```cpp
/*
Name: Rifa Safeer Shah
Class: CECS 282
Instructor: Minhthong Nguyen
*/

#include <iostream>
#include "Instructor.h"
#include "Course.h"

using namespace std;

extern int const MAXCOURSE = 3;
extern int const CAPACITY;

/* Default Constructor */
Instructor::Instructor()
{
    this->name = ""; //name of the instructor
    this->numOfCoursesTaught = 0; //number of courses taught
    this->status = ""; //status of the instructor
    this->courses = new Course[MAXCOURSE]; //pointer pointing to the array of courses
}

/* Overload Constructor */
Instructor::Instructor(string name, string status, Course* crs, int num)
{
    this->name = name;
    this->status = status;
    this->courses = new Course[MAXCOURSE];
    this->numOfCoursesTaught = num; //number of courses taught
    for (int i = 0; i < numOfCoursesTaught; i++)
    {
        courses[i] = crs[i];
    }
} //ends Overload Constructor

/* Copy Constructor */
Instructor::Instructor(const Instructor& i)
{
    (*this).name = i.name; //name of the instructor
    (*this).status = i.status; //status of the instructor
    (*this).courses = i.courses; //name of the course
    (*this).numOfCoursesTaught = i.numOfCoursesTaught; //number of courses taught
    for (int x = 0; x < numOfCoursesTaught; x++)
    {
        courses[x] = i.courses[x];
```

```
48      }
49  } //ends Copy Constructor
50
51  /* Destrcutor */
52  Instructor :: ~Instructor()
53  {
54      delete[] courses;
55  }
56
57  /* Get Name */
58  string Instructor::getName() const
59  {
60      return name; //returns name of the instructor
61  } //ends Name
62
63  /* Get Course */
64  Course* Instructor::getCourse() const
65  {
66      return courses; //returns name of the course
67  } //ends Get Course
68
69  /* Get Number of Courses Taught */
70  int Instructor::getNumberOfCoursesTaught() const
71  {
72      return numOfCoursesTaught; //returns number of courses taught
73  } //ends Get Number of Courses Taught
74
75  /* Get Status */
76  string Instructor::getStatus() const
77  {
78      return status; //returns status of the instructor
79  } //ends Get Status
80
81  /* Get Student Status */
82  string Instructor::getStudentStatus(const Student& s, const Course& c) const
83  {
84      Student* stud = c.getStudent();
85      string Stat = "";
86      for (int i = 0; i < c.getNumberOfEnrollment(); i++)
87      {
88          if ((stud + i)->getId() == s.getId())
89          {
90              Stat = (stud + i)->getStatus();
91          }
92      }
93      delete stud;
94      return Stat;
95  }
96
```

```
 97  /* Add Student */
 98  int Instructor::addStudent(const Student& s, Course& c)
 99  {
100      int course = findCourse(c); //find the course
101      if (course == 0)
102      {
103          return -1; //returns -1 if course does not exist
104      }
105      else
106      {
107          int student = findStudent(s, c); //find the student in the course
108          if (student == 0)
109          {
110              int cap = c.getNumberOfEnrollment();
111              if (cap >= CAPACITY)
112              {
113                  Student* ptr = new Student[cap + 1];
114                  Student* temp = c.getStudent();
115                  for (int i = 0; i < cap; i++)
116                  {
117                      ptr[i] = temp[i];
118                  }
119                  ptr[cap] = s;
120                  ptr[cap].setStatus("Added");
121                  if (s.getTimeOfAction().compareTime(c.getTimeLastDateToEnroll())
                        == -1 || s.getTimeOfAction().compareTime
                       (c.getTimeLastDateToEnroll()) == 0)
122                  {
123                      ptr[cap].setStatus("Enrolled");
124                  }
125                  c.setNumberOfEnrollment(cap + 1);
126                  c.setRoster(ptr);
127                  for (int i = 0; i < numOfCoursesTaught; i++)
128                  {
129                      if (c.getCourseNumber() == courses[i].getCourseNumber())
130                      {
131                          courses[i] = c;
132                      }
133                  }
134                  return 1; //if student not already enrolled add the student into
                        the course
135              }
136              else
137              {
138                  Student* ptr = c.getStudent();
139                  ptr[cap] = s;
140                  ptr[cap].setStatus("Added");
141                  if (s.getTimeOfAction().compareTime(c.getTimeLastDateToEnroll())
                        == -1 || s.getTimeOfAction().compareTime
```

```
                    (c.getTimeLastDateToEnroll()) == 0)
142                 {
143                     ptr[cap].setStatus("Enrolled");
144                 }
145                 c.setNumberOfEnrollment(cap + 1);
146                 c.setRoster(ptr);
147                 for (int i = 0; i < numOfCoursesTaught; i++)
148                 {
149                     if (c.getCourseNumber() == courses[i].getCourseNumber())
150                     {
151                         courses[i] = c;
152                     }
153                 }
154                 return 1;
155             }
156         }
157         else
158         {
159         return -1; //returns -1 if the student is already in the course
160         }
161     }
162 } //ends Add Student
163
164 /* Drop Student */
165 int Instructor::dropStudent(const Student& s, Course& c, Time t)
166 {
167     int course = findCourse(c);
168     if (course == 0)
169     {
170         return 0; //returns 0 if the course does not exist
171     }
172     else
173     {
174         int n = findStudent(s, c);
175         if (n == 0)
176         {
177             return 0;
178         }
179         else
180         {
181             Student* ptr = c.getStudent();
182             for (int i = 0; i < c.getNumberOfEnrollment(); i++)
183             {
184                 if ((ptr + i)->getId() == s.getId())
185                 {
186                     (ptr + i)->setStatus("Dropped"); //set status to dropped
187                 }
188             }
189             for (int i = 0; i < numOfCoursesTaught; i++)
```

```
190             {
191                 if (c.getCourseNumber() == courses[i].getCourseNumber())
192                 {
193                     courses[i] = c;
194                 }
195             }
196             return 1;
197         }
198     }
199 } //ends Drop Student
200
201 /* Add Course */
202 int Instructor::addCourse(const Course& c)
203 {
204     int c_exists = findCourse(c);
205     if (c_exists == 1)
206     {
207         return -1;
208     }
209     else if (numOfCoursesTaught == MAXCOURSE)
210     {
211         Course* ptr = new Course[numOfCoursesTaught + 1];
212         Course* temp = courses;
213         for (int i = 0; i < numOfCoursesTaught; i++)
214         {
215             ptr[i] = temp[i];
216         }
217         ptr[numOfCoursesTaught] = c;
218         numOfCoursesTaught = getNumberOfCoursesTaught() + 1;
219         this->courses = ptr;
220         return 0;
221     }
222     else //if the course does not exist
223     {
224         courses[numOfCoursesTaught] = c; //add new course to the array of
              courses
225         numOfCoursesTaught = getNumberOfCoursesTaught() + 1;
226         return 1;
227     }
228 } //ends Add Course
229
230 /* Find Course */
231 int Instructor::findCourse(const Course& c)
232 {
233     for (int i = 0; i < numOfCoursesTaught; i++)
234     {
235         if ((courses + i)->getCourseNumber() == c.getCourseNumber())
236         {
237             return 1;
```

```
238         }
239     }
240     return 0;
241 } //ends Find Course
242 
243 /* Find Student */
244 int Instructor::findStudent(const Student& s, const Course& c)
245 {
246     Student* roster = c.getStudent();
247     for (int i = 0; i < c.getNumberOfEnrollment(); i++)
248     {
249         string id = (roster + i)->getId();
250         if (id == s.getId())
251         {
252             return 1;
253         }
254     }
255     return 0;
256 } //ends Find Student
```