國立台灣海洋大學資訊工程系 2A C++ 程式設計 期末考 參考答案

姓名：_____

系級：_____

學號：_____

105/06/21 (二)

考試時間：**09:30 – 12:00**

考試規則：1. 不可以翻閱參考書、作業及程式
2. 不可以使用任何形式的電腦 (包含手機、計算機、相機以及其它可運算或是連線的電子器材)
3. 請勿左顧右盼、請勿交談、請勿交換任何資料、試卷題目有任何疑問請舉手發問 (看不懂題目不見得是你的問題, 有可能是中英文名詞的問題)、最重要的是隔壁的答案可能比你的還差, 白卷通常比錯得和隔壁一模一樣要好
4. 提早繳卷同學請直接離開教室，請勿逗留喧嘩
5. 違反上述任何一點之同學一律依照學校規定處理
6. 繳卷時請繳交***簽名過之試題卷及答案卷***

1. 根據下圖中 Element 及 Stack 類別回答相關問題

```
1.  class Element
2.  {
3.  public:
4.      Element(char *name);
5.      virtual ~Element();
6.      virtual void display(ostream &os);
7.  private:
8.      char *m_name;
9.  };
10.
11. class Stack
12. {
13.     struct Node
14.     {
15.         Element* data;
16.         Node* next;
17.         Node(Element* _data, Node* _next);
18.         ~Node();
19.     };
20.     Node m_head;
21. public:
22.     Stack();
23.     ~Stack();
24.     void push(Element* data);
25.     Element* top();
26.     Element* pop();
27. };
28.
```

```
29. Stack::Node::Node(Element* _data, Node* _next)
30.     : data(_data), next(_next)
31. {}
32.
33. Stack::Node::~Node(){ delete data; delete next; }
34.
35. void Stack::push(Element* data)
36. {
37.     m_head.next = new Node(data, m_head);
38. }
39.
40. Element* Stack::top()
41. {
42.     if (m_head.next != 0)
43.         return (m_head.next)->data;
44.     else
45.         return 0;
46. }
47.
48. void Stack::pop()
49. {
50.     if (m_head.next == 0) return;
51.     Node* toBeDeleted = m_head.next;
52.     m_head.next = (m_head.next)->next;
53.     toBeDeleted->data = toBeDeleted->next = 0;
54.     delete toBeDeleted;
55. }
```

a. [5] 請製作 Stack 類別的建構元 (constructor), 請運用初始化串列 (initialization list) 初始化 m_head 成員物件

Sol.

```
Stack::Stack() : m_head(0,0) {}
或是
Stack::Stack() : m_head(Node(0,0)) {}// Node 類別的拷貝建構元直接使用預設的
```

b. [5] 請製作 Stack 類別的解構元 (destructor), 請刪除 m_head 串列中所有節點以及節點所指向的資料

Sol.

```
Stack::~Stack() {} // m_head 解構時會遞迴刪除全部
```

c. [5] 請為 Stack 類別定義一個拷貝建構元 (copy constructor) Stack(const Stack &), 以便拷貝 Stack 類別的物件

Sol.

```
Stack::Stack(const Stack &src)
{
	Node *ptrSrc = src.m_head.next;
	Node *ptrDest = &m_head;
	while (ptrSrc!=0)
	{
		ptrDest->next = new Node(new Element(ptrSrc->data), 0);
		ptrDest = ptrDest->next;
		ptrSrc = ptrSrc->next;
	}
}
//  此處假設 Element 類別有適當的拷貝建構元, 例如
Element::Element(const Element &src):m_name(new char[strlen(src.m_name)+1])
{
	strcpy(m_name, src.m_name) ;
}
```

d. [5] 請替 Stack 類別定義一個列印內容的 dump(ostream &os) 成員函式, 請運用 Element::display() 來輸出個別元素的內容, 並且撰寫一個全域的 ostream& operator<<(ostream &os, Stack &stack) 函式呼叫這個 dump() 成員函式來列印這個類別的內容.

Sol.

```
void Stack::dump(ostream &os)
{
	Node *ptr = m_head.next;
	while (ptr!=0)
	{
		ptr->data->display(os);
		ptr = ptr->next;
	}
}
ostream& operator<<(ostream &os, Stack &stack)
{
	stack.dump(os);
	return os;
}
```

e. [10] 我們希望同一個 Stack 物件裡面可以放不同種類的 ElementOne 或是 ElementTwo 類別的物件, 請將 Element 定義成抽象類別(Abstract Base Class), 由 Element 類別衍生出 ElementOne 類別, 請說明上題的 dump() 函式能不能在不修改的情況下列印衍生類別的物件內容?

Sol.

```
class Element
{
public:
   Element(char *name);
   virtual ~Element();
   virtual void display(ostream &os)=0; // Abstract Base Class
private:
   char *m_name;
};

class ElementOne : public Element
{
public:
   ElementOne(char *name);
   ~ElementOne();
   void display(ostream &os); // override Element::display
};
```

可以完全不用修改, ptr->data->display(os); 這個敘述裡面 ptr->data 是一個多型指標

2. [10] 如果有一個 Base 類別，提供 Base::service()，一個 Derived 類別繼承 Base 類別，提供 Derived::service()，請問如果想要使用 C++ 動態多型的機制，這兩個成員函式應該如何定義，請舉例說明客戶端需要如何使用 Base 或是 Derived 的 service()?

Sol.

1. Derived 類別繼承 Base 類別
2. Base::service() 需要是虛擬函式

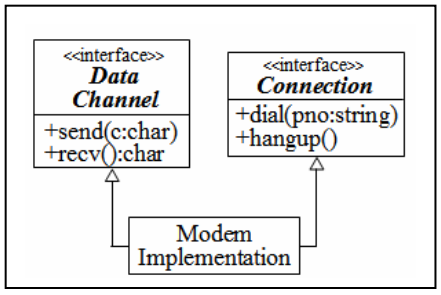
```
class Base {
public:
    virtual void service();
};
class Derived: public Base {
public:
    void service();
};
```

3. 客戶端呼叫的時候需要運用多型指標/參考 (基礎類別的指標/參考)

```
void client1(Base *bp) {
    bp->service(); // 動態多型
}
…
Base bObj; Derived dObj;
client1(&bObj); client1(&dObj);
或是
void client2(Base &b) {
    b.service(); // 動態多型
}
…
Base bObj; Derived dObj;
client2(bObj); client2(dObj);
```

3. [10] 以下列 Modem 類別為例，如果處理連線的功能和處理傳送接收資料的功能可能因為不同的理由分別修改，我們說這個類別有兩種不同的責任，請問這是違反了 OOD 的什麼原則? 該如何修改?

```
1.  class Modem {
2.  public:
3.      void dial(string phoneNo);
4.      void hangup();
5.      void send(char c);
6.      char recv();
7.  };
```



Sol.

SRP, Single Responsibility Principle

如果連線/斷線需求修改時資料傳送/接收需求也會同步修改，資料傳送/接收需求修改時連線/斷線需求也需要同步修改，那麼這個類別就可以看成只有一個責任

題目的描述顯示應該是兩種不同的責任，所以違反了 SRP

如果不會造成兩個耦合度很高的類別，可以嘗試把類別的功能拆為兩個獨立類別，

否則也可以運用獨立的抽象介面來區隔這兩種責任，如上圖右所示，如此類別的客戶不會因為這個類別多重責任的耦合而不斷地修改

4. [10] 在看 Zed Shaw 寫的 Learning Python the Hard Way 這本書的時候，作者說「幾乎所有運用繼承可以得到的程式碼重用的好處都可以用組合來完成，所以儘量不要用繼承的語法來寫程式，尤其可以避免多重繼承的複雜架構」，根據你所知道的繼承和組合的差別，你可以分析一下這樣子講法的問題在哪裡嗎?

Sol.

他應該忽略了介面繼承帶來的重用效果了，繼承的語法可以重用父類別裡的資料與程式，這一部份可以用組合來取代，但是這一部份的重用只佔了非常小的比重，繼承真正強大的威力是在於介面繼承，可以重用所有父類別的客戶端程式

5. [10] 請問 C++私有繼承的語法為何? 效果為何? 請舉例說明為什麼我們說這個語法不算是 OOP 的繼承?

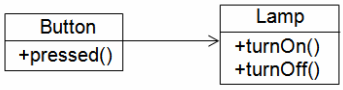
Sol.

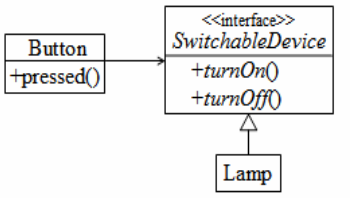
C++ 私有繼承的語法如下

```
class Base
{
};
class Derived: Base  或是  class Derived: private Base
{
};
```

Derived 類別裡自動就有 Base 類別裡的資料和程式, 可以不用重寫, 但是 Base 類別的介面並沒有被 Derived 類別繼承下來, 也就是操作 Derived 類別物件的方法和操作 Base 類別的物件是不一樣的, 所以一個 Derived 類別的物件不能夠當成是 Base 類別的物件來使用, 不滿足 IS-A 的關係。所以私有繼承幾乎完全等效於組合, 需要透過委託來運作。

6. [10] 右圖中 Button 類別會因為 Lamp 類別的修改而需要重新編譯, 但是其實 Button 的動作和 Lamp 的實作幾乎沒有關聯, 只是運用到 turnOn()和 turnOff()兩個介面而已, 這種狀況在設計物件系統時很容易遇到, 我們通常運用什麼 OOD 的法則來除去這種狀況? 請繪圖說明用什麼架構來改善?



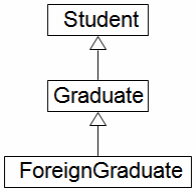


Sol.

DIP, Dependency Inversion Principle

如右圖中新增加的 *SwitchableDevice* 抽象類別以及裡面定義的兩個純粹虛擬函式介面, 如此 Button 類別只和抽象類別 *SwitchableDevice* 有關聯, 不再和 Lamp 類別直接相關, Lamp 類別有任何實作的修改都不會影響到 Button 類別。

7. [10] 在上課的時候談到右圖的繼承架構, ForeignGraduate 類別繼承 Graduate 類別的原因是因為外國學生剛好都是研究所學生, 但是我們的 Graduate 類別裡其實有一個 m_stipend 的欄位, 假設今天外國學生是不允許工作的, 所以也不能夠領取津貼, 如果還是維持這個繼承架構, 但是把這個欄位設成 -1, 請舉例說明這樣的設計會造成什麼問題? 有違反什麼原則嗎?



Sol.

基本上父類別具有的功能很難關閉, 關閉很容易牴觸 LSP, Liskov Substitution Principle, 原本父類別的物件因為有某種特性所以能夠運作, 一旦失去那種特性就沒有辦法運作了, 以上面的例子來說假設原本 Graduate 類別的物件需要課徵所得稅, 繼承下去的類別所作出來的物件都應該能夠課稅, 如果關閉就需要在程式的各個地方判斷是不是屬於這種型態, 程式需要把這種型態和不是這種型態的完全分離開來處理, 但是型態系統卻因為繼承而完全無法區分, 就失去繼承某一個型態的意義了。

8. [10] 請舉例說明什麼時候該使用 try-throw-catch 來處理錯誤狀況, 什麼時候該使用 assert 來處理, 什麼時候該使用 if 來處理?

Sol.

Assert 是程式開發團隊內部確認假設使用的, 在程式拿到消費者手上之前需要完全移除; 剩下的狀況可以再分為正常情況下很容易發生的, 例如十次發生一次以上的, 通常就運用 if 來處理, 發生機率低到百分之一以下的還有一些沒有辦法有回傳值的地方, 通常就運用 try-throw-catch 的例外機制來處理