LESSON SET

1 Introduction to Programming and the Translation Process

PURPOSE

1. To become familiar with the login process and the C++ environment used in the lab
2. To understand the basics of program design and algorithm development
3. To learn, recognize and correct the three types of computer errors:

   syntax errors
   run time errors
   logic errors

4. To learn the basics of an editor and compiler and be able to compile and run existing programs
5. To enter code and run a simple program from scratch

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 1.1 through 1.4 in sequence. Your instructor will give further instructions as to grading and completion of the lab.

Contents	Pre-requisites	Approximate completion time	Page number	Check when done
Pre-lab Reading Assignment		20 min.	2	
Pre-lab Writing Assignment	Pre-lab reading	10 min.	6	
Lesson 1A				
Lab 1.1 Opening, Compiling and Running Your First Program	Pre-lab reading	20 min. (Including overview of local system)	7	
Lab 1.2 Compiling a Program with a Syntax Error	Familiarity with the environment Finished Lab 1.1	15 min.	7	

continues

Lab 1.3 Running a Program with a Run Time Error	Understanding of the three types of errors	15 min.	8
Lesson 1B			
Lab 1.4 Working with Logic Errors	Understanding of logic errors	15 min.	9
Lab 1.5 Writing Your First Program	Finished Labs 1.1 through 1.4	30 min.	11

PRE-LAB READING ASSIGNMENT

Computer Systems

A **computer system** consists of all the components (hardware and software) used to execute the desires of the computer user. **Hardware** is the electronic physical components that can retrieve, process and store data. It is generally broken down into five basic components:

Central Processing Unit (C.P.U.)	This is the unit where programs are executed. It consists of the **control unit**, which oversees the overall operation of program execution and the **A.L.U.** (Arithmetic/Logic Unit), which performs the mathematical and comparison operations.
Main Memory	The area where programs and data are stored for use by the CPU
Secondary Storage	The area where programs and data are filed (stored) for use at a later time
Input Devices	The devices used to get programs and data into the computer (e.g., a keyboard)
Output Devices	The devices used to get programs and data from the computer (e.g., a printer)

Software consists of a sequence of instructions given to perform some pre-defined task. These labs concentrate on the software portion of a computer system.

Introduction to Programming

A **computer program** is a series of instructions written in some computer language that performs a particular task. Many times beginning students concentrate solely on the language code; however, quality software is accomplished only after careful design that identifies the needs, data, process and anticipated outcomes. For this reason it is critical that students learn good design techniques before attempting to produce a quality program. Design is guided by an **algorithm**, which is a plan of attacking some problem. An algorithm is used for many aspects of tasks, whether a recipe for a cake, a guide to study for an exam or the specifications of a rocket engine.

Problem example: Develop an algorithm to find the average of five test grades.

An algorithm usually begins with a general broad statement of the problem.

From here we can further refine the statement by listing commands that will accomplish our goal.

Each box (called a node) may or may not be refined further depending on its clarity to the user. For example: **Find the Average** may be as refined as an experienced programmer needs to accomplish the task of finding an average; however, students learning how to compute averages for the first time may need more refinement about how to accomplish the goal. This refinement process continues until we have a listing of steps understandable to the user to accomplish the task. For example, **Find the Average** may be refined into the following two nodes.

Starting from left to right, a node that has no refinement becomes part of the algorithm. The actual algorithm (steps in solving the above program) is listed in bold.

Find the Average of Five Test Grades

Read in the Grades

Find the Average

Total = sum of 5 grades
Average = Total / 5

Write Out the Average

From this algorithm, a program can be written in C++.

Translation Process

Computers are strange in that they only understand a sequence of 1s and 0s. The following looks like nonsense to us but, in fact, is how the computer reads and executes everything that it does:

10010001111010101110010001110001000

Because computers only use two numbers (1 and 0), this is called **binary** code. can imagine how complicated programming would be if we had to learn this very complex language. That, in fact, was how programming was done many years ago; however, today we are fortunate to have what are called **high level languages** such as C++. These languages are geared more for human understanding and thus make the task of programming much easier. However, since the computer only understands low level binary code (often called machine code), there must be a translation process to convert these high level languages to machine code. This is often done by a **compiler**, which is a software package that translates high level

languages into machine code. Without it we could not run our programs. The figure below illustrates the role of the compiler.

The compiler translates source code into object code. The type of code is often reflected in the extension name of the file where it is located.

Example: We will write source (high level language) code in C++ and all our file names will end with `.cpp`, such as:

```
firstprogram.cpp    secondprogram.cpp
```

When those programs are compiled, a new file (object file) will be created that ends with `.obj`, such as:

```
firstprogram.obj    secondprogram.obj
```

The compiler also catches grammatical errors called **syntax errors** in the source code. Just like English, all computer languages have their own set of grammar rules that have to be obeyed. If we turned in a paper with a proper name (like John) not capitalized, we would be called to task by our teacher, and probably made to correct the mistake. The compiler does the same thing. If we have something that violates the grammatical rules of the language, the compiler will give us error messages. These have to be corrected and a grammar error free program must be submitted to the compiler before it translates the source code into machine language. In C++, for example, instructions end with a semicolon. The following would indicate a syntax error:

```
cout << "Hi there" << endl
```

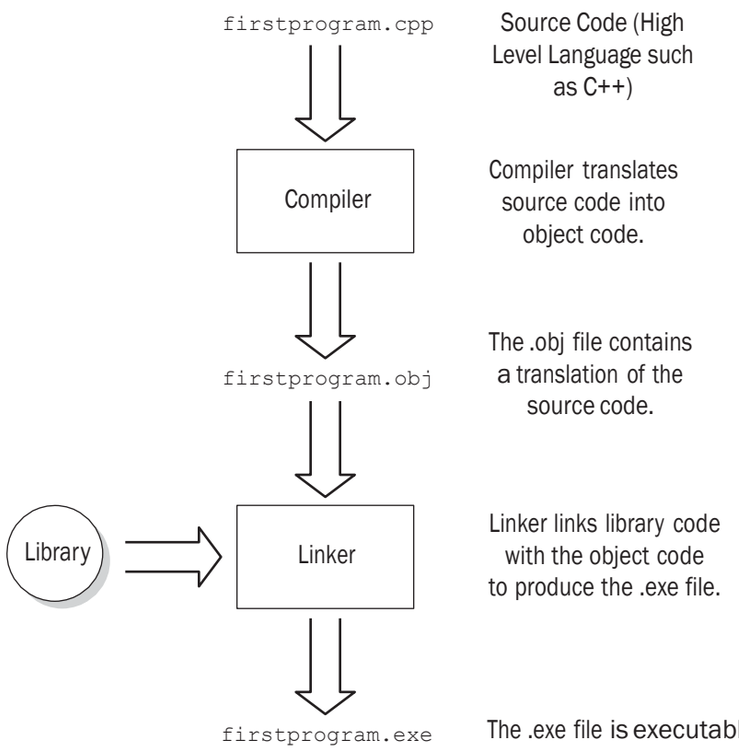
Since there is no semicolon at the end, the compiler would indicate an error, which must be corrected as follows:

```
cout << "Hi there" << endl;
```

After the compile process is completed, the computer must do one more thing before we have a copy of the machine code that is ready to be executed. Most programs are not entirely complete in and of themselves. They need other modules previously written that perform certain operations such as data input and output. Our programs need these attachments in order to run. This is the function of the **linking process**. Suppose you are writing a term paper on whales and would like a few library articles attached to your report. You would go to the library, get a copy of the articles (assuming it would be legal to do so), and attach them to your paper before turning it in. The **linker** does this to your program. It goes to a "software library" of programs and attaches the appropriate code to your program. This produces what is called the executable code, generated in a file that often ends with `.exe`.

Example: `firstprogram.exe    secondprogram.exe`

The following figure summarizes the translation process:



Once we have the executable code, the program is ready to be run. Hopefully it will run correctly and everything will be fine; however that is not always the case. During "run time", we may encounter a second kind of error called a **run time error**. This error occurs when we ask the computer to do something it cannot do. Look at the following sentence:

You are required to swim from Naples, Italy to New York in five minutes.

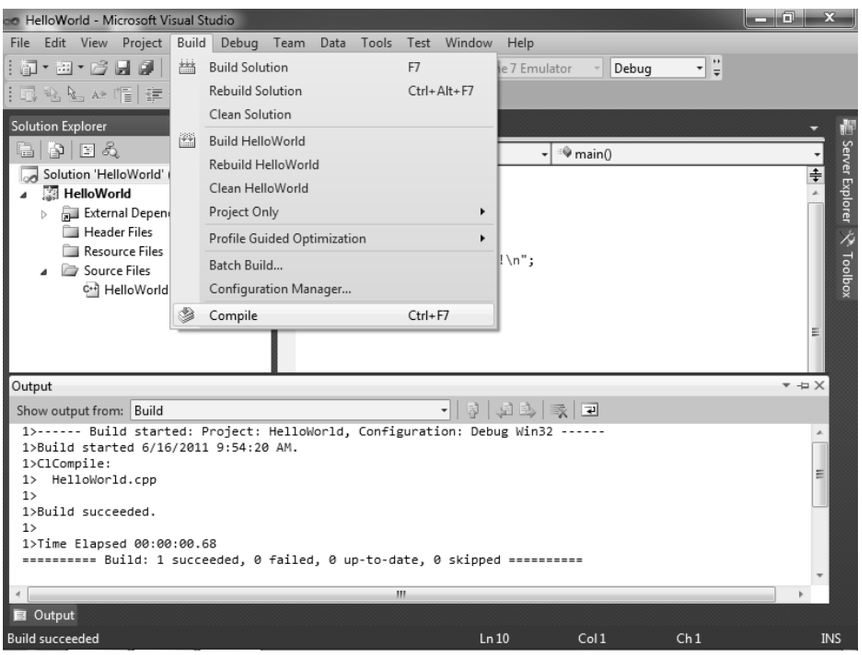
Although this statement is grammatically correct, it is asking someone to do the impossible. Just as we cannot break the laws of nature, the computer cannot violate the laws of mathematics and other binding restrictions. Asking the computer to divide by 0 is an example of a run time error. We get executable code; however, when the program tries to execute the command to divide by 0, the program will stop with a run time error. Run time errors, particularly in C++, are usually more challenging to find than syntax errors.

Once we run our program and get neither syntax nor run time errors, are we free to rejoice? Not exactly. Unfortunately, it is now that we may encounter the worst type of error: the dreaded **Logic error**. Whenever we ask the computer to do something, but mean for it to do something else, we have a logic error. Just as there needs to be a "meeting of the minds" between two people for meaningful communication to take place, there must be precise and clear instructions that generate our intentions to the computer. The computer only does what we ask it to do. It does not read our minds or our intentions! If we ask a group of people to cut down the tree when we really meant for them to trim the bush, we have a communication problem. They will do what we ask, but what we asked and what we wanted are two different things. The same is true for the computer. Asking it to multiply by 3 when we want something doubled is an example of a

logic error. Logic errors are the most difficult to find and correct because there are no error messages to help us locate the problem. A great deal of programming time is spent on solving logic errors.

Integrated Environments

An integrated development environment (IDE) is a software package that bundles an editor (used to write programs), a compiler (that translates programs) and a run time component into one system. For example, the figure below shows a screen from the Microsoft Visual C++ integrated environment.



Other systems may have these components separate which makes the process of running a program a little more difficult. You should also be aware of which Operating System you are using. An **Operating System** is the most important software on your computer. It is the "grand master" of programs that interfaces the computer with your requests. Your instructor will explain your particular system and C++ environment so that you will be able to develop, compile and run C++ programs on it.

PRE-LAB WRITING ASSIGNMENT

Fill-in-the-Blank Questions

1. Compilers detect _____errors.
2. Usually the most difficult errors to correct are the _____errors, since they are not detected in the compilation process.
3. Attaching other pre-written routines to your program is done by the _____process.
4. _____code is the machine code consisting of ones and zeroes that is read by the computer.
5. Dividing by zero is an example of a _____error.

Learn the Environment That You Are Working In

The following information may be obtained from your instructor.

1. What operating system are you using?
2. What C++ environment are you working in?
3. If you are not working in an integrated environment, what are the compile, run and edit commands that you will need?

LESSON 1A

Your instructor may assign either Appendix A or Appendix B depending on your environment. Appendix A is for labs using Visual C++ and Appendix B is for labs using UNIX. If you are using an environment other than these two, your instructor will give you instructions for this first lesson and ask you to complete Lab 1.1 below.

LAB 1.1 Opening, Compiling and Running Your First Program

Exercise 1: Logon to your system based on your professor's instructions.

Exercise 2: Bring in the `firstprog.cpp` program from the Lab 1 folder.

Exercise 3: Compile the program.

Exercise 4: Run the program and write what is printed on the screen.

The code of `firstprog.cpp` is as follows:

```
// This is the first program that just writes out a simple message
// Place your name here

#include <iostream>               // needed to perform C++ I/O
using namespace std;

int main ()

{

   cout << "Now is the time for all good men" <<  endl;
   cout << "To come to the aid of their party" <<  endl;

   return 0;

}
```

LAB 1.2 Compiling a Program with a Syntax Error

Exercise 1: Bring in program `semiprob.cpp` from the Lab 1 folder.

Exercise 2: Compile the program. Here we have our first example of the many syntax errors that you no doubt will encounter in this course. The error message you receive may be different depending on the system you are using, but the compiler insists that a semicolon is missing somewhere. Unfortunately, where the message indicates that the problem exists, and where the problem actually occurs may be two different places. To correct

the problem place a semicolon after the line `cout << "Today is a great day for Lab"`.

Most syntax errors are not as easy to spot and correct as this one.

Exercise 3: Re-compile the program and when you have no syntax errors, run the program and input 9 when asked. Record the output.

Exercise 4: Try running it with different numbers. Record your output.

Do you feel you are getting valid output?

The code of `semiprob.cpp` is as follows:

```
// This program demonstrates a compile error.

// Place your name here

#include <iostream>
using namespace std;

int main()

{
   int number;
   float total;


   cout << "Today is a great day for Lab"
   cout << endl << "Let's start off by typing a number of your choice" << endl;
   cin  >> number;

   total = number * 2;
   cout << total << " is twice the number you typed" << endl;

   return 0;

}
```

LAB 1.3 Running a Program with a Run Time Error

Exercise 1: Bring in program `runprob.cpp` from the Lab 1 folder.

Exercise 2: Compile the program. You should get no syntax errors.

Exercise 3: Run the program. You should now see the first of several run time errors. There was no syntax or grammatical error in the program; however, just like commanding someone to break a law of nature, the program is asking the computer to break a law of math by dividing by zero. It cannot be done. On some installations, you may see this as output that looks very strange. Correct this program by having the code divide by 2 instead of 0.

Exercise 4: Re-compile and run the program. Type 9 when asked for input. Record what is printed.

Exercise 5: Run the program using different values. Record the output. Do you feel that you are getting valid output?

The code of `runprob.cpp` is as follows:

```
// This program will take a number and divide it by 2.

// Place your name here


#include <iostream>
using namespace std;

int main()

{

    float number;
    int divider;

    divider = 0;

    cout << "Hi there" << endl;
    cout << "Please input a number and then hit return" << endl;
    cin  >> number;

    number = number / divider;

    cout << "Half of your number is " << number << endl;

    return 0;
}
```

LESSON 1B

LAB 1.4 Working with Logic Errors

Exercise 1: Bring in program `logicprob.cpp` from the Lab 1 folder. The code follows.

```
// This program takes two values from the user and then swaps them
// before printing the values.  The user will be prompted to enter
// both numbers.

// Place your name here

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

continues

```
int main()

{

    float firstNumber;
    float secondNumber;

    // Prompt user to enter the first number.

    cout << "Enter the first number" << endl;
    cout << "Then hit enter" << endl;
    cin  >> firstNumber;

    // Prompt user to enter the second number.

    cout << "Enter the second number" << endl;
    cout << "Then hit enter" << endl;
    cin  >> secondNumber;

    // Echo print the input.

    cout << endl << "You input the numbers as " << firstNumber
         << " and " << secondNumber << endl;

      // Now we will swap the values.

    firstNumber = secondNumber;
    secondNumber = firstNumber;

    // Output the values.

    cout  << "After swapping, the values of the two numbers are "
          << firstNumber << " and " << secondNumber << endl;

    return 0;

}
```

Exercise 2: Compile this program. You should get no syntax errors.

Exercise 3: Run the program. What is printed?

Exercise 4: This program has no syntax or run time errors, but it certainly has a logic error. This logic error may not be easy to find. Most logic errors create a challenge for the programmer. Your instructor may ask you not to worry about finding and correcting the problem at this time.

LAB 1.5 Writing Your First Program (Optional)

Exercise 1: Develop a design that leads to an algorithm and a program that will read in a number that represents the number of kilometers traveled. The output will convert this number to miles. 1 kilometer = 0.621 miles. Call this program `kilotomiles.cpp`.

Exercise 2: Compile the program. If you get compile errors, try to fix them and re-compile until your program is free of syntax errors.

Exercise 3: Run the program. Is your output what you expect from the input you gave? If not, try to find and correct the logic error and run the program again. Continue this process until you have a program that produces the correct result.