Programming Design, Spring 2013
Final Exam

Instructor: Ling-Chieh Kung
Department of Information Management
National Taiwan University

Name: _____ Student ID: _____

Note 1. In total there are 110 points for this exam. If you get more than 100 points, your official score for this exam will only be 100.

Note 2. You do not need to return these problem sheets. Write down all your answers on the answer sheets provided to you.

0. (5 points) In your opinion, what is the most difficult part of this course? If you are the instructor of this course, what will you do to make students learn more?

   Note. As you may expect, you will get 5 points as long as you write down anything reasonable. Therefore, work on this problem seriously only if you have time.

1. (25 points; 5 points each) Answer the following questions.

   (a) (1 point each) Which of the following are required to implement polymorphism?
      i. Pointers.
      ii. Recursion.
      iii. A copy constructor.
      iv. Inheritance.
      v. Virtual functions.

   (b) What are the main ideas of encapsulation? Briefly explain those ideas in your answer.

   (c) (1 point each) Which of the following are included in a function signature?
      i. The return type.
      ii. The function name.
      iii. The parameter types.
      iv. The number of parameters.
      v. The parameter names.

   (d) What are the differences between function overloading and function overriding?

   Note. One way to illustrate their differences is to clearly explain both of them.

   (e) What is a memory leak? When do we see one? In a class with dynamic memory allocation, how may we avoid it?

2. (45 points) In a university, a course may be offered during the weekdays (Monday, Tuesday, ..., and Friday). As each weekday is divided into nine hours (1, 2, ..., and 9), a course occupies some consecutive hours (e.g., your computer programming course occupies hours 6, 7, and 8 on Monday). To describe these course, the following class `Course` is (partially) defined. Please note that as the number of instructors can vary, we need to use dynamic memory allocation to store these names.

```
class Course
{
private:
    string title;       // the course title
    string dept;        // the department offering this course
    int unit;           // the number of units
    int instCount;      // the number of instructors
    string* instructor; // the names of all the instructors
    int day;            // the day of this course (Monday, Tuesday, etc.)
    int startTime;      // the starting time of this course (1, 2, 3, ..., 9)
public:
    // some functions for you to implement
};
```

(a) (5 points) Implement a constructor that has three arguments as the initial values of the course title, department, and units. For those attributes with no initial value, initialize them by yourself.

(b) (5 points) Implement an instance function `bool setTime(int day, int startTime)` that validates `day` and `startTime` before the values of these arguments are assigned into the corresponding member variables. In particular, `day` can only be an integer between 1 and 5 and `startTime` can only be an integer between 1 and 9. Moreover, `startTime` plus `unit` cannot be greater than 10. If any argument fails the validation, return `false` and do not store these two values; otherwise, return `true` and store these two values.

(c) (5 points) Consider the following member function

```
void Course::print()
{
    cout << title << ", " << dept << ", " << unit << " unit(s)" << endl;
    cout << instCount << " instructor(s):";
    for(int i = 0; i < instCount; i++)
        cout << instructor[i] << " ";
    cout << endl;
    // print out date and time
}
```

which prints out the basic information of this course. Write codes to replace the comment to print out the lecture day and hours of this course. The output format should be "d: i-j", where d is `Monday` if `day` is 1, `Tuesday` if `day` is 2, ..., and `Friday` if `day` is 5, i is the starting hour, and j is the ending hour. For example, if for a course `day` is 4, `startTime` is 6, and `unit` is 3, the output should be `Thursday: 6-8`.

(d) (10 points) Implement an instance function `void enterInstructor()` in which the user can first enter (`cin`) the number of instructors of this course and then enter those instructors' names. These names should then be stored in a dynamic array that can be accessed through the instance variable `instructor`. Note that this function may be called when there are already some instructors' names recorded. In this case, all the existing names should be abandon and those ones entered in this invocation should be recorded.

(e) (10 points) Implement a copy constructor for this class with deep copy.

(f) (5 points) In the copy constructor, why the argument must be passed with call by reference instead of call by value? Why the argument is set to be a constant variable?

(g) (5 points) Implement the destructor.

3. (15 points) Continue from Problem 2. Suppose now we want to implement a class `LECourse` to describe liberal education courses. Beside all the attributes of `Course`, a liberal education course has one more attribute, its category. For simplicity, we assume a liberal education course can belong to only one course, which is recorded as an integer between 1 and 8. All the operations that can be done on a `Course` must can also be done on a `LECourse`.

   (a) (5 points) Write down the definition of this class `LECourse` by inheriting from `Course`. Include a member variable `int category` for category and a constructor with four parameters for title, department, unit, and category. Modify the definition of `Course` in Problem 3 if you need.

   (b) (5 points) Implement the four-parameter constructor.

   (c) (5 points) For an `LECourse` object, we want the function `void print()` can also print out the attribute `category`. To do this, you need to override this function. Point out what do you need to modify in the class `Course` and what do you need to add into the class `LECourse`.

4. (20 points) Continue from Problems 2 and 3. Suppose now we want to implement a class `Schedule` to describe a student's schedule. To do so, first a structure `CNode` is defined for course nodes:

```
class CNode
{
private:
    Course c;
    CNode* next;
public:
    CNode(Course c) { this->c = c; next = NULL; }
};
```

   The class `Schedule` is then partially defined as

```
class Schedule
{
// friend declaration
private:
    int courseCount;        // number of courses enrolled
    CNode* head;            // head of the linked list of courses enrolled
    string timeTable[5][9]; // time table for this student
public:
    void insert(CNode cn, int index);
    CNode remove(int index);
    // some functions for you to implement
};
```

   You may assume that the functions `insert` and `remove` have been implemented correctly.

   (a) (5 points) Write down friend declaration statements to declare `Course`, `LECourse`, and `CNode` as friends of `Schedule`.

   (b) (5 points) Implement a constructor for `Schedule` with no parameter. Initialize `courseCount` to 0, `head` to what it should be, and all elements in `timeTable` to "idle".

   (c) (10 points) Implement a function `void setTimeTable()` to set up the time table according to the course linked list. For each course contained in the linked list, the corresponding cells in `timeTable` that are the lecture hours of this course should be marked with the course title. For example, if the course "programming" is offered at the sixth, seventh, and eighth hours on Monday, we should assign "programming" to `timeTable[0][5]`, `timeTable[0][6]`, and `timeTable[0][7]`. You may assume that there is no overlapping of lecture hours among different courses.