Session 2

Student Name _____

Other Identification _____

Data Types and the while Statement

The goals of this laboratory session are to:

1. Introduce three of the primitive data types in C++.
2. Discuss some of the elementary operations that can be performed on these types.
3. Introduce the while control statement.

Your instructor will tell you which of the proposed experiments you are to perform.

In preparation for this laboratory session, you should read Chapter One of *Computer Science: An Overview.*

Primitive Data Types

Intuitively, something is different between data consisting of numeric values (such as food prices at McDonald's or the number of tacos you can get for a buck at Taco Bell) and data consisting of strings of characters (such as the names of your best friends or the lyrics to your favorite song). In one case we can meaningfully apply the mathematical operations of multiplication and division, while in the other we cannot. In programming terminology we say that the two groups of data are of different type.

The C++ programming language accounts for several elementary types of data. All other types of data are constructed as conglomerates of these primitive types. Our goal for now is to learn the fundamentals of three basic types: integer, real, and character.

The integer data type encompasses the counting numbers and their negatives:

. . . -4, -3, -2, -1, 0, 1, 2, 3, 4, . . .

In contrast, the data type real includes numeric values with fractional components such as 1.5, 10.999, and -2.53. As explained in *Computer Science: An Overview*, objects of type integer are normally stored within a machine's memory using two's complement notation, whereas objects of type real are normally stored using floating-point notation.

The data type character accounts for data consisting of a single symbol such as a letter of the alphabet, a punctuation mark, or any special symbol, such as $, %, and @. Data items of type character are normally stored using ASCII code. In future laboratory sessions you will learn how words and sentences can be constructed as collections of data of type character.

Variables

A variable is a name used in a program to refer to an item of data. Such names, also called identifiers, consist of any combination of letters, digits, and underscores, as long as they don't begin with a digit. Thus, Ritz_bits, X25, and Didi_7 would be valid variable names, whereas 7eleven, num miles, and 409 would not be. Good programming practice promotes the use of identifiers that reflect their role within the program. For example, a variable referring to temperature values might be called temperature or degrees.

Before a variable name can be used in a program it must be declared; that is, the particular name along with its data type must be described. This is done via a variable declaration statement having the form

data_type variable_list;

where *data_type* indicates the type associated with the variables being declared and *variable_list* is a list of variable names separated by commas. Keywords are used to represent the data types. These words have strict, preassigned meanings and cannot be used for other purposes (such as names of variables) elsewhere in a program. The keywords that represent the data types of integer, real, and character are int, float, and char, respectively. Thus, the declarative statements

```
int    servings, fat, calories;
float sodium;
char   ingredients;
```

establish servings, fat, and calories as variables of type int; sodium as a variable of type float; and ingredients as a variable of type char. From the machine's point of view, this

program segment states that the program requires areas of memory for the storage of three values of type int, one value of type float, and one value of type char. Moreover, it informs the translator that these memory locations will be referenced in the remaining part of the program by the names servings, fat, calories, sodium, and ingredients, respectively.

The language C++ allows a value to be assigned to a variable at the time the variable is declared. Such initialization of variables is done by following the variable name by an equal sign (=) and the value to be assigned. Thus, the statements

```
int     x = 1,
          y = 2;
float z = 3.75;
char   sym = 'a';
```

not only establish x, y, z, and sym as variables of type int, int, float, and char, respectively, but assign the variables the starting values 1, 2, 3.75, and a. (Note that data of type character is enclosed with single quotation marks when it appears within a program.)

Reading and Writing Data of Primitive Types

When you communicate with the world outside of your C++ program you use something called an I/O stream (I/O is a commonly used abbreviation for Input/Output). An I/O stream is a sequence of characters that can be either directed into or out of your program. As you learned in the previous laboratory session, the object cout can be used to display messages on the monitor screen. cout is the stream that is associated with the screen output device. Recall that the operator << was used to display information on the screen. You can also use the << operator to have cout display values that are stored in variables in your program. For example,

```
cout << "One gallon of ice cream = " << scps_gal << " scoops.\n";
```

If the variable scps_gal is assigned the value 103, then the statement causes the message

```
One gallon of ice cream = 103 scoops.
```

to appear on the monitor screen.

We can use the << operator multiple times in the same command to display the value of more than one variable. For example, if the variable v (of type char) is assigned the value A and the variable p (of type int) is assigned the value 75, the statement

```
cout << "U.S.RDA: vitamin " << v << " = " << p << " percent.\n";
```

produces the line

```
U.S.RDA: vitamin A = 75 percent.
```

Finally, note that the combination \n represents the "new line" marker so that

```
cout << "Over the river\nAnd through the woods\n";
```

produces the output

```
Over the river
And through the woods
```

One way of assigning a value to a variable is to use the predefined object cin, which can be thought of as the complement of the cout object. Whereas the latter object can be used to display data on the screen, cin can be used to receive data from the keyboard. cin is the stream that is associated with the keyboard input device.

Recall that the << operator directs information from your program onto the screen output stream cout. Similarly, the >> operator directs information from the keyboard input stream cin. Following the >> operator are parameters indicating where the data retrieved should be stored. For example, consider the statements

```
cout << "How many scoops per gallon?\n";
cin >> scps_gal;
```

The first line prompts the user for a response by printing a message on the monitor screen. The second line retrieves the response from the keyboard and stores it in the variable scps_gal. We can use the >> operator more than one time in the same statement to retrieve several data items, which may be of different type For example, if sym is a variable of type char and num is a variable of type int, then the statement

```
cin >> sym >> num;
```

would tell cin to read two pieces of information from the keyboard and assign the first to the variables sym and the second to num. How does cin determine which information should be stored in sym and which in num? Users must use whitespace (such as a space or an end-of-line) to separate the values as they are entered. You will be experiementing with this later in this session.

Observe that cin changes the value of the variables scps_gal, sym, and num in the preceding examples. Then cin copies new values into the locations that are represented by these variable names. The output stream cout does not change the value of the variables is displays on the screen.

The object cin can be a little tricky to work with, so let us take a moment to consider a few subtle details about its actions. When cin is used, characters representing the keys typed at the keyboard are collected in the input stream until a new line marker (generated by typing the ENTER key[1]) appears in the input stream. At this time, cin processes the contents of the stream by putting values into variables as dictated by the use of the >> operator. If all of the >> operators used can be satisfied by the data in the input stream, cin makes the appropriate variable assignments and terminates its activities; otherwise, cin must wait for more data to be entered at the keyboard, followed by another new line marker.

If cin is able to satisfy its >> operators without consuming the entire input stream, the unused portion of the stream remains and will be found in the input stream when cin is called again. This unused portion of the stream will include the new line marker that terminated the previous input. Thus, when cin is executed again, the input stream will already contain a new line marker, and cin will immediately process the input stream left from the previous call rather than wait for new input from the keyboard. Experiment 2.2

1. On some systems this is labeled the RETURN key. If this is the case on your system, simply make the appropriate translation when this manual refers to the ENTER key.

investigates the consequences of this process.

For now, suppose that age is a variable of type int and sym is a variable of type char. If the keys 1, 2, 5, and ETNER are typed in response to the command

```
cin >> age >> sym;
```

then cin will assign the value 125 to the variable age and your program will seem to be stuck as cin waits for a character to be entered. cin uses whitespace characters such as space and ENTER to separate inputs, so it is tricky to use cin to receive whitespace characters. Because of such technicalities, many C++ programmers prefer to write their own routines for reading data from the keyboard when robustness is important. We will see how this can be done in later laboratory sessions.

Experiment 2.1

Step 1. Execute the following program (CP02E01), and record the results below.

```
#include <iostream.h>

void main(void)
{
 int num = 2;
 char sym_1 = 'c',
         sym_2 = 'i',
         sym_3 = 'n';

 cout<<"Give me "<< num << " scoops of \n";
 cout<<sym_1<<"ho"<<sym_1<<"olate ";
 cout<<sym_1<<"ho<<sym_1<<"olate "<<sym_1<<"hip ";
 cout<<sym_2<<sym_3<<" a waffle cone please!\n";
}
```

Step 2. Change the line

```
cout<<"Give me "<< num << " scoops of \n";
```

in the program above to

```
cout<<"Give me "<< num << " scoops of" << endl;
```

Execute the modified program and record the results.

_____

_____

_____

_____

_____

_____

Step 3. Based on your results, what does endl do?

_____

_____

_____

_____

_____

_____

Experiment 2.2

Step 1. Execute the following program (CP02E02). In response to the first request, type an a followed by W. In response to the second request, type bcd, followed by W. Record the results.

```
#include <iostream.h>

void main(void)

{
 char a, b, c;

 cout << "Enter a character.\n";
 cin >> a;
 cout << "\nThe character entered was " < a << ".\n";
 cout << "Enter three more characters." << endl;
 cin >> a >> b >> c;
 cout << "\nThe characters entered were " << a << b << c ".\n";
}
```

Step 2. Explain the results obtained in Step 1. In particular, what was left in the input stream after executing cin the first time?

Step 3. Execute the program again. This time answer the first request by typing efgh followed by W. Record and explain the results.

Experiment 2.3

Step 1. Execute the following program (CP02E03). In response to the first request, type two integers followed by the ENTER key. In response to the second request, type two integers followed by the ENTER key. Record and explain the results.

```
#include <iostream.h>

void main(void)
{
 int a, b, c;

 cout << "Enter an integer.\n";
 cin >> a;
 cout << "\nThe value entered was " << a << endl;
 cout << "Enter two integers.\n";
 cin >> b >> c;
 cout << "\nThe values entered were "<<b<<" and "<<c << endl;
}
```

Step 2. Execute the program in Step 1 again. In response to the first request, type a single integer followed by the ENTER key. In response to the second request, type a single integer, followed by the ENTER key, followed by a single integer, followed by the ENTER key. Record and explain the results.

Elementary Operations and the Assignment Statement

Another means of assigning values to variables is through the use of an assignment statement. The statement

```
pizza = 14.50;
```

requests that the value 14.50 be assigned to the variable pizza using floating-point notation. In general, the assignment statement is used to assign the value found on the right of the = symbol to the variable found on the left of the = symbol.

The right side of an assignment statement can be any expression whose value is compatible with the variable on the left side. Thus, assuming that calories and fat are variables of type integer, the following assignment statements are valid.

```
calories = 270;
fat = calories;
```

In the case of numeric data, the traditional arithmetic operations can be used in an expression on the right side of the assignment statement to direct the computation of the value to be assigned. Here the symbols +, -, *, and / are used to represent addition, subtraction, multiplication, and division, respectively. Thus, the statement

```
pounds = calories * fat;
```

assigns the product of calories and fat to pounds.

Parentheses can be used to clarify the order in which the expression on the right side of an assignment statement is to be evaluated. Thus,

```
cal_serving = (fat_calories + sugar_calories) / servings;
```

would correctly assign the true value to calories per serving (cal_serving), whereas

```
cal_serving =  fat_calories + sugar_calories / servings;
```

would not.

A common operation is that of incrementing a value as accomplished by the statement

```
x = x + 1;
```

where x is an integer variable. C++ provides a shorthand notation for this assignment statement. It has the form

```
x++;
```

which consists of the name of the variable to be incremented followed by two plus signs. A similar expression

```
x--;
```

is a shorthand notation for the statement

```
x = x - 1;
```

that decrements the value of x by one.

Expressions such as x++ and x-- can be used in more complex expressions such as in the assignment statement

```
y = 5 + x++;
```

Here, y is assigned the result of adding 5 to x and then x is incremented by 1. In particular, the combination

```
x = 2;
y = 5 + x++;
```

would result in y being assigned the value 7 and x being assigned 3. Note that the value

of x is not incremented until its original value has already been used in the computation. If we had wanted the incremented value to be used in the computation, we would have used the expression ++x rather than x++. In short, ++x means to increment the value of x and then use this new value in the remaining computation, whereas x++ means to increment the value of x after its original value has been used. Thus, the sequence

```
x = 2;
y = 5 + ++x;
```

would result in y being assigned 8 and x being assigned 3. The expression x-- has a similar variation.

The C++ language recognizes a close relationship between data of type integer and character. Indeed, it is often convenient to think of the bit pattern representing a symbol (type character) as an integer value. (The printable characters in ASCII are represented by the integer values 32 through 126.) In particular, if sym is of type char and num is of type int, then statements such as

```
sym = 99;
num = 'a';
```

and

```
sym++;
```

are permitted (although not always considered good programming practice). Assuming that characters are represented in ASCII, the first of these statements assigns the character c to the variable sym since c is represented by 99 in ASCII. The second statement assigns the value 97 to the variable num since the character a is represented by 97 in ASCII. The third statement is often used to convert a letter in the alphabet to the next sequential letter (except for the letter z).

Experiment 2.4

Step 1. Execute the following program (CP02E04) and record the results.

```
#include <iostream.h>
void main(void)
{
 int x, y;
 cout << "x = " << x << endl;
 cout << "y = " << y << endl;
 y = 25;
 cout << "x = " << x << endl;
 cout << "y = " << y << endl;
}
```

Step 2. Explain the results in Step 1. (What can be said about the value of a variable that has not been assigned a value?)

Experiment 2.5

Step 1. Execute the following program (CP02E05), and record the results.

```
#include <iostream.h>

void main(void)
{
 int x, y, z;

 x = y = z = 5 + 3;
 cout << "x = " << x << endl;
 cout << "y = " << y << endl;
 cout << "z = " << z << endl;
}
```

Step 2. Change the assignment statement in Step 1 to read as follows:

```
x = 1 + y = 5 + z = 7;
```

Execute the modified program, and record the results.

Step 3. Explain the results in the preceding steps.

Experiment 2.6

Step 1. Execute the following program (CP02E06), and record the results.

```
#include <iostream.h>

void main(void)
{
 int integer_result;
 float float_result;

 integer_result = 7/3;
 float_result = 7/3;
 cout << integer_result << endl;
 cout << float_result << endl;
 integer_result = 12.6/3;
 float_result = 12.6/3;
 cout << integer_result << endl;
 cout << float_result << endl;
}
```

Step 2. Explain the results and the usage of the division operator (/).

Experiment 2.7

Step 1. Execute the following program (CP02E07), and record the results.

```
#include <iostream.h>
void main(void)
{
 int a, b, c;
 a = 7 % 3;
 b = 8 % 3;
 c = 8 % 4;
 cout << a << " " << b << " " << c << endl;
}
```

Step 2. The operator % is called the modulus operator. What does it do?

Step 3. Change the statement

```
a = 7 % 3;
```

to

```
a = 7.0 % 3.0;
```

and try to execute the modified program. What can you conclude?

Experiment 2.8

Step 1. Assuming that x and y are variables of type integer, what ambiguity exists in the following two statements? (Hint: If x starts with the value 3, will y in the first statement be assigned 9 or 11?)

```
y = x + x + x++;
y = x++ + x + x;
```

Step 2. Write a short program to discover what actually happens when these statements are executed. Record your program and your findings below.

Step 3. Change the program you wrote in Step 2 to investigate the expressions

```
x + x + ++x
++x + x + x
x-- + x + x
```

and

```
x + x + x--
```

Record your findings and explain them.

The while Control Statement

Normally, instructions within a function in a C++ program are executed in the order in which they appear. While this may be sufficient for simple programs, it is not flexible enough for more complex tasks. Thus, C++ provides a variety of control statements for redirecting the flow of execution. One of these is the while control statement.

The while control statement is used to execute a certain block of code repetitively. The general form of the while statement is

```
while (condition)
{
        body
}
```

where *body* is a statement or a sequence of statements. If *body* consists of only one statement, then the opening and closing braces of the while statement can be omitted.

Execution of the while statement begins with testing the *condition*. If true, the statement or statements within the *body* will be executed, and the *condition* will be tested once again. This cyclic process continues until the *condition* is found to be false, at which time control skips to the statements following the closing brace of the while statement. Thus, the following program

```
#include <iostream.h>

void main(void)
{
  int i;

  i = 5;
  while (i > 0)
  {
    cout << "Recycle!\n";
    i--;
  }
}
```

produces

```
Recycle!
Recycle!
Recycle!
Recycle!
Recycle!
```

We will discuss the details of the *condition* part of a while statement in Session 5. For now we need merely note that the symbols <, >, == , and != are used to represent less than, greater than, equal to, and not equal to, respectively, and combined as in <= and >=

to represent less than or equal to and greater than or equal to.

Experiment 2.9

Step 1. The following program (CP02E09) is designed to print the integers 1 through 10. Execute the program and record the results.

```
#include <iostream.h>

void main(void)
{
  int i;

 i = 1;
 while (i < 10)
    {
      i++;
     cout << i << endl;
    }
}
```

_____

_____

_____

_____

_____

_____

Step 2. Explain the changes required to make the program perform as initially intended. Confirm your answer by making these changes and executing the corrected program.

_____

_____

_____

_____

_____

_____

Experiment 2.10

Step 1. Execute the following program(CP02E10), and record the results.

```
#include <iostream.h>
void main(void)
{
 char x = 'a';
 while (x < 'z')
    {
     x++;
     cout << x << endl;
    }
}
```

Step 2. Explain the results obtained in Step 1.

Post-Laboratory Problems

2.1. Write a program that neatly prints a table of powers—squares, cubes, and quartics. Print these powers for the integers from 0 to 100. The first few lines of the table might appear as follows:

integer	square	cube	quartic
0	0	0	0
1	1	1	1
2	4	8	16

2.2. Write a program that calculates the area of a right triangle whose dimensions are supplied by the user.

2.3. Write a program with the declarations

```
int x, y, w, z;
x = 2;
y = -3;
w = 7;
z = -10;
```

that computes and reports the results of each of the following arithmetic problems. According to your results, summarize the precedence rules for the mathematical operators /, %, -, +, *.

a. `x / y`

b. `w / y / x`

c. `z / y % x`

d. `x % - y * w`

e. `x % y`

f. `- z % w - y / x * 5 + 5`

g. `9 - x % ( 2 + y )`

2.4. Write a program that computes the sum of the integers from 1 up to an integer provided by the program's user. For example, if the user types 10, the program should reply 55 because 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55.

2.5. Write a "Guess what number I'm thinking of..." game. Allow the player a maximum of five guesses, and if he or she hasn't guessed the number by then, print a losing message. If the player guesses the target number, print a winning message.

2.6. Write a program that calculates the mean of a set of positive integers that are supplied by the user. Write the program so that a negative number terminates the input string of integers. Thus, the user can supply a different number of integers each time the program is executed.

2.7. Write a program that asks the user for an integer and then displays all of the positive prime numbers that are equal to or less than that supplied integer. For example, if the user's number is 12, the program should print 11, 7, 5, 3, 2, 1.

2.8. In this laboratory session you learned a shorthand notation for incrementing and decrementing variables. A similar shorthand consists of an operation followed by an assignment symbol such as *= or +=. Execute the program below and, based on the results, state what += appears to represent. Then, experiment with other

arithmetic operations to confirm your hypothesis. Present your findings in an organized manner. To extend your investigation, experiment with such expressions as x *= y + z.

```
#include <iostream.h>

void main(void)
{
 int x;

 x = 5;
 cout << x << endl;
 x += 3;
 cout << x << endl;
 x += 3;
 cout << x << endl;
}
```