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：_____

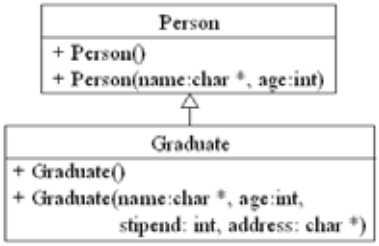
96/06/26

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，該舉例說明的如果只有簡單的答案，只會得到基礎分數，總分有 150，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] 請舉例說明 C++ 中的靜態多型機制有哪些?
   b. [5] 請舉例說明 C++ 中的動態多型是指什麼?
   c. [5] 請問上述兩種機制運作時最主要的差異點是什麼?
   d. [5] 請問多型機制對程式設計者而言最主要的目的是什麼?
   e. [10] 請根據下圖左，運用 C++ 繼承的語法定義 Person 及 Graduate 類別，並且對這兩個類別各定義兩個建構元函式 (請注意 Graduate 類別的第二個建構元函式需要運用 Person 類別的第二個建構元函式來建構，建構元的實際內容不需要撰寫)?
   f. [10] 請問下圖右程式中哪幾列敘述會有編譯的錯誤? [4] 什麼錯誤? [6]




```
1.  Person    person1("Joe", 19), person2, *personPtr;
2.  Graduate graduate1("Michael", 24, 6000, "8899 Storkes"), graduate2, *graduatePtr;
3.  person2 = graduate1;
4.  graduate2 = person1;
5.  personPtr = &graduate1;
6.  graduatePtr = &person1;
```

Sol:

a. C++ 支援靜態多型的語法主要是函式的多載 (function overloading) 與運算子的多載 (operator overloading)，其中運算子的多載在 C 裡就有，例如 "+" 號有好幾種不同的意義，只是不允許程式設計者自己重新定義，例如: int x1, x2; double y1, y2; x1 = x1 + x2; y1 = y1 + y2; 這兩個 "+" 實際上就執行不一樣的動作，在 C++ 中則允許程式設計者根據程式的需要定義一些類別的運算子所對應的動作,。
函式的多載則是同一個函式名稱可以重複定義多個函式，各個函式以不同的參數型態或是參數個數來區分。例如: int service(int p1, int p2); int service(double p);

b. C++ 中動態的多型是指透過多型指標或是多型參考呼叫物件的成員函式時，呼叫到的函式是由執行時實際上所指到的物件的型別 (type) 來決定的，例如:

```
class Base
{
public:
      virtual void service() {}
};
class Derived : public Base
{
public:
      void service() {}
};
```

```
void main()
{
      Base bObj, *bPtr;
      Derived dObj;

      bPtr = &bObj; bPtr->service(); //  執行 Base::service()

      bPtr = &dObj; bPtr->service(); //  執行 Derived::service()
}
```

c. 不論是靜態的多型或是動態的多型，主要目的是希望程式設計者可以用同樣的語法/程序去操作概念相同的物件，上述靜態多型和動態多型最主要的差別在於靜態多型是編譯器在翻譯你的程式的時候就可以決定到底你的指令是要呼叫哪一個函式或是要執行哪些步驟，動態多型則是需要等到程式執行的時候，視當時資料的內容為何才能夠決定到底要執行哪一個函式。

d. “多型”這個機制是希望程式設計者可以用同樣的語法去操作概念相同的物件，希望程式設計者能夠專注在重要的觀念，不要被細微的差異混淆，能夠用最簡單的方法來操作這些物件，“簡單”是大家都喜歡的，“簡單”能夠減少不必要的錯誤，雖然有的時候“簡單”也可能代表他的擴展性比較差，但是整體而言常常還是節省程式設計者許多的時間，簡化程式設計工作的複雜度。

e.

```
class Person
{
public:
        Person();
        Person(char *name, int age);
private:
        …
};

Person::Person()
{
…
}

Person::Person(char *name, int age)
{
…
}
```

```
class Graduate : public Person
{
public:
        Graduate();
        Graduate(char *name, int age, int stipend, char *address);
};

Graduate::Graduate()
{
…
}

Graduate::Graduate(char *name, int age,
                        int stipend, char *address)
        : Person(name, age)
{
…
}
```

f. 第 4 列 graduate2 = person1; 語法是不正確的，編譯器會發現它從等號右邊的 person1 物件裡拿不到足夠的資訊來設定等號左邊的 graduate2 物件
第 6 列 graduatePtr = &person1; 語法也是不正確的，編譯器會發現等號右邊的記憶體位址的變數型態和等號左邊的指標的型態沒有直接關係，它不允許你將 person1 物件的位址存在 graduatePtr 這個指標變數裡，簡單的驗證方法就是想說如果編譯器允許這個設定，那麼萬一在這一列之後增加 graduatePtr->service() 的呼叫，而 service() 又是 Graduate 類別才有的函式，在執行這個呼叫時就掛了，因為 graduatePtr 實際上指的是一個 Person 類別的物件，它可能根本沒有能夠支援 service() 成員函式的資料 (參考第 4 題的 Derived::service() 函式)。

2. a. [5] 請舉例說明設計物件系統時常用的委託 (delegation) 機制是什麼? 目的為何?
   b. [5] 請問在封裝一個類別的時候你如何決定哪些資料應該作為這種物件的資料成員?
   c. [5] 請問 Object-based language 和 Object-oriented language 之間最主要的差別在於?
   d. [5] 請問成員函式宣告成 virtual 的目的是什麼?
   e. [10] 請說明動態繫結 (dynamic binding) 的表現為何，是用什麼方法實作的?
   f. [5] 請舉例說明繼承時衍生類別會得到基礎類別的哪兩種東西?

Sol:

a. 設計物件系統時，某一個物件常常因為自己沒有管理相關的資料，而需要請求擁有資源的物件來協助它完成一些功能，如果它可以直接和擁有資源的物件溝通 (例如：它知道那個物件的指標)，它就可以直接請求服務，不過有的時候它沒有管道和擁有資源的人直接溝通，此時就需要透過其它的物件來請求目標物件提供服務，這些中間的物件基本上接受來源物件的請求，但是本身沒有辦法完成，因此只是把需求很快地轉向目標物件，由目標物件來提供服務，這種機制就是所謂的 “委託” 機制。

b. 在封裝一個物件時，基本上就是替這個物件設計操作它的介面，這個時候也必須確定這個物件

管理哪些資源，例如在設計一個“學生”的物件時，我們會在這個物件裡記錄學生的姓名，ID 等等資料，這些資料以後就會由這個物件來管理，要使用這些資料時都要透過物件的介面來存取，像剛才舉的這個例子基本上很容易決定是否為“學生”物件管理的資料，有些時候你會發現決定資料屬於哪一個物件好像不那麼直覺，這個時候就要從資料耦合的角度來檢視，簡單地說就是看看如果資料放在物件 A 中的話，會不會有很多其他類別的物件需要常常要求 A 來提供運用此資料的服務，甚至要求 A 來提供原始資料，如果是的話，也許這些原始資料應該放在最常需要它的那個物件裡面由它來管理，如果有很多類別的物件都非常需要這個資料，放在哪一個物件裡都躲不掉密切的資料耦合，那就需要類似專屬管理資料的物件，或是資料庫了。

c. 使用 Object-based language 你可以設計物件，可以設計介面，可以封裝物件，可以運用一整組物件合作來完成你的程式功能，Object-oriented language 除了提供物件的抽象化功能之外，最主要是提供繼承的語法和動態的多型，大大加強程式碼重用的特質。

d. 在 C++ 中如果一個類別有一個成員函式宣告成 virtual 以後，編譯器會替這個類別的物件準備一個虛擬函式表 (virtual function table)，每一個虛擬函式的函式指標在這個虛擬函式表中放在固定的位置，在呼叫這個虛擬函式的時候，編譯器會根據是否使用多型指標或是參考，決定是否使用動態繫結，如果編譯器決定使用靜態繫結的話，雖然是 virtual 函式，卻完全不使用這個虛擬函式表。

e. 動態繫結也叫做晚期繫結 (late binding)，是指程式在執行的時候，才能根據物件的種類來判定某一個透過多型指標的函式呼叫究竟是要執行哪一個類別的成員函式，主要是透過函式指標 (function pointer) 以及虛擬函式表 (virtual function table) 來實作的一種機制，編譯器在看到例如 ptr->service() 這樣的函式呼叫時，不一定會把它直接翻譯成低階的 CALL service 敘述，最主要的原因是在編譯時根本不知道 ptr 這個指標指向什麼型態的變數，有可能需要執行 Base::service() 也有可能需要執行 Derived::service()，必須等到執行時才會知道，因此編譯器在這個地方必須使用間接的函式呼叫方式，先由 ptr 所指到的物件的虛擬函式表中讀出 service() 函式的記憶體位址，然後再移轉控制到該記憶體位址去執行。

f. 衍生類別會由基礎類別得到所有基礎類別內配置的資料儲存空間，所有定義的函式，以及所有基礎類別定義的介面，例如:

```
class Base
{
public:
        void service1() {
                cout << m_data1 << endl;
        }
private:
        int m_data1;
};

class Derived: public Base
{
public:
        void service2(){
                cout << m_data1
                        << m_data2 << endl;
}
private:
        int m_data2;
};
```

```
void main()
{
        Derived dObj;
        dObj.service1();
        dObj.service2();
}
```

dObj 中有一份 Base 型態的父類別物件，所以包含了 m_data1 這個變數的存放空間，同時也有所定義的成員函式 service1() 的程式馬，除此之外，無論 bObj 可以如何被使用，dObj 就可以被那樣的方法來使用。

3. 請參考圖中 IntArray 類別的定義和 fun1(), fun2(), fun3(), main() 函式的定義，回答下列問題：

```
1.  class IntArray
2.  {
3.  public:
4.          IntArray(int size):data(0) {
5.                  assert(size>0);
6.                  data = new int[size];
7.          }
8.          ~IntArray() {
9.                  delete[] data;
10.         }
11.         int &operator[](int index) {
12.                 return data[index];
13.         }
14. private:
15.         int *data;
16. };
```

a. [5] 請問如果在第 22 列當 id 值為 2 時發生一個例外狀況(rand()回傳值為偶數)後，程式的流程為何 (請以列號表示)

b. [5] 接題 (a) 請問在螢幕上列印了哪些資料?

c. [5] 請問 IntArray 的解構元在程式執行到哪一列時會被呼叫到?

d. [5] 請問如果在第 47 列改為 int *iary = new int[100]; 並且在 49 列之後加入 delete[] iary; 的話，這個程式在什麼狀況下會遇到記憶體遺漏 (memory leakage) 的問題?

```
17. void fun1(int id)
18. {
19.         if (rand()%2 == 0)
20.         {
21.                 cout << "       fun1() exception occurs\n";
22.                 throw id;
23.         }
24.         cout << "leaving fun1() normally\n";
25. }
```

```
26. void fun2()
27. {
28.         int *ptr1, *ptr2;
29.         try
30.         {
31.                 IntArray ary1(100);
32.                 fun1(1);
33.                 IntArray ary2(200);
34.                 fun1(2);
35.         }
36.         catch (int id)
37.         {
38.                 cout << "       fun2() exception catched, id=" << id << "\n";
39.                 cout << "       fun2() deleting all resources\n";
40.                 cout << "       leaving fun2() with exception\n";
41.                 throw;
42.         }
43.         cout << "leaving fun2() normally\n";
44. }
```

```
45. void fun3()
46. {
47.         IntArray iary(100);
48.         fun2();
49.         iary[1] = 10;
50. }
51. void main()
52. {
53.         cout << "Entering main()\n";
54.         srand(time(0));
55.         int *ptr=0;
56.         try
57.         {
58.                 IntArray ary(300);
59.                 fun3();
60.         }
61.         catch (int)
62.         {
63.                 cout << "       main() exception catched\n";
64.         }
65.         cout << "leaving main()\n";
66. }
```

Sol:

a. 22 -> 23 -> 25 ->
35 -> 8 -> 9 -> 10 ->
8 -> 9 -> 10 ->
36 -> 37 -> 38 -> 39 -> 40 -> 41 -> 42 -> 44 ->
50 -> 8 -> 9 -> 10 ->
60 -> 8 -> 9 -> 10 ->
61 -> 62 -> 63 -> 64 -> 65 -> 66

b. Entering main()
leaving fun1() normally
fun1() exception occurs
fun2() exception catched id=2
fun2() deleting all resources
leaving fun2() with exception
main() exception catched
leaving main()

c. 如題 a, 呼叫 ~IntArray() 的地方共有四個， 35 列呼叫 2 次， 50 列及 60 列各呼叫一次

d. 如題 a, 只要在 fun1() 或是 fun2() 裡發生任何例外，就會跳過 delete[] iary; 的敘述， 也就會發生記憶體遺漏的問題

4. 下圖中是一個編譯時不會有任何錯誤，但是執行時會遇見錯誤的程式:

a. [5] 請問為什麼第 30 列傳進一個 Derived 物件陣列, 和第 22 列參數要求 Base 物件陣列並不同，但是編譯器可以接受?

```
1.  #include <iostream>
2.  using namespace std;
3.  class Base {
4.  public:
5.      virtual void service() {
6.          cout << "Base::service()\n";
7.      }
8.  };
9.
10. class Derived : public Base {
11. public:
12.     Derived():m_id(m_serial++) {}
13.     void service(){
14.         cout << "Derived::service() id="
                << m_id << "\n";
15.     }
16. private:
17.     static int m_serial;
18.     int m_id;
```

```
20. int Derived::m_serial=0;
21.
22. void callingService(Base bArray[10]) {
23.     int i;
24.     for (i=0; i<10; i++)
25.         bArray[i].service();
26. }
27.
28. void main() {
29.     Derived dArray[10];
30.     callingService(dArray);
31. }
```

b. [5] 請問程式執行到第 25 列時是呼叫到哪一個函式?

c. [5] 請問你預期程式會列印什麼資料? 迴圈會執行幾次然後出錯呢? (為什麼?)

d. [10] 上面這種錯誤基本上是 C++ 的多型和 C 的陣列的衝突: 本來 Derived 物件是可以看成是 Base 型態的物件的，但是 C 的陣列運用指標來實作，使得 Derived 物件陣列也就自動看成是 Base 物件陣列，解決這種衝突的方法就是在這種狀況下不要用 C 的陣列，請使用 template array，例如 C++ 標準函式庫的 vector。請修改第 22 列至 26 列的函式定義，運用 vector 來達成 callingService() 函式想要做的工作，請將第 29 列也運用 vector 宣告成陣列，請問編譯器看到第 30 列的函式呼叫時會發出什麼樣的錯誤訊息?

Sol:

a. 在 C/C++ 中內建的陣列是運用指標來實作的，第 30 列 callingService(dArray); 中 dArray 雖然宣告為 Derived 型態的陣列: Derived dArray[10]; 但是對於編譯器而言，dArray 卻是一個 Derived 型態的指標， 同樣地第 22 列 void callingService(Base bArray[10]) 的參數雖然宣告為 Base 型態的陣列: Base bArray[10]; 但是對於編譯器而言，bArray 卻是一個 Base 型態的指標，繼承的基本概念就是 Derived 類別的物件可以視為 Base 類別的物件，也就是一個 Derived 物件的位址自然可以放在 Base 型態的指標變數中，所以編譯器可以接受第 30 列的用法。

b. 敘述 bArray[i].service() 對於編譯器而言其實是 (bArray+i)->service()，這是一個透過多型指標的函式呼叫，所以呼叫到的函式是 Derived::service()

c. 會列印 Derived::service() id=0 然後就發生執行錯誤，主要因為 bArray+1 所指到的並不是下一個 Derived 類別的物件，基本上這個 bArray+1 的動作只會增加 sizeof(Base) 個位元組，所以當

Derived::service() 執行時 bArray+1 是指在 bArray 所指的 Derived 物件之中，並無法得到正確的結果

d. void callingService(vector<Base> &bArray) {
```
    int i;
    for (i=0; i<10; i++)
        bArray[i].service();
}
void main() {
    vector<Derived> dArray(10);
    callingService(dArray);   // 編譯錯誤
}
```
上面的編譯錯誤是因為 vector<Derived> 並不是 vector<Base> 型態，vector<Derived> 也沒有繼承 vector<Base>，所以並不能當成是 vector<Base> 物件傳入 callingService() 函式

5. a. [5] 右圖是一個將一個整數變數內容拷貝到另一個整數變數的函式，請修改此程式成一個函式樣板 (template function)，可以拷貝任意兩個同一型態 (例如: int, double, char, …) 的物件的資料?

```
1.  void clone(int &target,
                int &source) {
2.      target = source;
3.  }
```

b. [5] 請問配合題 (a) 的答案, 編譯器在編譯左下圖主程式時會產生哪幾個函式一起編譯? (請描述這些函式的參數型態)

```
1.  struct Point { int x, y; };
2.  void main() {
3.      int a, b, c;
4.      double d, e;
5.      Point x, y;
6.      …
7.      clone(a, b);
8.      clone(b, c);
9.      clone(d, e);
10.     clone(x, y);
11. }
```

```
1.  class Student {
2.  public:
3.      Student(char *name);
4.      ~Student();
5.  private:
6.      char *m_name;
7.  };
8.  Student::Student(char *name) {
9.      m_name = new char[strlen(name)+1];
10.     strcpy(m_name, name);
11. }
12. Student::~Student() {
13.     delete[] m_name;
14. }
```

c. [5] 請問題 (a) 答案的函式樣板對於右上圖中的這個 Student 類別的物件可以正常運作嗎? (執行時會遇見什麼問題)

d. [10] 接上題，請修改 Student 類別使得下圖程式可以正確地拷貝 Student 物件的資料

```
1.  Student obj1, obj2;
2.    …
3.  clone(obj1, obj2);
```

e. [5] 接上題，請問修改過的類別可以用於下圖之程式嗎? 請說明原因?

```
1.  vector<Student> studAry1, studAry2;
2.  …
3.  clone(studAry1, studAry2);
```

Sol:

a. template <class T>
```
void clone(T &target, T &source) {
```

```
    target = source;
}
```

b. 共產生三個 template function, 分別是

```
void clone(int &target, int &source);
void clone(double &target, double &source);
void clone(Point &target, Point &source);
```

c. 如果程式中使用

```
Student s1;
…
Student s2;
…
clone(s1, s2);
```

可以編譯，可以執行，但是在其中一個物件解構之後會產生 dangling reference 的問題，最主要的問題就在於 Student 類別沒有定義 assignment operator，所謂的 Big Three 中缺了 assignment operator。

d. 為了使 clone 的動作正確執行，必須替 Student 類別定義 assignment operator 如下：

在類別定義中增加

```
Student &operator=(Student &rhs);
```

在 cpp 檔案裡增加

```
Student &Student::&operator=(Student &rhs)
{
    if (this == &rhs) return *this;
    delete m_name;
    m_name = new char[strlen(rhs.m_name)+1];
    strcpy(m_name, rhs.m_name);
    return *this;
}
```

e. 可以，主要的原因是 vector<> 類別有定義 assignment operator，如果 Student 類別也有定義 assignment operator，直接運用 operator= 複製兩個 vector<Student> 物件就可以正確地運作。