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 Sections 1.1 through 1.7 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 mathematical operations such as addition and subtraction, 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:

\[ \ldots -4, -3, -2, -1, 0, 1, 2, 3, 4, \ldots \]

In contrast, the data type real includes numeric values with fractional components 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 C++ applications. In future 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* 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 stated. 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. (It is common for C++ programmers to use these terms to refer the corresponding data type. For example, we may speak of the type *int* rather than integer and *float* rather than real.) 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

```cpp
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, consider the second line in the sequence

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

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

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

to appear on the monitor screen.

The `<<` operator can be used multiple times in a single statement to display multiple items of data. Thus, assuming that the variable `v`, of type `char`, is assigned the character A and the variable `p`, of type `int`, is assigned the value 75, the statement

```cpp
cout << "U.S.RDA: vitamin " << v << " = " << p << " percent."
```

produces the line

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

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

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

produces the output

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

Whereas the `<<` operator is used to send data to the monitor screen via output stream `cout`, the `>>` operator is used to receive data from the keyboard. The input stream from the keyboard is known as `cin`. Thus the statement

```cpp
cin >> scps_gal;
```
retrieves the value typed at the keyboard and stores it in the variable scps_gal. (Notice that the object cin determines the type of the variable scps_gal and in this case converts the string of characters into the proper numeric encoding of the value.)

We can use the >> operator multiple times in a single statement to receive several values from the keyboard. For example, if sym is of type char and num is of type int, then

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

would tell cin to read two pieces of information from the keyboard and assign the first to the variable sym (encoded using floating-point notation) and the second to num (encoded using two's complement notation).

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) 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 activities; otherwise, cin must wait for more data to be entered at the keyboard, followed by another new line marker. Thus, the statement

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

will cause the program's execution to pause until cin has assigned values to both variables.

If cin is able to satisfy its >> operators without consuming the entire input stream (if more values are typed on the line than there are variables to be assigned), 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 contain a new line marker already, and therefore cin will immediately process the leftover input stream remaining from the previous call rather than wait for new input from the keyboard. Experiment 2.2 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 Enter are typed in response to the command

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

then cin will assign the value 125 to the variable age and the program's execution will seem to be stuck as cin waits for another character to be entered.

**Experiment 2.1**

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

```cpp
#include <iostream.h>
int main()
{ 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";
  return 0;
}
```
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 the character a followed by the Enter key. In response to the second request, type bcd, followed by the Enter key. Record the results.

```cpp
#include <iostream.h>
int main()
{
    char a, b, c;
    cout << "Enter a character.\n";
    cin >> a;
    cout << "The character entered was " << a << "\n";
    cout << "Enter three more characters." << endl;
    cin >> a >> b >> c;
    cout << "The characters entered were " << a << b << c << "\n";
    return 0;
}
```

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.

```cpp
#include <iostream.h>
int main()
{
    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;
    return 0;
}
```

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

In addition to using `cin`, values can be assigned to variables by means of assignment statements. The statement

```c
pizza = 14.50;
```

requests that the floating-point value 14.50 be assigned to the variable `pizza`. 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.

```c
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

```c
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,

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

would correctly assign the true value to `cal_serving`, whereas

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

would not.

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

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

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

```c
x++;
```

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

```c
x--;
```

is a shorthand notation for the statement

```c
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

```c
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

```c
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

\[
\begin{align*}
x &= 2; \\
y &= 5 + ++x;
\end{align*}
\]

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 \( \text{sym} \) is of type \( \text{char} \) and \( \text{num} \) is of type \( \text{int} \), then statements such as

\[
\begin{align*}
\text{sym} &= 99; \\
\text{num} &= 'a';
\end{align*}
\]

and

\[
\text{sym}++;
\]

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 \( \text{sym} \) since the symbol \( c \) is represented by 99 in ASCII. The second statement assigns the value 97 to the variable \( \text{num} \) 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.

```cpp
#include <iostream.h>
int main()
{
    int x, y;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    y = 25;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    return 0;
}
```

...
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.

```cpp
#include <iostream.h>
int main()
{
    int x, y, z;
    x = y = z = 5 + 3;
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;
    return 0;
}
```

__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________

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

```cpp
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>
    int main()
    {
        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;
        return 0;
    }

__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________
__________________________________________________________________________________

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.

```cpp
#include <iostream.h>
int main()
{
    int a, b, c;
    a = 7 % 3;
    b = 8 % 3;
    c = 8 % 4;
    cout << a << " " << b << " " << c << endl;
    return 0;
}
```

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

Step 3. Change the statement

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

to

```cpp
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

\[
\begin{align*}
x + x + +++x \\
++x + x + x \\
x-- + x + x
\end{align*}
\]

and

\[
\begin{align*}
x + x + x--
\end{align*}
\]

Record your findings and explain them.
The while Control Statement

Normally, instructions 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

```cpp
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

```cpp
#include <iostream.h>
int main() {
    int i;
    i = 5;
    while (i > 0) {
        cout << "Recycle!\n";
        i--;
    }
    return 0;
}
```

produces
We will discuss the details of the condition part of a \texttt{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 $\leq$ and $\geq$ to represent less than or equal to and greater than or equal to.

\textbf{Experiment 2.9}

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

```cpp
#include <iostream.h>
int main()
{
    int i;
    i = 1;
    while (i < 10)
    {
        i++;
        cout << i << endl;
    }
    return 0;
}
```

\textbf{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.

```c++
#include <iostream.h>
int main()
{
    char x = 'a';
    while (x < 'z')
    {
        x++;
        cout << x << endl;
    }
    return 0;
}
```

**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 and cubes. Print these powers for the integers from 1 to 100. The first few lines of the table might appear as follows:

<table>
<thead>
<tr>
<th>integer</th>
<th>square</th>
<th>cube</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

2.2. Write a program that calculates the area of a circle whose radius is supplied by the user.

2.3. Write a program with the statements

```c
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 expressions.

```c
x / y
w / y / x
z / y % x
x % y
z % w - y / x * 5 + 5
9 - x % (2 + y)
```

According to your results, summarize the precedence rules for the mathematical operators /, %, -, +, and *.

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. 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.

```c
#include <iostream.h>
int main()
{
    int x = 5;
    cout << x << endl;
x += 3;
    cout << x << endl;
x += 3;
    cout << x << endl;
    return 0;
}
```