國立台灣海洋大學資訊工程系 2A　C++ 程式設計 期末考試 參考答案

104/06/23（二）

考試時間：**09:30 – 12:00**

考試規則：
1. 不可以翻閱書本、參考資料、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. 可以用鉛筆作答，答案卷上**請清楚標示題號**
7. 繳卷時請繳交***簽名過之試題卷及答案卷***

1. 請配合右圖程式，設計一個 Item 類別代表超市裡的商品，商品的基本資料包括品名(name)、條碼(id)、價格(price)、折扣(discount)、免稅/應稅(taxFree)、到貨日(date of arrival)、到期日(date of expiration)、貨架位置(aisle)，其中到貨日以及有效日期使用 Date 類別物件，其餘請參考右圖應用範例自行設計適當的資料型態(商品價格為整數，銷售稅稅率請用 5%，四捨五入，請標示檔案名稱，並引入需要的.h 檔案)

```
// date.h
class Date
{
public:
    Date(int month, int day, int year);
private:
    int m_month;
    int m_day;
    int m_year;
};
// date.cpp
Date::Date(int month, int day, int year)
    : m_month(month), m_day(day), m_year(year)
{
}
// main.cpp
int main()
{
    Item uccCoffee("UCC BLACK  無糖咖啡",
                   "4901201226090",
                   45,
                   0.75,
                   false,
                   Date(03,01,2015),
                   Date(02,15,2016),
                   "A5");
    uccCoffee.printPrice(cout);
    return 0;
}
/* ====範例輸出結果====
UCC BLACK  無糖咖啡:35
*/
```

a. [6] 請設計 Item 類別的成員函式與資料成員

Sol:

```
// item.h
#pragma once
#include "date.h"
#include <string>
using std::string;
#include <iostream>
using std::ostream;
class Item
{
public:
    Item(char *name, char *id, int price, double discount, bool taxFree,
        Date doa, Date doe, char *aisle);
    void printPrice(ostream& os);
private:
    string  m_name;
    string  m_id;
    int     m_price;
    double m_discount;
    bool    m_taxFree;
    Date    m_arrivalDate;
    Date    m_expirationDate;
    string  m_aisle;
};
```

b. 配合 main 函式內 uccCoffee 物件的運用，請實作 Item 建構元函式?[6] 以及 Item::printPrice() 成員函式?[6] (引入哪些檔案?[2])

Sol:

```
// item.cpp
#include "item.h"
Item::Item(char *name, char *id, int price, double discount, bool taxFree,
          Date doa, Date doe, char *aisle)
```

```
    : m_name(name), m_id(id), m_price(price), m_discount(discount),
      m_taxFree(taxFree), m_arrivalDate(doa), m_expirationDate(doe), m_aisle(aisle)
{
}

void Item::printPrice(ostream& os)
{
    os << m_name << ":";
    if (m_taxFree)
        os << (int) (m_price*m_discount+0.5);
    else
        os << (int) (m_price*m_discount*1.05+0.5);
}
```

2. a. [6] 請問 C++裡有哪些多型 (polymorphism) 的機制? 用什麼語法來完成?

Sol:

靜態多型 (static polymorphism):

主要就是函式多載 (function overloading) 以及運算子多載 (operator overloading) 的語法

動態多型 (dynamic polymorphism):

虛擬函式 (virtual function) 以及多型指標/參考 (polymorphic pointer/reference) 的語法

參數化多型 (parametric polymorphism):

樣板 (template) 函式以及樣板類別的語法

b. [3] 請問什麼是靜態繫結?

Sol:

編譯器剖析程式以後就可以根據某一個函式呼叫敘述的目標物件、物件指標、物件參考、函式名稱和函式引數來決定執行時需要呼叫哪一個函式

c. [3] C++的多型機制中哪些是運用靜態繫結來完成的? 和動態繫結比最主要好處是什麼?

Sol:

靜態多型和參數化多型都是使用靜態繫結的，最主要的好處是執行速度快

d. [3] 請舉例說明多型(polymorphism)機制對於程式設計者的好處是什麼?

Sol:

多型的機制讓程式設計者用一致的抽象方法來完成對不同資料型態的處理，簡輕程式設計者區分不同資料型態以及對應的動作的負擔，例如 x + y 的'+'號就是多型的運算符號，編譯器會幫忙分辨如果 x 或是 y 有一個是浮點數，'+' 就是浮點數加法，如果 x, y 都是整數，'+'是整數加法，如果 x, y 有一個是記憶體位址，另一個是整數，'+'是記憶體位址加法，但是概念上都是一致的加法

3. [9] 請問在撰寫 void Complex::print(ostream &os) 介面時，我們定義參數是 ostream 的參考，但是在使用時我們卻可以定義 ofstream ofs("outfile.txt"); Complex x; … x.print(ofs); 顯然 print()的參數 ofs 的型態不太一致，請解釋為什麼編譯器可以允許這樣子使用? 請解釋為什麼我們不定義為 void Complex::print(ofstream &os)? 再請問為什麼這個參數要設計為參考變數? (有別的選擇嗎?)

Sol:

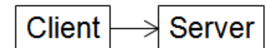
基本上 ofstream 類別描述輸出檔案串流，在 iostream 函式庫中繼承自 ostream 類別，因此 ofstream 類別的物件本來就可以看成是一個基礎類別 ostream 的物件，所以在 Complex 類別中 void

Complex::print(ostream &os)的設計，允許 Complex 類別的客戶程式使用

```
Complex obj;
ofstream ofs("outfile.txt"):
…
obj.print(ofs);
或是
obj.print(cout);
```

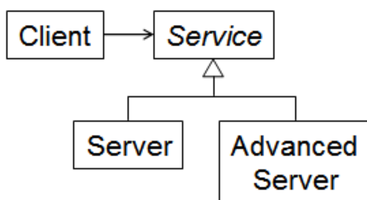
來輸出資料，如果定義為 void Complex::print(ofstream &os)的話，就只能允許檔案串流的輸出了；另外這個參數設計成參考是為了避免離開 print()函式時將輸出串流解構掉，導致後續的輸出發生錯誤; 除了使用參考之外，也可以使用指標 ostream *os，不過語法上會多出許多 dereference 的動作

4. [5] 請問什麼是 Open-Closed Principle(OCP), 請修改右方類別圖簡單的 Client-Server 類別架構來說明如何使得 Server 類別的客戶(Client)類別能夠遵守 OCP?



Sol:

所謂 Open-Closed Principle 是指在設計物件導向系統時，希望每一個部份能夠允許持續地擴充功能(Open for extension)，但是那一部份的模組卻是完全不需要編輯修改的 (Closed for modification)，從程序化程式設計的角度去看是很難看出道理的，在物件導向的設計裡，要達成這個目標通常是透過繼承的語法，把在擴充功能時會持續變動的模組的界面先抽象化出來，以上面 Client-Server 為例，整個 Client-Server 系統的功能可以透過修改 Server 為 AdvancedServer 來加強，但是因為 Client 類別依賴 Server 類別，如果希望這個更改不影響 Client 類別，我們可以用左圖的設計，先抽象化出一個 Service 界面，不管是 Server 還是 AdvancedServer 都具有相同的操作方法，如此在設計 AdvancedServer 擴充整體功能時，Client 類別只依賴不變動的 Service 抽象類別，所以完全不需要修改，藉由繼承 Service 抽象界面類別來設計 AdvancedServer，可以重用所有 Service 界面的客戶，但是又不需要更改這些客戶，這就是 Open-Closed Principle 的應用



5. 接續題 1，由於商品的折扣常常變動，打折的方式也會有很多的變化(也許和購物時段相關，也許和會員種類相關)，假設折扣只和商品的分類有關(目前只有兩類：農產品和日用品)；在題 1 的程式中如果把折扣設計成商品類別裡一個資料成員的話，很難應付各種不同的折扣計算方式；比較有彈性的方法也需要在修改折扣方式時完全不需要更動已經測試過、正在運作中的程式，純粹只是新增一個類別。請配合右側 main()函式，修改並且衍生 Item 類別的定義，設計 Discount 類別：

   a. [3] 請設計抽象 Discount 介面類別提供 double getDiscount(AgriculturalProduct &) 以及 double getDiscount(DailyCommodity &) 介面，以使其客戶 Item 類別符合 OCP

```
int main() {
    Item *item1 = new
        AgriculturalProduct ("福樂北海道一番鮮鮮乳",
                             "4716873932091",
                             145,
                             Date(06,18,2015),
                             Date(06,29,2015),
                             "3B");
    Item *item2 = new
        DailyCommodity("UCC BLACK 無糖咖啡",
                       "4901201226090",
                       45,
                       Date(03,01,2015),
                       Date(02,15,2016),
                       "5A");
    double agriProdDiscount, dailyCommDiscount;
    cout << "請輸入農產品折扣: ";
    cin >> agriProdDiscount;
    cout << "請輸入日用品折扣: ";
    cin >> dailyCommDiscount;
    CategoryDiscount catDiscount(agriProdDiscount,
                                 dailyCommDiscount);

    item1->printPrice(cout, catDiscount);
    item2->printPrice(cout, catDiscount);

    delete item1; delete item2;
    return 0;
}
/* ====範例輸出結果====
請輸入農產品折扣: 0.7
請輸入日用品折扣: 0.95
福樂北海道一番鮮鮮乳:102
UCC BLACK 無糖咖啡:45
*/
```

Sol:

```
// discount.h
#pragma once
class AgriculturalProduct;
```

```
class DailyCommodity;

class Discount
{
public:
    virtual double getDiscount(AgriculturalProduct &item) = 0;
    virtual double getDiscount(DailyCommodity &item) = 0;
};
```

b. [6] 請修改 Item 類別中關於折扣計算的 printPrice()函式，增加一個 Discount 類別的參數，修改計算價格的方法，由於 Discount 類別需要判斷 Item 類別的物件是農產品還是日用品，如果希望以後新增分類時不要持續修改舊程式，需要設計商品的繼承階層：AgriculturalProduct 和 DailyCommodity 類別來繼承 Item 類別，各自需要實作 printPrice()函式，其中 AgriculturalProduct 類別的產品免銷售稅的，DailyCommodity 類別的產品則需要 5%的銷售稅

c. [6] 在上述設計中 Item::printPrice()是虛擬函式，但是還是需要列印商品的名稱，請設計 Item::getPrice()成員以提供上題 AgriculturalProduct::printPrice()以及 DailyCommodity::printPrice()使用

Sol:

```
class Item
{
public:
    …
    virtual void printPrice(ostream& os, Discount& dObj) = 0;
protected:
    int getPrice();
    …
};
void Item::printPrice(ostream& os, Discount& dObj)
{
    os << m_name << ":";
}
int Item::getPrice()
{
    return m_price;
}
class AgriculturalProduct: public Item
{
public:
    AgriculturalProduct(char *name, char *id, int price,
                        Date doa, Date doe, char *aisle);
    void printPrice(ostream& os, Discount& dObj);
};

AgriculturalProduct::AgriculturalProduct(char *name, char *id,
                                         int price, Date doa,
                                         Date doe, char *aisle)
    : Item(name, id, price, doa, doe, aisle)
{
}

void AgriculturalProduct::printPrice(ostream& os, Discount& dObj)
{
    Item::printPrice(os, dObj);
    os << (int) (getPrice()*dObj.getDiscount(*this)+0.5) << endl;
}
class DailyCommodity: public Item
{
public:
```

```
        DailyCommodity(char *name, char *id, int price,
                       Date doa, Date doe, char *aisle);
        void printPrice(ostream& os, Discount& dObj);
};

DailyCommodity::DailyCommodity(char *name, char *id, int price,
                               Date doa, Date doe, char *aisle)
        : Item(name, id, price, doa, doe, aisle)
{
}

void DailyCommodity::printPrice(ostream& os, Discount& dObj)
{
        Item::printPrice(os, dObj);
        os << (int) (getPrice()*dObj.getDiscount(*this)*1.05+0.5) << endl;
}
```

d. [3] 請繼承 Discount 類別，設計 CategoryDiscount 類別，以農產品和日用品個別的折扣來建構 CategoryDiscount 類別的物件，實作 CategoryDiscount::getDiscount(AgriculturalProduct &) 以及 CategoryDiscount::getDiscount(DailyCommodity &)直接回傳初始化設定的折扣

Sol:

```
class CategoryDiscount: public Discount
{
public:
        CategoryDiscount(double agriProdDiscount, double dailyCommDiscount);
        double getDiscount(AgriculturalProduct &item);
        double getDiscount(DailyCommodity &item);
private:
        double m_agriProdDiscount;
        double m_dailyCommDiscount;
};

CategoryDiscount::CategoryDiscount(double agriProdDiscount, double dailyCommDiscount)
        : m_agriProdDiscount(agriProdDiscount), m_dailyCommDiscount(dailyCommDiscount)
{
}

double CategoryDiscount::getDiscount(AgriculturalProduct &item)
{
        return m_agriProdDiscount;
}

double CategoryDiscount::getDiscount(DailyCommodity &item)
{
        return m_dailyCommDiscount;
}
```

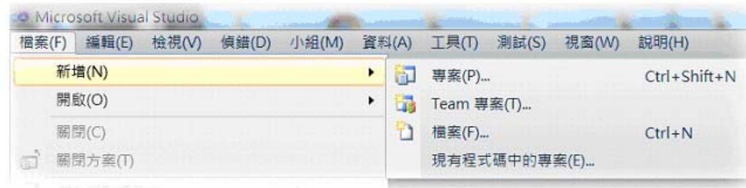
e. [3] 請問在 main()函式中，item1->printPrice(cout, catDiscount)根據*item1 以及 catDiscount 這兩個物件的實際型態來決定呼叫哪一個 printPrice()，這個分派機制在 C++ 中稱為?

Sol:

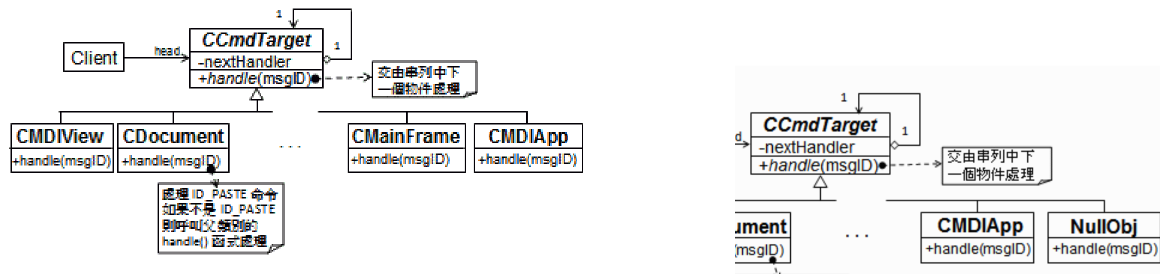
根據接受訊息的物件*item1 的型態以及訊息的參數物件 catDiscount 的型態來決定呼叫哪一個函式的機制稱為 “double dispatch”, 同時使用動態繫結和靜態繫結

6. (下面的題目說明比較多，但是請不要覺得沒有學過就跳過去，就算你會用 MFC/Qt/Java 寫 Win32 界面，你也會發現大多只是會使用，也沒弄清楚過它的設計原理，事實上下面的程式也和真正 framework 的實作不同，這個題目問的只是基本的物件設計概念而已，要求你回答的程式都不長) 在運用 MFC 撰寫具有多個文件視窗(MDI)的圖形化介面時，使用者可以藉由選擇視窗上方的選

單項目來對程式輸入命令，如下圖是 Visual Studio 的選單介面，這些選單命令必須由你設計的



軟體物件來處理，如下圖左在 Microsoft Foundation Class framework 中設計了可以處理這些命令的 MFC 類別，依序為 CMDIView, CDocument, CDocTemplate, CChildFrame, CMainFrame, CMDIApp 六個類別，這些類別包含螢幕上的視窗物件: CMDIView, CChildFrame, CMainFrame，應用程式裡的文件: CDocument, CDocTemplate，以及應用程式的執行緒: CMDIApp，使用者操作選單所產生的命令就會依序詢問這六個類別的物件，如果其中有一個你設計的物件能夠處理使用者的命令，就由它處理完畢，如果六個類別沒有一個能夠處理的話，這個選單就顯示成灰色的，使用者沒有辦法選它，在此題裡我們模擬設計這個命令傳送的機制，請根據下左圖來實作 CCmdTarget, CMDIView, CDocument, CMDIApp 四個類別中關於命令處理的機制，請注意



CCmdTarget 是一個抽象類別，其中 nextHandler 是指向下一個 CCmdTarget 衍生類別的物件的指標，串列的最後一個物件中 nextHandler 指標為 0，因此上圖描述了一個 CCmdTarget 異質串列

a. [5] 請依照圖中所示，在 CCmdTarget 類別中設計串列的產生和刪除功能

Sol:

```
// CmdTarget.h
class CCmdTarget
{
public:
      CCmdTarget(CCmdTarget *next);
      virtual ~CCmdTarget();
private:
      CCmdTarget *nextHandler;
};
// CmdTarget.cpp
#include "CmdTarget.h"
CCmdTarget::CCmdTarget(CCmdTarget *next)
      : nextHandler(next)
{
}
CCmdTarget::~CCmdTarget()
{
      delete nextHandler;
}

// Client.h   題目沒有要求寫
#include "CmdTarget.h"
class Client
{
public:
```

```
        Client();
        ~Client();
        …
private:
        CCmdTarget *head;
};
// Client.cpp
#include "Client.h"
#include "MDIView.h"
#include "Document.h"
#include "MDIApp.h"
Client::Client()
{
        head = new CMDIView(new CDocument(new CMDIApp(NULL)));
}
Client::~Client()
{
        delete head;
}
```

b. [5] 請依照圖中說明設計純粹虛擬函式 void CCmdTarget:: handle(int msgID)來順序處理命令

Sol:

```
// CmdTarget.h
class CCmdTarget
{
public:
        virtual void handle(int msgID) = 0;
        …
};
// CmdTarget.cpp
#include "CmdTarget.h"
void CCmdTarget::handle(int msgID)
{
        if (nextHandler)
                nextHandler->handle(msgID);
}
```

c. [5] 假設使用者的選單命令是「貼上」(此命令在程式中用一個前處理器常數 ID_PASTE 來代表)，假設這個命令需要由代表文件的 CDocument 類別物件來處理，請在 void CDocument::handle(int msgID) 函式中加入處理的程式碼，暫時運用 TRACE 列印簡單的訊息即可，如果不是 ID_PASTE 命令，則交由父類別的 handle()來處理

Sol:

```
// Document.h
#include "CmdTarget.h"
#define ID_PASTE 1001
class CDocument: public CCmdTarget
{
public:
        CDocument(CCmdTarget *next): CCmdTarget(next) {}
        void handle(int msgID);
};
// Document.cpp
#include "Document.h"
void CDocument::handle(int msgID)
{
        switch (msgID)
        {
        case ID_PASTE:
                TRACE("CDocument::handle() 處理貼上選單\n");
                break;
```

```
        default:
            CCmdTarget::handle(msgID);
        }
}
```

d. [5] CMDIView, CMDIApp 類別的 handle 暫時不處理任何選單命令，但是在處理到它不知如何處理的命令時，還是要交由串列中下一個物件來處理

Sol:

```
// MDIView.h
#include "CmdTarget.h"
class CMDIView: public CCmdTarget
{
public:
        CMDIView(CCmdTarget *next):CCmdTarget(next) {}
        void handle(int msgID);
};
// MDIView.cpp
#include "MDIView.h"
void CMDIView::handle(int msgID)
{
        switch (msgID)
        {
        default:
            CCmdTarget::handle(msgID);
        }
}
// MDIApp.h
#include "CmdTarget.h"
class CMDIApp: public CCmdTarget
{
public:
        CMDIApp(CCmdTarget *next):CCmdTarget(next) {}
        void handle(int msgID);
};
// MDIApp.cpp
#include "MDIApp.h"
void CMDIApp::handle(int msgID)
{
        switch (msgID)
        {
        default:
            CCmdTarget::handle(msgID);
        }
}
```

e. [5] 上圖左方的 Client 基本上是 framework 幫你寫好的類別，請問 framework 在使用者選擇「貼上」時，應該用怎樣的程式碼來傳送這個訊息給能夠處理的物件

Sol:

```
#define ID_PASTE 1001
class Client
{
public:
        Client();
        ~Client();
        void menuTest();
private:
        CCmdTarget *head;
};
void Client::menuTest()
{
```

```
        head->handle(ID_PASTE);
}
```

f. [5] 上圖右是另外一種異質串列的設計，主要在串列結束的地方運用一個 NullObj 類別來取代 nextHandler 為 0 的設計，請說明 CCmdTarget::handle(int)的修改以及 NullObj::handle(int)的內容

Sol:

需要修改與新增的地方如下：

```
// Client.cpp
#include "MDIView.h"
#include "Document.h"
#include "MDIApp.h"
#include "NullObj.h"
Cient::Client()
{
        head = new CMDIView(new CDocument(new CMDIApp(new NullObj)));
}
// CmdTarget.cpp
void CCmdTarget::handle(int msgID)
{
        nextHandler->handle(msgID);
}
// NullObj.h
#include "CmdTarget.h"
class NullObj: public CCmdTarget
{
public:
        NullObj(): CCmdTarget(NULL) { }
        void handle(int msgID) { TRACE("%d 未處理\n", msgID); }
};
```

在這個題目裡面其實大家看到了兩個簡單的 Design Pattern, 第一個是 Chain of Responsibility, 第二個是 Null Object, 要能夠很容易地上手來使用這些 Design Pattern, 第一你需要很清楚界面繼承的意義, 你需要很清楚動態多型的機制, 然後你需要一個很直覺、很基本的應用範例