**C Tutorial**

**What is C?**

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:31.4M

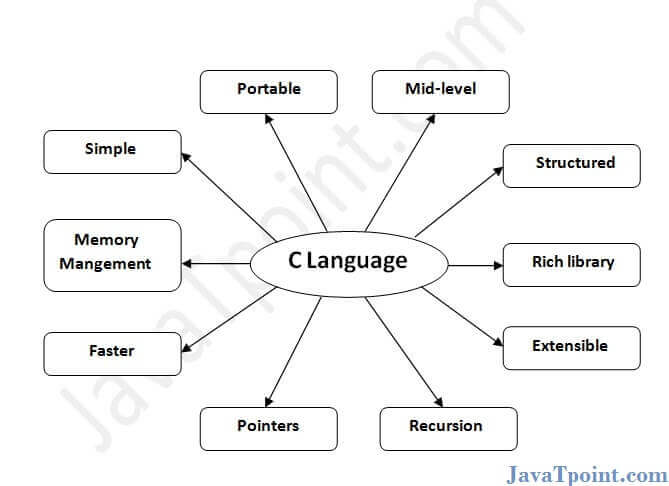
720

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

**Why to learn C?**

C is the widely used language. It provides many **features** that are given below.

1. Simple
2. Machine Independent or Portable
3. Mid-level programming language
4. structured programming language
5. Rich Library
6. Memory Management
7. Fast Speed
8. Pointers
9. Recursion
10. Extensible



**About this tutorial – Same as earlier course data**

**Table of Contents**

**Level 1:**

Installation

Variables

Data Types

Operators

Control Statements and Loops

**Level 2:**

Arrays

Pointer

Functions

Command Line Arguments

Strings

**Level 3:**

Structures

File Handling

Error Handling

**Level 1:**

1. **Installation**

There are many compilers available for c and c++. You need to download any one. Here, we are going to use **Turbo C++**. It will work for both C and C++. To install the Turbo C software, you need to follow following steps.

1. Download Turbo C++
2. Create turboc directory inside c drive and extract the tc3.zip inside c:\turboc
3. Double click on install.exe file
4. Click on the tc application file located inside c:\TC\BIN to write the c program

Before starting the abcd of C language, you need to learn how to write, compile and run the first c program.

To write the first c program, open the C console and write the following code:

#include <stdio.h>

**int** main(){

printf("Hello C Language");

**return** 0;

}

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

**int 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.

**return 0** The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution.

## How to compile and run the c program

There are 2 ways to compile and run the c program, by menu and by shortcut.

### By menu

Now **click on the compile menu then compile sub menu** to compile the c program.

Then **click on the run menu then run sub menu** to run the c program.

### By shortcut

**Or, press ctrl+f9** keys compile and run the program directly.

1. **Variables**

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.

Let's see the syntax to declare a variable:

type variable\_list;

The example of declaring the variable is given below:

10 Sec

Prime Ministers of India | List of Prime Minister of India (1947-2020)

**int** a;

**float** b;

**char** c;

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

We can also provide values while declaring the variables as given below:

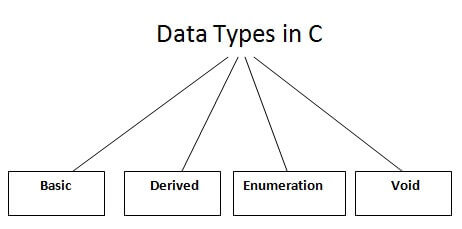
**int** a=10,b=20;//declaring 2 variable of integer type

**float** f=20.8;

**char** c='A';

1. **Data Types**

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



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 |

1. **Operators**

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise, etc.

There are following types of operators to perform different types of operations in C language.

* Arithmetic Operators
* Relational Operators
* Shift Operators
* Logical Operators
* Bitwise Operators
* Ternary or Conditional Operators
* Assignment Operator
* Misc Operator

**Please Insert the table of operators as per your convenience**

1. **Control Statements and Loops**

The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.

There are the following variants of if statement in C language.

* If statement
* If-else statement
* If else-if ladder
* Nested if

## If Statement

The if statement is used to check some given condition and perform some operations depending upon the correctness of that condition. It is mostly used in the scenario where we need to perform the different operations for the different conditions. The syntax of the if statement is given below.

**if**(expression){

//code to be executed

}

## If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simiulteneously. Using if-else statement is always preferable since it always invokes an otherwise case with every if condition. The syntax of the if-else statement is given below.

**if**(expression){

//code to be executed if condition is true

}**else**{

//code to be executed if condition is false

}

## If else-if ladder Statement

The if-else-if ladder statement is an extension to the if-else statement. It is used in the scenario where there are multiple cases to be performed for different conditions. In if-else-if ladder statement, if a condition is true then the statements defined in the if block will be executed, otherwise if some other condition is true then the statements defined in the else-if block will be executed, at the last if none of the condition is true then the statements defined in the else block will be executed. There are multiple else-if blocks possible. It is similar to the switch case statement where the default is executed instead of else block if none of the cases is matched.

**if**(condition1){

//code to be executed if condition1 is true

}**else** **if**(condition2){

//code to be executed if condition2 is true

}

**else** **if**(condition3){

//code to be executed if condition3 is true

}

...

**else**{

//code to be executed if all the conditions are false

}

# C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in [c language](https://www.javatpoint.com/c-programming-language-tutorial) is given below:

**switch**(expression){

**case** value1:

//code to be executed;

**break**; //optional

**case** value2:

//code to be executed;

**break**; //optional

......

**default**:

code to be executed **if** all cases are not matched;

}

## Types of C Loops

There are three types of loops in [C language](https://www.javatpoint.com/c-programming-language-tutorial) that is given below:

1. do while
2. while
3. for

### do-while loop in C

The do-while loop continues until a given condition satisfies. It is also called post tested loop. It is used when it is necessary to execute the loop at least once (mostly menu driven programs).

The syntax of [do-while loop in c language](https://www.javatpoint.com/do-while-loop-in-c) is given below:

**do**{

//code to be executed

}**while**(condition);

[Flowchart and Example of do-while loop](https://www.javatpoint.com/do-while-loop-in-c)

### while loop in C

The while loop in c is to be used in the scenario where we don't know the number of iterations in advance. The block of statements is executed in the while loop until the condition specified in the while loop is satisfied. It is also called a pre-tested loop.

The syntax of while loop in c language is given below:

**while**(condition){

//code to be executed

}

[Flowchart and Example of while loop](https://www.javatpoint.com/while-loop-in-c)

### for loop in C

The for loop is used in the case where we need to execute some part of the code until the given condition is satisfied. The for loop is also called as a per-tested loop. It is better to use for loop if the number of iteration is known in advance.

The syntax of for loop in c language is given below:

**for**(initialization;condition;incr/decr){

//code to be executed

}

# C break statement

The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The break statement in C can be used in the following two scenarios:

1. With switch case
2. With loop

### Syntax:

//loop or switch case

**break**;

# C continue statement

The **continue statement** in C language is used to bring the program control to the beginning of the loop. The continue statement skips some lines of code inside the loop and continues with the next iteration. It is mainly used for a condition so that we can skip some code for a particular condition.

### Syntax:

//loop statements

**continue**;

//some lines of the code which is to be skipped

**Level 2:**

1. **Arrays**

# C Array

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc.

## Declaration of C Array

We can declare an array in the c language in the following way.

data\_type array\_name[array\_size];

Now, let us see the example to declare the array.

**int** marks[5];

# Two Dimensional Array in C

The two-dimensional array can be defined as an array of arrays. The 2D array is organized as matrices which can be represented as the collection of rows and columns. However, 2D arrays are created to implement a relational database lookalike data structure. It provides ease of holding the bulk of data at once which can be passed to any number of functions wherever required.

## Declaration of two dimensional Array in C

The syntax to declare the 2D array is given below.

data\_type array\_name[rows][columns];

Consider the following example.

**int** twodimen[4][3];

### Two-dimensional array example in C

#include<stdio.h>

int main(){

int i=0,j=0;

int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};

//traversing 2D array

for(i=0;i<4;i++){

for(j=0;j<3;j++){

printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);

}//end of j

}//end of i

return 0;

}

**Output**

arr[0][0] = 1arr[0][1] = 2arr[0][2] = 3arr[1][0] = 2arr[1][1] = 3arr[1][2] = 4arr[2][0] = 3arr[2][1] = 4arr[2][2] = 5arr[3][0] = 4arr[3][1] = 5arr[3][2] = 6

1. **Pointer**

# C Pointers

The pointer in C language is a variable which stores the address of another variable. This variable can be of type int, char, array, function, or any other pointer. The size of the pointer depends on the architecture. However, in 32-bit architecture the size of a pointer is 2 byte.

Consider the following example to define a pointer which stores the address of an integer.

int n = 10;

int\* p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integer.

## Declaring a pointer

The pointer in c language can be declared using \* (asterisk symbol). It is also known as indirection pointer used to dereference a pointer.

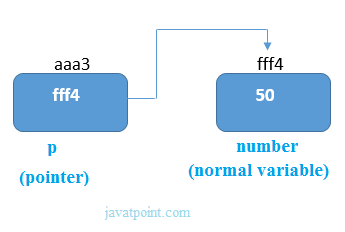
**int** \*a;//pointer to int

**char** \*c;//pointer to char

### Pointer Example

An example of using pointers to print the address and value is given below.

Difference between JDK, JRE, and JVM



As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of \* (**indirection operator**), we can print the value of pointer variable p.

Let's see the pointer example as explained for the above figure.

#include<stdio.h>

int main(){

int number=50;

int \*p;

p=&number;//stores the address of number variable

printf("Address of p variable is %x \n",p); // p contains the address of the number therefore printing p gives the address of number.

printf("Value of p variable is %d \n",\*p); // As we know that \* is used to dereference a pointer therefore if we print \*p, we will get the value stored at the address contained by p.

return 0;

}

**Output**

Address of number variable is fff4Address of p variable is fff4Value of p variable is 50

1. **Functions**

# C Functions

In c, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}. A function can be called multiple times to provide reusability and modularity to the C program. In other words, we can say that the collection of functions creates a program. The function is also known as *procedure*or *subroutine*in other programming languages

## Function Aspects

There are three aspects of a C function.

**Function declaration** A function must be declared globally in a c program to tell the compiler about the function name, function parameters, and return type.

**Function call** Function can be called from anywhere in the program. The parameter list must not differ in function calling and function declaration. We must pass the same number of functions as it is declared in the function declaration.

**Function definition** It contains the actual statements which are to be executed. It is the most important aspect to which the control comes when the function is called. Here, we must notice that only one value can be returned from the function.

|  |  |  |
| --- | --- | --- |
| **SN** | **C function aspects** | **Syntax** |
| 1 | Function declaration | return\_type function\_name (argument list); |
| 2 | Function call | function\_name (argument\_list) |
| 3 | Function definition | return\_type function\_name (argument list) {function body;} |

The syntax of creating function in c language is given below:

How to find Nth Highest Salary in SQL

return\_type function\_name(data\_type parameter...){

//code to be executed

}

1. **Command Line Arguments**

# Command Line Arguments in C

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

To support command line argument, you need to change the structure of main() function as given below.

**int** main(**int** argc, **char** \*argv[] )

Here, **argc** counts the number of arguments. It counts the file name as the first argument.

The **argv[]** contains the total number of arguments. The first argument is the file name always.

32M

## Example

Let's see the example of command line arguments where we are passing one argument with file name.

#include <stdio.h>

void main(int argc, char \*argv[] ) {

printf("Program name is: %s\n", argv[0]);

if(argc < 2){

printf("No argument passed through command line.\n");

}

else{

printf("First argument is: %s\n", argv[1]);

}

}

Run this program as follows in Linux:

./program hello

Run this program as follows in Windows from command line:

program.exe hello

**Output:**

Program name is: programFirst argument is: hello

1. **Strings**

# C Strings

The string can be defined as the one-dimensional array of characters terminated by a null ('\0'). The character array or the string is used to manipulate text such as word or sentences. Each character in the array occupies one byte of memory, and the last character must always be 0. The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

Let's see the example of declaring **string by char array** in C language.

**char** ch[10]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};

Let's see a simple example where a string is declared and being printed. The '%s' is used as a format specifier for the string in c language.

#include<stdio.h>

#include <string.h>

int main(){

char ch[11]={'j', 'a', 'v', 'a', 't', 'p', 'o', 'i', 'n', 't', '\0'};

char ch2[11]="javatpoint";

printf("Char Array Value is: %s\n", ch);

printf("String Literal Value is: %s\n", ch2);

return 0;

}

**Output**

Char Array Value is: javatpointString Literal Value is: javatpoint

## Accepting string as the input

Till now, we have used scanf to accept the input from the user. However, it can also be used in the case of strings but with a different scenario. Consider the below code which stores the string while space is encountered.

#include<stdio.h>

void main ()

{

char s[20];

printf("Enter the string?");

scanf("%s",s);

printf("You entered %s",s);

}

**Output**

Enter the string?javatpoint is the best You entered javatpoint

# C String Functions

There are many important string functions defined in "string.h" library.

|  |  |  |
| --- | --- | --- |
| **No.** | **Function** | **Description** |
| 1) | [strlen(string\_name)](https://www.javatpoint.com/c-strlen) | returns the length of string name. |
| 2) | [strcpy(destination, source)](https://www.javatpoint.com/c-strcpy) | copies the contents of source string to destination string. |
| 3) | [strcat(first\_string, second\_string)](https://www.javatpoint.com/c-strcat) | concats or joins first string with second string. The result of the string is stored in first string. |
| 4) | [strcmp(first\_string, second\_string)](https://www.javatpoint.com/c-strcmp) | compares the first string with second string. If both strings are same, it returns 0. |
| 5) | [strrev(string)](https://www.javatpoint.com/c-strrev) | returns reverse string. |
| 6) | [strlwr(string)](https://www.javatpoint.com/c-strlwr) | returns string characters in lowercase. |
| 7) | [strupr(string)](https://www.javatpoint.com/c-strupr) | returns string characters in uppercase. |

**Level 3:**

1. **Structures**

## Why use structure?

In C, there are cases where we need to store multiple attributes of an entity. It is not necessary that an entity has all the information of one type only. It can have different attributes of different data types. For example, an entity **Student** may have its name (string), roll number (int), marks (float). To store such type of information regarding an entity student, we have the following approaches:

* Construct individual arrays for storing names, roll numbers, and marks.
* Use a special data structure to store the collection of different data types.

## What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member. Structures ca; simulate the use of classes and templates as it can store various information

The **,struct** keyword is used to define the structure. Let's see the syntax to define the structure in c.

struct structure\_name

{

data\_type member1;

data\_type member2;

.

.

data\_type memeberN;

};

Let's see the example to define a structure for an entity employee in c.

**struct** employee

{ **int** id;

**char** name[20];

**float** salary;

};

## Accessing members of the structure

There are two ways to access structure members:

1. By . (member or dot operator)
2. By -> (structure pointer operator)

Let's see the code to access the *id* member of *p1* variable by. (member) operator.

p1.id

### C Structure example

Let's see a simple example of structure in C language.

#include<stdio.h>

#include <string.h>

struct employee

{ int id;

char name[50];

}e1; //declaring e1 variable for structure

int main( )

{

//store first employee information

e1.id=101;

strcpy(e1.name, "Sonoo Jaiswal");//copying string into char array

//printing first employee information

printf( "employee 1 id : %d\n", e1.id);

printf( "employee 1 name : %s\n", e1.name);

return 0;

}

**Output:**

employee 1 id : 101employee 1 name : Sonoo Jaiswal

1. **File Handling**

File handling in C enables us to create, update, read, and delete the files stored on the local file system through our C program. The following operations can be performed on a file.

* Creation of the new file
* Opening an existing file
* Reading from the file
* Writing to the file
* Deleting the file

## Functions for file handling

There are many functions in the C library to open, read, write, search and close the file. A list of file functions are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Function** | **Description** |
| 1 | fopen() | opens new or existing file |
| 2 | fprintf() | write data into the file |
| 3 | fscanf() | reads data from the file |
| 4 | fputc() | writes a character into the file |
| 5 | fgetc() | reads a character from file |
| 6 | fclose() | closes the file |
| 7 | fseek() | sets the file pointer to given position |
| 8 | fputw() | writes an integer to file |
| 9 | fgetw() | reads an integer from file |
| 10 | ftell() | returns current position |
| 11 | rewind() | sets the file pointer to the beginning of the file |

## Opening File: fopen()

We must open a file before it can be read, write, or update. The fopen() function is used to open a file. The syntax of the fopen() is given below.

C++ vs

**FILE** \*fopen( **const** **char** \* filename, **const** **char** \* mode );

The fopen() function accepts two parameters:

* The file name (string). If the file is stored at some specific location, then we must mention the path at which the file is stored. For example, a file name can be like **"c://some\_folder/some\_file.ext"**.
* The mode in which the file is to be opened. It is a string.

## Closing File: fclose()

The fclose() function is used to close a file. The file must be closed after performing all the operations on it. The syntax of fclose() function is given below:

**int** fclose( **FILE** \*fp );

# C fputs() and fgets()

The fputs() and fgets() in C programming are used to write and read string from stream. Let's see examples of writing and reading file using fgets() and fgets() functions.

## Writing File : fputs() function

The fputs() function writes a line of characters into file. It outputs string to a stream.

**Syntax:**

int fputs(const char \*s, FILE \*stream)

#include<stdio.h>

#include<conio.h>

void main(){

FILE \*fp;

clrscr();

fp=fopen("myfile2.txt","w");

fputs("hello c programming",fp);

fclose(fp);

getch();

}

**myfile2.txt**

hello c programming

## Reading File : fgets() function

The fgets() function reads a line of characters from file. It gets string from a stream.

**Syntax:**

char\* fgets(char \*s, int n, FILE \*stream)

**Example:**

#include<stdio.h>

#include<conio.h>

void main(){

FILE \*fp;

char text[300];

clrscr();

fp=fopen("myfile2.txt","r");

printf("%s",fgets(text,200,fp));

fclose(fp);

getch();

}

Output:

hello c programming

1. **Error Handling**

As such, C programming does not provide direct support for error handling but being a system programming language, it provides you access at lower level in the form of return values. Most of the C or even Unix function calls return -1 or NULL in case of any error and set an error code **errno**. It is set as a global variable and indicates an error occurred during any function call. You can find various error codes defined in <error.h> header file.

So a C programmer can check the returned values and can take appropriate action depending on the return value. It is a good practice, to set errno to 0 at the time of initializing a program. A value of 0 indicates that there is no error in the program.

## errno, perror(). and strerror()

The C programming language provides **perror()** and **strerror()** functions which can be used to display the text message associated with **errno**.

* The **perror()** function displays the string you pass to it, followed by a colon, a space, and then the textual representation of the current errno value.
* The **strerror()** function, which returns a pointer to the textual representation of the current errno value.

Let's try to simulate an error condition and try to open a file which does not exist. Here I'm using both the functions to show the usage, but you can use one or more ways of printing your errors. Second important point to note is that you should use **stderr** file stream to output all the errors.

#include <stdio.h>#include <errno.h>#include <string.h>extern int errno ;int main () { FILE \* pf; int errnum; pf = fopen ("unexist.txt", "rb"); if (pf == NULL) { errnum = errno; fprintf(stderr, "Value of errno: %d\n", errno); perror("Error printed by perror"); fprintf(stderr, "Error opening file: %s\n", strerror( errnum )); } else { fclose (pf); } return 0;}

When the above code is compiled and executed, it produces the following result −

Value of errno: 2Error printed by perror: No such file or directoryError opening file: No such file or directory