國立台灣海洋大學資訊工程系 C++ 程式設計 期中考參考答案

姓名：_____

系級：_____

學號：_____

101/04/17

考試時間：**09:30 – 11:30**

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 146，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. [10] 請舉例說明為什麼大家常說 Abstract Data Type = data + operation? (沒有 operation 就不是資料型態了嗎?) 有 operation 的 ADT 和沒有 operation 的 data type 對程式設計者的差別究竟在哪裡?

Sol:

抽象的資料型態 (Abstract Data Type) 描述很多同一個類型/功能的物件，抽象化得到的共通性質 (包括內部儲存的資料、狀態、以及外在表現出來的功能)。要能夠清楚、明確地描述這些資料欄位的型態以及它們如何參與程式，完成整體的功能，必須要以操作型的定義來描述它，也就是描述它的所有操作 (operation)，以 C++ 來說就是類別裡公開的成員函式，一般稱為類別的“介面”；一個沒有描述 operation 的資料型態在程式裡可以參與的運算太過廣泛，沒有任何限制，好像很有彈性，但是卻很容易出現意外的錯誤，比如在一個記錄人名的字元陣列裡存放電話號碼，在一個記錄考試成績的整數資料裡記錄一個負數一樣；從另一個角度看就好像一張白紙和一張電腦閱卷的答案卡之間的差別，一張白紙上你可以寫任何的東西，和考試題目有關係或沒有關係都可以，一張答案卡上你能夠寫的東西就很有限制了，一定會跟指定的問題有關係，甚至只有固定的幾個選項，當我不希望你在考試的時候拿到一張空白的答案卷有所誤會時，就希望能夠用有格式的答案卡來限制一下，這樣子可以避免很多的錯誤；設計程式的時候最容易發生的錯誤狀況就是資料變數裡面的資料和你的預期不一樣，不是嗎? 在一個程式裡你可以決定什麼時候用沒有 operation 的資料型態，什麼時候封裝成有固定介面的物件，並不是所有的東西都一定要是物件，最底層的資料儲存還是基本型態的變數，有些資料的組合，修改的方式很多種，不是很容易固定成一種介面，或是功能還沒有多到你以後用置換的方式來檢測程式，那就還是用 struct 設計就可以了。

2. [16] 請問封裝(encapsulation)一個物件時，物件的介面(interface)是指什麼? C++中用什麼關鍵字來實現存取權限的控制? 為什麼一般而言我們常常把資料成員設為私有的? 一個私有的資料成員在哪些程式段落裡可以存取?

Sol:

物件/類別的介面指的是類別中公開的成員函式或是資料成員、以及這些公開的東西的運用方法 (例如你有三個公開的函式 initialize(), service(), cleanup(), 你的介面會包含一些隱藏的順序，例如呼叫 service() 之前需要呼叫 initialize(), 物件銷毀前需要呼叫 cleanup()等等)。C++ 用

public/private/protected 來實現存取權限的控制，public 的部份就是允許其他物件來使用的介面。一般而言，資料成員都盡量設計為私有的(private)，主要原因在上一題有大略提到，每一個物件都不希望自己控管的資料能夠被其他物件任意使用和任意修改，很容易造成物件狀態的錯誤，例如一個堆疊物件，裡面有一個欄位是 top，代表目前存放資料的筆數，如果目前有 5 筆資料在堆疊上，可是因為 top 是公開的，不小心被程式其他部份改掉了，例如 stack.top=0；，此時就造成程式中有 5 筆資料不見了，要尋找這個錯誤你可能需要檢查整個程式中所有使用 stack 的程式碼，而不只去看實作 stack 的那些程式碼。一個類別裡私有的資料成員只有在這個類別的成員函式裡可以存取，任何其他地方都不能直接使用。

3. [10] 請問在一個物件化的程式裡，我們要求把物件很嚴格地封裝起來的好處是什麼? (請從設計面、程式除錯、程式修改維護各角度來回答)

Sol: (題目沒有表達清楚，原本希望同學回答封裝對於設計“整個程式”有什麼好處，而不是對於設計“單一的物件”有什麼好處，雖然前半學期我們都在設計單一的物件，但是你需要了解為什麼要做這樣的事情，設計整個程式的時候有太多的決策要做，我們不可能一一提到，因為了解物件化對整個設計的好處，很多時候你設計時的決策才有根據，才不會用“課本說的”、“講義寫的”、“老師講的”、“網路查的” 來搪塞)

程式設計：

在設計上我們希望程式的演算法中透過一些有固定規格的物件來組合，而不要都用基本的變數/迴圈/條件判斷/函式來實作，雖然後者由 CPU/RAM 的角度去看可能是很經濟很有效率的，但是因為沒有中間層的物件規格，將來沒有辦法運用這些中間層的物件來組成新的功能，沒有辦法運用抽換物件的方式來調整效能、尋找問題、分析錯誤。

想像一下如果你用的電腦，如果從運算、記憶、儲存、電池、輸入、網路、顯示、到音效多媒體全都用一個神奇的晶片就完成了，有效率、省資源、省電、體積小，很酷吧! (如果能夠想像的話)，不過當任何功能出現問題的時候，廠商只能跟你說...換吧! 整個換掉吧，送修比買還貴喔! 有 coco 的人當然這時候就很帥了，可是應該很多人就會懷念電腦可以組裝的日子了，換一換有可能出問題的東西搞不好損失一頓好一點的晚餐就打發掉了!! 有沒有想到這個例子和物件化程式之間的關係?! 你先前寫的程序化程式就好像是這個神奇的電腦，很多時候有問題的話你都會想全部重寫!!! 放心，這不是你的不好，很多公司專業的程式設計團隊如果用程序化的設計模式的話常常也是這樣。物件化的程式設計裡面，程式是要拆開成很多封裝完整的部份的，中間必須透過固定的介面來溝通，設計成這樣的架構一定是要多付出一些成本設計這些中間的架構和包裝各個部份的，運作的時候也稍微沒有效率一點，可是這些付出都慢慢會從別的地方回收的。(剛才講的神奇電腦，從某個角度看其實就是賈伯斯一直在推的 Apple's quality magic，從很早以前的 Macintach 到現在的 iXXX 都看得到那種精品的整體感，不過使用精品就是要付出代價的，也才叫做精品嘛!! 就像 iPhone 的電池，如果能自己換多簡單...那蘋果就少了可以服務客戶的機會了，會想自己換的也應該不是會使用精品的客戶層嘛! 不講這個，但是要設計可以更換的物件介面勢必要有相當的損失，要有客戶可以自己拆開來的機殼，電池的形狀要是規則有完整包裝的，客戶安裝的時候要不容易出錯的...)

在開始架構程式的時候，有的時候會覺得運用已有的物件會對演算法造成一些限制(就好像叫你算 123+789 等於多少，不准用心算，不准用筆算，限制你一定要用電腦裡面的小算盤完成反而有一點頭痛)，很多人因此偏好不用現有的東西組合程式，所有的功能都從頭開始寫；寫一個小的作業程式這樣子是沒有什麼不好啦，可以練習很多基本的語法，可是如果寫很多性質相同

的小程式，可能就開始覺得每次從頭開始寫有點討厭了；在程式規模逐步擴大時更會發現設計、除錯、修改反而開始變慢了，很多時候發現有太多的彈性反而是以工程化的方式來撰寫程式時很大的阻礙。

從好的地方去看，程式在設計的時候嚴格地根據其他物件的介面來設計的話(這是基本的 design by contract 概念)，不需要去思考或是配合其他物件裡面的實作邏輯，是一種適當分工的好方法。另一方面，你也看到當各個物件的規格確立了以後，物件的功能有限，比較容易設計，也容易去撰寫測試正確性的單元測試程式碼。

程式除錯：

使用者抱怨程式的功能有問題時，如果程式是用物件組合出來的，你可以決定把比較有可能造成問題的物件用同樣介面、同樣功能的物件換掉，觀察程式的表現來判斷是否為主要錯誤的原因；反之如果你沒有任何物件的描述，你只能直接去修改某一個函式裡的某一列程式，改了以後還能夠保證程式正確運作嗎? 像這樣有問題的程式又太多了，可能發生錯誤的組合形式也太多了，造成程式很難測試、很難修改。換一種說法，透過嚴格封裝的物件來組合程式，物件之間有很清楚的界限來釐清責任，任何物件發生錯誤時錯誤都不會隨意跨越物件的邊界，所以用相同功能、相同介面的物件來置換可以很快發現問題，可以很快得到正確的功能，可以很快提昇效能。當整個組合的程式裡面有錯誤發生時，要先看各個物件間介面的表現是不是正常，不要急急忙忙地跑去 debug 某一個物件的內部，很多時候因為物件使用其他物件的介面出現錯誤，被使用的物件因為裡面有適當保護，不會直接當掉，而會發出一些警告的訊息，此時使用這個物件的地方有可能就會出現邏輯上面的錯誤，這種情況就好像你在使用手機上某一個應用程式，你以為它應該怎樣表現的，事實上不是，你隨便按，基本上手機也不會爆掉，但是你可能進入一個不爽的迴圈，這種狀況常常可以透過下層物件的訊息或是 assert 錯誤，很快地找到問題的來源進行修改。

修改以及維護

基本上修改功能的時候還是要運用物件組合的方式來完成新的功能，但是因為透過組合物件的設計比較模組化，所以當你準備修改一個過去寫的物件化程式時，應該能夠很快地了解程式組合的架構，找到設計上的切入點，如果有些物件無法提供滿意的功能，因為界面的描述清楚，可以清楚地把不需要的物件切除，換用其他的物件來組合新的功能。最差最差的情況，更換了物件以後產生新的問題，那就再換舊的物件回來吧，至少有原來的功能。(你有沒有遇見過程式改一改以後產生新的錯誤，連原來有的功能都不太能運作了呢? 好悲情!)

4. [10] 有一個類別，在設計他的幾個成員函式時都發現函式內部需要使用一個動態配置大小的整數陣列，每個函式中都要配置，使用完畢後再釋放掉，設計的人覺得這樣子有點麻煩，不如把配置/釋放的程式碼放到類別的建構元/解構元中，把配置好的陣列指標變成類別的資料成員，請問這樣的設計有什麼問題? 不這麼做的話該怎麼解決程式碼重複好多次的問題呢?

Sol:

因為題目說使用完畢都要釋放掉，可見這些動態配置的陣列裡面存放的資料是暫時性的，沒有要給這個物件的其他成員函式使用，所以就概念上不應該把它設計成類別的資料成員，而應該讓它維持是一個函式內的區域變數，如果因為配置的空間很大，擔心動態的記憶體配置很耗費計算機的資源，可以嘗試用額外的物件來掌管記憶體的配置，或是有時候可以考慮函式裡面 static 的變數，如果純粹是覺得程式碼重複很多次的問題，應該要寫一個輔助的私有函式來解決，不要把不需要記錄在類別裡當成物件狀態的東西定義在類別的資料成員的地方。會造成程式閱讀時很大

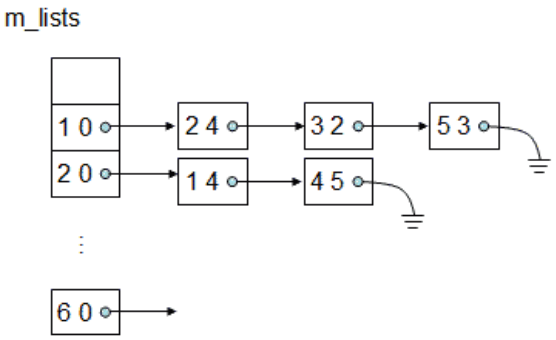
的困擾。

5. [60] 作業二及作業三中配合下圖左資料檔案 data1.dat 可以用下列程式碼讀取資料，並且以串列 (list) 建立所謂的相鄰矩陣 AdjcencyMatrix

```
6               ; number of nodes 1..6
9               ; number of edges
1 2 4           ; edge (1,2), distance 4
1 3 2           ; edge (1,3), distance 2
1 5 3
2 4 5
3 4 1
3 5 6
3 6 3
4 6 6
5 6 2
```

```
01 struct AdjMatNode
02 {
03      AdjMatNode(void):ID(0), distance(0), next(0) {}
04      AdjMatNode(int id, int distance):ID(id), distance(distance), next(0) {}
05      int ID;
06      int distance;
07      AdjMatNode *next;
08 };
09
10 class AdjacencyMatrix
11 {
12 public:
13      AdjacencyMatrix(ifstream &);
14      virtual ~AdjacencyMatrix(void);
15 private:
16      AdjMatNode *m_lists;
17      int m_numNodes;
18      int m_numEdges;
19 };
20
```

```
21 AdjacencyMatrix::AdjacencyMatrix(ifstream &infile)
22 {
23      char dummyStr[50]; int i;
24
25      infile >> m_numNodes;
26      infile.getline(dummyStr, sizeof(dummyStr));
27
28      infile >> m_numEdges;
29      infile.getline(dummyStr, sizeof(dummyStr));
30
31      m_lists = new AdjMatNode[m_numNodes+1];
32
33      for (i=0; i<m_numEdges; i++)
34      {


        }
    }
```



a) [10] 請配合上圖中程式，完成 AdjacencyMatrix(ifstream&) 建構元中 for 迴圈的部份，實作上圖右側的資料結構?

Sol:

```
int i, node1, node2, distance;
AdjMatNode *ptr;
for (i=0; i<m_numEdges; i++)
{
      infile >> node1 >> node2 >> distance;
      infile.getline(dummyStr, sizeof(dummyStr));

      ptr = &m_lists[node1];
      while (ptr->next != 0)
            ptr = ptr->next;
      ptr->next = new AdjMatNode(node2, distance);

      ptr = &m_lists[node2];
      while (ptr->next != 0)
            ptr = ptr->next;
      ptr->next = new AdjMatNode(node1, distance);
}
```

或是如果允許串列裡的順序顛倒過來，再修改一下 AdjMatNode 的建構元的話

```
AdjMatNode(int id, int distance, AdjMatNode *next=0):ID(id), distance(distance), next(next) {}
int i, node1, node2, distance;
for (i=0; i<m_numEdges; i++)
{
      infile >> node1 >> node2 >> distance;
      infile.getline(dummyStr, sizeof(dummyStr));
      m_lists[node1].next = new AdjMatNode(node2, distance, m_lists[node1].next);
      m_lists[node2].next = new AdjMatNode(node1, distance, m_lists[node2].next);
}
```

b) [10] 請完成對應的解構元 ~AdjacencyMatrix() 函式?

Sol:

```
AdjacencyMatrix::~AdjacencyMatrix()
{
    int i;
    AdjMatNode *ptr1, *ptr2;
    for (i=0; i<m_numNodes; i++)
    {
        ptr1 = m_lists[i].next;
        while (ptr1 != 0)
        {
            ptr2 = ptr1;
            ptr1 = ptr1->next;
            delete ptr2;
        }
    }
    delete[] m_lists;
}
```

c) [16] 在 Prim's MinimumSpanningTree 的作業中，AdjacencyMatrix 這個物件最主要需要依序提供每一個指定節點的所有相鄰節點，通常可以用類似 iterator 的概念來設計介面，提供兩個介面函式，一是 int getFirst(const int vertex, int *distance) const，另一是 int getNext(const int vertex, int *distance) const，getFirst()回傳第一個相鄰節點的 id，回傳 0 代表沒有任何相鄰節點，*distance 為那個邊的長度；getNext()回傳下一個相鄰節點的 id，回傳 0 代表已經沒有其它相鄰節點了，*distance 為那個邊的長度，請實作這兩個介面函式，並且適當地在類別中增加狀態變數? 請問這個介面中兩個 const 的意義? [6]

Sol:

```
class AdjacencyMatrix
{
public:
    ...
    int getFirst(const int vertex, int *distance) const;
    int getNext(const int vertex, int *distance) const;
    ...
private:
    ...
    int m_curVertex;
    AdjMatNode *m_iter;
    ...
};

---------------------------

#include <assert.h>

AdjacencyMatrix::AdjacencyMatrix(ifstream &infile): m_curVertex(0), m_iter(0)
{
    ...
}

int AdjacencyMatrix::getFirst(const int vertex, int *distance) const
{
    assert((vertex >= 1) && (vertex <= m_numNodes));
    m_iter = m_lists[vertex].next;
    if (m_iter == 0)
    {
        m_curVertex = 0;
        return 0;
    }
    m_curVertex = vertex;
    *distance = m_iter->distance;
    return m_iter->ID;
}

int AdjacencyMatrix::getNext(const int vertex, int *distance) const
{
    assert((vertex >= 1) && (vertex <= m_numNodes));
    if (m_curVertex != vertex)
        return getFirst(vertex, distance);
    else
        m_iter = m_iter->next;
    if (m_iter == 0)
    {
        m_curVertex = 0;
        return 0;
    }
    *distance = m_iter->distance;
    return m_iter->ID;
}
```

上面這個實作裡其實是沒有辦法編譯的，主要是因為 const 成員函式的關係；

int AdjacencyMatrix::getFirst(const int vertex, int *distance) const;

第一個 const 是說 vertex 是一個常數變數，在函式裡不會被改變，不過在這裡用途不大，通常是配合參考變數或是指標變數來使用會比較有意義，因為本來這個變數就是用 call-by-value 的方式傳進函式裡，vertex 在函式裡就算被改掉了其實也不影響呼叫端

第二個 const 是保證這個 getFirst 成員函式不改變 AdjacencyMatrix 物件的狀態(所有的資料成員)，可是因為在這裡我們嘗試直接把 iterator 和 container 實作在一個類別裡，所以 getFirst 其實會改變 AdjacencyMatrix 物件的狀態，所以編譯的時候看到 m_curVertex=...或是 m_iter=...都會錯，怎麼辦? 簡單一點就是把後面的 const 拿掉就可以了；或是如下面範例用函式內的 static 變數來實作這兩個成員函式所需要的狀態:

```
class AdjacencyMatrix
{
public:
      ...
      int getFirst(const int vertex, int *distance) const;
      int getNext(const int vertex, int *distance) const;
      ...
private:
      int getNext(int vertex, bool getFirst, int *distance) const;
      ...
      ...
};
int AdjacencyMatrix::getFirst(const int vertex, int *distance) const
{
      return getNext(vertex, true, distance) ;
}

int AdjacencyMatrix::getNext(const int vertex, int *distance) const
{
      return getNext(vertex, false, distance) ;
}

int AdjacencyMatrix::getNext(int vertex, bool getFirst, int *distance) const
{
      static int curVertex=0;
      static AdjMatNode *iter=0;

      assert((vertex >= 1) && (vertex <= m_numNodes));
      if (getFirst) curVertex = 0;
      if (curVertex != vertex)
      {
            iter = m_lists[vertex].next;
            curVertex = vertex;
      }
      else
            iter = iter->next;
      if (iter == 0)
      {
            curVertex = 0;
            return 0;
      }
      *distance = iter->distance;
      return iter->ID;
}
```

進階一點還是應該把 iterator 和 container 分開來兩個類別實作，以後課程裡會說明。

d) [4] 請問為什麼在這個設計裡不用一個二維陣列來實作而用串列來實作這個相鄰矩陣?

Sol:

從我們作業的資料檔案來看，其實用二維陣列來實作似乎沒有什麼問題，但是如果一個圖的節點很多，比方說 10000 個節點，從平面幾何上來看一個節點能夠相鄰的節點有限，所以很快地會發現這是一個所謂的稀疏矩陣(Sparse matrix)，每一個節點的相鄰節點雖然最多有 9999 個，但是實際上可能只有 8 個 10 個而已，用固定的二維 10000x10000 陣列來實作就會用掉很多的空間，因此通常都是用串列來實作，雖然這個相鄰矩陣描述的是一個靜態的圖，不會有新增或是刪除節點或是邊的狀況，用串列好像有一些功能沒有用到，但是如果考量輸入資料檔案的格式，因為使用串列，所以可以在每讀進一列 node1 node2 distance 資料時就在 node1 和 node2 串列中各新增一個節點就可以了；另外一種實作方法是用動態配置的陣列，看每個節點有幾個相鄰節點來決定每一列的元素個數，但是在資料檔案設計時可能就需

要先給定相鄰節點的個數，否則就需要先讀檔案一遍，計算出每一個節點有幾個相鄰節點，才能夠配置足夠大的陣列。

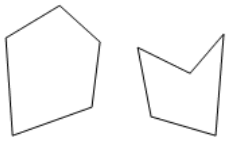
e) [20] 物件化的程式最大的優點就在於每一個物件都有明確的介面，相同介面的物件就可以互換，很多時候簡單的實作雖然效率不好，或是擴展性不好，可是可以快速地驗證 Prim 演算法的正確性，請重新設計一個運用二維矩陣實作的 AdjacencyMatrix，請依序定義 AdjacencyMatrix 類別，實作建構元，解構元與上述 getFirst(), getNext()介面

Sol:

```
#define MAX_NUM_NODES 30

class AdjacencyMatrix
{
public:
    AdjacencyMatrix(ifstream &);
    virtual ~AdjacencyMatrix(void);
    int getFirst(const int vertex, int *distance);
    int getNext(const int vertex, int *distance);
private:
    int m_curVertex;
    int m_iter;
    int m_numNodes;
    int m_numEdges;
    int m_matrix[MAX_NUM_NODES][MAX_NUM_NODES];
};


#include <assert.h>


AdjacencyMatrix::AdjacencyMatrix(ifstream &infile)
    : m_curVertex(0), m_iter(0)
{
    int i, j;
    char dummyStr[50];

    infile >> m_numNodes;
    infile.getline(dummyStr, sizeof(dummyStr));
    assert(m_numNodes+1<MAX_NUM_NODES);

    infile >> m_numEdges;
    infile.getline(dummyStr, sizeof(dummyStr));

    for (i=0; i<=m_numNodes; i++)
        for (j=0; j<=m_numNodes; j++)
            m_matrix[i][j] = 0;

    int node1, node2, distance;
    for (i=0; i<m_numEdges; i++)
    {
        infile >> node1 >> node2 >> distance;
        infile.getline(dummyStr, sizeof(dummyStr));
        m_matrix[node1][node2] = m_matrix[node2][node1] = distance;
    }
}

AdjacencyMatrix::~AdjacencyMatrix()
{
}

int AdjacencyMatrix::getFirst(const int vertex, int *distance)
{
    assert((vertex >= 1) && (vertex <= m_numNodes));
    for (m_iter=1; m_iter<=m_numNodes; m_iter++)
        if (m_matrix[vertex][m_iter]>0)
            break;
    if (m_iter>m_numNodes)
    {
        m_curVertex = 0;
        return 0;
    }
    m_curVertex = vertex;
    *distance = m_matrix[vertex][m_iter];
    return m_iter;
}

int AdjacencyMatrix::getNext(const int vertex, int *distance)
{
    assert((vertex >= 1) && (vertex <= m_numNodes));
    if (m_curVertex != vertex)
        return getFirst(vertex, distance);
    for (m_iter++; m_iter<=m_numNodes; m_iter++)
        if (m_matrix[vertex][m_iter]>0)
            break;
    if (m_iter>m_numNodes)
```

```
        {
            m_curVertex = 0;
            return 0;
        }
        *distance = m_matrix[vertex][m_iter];
        return m_iter;
    }
```

6. [40] 在一個圖形介面應用軟體中，需要處理各種平面上的多邊形，下面我們實作一個 Polygon 類別以及相關的資料型態：如右圖所示平面上的多邊形由多個相連的線段構成，左側的多邊形所有的內角都小於 180 度我們說是一個凸多邊形(convex polygon)，右側的多邊形有一個超過 180 度的角則是凹多邊形(concave polygon)，我們可以用順時鐘排列的頂點來表示一個多邊形，右圖是一個資料檔案，檔案內依序是頂點個數、第一個頂點的座標是(0,0)、第二個頂點的座標是(0,1)、第三個頂點的座標是(1,1)、第四個頂點的座標是(1,0)；也就是描述平面上一個正方形。請回答下述問題：



```
4
0 0
0 1
1 1
1 0
```

a. [5] 請定義一個 Point 結構，這個結構有兩個欄位分別是 double 型態的 x 座標和 y 座標；請定義一個 Polygon 類別，其中包含一個資料成員記錄動態配置的 Point 頂點陣列以及一個資料成員記錄有幾個頂點

Sol:

```
struct Point
{
    Point(void);
    Point(double x, double y);
    double x;
    double y;
};

#include <fstream>
using namespace std;

#include "Point.h"

class Polygon
{
public:
    Polygon(void);
    Polygon(ifstream &ifs);
    Polygon(const Polygon &polygon2);
    virtual ~Polygon(void);

    bool contains(const Point &point) const;
    bool isConvex(void) const;
    double area(void) const;

private:
    Point *m_pts;
    int m_npts;
};
```

b. [5] 請定義 Polygon(ifstream &infile) 建構元，由上面檔案串流中建構多邊形，頂點陣列請多配置一個頂點將多邊形封閉起來，例如 (0,0), (0,1), (1,1), (1,0), (0,0)

Sol:

```
Polygon::Polygon(ifstream &ifs):m_npts(0), m_pts(0)
{
    int i;
    ifs >> m_npts;
    m_pts = new Point[m_npts+1];
    for (i=0; i<m_npts; i++)
        ifs >> m_pts[i].x >> m_pts[i].y;
    m_pts[m_npts] = m_pts[0];
}
```

c. [7] 請定義一個拷貝建構元

Sol:

```
Polygon::Polygon(const Polygon &polygon2)
    :m_npts(polygon2.m_npts)
```

```
{
    int i;
    if (m_npts>0)
    {
        m_pts = new Point[m_npts+1];
        for (i=0; i<=m_npts; i++)
            m_pts[i] = polygon2.m_pts[i];
    }
}
```

d. [3] 請定義解構元

Sol:

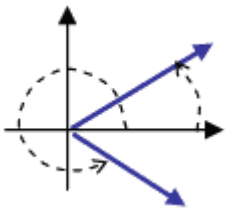
```
Polygon::~Polygon(void)
{
    delete[] m_pts;
}
```

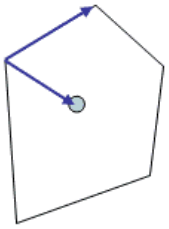
下面這三題希望你運用其他的物件介面來設計，請不要因為覺得從頭到尾設計起來比較容易，就在 Polygon 類別裡寫完全部的演算法，的確有時候很多人合作好像比自己一個人做還麻煩，但是合作的潛在能量比較大，比較容易擴充功能

e. [10] 請定義並且實作一個 double Polygon::area() const 成員函式來計算多邊形的面積；在撰寫這個成員時請假設已經有一個 Triangle 類別，你只要用三個 Point 型態的頂點就可以建構出一個 Triangle 物件，建構元為 Triangle::Triangle(Point vertices[3])，這個 Triangle 類別還提供一個 double Triangle::area() const 的介面可以計算三角形的面積

Sol: 以下我們只考量適用於凸多邊形比較簡單的演算法來驗證如何運用其他的物件實作自己這個物件的某些功能，進一步適用於各種狀況的演算法要到電腦圖學去學了

```
#include "Triangle.h"
…
double Polygon::area() const
{
    int i;
    Point vertice[3];
    double total = 0.0;
    vertice[0] = m_pts[0];
    for (i=1; i<m_npts-1; i++)
    {
        vertice[1] = m_pts[i];
        vertice[2] = m_pts[i+1];
        total += Triangle(vertice).area();
    }
    return total;
}
```





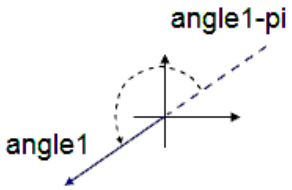
f. 或 g. 請選其中之一即可

f. [10] 請定義並且實作一個 bool Polygon::contains(const Point &point) const 成員函式來判斷 point 是否落於該多邊形內部；在寫這個成員函式時請假設已經有一個 Line 類別，你只要用兩個 Point 型態的點就可以建構出一個 Line 物件，建構元為 Line::Line(Point &start, Point &end)，如上左圖這個 Line 類別提供一個 double Line::angle() const 的介面，可以計算由 start 到 end 這個向量與 x 軸的夾角(圖中虛線所示)，單位為弧度，範圍在 0 到 2π；如上右圖如果 point 相對於每一邊來說都在同一側則是在多邊形之內；pi = atan(1.0)*4.0; atan()是 math.h 裡的函式

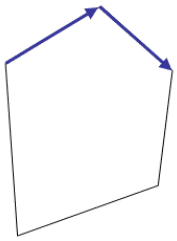
Sol:

```
#include "line.h"
…
bool Polygon::contains(const Point &point) const
{
    double angle1, angle2, pi=atan(1.0)*4.0;
    Line line1, line2;
    int i;

    if (m_npts > 2)
    {
        for (i=0; i<m_npts; i++)
        {
            line1 = Line(m_pts[i], m_pts[i+1]);
```

```
                angle1 = line1.angle();
                line2 = Line(m_pts[i], point);
                angle2 = line2.angle();
                if (angle1 > pi)
                {
                    if ((angle2 > angle1) || (angle2 < angle1 - pi))
                        return false;
                }
                else
                {
                    if ((angle2 > angle1) && (angle2 < angle1 + pi))
                        return false;
                }
            }
            return true;
        }
        return false;
    }
```



g. [10] 請定義並且實作一個 bool Polygon::isConvex() const 成員函式來判斷是否為一個凸多邊形；同樣假設有上題中的 Line 類別，如下圖：如果所有相鄰邊的夾角都小於 π 就是凸多邊形



Sol:

```
#include "line.h"
…
bool Polygon::isConvex(void) const
{
    double angle1, angle2, pi=atan(1.0)*4.0;
    Line line1, line2;
    int i;

    if (m_npts > 2)
    {
        line1 = Line(m_pts[m_npts-1], m_pts[m_npts]);
        for (i=0; i<m_npts; i++)
        {
            angle1 = line1.angle();
            line2 = Line(m_pts[i], m_pts[i+1]);
            angle2 = line2.angle();
            if (angle1 >= pi)
            {
                if ((angle2 > angle1) || (angle2 < angle1 - pi))
                    return false;
            }
            else
            {
                if ((angle2 > angle1) && (angle2 < angle1 + pi))
                    return false;
            }
            line1 = line2;
        }
        return true;
    }
    return false;
}
```

或是簡化一點點，用暫時性的 Line 物件取代

```
bool Polygon::isConvex(void) const
{
    double angle1, angle2, pi=atan(1.0)*4.0;
    int i;

    if (m_npts > 2)
    {
        angle1 = Line(m_pts[m_npts-1], m_pts[m_npts]).angle();
        for (i=0; i<m_npts; i++)
        {
            angle2 = Line(m_pts[i], m_pts[i+1]).angle();
            if (angle1 >= pi)
            {
                if ((angle2 > angle1) || (angle2 < angle1 - pi))
                    return false;
            }
            else
            {
                if ((angle2 > angle1) && (angle2 < angle1 + pi))
                    return false;
            }
            angle1 = angle2;
        }
        return true;
    }
    return false;
}
```