History of C Programming Language

- ➔ C is a programming language which born at "AT & T's Bell Laboratory" of USA in 1972.
- ➔ C was written by Dennis Ritchie, that's why he is also called as father of c programming language.
- ➔ C language was created for a specific purpose i.e. designing the UNIX operating system (which is currently base of many UNIX based OS).
- ➔ From the beginning, C was intended to be useful to allow busy programmers to get things done because C is such a powerful, dominant and supple language
- ➔ Its use quickly spread beyond Bell Labs in the late 70's because of its long list of strong features.

Why Name "C" was given to Language?

- ➔ Many of C's principles and ideas were derived from the earlier language B. (Ken Thompson was the developer of B Language.)
- ➔ BCPL and CPL are the earlier ancestors of B Language
- ➔ CPL is common Programming Language. In 1967, BCPL Language (Basic CPL) was created as a scaled down version of CPL
- ➔ As many of the features were derived from "B" Language that's why it was named as "C". After 7-8 years C++ came into existence which was first example of object oriented programming.

Summary of C Programming Language History

Summary –

1 B Language Developed By Ken Thompson.
2 Operating System Developed in C- UNIX Operating System.
3 Developed at AT & T Bell Laboratory.
4 Creator of Traditional C -Dennis Ritchie
5 Year- 1972

C Programming Language Timeline:

Programming Language	Development Year	Developed by
ALGOL	1960	International Group
BCPL	1967	Martin Richards
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K&R C	1978	Brain Kernighan and Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee

- ➔ The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons –

- ⇨ Easy to learn.
- ⇨ Structured language.
- ⇨ It produces efficient programs.
- ⇨ It can handle low-level activities.
- ⇨ It can be compiled on a variety of computer platforms.

Why use C?

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. Some examples of the use of C might be –

- ⇨ Operating Systems
- ⇨ Language Compilers
- ⇨ Assemblers
- ⇨ Text Editors
- ⇨ Print Spoolers
- ⇨ Network Drivers
- ⇨ Modern Programs
- ⇨ Databases
- ⇨ Language Interpreters
- ⇨ Utilities

High level languages vs Low level languages

➔ Meaning:

- ⇨ With the help of high level language one can write applications that are portable across various platforms and is independent of any architecture.

  But Low Level languages, these languages are very close to machine language, they are also known as assembly language.

➔ Benefits:

- ⇨ High level languages are easier to understand and is user-friendly.

  But Low Level languages are more appropriate for developing new operating systems or writing firmware codes for micro-controllers.

➔ Speed:

- ⇨ High level languages has lots of abstractions and layers of code before they reach the hardware itself.

  But Low level, machine code is nearer to the hardware, which is actually fast to process and return the output.

➔ Portability:

- ⇨ High level languages are extremely portable, they are mostly used to write software's which can run on multiple platforms and architectures.

  But Low Level languages codes are very hard to understand, and code written in assembly language is impossible to run on other machine or architecture.

➔ Which is better:

- ⇨ High level is where all the creative things take place and it can be debugged in very easier manner than a low-level.

But low level is challenging and requires a great deal of experience and knowledge.

What is High Level Languages?

You must be thing what is High level language? But tell me you they are an easier to understand and is user-friendly. With the help of these languages one can write applications that are portable across various platforms (such as Linux or windows) and is independent of any architecture (such as non-intel ARM or the infamous Intel). High level language Example, writing a program in python which by default works in any Linux system, then just compiling it into any exe using py2exe and then running it on windows.

Similar examples are Python, C, FORTRAN or Pascal. Such languages are considered as high-level language because they are closer to human languages and much further from machine languages. When I say human language, I don't mean what we talk in our day-to-day life. It means the code is something we can understand by knowing some basics in programming. The code written is almost readable by humans, something that can be read and pronounced. But however, since we are talking about computers, for a computer this is hard to understand. So in order to make this sensible to computers and run a program created with a high-level language, it must be compiled into machine language.

And this is where Low-level language comes in between. Unlike previously, where there were only a **few high-level languages, today there are n number of high-level languages such as C, COBOL, FORTRAN, Pascal, Java, Perl, Python, PHP, Ruby, C++, BASIC and Visual Basic.**

What are Low-level languages?

Low-level languages those languages which are extremely close to machine language. They are also known as Assembly languages. The closest languages after Assembly to Machine language are C and C++. Some people even call C and C++ as low level languages. Machine code is known as low level because unlike high level programming languages it doesn't need anything else like compilers or something. It runs directly on the processor and they are extremely architecture specific.

Low-level languages are more appropriate for developing new operating systems or writing firmware codes for micro-controllers. They can do anything with a little bit of hard work (actually a lot of hard work to be specific), but obviously you won't want to write some major application in it. Similar is the case with C (Actually called as Cee). C is actually a very vast language to start with. It allows you to register directly and give instant access to various memory locations.

But at the same time it also has a lots of constructs that allow the hardware to load abstraction. Frankly speaking, C and C++ dually represent variety of languages, since most languages have taken its libraries from them. In practice, both C and C++ are low-level as I told you previously because writing applications on enterprise level is quite difficult. But theoretically, both of them are actually high-level languages.

C Programming – The Low-level/High-level Confusion

Though C has lots of characteristics similar to that of Pascal Language, sometimes it is still considered as a low level language, reason being it supports operations of bits, pointers and direct access to memory. C actually is a high level language with the inclusive features of low level. This is the main reason why programmers depend on C-over anything for its unbeatable qualities. It may seem weird that C though treated as a low level language, is extremely portable. Fanatically speaking, C is actually extended to use hardware at its extreme limits as possible.

Assembly language, on the other hand is hardly portable. Though, trying to achieve portability is a big deal in case of low-level especially in the case of Java, which runs on a JVM i.e. a virtual machine. C or Assembly running in a VM will never have pure access to the hardware. To be more precise, a language becomes a low level if it is specifically structured to run directly on the hardware. Low level languages have very less syntax, unlike High level languages which have loads of codes. Languages that are low level which allow full access of the hardware would actually be a poor choice to write projects.

CONCEPT OF HARDWARE AND SOFTWARE

Hardware: The physical component of a computer is called as hardware. The hardware may be an electronic, electrical, magnetic or mechanical components.

Ex: RAM, Floppy Disk, Hard Disk, Key Board, Printer etc.

Software: Software is the set of computer programs which is used for some particular purpose. This may be some web applications like WhatsApp, Twitter, Flipkart or desktop applications like- MS Word, Ms excel etc.

Role of software in computers-

As we know that set of instructions is called as program, and collection of programs are called as software. Instead of doing some work manually, a software can perform all those work automatically only we need to program that software according to our need.

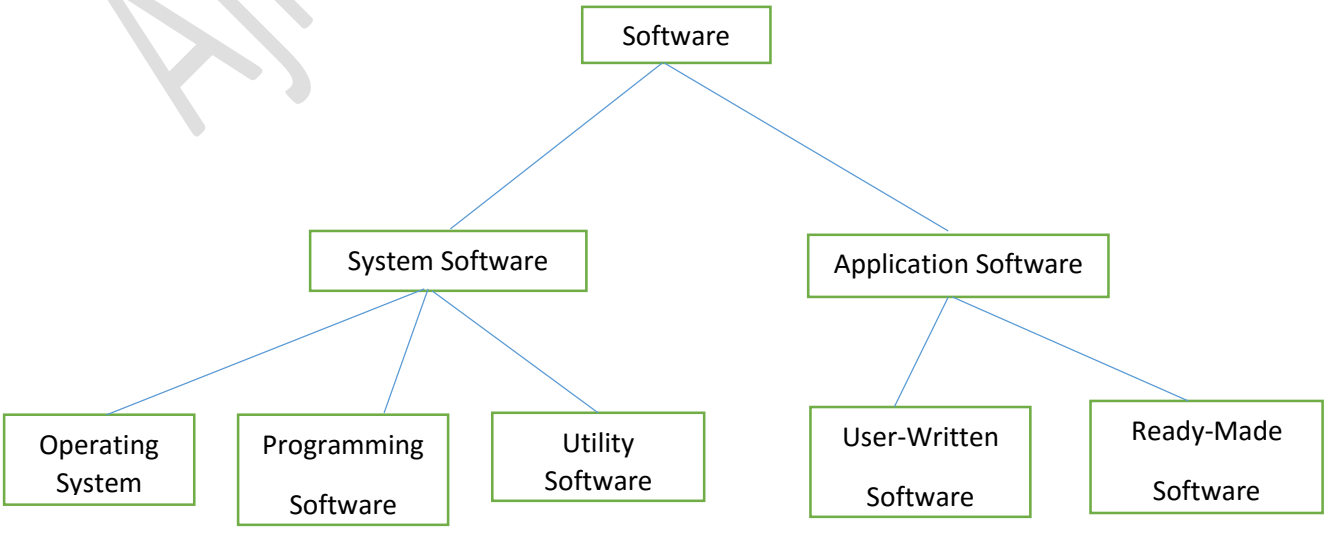
The each instruction in the program direct the computer to perform input operations, process the input data and display the result.

Types of Software

Basically there are two types of software-

1) System Software

2) Application software



System Software: The System Software is basically used by the computer system to control all the actions and operations of the system. These system software are also called as the system packages.

System software is a type of computer program that is designed to run a computer's hardware and application programs. If we think of the computer system as a layered model, the system software is the interface between the hardware and user applications.

Functions of system software-

a) Running of other software

b) Communications with attached or peripheral devices like- printers, scanners, card readers, CPU etc.

c) Development of other type of software.

Ex: Operating System, Compiler, Assembler, Loader, Linker etc.

Operating System

An operating system (OS) is system software that manages computer hardware and software resources and provides common services for computer programs.

Functions of operating System-

1) Job Management: The operating system responds to the command or instructions which are given by the user and load the desired application program from secondary storage to main memory (RAM) for their execution.

2) Task Management: Basically in single tasking computer system, the operating system doesn't need to have the task management. But in multi-tasking computer the operating system executes one or more than one program simultaneously at a time. To execute more than one program simultaneously the operating system needs to schedule the task so that every program gets the chance to execute.

3) Data Management: Data management is the biggest responsibility of operating system. Operating system keeps the track of entire data on disk. Actually the application or programs doesn't know where exactly the data is placed. So the operating system avails the data for the applications when needed.

4) Security: The computer system which is operated by more than one user needs to be separated from each other. So the operating system maintains the list of users and provides the password protection for each user. Only the valid user can get access to the computer system. It also provides the backup and recovery support to the user in case of forgot the password.

5) Bootstrap program: Boot means start or make the computer system ready to work. The word "boot" comes from "bootstrap". Since bootstrapping helps the computer to run the instruction written in ROM to execute and copy the operating system files from secondary to primary memory (RAM). After finishing the booting, our computer system is ready to function.

Programming Software:

Programming software is basically given for development of software or computer program. This computer programs can be written in different languages like- C, C++, Java, .Net etc. This software are used for different purpose or different work. Ex-Flipkart, WhatsApp, MS Word, MS Excel etc.

Examples of programming software are- Assembler, Compiler, interpreter etc.

Some Basic tools which are used in programming software

1) **Assembler:** Assembler is a programming software or tool that translates assembly language programs into machine language program. Because our machine directly doesn't understands the assembly language so assembler converts this assembly language code to binary format so that our machine can understands it easily. Assembler takes assembly language as an input and it produces object code/machine code as output.

2) **Compiler:** A Compiler is a program that converts high level language program to machine language program. Actually a compiler is much more intelligent than assembler because compiler checks all kind of limits, range and errors and syntax mistake. Compiler take more execution time for a program than assembler. If a compiler runs on the same computer for which it produces the object code then it is called as self/resident compiler. But if the compiler runs on different computer and generates the object code for other computer then it is called as cross compiler.

3) **Interpreters:** An interpreters is a program which translates the high level languages program to machine code one by one or statement by statements. It is basically used for debugging purpose to trace the errors and bugs in the program.

   a) Interpreter is slower than compiler because it executes line by line but compiler executes entire program at a time.

   b) Interpreter is a smaller program as compared to compiler.

   c) Interpreter occupies less memory in comparison to compiler.

4) **Linker**: In computing, a linker or link editor is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another object file.

Utility Software

Utility software is system software designed to help analyse, configure, optimize or maintain a computer.

Ex: antivirus is a best example for utility software.

Application Software

An Application software is a program or set of programs that is designed for end-user. Each and every applications software is designed to perform certain work.

Ex: Facebook- is a web application which is designed for chatting, Flipkart-For Shopping, WhatsApp for messaging etc.

Application software can be classified into two category:

1) **Readymade Software:** These are the software which are developed not for specific user but it can be used by any user.

   Ex: Tally, Ms Word.

2) **User-Written Software:** These are the software which are developed by end user for specific need. It may be the websites or some applications. WhatsApp is user written software which is

basically for messaging purpose. The requirement varies user to user so the development is also differ.

COMPUTER LANGUAGE

Languages plays a vital role in communications. Means when two or more device is in communication they should must understand one another. For this communication purpose computer follows some languages. This languages should be understand by both user and computer systems.

The Computer languages are broadly classified as-

1) Machine Language

2) Assembly Language

3) High Level Language

1) MACHINE LANGUAGE:

Binary notation (means 0 & 1) is directly understand by the computer. We don't need to convert/translate it. This binary notation is called is called machine code or Machine language program.

The programmer directly write the program using binary number (0, 1), which quite tough and need years of experience to do so. There are specific binary code for each instruction. So remembering this code is not that easy.

Advantage of Machine Language

1) The programs written in machine language can be executed very fast by the computer.

2) This program executed very fast because they don't need to translate, it can be directly understand by the computer system.

Dis-advantage of Machine Language

1) **Machine Dependent:** Machine dependency means the code written on one system can't be run on other system.

Ex: Intel processor have different instruction set while AMD processor has different instruction set.

2) **Difficult to Program:** As we know that the machine language is directly understand by the system, but writing this program is quite tough. Because the programmer needs to remember each and every instruction code to program which needs a years of experiences.

3) **Error prone:** Writing this code is bit complex and hence there is more chance of error. While programming we need to write the equivalent instruction code which is quite confusing. And also we need to remember the storage location of each and every instruction or data as well as the opcode.

4) **Difficult to modify:** Once the program has been written is very complex to modify.

ASSEMBLY LANGUAGE:

It is the programming language which is used to write the program to give the instruction to computer system to perform some activity. Previously in machine level language we are directly writing the instruction in numerical format but here in assembly language we are writing instruction in word.

Here instruction is written in word format is converted to numerical instruction by the help of assembler. An assembler is a program which converts assembly language instruction to machine level (binary) instruction.

Advantage of assembly Language

1) Assembly language program takes less execution time as well as memory as compared to high level language programs.

2) It needs less processing power of a CPU and thus runs faster than the high level programming language.

Dis-advantage of Assembly language program

1) Programming is difficult and debugging is time consuming.

2) It is slower than the machine level programming language because it needs on translator to convert assembly code to machine code.

3) The program written on one computer can't be run on an-other computer system having the different set of hardware.

4) In order to write the assembly language program the programmer must have the detailed knowledge of hardware of that system on which they are working including knowledge of registers, instruction sets, and connection of ports to the peripherals etc.

HIGH LEVEL LANGUAGE

This is language which is easy for the user, so it is called as user friendly language. It is the language which can be written in English. To overcome the problems of assembly level language, high level language we introduced.

It is basically a problem based rather than the computer based. The every instruction written in high level language is called as a statements. This statement can be a combination of English alphabets and mathematics.

Ex: FORTRAN, BASIC, PASCAL, C, C++, Java, Ruby, Python etc.

- ⇨ This high level language is user friendly but not machine friendly, so it is not directly understand by the machine. To make it understandable to the machine it needs to be translate. So compiler is used to convert this high level language to machine level language.

FIRST PROGRAM IN C PROGRAMMING LANGUAGE

```
/* First Demo Program in C Programming */
#define PI 3.14;
#include <stdio.h>
#include <conio.h>
void main ()
{
        clrscr ();
        printf ("Hello World!!!");
}
```

Comment

Macro Declaration

Pre-processor directives (#) + Standard Input output

Entry point to program

To clear the screen

Printing Hello World on Console

To compile: alt+f9

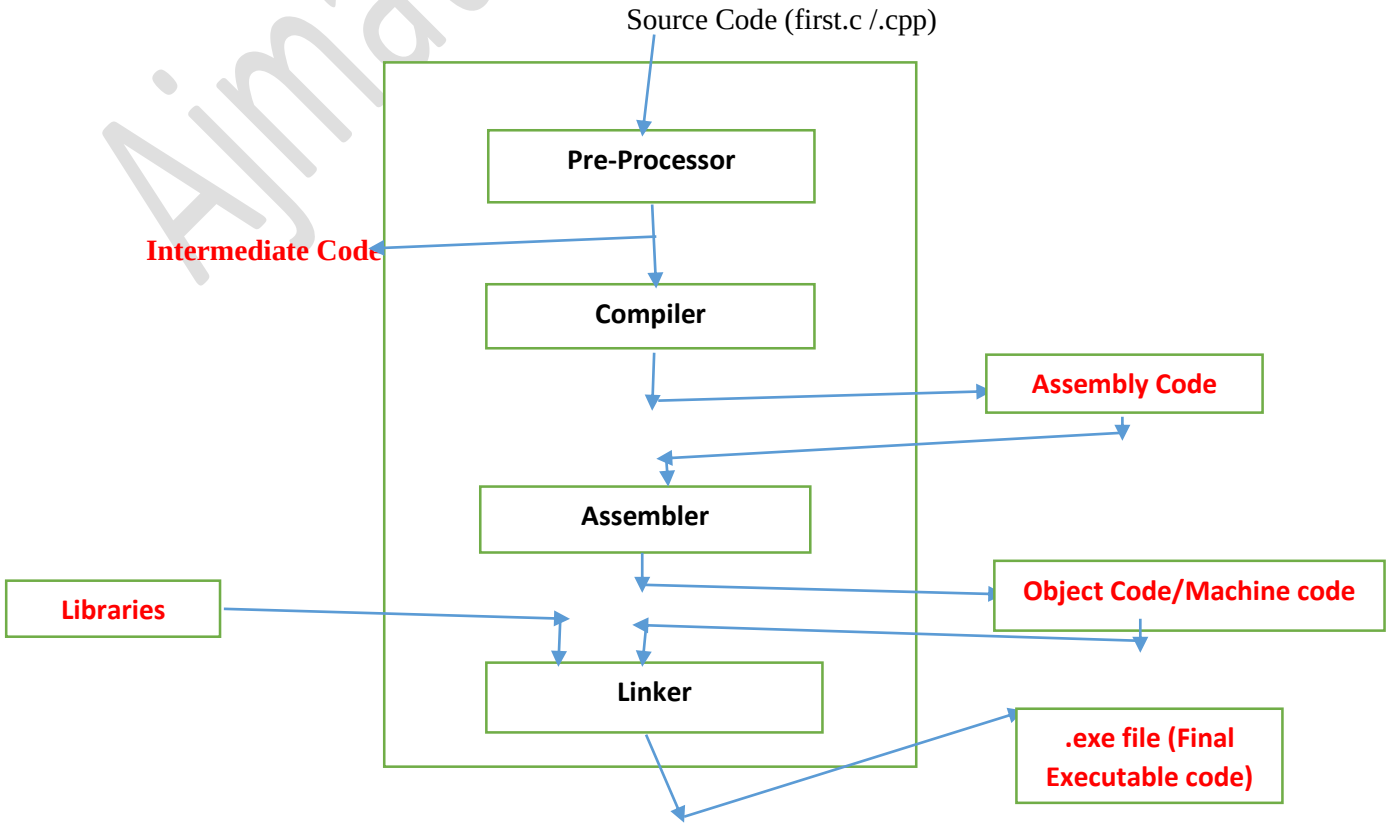
To Run: ctrl+f9

Output: Hello World!!!

Source Code

How exactly the code gets executed

Compiler is not a single module, but it is having a multiple modules. We can divide them into 4 modules which are given as follows-



Pre-processor: is the first pass of any C compilation. It processes include-files, conditional compilation instructions and macros. It basically takes the source code as input and it generates some intermediate code.

- ➔ Pre-processor ignores the comment.
- ➔ It will include the header file in our code instead of #include<stdio.h> & #include<conio.h>.
- ➔ It will replace all macro name with their original code/value if we are using any macro in our program.

Compiler: is the second pass. It takes the output of the pre-processor, and the source code, and generates assembly source code. This assembly code is input to the assembler.

- ➔ Compiler generates the assembly code in the form of mnemonics.
- ➔ It is basically a English equivalent code or words.

Assembler: is the third stage of compilation. It takes the assembly source code and produces an assembly listing with offsets. The assembler output is stored in an object file.

- ➔ Assembler converts the assembly code to pure binary code which is an object code/machine code.

Linker: is the final stage of compilation. It takes one or more object files or libraries as input and combines them to produce a single (usually executable) file. In doing so, it resolves references to external symbols, assigns final addresses to procedures/functions and variables, and revises code and data to reflect new addresses (a process called relocation).

- ➔ Suppose multiple programmers working on different modules of a project. In that case if we compile the entire code then we will get many object code. The linker will link all these object code together and gives the single executable code (.exe file).
- ➔ Suppose we are using some built-in function from library in our program, in that case the linker will link our code with that library function code.
- ➔ Basically there are two types of linking
  1) Static linking
  2) Dynamic linking

  1) Static linking:
  - Static linking is the process of copying all library modules used in the program into the final executable image. This is performed by the linker and it is done as the last step of the compilation process. The linker combines library routines with the program code in order to resolve external references, and to generate an executable image suitable for loading into memory. When the program is loaded, the operating system places into memory a single file that contains the executable code and data. This statically linked file includes both the calling program and the called program.
  - Static linking is performed by programs called linkers as the last step in compiling a program. Linkers are also called link editors.
  - Statically linked files are significantly larger in size because external programs are built into the executable files.
  - In static linking if any of the external programs has changed then they have to be recompiled and re-linked again else the changes won't reflect in existing executable file.

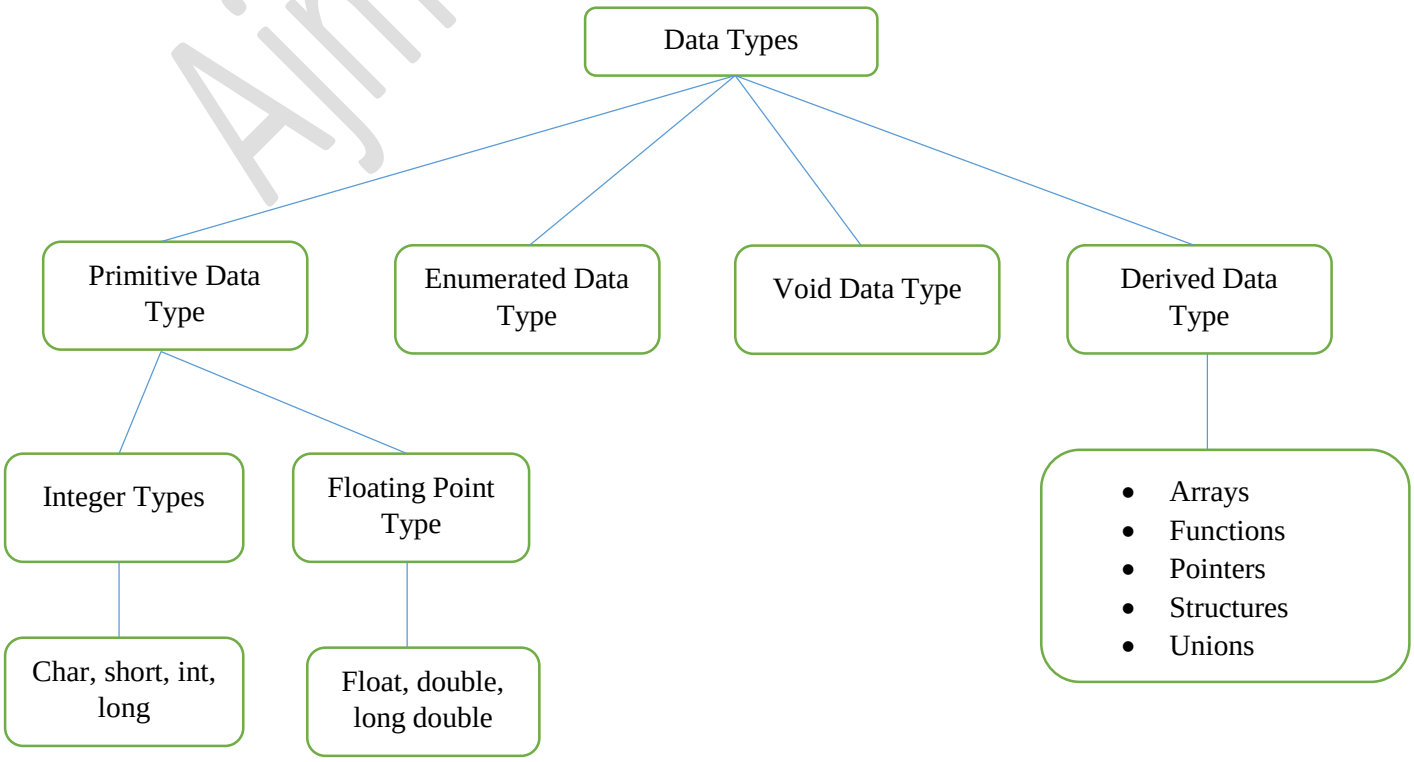
- Statically linked program takes constant load time every time it is loaded into the memory for execution.
- Programs that use statically-linked libraries are usually faster than those that use shared libraries.
- In statically-linked programs, all code is contained in a single executable module. Therefore, they never run into compatibility issues.

2) Dynamic Linking:

- In dynamic linking the names of the external libraries (shared libraries) are placed in the final executable file while the actual linking takes place at run time when both executable file and libraries are placed in the memory. Dynamic linking lets several programs use a single copy of an executable module.
- Dynamic linking is performed at run time by the operating system.
- In dynamic linking only one copy of shared library is kept in memory. This significantly reduces the size of executable programs, thereby saving memory and disk space.
- In dynamic linking this is not the case and individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers.
- In dynamic linking load time might be reduced if the shared library code is already present in memory.
- Programs that use shared libraries are usually slower than those that use statically-linked libraries.
- Dynamically linked programs are dependent on having a compatible library. If a library is changed (for example, a new compiler release may change a library), applications might have to be reworked to be made compatible with the new version of the library. If a library is removed from the system, programs using that library will no longer work.

Data Types

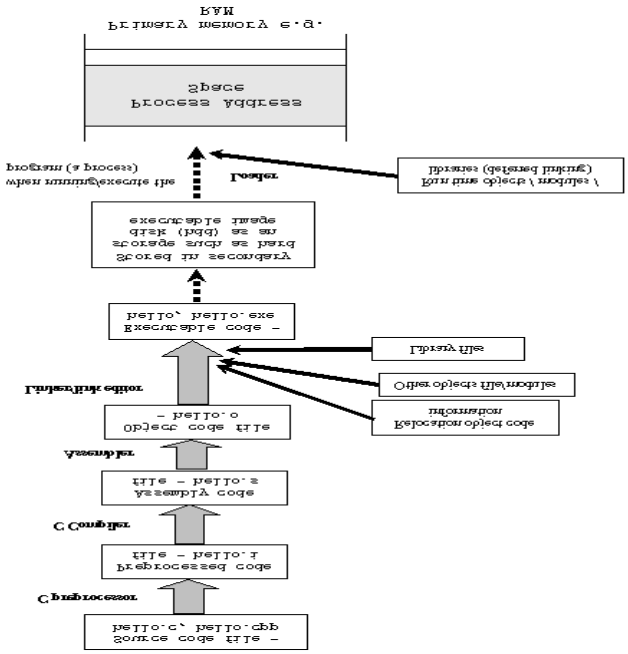
==========



Type	Storage size	Value range
char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	4 bytes	-2,147,483,648 to 2,147,483,647
unsigned long	4 bytes	0 to 4,294,967,295

Floating Point Data

e	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places



```c
#include <stdio.h>
#define PI 3.1415
#define circleArea(r) (PI*r*r)

int main()
{
    int radius;
    float area;
    printf("Enter the radius: ");
    scanf("%d", &radius);
    area = circleArea(radius);
    printf("Area = %.2f", area);
    return 0;
}
```

ASCII TABLE AND DESCRIPTION

ASCII stands for American Standard Code for Information Interchange. Computers can only understand numbers, so an ASCII code is the numerical representation of a character such as 'a' or '@' or an action of some sort. ASCII was developed a long time ago and now the non-printing characters are rarely used for their original purpose. Below is the ASCII character table and this includes descriptions of the first 32 non-printing characters. ASCII was actually designed for use with teletypes and so the descriptions are somewhat obscure. If someone says they want your CV however in ASCII format, all this means is

they want 'plain' text with no formatting such as tabs, bold or underscoring - the raw format that any computer can understand. This is usually so they can easily import the file into their own applications without issues. Notepad.exe creates ASCII text, or in MS Word you can save a file as 'text only'.

C CHARACTER SET

Whenever we write any C program then it consists of different statements. Each C Program is set of statements and each statement is set of different c programming leximes. In C Programming each and every character is considered as single lexim. i.e. [Basic Lexical Element].

Character Set Consists Of -

Types	Character Set
Lowercase Letters	a-z
Uppercase Letters	A to Z
Digits	0-9
Special Characters	!@#$%^&*
White Spaces	Tab Or New line Or Space

Valid C Characters: Special Characters are listed below –

Symbol	Meaning
~	Tilde
!	Exclamation mark

#	Number sign
$	Dollar sign
%	Percent sign
^	Caret
&	Ampersand
*	Asterisk
(	Left parenthesis
)	Right parenthesis
_	Underscore
+	Plus sign
|	Vertical bar
\	Backslash
`	Apostrophe
–	Minus sign

=	Equal to sign
{	Left brace
}	Right brace
[	Left bracket
]	Right bracket
:	Colon
”	Quotation mark
;	Semicolon
<	Opening angle bracket
>	Closing angle bracket
?	Question mark
’	Comma
.	Period
/	Slash

Special Backslash Character Constants in C:

Constant	Meaning
‘a’	Audible Alert (Bell)
‘b’	Back Space
‘f’	Form Feed
‘n’	New Line
‘r’	Carriage Return
‘t’	Horizontal Tab
‘v’	Vertical Tab
”’	Single Quote
‘”‘	Double Quote
‘?’	Question Mark
‘\’	Backslash
‘\0’	Null

How many Spaces Makes One Tab Space:

1. Generally **8 Spaces** makes one Tab in **Borland C, C++ 3.0 Compiler**
2. It differs from **Compiler to Compiler** And also different for different **Word Processors**

Backslash Characters: Properties

You need to learn what is Backslash Character in C Programming.

1. Although it consists of two characters, it represents **single character**.
2. Each escape sequence has **unique ASCII** value.
3. Each and Every combination starts with **back slash**()

4. They are **non-printable** characters.
5. It can also be expressed in terms of **octal digits or hexadecimal** sequence.
6. Escape sequence in character constants and string literals are replaced by their equivalent and then adjacent string literals are concatenated
7. Escape Sequences are pre-processed by **Pre-processor**.

1. Tab: '\t' Character

- It is Horizontal Tab
- Takes Control 8 spaces ahead in Borland CC++ 3.0 Compiler

```
#include<stdio.h>


int main()
{
printf("Hello\t");
return(0);
}
```

Cursor Position After Execution of Printf :

```
Hello       _
```

2. New Line Character: '\n' Character

- It is New Line Character
- Takes Control to new Line

```
#include<stdio.h>
int main()
{
printf("Hello\n");
return(0);
}
```

Cursor Position After Printf :

```
Hello

_
```

3. Backslash: ‘\b’ Character

- It is Backslash Character
- Takes Control one position back

```
#include<stdio.h>
int main()
{
printf("Hello\b");
return(0);
}
```

Cursor Position:

```
Hell_
```

Note: on ‘o’ character from Word Hello

4. Carriage Return: ‘\r’ Character

- It is Carriage Return Character
- Takes Control to First Position in the Line

```
printf("Hello\r");
```

Cursor Position:

Note : Cursor on ‘H’ character from Word Hello

```
_allo
```

4. Audible Return: ‘\a’ Character

- It is audible Return Character
- Beeps Sound

```
printf ("Hello\a");
```

What will happen?

```
Hello
```

After hello System Sound Beep will be started.