THIS WORK IS FOR INDIVIDUAL STUDENT

(An Informal Part of Assignment One)

(Informal Part of Quiz One will be in Next Class on Tue March 18)

DUE, HANDWRITTEN ONLY, IN CLASS TUE MARCH 18, 2025

CONCEPT TOPICS: CSTRING AND FILES (read the attached pages in this PDF from the textbook)

Task 1: Please write, in your own words, the answer to the following questions:

1. What is a Cstring?
2. What is the difference between a “char” array and a Cstring?
3. What is the difference between Size of array and length of cstring?

Task 2: Please write, in your own words, the answer to the following questions (read pages from textbook, use ChatGPT and internet):

1. In C++, what is an input stream?
2. How does “cin” works?
3. In C++, what is an output stream?
4. How does “cout” works?
5. In C++, what is fstream?
6. In C++, what is iftream?
7. In C++, what is ofstream?
8. How to read values from a text file?
9. How to write values to a text file?

CODING PROBLEMS ON NEXT PAGE

Task 3: Write codes as indicated by each problem below:

1. A complete C++ program that does the following steps:
    a. read an English word (word one) from the user and display on screen
    b. finds and displays the length of the word on screen
    c. Copy the word in to another word (word two)and display both words on screen
    d. Convert both words all to UPPERCASE and display both words on screen
    e. Read another word (word 3)from the user and display on screen
    f. Append the first word to the last word (3rd word) and display first and 3rd words
    g. Insert a space in the 3rd word after the first word and display the 3rd word
    h. Read a single character from the user and display if that character is present in the 3rd words or not
    i. Now display the number of times the single character is present in the 3rd word

   A sample run of the above program should look like the following (user inputs are shown in **bold and blue)**:

```
a. Please enter an English word: Fundamentals
You have entered: Fundamentals
b. The length of “Fundamentals” is 13
c. After copying both words: Fundamentals Fundamentals
d. After Conversion of both words to all UPPERCASE: FUNDAMENTALS FUNDAMENTALS
e. Please another word: Programming
f. after appending first word “FUNDAMENTALS” to 3rd word “Programming”:
ProgrammingFUNDAMENTALS
g. After insertion of space after the word “Programming”: Programming FUNDAMENTALS
h. Please enter a single character to find: m
m is present in the 3rd word “Programming Fundamentals”
i. m is present 2 times in the 3rd word “Programming FUNDAMENTALS”
```

2. Write a complete C++ program that reads an English sentence from the User and display each word in separate lines on screen
   A sample run of the above program should look like the following (user inputs are shown in **bold and blue)**:

```
Enter an English Sentence ending in ‘.’: This is good programming exercise.
You have entered:
        This
        is
        good
        programming
        exercise.
```

Pages from textbook are attached after this page

```
    cout << "Enter temperature in Fahrenheit: ";      //Line 7
    cin >> fahrenheit;                                 //Line 8
    cout << endl;                                      //Line 9

    celsius = static_cast<int>
                (5.0 / 9 * (fahrenheit - 32) + 0.5);   //Line 10

    cout << fahrenheit << " degree F = "
         << celsius << " degree C. " << endl;          //Line 11

    return 0;                                          //Line 12
}                                                      //Line 13
```

Sample Run: In this sample run, the user input is shaded.

```
Enter temperature in Fahrenheit: 110

110 degree F = 43 degree C.
```

As we can see, using temporary `cout` statements, we were able to find the problem. After correcting the problem, the temporary `cout` statements are removed.

The temperature conversion program contained logic errors, not syntax errors. Using `cout` statements to print the values of expressions and/or variables to see the results of a calculation is an effective way to find and correct logic errors.

File Input/Output

The previous sections discussed in some detail how to get input from the keyboard (standard input device) and send output to the screen (standard output device). However, getting input from the keyboard and sending output to the screen have several limitations. Inputting data in a program from the keyboard is comfortable as long as the amount of input is very small. Sending output to the screen works well if the amount of data is small (no larger than the size of the screen) and you do not want to distribute the output in a printed format to others.

If the amount of input data is large, however, it is inefficient to type it at the keyboard each time you run a program. In addition to the inconvenience of typing large amounts of data, typing can generate errors, and unintentional typos cause erroneous results. You must have some way to get data into the program from other sources. By using alternative sources of data, you can prepare the data before running a program, and the program can access the data each time it runs.

Suppose you want to present the output of a program in a meeting. Distributing printed copies of the program output is a better approach than showing the output on a screen. For example, you might give a printed report to each member of a committee before an important meeting. Furthermore, output must sometimes be saved so that the output produced by one program can be used as an input to other programs.

This section discusses how to obtain data from other input devices, such as a flash drive (that is, secondary storage), and how to save the output to a flash drive. C++ allows a program to get data directly from and save output directly to secondary storage. A program can use the file I/O and read data from or write data to a file. Formally, a file is defined as follows:

File: An area in secondary storage used to hold information.

The standard I/O header file, `iostream`, contains data types and variables that are used only for input from the standard input device and output to the standard output device. In addition, C++ provides a header file called `fstream`, which is used for file I/O. Among other things, the `fstream` header file contains the definitions of two data types: `ifstream`, which means input file stream and is similar to `istream`, and `ofstream`, which means output file stream and is similar to `ostream`.

The variables `cin` and `cout` are already defined and associated with the standard I/O devices. In addition, `>>`, `get`, `ignore`, `putback`, `peek`, and so on can be used with `cin`, whereas `<<`, `setfill`, and so on can be used with `cout`. These same operators and functions are also available for file I/O, but the header file `fstream` does not declare variables to use them. You must declare variables called **file stream variables**, which include `ifstream` variables for input and `ofstream` variables for output. You then use these variables together with `>>`, `<<`, or other functions for I/O. Remember that C++ does not automatically initialize user-defined variables. Once you declare the `fstream` variables, you must associate these file variables with the I/O sources.

File I/O is a five-step process:

1. Include the header file `fstream` in the program.
2. Declare file stream variables.
3. Associate the file stream variables with the I/O sources.
4. Use the file stream variables with `>>`, `<<`, or other I/O functions.
5. Close the files.

We will now describe these five steps in detail. A skeleton program then shows how the steps might appear in a program.

Step 1 requires that the header file `fstream` be included in the program. The following statement accomplishes this task:

```
#include <fstream>
```

Step 2 requires you to declare file stream variables. Consider the following statements:

```
ifstream inData;
ofstream outData;
```

The first statement declares `inData` to be an input file stream variable. The second statement declares `outData` to be an output file stream variable.

Step 3 requires you to associate file stream variables with the I/O sources. This step is called **opening the files**. The stream member function **`open`** is used to open files. The syntax for opening a file is:

```
fileStreamVariable.open(sourceName);
```

Here, **`fileStreamVariable`** is a file stream variable, and **`sourceName`** is the name of the I/O file.

Suppose you include the declaration from Step 2 in a program. Further suppose that the input data is stored in a file called **`prog.dat`**. The following statements associate **`inData`** with **`prog.dat`** and **`outData`** with **`prog.out`**. That is, the file **`prog.dat`** is opened for inputting data, and the file **`prog.out`** is opened for outputting data.

```
inData.open("prog.dat");  //open the input file;  Line 1
outData.open("prog.out"); //open the output file; Line 2
```

NOTE

IDEs such as Visual Studio .Net manage programs in the form of projects. That is, first you create a project, and then you add source files to the project. The statement in Line 1 assumes that the file **`prog.dat`** is in the same directory (subdirectory) as your project. However, if this is in a different directory (subdirectory), then you must specify the path where the file is located, along with the name of the file. For example, suppose that the file **`prog.dat`** is on a flash memory in drive H. Then the statement in Line 1 should be modified as follows:

```
inData.open("h:\\prog.dat");
```

Note that there are two \ after h:. Recall from Chapter 2 that in C++ , \ is the escape character. Therefore, to produce a \ within a string, you need \\. (To be absolutely sure about specifying the source where the input file is stored, such as the drive **`h:\\`**, check your system's documentation.)

Similar conventions for the statement in Line 2.

NOTE

Suppose that a program reads data from a file. Because different computers have drives labeled differently, for simplicity, throughout the book, we assume that the file containing the data and the program reading data from the file are in the same directory (subdirectory).

NOTE

We typically use **`.dat`**, **`.out`**, or **`.txt`** as an extension for the input and output files and use Notepad, Wordpad, or TextPad to create and open these files. You can also use your IDE's editor, if any, to create **`.txt`** (text) files. (To be absolutely sure about it, check your IDE's documentation.)

Step 4 typically works as follows. You use the file stream variables with **`>>`**, **`<<`**, or other I/O functions. The syntax for using **`>>`** or **`<<`** with file stream variables is exactly the same as the syntax for using **`cin`** and **`cout`**. Instead of using **`cin`** and **`cout`**, however, you use the file stream variable names that were declared. For example, the statement:

```
inData >> payRate;
```

reads the data from the file **prog.dat** and stores it in the variable **payRate**. The statement:

```
outData << "The paycheck is: $" << pay << endl;
```

stores the output—**The paycheck is: $565.78**—in the file **prog.out**. This statement assumes that the pay was calculated as **565.78**.

Once the I/O is complete, Step 5 requires closing the files. Closing a file means that the file stream variables are disassociated from the storage area and are freed. Once these variables are freed, they can be reused for other file I/O. Moreover, closing an output file ensures that the entire output is sent to the file; that is, the buffer is emptied. You close files by using the stream function **close**. For example, assuming the program includes the declarations listed in Steps 2 and 3, the statements for closing the files are:

```
inData.close();
outData.close();
```

NOTE

On some systems, it is not necessary to close the files. When the program terminates, the files are closed automatically. Nevertheless, it is a good practice to close the files yourself. Also, if you want to use the same file stream variable to open another file, you must close the first file opened with that file stream variable.

In skeleton form, a program that uses file I/O usually takes the following form:

```
#include <fstream>

//Add additional header files you use

using namespace std;

int main()
{
        //Declare file stream variables such as the following
    ifstream inData;
    ofstream outData;
    .
    .
    .

        //Open the files
    inData.open("prog.dat");  //open the input file
    outData.open("prog.out"); //open the output file

        //Code for data manipulation

        //Close files
    inData.close();
    outData.close();

    return 0;
}
```

Recall that Step 3 requires the file to be opened for file I/O. Opening a file associates a file stream variable declared in the program with a physical file at the source, such as a flash drive. In the case of an input file, the file must exist before the **open** statement executes. If the file does not exist, the **open** statement fails and the input stream enters the fail state. An output file does not have to exist before it is opened; if the output file does not exist, the computer prepares an empty file for output. If the designated output file already exists, by default, the old contents are erased when the file is opened.

NOTE

To add the output at the end of an existing file, you can use the option **ios::app** as follows. Suppose that **outData** is declared as before and you want to add the output at the end of the existing file, say, **firstProg.out**. The statement to open this file is:

```
outData.open("firstProg.out", ios::app);
```

If the file **firstProg.out** does not exist, then the system creates an empty file.

NOTE

Appendix E discusses binary and random access files.

PROGRAMMING EXAMPLE: Movie Tickets Sale and Donation to Charity

Watch the Video

A movie in a local theater is in great demand. To help a local charity, the theater owner has decided to donate to the charity a portion of the gross amount generated from the movie. This example designs and implements a program that prompts the user to input the movie name, adult ticket price, child ticket price, number of adult tickets sold, number of child tickets sold, and percentage of the gross amount to be donated to the charity. The output of the program is as follows.

```
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
Movie Name: ....................... Journey to Mars
Number of Tickets Sold: ...........      2650
Gross Amount: ..................... $ 9150.00
Percentage of Gross Amount Donated:     10.00%
Amount Donated: ................... $  915.00
Net Sale: ......................... $ 8235.00
```

Note that the strings, such as **"Movie Name:"**, in the first column are left-justified, the numbers in the right column are right-justified, and the decimal numbers are output with two decimal places.

Input The input to the program consists of the movie name, adult ticket price, child ticket price, number of adult tickets sold, number of child tickets sold, and percentage of the gross amount to be donated to the charity.

Output The output is as shown above.