Middle East Technical University   Department of Computer Engineering

CENG 213

Data Structures

Fall 2012-2013

Homework 1 - Course Registration

Due date: 1 November 2012, Thursday, 23:55

1 Objective

This homework aims to help you get familiar with object oriented programming using C++. You will implement a program which simulates the backend layer of a department course registration system.

Keywords: *Object Oriented Programming, C++*

2 Problem Definition

You will be given the exact definitions of the classes that you'll implement. You are not allowed to change neither the header file nor the given class definitions. You will submit only implementations of the given classes. You are asked to implement and submit Department, Student, Schedule, Offering and Course classes only. Instructor class implementation is provided.

- Every Department has a unique name, Instructor list, Student list, offered Course list, and curriculum information. Curriculum is kept as a *sorted* vector of Courses such that pre-requisite of a Course appears before that Course in the curriculum. Each Course appears only once in the curriculum. Instructor and Student lists are *sorted* as well, according to ID information.
- Every Instructor has a name, unique ID, busy hours array, schedule and Course List which are taught.
- Every Student has a name, unique ID, busy hours array, schedule, semester information and Course List which are to be taken in the current semester and another Course list which keeps courses that have been taken by now.
- Every Course has a unique course code and a pre-requisite Course code list, which are composed of integers and kept in a vector.
- Every Offering keeps a pointer to the corresponding Instructor, and a pointer to the corresponding Course object. Every Offering has section, day, hour, and duration information. Day information is kept in terms of integers (starting from zero). Duration is in terms of lessons. Hour information indicates the starting hour of the Offering. *Note that during the creation of an Offering object, busy hours and offered courses list of the corresponding Instructor are updated as well.*

- Every OfferingPair has a pointer to the Offering it keeps and another pointer to the next OfferingPair.
- Every Schedule has an array of singly-linked lists of OfferingPair struct. Note that there are exactly 5 pointers, each pointing to the daily OfferingPair lists. These lists are sorted in terms of hour information.

3 Specifications

3.1 Department

Using a Department object, one will be able to:

- Add a new Instructor to the Department. Check the ID of the new Instructor. If already added, throw `ERR_INSTRUCTOR_ALREADY_EXISTS` error.
- Remove an Instructor that leaves the Department.
- Add a new Student to the Department. Check the ID of the new Student. If already added, throw `ERR_STUDENT_ALREADY_EXISTS` error.
- Remove a Student from the Department.
- Add a new Course. Check code of the new course. If already added, throw `ERR_COURSE_ALREADY_EXISTS` error.
- Remove a Course from the Department.
- Add a new Offering. If already added, throw `ERR_OFFERING_ALREADY_EXISTS` error.
- Cancel an Offering (remove from the Offering list of the Department).
- List all Students in the Department.
- List all Instructors in the Department.
- List all Courses in the curriculum of the Department.
- List all Offerings offered by the Department.

3.2 Instructor

Using an Instructor object, one will be able to:

- Obtain *id* information.
- Obtain *name* information.
- Obtain *busy hours* array.
- Obtain *Offerings* taught in the current semester.
- Offer a new Course to be taught in the current semester.
  - Check busy hours array and add if there is no conflict. If there is a conflict, throw `ERR_CONFLICT` error.
  - Check if course is already added. If so, throw `ERR_COURSE_ALREADY_EXISTS` error.

  - Change *busy hours* array and schedule accordingly.
    - Indicate busy hours with 1 and other hours in the week with 0.
    - An Instructor can attend to courses five days in a week and up to 12 hours in a day, i.e. starting from 8.40 to 20.30, including lunch break.

- Remove a Course that is not offered in the current semester. Change *busy hours* array and schedule accordingly.

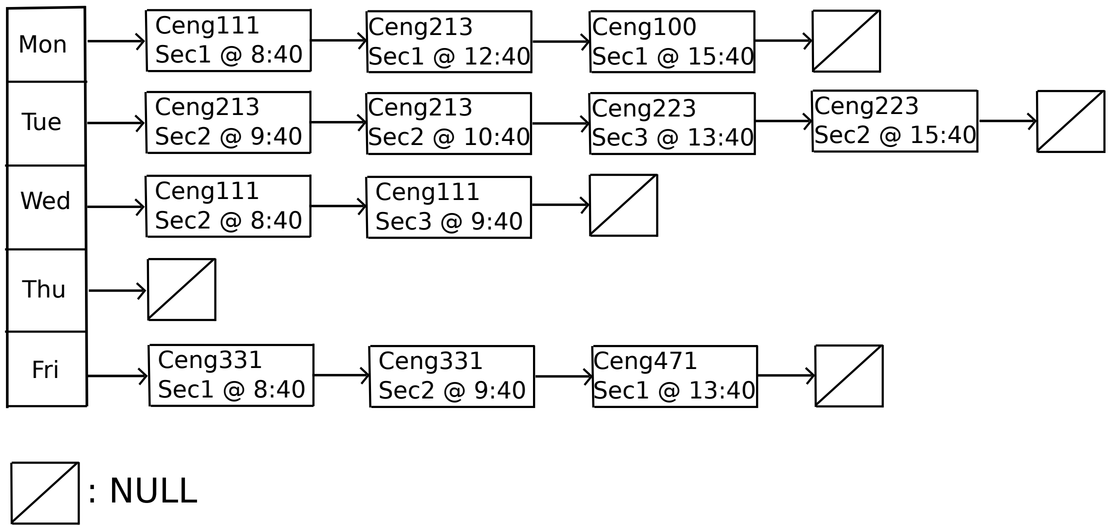

Figure 1: Course schedule data structure.

3.3 Student

Using a Student object, one will be able to:

- Obtain *id* information.
- Obtain *name* information.
- Obtain *busy hours* array.
- Obtain *previously taken Course list*.
- Obtain *Schedule* information.
- Add a Course to be taken in the current semester.
  - Check *busy hours* array and add if there is no conflict. If there is a conflict, throw `ERR_CONFLICT` error.
  - Change *busy hours* array and schedule accordingly.
  - Check if course is already added. If so, throw `ERR_COURSE_ALREADY_ADDED` error.
  - Check if pre-requisites of the course are previously taken. If not, throw `ERR_PREREQISITE_IS_NOT_TAKEN_YET` error.
  - Change *busy hours* array and schedule accordingly.
    - Indicate busy hours with 1 and other hours in the week with 0.

        * An Instructor can attend to courses five days in a week and up to 12 hours in a day, i.e. starting from 8.40 to 20.30, including lunch break.

- Remove a Course that is to be dropped. Change busy hours array and schedule accordingly.
- Change section of a Course the student is taking if there is other sections for the offered Course.
    - Change is allowed if there is no conflict.
    - Return whether changing section operation is successful or not.

3.4 Course

Using a Course object, one will be able to:

- Obtain course code.
- Obtain list of pre-requisite course codes.
- Add a pre-requisite course.

3.5 Offering

Using a Offering object, one will be able to:

- Obtain a pointer to the corresponding Course object
- Obtain a pointer to the corresponding Instructor object that teaches the class
- Obtain section information
- Obtain day information (in terms of integers)
- Obtain at which hour the class starts
- Obtain the duration (how long in terms of lesson count) the class lasts
- Set a pointer to the corresponding Instructor object that teaches the class
- Set day information
- Set at which hour the class starts
- Set the duration the class lasts

3.6 Schedule

Using a Schedule object, one will be able to:

- Obtain weekly schedule of a Student. This is kept as an array of singly-linked lists of OfferingPair struct. OfferingPair has a pointer to the Offering it keeps and another pointer to the next OfferingPair. Note that there are exactly 5 pointers, each pointing to the daily OfferingPair lists. These lists are sorted in terms of hour information. See Figure 1.
- Obtain the pointer of an Offering object in the schedule of the Student given the course code.
- Add an Offering to the schedule.
- Remove and Offering from the schedule.

3.7 Exceptions

Wherever necessary, classes are supposed to throw the following exceptions, defined as an enumeration `SystemError`:

- `ERR_STUDENT_ALREADY_EXISTS`
- `ERR_INSTRUCTOR_ALREADY_EXISTS`
- `ERR_COURSE_ALREADY_EXISTS`
- `ERR_OFFERING_ALREADY_EXISTS`
- `ERR_OFFERING_ALREADY_ADDED`
- `ERR_OFFERING_IS_NOT_TAKEN`
- `ERR_OFFERING_DOES_NOT_EXISTS`
- `ERR_PREREQISITE_ALREADY_EXISTS`
- `ERR_PREREQISITE_IS_NOT_TAKEN_YET`
- `ERR_CONFLICT`

3.8 Overloaded Operators

One will be able to output Department, Instructor, Student, Course, Offering, and Schedule objects using overloaded operator `<<`.

3.9 Destructors

Destructors are expected for Instructor and Student classes. Note that Instructor class implementation is provided.

4 I/O Specifications

Each attribute of the given class should be output with a newline between them.

4.1 Course

After printing the course code, there should be a colon (:) and codes of pre-requisite courses (if there are any) should be printed with a dash (-) in between. Output of a Course object will be in the following format:

```
<code>:<pre-requisite_1_code>-<pre-requisite_2_code>-  ...  -<pre-requisite_n_code>
```

Here is a sample output:

```
213:140-1099-1070
```

4.2 Offering

After printing the course code, there should be a dash (-) and then section should be printed and after another dash, initials of the Instructor who teaches this class should be printed. Output of a Offering object will be in the following format:

```
<course_code>-<section>-<initials_of_instructor>
```

Here is a sample output:

```
213-2-ST
```

4.3 Schedule

After printing the day information in terms of corresponding string (Monday, Tuesday etc.), there should be a colon (:) and then, for each offering taken that day, in chronological order, starting and ending hours of each offering should be printed with a dash (-) in between, then after a colon (:), corresponding Offering pointer should be called to be outstreamed as described in section 4.2. There should be a tab character between offerings. Each day's schedule should be printed in a separate line. Output of a Schedule object will be in the following format:

```
<string_day_1>:<start_hour_of_Offering_1>-<end_hour_of_Offering_1>:<Offering_1><tab>
<start_hour_of_Offering_2>-< end_hour_of_Offering_2>:<Offering_2><tab>... <tab>
<start_hour_of_Offering_n>-< end_hour_of_Offering_n>:<Offering_n><new_line>
<string_day_2>:<start_hour_of_Offering_1>-<end_hour_of_Offering_1>:<Offering_1><tab>
<start_hour_of_Offering_2>-< end_hour_of_Offering_2>:<Offering_2><tab>...<tab>
<start_hour_of_Offering_n>-< end_hour_of_Offering_n>:<Offering_n>
```

Here is a sample output:

```
Wed:3-4:200-1-AC
Thu:1-2:223-1-IK 3-4:140-2-AC 5-5:280-1-EG 6-8:260-1-EG
```

4.4 Student

After printing id, name, codes of previously taken courses, schedule of the student should be printed starting from the next line, i.e. there should be a newline character just before printing the schedule. Check the following format:

```
"ID:"<ID>"NAME:"<name>"PREV_COURSE_CODES:"<pre-requisite_1_code>-...-<pre-requisite_n_code>
<schedule_of_student>
```

Here is a sample output:

```
ID:100000 NAME:Veli PREV_COURSE_CODES:100-111
Wed:3-4:140-1-AC
Thu:1-2:223-1-IK 5-5:280-1-EG 6-8:260-1-EG
```

4.5 Instructor

After printing id and name, taught offerings are printed in the following format:

```
<"ID:"<ID>"NAME:"<name>"OFFERED_COURSES:"<offering_1><offering_2><offering_n>
```

Here is a sample output:

```
ID:2345678 NAME:EF GH OFFERED_COURSES:140-3-EG 280-1-EG 260-1-EG
```

4.6 Department

All lists are printed such that each element is printed to a separate line. The format below should be followed.

```
<name>
"Instructors:"
<instructor_1>
<instructor_2>
...
<instructor_n>
"All courses:"
<course_1>
< course _2>
...
< course _n>
"Offerings:"
<offering_1>
< offering _2>
...
< offering _n>
"Students:"
< student_1>
< student _2>
...
< student _n>
```

Here is a sample output:

```
CENG
Instructors:
ID:1234567   NAME:AB CD  OFFERED_COURSES:140-1-AC 140-2-AC
ID:2345678   NAME:EF GH  OFFERED_COURSES:140-3-EG 280-1-EG 260-1-EG
ID:3456789   NAME:IJ KL  OFFERED_COURSES:213-1-IK 223-1-IK 223-1-IK
All courses:
140:100-111
213:140-1099-1070
223:
280:
260:
Offerings:
140-1-AC
```

```
140-2-AC
140-3-EG
213-1-IK
223-1-IK
223-1-IK
280-1-EG
260-1-EG
Students:
ID:100000  NAME:Veli  PREV_COURSE_CODES:100-111
```

5 Header File

The accompanying header file named “hw1.h” is listed below:

```
//***************************************************************************

#ifndef HW1_H
#define HW1_H

//***************************************************************************

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

//***************************************************************************

#define DAYS_WORKING   5
#define HOURS_PER_DAY  12

//***************************************************************************

using namespace std ;

//***************************************************************************

enum SystemError
{
  ERR_STUDENT_ALREADY_EXISTS        = 1  ,
  ERR_INSTRUCTOR_ALREADY_EXISTS     = 2  ,
  ERR_COURSE_ALREADY_EXISTS         = 3  ,
  ERR_OFFERING_ALREADY_EXISTS       = 4  ,
  ERR_OFFERING_ALREADY_ADDED        = 5  ,
  ERR_OFFERING_IS_NOT_TAKEN        = 6  ,
  ERR_OFFERING_DOES_NOT_EXIST       = 7  ,
  ERR_PREREQISITE_IS_NOT_TAKEN_YET = 8  ,
  ERR_PREREQISITE_ALREADY_EXISTS    = 9  ,
  ERR_CONFLICT                      = 10

} ;

class Course     ;
class Instructor ;
class Offering   ;
class Schedule   ;
class Student    ;
class Department ;

struct OfferingPair
{
  Offering     * off     ;
  OfferingPair * nextOff ;

} ;



//************************************************************************

// Course

class Course
{
  public:

    Course ( int code ) ; // constructor

    // methods
    int         getCourseCode        () ;
    vector<int> getPreReqCourseCodes () ;

    void addPreReqCourseCodes ( int preCode ) ;

  private:

    int         code              ;
    vector<int> preReqCourseCodes ;

  friend ostream & operator<< ( ostream & out ,  Course & course ) ;
} ;

//************************************************************************

// Instructor

class Instructor
{
  public:

    Instructor ( int id , string name ) ;// constructor
   ~Instructor ()                       ;

    // methods
    string                getName           () ;
    int                   getID             () ;
    int                ** getBusyHours      () ;
    vector<Offering*>     getOfferedCourses () ;

    void offerNewCourse   ( Offering * newOffering       ) ;
    void cancelOffering   ( Offering * cancelledOffering ) ;
  private:

    int                  id             ;
    string               name           ;
    int               ** busyhours      ;
    vector<Offering*>    offeredCourses ;

  friend ostream & operator<< ( ostream & out , Instructor & instructor ) ;
} ;



//**********************************************************************

// Offering

class Offering
{
  public:

    Offering ( Course     * course  = NULL ,
               int          section = 1    ,
               Instructor * lec     = NULL ,
               int          day     = 0    ,
               int          hour    = 0    ,
               int          dur     = 1    ) ; // constructor

    Course     * getCourse     () ;
    int          getSection    () ;
    int          getDay        () ;
    int          getWhichHour  () ;
    int          getDuration   () ;
    Instructor * getInstructor () ;

    void setDay        ( int          d   ) ;
    void setWhichHour  ( int          wh  ) ;
    void setDuration   ( int          dur ) ;
    void setInstructor ( Instructor * ins ) ;

  private:

    Course     * course    ;
    int          section   ;
    Instructor * lecturer  ;
    int          day       ;
    int          whichHour ;
    int          duration  ;

  friend ostream & operator<< ( ostream & out , Offering & offering ) ;
} ;

//**********************************************************************

// Schedule

class Schedule
{
  public:

    Schedule () ;

    OfferingPair * getSchedule        ()                  ;
    Offering     * getCurrentOffering ( int courseCode ) ;

    void addOffering    ( Offering * o )    ;
    void removeOffering ( Offering * o )    ;

    private:

    OfferingPair * weeklySchedule ;

  friend ostream & operator<< ( ostream & out , Schedule & schedule ) ;
} ;

//*****************************************************************************

// Student

class Student
{
  public:

    Student ( int id , string name )  ;
   ~Student ()                        ;

    // methods
    string         getName        () ;
    int            getID          () ;
    vector<int>    getPrevCourses () ;
    int         ** getBusyHours   () ;
    Schedule    *  getSchedule    () ;

    void addPrevCourse ( int prevCourseCode )   ;
    int  addCourse     ( Offering * newOffering )   ;
    void dropCourse    ( Offering * newOffering )   ;
    bool changeSection ( int courseCode ,
              int newSection ,
              vector<Offering*> Offerings ) ;

  private:

    int            id               ;
    string         name             ;
    vector<int>    prevTakenCourses ;
    int         ** busyhours        ;
    Schedule       schedule         ;

  friend ostream & operator<< ( ostream & out , Student & student ) ;
} ;

//*****************************************************************************

// Department
```

```
class Department
{
  public:

    Department ( string name ) ;

    // methods
    string                getDeptName    ()                         ;
    vector<Instructor*>   getInstructors ()                         ;
    vector<Student*>      getStudents    ()                         ;
    vector<Offering*>     getOfferings   ()                         ;
    vector<Course*>       getCurriculum  ()                         ;
    Instructor          * getInstructor  ( int id )                 ;
    Student             * getStudent     ( int id )                 ;
    Course              * getCourse      ( int code )               ;
    Offering            * getOffering    ( int code , int section ) ;

    void addInstructor         ( Instructor * newIns         ) ;
    void removeInstructor      ( int          oldInsID       ) ;
    void addStudent            ( Student    * newStds        ) ;
    void removeStudent         ( int          gradutedStdID  ) ;
    void offerNewCourse        ( Offering   * no             ) ;
    void cancelOffering        ( Offering   * co             ) ;
    void addCourseToCurriculum ( Course     * curr           ) ;

  private:

    string              name           ;
    vector<Instructor*> instructors    ;
    vector<Student*>    students       ;
    vector<Offering*>   offeredCourses ;
    vector<Course*>     curriculum     ;

  friend ostream & operator<< ( ostream & out , Department & dept ) ;
} ;

//***************************************************************************

#endif

//***************************************************************************
```

6 Regulations

- **Programming Language:** You must code your program in C++. Your submission will be compiled with g++ on department lab machines. You are expected make sure your code compiles successfully with g++.
- **Late Submission:** You have a total of 7 days for late submission. You can spend this credit for any of the assignments or distribute it for all. If total of late submissions exceeds the limit, a penalty of $5 * day * day$ is applied.
- **Cheating:** In case of cheating, the university regulations will be applied.
- **Newsgroup:** You must follow the newsgroup (news.ceng.metu.edu.tr) for discussions and possible updates on a daily basis.

- **Grading:** This homework will be graded out of 100. It will make up 10% of your total grade.

- **Final Exam:** Please note that a student can take the final exam if and only if she/he (i) gets at least 30% points from each of the first two assignments and (ii) attends at least 1/2 of the quizzes / attendance checks. Otherwise; the student is not allowed to take the final exam and hence will get ”NA”.

7 Submission

Submission will be done via COW. Create a tar.gz file named `hw1.tar.gz` that contains all your source code files (Department.cpp, Student.cpp, Course.cpp, Offering.cpp, and Schedule.cpp). This archive should not contain a source file with a `main` function. You are not supposed to modify the header file, “hw1.h”. Assuming that “hw1.cpp” exists in the current directory, the following command sequence is expected to compile and run your program on department computers.

```
$ tar -xf hw1.tar.gz
$ g++ hw1.cpp Department.cpp Instructor.cpp Student.cpp Course.cpp Offering.cpp Schedule.cpp
  -o hw1
$ ./hw1
```