國立台灣海洋大學資訊工程系 C++ 程式設計 期中考試題

姓名：_____ 系級：_____ 學號：_____

96/04/24

考試時間：**10:00 – 12:00**

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，總分有 **155**，請看清楚每一題所佔的分數再回答

考試規則：1. 不可以翻閱參考書、作業及程式
2. 不得使用任何形式的電腦 (包含計算機)
3. 不得左顧右盼、不得交談、不得交換任何資料、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，不得逗留喧嘩
5. **違反上述任何一點之同學以作弊論，一律送請校方處理**
6. 繳卷時請繳交***簽名過之試題卷及答案卷***

1. a. [5] 物件導向程式中整個程式的功能是由許多物件合作來完成的，每一個物件對其它物件提供一些服務，請問這些服務是透過類別的什麼機制來提供的?

   b. [5] 接題 a, 傳統的變數只提供資料的儲存，不提供加值的服務，所以在運用類別 (class) 的語法定義時我們通常把資料成員定義為 private，請問定義為 private 和定義成 public 的資料成員在使用上有什麼差別?

   c. [5] 請問 C++ 語言中運用 struct 和 class 語法定義出來的自定型態有什麼差別?

   d. [5] 假設你和同事合作撰寫程式，你要求你的同事在呼叫你寫的 memory_alloc(int size) 函式時，一定要傳入一個大於 0 的數值，你在寫 memory_alloc(int size) 函式時也假設 size 參數是一個大於 0 的數值，如果等於或是小於 0 就可能造成不可預期的錯誤，請撰寫 assert 敘述來保證這件事? (請問為什麼不用 if 敘述來檢查?)

   e. [5] 請問為什麼物件需要封裝 (encapsulation)? (好處在哪裡?)

   Sol:

   a. 成員函式

   b. private 的資料成員只有在這個類別的成員函式中可以存取，其它類別的成員函式或是全域的函式中不能夠存取這些 private 的資料，這些資料成員基本上構成一個物件的狀態，物件狀態只有它自己可以更改；public 的資料成員則允許任意程式直接讀取或是修改它的內容。

   c. struct 定義出來的自定型態中所有的資料成員和成員函式預設都是 public 的；class 中資料成員和成員函式預設都是 private 的，通常在 C++ 中我們用 struct 定義不需要特別封裝的公用資料結構，用 class 來定義適當封裝的物件。

   d. assert(size>0);

   基本上如果要使用 if 敘述的話, 應該要運用如下的敘述,

```
if (size <= 0)
{
    進行當 size <= 0 時的相關處理
}
```

   但是如果你和你的同事之間基本上沒有談到當 size <= 0 時該如何處理, 你也許不應該自作主張去處理這種錯誤的狀況，而應該在發生這種狀況的時候請你的同事處理呼叫端的問題，所以在程式中我們通常把這種隱藏的假設都以 assert 敘述表達出來，以免日子一久就忘記了原先的假設，如果以後要修改程式的時候也不會因為不小心破壞了這個假設而導致程式的錯誤。

e. “物件封裝” ①對於使用這個物件的程式而言，可以透過簡潔的介面來操作這個物件，不會被它內部複雜的組成細節影響，只要專注於它所提供的功能即可；這個物件內部的問題也不至於擴大影響到使用它的客戶端程式，封裝也代表必須固定物件的介面，方便日後以元件型式取代；②對於這個物件本身的設計來說，封裝提供一個隔離發展的環境，可以確保這個物件內部的狀態不受其它客戶端程式影響。

```
1.  struct Point
2.  {
3.      int x, y;
4.  };
5.
6.  class DisplayedText
7.  {
8.  public:
9.      DisplayedText(char *txt, Point p);
10.     draw();
11. private:
12.     char *m_txt;
13.     Point m_pos;
14. };
```

```
1.  void foo(DisplayedText text)
2.  {
3.      …
4.  }
5.
6.  void main()
7.  {
8.      Point position = {10, 20};
9.      DisplayedText dtObj("Caption", position);
10.     foo(dtObj);
11.     DisplayedText dtObj2 = dtObj;
12.     …
13. }
```

2. a. [5] 請替上圖左中 DisplayedText 類別實作第 9 列的建構元函式 (運用 new 替 m_txt 配置一個可以容納 txt 指標所指向字串的字元陣列, 由 txt 拷貝字串, 並且運用初始化串列將傳入的 p 設定 m_pos 資料成員)

   b. [5] 請問上圖右中第 8 及第 9 列如果要用 DisplayedText dtObj("Caption", Point(10, 20)) 取代的話, 圖左中程式該如何修改?

   c. [5] 如題 a 在建構元中配置記憶體後，你應該要替這個類別加入解構元函式，否則圖右 main 函式執行後會有記憶體遺漏 (memory leakage) 的問題，請說明如何修改 DisplayedText 類別的定義，並且實作解構元函式?

   d. [5] 我們在課堂中說明過如果一個類別有實作解構元函式的話，通常它也需要拷貝建構元以及設定運算子，請說明如何修改 DisplayedText 類別的定義，以實作拷貝建構元? (請說明沒有撰寫拷貝建構元的話會發生什麼樣的錯誤?)

   e. [5] 請問在上圖右程式中哪幾列 compiler 會自動呼叫拷貝建構元?

   f. [10] 接上題，請說明如何修改 DisplayedText 類別的定義，以實作設定運算子? (請注意需要加入必要的檢查)

   g. [5] 請問 compiler 為什麼不會接受在 main 函式中的 DisplayedText dtArray[100]; 敘述? (如何修改 DisplayedText 類別使得 compiler 可以接受呢?)

   h. [5] 請在 main 函式中定義一個元素為 DisplayedText 物件的 vector 物件，名稱為 dtArray, 並且寫一小段程式在這個 vector 物件中加入三個 DisplayedText 物件 (你可以自己決定這些物件的內容)?

   i. [5] 請問在上題的答案中你預期 compiler 會如何使用 DisplayedText 的哪些建構元?

   j. [5] 請問這個 dtArray 物件在什麼地方解構? 解構時 compiler 會幫我們加入哪些函式的呼叫?

   k. [5] 請運用 vector 的 iterator 寫一個迴圈呼叫 dtArray 中每一個元素的 draw() 服務?

Sol:

a.
```
DisplayedText::DisplayedText(char *txt, Point p):m_pos(p) {
    m_txt = new char[strlen(txt)+1];
```

```
        strcpy(m_txt, txt);
    }
```

b. Point 結構必須增加兩個參數的建構元如下

```
    struct Point {
        Point(int x1, int y1):x(x1), y(y1) {}
        int x, y;
    };
```

c.
```
    class DisplayedText {
     public:
        ~DisplayedText();
        …
     };
     DisplayedText::~DisplayedText() {
        delete[] m_txt;
     }
```

d.
```
    class DisplayedText {
     public:
        DisplayedText(const DisplayedText &src);
        …
     };
     DisplayedText::DisplayedText(const DisplayedText &src):m_pos(src.m_pos) {
        m_txt = new char[strlen(src.m_txt)+1];
        strcpy(m_txt, src.m_txt);
    }
```

這個類別如果沒有撰寫拷貝建構元的話，C++ compiler 會提供一個 bitwise 的拷貝建構元，此種拷貝我們一般稱為 shallow copy，會產生兩個物件都指向相同的 m_txt 記憶體區段去，如此在釋放其中一個物件以後，另外一個物件也失去了它的 m_txt 記憶體區段，如果第二個物件持續去存取它的 m_txt 區段的話，會產生 illegal access；如果第二個物件沒有去存取的動作，在釋放第二個物件的時候還是會發生重複釋放的問題，使得記憶體管理模組產生內部錯誤。

e. 第 10 列呼叫 foo 函式時會拷貝 dtObj 到函式的 text 參數，此時就運用拷貝建構元來建構 text 物件；第 11 列運用 dtObj 來建構 dtObj2 時也會以拷貝建構元來建構 dtObj2

f.
```
    class DisplayedText {
     public:
        DisplayedText& operator=(DisplayedText &rhs);
        …
     };
     DisplayedText& DisplayedText::operator=(DisplayedText &rhs) {
        if (this == &rhs) return *this;
        delete[] m_txt;
        m_txt = new char[strlen(rhs.m_txt)+1];
        strcpy(m_txt, rhs.m_txt);
```

```
        m_pos = rhs.m_pos;   // 請注意此處運用 Point 類別預設的 assignment operator
        return *this;
    }
```

g. compiler 會產生找不到 default constructor DisplayedText() 的編譯錯誤

error C2512: 'DisplayedText' : no appropriate default constructor available

h.
```
#include <vector>
using namespace std;
    …
void main() {
        …
    vector<DisplayedText> dtArray;
    dtArray.push_back(DisplayedText("1st text string", Point(10,20)));
    dtArray.push_back(DisplayedText("2nd text string", Point(10,30)));
    dtArray.push_back(DisplayedText("3rd text string", Point(10,40)));
        …
}
```

i. 以 dtArray.push_back(DisplayedText("1st text string", Point(10,20))); 為例

首先 compiler 會加入呼叫 DisplayedText::DisplayedText(char *, Point) 建構元的程式碼來建構一個暫時性的物件，然後由於 vector<>::push_back() 這個函式運用 call-by-value 的方式傳入一個參數，所以會呼叫 DisplayedText::DisplayedText(const DisplayedText &src) 拷貝建構元來建構函式裡的參數，並且記錄在 vector 容器中；如果你真的去執行這個小程式的話，你可能會發現拷貝建構元不只呼叫了三次，好像每次 push_back 一個新的元素進到 vector 的時候, vector 物件都會來一次大搬家，針對每一個容器中的物件呼叫拷貝建構元，此時可以推測是因為你 push_back 一個新的元素進去時，vector 物件發現它所配置的記憶體不夠使用，重新配置更大的記憶體，然後把原來已經記錄在容器裡的元素拷貝一遍，請注意，因為如此，所以如果你很在意效率的話，也許應該使用記錄指標的容器物件 vector<DisplayedText *>

j. dtArray 物件在 main() 函式結束時會自動解構，compiler 會幫我們加入解構元函式的呼叫，針對每一個 dtArray 中的 DisplayedText 物件自動去呼叫 DisplayedText::~DisplayedText() 函式

k.
```
vector<DisplayedText>::iterator iter;
for (iter=dtArray.begin(); iter<dtArray.end(); iter++)
    iter->draw(); // 或是 (*iter).draw();
```

3. [25] 請解釋下表中各個 const 敘述的意義與程式設計者為什麼要如此寫的目的?

void service(**const** char *inString);
const int kMaxSize = 100;
const Complex &Complex::operator+(…some other things…);
double Complex::getReal() **const**;
int array[100]; int ***const** iPtr = array;

Sol:

a. void service(const char *inString)

此 const 使 compiler 知道 inString 指標所指到的字元陣列是常數，可以防止程式設計者在

service 函式內修改 inString 字元陣列的內容，例如：
*inString = 'c'; 或是 inString[10] = 'd'; char *ptr=inString; 都是不允許的
程式設計者運用這種 const 參數來告訴呼叫這個 service() 函式的客戶端程式：不論傳入哪一個字元陣列的指標，函式內部絕對不會去修改這個字元陣列的內容

b. const int kMaxSize = 100;
kMaxSize 的值固定為 100, 任何時候都不會改變，程式設計者通常利用這樣的敘述使得在程式中不會出現 100 這樣的常數，需要用到 100 時可以用一個有名稱的 kMaxSize 來撰寫程式，增加程式的可讀性，如果需要修改 100 為 200 時，也可以在單一的地方修改，不致於發生不一致的狀況

c. **const** Complex &Complex::operator+(…some other things…);
程式設計者希望不要藉由這個函式回傳的 Complex 物件呼叫非 const 的成員函式

d. double Complex::getReal() **const**;
告訴 compiler 這個 getReal() 成員函式不會修改 Complex 物件的狀態，一個 const Complex 物件只可以呼叫這種 const 的成員函式而不能呼叫一般的成員函式

e. int array[100]; int ***const** iPtr = array;
基本上 iPtr 和 array 一樣是內容不可以更改的指標變數，程式設計者其實只是想要設計一個 array 陣列變數的別名而已，可以運用 array[i] 或是 iPtr[i]，但是不可以進行 iPtr++ 這種修改 iPtr 內容的動作

4. a. [5] 請運用 typedef 敘述替 "二十個浮點數的陣列" 定義一個新的型態名稱 D20ARRAY (如此你可以用 D20ARRAY array; 的敘述取代 double array[20]; 的敘述)

b. [5] 請繪圖說明 double *matrix[10]; 的記憶體使用情形? (共使用多少個位元組，連續的嗎?)

c. [5] 如果我希望運用上述 matrix 定義來製作 10x20 的二維陣列，希望可以用 matrix[5][7] 來存取其中第 6 列第 8 行的元素，請寫一小段程式運用 new 來配置所需要的記憶體?

d. [5] 如果我需要把這個 matrix 資料傳進一個 process 函式中，請問函式為什麼可以宣告為 void process(double **matrix)? (matrix 變數的宣告似乎和 b 中的定義不一致)

e. [5] 接上題，請問上述宣告和 void process(double *matrix[10]) 的宣告使用上的差別在哪裡?

f. [5] 請問上述 matrix 的記憶體配置和另外一種二維陣列的定義方法: double matrix2d[10][20] 有什麼不同? (使用記憶體的大小? 連續性?)

g. [5] 如果我需要把這個 matrix2d 資料傳進一個 process2 函式中，請問函式為什麼可以宣告為 void process2(double (*matrix2d)[20])? (還有其它宣告方法嗎?)

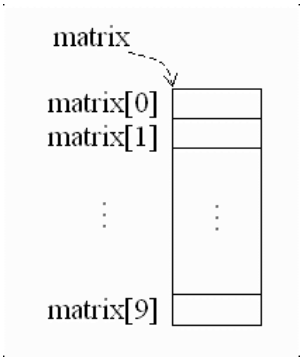
h. [5] 請運用 a. 題的定義簡化 f. 與 g. 兩題的宣告

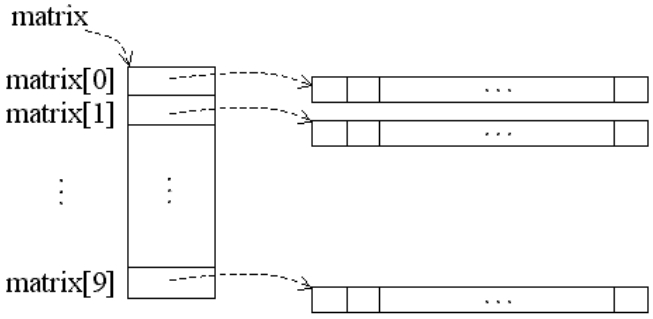
i. [5] 請問 matrix++ 和 matrix2d++ 這兩個敘述對於個別指標變數 matrix 和 matrix2d 的內容所作的動作有什麼差別?

Sol:

a. typedef double D20ARRAY[20];

b. 這是一個 10 個元素的陣列，每一個元素都是一個浮點數變數的記憶體位址，如下圖左所示共使用連續的 10 * sizeof(double*) = 40 bytes 記憶體





c. int i;

for (i=0; i<10; i++)

matrix[i] = new double[20];

上面這一段程式配置完以後產生上圖右的記憶體配置，雖然是 10 個分開的區段，每一個區段有連續的 20 個倍精準的浮點數，但是使用起來就好像它們是完全連續的，例如 matrix[5][7] 代表其中第 6 列，第 8 行的元素，不過嚴格地看這個使用方法，matrix[5] 代表的當然是上圖右陣列的第 6 個元素，由於它是一個 double * 型態的指標，所以內容所指的基本上是另一個一維的陣列，所以可以用 matrix[5][7] 來存取第二層的陣列裡的第 8 個元素，matrix[5][7] 也可以看成是 (matrix[5])[7]，也許比較容易了解

d.當我們把任何一個陣列傳入函式時都有兩種不同的宣告方法，例如

int x[20];

…

foo(x);

函式 foo 可以宣告為 void foo(int z[20]) {…} 或是

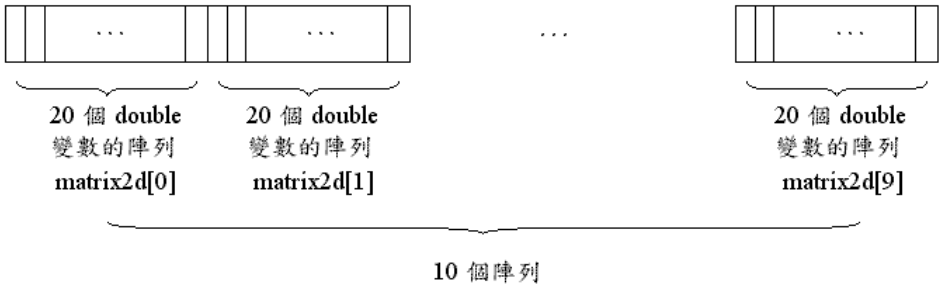
void foo(int *z) {…}

第一種宣告中 z 是一個位址常數，和呼叫端的 x 是相同的位址常數，它們的型態都是 int *，也就是說 *z 或是 *x 的型態都是整數

第二種宣告中 z 是一個位址變數(也就是指標變數)，函式開始時的初始值就是 x 這個位址常數，由於 z 是一個變數，所以裡面的資料可以更改，你可以做 z = z + 10; 或是 z++; 的動作，但是在第一種宣告中並不能這樣子使用

在題目中 matrix 的定義是 double *matrix[10];，當要把 matrix 這個陣列傳入函式中也就有兩種宣告方法第一種是 void process(double *matrix[10]) 第二種則是 void process(double **matrix)

e. 在 void process(double *x[10]) 的宣告中 x 和原來呼叫端的 matrix 都一樣是位址常數，*x 或是 *matrix 的型態都是倍精準浮點數的位址 double *；在 void process(double **x) 的宣告中 x 是指標變數，所記錄的記憶體位址的那個變數也是一個指標變數，存放的是倍精準浮點數的位址，同樣地第二種宣告方法彈性稍為大一點



f. double matrix2d[10][20]; 定義了如上圖連續的 10x20 個 double 型態的變數，總共佔用 10x20x8 = 1600 個位元組，使用前不需要像題 c 一樣另外再動態地配置記憶體，使用起來也是一樣用兩個方括號來存取，例如 matrix2d[5][7] 代表其中第 6 列，第 8 行的元素，仔細分析這個表示式，matrix2d[5] 代表的是連續的 10 個陣列裡的第 6 個陣列，matrix2d[5][7] 也就是 (matrix2d[5])[7] 則代表那個陣列裡的第 8 個元素

g. 要把 matrix2d 這個陣列傳入函式裡的話也是有兩種方式，第一種是 void process2(double z[10][20])，這是比較沒有彈性的方法，第二種則是 void process2(double (*z)[20])，也就是說 z 其實是一個指到 20 個倍精準浮點數元素陣列的指標

h. double matrix2d[10][20]; 可以改成 D20ARRAY matrix2d[10];
void process2(double (*z)[20]) 可以改成 void process2(D20ARRAY *z)
是不是比較容易了解呢?

i. matrix++ 把 matrix 指標變數的內容加上 sizeof(double*)=4 個位元組
matrix2d++ 把 matrix2d 指標變數的內容加上 sizeof(double[20])=20*4 個位元組