Lesson 1: Introduction to Programming

Computer Systems

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

Central Processing Unit (CPU) This is the unit where programs are executed. It consists of the **control unit,** which oversees the overall operation of program execution and the **ALU** (Arithmetic/Logic Unit), which performs the mathematical and comparison operations.

Main Memory -------- where programs and data are stored for use by the CPU
Secondary Storage -- where programs and data are filed (stored) for use at a later time.
Input Devices -------- devices used to get data into the computer (e.g., keyboard)
Output Devices ------ devices used to get data from the computer (e.g., printer)

Programming

Computers in this twenty-first century will be guided by new and innovative programs that will radically change every aspect of our lives. Medical diagnostics, legal research, business applications, weapons development and even sermon preparations are but a few areas that will be dramatically altered as software and the hardware where they run become more sophisticated.

A computer **program** is a set of instructions that tell computers what to do. **Software** is the general term given to these instructions. **Programming** is the term given to the process of developing software.

Computer Languages

Computer languages, like human languages, are guided by a precise set of grammatical rules that must be strictly adhered. There are various levels of computer languages. Ultimately all programs are translated into a series of ones and zeros, for this is the only format that a computer understands. Despite all their sophistication, a computer basically goes to a location and determines one of two values (a one or a zero). Primitive programming was done as a series of ones and zeros and was labeled **machine language.** Machine code was very difficult to develop and understand. The sophisticated programs that we know today would not be possible had it not been for the development of **high-level languages**. These languages are geared more for human understanding and logical development. High-level programming languages allow the use of vocabulary that is common to human communication thus making the task of programming easier. Although easier, these programs must be translated into the machine code described above.

Compilers and **interpreters** are programs that make those translations.

Types of Computer Errors

Compilers and interpreters also detect and indicate **grammatical** and **syntax errors** whenever the language is used incorrectly. It is thus very important to learn the vocabulary and syntax rules for a particular language. The complete translation will not take place as long as there are grammatical errors. Once the program is free of such errors the translation will take place and give us an executable code ready to run.

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 would be an example of a run time error. We would 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 are usually more challenging to find than syntax errors.

Once we compile our program without compile-time or run-time errors, are we free to rejoice? Not necessarily. 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 someone to cut down a tree when we really meant for them to trim a bush, we have a communication problem. They will do what we ask, but what we ask and what we want may be two different things. The same is true for a computer. Asking a computer to multiply a number by 3, when we really want the number 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.

Procedural Programming

There are several basic approaches to writing computer programs. In this course we will consider two: procedural programming and object oriented programming. Let us first consider procedural programming.

Procedural programming allows the use of memory locations in the computer to be reserved for **variables:** storage locations containing values that can be altered during the course of the program. Such a memory location is given a name that reflects the nature of the data stored there. For example, *hourlyRate* may be the name of a location that holds the amount of pay a person is paid per hour. These values will also have **operations** performed on them. The variable *hourlyRate* and *numOfHours* may have the multiplication operation performed on them to create a new value, *totalPay*. This can be accomplished in a language such as Java by the following statement:

```
totalPay = hourlyRate * numOfHours;
```

Operations can be grouped together into logical units called **procedures**. The instructions to input an hourly rate and number of hours worked and to calculate the total pay could be placed in one procedure. Another procedure could determine the Federal withholding tax. **Call** or **invokes** are used to activate these procedures. Although Java is not a procedural language, it uses these concepts that will be explored later. The following is a simple code that demonstrates these concepts. The code is not a true language code but rather pseudo (artificial) code. It consists of three procedures: *Main*, *FindPay*, *FindFedTax*. The whole purpose of *Main* is to call the other two procedures. *FindPay* is a procedure that reads in the values of pay rate & hours worked and then calculates and displays the total pay. *FindFedTax* determines the federal tax based on a 30% tax rate. Both *FindPay* and *FindFedTax* either return or need the value of totalPay. This is the reason that *totalPay* is placed in parenthesis in the call and title of those procedures. *TotalPay* is an argument of the procedures. These concepts are dealt with at greater length in later lessons.

Example:

```
Main program
   FindPay(totalPay);
   FindFedTax(totalPay);

FindPay(totalPay)
     Read(hourlyRate);          // This reads hourly Rate into the hourlyRate variable location
     Read(numOfHours);          // This reads the number of hours worked into numOfHours variable
     totalPay = hourlyRate * numOfHours;       // This calculates total pay and stores it at totalPay
     Write(totalPay);           // This displays (somewhere) the total pay

FindFedTax(totalPay)
     fedTax = totalPay * .30;
     Write(fedTax);
```

Lesson 1 Summary Outline

I. Computer Systems

A. **Central Processing Unit** (CPU) where programs are executed. It consists of the control unit and the Arithmetic Logic Unit (ALU).
B. **Main Memory** where programs and data are stored for use by the CPU.
C. **Secondary Memory** where programs and data are filed for use at a later time
D. **Input Devices** devices used to get programs and data into the computer
E. **Output Devices** devices used to send programs and data out from the computer

II. Programming Languages

A. Computer Languages
    A. **Machine Languages** (ones and zeros)
    B. **High Level Languages** (human oriented languages)
B. **Compilers** translate high level languages into machine code

III. Computer Errors

A. **Grammatical and Syntax Errors** errors exposed at compile time
B. **Run time Errors** errors exposed during program execution (dividing by 0)
C. **Logic Errors**

IV Procedural Programming

A. **Variables** memory locations that contain values that can change
B. **Procedures** a group of related instructions that perform a certain task
C. **Operations** mathematical or logical operations performed on values
D. **Calls** statements that activate procedures