1 Instructions

You may work in pairs (that is, as a group of two) with a partner on this lab project if you wish or you may work alone. If you work with a partner, only of you will submit the lab project to Canvas for grading. To ensure you each receive the same score, it is imperative that you follow the instructions in §5 *What to Submit for Grading and by When* (i.e., put both of your ASURITE ID's in the submission archive filename and write both of your names, ASURITE ID's, and email addresses in the AUTHOR1: and AUTHOR2: lines of the header comment block for each source code file). What to submit for grading, and by when, is discussed in §5; read it now.

2 Lab Objectives

- Complete all of the objectives of the previous lab projects.
- Continue learning to write functions definitions, call functions, pass parameters by-value and by-reference, define local variables, and return values from functions.
- Write function prototypes.
- Use the left, right, and setw stream manipulators.
- Continue learning to write programs using text file I/O.

3 Prelab Exercises

3.1 Prelab Exercise 1: Read Software Requirements

Read §4 *Software Requirements* of this lab project document which describes what the lab project program will do, i.e., the **software requirements**. Then come back here and complete the remaining prelab exercises.

3.2 Prelab Exercise 2: Create Repl.it Project and Add *main.cpp* to the Project

You may use any C++ development environment you wish to complete the project. As we did for Lab Project 4, we will be opening an input file for reading and an output file for writing in this program. If you wish to use an online C++ IDE, we recommend that you use Repl.it.

If you have not already done so, before your lab session, create a free Repl.it account. After doing that, create a new C++ project and rename the project "Lab 5" (Note: if you do not have an account and are logged in as @anonymous, you will not be able to rename the project from the default random project name that was assigned to it when the project was created.) Then upload the *main.cpp* source code file you extracted from the Lab Project 5 zip archive to your Repl.it project, overwriting the *main.cpp* file that was added to the project when you created it.

For help using Repl.it, click the ? button in the lower-right corner of the IDE window. Here is a direct link to the Repl.it documentation. If you require more assistance, ask your lab TA for help.

Before proceeding to the next prelab exercise, edit the header comment block of the provided *main.cpp* file and add your information to the AUTHOR1: comment. If you work with a partner, add the AUTHOR2: comment including his or her information. Remember that only one member of the team has to upload the submission zip archive to Canvas but you will each earn the same score as long as you follow the instructions in this section and in §5.

If you require assistance with Repl.it, please ask your TA or UGTA in the lab session for help. Note that when when you are finished with the lab project, you may download your project files in a zip archive: Click the three dots symbol in the title bar of the *File Pane* and select *Download as Zip* from the menu. This will download a zip archive named *Lab-5.zip* (assuming you named your project *Lab 5* as requested).

Note: the downloaded zip archive is **not** the zip archive you are required to upload to Canvas for grading. Please consult §5 for that information.

3.3 Prelab Exercise 3: Create Test Input File *stats-season.txt*

The downloaded zip archive contains a sample input file named *stats-season.txt* in the *input* subdirectory of the Lab 5 Project Zip Archive. The program you are writing for this lab will read this input file, perform calculations on the input data, and write a new output file named *stats-game.txt*. If you are using Repl.it, you may upload *stats-season.txt* by clicking on the three dots in the Files pane menu and selecting Upload File from the menu. Navigate to find *stats-season. txt* and then upload it.

If you are using some other C++ IDE, e.g., Eclipse, CodeLite, Code::Blocks, or Xcode, then you will have to determine for that particular IDE where to put the *stats-season.txt* input file within your project directory. If the input file is not in the proper project directory, then when your program runs it will fail to open the input file. Some IDE's expect the input file to be in the same directory as source code files, so you could start by putting it in the same project directory as *main.cpp*. The code in *open_input_file*() and *open_output_file*() is designed to check for this failure and if the input or output file cannot be opened, then

the *terminate*() function will be called which will gracefully shut down the program. If this happens, then try placing the input file in the project directory *above* the directory containing the source code file *main.cpp* and then re-run your program. If you continue to struggle with this, please ask your lab TA or UGTA for assistance.

3.4 Prelab Exercise 4: Study the Software Design

I did not document the design using a structure chart and pseudocode in this lab document. Rather, for this lab, you will complete the main.cpp source code template found in the *src* subdirectory of the Lab Project 5 Zip Archive by reading the comments and determining what code to write to implement the specifications in the comments. As we did in Lab Project 4, we shall design and write this program using multiple functions. There are seven required functions, including the *main*() function which is always required in every C++ program. The functions and the *prototypes* are:

- void compute_per_game_stats (ifstream& fin, ofstream& fout)
- void open_input_file (ifstream& fin, string filename)
- void open_output_file (ofstream& fout, string filename)
- void output_header (ofstream& fout)
- void output_player_stats (ofstream& fout, string name, int gp, double fgp, double ftp, double threep, double ppg, double rpg)
- void terminate (string msg)
- int main ()

Read the function comment block above every function in *main.cpp* to learn what the function does. The flow of control for the program is:

1. Execution begins with the first two statement in *main*() where *fin* and *fout* are defined.
2. *main*() calls *open_input_file*() passing *fin* as the argument (note that *fin* is passed by-reference because we modify *fin* in *open_input_file*() and those changes need to be sent back to *main*).
3. If the input file cannot be opened for reading because it does not exist or is located in the wrong project directory, then *open_input_file*() will call *terminate*() and the program will end. Otherwise, if the input file is successfully opened, then *open_input_file*() will return the file input stream object *fin* as an output parameter back to *main*().
4. *main*(() calls *open_output_file*() passing *fout* as the argument (note that *fout* is passed by-reference because we modify *fout* in *open_output_file*() and those changes need to be sent back to *main*). As with *open_input_file*(), *open_output_file*() will either fail to open the output file, in which case *terminate*() shall be called and the program will end, or, *open_output_file*() will successfully open the output file and shall return the *fout* file output stream object as the output parameter back to *main*().
5. *main*() calls *output_header*() to output the header line containing the column labels to the output file; then *output_header*() returns back to *main*().
6. *main*() calls *compute_per_game_stats*() which will compute the per game statistics for each player in the input file and shall write the remainder of the output to the output file; then, *compute_per_game_stats*() returns back to *main*().
7. *main*() closes the input and output files and then returns 0 to end the program.

4 Software Requirements

A text file named *stats-season.txt* contains season statistics for a basketball team in the format illustrated by this example:

```
F.Flintstone    23  88  190  50  82  27  74  253  100
B.Rubble        23  85  212  45  53  24  84  239   94
W.Flintstone    14  56  101  29  46  0   1   141   82
H.Simpson       23  77  118  62  107 1   3   216   143
N.Flanders      23  57  127  15  17  37  91  166   51
N.Explosion     23  53  109  9   14  46  95  161   63
K.Broflovski    21  31  100  57  79  8   29  127   38
E.Cartman       18  28  66   10  12  17  37  83    22
G.Jetson        17  26  58   19  29  3   16  74    37
S.Skwigelf      18  19  43   18  25  3   9   59    20
T.Wartooth      22  19  59   23  40  7   23  68    46
end
```

The contents of each column from left to right is,

Name	The player's name (a string).
GP	Games played (an integer).
FGM	Field goals made (an integer).
FGA	Field goals attempted (an integer).
FTM	Free throws made (an integer).
FTA	Free throws attempted (an integer).
THREEM	Three point field goals made (an integer).
THREEA	Thee point field goals attempted (an integer).
PTS	The total points scored (an integer).
REBS	The total rebounds (an integer).

The last line of the file must contain the word "end". As team statistician, your job is to write a C++ program which reads *stats-season.txt*, calculates the per game statistics for each player, and writes the per game statistics to an output file named *stats-game.txt*. The per game statistics that shall be calculated are,

PPG	Points per game which is PTS divided by GP.
RPG	Rebounds per game which is REBS divided by GP.
FGP	Field goal percentage which is FGM divided by FGA.
FTP	Free throw percentage which is FTM divided by FTA.
THREEP	Three point field goal percentage which is THREEM divided by THREEA.

For the example *stats-season.txt* input file, the contents of *stats-game.txt* will be,

```
Name          GP   PPG   RPG    FG%    FT%    3P%
F.Flintstone   23  11.0   4.3  0.463  0.610  0.365
B.Rubble       23  10.4   4.1  0.401  0.849  0.286
W.Flintstone   14  10.1   5.9  0.554  0.630  0.000
H.Simpson      23   9.4   6.2  0.653  0.579  0.333
N.Flanders     23   7.2   2.2  0.449  0.882  0.407
N.Explosion    23   7.0   2.7  0.486  0.643  0.484
K.Broflovski   21   6.0   1.8  0.310  0.722  0.276
E.Cartman      18   4.6   1.2  0.424  0.833  0.459
G.Jetson       17   4.4   2.2  0.448  0.655  0.188
S.Skwigelf     18   3.3   1.1  0.442  0.720  0.333
T.Wartooth     22   3.1   2.1  0.322  0.575  0.304
```

Note how the output is aligned in columns. This is accomplished by outputting each value in a field of a specific width using the *setw*() stream manipulator (study the comments in the code). Note that PPG and RPG are printed with 1 digit after the decimal point and all percentages are displayed with 3 digits after the decimal point. The names are printed left-justified in a column and the numerical values are printed right-justified in each column. This is accomplished with the *left* and *right* stream manipulators.

4.1 Additional Programming Requirements

1. Modify the **header comment block** at the top of *main.cpp* source code file so it contains the first author's information in the AUTHOR1: comment. If there is only one author, then delete the AUTHOR2: comment. However, if there are two authors, document the second author's information in the AUTHOR2: comment.

2. There are no required test cases for this project. Use the provided *stats-season.txt* file from the *input* subdirectory of the Lab Project 5 Zip Archive. When your program runs, it should produce *stats-game.txt*. Compare the output file your program produces to the correct output shown in §4. If there are any mismatches, then you have a bug in your program, so locate the bug(s), fix them, and make sure that the program you submit for grading produces the correct output file.

3. Always write your code in a way to **enhance readability**. This includes **writing comments** to explain what the code is doing, **properly indenting** the statements inside *main*() and the other functions, and **using blank lines** to separate the various parts of the program. For guidance, study the example programs in the textbook and the source code the instructor writes in class or posts online.

5 What to Submit for Grading and by When

- Using the file explorer program of your operating system, create a new empty folder named **lab05-*asurite*** where *asurite* is your ASURITE username that you use to log in to MyASU (e.g., mine is *kburger2* so my folder would be named lab05-kburger2). If you worked with a partner on the project, put both of your ASURITE usernames in the name of the new folder **lab05-*asurite1*-*asurite2***, e.g., Fred Flintstone's and Barney Rubble's project submission archive would be named **lab05-fflint-brubble.zip**.
- Copy your completed *main.cpp* source code file to the folder your just created. This folder should contain only one file named *main.cpp*. Do **not** copy the input file *stats-season.txt* or the output file *stats-game.txt* to this folder. We do not need the data files because we will use our own when testing your program.
- Compress the folder creating a **zip archive** named **lab05-*asurite*.zip** or **lab05-*asurite1*-*asurite2*.zip**.
- Submit the zip archive to Canvas by the deadline using the *Lab Project 5* submission link.
- If you worked with a partner, then only one partner of the team submits to Canvas, but you will each receive the same score as long as you follow the rules for documenting both authors in the header comment block and properly naming your zip archive.
- If your program does not compile or run correctly, upload what you have completed for grading anyway because you will generally receive some partial credit for effort.
- The submission deadline is **11:59pm Sat 03/13**, with a 48 hour late submission deadline of **11:59pm Mon 03/15.**
- The Canvas submission link will become unavailable exactly at 11:59:00pm on Mon 03/15 and you will not be able to submit your project for grading after that time. We do not accept emailed submissions.
- Consult the online syllabus for the late project submission and academic integrity policies.

Note that we may employ an automated grading script to grade your lab projects. This script will depend on you naming the submitted files as requested in this section. Failure to follow this naming convention may result in points being deducted, up to a score of 0 being assigned for your lab project, even if it builds and runs correctly.

6 Grading Rubric

1. Testing

Content intentionally omitted.

2. Lab Program (0 to 5 pts)

Build the student's program. If it does not compile due to syntax errors, then assign partial credit for effort as explained below in items e-g. If the program does build, then run the program using the *stats-season.txt* file and compare the output from the student's program to the correct output shown above; then assign points as explained in items a-d.

a. **Assign 5 pts**: Program compiles; it runs to completion without terminating early due to an I/O file not being opened; the values in the output file are correct and the output file is properly formatted.
b. **Assign 4.5 pts**: Program compiles; it runs to completion without terminating early due to an I/O file not being opened; most of the values in the output file are correct and/or the output file is mostly properly formatted.
c. **Assign 4 pts**: Program compiles; it runs to completion without terminating due to an I/O file not being opened; most or all of the values in the output file are *in*correct and/or the file formatting is significantly wrong.
d. **Assign 3.5 pts**: Program compiles; it terminates early due to either the input or output file not being opened.
e. **Assign 3 pts**: Program does not compile due to syntax errors; the student made a good attempt and implemented the majority of the required code:
f. **Assign 2 pts**: Program does not compile due to syntax errors; the student made a poor attempt and implemented only a little of the required code.
g. **Assign 0 pts**: Program does not compile due to syntax errors; the student did not implement any of the required code.

3. Deadline was 11:59pm Sat 13 Mar

a. Deduct 0.5 pt (-10%) for a submission on Sun 14 Mar.
b. Deduct 1 pt (-20%) for a submission on Mon 15 Mar.
c. After Mon 15 Mar, late projects are not accepted for any reason.