Assignment 4: C++ Programming Issues

Question 1:

What do you mean by buffer overflow in C++? Explain the concept in your terms using an example **(15 points)**

Question 2: (65 points)

In this assignment you are to prevent any possibility of buffer overflows in the code. This may involve various other secure coding tasks, such as input validation, but you are not expected to make any changes other than those supporting the effort to prevent buffer overflows, despite the fact that the program does have other security flaws. You will document your changes and submit it in PDF format.

The program you are modifying is written in C++. Note that we want to have the core classes protected even if they're used with a different interface, so think in terms of defense at every level of the program.

Note: I will be grading this on the Linux machines, so it would be wise to work there. At least test your code there and submit from there.

The following is the specification for the program you are starting with.

Dr. Goode, a popular English professor has finally been convinced that he should be keeping track of grades on the computer instead of in his (rather illegible) old-fashioned grade book. Of course, Dr. Goode is not comfortable with these new-fangled machines, so he doesn't want to have to enter formulas or have a program that is flexible. He needs a program designed specifically for grading his courses.

All of Dr. Goode's courses are graded in the same general way.

- There are up to 8 papers, each graded out of 40 points. The papers are worth 30 percent of the grade.
- There are up to 45 quizzes, each graded out of 5 points. The quizzes are worth 40% of the grade.
- There are up to 4 exams, each graded out of 100 points. The exams are worth 30% of the grade.

The program has the following classes:

Student:

The Student class will maintain information about an individual student's grades. The class will have knowledge of Dr. Goode's specific grading schema, though the percentages, max points, and max number of items for each type of assignment should be stored in constants in the class, so that the program can be quickly and easily modified should Dr. Goode ever change his basic grading schema (this is unlikely, but has actually happened once before—about 10 years ago).

The Student class will need to define the following attributes.

- The last name of the student
- The first name of the student
- The student ID
- The number of absences a student has
- The number of exams a student has grades for
- An array of exam scores
- The number of quizzes a student has grades for
- An array of quiz scores
- The number of papers a student has grades for
- An array of paper scores

The Student class will need to define the following methods:

- A constructor that takes the Student's names and ID and sets absences and numbers of grades to zero.
- A means to initialize a Student object with data from a file (currently a constructor that accepts an already open istream reference – file format is below)
- Accessors:
    - *getLastName*
    - *getFirstName*
    - *getID*
    - *getNumAbsences*
    - *getQuizScore* – this method will need to take an integer parameter that indicates which quiz score to retrieve
    - *getExamScore* – this method will need to take an integer parameter that indicates which exam score to retrieve
    - *getPaperScore* – this method will need to take an integer parameter that indicates which paper score to retrieve
- Mutators
    - *updateQuizScore* – this method will need to take an integer parameter that indicates which quiz score to change
    - *updateExamScore* – this method will need to take an integer parameter that indicates which exam score to change
    - *updatePaperScore* – this method will need to take an integer parameter that indicates which paper score to change
- A *calcGrade* method that returns the Student's current grade in the course.
- A *createReport* method that returns a string containing all data about the student in a nice neat format. Note that this includes the Student's overall course grade.
- A *writeData* method that take an open ostream reference and write the student's data to it in the standard file format
- A *recordAbsence* method that takes no parameters and adds one to the student's absences.

- A *recordQuiz* method that takes the quiz score the student received and adds it to the student's quiz scores.
- A *recordPaper* method that takes the paper score the student received and adds it to the student's paper scores.
- A *recordExam* method that takes the exam score the student received and adds it to the student's exam scores.

File format for student data:

ID

Last Name

First Name

Number of Absences

Number of Quiz scores

Quiz score 1

Quiz score 2

…

Number of Paper scores

Paper score 1

Paper score 2

…

Number of Exam scores

Exam score 1

…

Last exam score

Next student's last name

Course:

The Course class will have the following attributes:

- A static constant representing the maximum number of students Goode might have (40).
- The name of the course

- The number of students in the course
- An array of Student objects
- The number of quizzes given in the course (so far)
- The number of exams given in the course (so far)
- The number of papers given in the course (so far)

The Course class will have at least the following methods:

- A constructor that takes only the course name and sets everything to represent an empty course.
- A means to initialize a Course object with data from a file (currently a constructor that accepts an already open istream reference – file format is below)
- Getters for the non-array variables
- An addStudent method that adds a student with no grades or absences to the course. Takes id, first name, and last name parameters.
- A recordExam method that accepts a file name. The file will contain a list of exam scores in the same order as the Students in the course file. You must add the appropriate exam score to each student in the course.
- recordQuiz and recordPaper methods similar to recordExam.
- A private method that accepts a student ID and returns the index of that student in the array of students.
- A recordAbsence method that accepts a studentID and records an absence for that student.
- A correctExamScore method that accepts a studentID, an exam score, and an index between 0 and numExams-1, and updates the appropriate exam score.
- correctQuizScore and correctPaperScore methods similar to correctExamScore.
- A getCourseGrade method that accepts a studentID and returns that student's current grade in the course.
- A createStudentReport method that accepts a studentID and returns a string containing that student's information in a well-formatted report.
- A createCourseReport method that returns a formatted string listing the names, ids, and current course grades of all students

File format for course data:

Course Name

NumQuizzes

NumPapers

NumExams

NumStudents

Student 1

Information in format

Specified above

Student 2

Information in format

Specified above

…

Last student's

Information in format

Specified above

Gradebook.cpp

The Gradebook.cpp file was left to the students to plan in the original assignment. In this version, you will see just a menu driven application that manages a single course object and allows the user to add students, record grades, update grades for specific students, get a course report, record absences, and get reports on specific students.

Compiling and Running the Program:

You can easily compile the program by using

```
g++ Student.cpp Course.cpp Gradebook.cpp
```

If you have no other .cpp files in the folder, you can simplify that to

```
g++ *.cpp
```

You can then run the program simply by typing

```
a.out
```

at the command line.

Note that this program does take optional command line arguments. The first would be a file name from which to read course data at the beginning of the program. The second would be a file name to which to write course data at the end of the program to save it for the next program run. To use command line arguments, you just include in the command:

```
a.out inputfilename outputfilename
```

Your role:

You are fixing this program so that I can't overrun its arrays with file or user input. While the structure of the program is not ideal, I recommend limiting your changes to the fundamental structure of the program in order to keep your efforts reasonable. However, you may change whatever you feel you need to as long as you don't impact that basic functionality of the program. Things like return types are certainly fair game. The program should behave the same for anything that works correctly, but should not break or have other issues if I try to feed too much data.

You must also record all changes that you make (along with anything that you consider beyond the scope of the assignment but want to recommend). Explain what you did (or think should be done) and how it improves the security of the program. You are not forbidden to change things that won't impact buffer overflow, but your grade will primarily depend on the modifications that defend against buffer overflow, so don't get overly distracted by the other issues with the program. For example, there is no input validation at present. You probably only want to expend effort on input validation that will impact buffer overflow.

For your next assignment, you will asked to fix every security issue you can find, but that's the not the focus of this assignment.

Submission:

On the Linux machines, zip your source files and attach the zipped file to the assignment in ReggieNet. Also attach your document (on Windows, Mac, or wherever you chose to write it). When both items are attached, submit.