國立台灣海洋大學資訊工程系 C++ 程式設計 期中考解答

姓名：_____

系級：_____

學號：_____

98/04/14

考試時間：**10:00 – 12:00**

試題敘述蠻多的，看清楚題目問什麼，請針對重點回答，總分有 **120**，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)、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，不可以逗留喧嘩
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] 請簡要說明物件導向程式中物件封裝的目的? 以及 C++ 中實現物件封裝的語法? 請問 C++ 封裝的單位是類別還是物件? (舉例說明)

Sol:

a. 物件的封裝最主要是希望物件的狀態(資料)由物件自己維護，物件對外界僅提供完整定義、清楚且簡潔的操作介面，外界無法直接干預物件內部資料內容與實作的方法，封裝後的物件其功能明確地由其介面定義出來，相同介面的物件可以替代使用。

b. 由 class / struct 語法配合成員函式以及 private/protected/public 保留字規範資料與成員函式的存取權限來實作“封裝”，例如:

```
class XXX
{
public:
        attributes: data member
        operations: member function
protected:
        attributes: data member
        operations: member function
private:
        attributes: data member
        operations: member function
};
```

c. C++ 中以類別為封裝的單位，某一類別的 a 物件可以直接存取/修改同一類別的 b 物件中的資料例如:

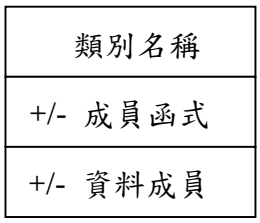
```
void CComplex::add(CComplex &rhs)
{
        m_real += rhs.m_real; // m_real 為 a 物件中的成員, rhs.m_real 則為 b 物件中的成員
        m_imaginary += rhs.m_imaginary;
}
```

2. [5] 請解釋類別和物件的差別?
   [5] 構成類別的要件為何?
   [5] 請運用 UML 圖形表示一個類別並註明各部份代表的意義?

Sol:

a. 類別是製作物件的範本，類別中規範物件需要哪些記憶體空間來存放資料，個別的型態為何，類別中也以成員函式的方式規範所有此類別物件的操作介面以及回應訊息的方法，規範此類別物件狀態的改變方法。物件則是實際軟體執行時一個具有類別內所規範特性的個體。一個類別可以產生許多的物件。

b. 構成類別最主要的要件包括: 類別名稱，成員函式 (介面函式，輔助函式，建構元，解構元，運算子等等)，資料成員



c. UML 圖形如右，其中 + 代表 public 存取, - 代表 private 存取

3. [5] 請舉例說明在 C++ 中如何在一個類別中實作物件間的關聯性?
   [10] 請問在物件導向設計中 “聚合” (aggregation) 和 “組合” (composition) 各代表什麼意義，有什麼差別，在 C++ 中通常如何實作?

Sol:

a. 在 C++ 中通常透過資料成員來實現與其他物件之間的關聯性，使得一個物件可以透過關聯使用到另外一個物件所提供的服務，其中又可以使用指標，物件，或是參考來實作這種關連性，例如:
   甚至於比較複雜的關連性可以設計一個物件來表達，例如: course - teacher - student

```
01. class Bag
02. {
03.    ….
04. private:
05.         Ball *m_balls[3];
06. };
```

```
01. class Bag
02. {
03.    ….
04. private:
05.         Ball m_balls[3];
06. };
```

```
01. class Bag
02. {
03.    ….
04. private:
05.         Ball &m_ball;
06. };
```

b. 聚合和組合基本上都是藉由已經設計好的物件來設計新的物件，由一整組物件合作提供功能，這兩種合作的關係差別在於物件的生命期，組合的關係中下層物件和上層物件的生命週期相同，當上層物件結束時，下層物件一併結束，聚合時生命週期則沒有那麼嚴格的要求。

c. 在 C++ 中組合常常以物件成員實作，聚合常常以物件的指標來實作，組合也可以運用物件的指標來實作，但是聚合不能夠使用物件成員來實作。

4. [10] 請問右圖程式片段中兩個 static 保留字代表的意義各為何?

```
01.  static void append(string str)
02.  {
03.          static string textline = "";
04.          if (str[0] == '.') textline += str;
05.  }
```

Sol:

第一個 static 限制此函式為 file scope 的函式，只有在相同 cpp 檔案中才可以呼叫到，第二個 static 代表此變數在資料區段中，不在系統堆疊上，每一次執行此函式時都會用到同樣的儲存空間，上一次執行的結果會保留在該變數中，下一次函式執行時就可以看到所儲存的資料，如此可以有限制地在兩次函式執行的時候傳遞一些資料。

5. [20] 下列程式中使用 stdio.h 以及 malloc.h 函式庫進行輸入輸出，請以 C++ 中的 iostream 函式庫及 new/delete/new[]/delete[] 改寫之

```
01 #include <stdio.h>
02 #include <malloc.h>
03 void main()
04 {
05        FILE *fp;
06        int i, j, ndata, **data;
07        fp = fopen("test.dat", "r");
08        fscanf(fp, "%d", &ndata);
09        data = (int **) malloc(ndata*sizeof(int*));
10        for (i=0; i<ndata; i++)
11        {
12              data[i] = (int *) malloc((ndata-i)*sizeof(int));
13              for (j=0; j<ndata-i; j++)
14                    fscanf(fp, "%d", &data[i][j]);
15        }
16        fclose(fp);
17        for (i=0; i<ndata; i++)
18              free(data[i]);
19        free(data);
20 }
```

test.dat

```
7
1 2 3 4 5 6 7
7 6 5 4 3 2
0 1 0 1 0
2 1 0 2
1 0 -1
1 3
5
```

Sol:

```
01 #include <fstream>
02 using namespace std;
03 void main()
04 {
05        ifstream ifs("test.dat");
06        int i, j, ndata, **data;
07        ifs >> ndata;
08        data = new int*[ndata];
09        for (i=0; i<ndata; i++)
10        {
11              data[i] = new int[ndata-i];
12              for (j=0; j<ndata-i; j++)
13                    ifs >> data[i][j];
14        }
15        for (i=0; i<ndata; i++)
16              delete[] data[i];
17        delete[] data;
18 }
```

6. [50] 請參考下圖中 max heap 程式?

```
01 #include "heap.h"
02
03 void siftdown(Element heap[], int i, int size)
04 {
05        int child;
06        Element temp = heap[i];
07
08        while (2*i <= size)
09        {
10              child = 2 * i;
11              if (child < size &&
12                    compareMonthlyPayment(&heap[child+1], &heap[child])==1)
13                    child = child + 1;
14              if (compareMonthlyPayment(&heap[child], &temp)==1)
15                    heap[i] = heap[child];
16              else
17                    break;
18              i = child;
19        }
20        heap[i] = temp;
21 }
22
23 int compareMonthlyPayment(Element *data1, Element *data2)
24 {
25      if (data1->monthly_payment > data2->monthly_payment)
26              return 1;
27      else if (data1->monthly_payment == data2->monthly_payment)
28              return 0;
29      else
30              return -1;
31 }
32
33 Element del(Element heap[], int *size)
34 {
35      Element tmp = heap[1];
36      heap[1] = heap[*size];
37      *size = *size - 1;
38      siftdown(heap, 1, *size);
39      return tmp;
40 }
41
```

```
42 void insert(Element data, Element heap[], int *size)
43 {
44        int i;
45        *size = *size + 1;
46        i = *size;
47        while (i>1 &&
48              compareMonthlyPayment(&data, &heap[i/2])==1)
49        {
50              heap[i] = heap[i/2];
51              i = i / 2;
52        }
53        heap[i] = data;
54 }
55
56 void heapify(Element heap[], int size)
57 {
58        int i;
59        for (i=size/2; i>=1; i--)
60              siftdown(heap, i, size);
61 }
```

```
1 struct Element {
2        char id[10];
3        char name[30];
4        int hours;
5        int payrate;
6        int monthly_payment;
7 };
```

a. [5] 請撰寫一個 swap 函式 void swap(Element *dataptr1, Element *dataptr2), 將兩個指標所指向的 Element 型態資料對調

Sol:

```
1 void swap(Element *dataptr1, Element *dataptr2)
2 {
3        Element tmp;
4        tmp = *dataptr1;
5        *dataptr1 = *dataptr2;
6        *dataptr2 = tmp;
7 }
```

b. [10] 請撰寫一個 sort 函式 void sort(Element heap[], int size), 直接呼叫上述 heapify(), siftdown(), 以及 swap() 函式將一個未排序過的 Element 型態資料陣列根據 monthly_payment 欄位的數值由小排到大

Sol:

```
1 void sort(Element heap[], int size)
2 {
3        heapify(heap, size);
4        for (int i=size; i>=2; i--)
5        {
6              swap(&heap[1], &heap[i]);
7              siftdown(heap, 1, i-1)
8        }
9 }
```

c. [10] 請撰寫一個 MaxHeap 類別的宣告 (不需要撰寫成員函式的內容, 部份成員函式的實作請參考 d,e,f,g), 此類別的物件基本上維護一個 Element 型態資料的 maximum heap, 任何時候其資料都維持合法的大小順序, 其資料請以一個動態配置的 Element 型態陣列來存放 (請決定類別裡需要哪些資料, 並且決定上圖中 insert(), del(), siftdown(), heapify()以及 compareMonthlyPayment() 何者應該為此類別的介面?理由為何, 請修改每一函式的參數)

Sol:

```
01 class MaxHeap
02 {
03 public:
04         MaxHeap(Element data[], int size, int maxSize);
05         ~MaxHeap();
06         void insert(Element data);
07         Element del();
08 private:
09         void siftdown(int i);
10         void heapify();
11         int compareMonthlyPayment(Element *data1, Element *data2);
12 private:
13         int m_maxSize;
14         int m_size;
15         Element *m_data;
16 };
```

MaxHeap 類別最主要的介面是 void insert(Element) 和 Element del(), 主要讓客戶端程式能夠把元素加入 MaxHeap 中, 同時能夠由 MaxHeap 中得到目前最大的資料, 其它的函式, 例如 siftdown(int), heapify(), 以及 compareMonthlyPayment(Element *, Element*) 只是實作此類別時輔助性的函式, 並不是介面函式, MaxHeap 中最主要的資料是 m_data 這個 Element 型態的指標變數, 所記錄的資料是在建構元中配置的 Element 陣列的指標, m_maxSize 則是記錄 m_data 陣列所能夠容納的最大 heap 容量, m_data 陣列有 m_maxSize+1 個元素, m_size 所記錄的是 m_data 內資料的筆數,資料是由 m_data[1] 一直到 m_data[m_size]。

d. [5] 請運用初始化串列 (initialization list) 語法撰寫一個建構元函式 (constructor), 傳入一個 Element 型態陣列、陣列內資料的個數、以及預期接受最多的元素個數, 並建構一個正確的 MaxHeap 出來?

Sol:

```
1 MaxHeap::MaxHeap(Element data[], int size, int maxSize)
2         : m_maxSize(maxSize), m_size(size), m_data(new Element[maxSize+1])
3 {
4         int i;
5         assert(size<=maxSize);
5         for (i=0; i<m_size; i++)
6                 m_data[i] = data[i];
7         heapify();
8 }
```

e. [5] 請撰寫解構元函式 (destructor)?

Sol:

```
1 MaxHeap::~MaxHeap()
2 {
3         delete[] m_data;
4 }
```

f. [5] 請實作 MaxHeap::siftdown() 函式?

Sol:

```
01 void MaxHeap::siftdown(int i)
02 {
03        int child;
04        Element temp = m_data[i];
05
06        while (2*i <= m_size)
07        {
08              child = 2 * i;
09              if (child < m_size &&
10                    compareMonthlyPayment(&m_data[child+1], &m_data[child])==1)
11                    child = child + 1;
12              if (compareMonthlyPayment(&m_data[child], &temp)==1)
13                    m_data[i] = m_data[child];
14              else
15                    break;
16              i = child;
17        }
18        m_data[i] = temp;
19 }
```

g. [10] 請實作 MaxHeap 類別的設定運算子 (assignment operator)?

Sol:

```
class MaxHeap
{
…
public:
      MaxHeap& operator=(MaxHeap &rhs);
…
};

01 MaxHeap& MaxHeap::operator=(MaxHeap &rhs)
02 {
03        if (&rhs == this) return *this;
04        delete[] m_data;
05        m_data = new Element[rhs.m_maxSize+1];
06        m_maxSize = rhs.m_maxSize;
07        m_size = rhs.m_size;
08        for (int i=1; i<m_size; i++)
09              m_data[i] = rhs.m_data[i];
10        return *this;
11 }
```