Session 1

Student Name ______________________

Other Identification __________________

Introduction to the Local Computer System and the Execution of C++ Programs

The purpose of this laboratory session is to introduce you to the computer system that you will use in the remaining sessions. The materials supplied during this period will teach you to:

1. Gain access to the machine.
2. Invoke the editor, create a simple text file, and save that file.
3. Retrieve the file mentioned above and modify it.
4. Enter a C++ program and execute it.

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

Gaining Access to the Machine

Those of you using this manual may be working with a wide variety of computer equipment. Some may use individual machines while others may use remote terminals connected to a central computing facility that serves many users at once. No matter what the machine, however, your first contact with the computer will be through its operating system—a program that coordinates the activities of the machine and performs tasks as directed by the machine's user (or users).

You will most likely communicate with the operating system through a keyboard, mouse, and monitor screen. Your instructor will explain how to establish contact with the operating system and will supply you with any customized information (such as user identification numbers or passwords) that you may need.

Through the machine's operating system you will be able to activate numerous auxiliary programs, some of which are designed to assist in program development. These are the programs you will learn to use in this laboratory session. In some cases these programs may be bundled as an integrated package; in other cases they may appear as individual programs whose services you must explicitly request. Your instructor will describe how your particular system operates.

High-Level Programming Languages

When we examine the internal design of a computer, we find that each individual operation which the machine performs is tedious in nature and contributes only a minute step toward completing any meaningful task. These operations form a language known as the machine language. It is a language in the sense that these operations constitute a set of commands that the circuitry within the machine "understands."

To develop a program in the form of a machine language would be a laborious task. Thus, programs are normally developed using a high-level language that more closely resembles a human's natural language. Once the program has been written in this form, it is translated into a low-level machine language form so that the machine can execute it.

Today there are literally hundreds of high-level programming languages, some of which you may have heard of—Ada, FORTRAN, BASIC, Pascal, C, Cobol. Some of these languages are designed for special types of programming tasks, while others are designed as general-purpose programming languages. This manual will teach you the rudiments of the general-purpose language known as C++.

The C++ programming language was developed at Bell Laboratories by Bjarne Stroustrup in the early 1980s and is built upon the programming language C (also developed at Bell Laboratories). C++ added support for object-oriented programming to C, retaining compatibility with existing C programs. Today, C++ enjoys wide popularity in both the business and academic communities. C++ is a very sophisticated and complex programming language. This manual, therefore, does not cover all of the features of C++.

C++ is an object-oriented programming language. Object-oriented programming is a way of organizing a program. For example, consider a video game. Following the object-oriented style, all of the program components dealing with a figure on the screen of the video game would be collected into a single program unit called an object. Such a unit would contain the location, direction, and speed of motion of the figure as well as the routines for interacting with other components of the video game. In turn, the organization of the entire program would be a collection of objects that interact with one another. An object-oriented programming language, such as C++, is designed to support such program organization. Objects are so central to C++ that we will meet one in the very first program we write. You will experiment with creating your own objects in later laboratory sessions.

The Program Preparation Process

The steps required to develop programs using the C++ language will depend on the computer installation being used. However, some features of the process are common to all systems.

As a first step, the programmer uses a program called an editor to type a C++ language version of the program being developed. This editor may be a stand-alone utility program or a part of an integrated software development package. Once the program has been typed, it is usually saved as a file in mass storage. This version of the program is known as the source program because it is the initial, or source, version of the program. It is this version to which you will return when alterations to the program are required.

A program in its source form cannot be executed by the computer; it must first be translated into the machine's own low-level language. This translation process is performed by a program known as a translator or compiler.

Your instructor will explain the details of how to type, save, translate, and finally execute programs using your particular computer system.

Experiment 1.1

Step 1. Use the editor to type the following lines. Then, save these lines in a file named Names. Summarize the editor features you use below.

```
A my name i$ Amy and I come from Alabama,
B my name i$ Bonita and I come from Brazil,
C my name i$ Carl and I come from Charlotte,
hello...hello...hello...hello...hello...
Z my name i$ Zak and I come from Zanzibar,
And I'm bringing you a lot of zebra$.
```

Step 2. Retrieve the file Names that was prepared in Step 1, make the following changes, and save the updated version. Summarize the editor features you use below.

a. Change all the $s to the letter s.

b. Change the word lot to load.

c. Insert a blank line between lines 2 and 3.

Step 3. Retrieve the file Names again and replace the line

```
hello...hello...hello...hello...hello...
```

with the following line. Summarize the editor features you use below.

```
My name is Mike and I come from Milwaukee,
```

Experiment 1.2

This experiment introduces the compiling process and the manner in which your compiler reports the errors that it finds. Error messages vary greatly from one system to another. Some systems are very helpful to the programmer, and some are not. Most C++ compilers pinpoint the line where the error has occurred and also display a short error message. It is then up to you to play the part of detective, going back into the source program and searching for the error yourself. Often one small error can spawn several misleading messages pointing to false errors. A common procedure when looking for errors (or "debugging") is trying to compile the program again after correcting the first real error that you find. It often happens that one correction will cause other misleading error messages to dissipate.

Step 1. The following document is a simple program in the C++ programming language. Using the editor in your particular software development environment, type the program as it appears here and save it for future reference. Be careful to include all of the punctuation marks and the braces.

```
// This is our first C++ program.

#include <iostream.h>

void main(void)
{
    cout << "What is your favorite\n";
    cout << "flavor of ice cream?\n";
}
```

Step 2. Retrieve the program from Step 1 and compile it. If the compiler finds errors (which would be typing errors), correct them and try again. Execute the final, correct program. What happens?

Step 3. In case you didn't have any typing errors on your first attempt, we'll introduce some now. Change the word cout in the source program to couts and try to compile the altered version. How does your compiler inform you of this error?

Step 4. Correct the error introduced in Step 3, and then remove the semicolon from the end of the line

```
cout << "What is your favorite\n";
```

Try to compile this altered version. How does your compiler respond?

____

____

____

____

____

Step 5. Correct the error introduced in Step 4, and then remove the closing brace } at the end of the program. Try to compile this altered version. How does your compiler respond?

____

____

____

____

A Simple C++ Program

Let us examine the program listed below, which is the program used in Experiment 1.2.

```
// This is our first C++ program.

#include <iostream.h>

void main(void)
{
 cout << "What is your favorite\n";
 cout << "flavor of ice cream?\n";
}
```

In addition to statements that will ultimately be translated into machine language, a C++ program can contain explanatory remarks for the aid of human readers (like your professor). These remarks, known as comments, are placed after the characters //. In turn, the C++ compiler ignores everything appearing on the same line after these special characters. In our example program, the first line

```
// This is our first C++ program.
```

is a comment. Comments have a variety of uses. They can be inserted to clarify a section of a program that might otherwise be hard to understand. They can also be used to give information about the creation of a program such as the date created and by whom.

The statement #include <iostream.h> in our example program is an example of a preprocessing directive. These directives cause the source program to be modified before the compiling process begins and are signified by beginning with a # symbol. The directive in our example causes a copy of the standard header file named iostream.h to be inserted at the beginning of the source code when compilation occurs. This file contains information that the compiler will need to perform its task. In particular, our example refers to the special object cout through which information can be sent to the monitor screen. Our program does not contain the details of cout but merely communicates with the object by means of the operator <<. The file iostream.h contains the information needed by the C++ compiler to create the required link between our program and the cout object. This file is one of many found in a collection known as the system library.

Names of standard header files are enclosed by angle brackets, < and >, as in our example. Names of nonstandard header files, such as those you may write yourself, are enclosed in quotation marks, as in

```
#include "HomeMade.h"
```

A C++ program contains units called functions. (We will learn about functions shortly.) Every program contains at least one function called main. Execution of a C++ program always begins in the function main. That is, the function main represents the beginning of the program even though the function may appear much later in the written program. The line

```
void main(void)
```

in our example indicates the beginning of the function main. This opening line of a function is known as a function header. We'll learn more about function headers in later laboratory sessions. For now we note that such a header consists of three parts: a return type (in our example void), the name of the function (in our example main), and a parameter list (in our example void) surrounded by parentheses.

The actual "meat" of a function is placed between braces. Immediately following the opening brace is the declarative part of the function. It is here that terminology relating to that particular function is introduced. Our example program is so simple that it does not contain a declarative part. Instead, the function main in our example consists of only a procedural part—the part containing the instructions to be followed when the function is executed. The procedural part of a function always follows the declarative part.

Statements in a C++ program are terminated by semicolons. Although not mandatory, it is customary to place each statement on a separate line and to use indentation to help the reader identify related portions of the program.

Each statement in the procedural part of our example program uses the predefined object cout and the operator <<. The object cout is just one of many standard pieces of functionality that are so commonly used in programs that C++ provides them for your use in standard C++ libraries. You will learn more about cout in later laboratory sessions. For now we need merely note that the statement

```
cout << "What is your favorite\n";
```

causes the characters that are enclosed in quotation marks to be printed on the monitor screen. Thus, when executed, our example program will cause the two lines

```
What is your favorite
flavor of ice cream?
```

to appear on the screen.

Experiment 1.3

Step 1. Omit the following line

```
#include <iostream.h>
```

from the program in Experiment 1.2 and try to compile the modified version. Record what happens.

Step 2. Remove both cout statements from the modified program in Step 1 and try to compile the program. Explain the results.

Experiment 1.4

Step 1. Insert the following lines into the procedural part of the function main in Experiment 1.2. Explain how the output produced by each of these statements differs from the others.

```
cout << "Chocolate, butterscotch, strawberry, vanilla?\n";
cout << "Chocolate, butterscotch,\nstrawberry, vanilla?";
cout << "Chocolate, butterscotch\n\nstrawberry, vanilla?";
```

Step 2. What does the \n character combination mean?

Experiment 1.5

Insert the following lines into the procedural part of the function main in Experiment 1.2. What rule can you derive about the placement of comments?

```
cout << "Send money quick!\n";  // To Mom!
cout << "Send money quick!\n"  // To Mom! );
cout << "Send // To Mom! money quick!\n");
```

Experiment 1.6

Step 1. Insert the following lines into the procedural part of the function main in Experiment 1.2. What rule can you derive about printing sentences containing quotation marks?

```
cout << "Oh, I love to program in C++...\n";
cout << "Oh, I love to "program" in C++...\n";
cout << "Oh, I love to ""program"" in C++...\n";
cout << "Oh, I love to \"program\" in C++...\n";
```

Step 2. What is the meaning of the backslash mark?

Post-Laboratory Problems

1.1. Use the editing features you learned in this laboratory session to create a file containing the speech below. Then, modify the speech to use more modern terminology. Print both copies of the speech.

```
Friends, Romans, countrymen, lend me your ears;
I come to bury Caesar, not to praise him.
The evil that men do lives after them;
The good is oft interred with their bones;
So let it be with Caesar. The noble Brutus
Hath told you Caesar was ambitious;
If it were so, it was a grievous fault;
And grievously hath Caesar answer'd it.
```

1.2. Write a program that prints the message

```
My name is Hector, I am
a vector.
I am the subject of many a
physics lecture!
```

a. all on one line

b. on two lines

c. on eight lines

d. inside a box drawn with asterisks

1.3. Find the errors in the following program.

```
#  include iore.h

mian(}
(
 couts << \n"I like to write/n before I've read it.\n\n;
 cout << "Then, with my pen, I always edit."\n;
 Cout >> "But, with computer\s, now I type;
 cout <<("An never, ever get it right./n")
{
```

1.4. Write a program that displays five of your favorite one-liners from advertisements.

1.5. What would be the output of the following program?

```
#include <iostream.h>
void main(void)
{
 cout << "The Hound of " << "t" << "h" << "e" << " ";
 cout << "Baskervi" << "l" << "l" << "es\n";
 cout << "by Arthur " << "C" << "onan Doyle\n";
}
```

1.6. Write a C++ program that will print another C++ program. For example, write a program to print the program discussed in Experiment 1.2.

1.7. Write a program that creates a bar graph for the following data on various kinds of snacks.

potato chips	$ 2.59
vanilla wafers	$ 1.09
hot dogs	$ 1.19
animal crackers	$ .99
chocolate chip cookies	$ .75

Model your graph after the following example.

```
             45  |            **
                 |      **    **
 MILES       35  |      **    **
  PER            |      **    **
 GALLON      30  |      **    **
                 |      **  **  **
             15  |      **  **  **  **
                        --  --  --  --

                        C   F   B   V
                        O   U   I   A
                        M   L   K   N
                        P   L   E
                        A
                        C   S
                        T   I
                            Z
                            E
```