Alexander Wise
November 28, 2023
EECE 2160
Homework 4

Problem 1:

First, the student class was created to take in the 3 private variables and implement the 3 public functions:

In the readStudentInfo() function, the *this* keyword is used to modify the private variables once information is sent for that specific student.

Next, the CourseSection class was created. This class had two private variables, 3 functions and a constructor and destructor:

The constructor took in the size of the class and used it to dynamically create an array with a size equal to the class size. Once created, this array was filled using the readAllInfo() function which took in the a students name, major, and gpa to create a new Student using the readStudentInfo() function from the student class.

For the printAllInfo() function, this iterated through each student on the roster and printed their information.

The printTopGPAStudent() function looped through the entire roster and found the student with the highest gpa and printed that student's info.

Finally, the destructor deleted the roster. When run, the following output was achieved:

```
-bash-4.2$ ./problem1
How many students in the course section? 3
Input the info of all students in the section:
Enter student name, major, and GPA separated by a space:
Steven ME 3.2
Enter student name, major, and GPA separated by a space:
Mike CivE 3.8
Enter student name, major, and GPA separated by a space:
Chuck EE 3.92
The following are the students in the section:
Name: Steven
Major: ME
GPA: 3.2

Name: Mike
Major: CivE
GPA: 3.8

Name: Chuck
Major: EE
GPA: 3.92

The info of the student with the highest GPA:
Name: Chuck
Major: EE
GPA: 3.92
```

Problem 2:

Question 1:

The question asks to create an operator overloader to return true if the first person's name alphabetically precedes the second person's name. This means that the ASCII value of the first character must be less than the ASCII value of the second person's first character. This was done with the following code:

```
        bool operator < (const NEPerson &p2) {
            return this->pname < p2.pname;
        }
```

The operator takes in the second person to compare to and checks to see if the name is alphabetically preceding the second person's name.

Question 2:

This question asks the same thing as question 1 but applies to the student's gpa. This was done with the following code:

```
        bool operator < (const NEStudent &s2){
            return this->gpa < s2.gpa;
        }
```

And will return true if the gpa is less than person 2's gpa.

Question 3:

The reason why the students are ordered by gpa and the teachers are ordered alphabetically is because the NEStudent class has an operator overloaded which takes precedence over the overloaded operator in the NEPerson class. This means that NEStudent will compare grades instead of names. Because NETeacher has no operator, it inherits the operator from the NEPerson class and uses that to print the info of the teachers.

Question 4:

To sort the students alphabetically, the following code was used:

```
    PrintOrdered(*ps1, *ps2);
```

Here, *ps1 and *ps2 are pointers that were initiated earlier in the main() function pointing to the NEStudent objects. . These pointers are of type NEPerson so they will use the overloaded operator created in the NEPerson class instead of the overloaded operator created in the NEStudent class:

```
  Students ordered alphabetically:
 Name: Emily      ID: 222         Address: Cambridge
          Student Major: CS       GPA = 3.6
          Taking Courses (enter only the course code with no spaces):
            c
            d
            e
 Name: John       ID: 111         Address: Boston
          Student Major: CE       GPA = 3.2
          Taking Courses (enter only the course code with no spaces):
            a
            b
```

Question 5:

The PrintReverseOrdered() template was created as follows:

```
template <class T>
    void PrintReverseOrdered(T &v1, T &v2) {
        if(v1 < v2) {
            v2.printInfo();
            v1.printInfo();
        } else {
            v1.printInfo();
            v2.printInfo();
        }
    }
```

This PrintReverseOrdered() function does the opposite of the PrintOrdered() function found in the other template. To print the reverse order, the v2.printInfo()was placed as the first statement instead of the second and vice versa for the else condition.

When run, the following output was achieved:

```
 Students ordered by GPA (Descending):
Name: Emily      ID: 222         Address: Cambridge
         Student Major: CS       GPA = 3.6
         Taking Courses (enter only the course code with no spaces):
           c
           d
           e
Name: John       ID: 111         Address: Boston
         Student Major: CE       GPA = 3.2
         Taking Courses (enter only the course code with no spaces):
           a
           b

 Teachers ordered alphabeticaly (Descending):
Name: Mary       ID: 888         Address: Dedham
         Teacher Rank: Assistant Professor       Department = EECE
         Teaching Courses:
           a
           b
           c
Name: Adam       ID: 999         Address: Foxboro
         Teacher Rank: Associate Professor       Department = CS
         Teaching Courses:
           d
           e
```

Question 6:

While NEStudent and NETeacher both have destructors, NEPerson does not so all NEPerson objects must be deleted. The implementation of this is as follows:

```
    delete ps1;
    delete ps2;
    delete pt1;
    delete pt2;
```

so, to free up memory, these NEPerson objects must be deleted.

Question 7:

If the *virtual* key word were to be deleted, the printInfo() function within NEPerson would be used whenever a pointer is used. For example, when the NEPerson objects were created for the students and teachers, the printInfo() function within NEPerson will be used to print the info instead of the virtual functions within the other classes.

Question 8:

For this question, three new functions were implemented across the three classes

First, a GetPersonInfo() function was implemented into the NEPerson class. This function took in inputs for the id, pname, and address.

Next, a GetStudentInfo() function was implemented into the NEStudent class. This function took in inputs for the student's major, gpa, and number of courses before calling the addEnrolledCourses() function within the NEStudent class.

Finally, a GetTeacherInfor() function was implemented into the NETeacher class. This function took in inputs for the rank, department, and num courses teaching before calling the addCoursesTeaching function().

When creating a new student, the GetStudentInfo() function would call the GetPersonInfo() function first to gather that information before prompting for the student's information.

This was also the same for the GetTeacherInfo() function. When run together, the following output was achieved:

```
 Input information for Student s3:
Enter ID: 002907473
Enter Name: Alexander
Enter Address: Seattle
Enter Major: CE
Enter GPA: 3.8
Enter number of current courses: 4
Enter the code of 4 course(s) for the student:
EECE2160
MATH2341
EECE2540
PHYS1155

 Input information for Teacher t3:
Enter ID: 01932
Enter Name: John
Enter Address: Portland
Enter Rank: Teacher
Enter Department: ECE
Enter Number of Courses: 2
Enter the code of the 2 course(s) the teacher is teaching (code with no spaces):
m
n

 Information for Student s3:
Name: Alexander  ID: 2907473     Address: Seattle
         Student Major: CE       GPA = 3.8
         Taking Courses (enter only the course code with no spaces):
           EECE2160
           MATH2341
           EECE2540
           PHYS1155

 Information for Teacher t3:
Name: John       ID: 1932        Address: Portland
         Teacher Rank: Teacher   Department = ECE
         Teaching Courses:
           m
           n
```

Question 9:

The NEGradStudent class inherited the functions of NEStudent and NETeacher. Next, the class itself had one private variable, one constructor (takes in the grad level of the student), destructor, and two functions.

The GetGradInfo() function calls the GetStudentInfo() and GetTeacherInfo() functions to create the new grad student.

Similarly, the printInfo() function called the printInfo() functions from NEStudent and NETeacher. When run, the following output was achieved:

```
 Input information for Grad Student g1:
Enter ID: 0987234
Enter Name: Jung
Enter Address: Seoul
Enter Major: Business
Enter GPA: 3.98
Enter number of current courses: 2
Enter the code of 2 course(s) for the student:
H33
N0987
Enter ID: 0987234
Enter Name: Jung
Enter Address: Seoul
Enter Rank: TA
Enter Department: Business
Enter Number of Courses: 1
Enter the code of the 1 course(s) the teacher is teaching (code with no spaces):
LK90

 Information for Grad Student g1Name: Jung       ID: 987234      Address: Seoul
         Student Major: Business         GPA = 3.98
         Taking Courses (enter only the course code with no spaces):
           H33
           N0987
Name: Jung       ID: 987234      Address: Seoul
         Teacher Rank: TA        Department = Business
         Teaching Courses:
           LK90
```

Here, the name, ID, and address are all printed twice because both printInfo functions call NEPerson's printInfo function resulting in the repeated output. This also applies to the input function. As seen in the image, the ID, Name, and Address are asked for twice.