

## LESSON SET

# 5

## Loops and Files

### PURPOSE

1. To introduce counter and event controlled loops
2. To work with the `while` loop
3. To introduce the `do-while` loop
4. To work with the `for` loop
5. To work with nested loops
6. To briefly introduce the concept of files

### PROCEDURE

1. Students should read the Pre-lab Reading Assignment before coming to lab.
2. Students should complete the Pre-lab Writing Assignment before coming to lab.
3. In the lab, students should complete labs assigned to them by the instructor.

Contents	Pre-requisites	Approximate completion time	Page number	Check when done
Pre-lab Reading Assignment		20 min.	56	
Pre-lab Writing Assignment	Pre-lab reading	10 min.	65	
LESSON 5A				
Lab 5.1 Working with the <code>while</code> Loop	Basic understanding of the <code>while</code> loop	25 min.	65	
Lab 5.2 Working with the <code>do-while</code> Loop	Basic understanding of <code>do-while</code> loop	25 min.	67	
LESSON 5B				
Lab 5.3 Working with the <code>for</code> Loop	Understanding of <code>for</code> loops	15 min.	69	
Lab 5.4 Nested Loops	Understanding of nested <code>for</code> loops	15 min	70	
Lab 5.5 Reading and Writing to a File	Basic understanding of reading and writing files	15 min.	72	<i>continues</i>

## Lab 5.6

Student Generated Code  
AssignmentsBasic understanding  
of loop control structures

30 min.

73

## PRELAB READING ASSIGNMENT

## Increment and Decrement Operator

To execute many algorithms we need to be able to add or subtract 1 from a given integer quantity. For example:

```
count = count + 1;    // what would happen if we used ==
                     // instead of = ?

count += 1;
```

Both of these statements **increment** the value of `count` by 1. If we replace “+” with “-” in the above code, then both statements **decrement** the value of `count` by 1. C++ also provides an **increment operator** `++` and a **decrement operator** `--` to perform these tasks. There are two modes that can be used:

```
count++;    // increment operator in the postfix mode
count--;    // decrement operator in the postfix mode

++count;    // increment operator in the prefix mode
--count;    // decrement operator in the prefix mode
```

The two increment statements both execute exactly the same. So do the decrement operators. What is the purpose of having postfix and prefix modes? To answer this, consider the following code:

```
int age = 49;
if (age++ > 49)
    cout << "Congratulations - You have made it to the half-century"
        << " mark !" << endl;
```

In this code, the `cout` statement will not execute. The reason is that in the postfix mode the comparison between `age` and 49 is made *first*. Then the value of `age` is incremented by one. Since 49 is not greater than 49, the `if` conditional is false. Things are much different if we replace the postfix operator with the prefix operator:

```
int age = 49;
if (++age > 49)
    cout << " Congratulations - You have made it to the half-century"
        << " mark !" << endl;
```

In this code `age` is incremented first. So its value is 50 when the comparison is made. The conditional statement is true and the `cout` statement is executed.

The `while` Loop

Often in programming one needs a statement or block of statements to repeat during execution. This can be accomplished using a **loop**. A loop is a control structure that causes repetition of code within a program. C++ has three types of loops. The first we will consider is the **while loop**. The syntax is the following:

```
while (expression)
{
    statement_1;
    statement_2;
    :
    statement_n;
}
```

If there is only one statement, then the curly braces can be omitted. When a `while` loop is encountered during execution, the expression is tested to see if it is true or false. The block of statements is repeated as long as the expression is true. Consider the following:

*Sample Program 5.1:*

---

```
#include <iostream>
using namespace std;

int main()
{
    int num = 5;
    int numFac = 1;

    while (num > 0)
    {
        numFac = numFac * num;
        num--;          // note the use of the decrement operator
    }

    cout << " 5! = " << numFac << endl;

    return 0;
}
```

---

This program computes  $5! = 5 * 4 * 3 * 2 * 1$  and then prints the result to the screen. Note how the `while` loop controls the execution. Since `num = 5` when the `while` loop is first encountered, the block of statements in the body of the loop is executed at least once. In fact, the block is executed 5 times because of the decrement operator which forces the value of `num` to decrease by one every time the block is executed. During the fifth **iteration** of the loop `num` becomes 0, so the next time the expression is tested `num > 0` is false and the loop is exited. Then the `cout` statement is executed.

What do you think will happen if we eliminated the decrement operator `num--` in the above code? The value of `num` is always 5. This means that the expression `num > 0` is always true! If we try to execute the modified program, the result is an **infinite loop**, i.e., a block of code that will repeat forever. One must be very cautious when using loops to ensure that the loop will terminate. Here is another example where the user may have trouble with termination.

*Sample Program 5.2:*

---

```
#include <iostream>
using namespace std;

int main()
{
    char letter = 'a';

    while (letter != 'x')
```

---

*continues*

```

    {
        cout << "Please enter a letter" << endl;
        cin >> letter;
        cout << "The letter your entered is " << letter << endl;
    }
    return 0;
}

```

---

Note that this program requires input from the user during execution. Infinite loops can be avoided, but it would help if the user knew that the 'x' character terminates the execution. Without this knowledge the user could continually enter characters other than 'x' and never realize how to terminate the program. In the lab assignments you will be asked to modify this program to make it more user friendly.

## Counters

Often a programmer needs to control the number of times a particular loop is repeated. One common way to accomplish this is by using a **counter**. For example, suppose we want to find the average of five test scores. We must first input and add the five scores. This can be done with a **counter-controlled** loop as shown in Sample Program 5.3. Notice how the variable named `test` works as a counter. Also notice the use of a constant for the number of tests. This is done so that the number of tests can easily be changed if we want a different number of tests to be averaged.

### *Sample Program 5.3:*

```

#include <iostream>
using namespace std;

const int NUMBEROFTESTS = 5;

int main()
{
    int score ;                // the individual score read in
    float total = 0.0;         // the total of the scores
    float average;             // the average of the scores
    int test = 1;              // counter that controls the loop

    while (test <= NUMBEROFTESTS) // Note that test is 1 the first time
                                    // the expression is tested
    {
        cout << "Enter your score on test " << test << ": " << endl;
        cin >> score;

        total = total + score;
        test++;
    }

    average = total / NUMBEROFTESTS;
}

```

```

cout << "Your average based on " << NUMEROFTESTS
    << " test scores is " << average << endl;

return 0;
}

```

---

Sample Program 5.3 can be made more flexible by adding an integer variable called `numScores` that would allow the user to input the number of tests to be processed.

## Sentinel Values

We can also control the execution of a loop by using a **sentinel value** which is a special value that marks the end of a list of values. In a variation of the previous program example, if we do not know exactly how many test scores there are, we can input scores which are added to total until the sentinel value is input. Sample Program 5.4 revises Sample Program 5.3 to control the loop with a sentinel value. The sentinel in this case is -1 since it is an invalid test score. It does not make sense to use a sentinel between 0 and 100 since this is the range of valid test scores. Notice that a counter is still used to keep track of the number of test scores entered, although it does not control the loop. What happens if the first value the user enters is a -1?

### *Sample Program 5.4:*

---

```

#include <iostream>
using namespace std;

int main()
{
    int score ;           // the individual score read in
    float total = 0.0;    // the total of the scores
    float average;        // the average of the scores
    int test = 1;         // counter that controls the loop

    cout << "Enter your score on test " << test
        << " (or -1 to exit): " << endl;
    cin >> score;          // Read the 1st score

    while (score != -1)    // While we have not entered the sentinel
                          // (ending) value, do the loop
    {

        total = total + score;
        test++;

        cout << "Enter your score on test " << test
            << " (or -1 to exit): " << endl;
        cin >> score;      // Read the next score
    }
}

```

*continues*

```

    }

    if (test > 1)                // If test = 1, no scores were entered
    {
        average = total / (test - 1);

        cout << "Your average based on " << (test - 1)
              << " test scores is " << average << endl;
    }

    return 0;
}

```

Notice that the program asks for input just before the `while` loop begins and again as the last instruction in the `while` loop. This is done so that the `while` loop can test for sentinel data. Often this is called **priming the read** and is frequently implemented when sentinel data is used to end a loop.

### Data Validation

One nice application of the `while` loop is data validation. The user can input data (from the keyboard or a file) and then a `while` loop tests to see if the value(s) is valid. The loop is skipped for all valid input but for invalid input the loop is executed and prompts the user to enter new (valid) input. The following is an example of data validation.

```

cout << "Please input your choice of drink "
     << "(a number from 1 to 4 or 0 to quit)" << endl;
cout << " 1  - Coffee" << endl
     << " 2  - Tea" << endl
     << " 3  - Coke" << endl
     << " 4  - Orange Juice" << endl << endl
     << " 0  - QUIT" << endl << endl;
cin >> beverage;

while (beverage < 0 || beverage > 4)
{
    cout << "Valid choices are 0 - 4. Please re-enter: ";
    cin >> beverage;
}

```

What type of invalid data does this code test for? If `beverage` is an integer variable, what happens if the user enters the character '\$' or the float 2.9?

### The `do-while` Loop

The `while` loop is a **pre-test** or **top test** loop. Since we test the expression before entering the loop, if the test expression in the `while` loop is initially false, then no iterations of the loop will be executed. If the programmer wants the loop to be executed at least once, then a **post-test** or **bottom test** loop should be used. C++ provides the **do-while** loop for this purpose. A `do-while` loop is similar to a `while` loop except that the statements inside the loop body are executed *before*

the expression is tested. The format for a single statement in the loop body is the following:

```
do
    statement;
while (expression);
```

Note that the statement must be executed once even if the expression is false. To see the difference between these two loops consider the code

```
int num1 = 5;
int num2 = 7;

while (num2 < num1)
{
    num1 = num1 + 1;
    num2 = num2 - 1;
}
```

Here the statements `num1 = num1 + 1` and `num2 = num2 - 1` are never executed since the test expression `num2 < num1` is initially false. However, we get a different result using a do-while loop:

```
int num1 = 5;
int num2 = 7;

do
{
    num1 = num1 + 1;
    num2 = num2 - 1;
} while (num2 < num1);
```

In this code the statements `num1 = num1 + 1` and `num2 = num2 - 1` are executed exactly once. At this point `num1 = 6` and `num2 = 6` so the expression `num2 < num1` is false. Consequently, the program exits the loop and moves to the next section of code. Also note that since we need a block of statements in the loop body, curly braces must be placed around the statements. In Lab 5.2 you will see how do- while loops can be useful for programs that involve a repeating menu.

## The `for` Loop

The **`for` loop** is often used for applications that require a counter. For example, suppose we want to find the average (mean) of the first  $n$  positive integers. By definition, this means that we need to add  $1 + 2 + 3 + \dots + n$  and then divide by  $n$ . Note this should just give us the value in the “middle” of the list  $1, 2, \dots, n$ . Since we know exactly how many times we are performing a sum, the `for` loop is the natural choice.

The syntax for the `for` loop is the following:

```
for (initialization; test; update)
{
    statement_1;
    statement_2;
    :
    statement_n;
}
```

Notice that there are three expressions inside the parentheses of the `for` statement, separated by semicolons.

1. The **initialization expression** is typically used to initialize a counter that must have a starting value. This is the first action performed by the loop and is done only once.
2. The **test expression**, as with the `while` and `do-while` loops, is used to control the execution of the loop. As long as the test expression is true, the body of the `for` loop repeats. The `for` loop is a pre-test loop which means that the test expression is evaluated before each iteration.
3. The **update expression** is executed at the end of each iteration. It typically increments or decrements the counter.

Now we are ready to add the first  $n$  positive integers and find their mean value.

#### *Sample Program 5.5:*

---

```
#include <iostream>
using namespace std;

int main()
{
    int value;
    int total = 0;
    int number;
    float mean;

    cout << "Please enter a positive integer" << endl;
    cin >> value;

    if (value > 0)
    {
        for (number = 1; number <= value; number++)
        {
            total = total + number;
        }
        // curly braces are optional since
        // there is only one statement

        mean = static_cast<float>(total) / value; // note the use of the typecast
        // operator

        cout << "The mean average of the first " << value
              << " positive integers is " << mean << endl;
    }
    else
        cout << "Invalid input - integer must be positive" << endl;

    return 0;
}
```

---

Note that the counter in the `for` loop of Sample Program 5.5 is `number`. It increments from 1 to `value` during execution. There are several other features of this code that also need to be addressed. First of all, why is the `typecast` operator needed to compute the mean? What do you think will happen if it is removed?



Finally, what would happen if we entered a float such as 2.99 instead of an integer? Lab 5.3 will demonstrate what happens in these cases.

## Nested Loops

Often programmers need to use a loop within a loop, or **nested loops**. Sample Program 5.6 below provides a simple example of a nested loop. This program finds the average number of hours per day spent programming by each student over a three-day week- end. The outer loop controls the number of students and the inner loop allows the user to enter the number of hours worked each of the three days for a given student. Note that the inner loop is executed three times for each iteration of the outer loop.

### *Sample Program 5.6:*

---

```
// This program finds the average time spent programming by a student each
// day over a three day period.

#include <iostream>
using namespace std;

int main()
{
    int numStudents;
    float numHours, total, average;
    int count1 = 0, count2 = 0;    // these are the counters for the loops

    cout << "This program will find the average number of hours a day"
         << " that each given student spent programming over a long weekend"
         << endl << endl;

    cout << "How many students are there ?" << endl << endl;
    cin >> numStudents;

    for (count1 = 1; count1 <= numStudents; count1++)
    {
        total = 0;
        for (count2 = 1; count2 <= 3; count2++)
        {
            cout << "Please enter the number of hours worked by student "
                 << count1 << " on day " << count2 << "." << endl;
            cin >> numHours;

            total = total + numHours;
        }
        average = total / 3;

        cout << endl;
        cout << "The average number of hours per day spent programming by"
             << " student " << count1 << " is " << average
             << endl << endl << endl;
    }
    return 0.
}
```

---

In Lab 5.4 you will be asked to modify this program to make it more flexible.

## Files

So far all our input has come from the keyboard and our output has gone to the monitor. Input, however, can come from files and output can go to files. To do either of these things we should add the `#include <fstream>` directive in the header to allow files to be created and accessed. A file containing data to be input to the computer should be defined as an `ifstream` data type and an output file should be defined as `ofstream`.

*Sample Program 5.7*

Suppose we have a data file called `grades.dat` that contains three grades, and we want to take those grades and output them to a file that we will call `finalgrade.out`. The following code shows how this can be done in C++.

---

```
#include <fstream> // This statement is needed to use files
using namespace std;

int main()
{
    float grade1, grade2, grade3; // This defines 3 float variables

    ifstream dataFile;            // This defines an input file stream.
                                   // dataFile is the "internal" name that is
                                   // used in the program for accessing the
                                   // data file.

    ofstream outFile;             // This defines an output file stream.
                                   // outFile is the "internal" name that is
                                   // used in the program for accessing the
                                   // output file.

    outFile << fixed << showpoint; // These can be used with output files as
                                   // well as with cout.

    dataFile.open("grades.dat");   // This ties the internal name, dataFile,
                                   // to the actual file, grades.dat.

    outFile.open("finalgrade.out"); // This ties the internal name, outFile, to
                                   // the actual file, finalgrade.out.

    dataFile >> grade1 >> grade2 // This reads the values from the input file
                                   >> grade3; // into the 3 variables.

    outFile << grade1 << endl;    // These 3 lines write the values stored in
    outFile << grade2 << endl;    // the 3 variables to the output file
    outFile << grade3 << endl;

    return 0;
}
```

---

## PRE-LAB WRITING ASSIGNMENT

---

### Fill-in-the-Blank Questions

1. A block of code that repeats forever is called \_\_\_\_\_.
2. To keep track of the number of times a particular loop is repeated, one can use a(n) \_\_\_\_\_.
3. An event controlled loop that is always executed at least once is the \_\_\_\_\_.
4. An event controlled loop that is not guaranteed to execute at least once is the \_\_\_\_\_.
5. In the conditional `if(++number < 9)`, the comparison `number < 9` is made \_\_\_\_\_ and `number` is incremented \_\_\_\_\_. (Choose first or second for each blank.)
6. In the conditional `if(number++ < 9)`, the comparison `number < 9` is made \_\_\_\_\_ and `number` is incremented \_\_\_\_\_. (Choose first or second for each blank.)
7. A loop within a loop is called a \_\_\_\_\_.
8. List the preprocessor directive that is used to allow data and output files to be used in the program. \_\_\_\_\_
9. To write out the first 12 positive integers and their cubes, one should use a(n) \_\_\_\_\_ loop.
10. A(n) \_\_\_\_\_ value is used to indicate the end of a list of values. It can be used to control a `while` loop.
11. In a nested loop the \_\_\_\_\_ loop goes through all of its iterations for each iteration of the \_\_\_\_\_ loop. (Choose inner or outer for each blank.)

## LESSON 5A

---

### LAB 5.1 Working with the `while` Loop

Bring in program `while.cpp` from the Lab 5 folder. (This is Sample Program 5.2 from the Pre-lab Reading Assignment). The code is shown below:

---

```
// PLACE YOUR NAME HERE

#include <iostream>
using namespace std;

int main()
{
    char letter = 'a';

    while (letter != 'x')
    {
        cout << "Please enter a letter" << endl;
        cin >> letter;
        cout << "The letter you entered is " << letter << endl;
    }

    return 0;
}
```

---

*Exercise 1:* This program is not user friendly. Run it a few times and explain why.

*Exercise 2:* Add to the code so that the program is more user friendly.

*Exercise 3:* How would this code affect the execution of the program if the `while` loop is replaced by a `do-while` loop? Try it and see.

Bring in program `sentinel.cpp` from the Lab 5 Folder. The code is shown below:

```
// This program illustrates the use of a sentinel in a while loop.
// The user is asked for monthly rainfall totals until a sentinel
// value of -1 is entered. Then the total rainfall is displayed.

// PLACE YOUR NAME HERE

#include <iostream>
using namespace std;

int main()
{
    // Fill in the code to define and initialize to 1 the variable month
    float total = 0, rain;

    cout << "Enter the total rainfall for month " << month << endl;
    cout << "Enter -1 when you are finished" << endl;
    // Fill in the code to read in the value for rain

    // Fill in the code to start a while loop that iterates
    // while rain does not equal -1
    {
        // Fill in the code to update total by adding it to rain
        // Fill in the code to increment month by one

        cout << "Enter the total rainfall in inches for month "
             << month << endl;
        cout << "Enter -1 when you are finished" << endl;
        // Fill in the code to read in the value for rain

    }

    if (month == 1)
        cout << "No data has been entered" << endl;

    else
        cout << "The total rainfall for the " << month-1
             << " months is " << total << " inches." << endl;

    return 0;
}
```

*Exercise 4:* Complete the program above by filling in the code described in the statements in bold so that it will perform the indicated task.

*Exercise 5:* Run the program several times with various input. Record your results. Are they correct? What happens if you enter -1 first? What happens if you enter only values of 0 for one or more months? Is there any numerical data that you should not enter?

*Exercise 6:* What is the purpose of the following code in the program above?

```
if (month == 1)
    cout << "No data has been entered" << endl;
```

## LAB 5.2 Working with the `do-while` Loop

Bring in the program `dowhile.cpp` from the Lab 5 folder. The code is shown below:

```
// This program displays a hot beverage menu and prompts the user to
// make a selection. A switch statement determines which item the user
// has chosen. A do-while loop repeats until the user selects item E
// from the menu.

// PLACE YOUR NAME HERE

#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
    // Fill in the code to define an integer variable called number,
    // a floating point variable called cost,
    // and a character variable called beverage

    bool validBeverage;

    cout << fixed << showpoint << setprecision(2);

    do
    {
        cout << endl << endl;
        cout << "Hot Beverage Menu" << endl << endl;

        cout << "A: Coffee          $1.00" << endl;
        cout << "B: Tea              $ .75" << endl;
        cout << "C: Hot Chocolate    $1.25" << endl;
        cout << "D: Cappuccino       $2.50" << endl << endl << endl;
    }
```

*continues*

```
cout << "Enter the beverage A,B,C, or D you desire" << endl;
cout << "Enter E to exit the program" << endl << endl;
// Fill in the code to read in beverage

switch(beverage)
{
case 'a':
case 'A':
case 'b':
case 'B':
case 'c':
case 'C':
case 'd':
case 'D': validBeverage = true;
           break;
default:  validBeverage = false;
}

if (validBeverage == true)
{
    cout << "How many cups would you like?" << endl;
    // Fill in the code to read in number
}

// Fill in the code to begin a switch statement
// that is controlled by beverage
{
case 'a':
case 'A': cost = number * 1.0;
           cout << "The total cost is $ " << cost << endl;
           break;
// Fill in the code to give the case for hot chocolate ($1.25 a cup)
// Fill in the code to give the case for tea ( $0.75 a cup)
// Fill in the code to give the case for cappuccino ($2.50 a cup)

case 'e':
case 'E': cout << " Please come again" << endl;
           break;
default: cout << // Fill in the code to write a message
           // indicating an invalid selection.
           cout << " Try again please" << endl;
}

} // Fill in the code to finish the do-while statement with the
// condition that beverage does not equal E or e.

// Fill in the appropriate return statement
}
```

---

*Exercise 1:* Fill in the indicated code to complete the above program. Then compile and run the program several times with various inputs. Try all the possible relevant cases and record your results.

*Exercise 2:* What do you think will happen if you do not enter A, B, C, D or E? Try running the program and inputting another letter.

*Exercise 3:* Replace the line

```
if (validBeverage == true)
```

with the line

```
if (validBeverage)
```

and run the program again. Are there any differences in the execution of the program? Why or why not?

## LESSON 5B

### LAB 5.3 Working with the `for` Loop

Bring in program `for.cpp` from the Lab 5 folder (this is Sample Program 5.5 from the Pre-lab Reading Assignment). This program has the user input a number  $n$  and then finds the mean of the first  $n$  positive integers. The code is shown below:

```
// This program has the user input a number n and then finds the
// mean of the first n positive integers

// PLACE YOUR NAME HERE

#include <iostream>
using namespace std;

int main()
{
    int value;          // value is some positive number n
    int total = 0;      // total holds the sum of the first n positive numbers
    int number;         // the amount of numbers
    float mean;         // the average of the first n positive numbers

    cout << "Please enter a positive integer" << endl;
    cin >> value;

    if (value > 0)
    {
        for (number = 1; number <= value; number++)
        {
            total = total + number;
        } // curly braces are optional since there is only one statement

        mean = static_cast<float>(total) / value; // note the use of the typecast
                                                    // operator here
        cout << "The mean average of the first " << value
              << " positive integers is " << mean << endl;
    }
}
```

*continues*

```

    }
    else
        cout << "Invalid input - integer must be positive" << endl;

    return 0;
}

```

---

*Exercise 1:* Why is the typecast operator needed to compute the mean in the statement `mean = static_cast<float>(total)/value;`? What do you think will happen if it is removed? Modify the code and try it. Record what happens. Make sure that you try both even and odd cases. Now put `static_cast<float>total` back in the program.

*Exercise 2:* What happens if you enter a float such as 2.99 instead of an integer for `value`? Try it and record the results.

*Exercise 3:* Modify the code so that it computes the mean of the consecutive positive integers  $n, n+1, n+2, \dots, m$ , where the user chooses  $n$  and  $m$ . For example, if the user picks 3 and 9, then the program should find the mean of 3, 4, 5, 6, 7, 8, and 9, which is 6.

## LAB 5.4 Nested Loops

Bring in program `nested.cpp` from the Lab 5 folder (this is Sample Program 5.6 from the Pre-lab Reading Assignment). The code is shown below:

---

```

// This program finds the average time spent programming by a student
// each day over a three day period.

// PLACE YOUR NAME HERE

#include <iostream>
using namespace std;

int main()
{
    int numStudents;
    float numHours, total, average;
    int student, day = 0;    // these are the counters for the loops

    cout << "This program will find the average number of hours a day"
        << " that a student spent programming over a long weekend\n\n";
    cout << "How many students are there ?" << endl << endl;
    cin >> numStudents;

    for(student = 1; student <= numStudents; student++)
    {
        total = 0;
        for(day = 1; day <= 3; day++)
        {
            cout << "Please enter the number of hours worked by student "
                << student << " on day " << day << "." << endl;
            cin >> numHours;

```

*continues*



```

        total = total + numHours;
    }

    average = total / 3;

    cout << endl;
    cout << "The average number of hours per day spent programming by "
        << "student " << student << " is " << average
        << endl << endl << endl;
    }

    return 0;
}

```

---

*Exercise 1:* Note that the inner loop of this program is always executed exactly three times—once for each day of the long weekend. Modify the code so that the inner loop iterates  $n$  times, where  $n$  is a positive integer input by the user. In other words, let the user decide how many days to consider just as they choose how many students to consider.

*Sample Run:*

This program will find the average number of hours a day that a student spent programming over a long weekend

How many students are there?

2

Enter the number of days in the long weekend

2

Please enter the number of hours worked by student 1 on day 1

4

Please enter the number of hours worked by student 1 on day 2

6

The average number of hours per day spent programming by student 1 is 5

Please enter the number of hours worked by student 2 on day 1

9

Please enter the number of hours worked by student 2 on day 2

13

The average number of hours per day spent programming by student 2 is 11

*Exercise 2:* Modify the program from Exercise 1 so that it also finds the average number of hours per day that a given student studies biology as well as programming. For each given student include two prompts, one for each subject. Have the program print out which subject the student, on average, spent the most time on.

## LAB 3.5 Reading and Writing to a File

Bring in `billfile.cpp` from the Lab 5 folder. The code is as follows:

---

```
// This program will read in the quantity of a particular item and its price.
// It will then print out the total price.
// The input will come from a data file and the output will go to
// an output file.

// PLACE YOUR NAME HERE

#include <fstream>
#include <iomanip>
using namespace std;

int main()
{
    ifstream dataIn;           // defines an input stream for a data file
    ofstream dataOut;          // defines an output stream for an output file
    int quantity;              // contains the amount of items purchased
    float itemPrice;           // contains the price of each item
    float totalBill;           // contains the total bill, i.e. the price of all items

    dataIn.open("transaction.dat");    // This opens the file.
    dataOut.open("bill.out");

    // Fill in the appropriate code in the blank below
    _____ << setprecision(2) << fixed << showpoint;    // formatted output

    // Fill in the input statement that brings in the
    // quantity and price of the item

    // Fill in the assignment statement that determines the total bill.

    // Fill in the output statement that prints the total bill, with a label,
    // to an output file

    return 0;
}
```

---

*Exercise 1:* Notice that this is an altered version of Lab 3.1. This program gets the information from a file and places the output to a file. You must create the data file. Your instructor will tell you how to create the data file and where to put it. Create a data file called `transaction.dat` that has the following information:

22  
10.98

*Exercise 2:* Fill in the blank and the statements that will read the data from the file and print the following to `bill.out`:

**The total bill is \$241.56**

## LAB 5.6 Student Generated Code Assignments

*Option 1:* Write a program that performs a survey tally on beverages. The program should prompt for the next person until a sentinel value of -1 is entered to terminate the program. Each person participating in the survey should choose their favorite beverage from the following list:

1. Coffee      2. Tea      3. Coke      4. Orange Juice

*Sample Run:*

Please input the favorite beverage of person #1: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

4

Please input the favorite beverage of person #2: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

1

Please input the favorite beverage of person #3: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

3

Please input the favorite beverage of person #4: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

1

Please input the favorite beverage of person #5: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

1

Please input the favorite beverage of person #6: Choose 1, 2, 3, or 4 from the above menu or -1 to exit the program

-1

The total number of people surveyed is 5. The results are as follows:

Beverage	Number of Votes
*****	
Coffee	3
Tea	0
Coke	1
Orange Juice	1

*Option 2:* Suppose Dave drops a watermelon off a high bridge and lets it fall until it hits the water. If we neglect air resistance, then the distance  $d$  in meters fallen by the watermelon after  $t$  seconds is  $d = 0.5 * g * t^2$ , where the acceleration of gravity  $g = 9.8$  meters/second<sup>2</sup>. Write a program that asks the user to input the number of seconds that the watermelon falls and the height  $h$  of the bridge above the water. The program should then calculate the distance fallen for each second from  $t = 0$  until the value of  $t$  input by the user. If the total distance fallen is greater than the height of the bridge, then the program should tell the user that the distance fallen is not valid.

*Sample Run 1:*

```

Please input the time of fall in seconds:
2
Please input the height of the bridge in meters:
100

Time Falling (seconds) Distance Fallen (meters)
*****
0                      0
1                      4.9
2                      19.6

```

*Sample Run 2:*

```

Please input the time of fall in seconds:
4
Please input the height of the bridge in meters:
50

Time Falling (seconds) Distance Fallen (meters)
*****
0                      0
1                      4.9
2                      19.6
3                      44.1
4                      78.4

```

Warning-Bad Data: The distance fallen exceeds the height of the bridge

*Option 3:* Write a program that prompts the user for the number of tellers at Nation's Bank in Hyatesville that worked each of the last three years. For each worker the program should ask for the number of days out sick for each of the last three years. The output should provide the number of tellers and the total number of days missed by all the tellers over the last three years. See the sample output below.

*Sample Run:*

```

How many tellers worked at Nation's Bank during each of the last three years ?
2
How many days was teller 1 out sick during year 1 ?
5
How many days was teller 1 out sick during year 2 ?
8
How many days was teller 1 out sick during year 3 ?
2
How many days was teller 1 out sick during year 1 ?
1
How many days was teller 1 out sick during year 2 ?
0
How many days was teller 1 out sick during year 3 ?
3

```

The 2 tellers were out sick for a total of 19 days during the last 3 years.