，呼叫 ~string() 解構元

解構 1 次 vector<string> 類別的物件 lines

解構 1 次 string 類別的物件 line

解構 1 次 ifstream 類別的物件 inf

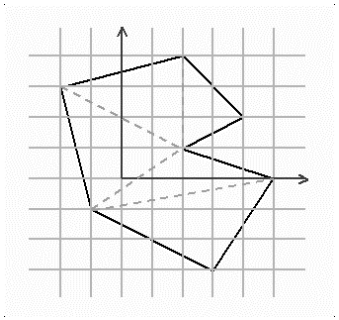
Problem #3 (cont'd)

c. 請使用 vector<string> 類別的 iterator 來存取容器物件的內容，撰寫一小段程式取代第 15 列和第 16 列

```
vector<string>::iterator iter;
for (iter=lines.begin(), i=0; iter<lines.end(); iter+=2, i+=2)
   cout << i << ":" << *iter << endl;
```

Problem #4

簡易繪圖程式架構撰寫



主程式

```
ifstream inputFile("data.txt");
Polygon graphicObject(inputFile);
graphicObject.draw();
```

資料檔案

Polygon 1	Triangle 2	Triangle 4
5	2 1	2 1
Triangle 1	2 4	-1 –1
4 2	-2 3	5 0
2 4	blue	blue
2 1	Triangle 3	Triangle 5
red	2 1	-1 –1
	-2 3	3 –3
	-1 –1	5 0
	green	yellow

Problem #4 (cont'd)

Point 類別

```
// Point.h
#ifndef POINT_H
#define POINT_H
#include <fstream>
using namespace std;
class Point
{
public:
   Point (ifstream &);
   int getX();
   int getY();
private:
   int m_x, m_y;
};
#endif
```

```
// Point.cpp
#include "Point.h"
Point::Point(ifstream &infile)
{
   infile >> m_x >> m_y;
}
int Point::getX()
{
   return m_x;
}
int Point::getX()
{
   return m_x;
}
```


Problem #4 (cont'd)

Triangle 類別

```
// Triangle.h
#ifndef TRIANGLE_H
#define TRIANGLE_H
enum Color {red, green, blue, yellow,
            cyan, magenta};
#include <fstream>
using namespace std;
class Triangle
{
public:
   Triangle(ifstream &);
   void draw();
private:
   Point m_point1, m_point2, m_point3;
   Color m_color;
};
#endif
```

```
// Triangle.cpp
#include "Triangle.h"
#include "Point.h"
#include <string>
using namespace std;
Triangle::Triangle(ifstream &infile):
   m_point1(infile), m_point2(infile),
   m_point3(infile)
{
   char cbuf[100]; string buf;
   infile.getline(cbuf, 99, '\n'); buf = cbuf;
   if (buf == "red") m_color = red;
   else if (buf == "green") m_color = green;
   else if (buf == "blue") m_color = blue;
   else if (buf == "yellow") m_color = yellow;
   else if (buf == "cyan") m_color = cyan;
   else if (buf == "magenta") m_color = magenta;
}
```


Problem #4 (cont'd)

```
// Triangle.cpp continued
void Triangle::draw()
{
   drawTriangle(m_point1.getX(), m_point1.getY(),
            m_point2.getX(), m_point2.getY(),
            m_point3.getX(), m_point3.getY(),
            m_color);
}
```

Problem #4 (cont'd)

✧ Polygon 類別

```
// Polygon.h
#ifndef POLYGON_H
#define POLYGON_H
#include <fstream>
#include <vector>
using namespace std;
class Triangle;
class Polygon
{
public:
   Polygon(ifstream &);
   void draw();
   ~Polygon();
private:
   vector<Triangle *> m_triangles;
};
#endif
```


Problem #4 (cont'd)

```
// Polygon.cpp
#include "Polygon.h"
#include "Triangle.h“
#include <fstream>
using namespace std;
Polygon::Polygon(ifstream &infile)
{
   Triangle *tmp;
   char buf[100];
   infile.getline(buf, 99, '\n');
   int nTriangles;
   infile >> nTriangle;
   infile.getline(buf, 99, '\n');
   for (int i=0; i<nTriangles; i++)
   {
      infile.getline(buf, 99, '\n');
      tmp = new Triangle(infile);
      m_triangles.push_back(tmp);
   }
}
```

```
void Polygon::draw()
{
   vector<Triangle *>::iterator iter;
   for (iter=m_triangles.begin();
        iter<m_triangles.end(); iter++)
      (*iter)->draw();
}
Polygon::~Polygon()
{
   vector<Triangle *>::iterator iter;
   for (iter=m_triangles.begin();
        iter<m_triangles.end(); iter++)
      delete (*iter);
}
```


Summary

✧ 本次考試主要評估下列概念與語法：

- ⋆ 物件
- ⋆ 建構元/解構元/初始化串列的設計與呼叫時機
- ⋆ const function 與 inline function
- ⋆ 封裝與物件分工
- ⋆ 類別的設計
- ⋆ 動態配置記憶體 new / delete
- ⋆ vector 類別的使用以及 vector::iterator 類別的使用
- ⋆ ifstream, fstream 輸入
- ⋆ ostream 輸出
- ⋆ 運用建構元讀取物件狀態的方法
- ⋆ polygon::draw() 呼叫 triangle::draw() 的作法是一種標準的 “委託 (delegation)”