國立台灣海洋大學資訊工程系 C++ 程式設計 期末考參考答案

姓名：_____ 系級：_____ 學號：_____

97/06/17

考試時間：**10:00 – 12:10**

試題敘述蠻多的，看清楚題目問什麼，針對重點回答是很重要的;不確定的請一定要當場提出來，不要白花力氣在誤會我的題意上，題目敘述很長的不一定很難，短的也不一定簡單；總分有 **178**，超過 **150** 的部份不計分，請看清楚每一題所佔的分數再回答

考試規則：1. 不可以翻閱課本，參考書、作業及程式
2. 不可以使用任何形式的電腦 (包含計算機)
3. 不可以左顧右盼、不可以交談、不可以交換任何資料、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)、隔壁的答案可能比你的還差，白卷通常比**錯得和隔壁同學一模一樣**要好很多很多
4. 提早繳卷同學請直接離開教室，不得逗留喧嘩
5. **違反上述任何一點之同學一律送請校方處理**
6. 請在答案卷上標示題號回答，繳卷時請繳交***簽名過之試題卷及答案卷***

1. a. 請解釋函式呼叫時的繫結 (binding) 和函式的連結 (linking) 有什麼差別? [5]
   b. C++語言中繫結有動態和靜態的兩種，請說明其語法和工作方式的異同? [10]
   c. 請問為什麼物件導向概念中希望使用動態繫結的方式? [5]
   d. 那麼 C++ 為什麼還要提供靜態繫結的語法呢? [5]

Sol:

a. 繫結 (binding) 是指針對程式裡某一函式呼叫敘述，如何決定呼叫哪一個函式的機制
連結 (linking) 是指將不同程式模組 (.cpp, .c, .asm, …) 裡撰寫的函式進入點與函式之呼叫藉由函式的名稱對應起來，另外程式模組裏面全域的變數也是藉由 linker 對應起來，如此在不同檔案裡不同的函式可以互相使用

b. 靜態繫結即為一般程式語言中的函式呼叫，compiler 根據 function signature 決定要呼叫哪一個函式 (全域或是成員函式), 這是在程式開始執行前就已經確定好了的事情
動態繫結為物件導向程式語言中一般支援的成員函式呼叫方法，C++ 語言藉由 function pointer, virtual function 及 virtual function table 來實作，當透過多型指標呼叫某一類別的成員函式時，compiler 將函式呼叫換成透過 virtual function table 中的 function pointer 來間接地呼叫該成員函式，因此會在執行時依據物件的種類來呼叫正確的成員函式 (該類別的成員函式)，兩者出了都使用函式呼叫的形式之外，運作機制完全不同。

c. 物件導向中物件接受其它物件的訊息來運作，並且針對所接受到的訊息產生回應的動作，成員函式所執行的事代表動作的內容，不論多型指標或是多型參考指向何種類型的物件，回應應該事由該類別的成員函式來負責，因此動態繫結會使得訊息送到正確的物件去處理。

d. 基本上 C++ 為了效率考量所以支援靜態繫結的機制，很多類別根本不需要繼承或是被繼承，使用相關物件的成員函式時就可以由 Compiler 在編譯時就確定是呼叫哪一個函式，可以減少函式呼叫的額外負擔 (overhead)，物件導向程式中又特別需要許多小成員函式來維持封裝的功能，因此 C++ 才特別提供靜態繫結的機制，基本上只要呼叫的不是 virtual function，或是並不是透過多型指標或是多型參考來呼叫成員函式，compiler 都會用靜態的細節來實作。

2. a. 請舉例說明 C++中多型 (Polymorphism) 有哪三種，分別以何種語法完成? [15]
   b. 請問多型對於程式設計者的好處為何? [5]
   c. 請問是哪一個機制可以達成 “先編譯好的程式呼叫後撰寫的程式 (old code call new code)”? [5]

Sol:

a. 靜態多型: function overloading, operator overloading 例如: int add(int); double add(double);
動態多型: virtual function call + inheritance 例如: class Base{public: void service();};
class Derived: public Base { public: void service();};
Derived obj;
Base *bptr = &obj;
bptr->service();
參數化多型: template function and template class

```
template <class T>
void swap(T x, T y)
{
    T tmp = x;
    x = y; y = tmp;
}
```

```
template <class T>
class Array
{
    …
 private:
    T data[100];
};
```

b. 可以用單一抽象的指令來完成各種概念接近的動作，設計的人不需要仔細去區分所有不同的狀況，可以使得設計起來比較簡化

c. 動態多型: 透過多型指標或是多型參考的函式呼叫，可以使得我們在撰寫一個類別的程式碼的時候先根據所需要使用的物件的公開“介面”來設計，不需要管實際使用的物件究竟如何實作該界面，尤其是實際使用的物件可能是未來繼承該介面的類別所產生出來的物件，此時就發現先編譯完成的程式碼是可以和還沒有寫出來的程式碼一起合作的，我們稱這種狀況為“old codes call new codes”，相對應於以往在程序化的程式設計裏面，你想要使用的函式庫一定要比你的應用程式先寫好，有這種機制以後程式的彈性會變得大的多。

3. a. 請舉例說明什麼叫做初始化串列 (initialization list)? [5]
b. 請問初始化串列中初始化多個資料成員時的執行順序為何? [5]
c. 請問沒有在初始化串列中出現的資料成員 compiler 會做什麼處理? [5]
d. 請列舉不用初始化串列就無法完成初始化的情況? [10]

Sol:

a.
```
class A
{
public:
    A(): m_y(0), m_x(1) {}
private:
    int m_x;
    int m_y;
};
```

b. 執行順序並非在初始化序列中撰寫的順序，以上例來說，並非先設定 m_y 為 0，再設定 m_x 為 1，而是依照資料成員 m_x, m_y 在類別宣告裏面定義的順序來執行，也就是先 m_x 再 m_y

c. 如果是父類別物件或是物件成員，compiler 會自動使用 default ctor 來初始化，但是如果是基礎型態 int, int*, double, char … 等等則 compiler 不做任何額外的動作

d. i. 使用有參數的父類別建構元建構父類別物件時
ii. 使用有參數的建構元建構物件成員時
iii. 初始化類別裡定義的常數資料成員時
iv. 初始化類別裡定義的參考型態資料成員時
v. 多重繼承時 virtual base class 裏面成員的初始化時
以上這五個地方是你一定要使用初始化串列的地方，也許有時 (運氣好) 可以有替代的方法，但是大概都伴隨著比較差的效率或是多做了許多額外的事，不如使用初始化串列來得簡潔。

4. 參考下圖中程式片段回答相關問題

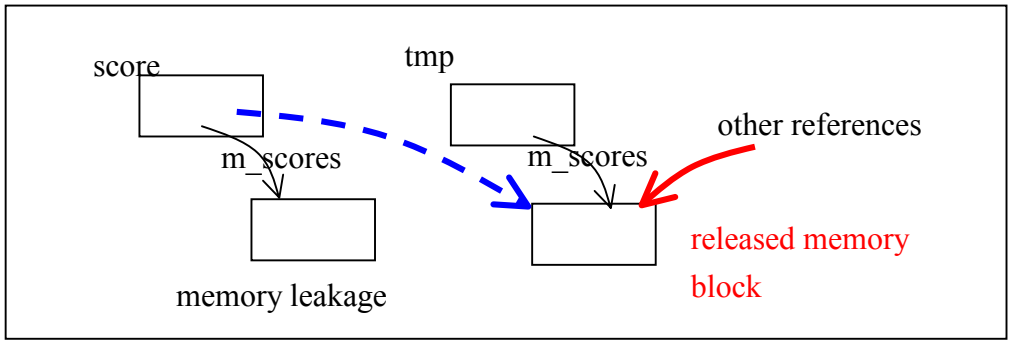
```
1.  class Scores
2.  {
3.  public:
4.        Scores();
5.        virtual ~Scores();
6.  private:
7.        int *m_scores;
8.  };
9.
10. Scores::Scores()
11. {
12.       m_scores = new int[5];
13.       for (int i=0; i<5; i++)
14.             m_scores[i] = 0;
15. }
```

```
16. Scores::~Scores()
17. {
18.       delete [] m_scores;
19. }
20. void init(Scores &score)
21. {
22.       Scores tmp;
23.       score = tmp;
24. }
25. int main()
26. {
27.       Scores myScores;
28.       …
29.       init(myScores);
30.       …
31. }
```

a. 程式執行時發現 main() 函式中的 myScores 這個物件裡的資料會不斷地自己更改，明明沒有修改它的資料，但是資料卻隨著程式執行到不同地方而不斷地改變，請解釋其發生之原因? [10]
b. 請說明該如何藉由增加一個 Scores 類別的成員函式來避免這個錯誤? [10]

Sol:

a. 主要原因是第 23 列 score = tmp; 時因為 Scores 類別沒有定義 Big 3 中的 assignment operator 而在第 24 列又自動解構 tmp.m_scores, 因此 score 物件裡指到的 m_scores 其實是已經歸還給系統的記憶體區塊，系統會把這塊記憶體給任何其他配置敘述拿去使用，於是別部份的程式在修改資料時，score 就會發現自己的資料莫名奇妙地在變動著，如下圖：



b. 實作 Scores::operator=(Scores &) 成員函式

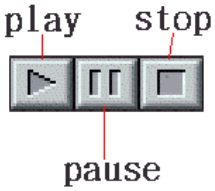
```
Scores &Scores::operator=(Scores &rhs)
{
    if (&rhs = = this) return *this;
    // delete [] m_scores;
    // m_scores = new int[5];
    for (int i = 0; i<5; i++)
```

```
            m_scores[i] = rhs.m_scores[i];
        return *this;
    }
```

5. 下圖是一個 mp3 播放程式的操作按鍵，其基本動作說明如下
   i. 系統在暫停或是停止狀態下，按下 play 按鍵可以繼續/開始播放
   ii. 系統在暫停狀況下，按下 pause 按鍵可以繼續播放
   iii. 系統在暫停或是播放狀況下，按下 stop 按鍵可以停止播放
   vi. 系統在播放狀況下，按下 pause 按鍵可以暫停播放

   假設已經寫好了 do_play(startingSecond), do_stop(), do_pause(pausedPosition) 這三個底層操作聲音界面和媒體檔案的函式供你使用，請注意你不需要寫顯示圖形或是當使用者用滑鼠按下按鈕時的動作，這些都假設已經寫好了

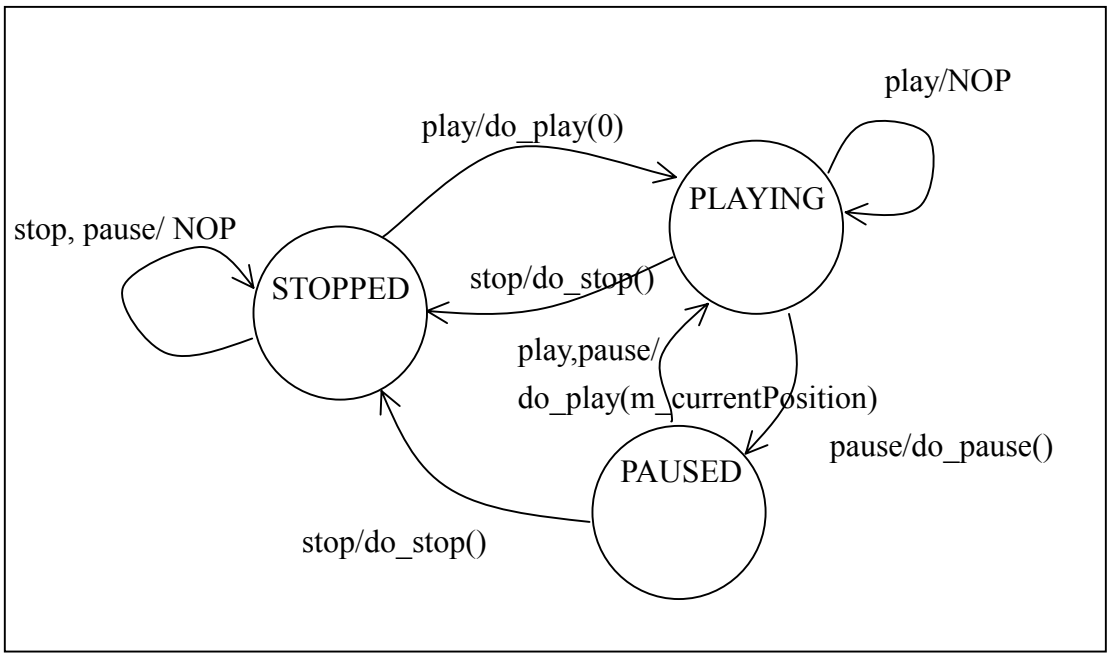
   a. 請分析系統所有可能接受到的訊息? [3]
   b. 請畫一個狀態圖說明上述運作狀態? [12]
   c. 請實作一個單一狀態變數，並且實作 mp3Player 類別的三個界面函式 [9]?



Sol:

a. 至少有 play, pause, and stop 三種使用者透過界面送近來的訊息, 可能還有播放完畢的訊息, 或是播放進度的訊息, 後兩者和底層多媒體播放 API 有關係

b. 我們以三種使用者操作界面的訊息為主繪製其狀態圖 (state diagram)



c.
```
class mp3Player
{
public:
    enum State {PLAYING, STOPPED, PAUSED};
    mp3Player();
    void play();
```

```
        void stop();
        void pause();
    private:
        State m_state;
        int m_currentPosition;
    };
    mp3Player::mp3Player():m_state(STOPPED), m_currentPosition(0) {}
    void mp3Player::play()
    {
        if (m_state == STOPPED)
        {
            m_currentPosition = 0;
            do_play(0);
            m_state = PLAYING;
        }
        else if (m_state == PAUSED)
        {
            do_play(m_currentPosition);
            m_state = PLAYING;
        }
        else if (m_state == PLAYING);
    }
    void mp3Player::stop()
    {
        if ((m_state == PLAYING)||(m_state == PAUSED))
        {
            m_currentPosition = 0;
            do_stop();
            m_state == STOPPED;
        }
        else if (m_state == STOPPED);
    }
    void mp3Player::pause()
    {
        if (m_state == PLAYING)
        {
            m_currentPosition = do_pause();
            m_state = PAUSED;
        }
        else if (m_state == PAUSED)
        {
            do_play(m_currentPosition);
            m_state = PLAYING;
        }
        else if (m_state == STOPPED);
    }
```

以上程式其實有一些假設，例如 do_pause() 會回傳目前播放的位置...

6. a. 請問 C++ 用什麼語法實現 Generic Programming? [5]

   b. 請將右圖中的函式轉為一個樣版函式? [5]

   c. 請定義一個 C++ 類別 MyType 使得下面的程式碼能夠配合題 b. 中的樣版函式 sum 運作? [10]

```
MyType<int> s1, s2;
MyType<int> s3 = sum(s1, s2, s2);
```

```
1.  int sum(int a, int b, int c)
2.  {
3.      return a + b + c;
4.  }
```

Sol:

a. template function 及 template class, 亦即 parameterized polymorphism

b.
```
template <class T>
T sum(T a, T b, T c)
{
```

```
        return a + b + c;
    }
```

c.
```
template <class T>
class MyType
{
public:
    MyType();
    MyType operator+(MyType &);
private:
    T m_data[10];
};
template <class T>
MyType<T>::MyType()
{
    for (int i=0; i<10; i++)
        m_data[i] = T(0);
}
template <class T>
MyType<T> MyType<T>::operator+(MyType<T> &rhs)
{
    MyType<T> tmp = *this;
    for (int i=0; i<10; i++)
        tmp.m_data[i] += rhs.m_data[i];
    return tmp;
}
```

這個類別必須設計 operator+() 否則題 b. 中的 a+b+c 無法運作，另外由於這個類別沒有自行配置記憶體，所以可以用預設的拷貝建構元處理 tmp = *this; 另外由於平常 operator+() 都是沒有 side effect 的，不可以在加的時候改掉物件的內容，所以另外配置一個暫時的變數 tmp

7. 參考下列的程式片段回答相關問題

```
1. #include <iostream>
2. using namespace std;
3. class Fruit
4. {
5. public:
6.         virtual void printClassName() = 0;
7.         virtual ~Fruit() {}
8. };
9. class Apple : public Fruit
10. {
11. public:
12.         void printClassName()
13.         {
14.                 cout << "Apple" << endl;
15.         }
16. };
17. class Pear : public Fruit
18. {
19. public:
20.         void printClassName()
21.         {
22.                 cout << "Pear" << endl;
23.         }
24. };
25.
26. class Full {};
27. class Empty {};
28.
29. class BagOfFruit
30. {
31. public:
32.         BagOfFruit() : m_numElems(0) {}
33.         unsigned int numElems() const
34.         {
35.                 return m_numElems;
36.         }
37.         virtual void insert(Fruit &f) throw(Full);
38.         virtual Fruit &remove() throw(Empty);
39. private:
40.         enum { NumMaxElems = 20 };
41.         unsigned int m_numElems;
42.         Fruit *m_data[NumMaxElems];
43. };
```

a. [10] 請完成類別 BagOfFruit 的製作：BagOfFruit 類別是一個藉由父類別指標 (此處指的是 Fruit 類別物件的指標) 來操作的“異質容器類別”，在一袋水果中最多可以存放 20 (NumMaxElems) 個各式的水果 (由 Fruit 類別衍生出來的類別)，類別資料成員 m_numElems 記錄袋中共有多少個水果物件，注意這個數字必須保證在 0 到 NumMaxElems-1 之間，m_data 這個 Fruit * 型態

的指標陣列是用來記住袋中有哪些物件的，請完成 insert() 及 remove() 函式的製作，其中 insert 函式可以將一個 Fruit 類別的物件記錄在 m_data 陣列中(應該說是將一個水果置入袋中)，如果不幸 insert 動作失敗的時候，請依照類別宣告中的 exception specifier 丟出適當的例外事件，remove 則是把放在最上面的水果 (最後放進去的水果) 取出來，在 BagOfFruit 物件中就不記錄這個被取出的物件了，同樣地在發生 remove 錯誤時也請產生例外物件。

Sol:

```
void BagOfFruit::insert(Fruit &f) throw(Full)
{
    if (m_numElems >= NumMaxElems) throw Full();
    m_data[++m_numElems] = &f;
}
Fruit &BagOfFruit::remove() throw(Empty)
{
    if (m_numElems <= 0) throw Empty();
    return *m_data[m_numElems--];
}
```

b. [10] 請撰寫一小段程式，產生 11 個 Apple 類別的物件及 11 個 Pear 類別的物件，然後交替地放入一個 BagOfFruit 類別的物件中，記得處理例外狀況。

Sol:

```
BagOfFruit bof;
Apple *aptr = 0;
Pear *pptr = 0;
try
{
    for (int i=0; i<11; i++)
    {
        aptr = new Apple();
        bof.insert(*aptr);
        pptr = new Pear();
        bof.insert(*pptr);
    }
}
catch (Full &e)
{
    cout << "Bag full\n";
}
catch (Empty &e)
{
    cout << "Bag empty\n";
}
```

c. [4] 假設在系統中需要一種 BagOfApple 的物件，這種物件只能放入 Apple 類別的物件，這種物件提供的界面包括 unsigned int numElems() const;
void insert(Apple&a) throw(Full); 及
Apple &remove() throw(Empty);
因為這個界面和 BagOfFruit 實在很接近，因此某甲就運用繼承的語法製作了下面的類別

```
1.  class BagOfApple : public BagOfFruit
2.  {
3.  public:
4.          BagOfApple() : BagOfFruit() {}
5.          void insert(Apple &a) throw(Full)
6.          {
7.                  BagOfFruit::insert(a);
8.          }
9.          Apple &remove() throw(Empty)
10.         {
11.                 return (Apple &)BagOfFruit::remove();
12.         }
13. };
```

請問下面的程式碼中第 4 列在執行時會呼叫哪一個函式? 第 5 列在執行時會呼叫哪一個函式?

```
1. BagOfApple bOA;
2. BagOfFruit *pBOF = &bOA;
3. Apple a;
4. pBOF->insert(a);
5. pBOF->remove();
```

Sol:

(1) 請注意第 5 列到第 8 列 void &BagOfApple::insert(Apple &) 並沒有覆寫(override) void &BagOfFruit::insert(Fruit &)，因為參數的型態是不一樣的，基本上只是 hide 父類別的同名函式而已

(2) 請注意第 9 列到第 12 列運用到所謂的 Contravariance 的概念，很多編譯器 (VC 6.0) 並不支援，VC 6 在編譯的時候會告訴你 Apple &BagOfApple::remove() 嘗試覆寫 Fruit &BagOfFruit::remove() 函式，而且兩個函式只有回傳值是不一樣的，這是不允許的；相反地，在 Linux 上的 GNU g++ 就允許這個語法，讓 Apple &BagOfApple::remove() 覆寫 Fruit &BagOfFruit::remove() 函式， 以 g++ 為例：

第 4 列呼叫 BagOfFruit::insert(Fruit &)

第 5 列呼叫 BagOfApple::remove()

d. [4] 假設在 BagOfFruit 的定義中 (第 37, 38 列), insert() 及 remove() 函式並不是虛擬函式 (virtual function) 請問上圖中第 4 列及第 5 列會呼叫哪一個函式?

Sol:

第 4 列呼叫 BagOfFruit::insert(Fruit &)

第 5 列呼叫 BagOfFruit::remove()

e. [6] 請閱讀下圖程式，說明第 11 列中 compiler 會自動將物件 p 做什麼樣的型態轉換? 請問第 12 列會印出什麼信息? Pear 類別的物件到底可不可以置入 BagOfApple 類別的容器物件中呢?

```
1. void insertFruit(BagOfFruit &bag, Fruit &f)
2. {
3.         bag.insert(f);
4. }
5. ...
6. void main()
7. {
8. ...
9.         BagOfApple bOA;
10.        Pear p;
11.        insertFruit(bOA, p);
12.        bOA.remove().printClassName();
13.}
```

Sol:

Pear 型態的物件先轉換成 Fruit 物件的參考, 會印出 "Pear"

目前程式可以將 Pear 物件置入 BagOfApple 容器物件中

而且在第 3 列是呼叫 BagOfFruit::insert(Fruit &) 而不是 BagOfApple::insert(Apple &)

f. [5] 請問像 insertFruit() 這樣子一個基於基礎類別 BagOfFruit 和 Fruit 的定義所寫出來的函式是否可以處理衍生類別 BagOfApple 的物件呢? (為什麼?)

Sol:

實際運作中你會發現在 insertFruit() 函式中，並無法正確地將 Apple 物件 insert 到 BagOfApple 中，除非運用 downcast 將第三列改成

```
bag.insert((Apple) f);
```

但是這樣又沒有辦法處理一般的 BagOfFruit 物件，所以基本上這個設計是不理想的，應該用組合來重新設計 BagOfApple 類別