The University of Alabama in Huntsville
Electrical and Computer Engineering
Project 6 (30 points)

Submit Your Solution Using Canvas by 10pm on Friday September 28, 2018 (late submissions will be accepted on 9/28/18 from 10pm to 11:59pm)

<Project 6 Directions>

Move into your Project_06 directory. This directory was created at the end of Project_01 (The directory path is ~/ **CPE211_FALL18/Project_06** where the ~ represents your home directory path).
Using your favorite text editor, open the file **Project_06.cpp**, type in your solution and save it. If there are syntax errors, correct them and compile again. Once your program successfully compiles, run it and verify that the output for your program matches the output from the provided solution executable – **Project_06_solution**.

Once you are satisfied with your solution, submit Project_06.cpp via Canvas.

NOTE: make sure that you do not change the order in which the information is entered. An automatic script is used to process all lab submissions, and if the order of the input information is modified, the script will not work properly with your program.

Obtaining Project 6 Input Order and Output Requirements: (Sample Solution and Comparison Script)

To view the order and style of the input and output information for the project, run the provided solution by **typing the following at a terminal window prompt.** (**Note**: Input files must be present in the directory that the terminal window is currently in.)

Sample Solution path

/home/work/cpe211/Executables/Project_06/Project_06_solution

Provided sample input files are containged in the file P6_in.zip –
Be sure to test your program with all input files

Comparison Script Path

/home/work/cpe211data/Project_06/CompareSolution.bash Project_06.cpp

Input is to follow the same order as that illustrated by the provided solution executable. Output is to match that shown by the provided solution.

<Project 6 Restrictions>

On Project 6, you may only use concepts presented in Chapters 1 - 7 of your textbook!!
The following specific concepts are not allowed at all:
Global Variables, Arrays and User Defined Functions
➔ Input file can be opened one time only. ➔

If necessary, **global constants** may be used. Global constants are declared above the int main() line

<Project 6 Description>

You will write a program that performs the following tasks (read and understand number 6 and 7):

(1) Command line arguments are to be used to open the input file and the output file. The first attempt for the open is using the file names provided as command line arguments. The first argument is the input file and the second is the output file. The program is to test for the correct number of arguments. **If an incorrect number of arguments is used, output a message and terminate the program** – run the sample solution to see what the message is.

(2) Open a user-specified file for input. If the input file name provided as a command line argument fails to open successfully, enter into a loop that prints out an error message, resets the input file stream variable **(see the hints section #1),** obtains a new file name, echo prints the filename and tries to open the new file. The loop continues until the user successfully enters a valid file name or presses ctrl-c to exit. The first file name tried will come from the command line **(hints #12)**

(3) Open a user-specified file for output. . If the output file name provided as a command line argument fails to open successfully, enter into a loop that prints out an error message, resets the output file stream variable **(see the hints section #1),** obtains a new file name, echo prints the filename and tries to open the new file. The loop continues until the user successfully enters a valid file name or presses ctrl-c to exit. The first file name tried will come from the command line **(hints #12)**

(4) If the input file is empty, your program shall **write a message to the terminal AND the output file** indicating that the input file was empty. Note if the input file is empty, an empty input file message only is written to the output file (no header information). **(see the hints section #2)**

(5) All other output is to be written to the output file

(6) The input file contains **integer assignment scores** for several students (one student and scores per line). For each student in the file, the program reads a student's last name, first name, their quiz grades, their homework grades and their exam scores. Score values are in the range 0 to 100 inclusive. Look at all of the provided input files to see what the information looks like.

(7) The first line in the input file contains 3 integer values separated by spaces: The number of quiz scores, the number of homework grades and the number of exam scores. These values are used with count controlled loops that read in the score information for each student.

(8) The second line in the input file is the key line which provides the maximum score for each assignment. Non empty files will have a key line and information for zero or more students. Empty files will contain no information.

(9) Column header information and the key values are written to the output file as shown by running the sample solution. All column values are placed in a specific field width as specified in hints section #10.

(10) As shown in the sample solution, print a summary of each student's information:
   a. The number associated with the student (from their order in the input file)
   b. Up to 10 characters (use substring function) of the last name,
   c. Up to 5 characters (use substring function) of the first name,
   d. The total of the quiz scores,
   e. The total of the homework scores,
   f. The total of the exam scores, and
   g. The students average (sum of their scores divided by the sum of the key values)

   Print out the average as a percent with a precision of 2 decimal places (i.e. 67.87).

(11) If one or more students are in the class, compute and output the class average after the information for all students has been read and output, Compute and output the class average. If no student information is in the file output a message to the output file and the terminal (as shown in the sample solution)

Possible valid assignment scores are **integers** in the range from 0 to 100, inclusive. If an input file is not empty, you may assume that any data that appears within that file is valid data. Valid files will have at least the key line. No score will exceed 100 points.

Remember the first line of input in the file contains the number of assignments for each category and the second line contains the key values. These are used to determine the maximum score possible for a student, and this maximum score is what is used to determine a student's average. The algorithm provided in the slides indicates one method that the line of key values can be processed differently from the student information.

All scores are integer values, so use integer variables to hold the values read and the sum of the values. Use floating point variables to hold all averaged values (watch out for integer division). **Note: setprecision and fixed affect the number of decimal places shown for floating point values and not integer values.**

<Project 6 C++ Hints>

1. Resetting the input stream variable: In Dev C++ and g++, the clear function must be used on a file stream that is repeatedly reopened in a loop. Some compilers do not require this clearing. Therefore, to allow for more portability of the code, use the following statement in your while loop that executes until a valid filename is entered:

   Input_file_stream_var.clear();

   Where **Input_file_stream_var** is the name of your input file stream variable

2. Testing for an empty input file is performed by tyring to read the number of quiz, homework and exam scores, and then testing the status of the file stream after the read. If the file stream is in the fail state and the end of file status is true, print out an the empty file message and terminate the program. If the file stream is valid after the priming read, , print out the headings and then perform a priming read of a student's last name – before the while loop is entered. If the file stream is valid after this read, proceed to the loop to process and read the rest of the input file

3. If the priming read of the last name results in the input stream entering the fail state, print out a message to the **terminal and the output file** stating that no student information is in the file.

4. Use an end-of-file while loop to read and process all information in the input file – this will be an outer loop. Priming read for this loop is the last name of the student

5. Use three count controlled loops for reading in the assignment scores for the three categories: quiz scores, homework scores and exam scores.

6. All assignment scores will be integer values. Scores used will be valid values in the range of 0 to 100. A negative score indicates that the assignment was not submitted (score counted as a 0) – this value is not added to the total score for the category (quiz, homework or exam) in which it belongs.

7. Integer variables to hold the points a student earned for each category (quizzes, homework and exam) are needed. A floating point variable for holding the average test score for a student is also needed.

8. A floating point variable to hold the sum of the averages for all students is required. An integer variable to hold the number of students in the class is required.

9. Summing the averages for each student and then dividing that sum by the number of students in the class determines the class average.

10. **Procedure for processing a file:** read in the information for a single student, process it, output the necessary information and then read in information for the next student. Use a loop

11. The following line of code creates the dashes under the column headings. It also provides the setw field values for the columns. **All output is left justified in the fields specified**.

```
coutFile << setw(3) << "-" << setw(12) << "----------" << setw(7) << "-----"
         << setw(6) << "----" << setw(6) << "---" << setw(6) << "----"
         << setw(7) << "-----" << setw(9) << "-------" << endl;
```

12. Run the sample solution to see what is contained in the output file – column headings, information output and general format of what the output is supposed to look like.

13. In this assignment the first attempt at input and output filenames will come from the command line when the program is run

    ./Project_06 InputFileName OutputFileName

    If these files fail to open then a loop is entered as described in the project description steps 1 and 2 on page 2

<Project 6 Command Line Arguments>

In this project the initial input and output file names are provided to the program via command line arguments. The program is to initially try to open the filenames provided for input and output. If the open fails for a file, then a loop is entered to prompt the user for the name of the input or output file until the file is successfully opened.

The command typed at the terminal prompt is as shown below and in the hints section:

./Project_06 InputFileName OutputFileName

For the program to be able to use the command line arguments, code must be added to the program to read the arguments.
First, change the main function heading as shown below:

Change: **`int main() to int main(int argc, char* argv[])`**

The variable argc contains the number of command line arguments. This count includes the executable file name. The variable argv[ ] is a character array holding the command line arguments as c-string values. For the following run of a program

```
./Project_06  P6_in1.txt  Out1.txt
```

argc has a value of 3 and the character array has values of argv[0] = ./Project_06, argv[1] = P6_in1.txt and argv[2] = Out1.txt. To use the file names in the program, they can be assigned to a string variable or used directly in the open function. Below, the input file name is assigned to a string variable and the string variable is used in the open function. For the output file, the specified output file is used directly in the open function.

```
string inputFileName;
inputFileName = argv[1]; // assign command line argument value to a string variable
inFile.open(inputFileName.c_str());

outFile.open(argv[2]);   // use command line argument directly in the open function
```