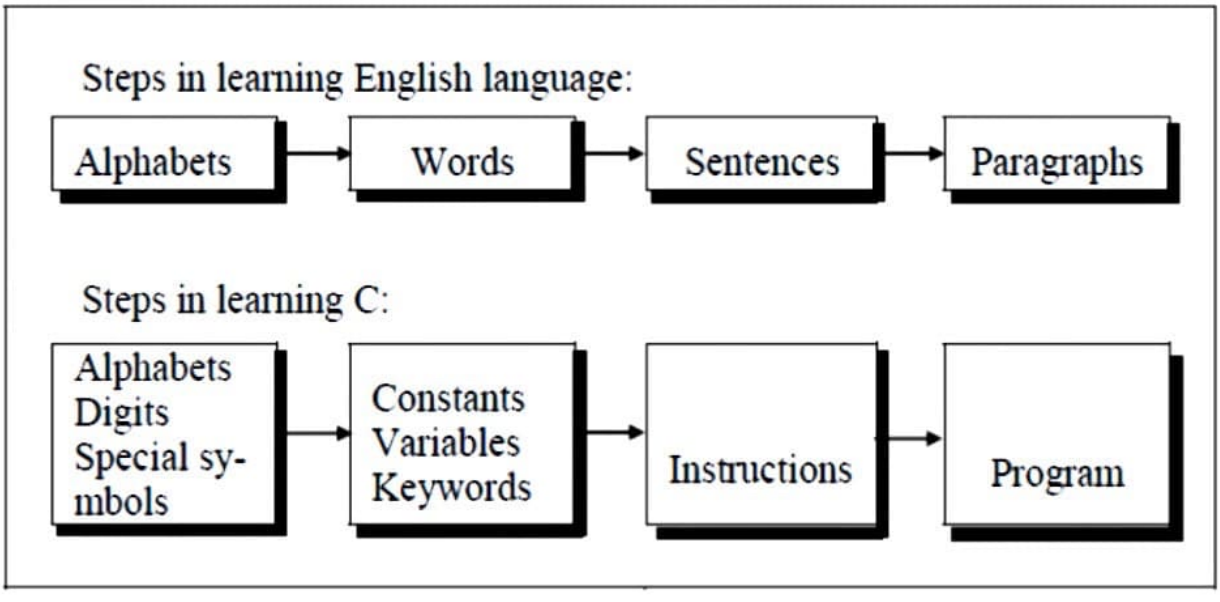
Introduction to C

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, this book was split into two titles: The original was still called ***Programming in C***, and the title that covered ANSI C was called ***Programming in ANSI C***. This was done because it took several years for the compiler vendors to release their ANSI C compilers and for them to become ubiquitous. It was initially designed for programming UNIX operating system. Now the software tool as well as the C compiler is written in C. Major parts of popular operating systems like Windows, UNIX, Linux is 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 is because it is **reliable, simple** and **easy** to use. often heard today is – "C has been already superceded by languages like C++, C# and Java.

There is a close analogy between learning English language 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. 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***.

So for as programming language concern these are of two types.

1) Low level language

2) High level language

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 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 as ADD, SUM, MOV etc. It is easy to write and understand but not understand by the machine. So the translator used here is assembler to translate into machine level. Although 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 it is not portable.

High level language:

These languages are machine independent, means it is portable. The language in this category is Pascal, Cobol, Fortran etc. High level languages are understood by the machine. So it need to translate by the translator into machine level. A translator is software which is used to translate high level language as well as low level language in to machine level language.

Three types of translator are there:

Compiler

Interpreter

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 there working 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. If it check error, statement by statement and hence of take more time.

Integrated Development Environments (IDE)

The process of editing, compiling, running, and debugging programs is often managed by a single integrated application known as an Integrated Development Environment, or IDE for short. An IDE is a windows-based program that allows us to easily manage large software programs, edit files in windows, and compile, link, run, and debug programs.

On Mac OS X, CodeWarrior and Xcode are two IDEs that are used by many programmers. Under Windows, Microsoft Visual Studio is a good example of a popular IDE. Kylix is a popular IDE for developing applications under Linux. Most IDEs also support program development in several different programming languages in addition to C, such as C# and C++.

Structure of C Language program

1) Comment line

2) Preprocessor directive

3) Global variable declaration

4) main function( )

```
        {
          Local variables;
  Statements;
  }
  User defined function
 }
}
```

Comment line

It indicates the purpose of the program. It is represented as

/*.....*/

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed within the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Preprocessor Directive:

#include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition &declaration of system defined function such as printf( ), scanf( ), pow( ) etc. Generally printf() function used to display and scanf() function used to read value

Global Declaration:

This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function :

main()

It is the user defined function and every function has one main() function from where actually program is started and it is encloses within the pair of curly braces.

The main( ) function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
  main()
{
……..
……..
……..
}
```

The main( ) function return value when it declared by data type as

```
 int main( )
{
return 0
```

}

The main function does not return any value when void (means null/empty) as

void main(void) or void main()

{

printf ("C language");

}

Output: C language

The program execution start with opening braces and end with closing brace.

And in between the two braces declaration part as well as executable part is mentioned. And at the end of each line, the semi-colon is given which indicates statement termination.

/*First c program with return statement*/

#include <stdio.h>

int main (void)

{

printf ("welcome to c Programming language.\n");

return 0;

}

Output: welcome to c programming language.

Steps for Compiling and executing the Programs

A compiler is a software program that analyzes a program developed in a particular computer language and then translates it into a form that is suitable for execution

on a particular computer system. Figure below shows the steps that are involved in entering, compiling, and executing a

computer program developed in the C programming language and the typical Unix commands that would be entered from the command line.

Step 1: The program that is to be compiled is first typed into a *file* on the computer system. There are various conventions that are used for naming files, typically be any name provided the last two characters are **".c"** or file with extension .c. So, the file name **prog1.c** might be a valid filename for a C program. A text editor is usually used to enter the C program into a file. For example, vi is a popular text editor used on Unix systems. The program that is entered into the file is known as the ***source program*** because it represents the original form of the program expressed in the C language.

Step 2: After the source program has been entered into a file, then proceed to have it compiled. The compilation process is initiated by typing a special command on the system. When this command is entered, the name of the file that contains the source program must also be specified. For example, under Unix, the command to initiate program compilation is called **cc**. If we are using the popular GNU C compiler, the command we use is **gcc**.

Typing the line

gcc prog1.c or cc prog1.c

In the first step of the compilation process, the compiler examines each program

statement contained in the source program and checks it to ensure that it conforms to the syntax and semantics of the language. If any mistakes are discovered by the compiler during this phase, they are reported to the user and the compilation process ends right there. The errors then have to be corrected in the source program (with the use of an editor), and the compilation process must be restarted. Typical errors reported during this phase of compilation might be due to an expression that has unbalanced parentheses (**syntactic error**), or due to the use of a variable that is not "defined" (**semantic error**).

Step 3: When all the syntactic and semantic errors have been removed from the program, the compiler then proceeds to take each statement of the program and translate it into a "lower" form that is equivalent to assembly language program needed to perform the identical task.

Step 4: After the program has been translated the next step in the compilation process is to translate the assembly language statements into actual machine instructions. The assembler takes each assembly language statement and converts it into a binary format known as ***object code***, which is then written into another file on the system. This file has the same name as the source file under Unix, with the last letter an "**o**" (**for *object***) instead of a "**c**".

Step 5: After the program has been translated into object code, it is ready to be ***linked.*** This process is once again performed automatically whenever the cc or gcc command is issued under Unix. The purpose of the linking phase is to get the program into a final form for execution on the computer.

If the program uses other programs that were previously processed by the compiler, then during this phase the programs are linked together. Programs that are used from the system's program ***library*** are also searched and linked together with the object program during this phase.

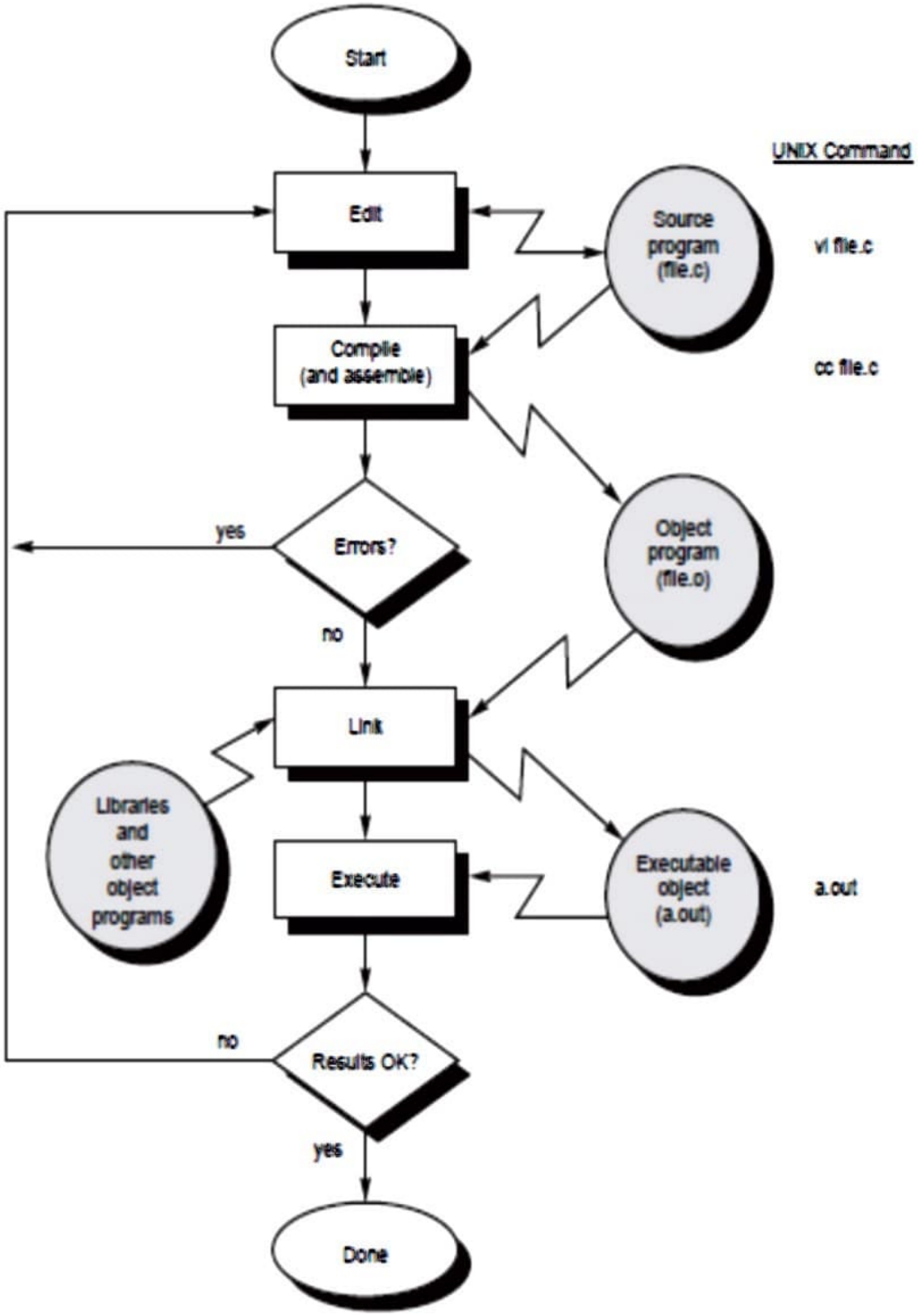
The process of compiling and linking a program is often called ***building.***

The final linked file, which is in an executable ***object*** code format, is stored in another file on the system, ready to be run or ***executed.*** Under Unix, this file is called **a.out** by default. Under Windows, the executable file usually has the same name as the source file, with the c extension replaced by an exe extension.

Step 6: To subsequently execute the program, the command **a.out** has the effect of *loading* the program called **a.out** into the computer's memory and initiating its execution.

When the program is executed, each of the statements of the program is sequentially executed in turn. If the program requests any data from the user, known as ***input***, the program temporarily suspends its execution so that the input can be entered. Or, the program might simply wait for an ***event***, such as a mouse being clicked, to occur. Results that are displayed by the program, known as ***output***, appear in a window, sometimes called the ***console***. If the program does not produce the desired results, it is necessary to go back and reanalyze the program's logic. This is known as the ***debugging phase***, during which an attempt is made to remove all the known problems or ***bugs*** from the program. To do this, it will most

likely be necessary to make changes to original source program.



/* Simple program to add two numbers……*/

```
#include <stdio.h>
int main (void)
{
int  v1, v2, sum;                    //v1,v2,sum are variables and int is data type declared
v1 = 150;
v2 = 25;
sum = v1 + v2;
printf ("The sum of %i and %i is= %i\n", v1, v2, sum);
return 0;
}
```

Output:

The sum of 150 and 25 is=175