The Hong Kong University of Science & Technology

COMP 151: Object-Oriented Programming

Fall Semester, 1999
FINAL EXAMINATION
December 15, 1999

Student name: ____________________
Student id: ____________________
Department: ____________________
Lab section: ____________________

Instructions:

1. There are 6 problems on 10 pages. Check that you have all 10 pages.
2. Write your name, student id, department, and lab section number on this page.
3. Write your student id on each of the following pages.
4. Answer all questions in the space provided. Rough work should be done only on the back pages, and crossed out with a diagonal line afterwards to show that it is not part of your answer.
5. Leave all pages stapled together.
6. The examination will last for **180 minutes**.
7. Stop writing immediately when the time is up, and remain seated in silence until all work has been collected.
8. You are not allowed to leave during the first half hour and the last half hour of the exam.

For grading purposes only

Problem 1	______	/	18
Problem 2	______	/	12
Problem 3	______	/	16
Problem 4	______	/	12
Problem 5	______	/	21
Problem 6	______	/	21
TOTAL:	______	/	**100**

1. (18 points) **Polymorphism**
C++ supports three different kinds of polymorphism. Describe those three kinds of polymorphism, and give small examples in C++ code to illustrate them. Your examples need not be complete working programs; you only have to give the essential parts that illustrate your point.

Student id:

2. (12 points) **Class Hierarchies**
The program below contains a class hierarchy of three classes: `Top` (the base class), `Middle` (derived from `Top`, and itself a base class for `Bottom`), and `Bottom` (derived from `Middle`). The class definitions are legal C++ and compile without any errors. They do, however, contain a typical example of very bad programming style, which leads to surprising polymorphic behaviour.

The code continues on the next page. The `main` function consists of six parts and a return statement. Each part is either completely correct C++, or contains one or more errors.

For the parts that are correct, write the output to the right of the code in the `main` function.

For the parts that contain one or more errors, briefly explain the errors to the right of the code in the `main` function. You don't have to give any output for the parts that contain errors.

```
#include <iostream>

using std::cout;

class Top {
public:
  virtual ~Top() { }
  virtual void f() const = 0;
  virtual void g() const { cout << "Top::g()\n"; }
  void h() const { cout << "Top::h()\n"; }
};



class Middle : public Top {
public:
  virtual ~Middle() { }
  virtual void f() const { cout << "Middle::f()\n"; }
  void g() const { cout << "Middle::g()\n"; }
  virtual void h() const { cout << "Middle::h()\n"; }
};



class Bottom : public Middle {
public:
  virtual ~Bottom() { }
  void f() const { cout << "Bottom::f()\n"; }
  virtual void g() const { cout << "Bottom::g()\n"; }
  virtual void h() const { cout << "Bottom::h()\n"; }
};
```

(Problem 2, continued.)

```
int main(){
  // part 1
  Top top;
  top.g();



  // part 2
  Bottom bottom;
  bottom.f();



  // part 3
  Middle middle = bottom;
  middle.f();



  // part 4
  Top* tp = new Bottom;
  tp->f();
  tp->g();
  tp->h();



  // part 5
  Middle* mp = new Bottom;
  mp->f();
  mp->g();
  mp->h();



  // part 6
  Bottom* bp = new Middle
  bp->f();
  bp->g();
  bp->h();



  return 0;
}
```

3. (16 points) **Relationships between Classes**
   Name the four kinds of relationships between classes that we have discussed in the course, and give an explanation of each of the four relationships.

4. (12 points) **Virtual Destructors**
Explain why it is important to make the destructor of a base class in a class hierarchy virtual. Give C++ code that illustrates the problem with a non-virtual destructor. Your example code needs not be a complete working program; you only have to give the essential parts that illustrate your point.

5. (21 points) **Algorithms and Function Objects**
The program below defines a very simple Person class; a Person has a _name and an _age, which are initialized by the constructor, and which can be retrieved by the inspector functions name and age. Furthermore, the ouput operator is overloaded for the Person class. The Person class has a static member _namelen to facilitate nicely formatted output.

In the main function (on the next page), we declare a vector of Persons, and fill it with six objects of type Person. Next, we want to print all persons in the vector people who's age is in between 15 and 35 years (inclusive). We decide to use the for_each function from the STL.

Complete the definition of class PrintIfAgeInbetween on the next page, so that the program works as desired and gives the output printed on the bottom of the next page.

```
#include <iostream>
#include <iomanip>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

class Person {
  string _name;
  int _age;
  static const int _namelen = 30;
public:
  Person(string name, int age)
    : _name(name), _age(age) { }
  string name() const { return _name; }
  int age() const { return _age; }
  friend ostream& operator<<(ostream&, const Person&);
};

// Definition of the static data member of the Person class
const int Person::_namelen;

// output operator for printing a Person
ostream& operator<<(ostream& os, const Person& person){
  os << person.name();
  for(int i=person.name().size();i<person._namelen;i++)
    os << ' ';
  os << setw(5) << person.age();
  return os;
}
```

(Problem 5, continued.)

```
// Complete the definition of class PrintIfAgeInbetween
class PrintIfAgeInbetween {











}; // end of definition of class PrintIfAgeInbetween

int main(){
  vector<Person> people;

  people.push_back(Person("John", 21));
  people.push_back(Person("Chantal", 31));
  people.push_back(Person("July", 55));
  people.push_back(Person("Hee-Kap", 12));
  people.push_back(Person("Frank", 16));
  people.push_back(Person("Betty", 3));

  for_each(people.begin(), people.end(), PrintIfAgeInbetween(15,35));

  return 0;
}
```

The output of the above program:

```
John                            21
Chantal                         31
Frank                           16
```

6. (21 points) **Templates**
A `map` is an associative container that holds (*key, value*) pairs, and allows efficient lookup based on the *key*.

In the `main` function in the code on the next page, we declare two different maps:
`map<string, int> phonebook`, and
`map<int, string> numbers`.
In each map, we store four pairs of the appropriate types.

We would like to print both maps as follows: each pair in the map is printed on a separate line; the *key* is printed first, followed by a colon, a space, and finally the *value*. Sample output is given on the bottom of the next page.

Complete the code on the next page by defining a function template `print_map`. Your code should work properly for all maps for which the *keys* and the *values* can be printed using the output operator `<<`.

Advice: the whole function template definition needs less than 10 lines of code.

(Problem 6, continued.)

```
#include <iostream>
#include <string>
#include <map>

// Write the definition of the required function template in the space below.
```

```
int main(){
  map<string, int> phonebook;
  phonebook["Otfried"] = 23588770;
  phonebook["Rene"] = 23587011;
  phonebook["Dept. Office"] = 23587000;
  phonebook["Dept. Fax"] = 23581477;

  print_map(phonebook);
  cout << '\n';

  map<int, string> numbers;
  numbers[1] = "yat";
  numbers[23] = "yi-sap sam";
  numbers[9] = "gau";
  numbers[46] = "sei-sap lohk";

  print_map(numbers);
  return 0;
}
```

The output of the above program:

```
Dept. Fax: 23581477
Dept. Office: 23587000
Otfried: 23588770
Rene: 23587011

1: yat
9: gau
23: yi-sap sam
46: sei-sap lohk
```