

C programming Class-1

It can be defined by the following ways:

1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

C as a mother language

1. C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.
2. It provides the core concepts like the array, strings, functions, file handling, etc. that are being used in many languages like C++, Java, C#, etc.

C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

It can't be used for internet programming like Java, .Net, PHP, etc.

C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.

A procedural language breaks the program into functions, data structures, etc. C is a procedural language. In C, variables and function prototypes must be declared before being used.

C as a structured programming language

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.

In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

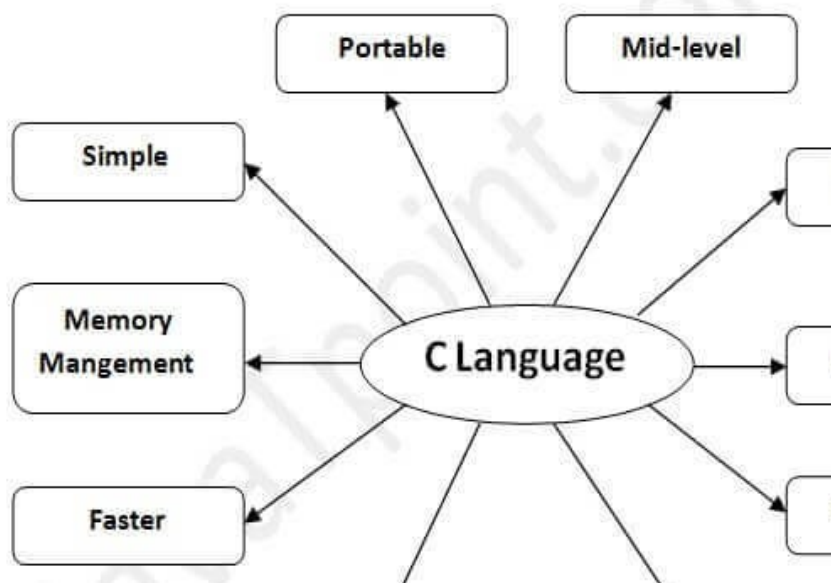
C as a mid-level programming language

C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).

A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.

A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

Features of C Language



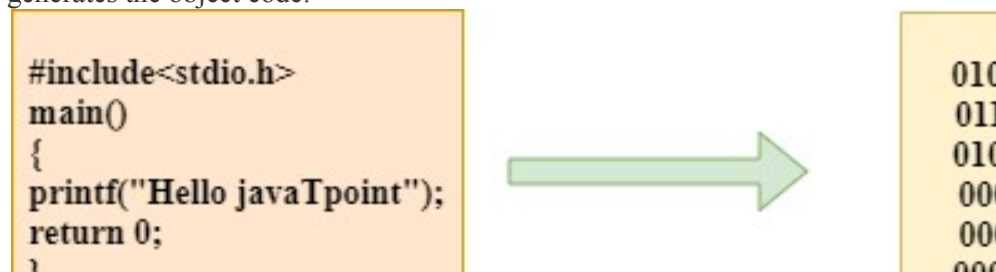
CHARACTERISTICS OF A C PROGRAM

- ✓ Middle level language.
- ✓ Small size – has only 32 keywords
- ✓ Extensive use of function calls- enables the end user to add their own functions to the C library.
- ✓ Supports loose typing – a character can be treated as an integer and vice versa.
- ✓ Structured language
- ✓ Low level (Bit Wise) programming readily available
- ✓ Pointer implementation extensive use of pointers for memory, array, structures and functions.
- ✓ It has high-level constructs.
- ✓ It can handle low-level activities.
- ✓ It produces efficient programs.
- ✓ It can be compiled on a variety of computers.

Compilation process in c

What is a compilation?

The compilation is a process of converting the source code into object code. It is done with the help of the compiler. The compiler checks the source code for the syntactical or structural errors, and if the source code is error-free, then it generates the object code.



The c compilation process converts the source code taken as input into the object code or machine code. The compilation process can be divided into four steps, i.e., Pre-processing, Compiling, Assembling, and Linking.

The preprocessor takes the source code as an input, and it removes all the comments from the source code. The preprocessor takes the preprocessor directive and interprets it. For example, if **<stdio.h>**, the directive is available in the program, then the preprocessor interprets the directive and replace this directive with the content of the '**stdio.h**' file.

The following are the phases through which our program passes before being transformed into an executable form:

- **Preprocessor**
- **Compiler**
- **Assembler**
- **Linker**

Preprocessor

The source code is the code which is written in a text editor and the source code file is given an extension ".c". This source code is first passed to the preprocessor, and then the preprocessor expands this code. After expanding the code, the expanded code is passed to the compiler.

Compiler

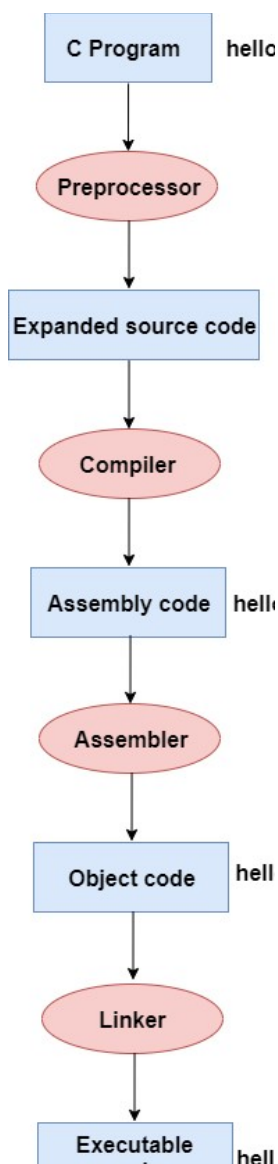
The code which is expanded by the preprocessor is passed to the compiler. The compiler converts this code into assembly code. Or we can say that the C compiler converts the pre-processed code into assembly code.

Assembler

The assembly code is converted into object code by using an assembler. The name of the object file generated by the assembler is the same as the source file. The extension of the object file in DOS is '.obj,' and in UNIX, the extension is '.o'. If the name of the source file is '**hello.c**', then the name of the object file would be 'hello.obj'.

Linker

Mainly, all the programs written in C use library functions. These library functions are pre-compiled, and the object code of these library files is stored with '.lib' (or '.a') extension. The main working of the linker is to combine the object code of library files with the object code of our program. Sometimes the situation arises when our program refers to the functions defined in other files; then linker plays a very important role in this. It links the object code of these files to our program. Therefore, we conclude that the job of the linker is to link the object code of our program with the object code of the library files and other files. The output of the linker is the executable file. The name of the executable file is the same as the source file but differs only in their extensions. In DOS, the extension of the executable file is '.exe', and in UNIX, the executable file can be named as '.out'. For example, if we are using printf() function in a program, then the linker adds its associated code in an output file.



- Firstly, the input file, i.e., **hello.c**, is passed to the preprocessor, and the preprocessor converts the source code into expanded source code. The extension of the expanded source code would be **hello.i**.
- The expanded source code is passed to the compiler, and the compiler converts this expanded source code into assembly code. The extension of the assembly code would be **hello.s**.
- This assembly code is then sent to the assembler, which converts the assembly code into object code.
- After the creation of an object code, the linker creates the executable file. The loader will then load the executable file for the execution.

Which of the following languages does not need any translation? [Com (o-IT)-2020]

- a) Machine language b) 4GL
- b) 3GL d) Assembly language

Which type of following errors is generated when the program is being executed? [Com (o-IT)-2020]

- a) Syntax error
- b) Semantic error
- c) Run-time error
- d) Linker error

Explanation: Syntax error generated in compiles time.

==. **Identify the correct sequence of steps to run a program**

- a) link, load, code, compile and execute b) code, compile, link, execute and load
- c) Code, compile, link, load and execute d) compile, code, link, load and execute Ans.: c

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Types of Variables in C

There are many types of variables in c:

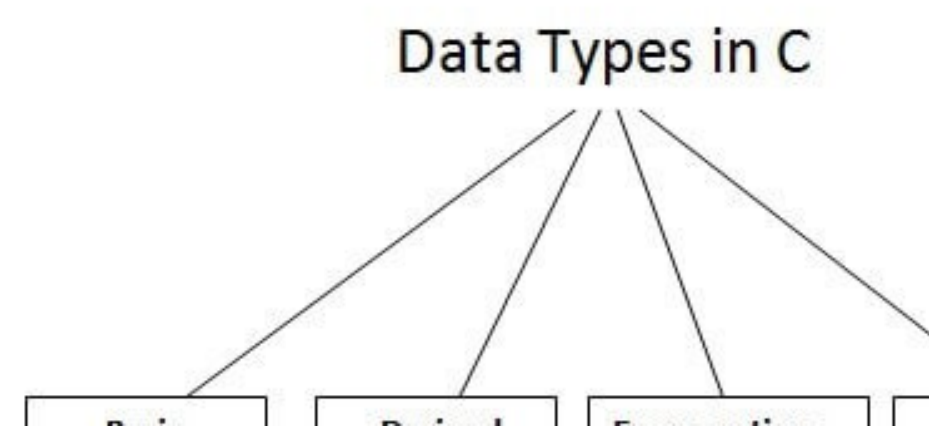
- 1. local variable
- 2. global variable
- 3. static variable : A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls. It can't change its value if you call it multiple times.

- 4. automatic variable
- 5. external variable

Data Types in C

A data type specifies the type of data that a variable can store such as integer, floating, character, etc.



here are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Which one is the not basic data type of c programming?[SBL (AP)-2016]

- a) Char
- b) int
- c) void
- d) None of this
- Ans.:

What is the output of the following program?[SBL (AP)-2016]

```
int main ()
{
printf("%.3f", 8/((3*8)*3));
return 0;
}
```

- a) 0.000
- b) 1.000
- c) 0.111
- d) None
- Ans.:

Keyword in c :

A keyword is a **reserved word**. You cannot use it as a variable name, constant name, etc. There are only 32 reserved words (keywords) in the C language.

A list of 32 keywords in the c language is given below:

auto	break	case	char	const	continue	default
double	else	enum	extern	float	for	goto
int	long	register	return	short	signed	sizeof

What is the output of following program: [ICML (AP)-2019]

```
int main ()
{
printf ("%d\t", sizeof (6.5));
printf ("%d\t", sizeof (90000));
printf ("%d\t", sizeof ("A"));
printf ("%d\t", sizeof ('x'));
return 0;
}
```

Ans.: 8 4 2 4

What is the output of the following program?

```
#include<stdio.h>
int main()
{
printf ("%d %d%d", sizeof (3.14f) , sizeof (3.14) , sizeof (3.1414) ) ;
return 0;
}
```

Ans.: 4 8 8 (sizeof(3.14f) is consider as float then next two are consider as double)

Rules for constructing C identifiers

- The first character of an identifier should be either an alphabet or an underscore, and then it can be followed by any of the character, digit, or underscore.
- It should not begin with any numerical digit.
- In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
- Commas or blank spaces cannot be specified within an identifier.
- Keywords cannot be represented as an identifier.
- The length of the identifiers should not be more than 31 characters.
- Identifiers should be written in such a way that it is meaningful, short, and easy to read.

Some Example of valid and invalid identifier

Valid identifiers	Invalid identifiers
RollNo	2name
Roll_No	Roll No
_Roll_No	int void
rollno	
Year	

==. Which of the declaration is correct? [ICB(AP)-2017] [Combined (Officer- IT/ICT)-2019]

- a) int length
- b)char int
- c) int long
- d) float double
- Ans.: a

C Format Specifier

%d or %i	It is used to print the signed integer value where signed integer means that th can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer mean variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always s a 0 value.
%x	It is used to print the hexadecimal unsigned integer where the hexadecim value always starts with a 0x value. In this, alphabetical characters are printe letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alp characters in uppercase such as A, B, C, etc.

Which Format Specifier is used for typing double data? [BREB 2019]

- a) %f
- b)%lf
- c)%d
- d)%s
- Ans.: b

Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc.

```
#include<stdio.h>

int main(){

const float PI=3.14;

PI=4.5;

printf("The value of PI is: %f",PI);

return 0;
```


Compile Time Error: Cannot modify a const

The value 9.87 to 10 when use?^[BB(AP)-2016]
a) floor () b) ceil() c) both d) None **Ans.: b**

What is the output of the following program?

```
int main()
{
    int f=11,i=3;
    i+=(f>3)? i&2:5;
    printf("%d ",i);
    return 0;
}
```

Ans.: 5

 **Explanation:**
i+=(f>3)? i&2:5
i+= (11>3)? 3&2 :5 [so statement(f>3) is true and i&2 executed]
i+=3&2 [0011 & 0010 =0010 =2]
i+=2
5

CONDITIONAL OR TERNARY OPERATORS IN C:
Syntax: (Condition? true_value: false_value);
Example: (A >100? 0 : 1);
int x=1, y ;
y = (x ==1 ?2 : 0) ;

output: 2

Example:
int a=100,b=200,c=300,c;
x= (a>b)? ((a>c)?a:c) : ((b>c)?b:c) ;
1st part 2nd part 3rd part

Explanation:

Here a> b is false so 3rd part is working

In 3rd part, b>c also false.soC is working

Output: x=300

```
int main ()
{
    unsigned int a = 29;          /* 29 = 0001 1101 */
    unsigned int b = 48;          /* 48 = 0011 0000 */
    int c = 0;
    c = a & b;                     /* 0001 1101 & 0011 0000=10000=16 */
    printf ("%d & %d = %d\n", a, b, c);
    c = a | b;                     /* 61 = 0011 1101 */
    printf ("%d | %d = %d\n", a, b, c);
    c = a ^ b;                     /* Xor operation: 45 = 0010 1101 */
    printf ("%d ^ %d = %d\n", a, b, c);
    c = ~a;                        /* -30 = 1110 0010 */
    printf ("~%d = %d\n", a, c);
```

```

    c = a << 2;      /* left shift: 0001 1101<<2= 01110100= 116 */
printf ("%d << 2 = %d\n", a, c);
    c = a >> 2;      /* Right Shift:0001 1101>>2= 0000111= 7 */
printf ("%d >> 2 = %d\n", a, c);
    return 0;
}

```

==. What is the final values of a and c in the following C statement? *[Combined AME-2019]*

(initialize value a=2,c=1) c=c? a=0:2;

a) a=0,c=0 b) a=2,c=2 c) a=2,c=2 d) a=1,c=2 **Ans. a**