國立台灣海洋大學資訊工程系 C++ 程式設計 期末考參考解答

姓名：_____ 系級：_____ 學號：_____

100/06/14

1. [12] 請寫出下列程式在螢幕上之輸出?

```
01 #include <iostream>
02 using namespace std;
03
04 class Base {
05 public:
06        Base();
07        Base(int i);
08        ~Base();
09 private:
10        int m_i;
11 };
12
13 Base::Base(): m_i(0){
14        cout << "Base 0" << endl;
15 }
16
17 Base::Base(int i): m_i(i){
18        cout << "Base " << i << endl;
19 }
20
21 Base::~Base(){
22        cout << "Destruct Base " << m_i << endl;
23 }
24
```

```
25 class Derived: public Base{
26 public:
27        Derived();
28        Derived(int i);
29        ~Derived();
30 };
31
32 Derived::Derived(){
33        cout << "Derived" << endl;
34 }
35
36 Derived::Derived(int i): Base(i) {
37        cout << "Derived" << endl;
38 }
39
40 Derived::~Derived(){
41        cout << "Destruct Derived" << endl;
42 }
43
44 int main(){
45        Base b;
46        Derived d(2);
47        return 0;
48 }
```

Sol:

Base 0
Base 2
Derived
Destruct Derived
Destruct Base 2
Destruct Base 0

2. [10] 請寫出下列程式在螢幕上之輸出? [5]請問七次函式呼叫中哪些是動態繫結?

Sol:

```
01 #include <iostream>
02 using namespace std;
03
04 class A{
05 public:
06        int f();
07        virtual int g();
08 };
09
10 int A::f(){
11        return 1;
12 }
13
14 int A::g(){
15        return 2;
16 }
17
18 class B: public A{
19 public:
20        int f();
21        virtual int g();
22 };
23
24 int B::f(){
25        return 3;
26 }
```

```
27
28 int B::g(){
29        return 4;
30 }
31
32 class C: public A{
33 public:
34        virtual int g();
35 };
36
37 int C::g(){
38        return 5;
39 }
40
41 int main(){
42        A *aPtr, aObj;
43        B bObj;
44        C cObj;
45        aPtr = &aObj; cout << aPtr->f() << endl; cout << aPtr->g() << endl;
46        aPtr = &bObj; cout << aPtr->f() + aPtr->g() << endl;
47        aPtr = &cObj; cout << aPtr->f() + aPtr->g() << endl;
48        cout << cObj.g() << endl;
49        return 0;
50 }
```

1
2
5
6
5

```
45      aPtr = &aObj; cout << aPtr->f() << endl; cout << aPtr->g() << endl;
46      aPtr = &bObj; cout << aPtr->f() + aPtr->g() << endl;
47      aPtr = &cObj; cout << aPtr->f() + aPtr->g() << endl;
48      cout << cObj.g() << endl;
```

動態繫結只有在兩個條件都滿足時才會發生: 1.需要定義成虛擬函式 2. 需要透過多型指標/參考來呼叫, 其他不滿足這兩個條件的呼叫都是靜態繫結

3. [8] 請將下列程式在螢幕上之輸出完整寫出?

```
01 #include <iostream>
02 using namespace std;
03
04 class A{
05 public:
06      A(int x=1) {a=x;}
07      void f() {a+=2;}
08      virtual int g() {a+=1; return a;}
09      int h() {f(); return a;}
10      int j() {return g();}
11 private:
12      int a;
13 };
14
```

```
15 class B: public A{
16 public:
17      B(int y=5) {b=y;}
18      void f() {b+=10;}
19      int g() {b+=7; return b;}
20 private:
21      int b;
22 };
23
24 int main(){
25      A obj1;
26      B obj2;
27      cout << obj1.h() << endl;
28      cout << obj1.j() << endl;
29      cout << obj2.h() << endl;
30      cout << obj2.j() << endl;
31      return 0;
32 }
```

Sol:

3
4
3
12

[5] 請以下列程式為例說明動態繫結的程式運作?

Sol:

動態繫結發生在函式 j() 中呼叫 g(); 時, 因為呼叫 g() 相當於 this->g();, g() 又是一個虛擬函式, 因此是一個透過多型指標呼叫的動態繫結, 第 30 列 obj2.j() 呼叫時執行 A::j() 函式, 此時 this 指標指向 obj2 物件, 呼叫 g() 時執行的是 B::g() 函式, 此即動態繫結的運作, 在這個例子裡運用動態繫結的方法常常在 template method pattern 中看到, 函式 j() 就是 template method。

[5] 請問下列程式中將函式的實作寫在類別的宣告裡面有什麼作用?

Sol:

在 C++ 中這樣子寫代表一個 inline function, inline function 和 macro 的運作方式很像, 最主要是授權編譯器決定是否在函式呼叫的地方將函式參數轉換以及函式內容的程式碼直接展開, 希望以重複的程式碼節省下函式呼叫時額外的運算負擔, 以提昇程式執行時的效率。

[3] 從物件導向的觀點來看"將函式的實作寫在類別的宣告裡面"的缺點是什麼?

Sol:

物件的實作應該和物件的界面有適當的區隔, 才不會誤導物件的客戶端基於物件的實作來使用物件, 物件化的程式中物件的客戶端應該完全基於界面來設計(Design by Contract)自己

的程式，如此在物件實作修改或是以同功能物件或是衍生類別的物件取代時才不會發生錯誤或是效能的低落。

[5] “將函式的實作寫在類別的宣告裡面”和動態繫結的目標有什麼衝突? (兩種機制能不能同時運作? 編譯器如何處理?)

Sol:

Inline function 是在程式編譯時如果可以確定呼叫哪一個函式，函式本體夠短，同時編譯器可以取得該函式內容時，將函式內容在呼叫處直接展開，動態繫結在編譯的時候完全無法確定要呼叫哪一個函式，所以無法做 inline 展開，編譯器會自行忽略此處的 inline 展開要求。

4. [20] 請宣告一個有理數類別 Rational，其中有兩個私有的 int 型態資料 m_numerator 以及 m_denumerator 分別代表分子及分母，請定義一個建構元 Rational(int numerator, int denumerator)，如果使用者沒有提供參數的話，請將 numerator 預設為 0, denumerator 預設為 1；假設在類別裡已經寫好一個輔助成員函式 int gcd(int x, int y) 來計算最大公因數，請覆載 (overload) 下列運算子: a. 二元加號: 將兩個有理數相加並回傳結果，請不要改變加號前後兩個物件的內容，請運用成員函式 gcd() 撰寫一個私有的成員函式 normalize() 將有理數化為最簡; b. 一元的 post-increment ++: 回傳該有理數物件後將該有理數值加一; c. 型態轉換運算子將此有理數轉換為 double 型態。

Sol:

```
01 class Rational
02 {
03 public:
04        Rational(int numerator=0, int denumerator=1);
05        Rational operator+(const Rational &rhs) const;
06        Rational operator++(int);
07        operator double () const;
08 private:
09        void normalize();
10        int gcd(int x, int y);
11        int m_numerator;
12        int m_denumerator;
13 };
14
15 Rational::Rational(int numerator, int denumerator):
16        m_numerator(numerator), m_denumerator(denumerator) {
17        normalize();
18 }
19
20 void Rational::normalize() {
21        if (m_numerator == 0)
22              return;
23        int tmp = gcd(m_numerator, m_denumerator);
24        m_numerator /= tmp;
25        m_denumerator /= tmp;
26 }
```

```
28 Rational Rational::operator+(const Rational &rhs) const {
29        Rational result;
30        result.m_denumerator = m_denumerator * rhs.m_denumerator;
31        result.m_numerator = m_numerator * rhs.m_denumerator +
                               rhs.m_numerator * m_denumerator;
32        result.normalize();
33        return result;
34 }
35
36 Rational Rational::operator++(int) {
37        Rational result(*this);
38        m_numerator += m_denumerator;
39        normalize();
40        return result;
41 }
42
43 Rational::operator double () const {
44        return m_numerator / (double) m_denumerator;
45 }
```

5. [10] 請撰寫一個 rotate()樣板函式 (template function)，這個函式有一個樣板參數，函式本身有一

```
void main() {
      vector<int> x;
      x.push_back(1); x.push_back(2); x.push_back(3); x.push_back(4);
      vector<double> y;
      y.push_back(.7); y.push_back(.8); y.push_back(.9); y.push_back(1.0); y.push_back(1.1);
      display(x); rotate(x); display(x);
      display(y); rotate(y); display(y);
}
```

```
1 2 3 4
4 1 2 3
0.7 0.8 0.9 1 1.1
1.1 0.7 0.8 0.9 1
```

個 vector 型態的參數，函式的功能是將 vector 裡面的元素右移一位，亦即 x[i] → x[i+1]，最後一個元素則移到第一個元素的位置上；請再撰寫一個 display()樣板函式，這個函式有一個樣板參數，函式本身有一個 vector 型態的參數，函式的功能是將 vector 裡面的元素列印在螢幕上；測試的 main()函式如下圖左所示，執行結果如右下圖:

Sol:

```
01 #include <vector>
02 #include <iostream>
03 using namespace std;
04
05 template <class Type>
06 void rotate(vector<Type> &data)
07 {
08        int i;
09        int len = data.size();
10        Type tmp = data[len-1];
11        for (i=len-1; i>=1; i--)
12              data[i] = data[i-1];
13        data[0] = tmp;
14 }
15
```

```
16 template <class Type>
17 void display(vector<Type> data)
18 {
19        int i;
20        int len = data.size();
21        for (i=0; i<len; i++)
22              cout << data[i]<< ' ';
23        cout << endl;
24 }
```

6. 在一個繪圖系統中有如下的 Window 類別，請根據這個類別宣告回答下列問題:

```
1. class Window
2. {
3. public:
4.        virtual void draw() = 0;
5.        void move(int x, int y) { this->x = x; this->y = y; }
6. private:
7.        int x, y;
8. };
```

a. [5] 請問第 4 列中 = 0 代表什麼意思? (這種函式稱為什麼函式? 這個類別稱為什麼類別? 使用這樣的類別和使用一般的類別有什麼不同?)

Sol:

一個類別中只要有一個虛擬函式後面加上 = 0 代表這個類別(此例中即 Window)為一個抽象基礎類別(Abstract Base Class, ABC)，這個類別不能夠直接產生物件，這個 draw()函式稱為純粹的虛擬函式(pure virtual function)。

b. [5] 請繼承 Window 類別宣告一個 Dialog 類別 (請寫出完整的類別宣告，不需要實作各個成員函式的內容，但是這個 Dialog 類別裡需要重新定義 draw(), move() 函式)

Sol:

```
class Dialog: public Window
{
public:
      Dialog();
      ~Dialog();
      virtual void draw();
      void move(int x, int y);
};
```

c. [5] 請問 Dialog 和 Window 這兩個類別的物件需要具有什麼性質我們才可以運用繼承的語法來實作?

```
1.    void main()
2.    {
3.          Dialog dialog;
4.          int num;
5.
6.          Window *ptrAry[100];
7.          ptr[0] = &circle;
8.          ...
9.          drawAllWindows(ptrAry, num);
10.   }
```

Sol:

一個 Dialog 類別的物件如果都是 Window 類別的物件 (A Dialog IS-A Window.)，或是在系統中每一個 Dialog 類別的物件都可以取代 Window 類別的物件配合 Window 類別的客戶來

運作 (Liskov Substitution Principle, LSP)。

d. 請參考右圖程式回答問題

i. [5] 請問 ptrAry[i] 這種基礎類別的指標在物件導向的程式中我們稱為什麼指標，在使用時有什麼特性?

Sol:

多型指標，使用時可以指向任意衍生類別的物件，可以實作異質的容器物件，運用多型指標呼叫虛擬函式可以根據物件的型態來動態繫結所呼叫的函式。

ii. [10] 請根據 Window 類別的界面，撰寫一個全域的函式 drawAllWindows(Window *ptr[], int numWindows)，其參數是一個 Window 指標的陣列，陣列內有 numWindows 個 Window 衍生類別的物件的指標，這個函式需要呼叫所有傳入的 Window 物件的 draw() 成員函式來繪出所有的物件，另外請說明為什麼我們說這樣的函式具有 “Old codes call new codes” 的性質?

Sol:

```
void drawAllWindows(Window *ptr[], int numWindows)
{
    int i;
    for (i=0; i<numWindows; i++)
        ptr[i]->draw();
}
```

撰寫這個函式的時候也許程式設計者只實作了 Window 和 Dialog 兩個類別，以及這兩個類別的 Window::draw()和 Dialog::draw()函式，如果有一天程式設計者繼承 Dialog 類別設計一個 FileDialog 的類別，並且實作 FileDialog::draw()函式，如果 ptr[i]指向一個 FileDialog 物件，執行 drawAllWindows()的 ptr[i]->draw()時仍然可以呼叫到新的程式碼 FileDialog::draw()。