Study Unit 1: Introduction to Programming

Introduction

Programming is the art of writing computer programs. A program in this case is a set of instructions to be executed by the computer. Programming involves breaking down a task into small steps. In programming, we study the problem at hand, look for the solution to the problem and implement that solution using a particular programming language.

Learning Outcomes of Study Unit 1

Upon completion of this study unit, you should be able to

1.1 Explain the concept of programming and programs

1.2 List and discuss the different programming languages

1.3 Explain the concept of syntax, semantics and programming errors

1.4 Understand the program development approach

1.1 Concepts of Programming

1.1.1 What is programming

Programming is the art of writing computer programs. A program in this case is a set of instructions to be executed by the computer. Programming involves breaking down a task into small steps. In programming, we study the problem at hand, look for the solution to the problem and implement that solution using a particular programming language.

Box 1.1: Definition of Programming

Programming is the art of writing computer programs.

A program in this case is a set of instructions to be executed by the computer

1.2 Programming Languages

A programming language is a language that a computer can understand and provides a programmer an environment to write and execute programs in it. A number of programming languages exist

but the choice mainly depends on the nature of the problem at hand and the programmer's ability to use the language.

There has been a great revolution in programming languages right from the time when programming started. The first generation of programming languages involved the use of machine language, the second involved the use of assembly language, the third generation involved high-level programming languages like C, C++, Java e.t.c, the forth generation involved languages mainly used for database manipulation like Structured Query Language (SQL) and the fifth generation involved languages mainly used for artificial intelligence and neural networks like ProLog and LISP

Box 1.2: Definition of a Programming Language

A programming language is a language that a computer can understand and provides a programmer an environment to write and execute programs in it.

1.2.1 First Generation: Machine language

Machine language consists of binary sequences of 0's and 1's which represent an instruction in the control logic of a specific CPU type, often expressed in hexadecimal (base 16) for clarity. It is different for each machine (i.e. machine dependant).

All programs must be written in machine language (code) or translated into machine language before being executed by the machine.

For example. A program that adds over time pay to base pay and stores the result in gross pay could look like this one below.

```
.....
+1300042774
+1400593419
+1200274027
.....
```

1.2.2 Second Generation: Assembly language

As you can see from the above example, machine language instructions would be cumbersome for humans. That is why the assembly language came into programming to make life easier.

Assembly languages use short abbreviations (mnemonics) corresponding to machine language (e.g. JSR DECOUT instead of 0100111010000000) are used for instructions

– Example corresponding to the above machine code could be

```
.....
Load Basepay
Add OverPay
Store GrossPay
.....
```

1.2.3 Third Generation: High-level Languages (HLL)

Writing programs in Assembly language was a bit easy but these required many instructions to accomplish even the simplest tasks. HLL codes are similar to everyday English language which makes it easier for programmers to understand. HLLs are machine independent and must be interpreted or compiled.

HLLs use mathematical notations (translated via compilers)

E.g GrossPay = basepay + overTimePay

HLLs are meant to correspond (roughly) to the way humans describe solutions. They are more abstract than low level languages, and therefore are portable between machines

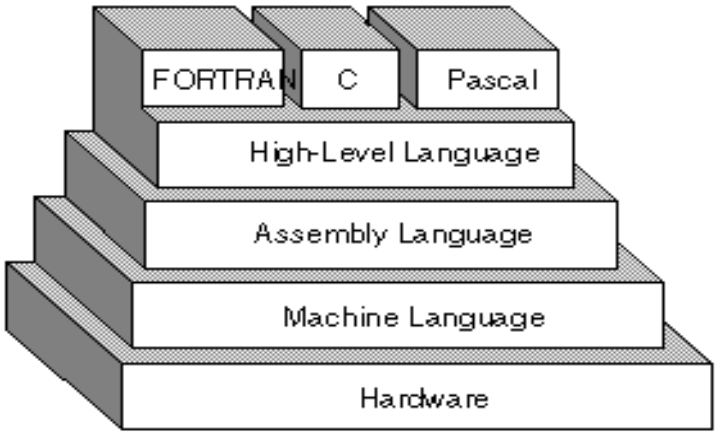
Compilers in this case are programs that translate high level language instructions into machine code for later execution.

There is also what we call Interpreters. They are similar to compilers; they interweave the translation and execution activities. They translate a small part of the source and execute them one by one. Interpreters are mainly used to interpret the source code for scripting languages

Often a single HLL instruction will be represented by several machine code instructions and HLLs are often tailored to specific kinds of problems. For example,

1. FORTRAN and Pascal are normally used for scientific calculation
2. COBOL – Common Business-Oriented Languages used for report generation, handling large amounts of business data
3. C, C++, Java, Visual Basic - general purpose problem solving languages.
4. SQL used for manipulating databases.

The figure below summarizes the relationship between Machine, Assembly and HL languages.



1.2.4 Fourth Generation languages (4GLs)

More than the previous levels of languages, 4GLs are similar to human languages and are application-specific. Often used with database systems; for example, SQL (Structured Query Language)

4GLs can be easily written with the use of simple, sentence-like commands, such as SELECT NAME FROM RECORDS WHERE NAME=“JOHN”;

Isn’t this pretty simple?

Other examples of 4GLs include the UNIX shell, PostScript, Mathematica e.t.c. They may include special facilities for automatic report generation or interaction with the database.

1.2.5 Classifications of Programming languages

Programming languages are classified based on what they are able to do. For example, we have classifications like Procedural languages, object-oriented languages and scripting languages e.t.c

a. Procedural Programming Languages

- Procedural programming is a programming paradigm based upon the concept of the modularity and scope of program code (i.e., the data viewing range of an executable code statement). A main procedural program is composed of one or more modules (also called packages or units), either coded by the same programmer or pre-coded by someone else and provided in a code library.
- Each module is composed of one or more subprograms (which may consist of procedures, functions, subroutines or methods, depending on programming language).
- Examples include, C, FORTRAN ,Ada, and ALGOL.

b. Object-Oriented Programming Languages

An object-oriented programming language is one that allows or encourages, to some degree, object-oriented programming methods. These languages include

- "pure" object-oriented languages such as Smalltalk, Eiffel and Ruby, which were designed specifically to facilitate - even enforce - object-oriented methods
- languages such as Java and Python, which are primarily designed for object-oriented programming but have some procedural elements; and
- languages such as C++, Fortran 2003, and Perl, which are historically procedural languages that have been extended with some object-oriented features.

c. Scripting Programming Languages

Scripting programming languages (commonly called scripting languages or script languages) are computer programming languages designed for "scripting" the operation of a computer. Early script languages were often called *batch languages* or *job control languages*.

Many scripting languages emerged as tools for executing one-off tasks, particularly in system administration. One way of looking at scripts is as "glue" that puts several components together; thus they are widely used for creating graphical user interfaces.

Scripting languages tend to be interpreted rather than compiled, which means that you don't need to compile them - they're compiled "on the fly" (i.e. when necessary, right before they're run). This can make it faster to program in them (since you always have the source code, and don't need to take the deliberate extra step of compiling)

Examples include: PHP, Perl, JavaScript, Coldfusion, Awk, csh, bash e.t.c

1.3 Syntax, Semantics and Programming Errors

1.3.1 Syntax

Syntax of a programming language means the rules that dictate how exactly the vocabulary elements of the language can be combined to form statements. Each programming language has its own unique syntax. For example, in English we know that a sentence must end with a question mark, a full stop or an exclamation mark. That is the rule that must be obeyed

During compilation, all syntax rules are checked. If a program is not syntactically correct, the compiler will issue error messages. Such errors could be like attempt to add two integers but the addition operator in not put between them or forgetting to close a procedure/function.

1.3.2 Semantics

The semantics of a statement in a programming language define what will happen when that statement is executed. In other words, semantics refer to the meaning of the statement.

Programming languages are generally unambiguous, which means the semantics are well defined. This means there’s only one and only one interpretation of each statement. Natural languages e.g. French and English are full of ambiguities.

1.3.3 Program Errors

You will encounter three types of errors as you develop programs

1. Compile time errors. These are errors identified by the compiler when it is compiling your program. They may include syntax errors and wrong data types. A program will not execute until when a compile time error is corrected.

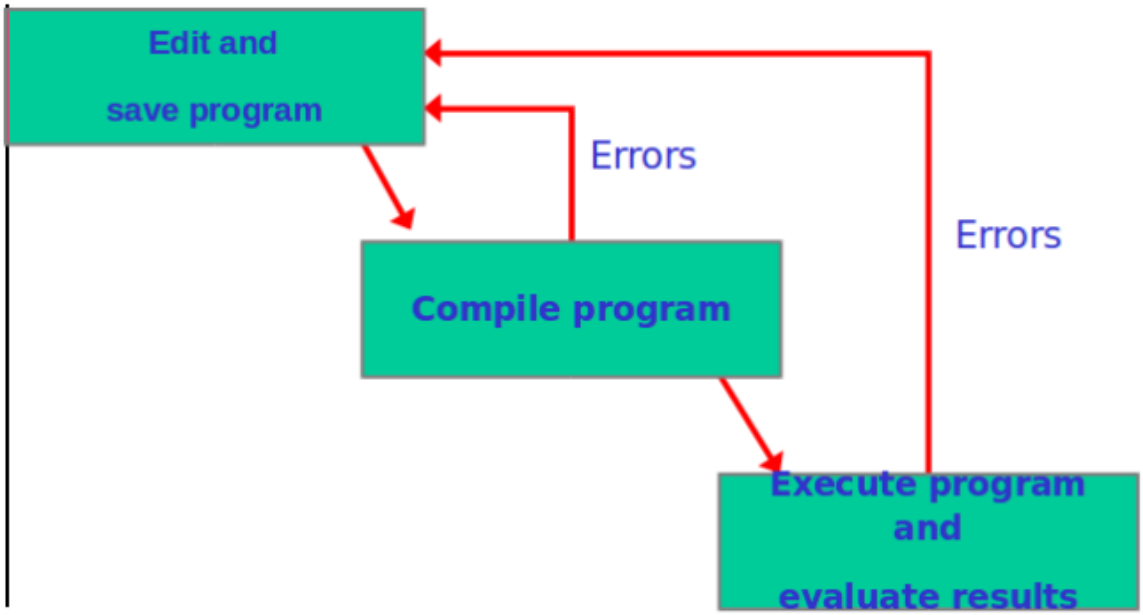
2. Runtime errors. These occur during the execution of a program and cause a program to abort abnormally e.g. division by zero, infinite looping.

3. Logical errors. The program compiles and runs normally but it produces wrong results. A logical error may occur when:

    - A value is calculated incorrectly e.g instead of addition you subtract.
    - When a graphical button does not appear in the correct place.

Logical errors are the most difficult to debug because they manifest themselves in many ways when their root cause is different.

Debugging is the process of finding and correcting errors/bugs in a program. The figure below shows the basic program development process.



1.4 Program Development and the Top-Down Design Approach

1.4.1 Program development

During the program development process, the following steps must be followed;

Step1. Understand the problem

Step2. Analyzing the problem we're given

Step3. Dissect the problem into manageable pieces

Step4. Developing a solution technique (algorithm)

Step5. Documenting the program/technique

Step6. Translating (implementing) the technique into code

Step7. Compiling and running the program

Step8. Testing the results with appropriate data

The above steps must be followed carefully. After step 8, if there are logical errors (i.e. program running properly but results are not correct), you must go back to step 6.

1.4.2 Top-Down Design Approach

Many problems are too large to immediately grasp and solve all the details; top-down design approach tries to address this issue.

- Take a problem, divide it into smaller logical sub-problems, and remember how they fit together
- For each sub-problem, divide it into still smaller sub-problems
- Continue sub-dividing until each little problem is small enough to be easily solved
- Solving a collection of small problems thus allows us to solve one much larger problem