COMP151 Final Exam, Dec. 15 1999
Model answers and grading instructions.

Question 1 (18 points): Polymorphism
The three kinds of polymorphism we have seen are

1. Function overloading. For example:

```
int min(int a, int b){
  if(a <= b) return a;
  else return b;
}

string min(const string& a, const string& b){
  if(a <= b) return a;
  else return b;
}

// ...
int i = min(14, 67);
string s = min("Hello", "World");
```

2. The virtual function mechanism, combined with pointers/refences to base classes. For example:

```
class B {
public:
  virtual void f() { cout << "B::f()\n"; }
};

class D: public B {
public:
  virtual void f() { cout << "D::f()\n"; }
};

// ...
B* bp = new D;
bp->f()
```

3. Templates. For Example:

```
template<class T>
T min(T a, T b){
  if(a <= b) return a;
  else return b;
}

// ...
int i = min(14, 67);
string s = min("Hello", "World");
```


Grading: for each of the three kinds of polymorphism, 6 points can be earned:

2 points for (more or less) correctly naming the kind of polymorphism, and
4 points for a correct example.

Of course, students will probably come op with many different examples, and that is OK.

For function overloading and templates, it is OK if they only give 2 function definitions and a template definition, respectively, without calling the function.

For the virtual function definition, it is not sufficient if they only give a base class and a derived class with an overloaded function; the use through a pointer of reference must be given too. Substract 2 points for the example of the virtual function mechanism if the use is missing.

Question 2 (12 points): Class Hierarchies
Note: the example of bad programming style that was mentioned in the text of the assignment refers to the function `h()`. This function is not virtual in the base class `Top`, but it is overridden in the derived class `Middle` anyway (and made virtual too!).

Solution:

part 1: contains an error. It is not posible to declare an object op type `Top`, since that class contains a pure virtual function.

part 2: prints `Bottom::f()`

part 3: prints `Middle::f()` (even though `middle` is copied from `bottom`, we don't get polymorphism here, since we don't use pointers or references.)

part 4: prints

```
Bottom::f()
Bottom::g()
Top::h()
```

(The last line can be explained by the fact that `Top::h()` is not virtual.)

part 5: prints

```
Bottom::f()
Bottom::g()
Bottom::h()
```

(The last line can be explained by the fact that `Middle::h()` has been made virtual.)

part 6: contains an error: it is not possible to have a pointer to a derived class (i.e., a `Bottom*`) point to a base class (i.e., a `Middle`.)

Grading: 2 points for each correct part. For the parts that contain an error, the error must be explained; give 0 points for that part otherwise.

Question 3 (16 points): Relationships between Classes
The four relationships are the *IS* relationship (also called *ISA* or IS-A), the *HAS* relationship, the *OWNS* relationship, and the *USES* relationship.

We speak of an *IS* relationship when one class is derived from another class, or when a class implements an interface class. The derived class can be used anywhere where the base class is expected (the substition principle).

We speak of a *HAS* relationship when one class is contained in the other (as a class-type member). This is also called *aggregation*.

We speak of an *OWNS* relationship when (an object of) one class is on the heap, and (an object of) the other class is responsible for it (i.e., it should delete the object on the heap when it is no longer needed, or transfer ownership to another object). Each object on the heap should be owned by precisely one object, but an object may own more than one object.

We speak of a *USES* relationship when there is no relationship of the previous three kinds, but when (an object of) one class calls member funcions of (an object of) the other class.

Grading: 4 points for each relationship, but only 1 point if the explanation is missing.

Question 4 (12 points): Virtual Destructors
A derived class in a class hierarchy may allocate memory on the heap, and must release that memory when it is deleted itself. This is usually done in the destructor of the derived class. To make sure that the destructor of the derived class is called when we use polymorphism (i.e., we have a pointer to the base class that actually points to the derived class), the destructor of the base class must be virtual.

The following example ilustrates the problem with a non-virtual destructor:

```
class B {
  int* ip;
public:
  B() { ip = new int[100]; }
  ~B() { delete[] ip; }
};

class D : public B {
  double* dp;
public:
  D() { dp = new double[100]; }
  ~D() { delete[] dp; }
};

// ...
B* bp = new D; // allocates int[100] and double[100]
// ...
delete bp;     // releases only the int[100]; ~B() is not virtual
```

Grading: 6 points for a correct explanation, 6 points for a correct example (there must be a call to `new` in the derived class, and there must be a pointer to the base class that actually points to an object of the derived class.)

Question 5 (21 points): Algorithms and Function Objects

A sample solution is the following:

```
class PrintIfAgeInbetween {
  int _min;
  int _max;
public:
  PrintIfAgeInbetween(int min, int max)
    : _min(min), _max(max) { }
  void operator()(const Person& person) const {
    if(person.age() >= _min && person.age() <= _max)
      cout << person << '\n';
  }
};
```

Grading: the class `PrintIfAgeInbetween` should have

Two private data members of type `int` to keep track of the minimum and maximum age to be printed	3 points
A constructor that has two arguments of type int and initializes the two private data members	3 points
a member function `operator()`	3 points
`operator()` has return type `void`	3 points
`operator()` has an argument of type `Person` or `const Person&` (substract 2 points if the argument is a non-const `Person&`)	3 points
`operator()` is a const member function	3 points
the body of `operator()` compares the age of `Person` with the private data members, and prints the Person if his/her age is in between the values of these data members. (Substract 2 points if the name and age of a `Person` are printed individually, instead of re-using the overloaded `operator<<`, and substract 1 point if the `'\n'` (or `endl`) is missing.)	3 points

Question 6 (21 points): Templates
A sample solution:

```
template<class Key, class Value>
void print_map(map<Key, Value> m){
  typedef map<Key, Value>::const_iterator CI;
  for(CI p = m.begin(); p!= m.end(); ++p)
    cout << p->first << ": " << p->second << '\n';
}
```

Alternatively:

```
template<class M>
void print_map(M m){
  typedef typename M::const_iterator CI;
  for(CI p = m.begin(); p!= m.end(); ++p)
    cout << p->first << ": " << p->second << '\n';
}
```

(The first solution only works for `map`s, while the second solution also works for other containers that contain `pair`s, such as `list<pair<int, string> >`. In the second solution, we need the keyword `typename`; we didn't discuss this in class, see the textbook for an explanation.)

The iterator should be a `const_iterator`, since we don't modify the contents of the `map` (an `iterator` will work too, but is bad programming style.) However, I have not been explicit about this in the lecture notes, and in the sample code that I gave I haven't been very consistent in this respect myself, so no points will be deducted for using an `iterator` instead of a `const_iterator`.

Grading:

Correct template arguments (`template<class Key, class Value>`), and correct use of the template arguments in the function declaration (`void print_map(map<Key, Value> m)`). Deduct 3 points if the return type of `print_map` is incorrect	7 points
correct type of the iterator (either a `const_iterator` or an `iterator`). This may be in a `typedef`, as in the sample code, or as a "normal declaration" before the for-loop, or as a declaration in the initializer of the for-loop.	7 points
Correct for-loop (or programming structure that does the same job). Deduct 2 points if the colon and/or space is forgotten, and 2 points if the newline is forgotten.	7 points