FUNDAMENTALS OF PROGRAMMING WITH C LANGUAGE

“Any fool can write code that a computer can understand. Good programmers write code that humans can understand.” – Martin Fowler.

Prepared by Aphrodice Rwagaju

Rwanda Coding Academy

CHAP 1. INTRODUCTION TO C

C is a programming language developed at AT & T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL, etc. ANSI C standard emerged in the early 1980s, it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous. C was initially designed for programming UNIX operating system. Now the software tools as well as the C compiler are written in C. Major parts of popular operating systems like Windows, UNIX, Linux are still written in C. This is because even today when it comes to performance (speed of execution) nothing beats C. Moreover, if one is to extend the operating system to work with new devices, one needs to write device driver programs. These programs are exclusively written in C. C seems so popular because it is **reliable**, **simple and easy to use**. Often heard today is – "C has been already superseded by languages like C++, C# and Java."

1.1 Program

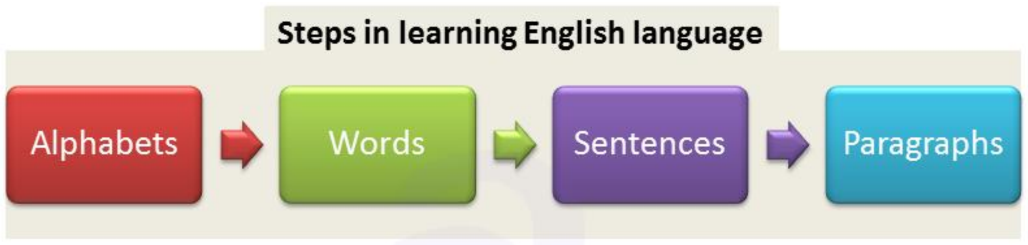
There is a close analogy between learning a language like English and learning C language. The classical method of learning English is to first learn the alphabets used in the language, then learn to combine these alphabets to form words, which in turn are combined to form sentences and sentences are combined to form paragraphs. Learning C is similar and easier. Instead of straight-away learning how to write programs, we must first know what alphabets, numbers and special symbols are used in C, then how using them. Constants, variables and keywords are constructed, and finally how are these combined to form an instruction. A group of instructions would be combined later on to form a program.

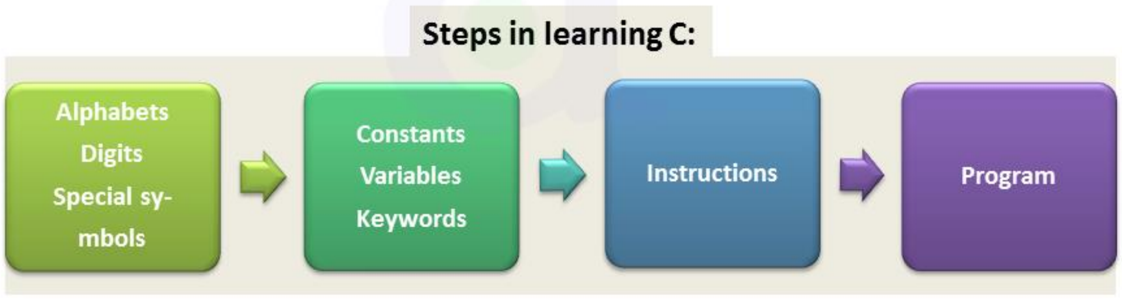
So, a **computer program** is just a collection of the instructions necessary to solve a specific problem when it is executed by a computer. The basic operations of a computer system form what is known as the **computer's instruction set**. And the approach or method that is used to solve the problem is known as an **algorithm**.

A computer program is usually written by a **computer programmer** and can be written in either high-level or low-level languages, depending on the task and the hardware being used.

1.2 Programming language

A **programming language** is a formal language, which comprises a set of instructions that produce various kinds of output. Programming languages consist of instructions for computers and they are used in computer programming to implement specific algorithms.





Programming languages are divided into two different levels:

1. Low level language
2. High level language

1.2.1 Low-level language

Low level languages are **machine level** and **assembly level** language.

In machine level language computer only understand digital numbers i.e. in the form of 0 and 1. So, instruction given to the computer is in the form of binary digit, which is difficult to implement instruction in binary code. This type of program is not portable, difficult to maintain and also error prone.

The assembly language is on other hand modified version of machine level language. Where instructions are given in English like word such as ADD, SUM, MOV etc. It is easy to write and understand but not understood by the machine. So the translator used here is **assembler** to translate into machine level. Although the language is bit easier, programmer has to know low level details related to low level language. In the assembly level language, the data are stored in the computer register, which varies for different computer. Hence, the **assembly language is not portable**.

1.2.2 High level language

These languages are machine independent, means **they are portable**. Examples of languages in this category are Pascal, Cobol, Fortran, C, Python, Java, etc.

High level languages are not understood by the machine. So it needs to be translated into machine level by the translator.

1.2.2.1 Translator

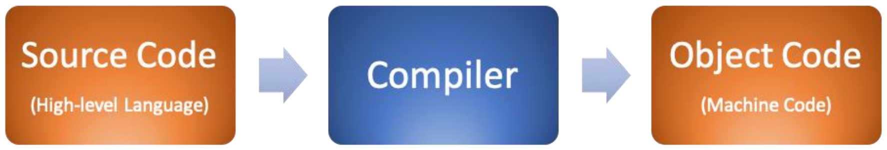
A **translator** is software which is used to translate high level language as well as low level language in to machine level language.

There are three types of translators namely: **Compiler, Interpreter** and **Assembler.**

Compiler and **interpreter** are used to convert the high-level language into machine-level language.

The program written in high-level language is known as source program and the corresponding machine-level language program is called as object program.

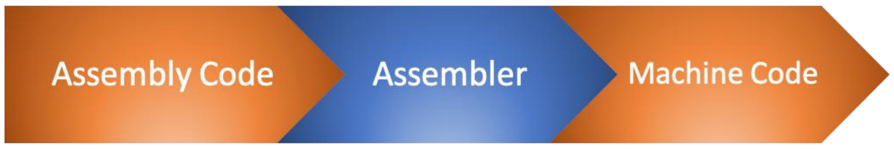
Both compiler and interpreter perform the same task but how they work is different. Compiler read the program at-a-time and searches the error and lists them. If the program is error free then it is converted into object program. When program size is large then compiler is preferred. Whereas interpreter read only one line of the source code and convert it to object code. Interpreter checks error, statement by statement and hence this process takes more time.



An **assembler** translates assembly language into machine code and is effectively a compiler for the assembly language, but can also be used interactively like an **interpreter**.

Assembly language uses words called '**mnemonics'**, such as **LOAD**, **STORE** and **ADD**.
The instructions are specific to the hardware being programmed because different
CPUs use different programming languages.

And finally, every assembly language instruction is translated into a single machine
code instruction.



Difference between Compiler and Interpreter

Compiler	Interpreter
Translates the whole program to produce the executable object code.	Translates and executes one line at a time.
Compiled programs execute faster as they are already in machine code.	Interpreted programs take more time to execute because each instruction is translated before being executed.
Users do not need to have the compile installed on their computer to run the software.	Users must have the interpreter installed on their computer and they can see the source code.
Users cannot see the actual source code when you distribute the program.	Users can see the source code and can copy it.
Used for software that will run frequently or copying software sold to a third party.	Used for program development and when the program must be able to run on multiple hardware platforms.

Difference between low-level and high-level languages

Type of Language	Example Language	Description	Example Instructions

High-level Language	Pascal, Cobol, Fortran, Visual Basic, C, C++, Python, Java	Independent of hardware (portable). Translated using either a compiler or interpreter. One statement translated into many machine code instructions.	payRate = 10000
			hours = 33.5
			salary = payRate * hours

Low-level Language	Assembly language	Translated using an assembler. One statement translated into machine code instruction.	LDA181 ADD93 STO185
	Machine Code	Executable binary code produced either by a compiler, interpreter or assembler.	1101001010100000111010100101101

1.2.3 Advantages of high-level and low-level programming language

1.2.3.1 Advantages of high-level language

Most software are developed using high-level languages for the following reasons:

- High-level languages are relatively easy to learn and much faster to program in.
- Statements within these languages look like the natural English language, including mathematical operators making it easier to read, understand, debug and maintain.
- Complexed assignment statements such as **x=(sqr(b^2-4*a*c))/(2*a)**
- Allowing the programmer to show how the algorithm will solve a problem in a clearer and more straightforward way.

- Specialized high-level languages have been developed to make the programming as quick and easy as possible for particular applications such as, SQL specially written to make it easy to search and maintain databases. HTML, CSS and JavaScript were also developed to help people create web pages and websites.

1.2.3.2 Advantages of low-level languages

- Assembly language is often used in embeded systems such as systems that control a washing machine, traffic lights, robots, etc.
- It has features that make it suitable for the following types of applications:
  - It gives complete control to the programmer over the system components, so it can be used to control and manipulate specific hardware components.
  - It has very efficient code that can be written for a particular processor architecture, so will occupy less memory and execute (run) faster than a compiled/interpreted high-level language.

1.3 Integrated Development Environments (IDE)

Integrated Development Environment or IDE for short is an application or software which programmers use for programming.

An IDE is a windows-based program that allows programmers to easily manage large software programs. It often helps them in editing, compiling, linking, running, and debugging programs.

On Mac OS X, CodeWarrior, Xcode and Eclipse are examples of IDEs that are used by many programmers. Under Windows OS, Visual Studio Code, Eclipse, NetBeans, CodeBlocks are good example of a popular IDEs. Kylix, Dev-C++, Eclipse, NetBeans are popular IDEs for developing applications under Linux OS.

Most IDEs also support program development in several different programming languages in addition to C, such as C#, C++, Java, etc.