Object Oriented Programming
V22.0470 Fall 2001

Final Exam
Solution

Write the answers to question 1 and 2 on this sheet. Write the other answers in the exam booklet.

1. True/False. Please circle the correct answer on this sheet.

(a) T The C++ declaration const int *p; declares a pointer whose value can change, but which cannot be used to change the value contained in the memory location it points to.

(b) T Java does not have the equivalent of C++’s private and protected derivation.

(c) F In C++, exceptions raised by the throw construct must be objects belonging to a subclass of Exception.

(d) T A fundamental difference between the vector and list class templates in the C++ STL is that elements of a vector are contiguous in memory.

(e) F If two Java classes implement the same interface, they are considered subtypes of each other.

(f) T An object of class Thread only starts a new thread executing when the start() method is called.

(g) T F “C++” stands for “Complexity and more complexity”.

(h) F The C++ copy constructor is invoked when an object is passed by reference to a function.

(i) F A Java program can be written without ever using the word class.

(j) T A C++ program can be written without ever using the word class.

2. Multiple Choice. Circle the one desired response.

(a) Which one of the following Java code will not compile?

i. class A { A() {} } class B extends A { B(int x) {} }

ii. class A { A(int x) {} } class B extends A { B(int x) { super(x); } }

iii. class A { A() {} } class B extends A { B() {} }

iv. class A { A(int x) {} } class B extends A { B(int x) {} }

(b) Given the declaration

    template<class T> class A {};

which one of the following will not compile?

i. class B: public A<int> {};

ii. template <class T> class B: public A {};

iii. template <class T> class B: public A<T> {};

iv. template <class T> class B: public A<int> {};


1
(c) Given the class definition

```cpp
class A {
    public:
        int operator()() { return 5; }
        A operator+(int) { return *this; }
};
```

which one of the following will not compile?

1. `A f; int x = f();`
2. `A x,y; x = x+y;`
3. `A x; x = x+3;`
4. `A f,x; f = f+x();`

(d) Java's support for GUI programming (provided by the java.awt and java.swing packages) is considered “event driven” because

i. the programmer must write loops that continually check if events, such as mouse clicks, have occurred.

ii. **events, such as mouse clicks, happen asynchronously and cause parts of the GUI code to be executed.**

iii. when events, such as mouse clicks, occur, Java exceptions are raised.

iv. just getting GUI code to work correctly is a notable event.

(e) Which one of the following statements is not true about multiple inheritance in C++?

i. It allows a class to be considered a subclass of two other classes.

ii. It allows objects of the child class to have all the members of two parent classes.

iii. **If a class is derived from two parent classes which both have a method with the same signature, the compiler will generate an error message.**

iv. A class can be derived from more than two classes.

3. Consider the definition of a class that acts like an array with bounds-checking.

```cpp
class my_array {
    public:
        my_array(int s): size(s) { a = new int[size]; }
        int &operator[](int i);
        int get_size() { return size; }
    private:
        int size;
        int *a;
};
```

(a) Give the code for `my_array`'s `operator[]` method, as one would define it outside the class.

```cpp
int &my_array::operator[](int i)
{
    if ((i>=0) && (i<size))
        return a[i];
    else
        throw "Array reference out of bounds";
}
```
(b) Overload the insertion (<<) and extraction (>>) operators so that they behave as follows:

- The insertion operator outputs all the elements of the array, separated by spaces, to the output stream.
- The extraction operator reads the elements of the array from the input stream.

Do not modify the my_array class to accomplish this.

```cpp
ostream &operator<<(ostream &o, my_array &m)
{
    int the_size = m.get_size();
    for (int i = 0; i < the_size; i++)
        o << m[i] << " ";
    return o;
}

istream &operator>>(istream &str, my_array &m)
{
    int the_size = m.get_size();
    for (int i = 0; i < the_size; i++)
        str >> m[i];
    return str;
}
```

(c) What is the best way for class my_array be modified to allow for more efficient insertion and extraction operators?

The inefficiency in part (b) results from having to call the myarray::operator[], which performs bounds checking, every time the insertion and extraction operators want to access and element of my_array. This is due to the fact that the actual array is within my_array is private. If you add the declarations:

```cpp
friend ostream &operator<<(ostream &, my_array &);
friend istream &operator>>(istream &, my_array &);
```

to the my_array class, then the insertion and extraction operators can be rewritten to access the internal array a directly without having bounds checking going on.

(d) Why is it necessary for the return type of operator[] to be “int &”?

So that, given a variable m of type my_array, one can write things like

```cpp
m[i] = 6;
```

If operator[] returned it’s result by value (i.e. copy), m[i] could not appear on the left hand side of an assignment statement.

4. In Java, write the code to construct two frames (of type java.awt.Frame or javax.swing.JFrame, it doesn’t matter). In the second frame, there should be a button that, when clicked on, causes a button to appear in the first frame. FYI, in class Button there is a method

```java
void addActionListener(ActionListener l)
```

where ActionListener is defined by

```java
interface ActionListener {
    void actionPerformed(ActionEvent e);
}
```
There are many different ways to do this. Here's one way:

```java
import java.awt. *
import java.awt.event. *

class test {

    static Frame f1 = new Frame();
    static Frame f2 = new Frame();

    public static void main(String[] args) {
        f1.setSize(400,400);
        f2.setSize(200,200);
        f1.setLayout(new FlowLayout());
        f2.setLayout(new FlowLayout());
        Button b2 = new Button("click here");
        b2.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                f1.add(new Button("Here I am"));
                f1.setVisible(true);
            }
        });
        f2.add(b2);
        f1.setVisible(true);
        f2.setVisible(true);
    }
}
```

5. (a) In C++, write a template function `sum` that can be used to sum up the elements of an array, as long as addition is defined on those elements. For example, given your template function definition, the code

```cpp
int a[] = {1,2,3,4};
float b[] = {1.2,3.4,6.7};
cout << sum(a,4) << endl; // the parameters are the array
cout << sum(b,3) << endl; // and the size of the array
```

would print

```
10
11.3
```

```cpp
template<class T>
T sum(T *a, int size)
{
    T total = a[0];
    for(int i=1;i<size;i++)
    { 
        total = total + a[i];
    }
    return total;
}
```

(b) Given the following class definition,
class student {
  public:
    student(int a) : age(a) {};
  private:
    int age;
};

what method could be added to the student class in order to be able to pass an array of students to sum, where the desired result is a student whose age is the sum of the ages of the students in the array? Write the code for this method.

You would add the operator+ method:

```cpp
student operator+(student &other)
{ student y(age + other.age);
  return y;
}
```

(c) How can an array of students be passed to sum without adding a new method to the student class? Write the code that accomplishes this. If necessary, you can modify the student class, but you must keep age private (and cannot add a new method).

Instead of adding operator+ as a method in student, simply overload the global + to handle students (analogous to overloading the global << and >> operators):

```cpp
student operator+(student &s1, student &s2)
{ student y(s1.age + s2.age);
  return y;
}
```

In order for the global + to access the age field, the declaration:

```cpp
friend student operator+(student &s1, student &s2);
```

has to be added to the student class.

6. (a) Define the terms dynamic overloading (also known as dynamic dispatch and overriding) and static overloading.

The term overloading refers to giving several different functions the same name. The terms static and dynamic refer to when the use of this name is resolved (i.e. disambiguated). Static overloading is when the compiler can determine which function is being referred to, because the functions have different numbers and/or types of parameters. Dynamic overloading refers to overloading that is resolved during execution. This generally occurs due to re-definition of a method of a parent class in the child class, with the same number and types of parameters.

(b) Write some Java code that illustrates both dynamic and static overloading. Be sure to include both the definitions of statically and dynamically overloaded methods and the use of them.

```java
class A {
  int f(int x, int y) { return x+y; } //these two f's are an example
  int f(int x) { return x+10; } //of static overloading. They are
} //unrelated
```
class B extends A {
    int f(int x) { return x-10; } // this f(int) overrides the f(int) in class A,
    } // which is an example of dynamic overloading

class temp {
    static int g(A p) {
        int i = p.f(3,4); // compiler can determine what f this is. No run-time
        // overload resolution needed.
        i = i + p.f(7); // run-time overload resolution needed here to figure out which
        // f called, depending on whether p really is an A or a B.
        return i;
    }
}

(c) Do the same thing as in part (b), but in C++.

class A {
    int f(int x, int y) { return x+y; }
    virtual int f(int x) { return x+10; }
};

class B: public A {
    int f(int x) { return x-10; }
};

int g(A &p) {
    int i = p.f(3,4);
    i = i + p.f(7);
    return i;
}