Unit – 1

Computer Language

As we know, to communicate with a person, we need a specific language, similarly to communicate with computers, programmers also need a language is called Programming language.

What is Language?

Language is a mode of communication that is used to **share ideas, opinions with each other**. For example, if we want to teach someone, we need a language that is understandable by both communicators.

What is a Programming Language?

A programming language is a **computer language** that is used by **programmers (developers) to communicate with computers**. It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

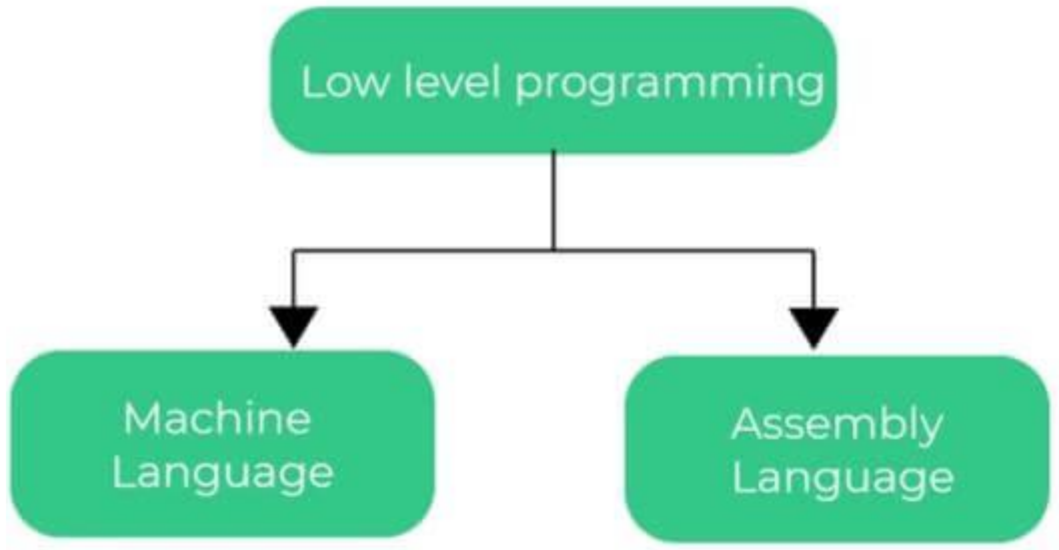
Types of programming language

1. Low-level programming language

Low-level language is **machine-dependent (0s and 1s)** programming language. The processor runs low- level programs directly without the need of a compiler or interpreter, so the programs written in low-level language can be run very fast.

Low-level language is further divided into two parts –

Low Level Programming In C Language



i. Machine Language

The machine-level language is a language that consists of a set of instructions that are in the binary form 0 or 1. As we know that computers can understand only machine instructions, which are in binary digits, i.e., 0 and 1, so the instructions given to the computer can be only in binary codes. Creating a program in a machine-level language is a very difficult task as it is not easy for the programmers to write the program in machine instructions. It is error-prone as it is not easy to understand, and its maintenance is also very high. A machine-level language is not portable as each computer has its machine instructions, so if we write a program in one computer will no longer be valid in another computer.

Advantages

1. Fast execution
2. No need of translator

Disadvantages

1. Difficult to learn
2. Difficult to find errors

3. Difficult to modify
4. Machine dependent

ii. Assembly Language

The assembly language contains some human-readable commands such as mov, add, sub, etc. Since assembly language instructions are written in English words like mov, add, sub, so it is easier to write and understand. As we know that computers can only understand the machine-level instructions, so we require a translator that converts the assembly code into machine code. The translator used for translating the code is known as an assembler.

The assembly language code is not portable because the data is stored in computer registers, and the computer has to know the different sets of registers.

Advantages

1. Easy to learn
2. Easy to find errors
3. Easy to modify

Disadvantages

1. Slow Execution
2. Limited Mnemonics
3. Machine Dependent

2. High-level programming language

The high-level language is a programming language that allows a programmer to write the programs which are independent of a particular type of computer. The high-level languages are considered as high-level because they are closer to human languages than machine-level languages.

When writing a program in a high-level language, then the whole attention needs to be paid to the logic of the problem. A compiler is required to translate a high-level language into a low-level language.

Advantages

1. High level languages are programmer friendly. They are easy to write, debug and maintain.

2. It is machine independent language.
3. Easy to learn.
4. Less error prone, easy to find and debug errors.
5. High level programming results in better programming productivity.

Disadvantages

1. Slow Execution
2. Required a translator
3. High Memory required

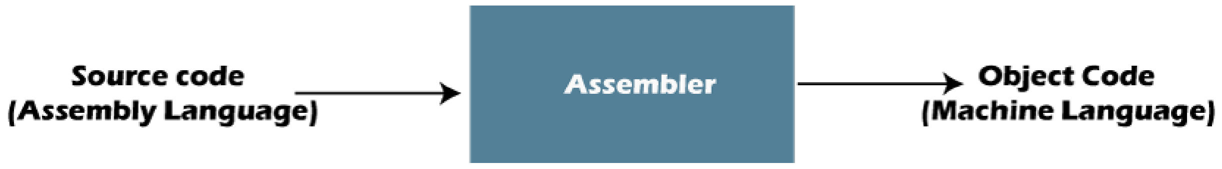
Translators

Mostly languages like Java, C++, Python, Assembly and more are used to write the programs, called source code. These source codes need to translate into machine language to be executed because they cannot be executed directly by the computer. Hence, a special translator program, a language processor, is used to convert source code into machine language.

Types of language processors (Translators)

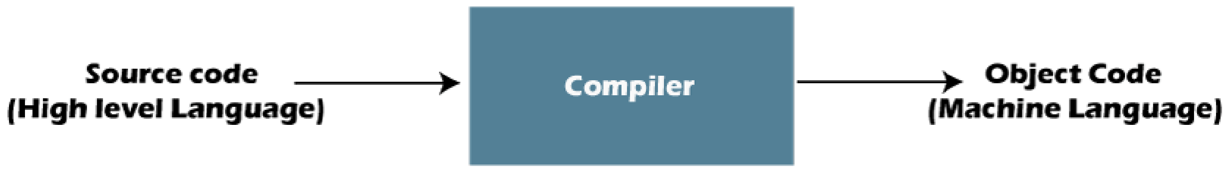
There are mainly three kinds of language processors, which are discussed below:

1. **Assembler:** An assembler converts programs written in assembly language into machine code. It is also referred to assembler as assembler language by some users. The source program has assembly language instructions, which is an input of the assembler. The assemble translates this source code into a code that is understandable by the computer, called object code or machine code.

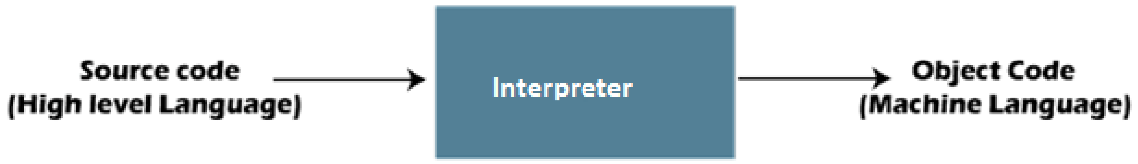


2. **Compiler:** The language processor allows the computer to run and understand the program by reading the complete source program in one time, which is written in a high-level language. The computer can then interpret this code because it is translated into machine language. While working on the Harvard

Mark I computer, Grace Hopper created the first compiler.



3. **Interpreter:** An interpreter is a computer program that allows a computer to interpret or understand what tasks to perform. The programs written with the help of using one of the many high-level programming languages are directly executed by an interpreter without previously converting them to an object code or machine code, which is done line by line or statement by statement. When the interpreter is translating the source code, it displays an error message if there is an error in the statement and terminates this statement from translating process. When the interpreter removed errors on the first line, then it moves on to the next line.



Introduction to C Language

The C Language is developed by Dennis Ritchie for creating system applications that directly interact with the hardware devices such as drivers, kernels, etc. C programming is considered as the base for other programming languages, that is why it is known as mother language.

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

History of C Language

C programming language was developed in 1972 by Dennis Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

Dennis Ritchie is known as the **founder of the c language**. It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in **UNIX operating system**.

These programming languages that were developed before C language -

Language	**Year**	**Developed By**
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie
K & R C	1978	Kernighan & Dennis Ritchie
ANSI C	1989	ANSI Committee
ANSI/ISO C	1990	ISO Committee
C99	1999	Standardization Committee
C11	2011	Standardization Committee
C18	2018	Standardization Committee

Features of C Language

1. **Simple and Efficient**

C is a simple language in the sense that it provides a **structured approach** (to break the problem into parts), **the rich set of library functions**, **data types**, etc.

2. Portability

Unlike assembly language, c programs **can be executed on different machines** with some machine specific changes. Therefore, C is a machine independent language or portable language.

3. Structured Programming language

C is a structured programming language in the sense that **we can break the program into parts using functions**. So, it is easy to understand and modify. Functions also provide code reusability.

4. Rich Standard Library

C **provides a lot of inbuilt functions** that make the development fast.

5. Mid-level programming language

Although, C is **intended to do low-level programming**. It is used to develop system applications such as kernel, driver, etc. It **also supports the features of a high-level language**. That is why it is known as mid-level language.

6. Memory Management

It supports the feature of **dynamic memory allocation**. In C language, we can free the allocated memory at any time by calling the **free()** function.

7. Pointer

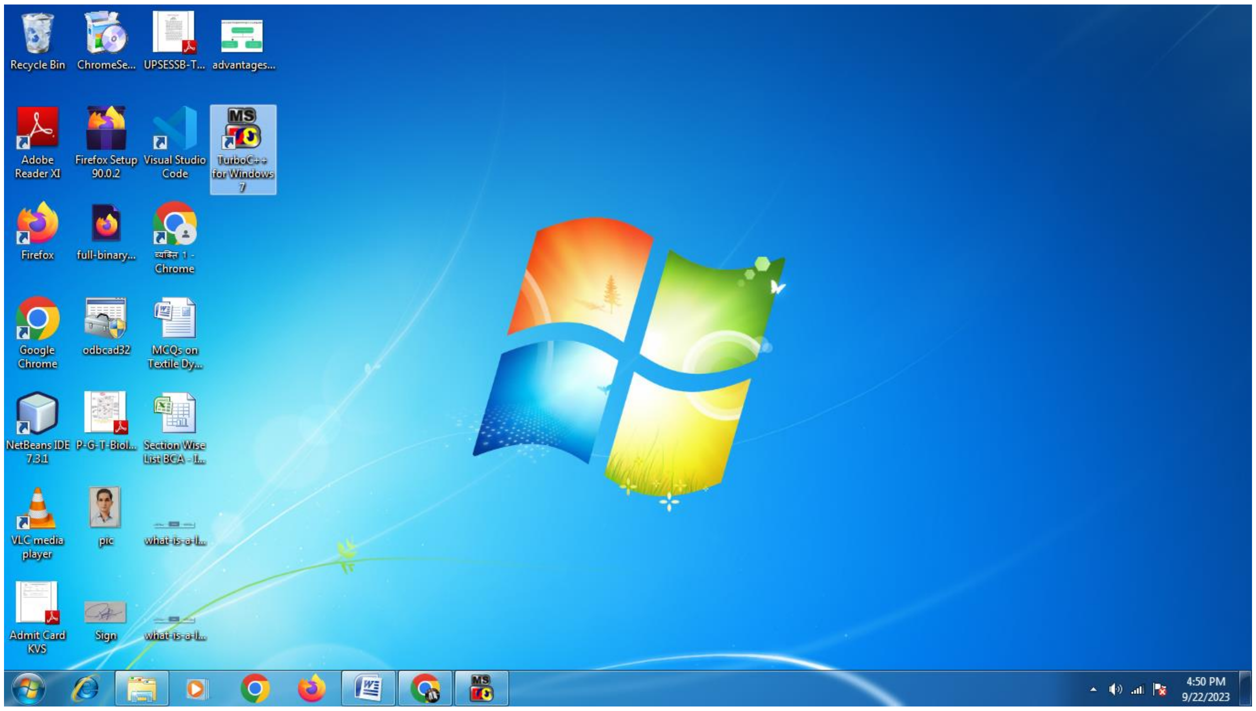
C provides the feature of pointers. We can directly interact with the memory by using the pointers. We **can use pointers for memory, structures, functions, array**, etc.

8. Recursion

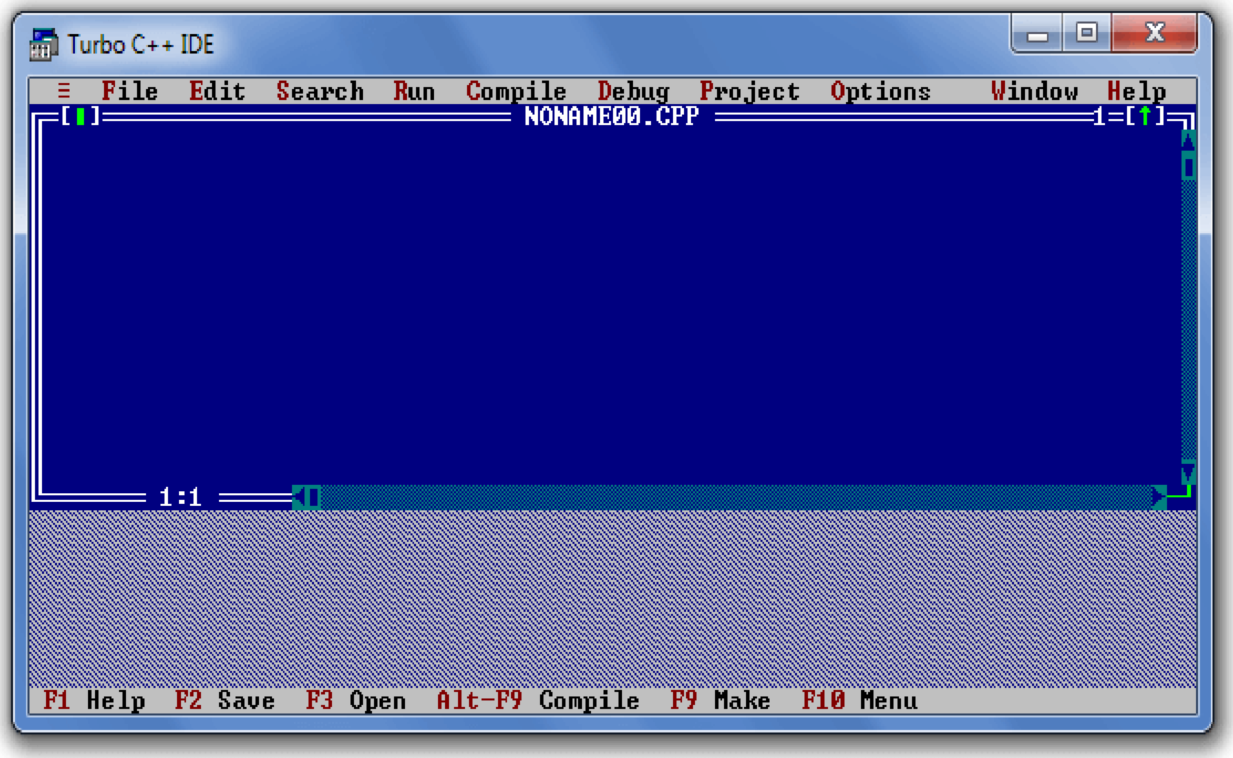
In C, we **can call the function within the function**. It provides code reusability for every function. Recursion enables us to use the approach of backtracking.

How to open Turbo C++

Double click on Tutbo C++ icon from the desktop.



It open Turbo C++ IDE as-


Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP
1:1
F1 Help F2 Save F3 Open Alt-F9 Compile F9 Make F10 Menu

First C Program

To write the first c program, open the Turbo C++ IDE and write the following code:

```
#include <stdio.h>
void main()
{
printf("Hello C Language");
}
```

Before compile the program, it must be saved with any name but with extension .c-

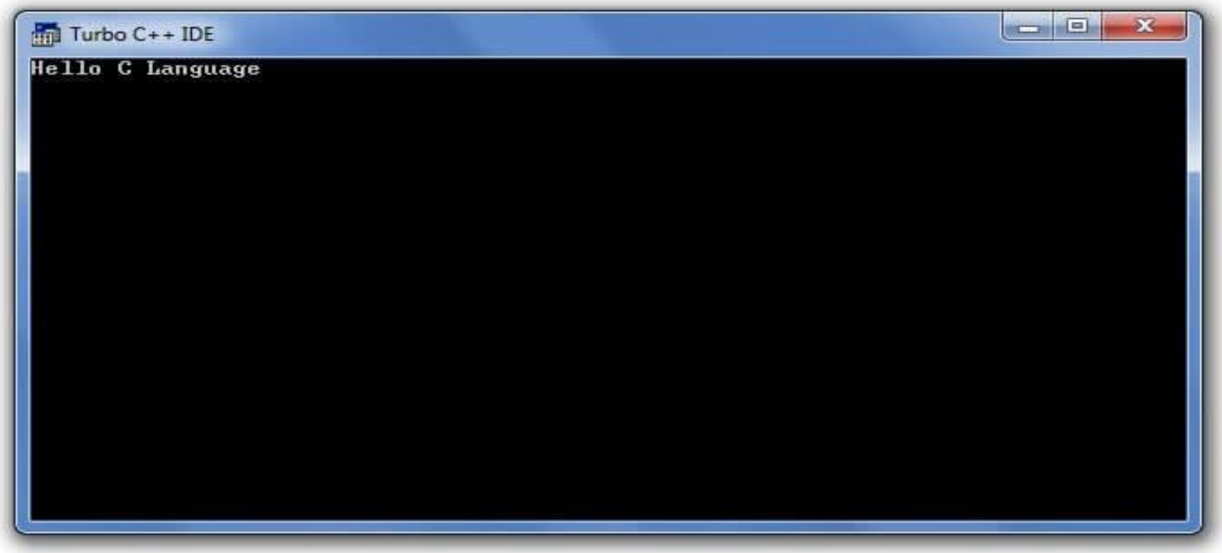
Save the Program : Demo.c

To Compile: Press Alt + F9

To Run: Ctrl + F9

To Show Output: Alt + F5

You will see the following output on user screen.



```
Hello C Language
```

Now **press any key** to return to the turbo c++ screen.

Here,

#include <stdio.h> includes the **standard input output** library functions. The printf() function is defined in stdio.h .

void main() The **main() function is the entry point of every program** in c language.

printf() The printf() function is **used to print data** on the console screen.

Other Examples

Program 1: WAP to print “Hello C” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Hello C");
getch();
}
```

Output:

```
Hello C
```

Program 2: WAP to print Your Name

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Pramod Kumar");
getch();
}
```

Output:

```
Pramod Kumar
```

Program 3: WAP to print “Hello How Are You” Message

```
#include <stdio.h>
#include <conio.h>
```

```
void main()
{
clrscr();
printf("Hello How Are You");
getch();
}
```

Output:

```
Hello How Are You
```

Program 4: WAP to print “C is a Middle Level Language” Message

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("C is a Middle Level Language");
getch();
}
```

Output:

```
C is a Middle Level Language
```

Program 5: WAP to print a Message as-
Hello
How
Are

You

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Hell0");
printf("\n How");
printf("\n Are");
printf("\n You");
getch();
}
```

Output:

```
Hello

How

Are

You
```

Program 6: WAP to print a Message as-
**Hello How	Are	You**

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Hell0");
printf("\t How");
printf("\t Are");
printf("\t You");
getch();
}
```

Output:

```
Hello       How        Are        You
```

Program 6: WAP to print a Message as-

```
Hello
      How
            Are
                  You
```

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
printf("Hell0");
printf("\n\t How");
printf("\n\t\t Are");
printf("\n\t\t\t You");
getch();
}
```

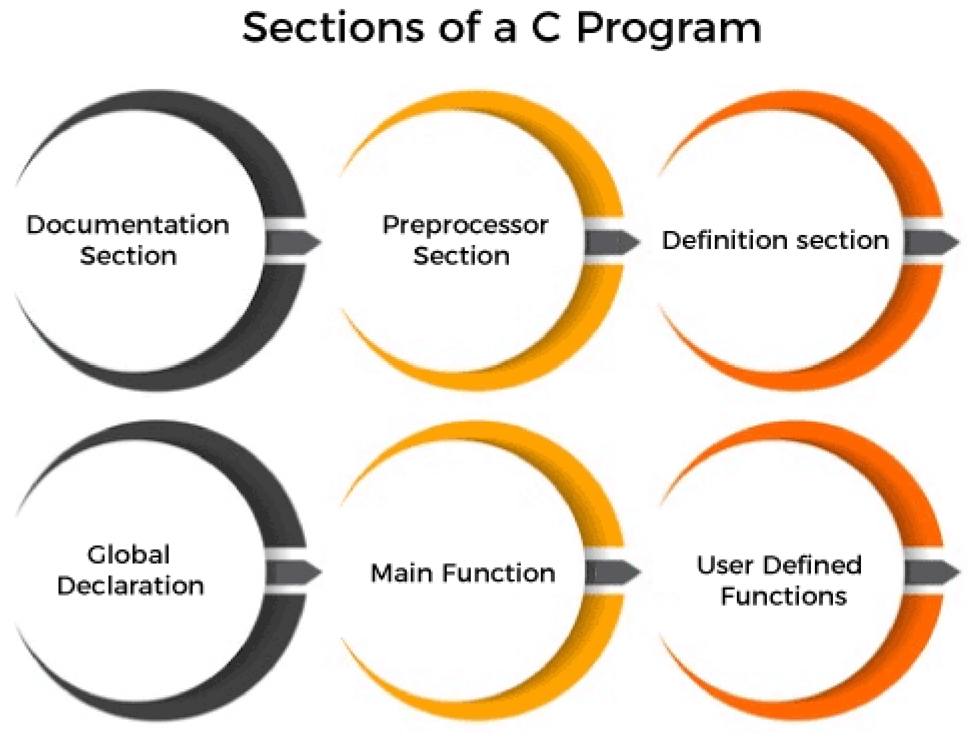
Output:

```
Hello
        How
                Are
                        You
```

Structure of a C program

The structure of a C program means the specific structure to start the programming in the C language. Without a proper structure, it becomes difficult to analyze the problem and the solution. It also gives us a reference to write more complex programs.

Sections of a C program



The sections of a C program are listed below:

1. Documentation section
2. Preprocessor section
3. Definition section
4. Global declaration
5. Main function
6. User defined functions

1. Documentation section

It includes the statement specified at the beginning of a program, such as a program's **name, date, description,** and **title**. It is represented as:

```
//name of a program

Or

/*
Overview of the code
…………..
…………..
*/
```

2. Preprocessor section (Linking Section)

The preprocessor section contains all the header files used in a program. It informs the system to link the header files to the system libraries. It is given by:

```
#include<stdio.h>
#include<conio.h>
```

3.Define section (Declaration Section)

The define section comprises of different constants declared using the define keyword. It is given by:

```
#define a = 2
#define PI = 3.14
void sum();
int result();
```

4. Global declaration

The global section comprises of all the global declarations in the program. It is given by:

```
float num = 2.54;
 int a = 5;
char ch ='z';
```

5. Main function

main() is the first function to be executed by the computer. It is necessary for a code to include the main(). It is like any other function available in the C library.

```
main()
```

We can also use int or main with the main (). The void main() specifies that the program will not return any value. The int main() specifies that the program can return integer type data.

```
int main()
```

Or

```
void main()
```

6. User defined functions

The user defined functions specified the functions specified as per the requirements of the user. For example, color(), sum(), division(), etc.

```
void sum()
{
…….
……
…….
}
```

Compilation process in c

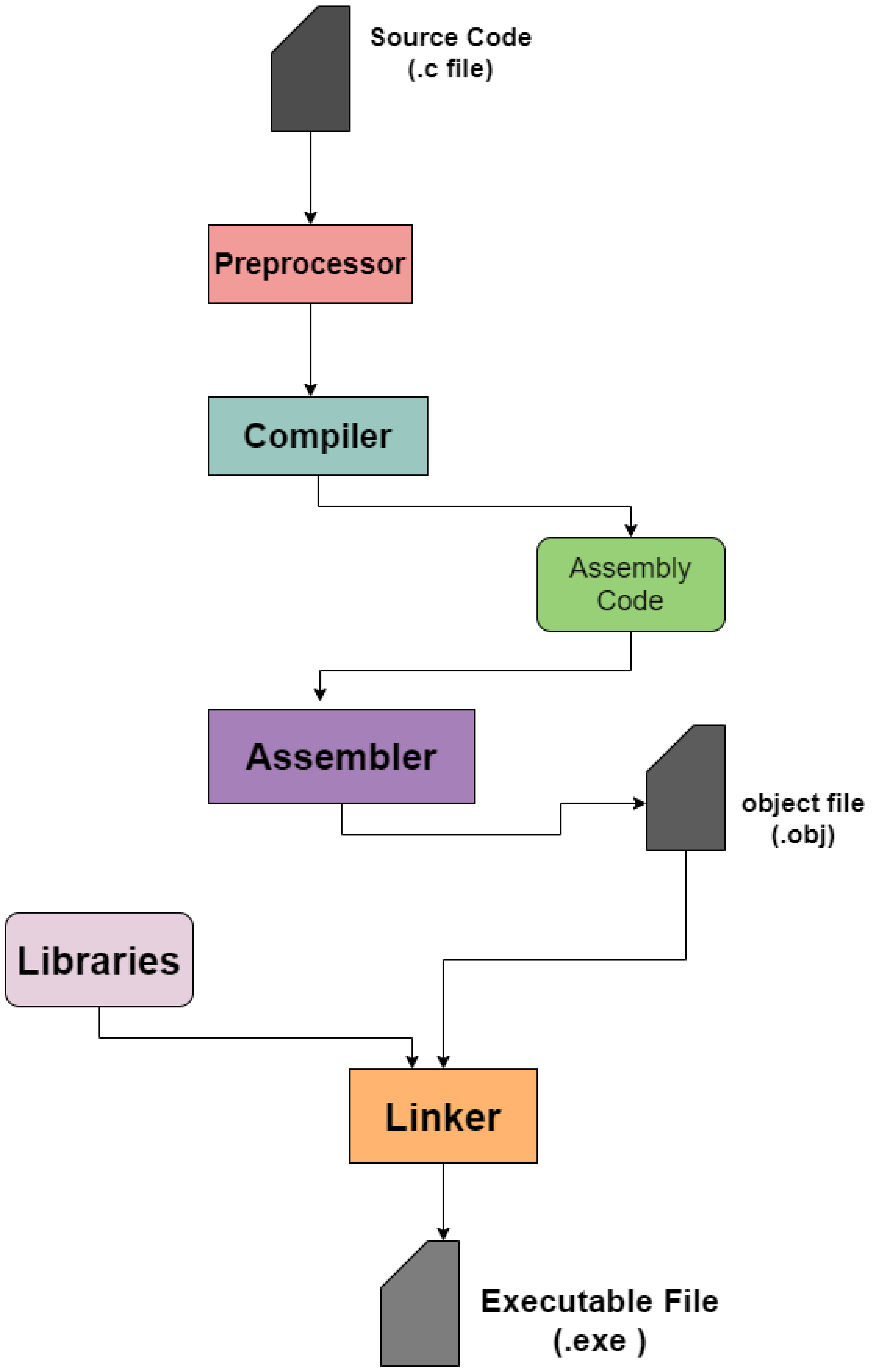
What is a compilation?

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

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


Source Code
(.c file)
Preprocessor
Compiler
Assembly Code
Assembler
object file
(.obj)
Libraries
Linker
Executable File
(.exe)

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,' 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' extension. The main working of the linker is to combine the object code of library files with the object code of our program.

C Character Set

As every language contains a set of characters used to construct words, statements, etc., C language also has a set of characters which include alphabets, digits, and special symbols. C language supports a total of 256 characters.

Every C program contains statements. These statements are constructed using words and these words are constructed using characters from C character set. C language character set contains the following set of characters...

1. Alphabets
2. Digits
3. Special Symbols

Alphabets

The C programming language provides support for all the alphabets that we use in the English language. Thus, in simpler words, a C program would easily support a total of 52 different characters- 26 uppercase and 26 lowercase.

Type of Character	Description	Characters
Lowercase Alphabets	a to z	a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z
Uppercase Alphabets	A to Z	A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z

Digits

The C programming language provides the support for all the digits that help in constructing/ supporting the numeric values or expressions in a program. These range from 0 to 9. Thus, the C language supports a total of 10 digits for constructing the numeric values or expressions in any program.

Type of Character	Description	Characters
Digits	0 to 9	0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Special Characters

We use some special characters in the C language for some special purposes, such as logical operations, mathematical operations, checking of conditions, backspaces, white spaces, etc.

The C programming language provides support for the following types of special characters:

Type of Character	Examples
Special Characters	` ~ @ ! $ # ^ * % & ( ) [ ] { } < > + = _ – | / \ ; : ‘ “ , . ?

White Spaces

The white spaces in the C programming language contain the following:

- Blank Spaces
- Carriage Return
- Tab
- New Line

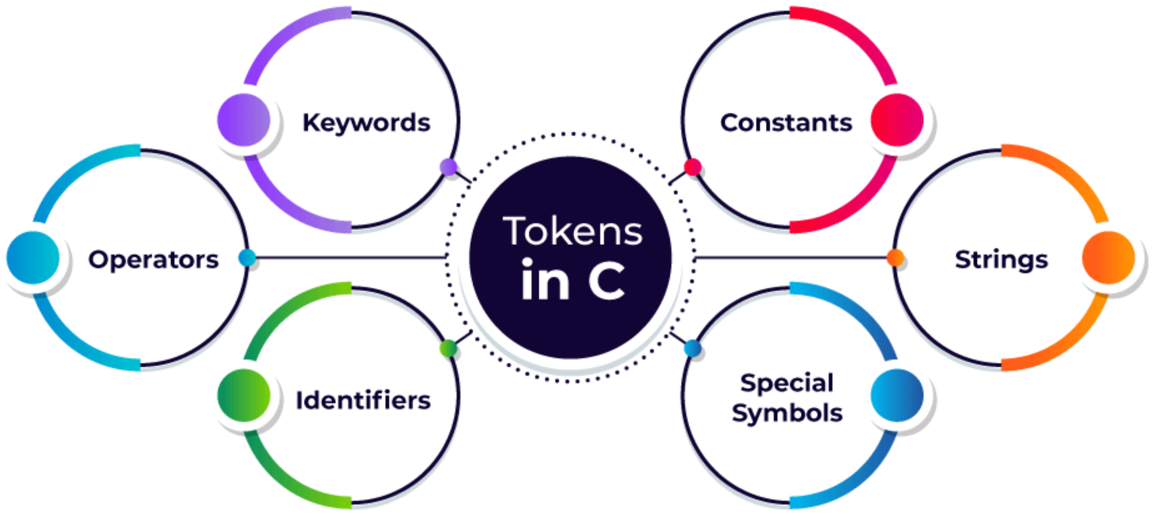
Tokens in C

Tokens in C is the most important element to be used in creating a program in C. We can define the token as the smallest individual element in C. For `example, we cannot create a sentence without using words; similarly, we cannot create a program in C without using tokens in C. Therefore, we can say that tokens in C is the building block or the basic component for creating a program in C language.

Types of tokens in C

Tokens in C language can be divided into the following categories:

`



Keywords

Keywords in C can be defined as the **pre-defined** or the **reserved words** having its own importance, and each keyword has its own functionality. C language supports 32 keywords given below:

auto	double	int	struct
break	else	long	switch

case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Identifiers

Identifiers in C are used for naming variables, functions, arrays, structures, etc. Identifiers in C are the user-defined words. It can be composed of uppercase letters, lowercase letters, underscore, or digits, but the starting letter should be either an underscore or an alphabet. Identifiers cannot be used as keywords. Rules for constructing identifiers in C are given below:

1. The first character of an identifier should be either an alphabet or an underscore.
2. It should not begin with any numerical digit.
3. In identifiers, both uppercase and lowercase letters are distinct. Therefore, we can say that identifiers are case sensitive.
4. Commas or blank spaces cannot be specified within an identifier.
5. Keywords cannot be represented as an identifier.
6. The length of the identifiers should not be more than 31 characters.

Strings

Strings in C are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string. Strings in C are enclosed within double quotes, while characters are enclosed within single characters. The size of a string is a number of characters that the string contains.

Now, we describe the strings in different ways:

char a[10] = "Welcome"; // The compiler allocates the 10 bytes to the 'a' array.

char a[] = "Welcome"; // The compiler allocates the memory at the run time.

char a[10] = {'W','e','l','c','o','m','e','\0'}; // String is represented in the form of characters.

Operators

Operators in C is a special symbol used to perform the functions. The data items on which the operators are applied are known as operands. Operators are applied between the operands. Depending on the number of operands, operators are classified as follows:

Unary Operator

A unary operator is an operator applied to the single operand. For example: increment operator (++), decrement operator (--), sizeof, (type)*.

Binary Operator

The binary operator is an operator applied between two operands. The following is the list of the binary operators:

- Arithmetic Operators
- Relational Operators
- Shift Operators
- Logical Operators

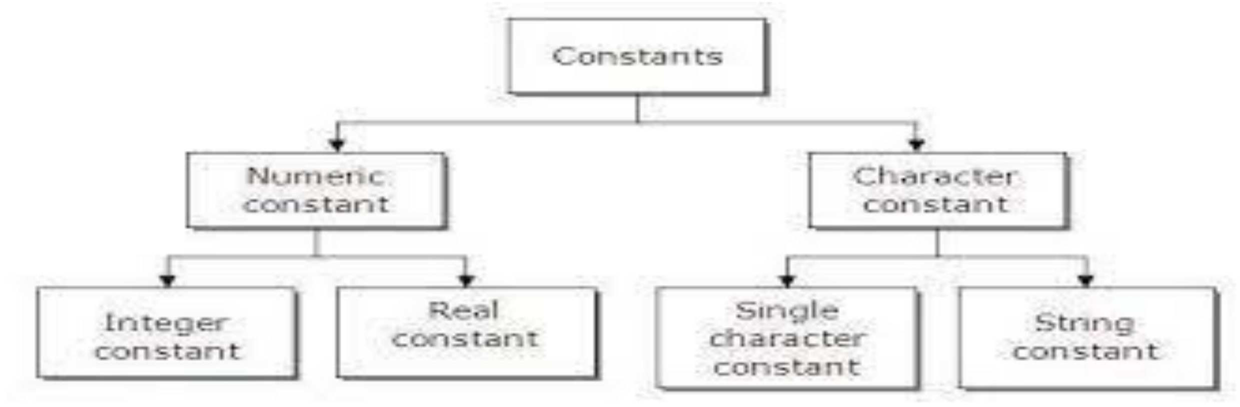
- Bitwise Operators
- Conditional Operators
- Assignment Operator

Constants

A constant is a value assigned to the variable which will remain the same throughout the program, i.e., the constant value cannot be changed.

There are two ways of declaring constant:

- Using const keyword
- Using #define pre-processor



Constant	Example
Integer constant	10, 11, 34, etc.
Floating-point constant	45.6, 67.8, 11.2, etc.
Octal constant	011, 088, 022, etc.
Hexadecimal constant	0x1a, 0x4b, 0x6b, etc.
Character constant	'a', 'b', 'c', etc.

String constant	"java", "c++", ".net", etc.

Special characters

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling. For example, printf() is a pre-defined function.
- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement and for example, separating function parameters in a function call.
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Tilde (~):** It is used as a destructor to free memory.
- **Period (.):** It is used to access a member of a structure or a union.

Variables in C

A **variable** is a name of the memory location. It is used to store data. Its value can be changed, and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Syntax to declare a variable:

datatype variablename;

Example of declaring the variable is given below:

int a;
float b;
char c;

Here, a, b, c are variables. The int, float, char are the data types.

We can also declare multiple variables at the same time as:

int a, b, c;

float x, y;

Initialization of variables:

A=10;

B=20.56;

C='x';

We can also declare and initialize a variable together as-
int a=10;

float b=20.56;

char c='A';

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.

Valid variable names:

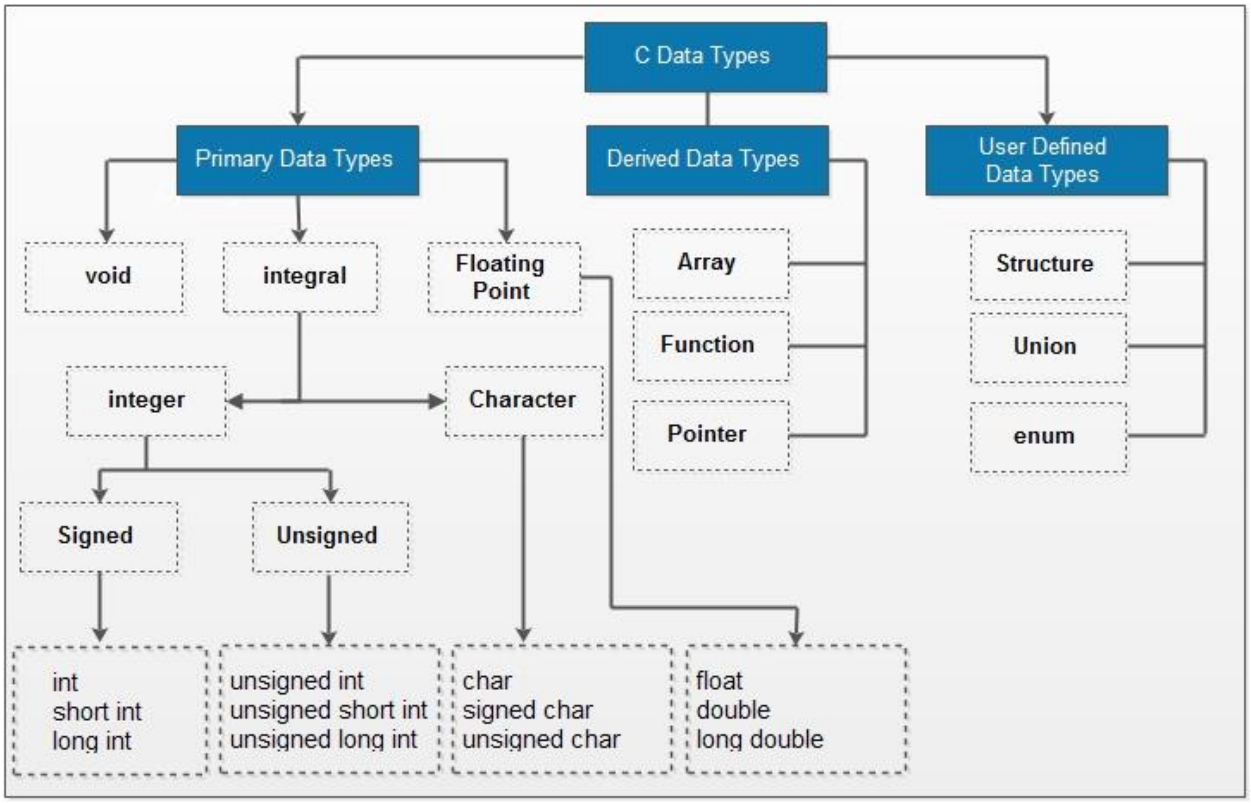
int a;
int _ab;
int a30;

Invalid variable names:

int 2;
int a b;
int long;

Datatypes in c

Each variable in C has an associated data type. It specifies the type of data that the variable can store like integer, character, floating, double, etc. Each data type requires different amounts of memory and has some specific operations which can be performed over it. The data type is a collection of data with values having fixed values, meaning as well as its characteristics.



There 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

Basic Data Types

The basic data types are integer-based and floating-point based. C language supports both signed and unsigned literals.

The memory size of the basic data types may change according to 32 or 64-bit operating system.

Let's see the basic data types. Its size is given **according to 32-bit architecture**.

Data Types	Memory Size	Range
Char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
Short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
Int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535

short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
Float	4 byte	3.4E-38 to 3.4E+38
double	8 byte	1.7E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

Format Specifier

The Format specifier is a string used in the formatted input and output functions. The format string determines the format of the input and output. The format string always starts with a '%' character.

The commonly used format specifiers in printf() function are:

Format specifier	**Description**
%d or %i	It is used to print the signed integer value where signed integer means that the variable can hold both positive and negative values.
%u	It is used to print the unsigned integer value where the unsigned integer means that the variable can hold only positive value.
%o	It is used to print the octal unsigned integer where octal integer value always starts with a 0 value.

%x	It is used to print the hexadecimal unsigned integer where the hexadecimal integer value always starts with a 0x value. In this, alphabetical characters are printed in small letters such as a, b, c, etc.
%X	It is used to print the hexadecimal unsigned integer, but %X prints the alphabetical characters in uppercase such as A, B, C, etc.
%f	It is used for printing the decimal floating-point values. By default, it prints the 6 values after '.'.
%e/%E	It is used for scientific notation. It is also known as Mantissa or Exponent.
%g	It is used to print the decimal floating-point values, and it uses the fixed precision, i.e., the value after the decimal in input would be exactly the same as the value in the output.
%p	It is used to print the address in a hexadecimal form.
%c	It is used to print the unsigned character.
%s	It is used to print the strings.
%ld	It is used to print the long-signed integer value.

Escape Sequence in C

An ***escape sequence*** in the C programming language consists of a ***backslash (\ )*** and a character that stands in for a ***special character*** or ***control sequence***. During the compilation process, the ***C compiler*** substitutes any escape sequences it comes across with the ***relevant character*** or ***control sequence***.

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\'	Single Quote
\"	Double Quote
\?	Question Mark

Comments in C

Comments in C language are used to provide information about lines of code. It is widely used for documenting code. There are 2 types of comments in the C language.

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

Single line comments are represented by double slash \\. Let's see an example of a single line comment in C.

Example 1:

```
#include<stdio.h>
#include<conio.h>
void main()
{
	clrscr();
	printf("Pramod");
	// printf("\nKumar");
	printf("\nSharma");
	getch();
}
```

Output:

```
Pramod

Sharma
```

Example 2:

```
#include<stdio.h>
#include<conio.h>
void main()
{
	clrscr();
	printf("Pramod");
	// printf("\nKumar");
	// printf("\nSharma");
	getch();
}
```

Output:

```
Pramod
```

Mult Line Comments

Multi-Line comments are represented by slash asterisk * ... *\. It can occupy many lines of code, but it can't be nested. Syntax:

```
/*
code
to be commented
*/
```

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{
        clrscr();
        /* printf("Pramod");
        printf("\nKumar"); */
        printf("\nSharma");
        getch();
}
```

Output:

```
Sharma
```