Programming Design, Spring 2013
Suggested Solution for Final Exam

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

1. (a) i, iv, and v.

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

   The main ideas of encapsulation include packaging and data hiding. Packaging means to group several variables and functions into a single unit (a structure or a class). Data hiding provides the possibility for a programmer to set different visibility levels to different unit members. This allows the programmer to control the ways of accessing the hidden data.

   (c) ii, iii, and iv.

   (d) Function overloading allows a programmer to implement multiple functions with the same function name. The invocation of functions will be determined based on the parameters of these function. As long as two functions have different parameter lists (either the number of parameters or parameter types are different), they can be differentiated as two different functions. It is applied on global functions or member functions in a single class.

   Function overriding allows a programmer to implement a member function in a child class that has the same function signature as one in its ancestors. The function implemented in the child class will override its ancestors' and become the default one to be invoked. It is applied with inheritance.

   (e) When we dynamically allocate a memory space, it has no name and can be accessed only through a pointer pointing to it. If we do not release this space before we modify the value stored in the pointer or before the pointer diminishes, such a space can never be accessed and will be wasted. This situation is called a memory leak. In a class, we should make sure that all these dynamically allocated spaces are released in the destructor.

2. (a)
```
Course::Course(string title, string dept, int unit)
{
    this->title = title;
    this->dept = dept;
    this->unit = unit;
    this->instCount = 0;
    this->instructor = NULL;
    this->day = 0;
    this->startTime = 0;
}
```

   (b)
```
bool Course::setTime(int day, int startTime)
{
    if(day < 1 || day > 5)
        return false;
    else if(startTime < 1 || startTime + this->unit > 10)
        return false;
    else
    {
        this->day = day;
        this->startTime = startTime;
        return true;
    }
}
```

   (c)
```
string weekday[5] = {"Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};
cout << weekday[this->day - 1] << ": " << this->startTime << "-"
     << this->startTime + this->unit - 1 << endl;
```

(d)
```
void Course::enterInstructor()
{
    delete [] this->instructor;
    cin >> this->instCount;
    this->instructor = new string[this->instCount];
    for(int i = 0; i < this->instCount; i++)
        cin >> this->instructor[i];
}
```

(e)
```
Course::Course(const Course& c)
{
    this->title = c.title;
    this->dept = c.dept;
    this->unit = c.unit;
    this->instCount = c.instCount;
    this->instructor = new string[this->instCount];
    for(int i = 0; i < this->instCount; i++)
        this->instructor[i] = c.instructor[i];
    this->day = c.day;
    this->startTime = c.day;
}
```

(f) Suppose the argument is passed using call by value, like `Course::Course(Course c)`, then when one invokes the copy constructor, the call-by-value argument passing invokes the copy constructor again! In other words, before any instruction is executed, the copy constructor will be invoked again, again, and again. The program will stock here forever and never terminate. This is why call by reference should be used. As the argument is passed with call by reference, it may be modified, which should not happen in a copy constructor. Therefore, we set it to be a constant variable to protect it from being modified.

(g)
```
Course::~Course()
{
    delete [] this->instructor;
}
```

3. (a)
```
class LECourse : public Course
{
private:
    int category;
public:
    LECourse(string title, string dept, int unit, int category);
};
```
We also need to replace the `private` modifier in `Course` to `protected`.

(b)
```
LECourse::LECourse(string title, string dept, int unit, int category)
    : Course(title, dept, unit)
{
    this->category = category;
};
```

(c) First, we need to modify the parent class `Course` and set `void print()` as a virtual function. To do this, we should add `virtual` in front of `void print();`. Then we should add a new implementation of `void print()` into the child class `LECourse`. In the new implementation, we should print out `category` in a way we like.

4. (a)
```
friend class Course;
friend class LECourse;
friend class CNode;
```
Note. As you may have found, declaring `Course`, `LECourse`, and `CNode` as friends of `Schedule` is not meaningful. Instead, we should declare `Schedule` as a friend of the first three classes. Nevertheless, this question is just to test your knowledge regarding how to declare friends.

(b)
```
Schedule::Schedule()
{
    this->courseCount = 0;
    this->head = NULL;
    for(int i = 0; i < 5; i++)
    {
        for(int j = 0; j < 9; j++)
            this->timeTable[i][j] = "idle";
    }
}
```

(c)
```
void Schedule::setTimeTable()
{
    CNode* temp = this->head;
    for(int i = 0; i < this->courseCount; i++)
    {
        // the following four instructions require
        // Schedule to be a friend of Course
        string title = temp->Course.title;
        int day = temp->Course.day;
        int startTime = temp->Course.startTime;
        int unit = temp->Course.unit;

        for(int j = 0; j < unit; j++)
            this->timeTable[day - 1][startTime + j - 1] = title;

        temp = temp->next;
    }
}
```

Note. In the exam held on June 17, 2013, the member variable `next` of class `CNode` was set to be a `Course*`. However, it should be `CNode*`. Therefore, these 10 points are given to all students for free.