COP 2334 – Programming Project 4

Project Outcomes

Develop a C++ program that:

- uses C++ functions for procedural abstraction and
- uses file input and output to perform computing tasks including generating numerical data from numerical input.

Project Requirements

1. Write a program that reads a series of golfer names and scores from an input file and writes the golfer names and average scores to an output file.
   a. Use top-down design and procedural abstraction to identify the functions to use in the program. The program should include functions for reading a file, calculating an average score, and writing the results to a file.
2. The program should **prompt the user for** both **the input file name and the output file name**. The format of the prompts is at your discretion, but the prompts **must end with a greater than symbol (>), followed by a space**, as in the following:
   `Please enter the name of the input file >[space here]`
3. The golfers' names and scores are in a text file with the following format (see sample included with the assignment):
   b. The first entry is an integer representing the number golfer records.
   c. The second entry is an integer representing the number scores for each golfer.
   d. Subsequent entries are in the following sequence and repeat for each golfer.
      1) Golfer's name
      2) Golfer's Scores
   e. Example of the input file format:

```
3
5
GolferName1
76 80 79 81 140
GolferName2
86 90 99 91 84
GolferName3
71 70 69 71 64
```

4. Use the following to compute the average score for a golfer:
   a. The average score is the sum of scores divided by the number of scores.
   b. The sum of scores **will not include the highest score**.
   c. The sum of scores will **include valid scores only**.
      1) To be valid, a score must be **between 50 and 130 inclusive**.
   d. **If an invalid score is encountered** while reading the golfers' scores from the input file,
      1) **do not include the invalid score in the total of all scores or the number of scores**, and

2) **display an error message that includes the golfer's name and the invalid score**.
   a) The format of the error message is at your discretion, but it must include the golfer's name and the invalid score, in that order. For example:
   ```
   GolferName1 score of 140 is invalid
   ```
3) Note: an invalid score should be discarded completely, as if it did not exist in the file. Therefore, an invalid score cannot be the highest score, is not included in the total of scores, and is not included in the number of scores.

5. The format of the output file is at your discretion, but **must include the golfer name and average score**, in that order, such as:
```
Player: GolferName1 Average Score: 78.33
Player: GolferName2 Average Score: 87.75
Player: GolferName3 Average Score: 68.50
```
6. **The following functions are required**:
   a. ***openInputFile*** function – gets name of input file from user and opens the file
      1) Has an ifstream reference parameter.
      2) Prompts the user for a file name.
      3) Returns a bool that indicates whether the file was opened successfully.
   b. ***openOutputFile*** function – gets name of output file from user and opens the file
      1) Has ofstream reference parameter.
      2) Prompts the user for a file name.
      3) Returns a bool that indicates whether the file was opened successfully.
   c. ***processInputData*** function – processes average and prints data to output file
      1) Has ifstream and ofstream reference parameters.
      2) Reads the data from the input file.
      3) Uses appropriate loops and variables to process the data.
      4) Calls the ***isValidScore*** function to test the validity of each score.
      5) Calls ***setMaxScore*** to set the max score.
      6) Calls the ***writeHeader*** and the ***writeGolferInfo*** functions to write the appropriate data to the output file.
   d. ***writeHeader*** function – writes the header for the output
      1) Has an ofstream reference parameter.
      2) Writes appropriate file header to the output file.
      3) The header should contain the **total number of golfers and the number of scores**.
      ```
      **********************
      **  GOLFER REPORT  **
      **********************
      Number of golfers processed: 3
      Number of scores processed: 5
      ======================================
      ```
   e. ***writeGolferInfo*** function - writes data for a golfer to file
      1) Has an ofstream reference parameter.

2) Writes the golfer's name and average score to the output file. The average score must have **exactly two digits after the decimal point**. For example:
   `Player: firstName1 Average Score: 78.33`

f. ***isValidScore*** function – determines whether a score is valid
   1) Has an int parameter.
   2) Returns true if score is valid (between 50 and 130 inclusive) or false if not.

g. ***setMaxScore*** function – sets the max score
   1) Has an int parameter for the current score and an int reference parameter for the max score for that player.
   2) Updates the max score if the current score is greater than the max score.

Implementation Notes

1. **Do not use arrays in the program.**
2. You may assume that the input file is in correct format.
3. The program should have **no output to the display other than the prompts for file names, error messages for invalid scores, and (optionally) a terminating message if the input or output file could not be opened**.
4. Note that you can test your program on data that is easy to hand tally so that you know the correct output you should get before your program ever runs. That is the only way to know that your program works properly.

Important Notes

1. Projects will be graded on whether they correctly solve the problem and whether they adhere to good programming practices.
2. Projects must be submitted via Canvas/eLearning by the time specified on the due date. Projects submitted after that time will receive a grade of zero.
3. Please review UWF's academic conduct policy that was described in the syllabus. Note that viewing another student's solution, whether in whole or in part, is considered academic dishonesty. Further, allowing another student to view your code is considered academic dishonesty. Finally, submitting code obtained via the internet or other sources, whether in whole or in part, is considered academic dishonesty. All programs submitted will be reviewed for evidence of academic dishonesty, and all violations will be handled accordingly.