SYLLABUS

Module –I

C Language Fundamentals.
Character set, Identifiers, keyword, data types, Constants and variables, statements, expression, operators, precedence of operators, Input-output, Assignments, control structures decision making and branching.

Module -II

Arrays, Functions and Strings: Declaration, manipulation and String – handling functions, monolithic vs. Modular programs, user defined vs. standard functions, formal vs. actual arguments, function – category, function prototypes, parameter passing, recursion, and storage classes: auto, extern, global, static.

Module –III

Pointers, Structures, Unions, File handling:

Pointer variable and its importance, pointer arithmetic, passing parameters, Declaration of structures, pointer to pointer, pointer to structure, pointer to function, union, dynamic memory allocation, file managements.

*Under revision

CONTENTS

Lecture 13: String library functions

Lecture 14: functions, categories

Lecture 15: functions categories cont..

Lecture 16: Actual arguments and Formal arguments, call by value call by reference

Lecture 17:local, global, static variable

Lecture 18: monolithic vs modular programming, Storage classes

Lecture 19:storage class cont.., pointer

Lecture 20: pointer comparison, increment decrement

Lecture 21: precedence level of pointer, pointer comparison

Lecture 22: pointer to pointer, pointer to structure

Lecture 23: pointer initialization, accessing elements

Module: 3

Lecture 24: size of Structure in, array vs structure, array within structure

Lecture 25: passing structure to function, Nested Structure

Lecture 26: Union

Lecture 27: nesting of unions, dynamic memory allocation

Lecture 28: dynamic memory allocation conti…

Lecture 29: dynamic array, file

Lecture 30: file operation

Lecture 31: file operation on string

Lecture 32:

Lecture 33:

Lecture Note: 1

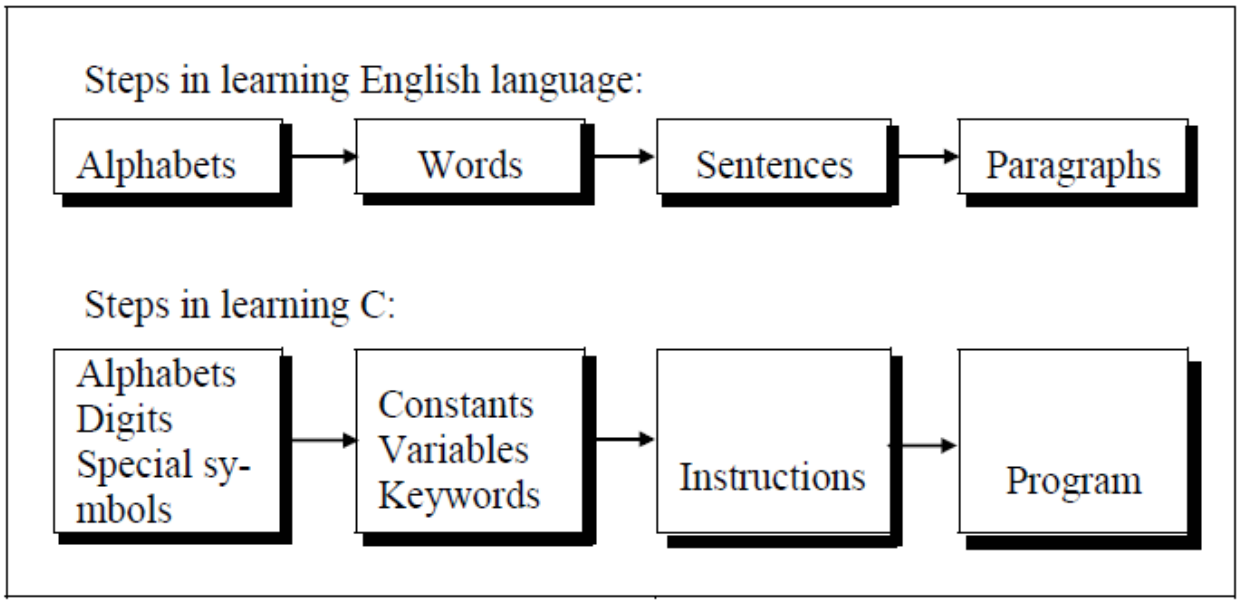
Introduction to **C**

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, 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.

Program

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.