**C Language**

**C language was developed by “Dennis Ritchie” in 1997 at and t bell lab.**

**It is simplest and basic language to make software.**

**Notes:- ‘Main’ Function is important for run program .**

**Only main function se program run to ho jayega pr output windows stop nhi hogi**

**EX-: void main(){**

**}// no return type of function.**

* **block of code is know as function.**

**getch() :- to stop the output windows .**

**if we use only getch() it will give error “Function ‘getch’ should have prototype”.**

* **To known the location of getch :- Go in C drive in ‘Turbo’ folder There is ‘include’ name file ,in include file there is ‘conio.h’ kept. And ‘conio.h’ file se pta chal jayega apne ko ‘getch()’ function ka**
* **‘conio.h ‘ file me hi ‘getch()‘ function present hota hai .**

**‘getch()’ = get charcter.**

**Clrscr():- clrscr() function ka use screen ko clear karne ke liye kiya jata hai. Ye bhi “conio.h” headder file se belong karat hai.**

**Stdio.h:- it is also headder file ,it has ‘printf()’ and scanf().**

**Printf():- it is use for printf data on consol screen.**

**Scanf():- it is use for take input from user.**

**Scape charecters-:**

* **\n is use to change the current line.**
* **\t is use to tabe space.**
* **\r is use for move curser at first position.**
* **\” is used to print single “**
* **\’ is used to print single ‘**
* **\\ is used to print single \**
* **\b is used for bace space**

**All scape charecters are apply only with \ (back slace) not with / (Forword slace).**

**We can use getch() and clrscr() N number of times in our program .**

**Variable**

**Note:- kise bhi variable ko use karne se pahle us variable ke corresponding memory ko creat karna compulsory** **hai.**

* **Variable ke corresponding data type spacify karne se memory creat hoti hai.**
* **Jis type ka data hota hai use type ke data type ko use karte hai .**

**Ex:- int x=10;**

**%i spacefier is use for print int data**

* **Memory name should be always unique .**

**Number System**

* **Koi bhi number ke agge ‘0’(Zero) lag jane se vo number octal ban jata hai . Ex:-int x=075;**
* **Printf hamesa decimal number return karta hai.**
* **Koi bhi number ke agge 0x lag jane se vaha number Hexadecimal number ban jata hai. Ex:-int x=0x75;**
* **C prgraming me printf value ko spaceifire me store karta hai.**

**Binory number= 0,1**

**Ocatal number= 0 to 7**

**Decimal number= 0 to 9**

**Hexadecimal number = 0 to 9 a=10 , b=11 , c=12 , d=13 , e=14 , f=15**

**Octal=8**

**Decimal=10**

**Hexadecimal=16**

**B -> D=\*2**

**D->B=/2**

**O->D=\*8**

**D->O=/8**

**H->D=\*16**

**D->H=/16**

**Ager octal me 7 se jiyada number liya to error ayege kyo ki octal ke pass serf 0 to 7 number hi hota hai**

**Operators**

**Arithmetic operator**

**There are 5 operators in arithmetic operator .**

**+ , -,\* , / , %**

**10+20=30;**

**10-20=-10**

**10\*20=200**

**10/2=5**

**10%2=0**

**Relational operator or Compilation operator**

**There are 6 type of relation operator:-**

**< , > , <= , >= , == , !=**

**Ex:- int x=10,y=20;**

**X < y =1 ( true )**

**X > y =>0 ( false )**

**X <= y =>1 ( true )**

**X >= y =>0 ( false )**

**X == y =>0 ( false )**

**X != y =>1 ( true )**

**Bitwise operator**

**Bitwise operaot is operator which works on every bit of number.**

**There are 6 type of Bitwise operator.**

**Bitwise AND operator (&)**

**Bitwise OR operator (|)**

**Bitwise XOR operator (^)**

**Bitwise LeftShift operator (<<)**

**Bitwise RightShift operator (>>)**

**Bitwise compliment operator (~)**

**C1 C2 (C1 & C2)**  **(C1 | C2) (C1 ^ C2)**

**0 0 0 0 0**

**0 1 0 1 1**

**1 0 0 1 1**

**1 1 1 1 0**

**AND operator me ek bhi condition true pr op=true.**

**OR operator me ek bhi condition false pr op=false.**

**XOR operator me uposite condition pr op=true.**

**Increment Decriment operator**

**Increment :-it is use for incrise the value with the help of ++.**

**Decriment:- it is use for dicrise the value with the help of --.**

**X++ => x+1.**

**X-- => x-1.**

**PreIncriment => first Incriment second Assign value**

**Ex=> y = ++x.**

**postIncriment => first Assign second Incriment**

**Ex=> y = x++.**

**Assignment operator**

**assignment operator is use to assign the value of variable with the help of it’s operators.**

**There are 6 type of assignment operator -:**

**= , += , -= , \*= , /= , %=**

**X+=y => x=x+y**

**X-=y => x=x-y**

**Ex:- 10 += 2 => 30**

**10 -= 2 => 8**

**10 \*= 2 => 20**

**10 /= 2 => 5**

**10 %= 2 => 0**

**10 = 2 => 2**

**Reminder (%) ke case me hamesa jis value se divide ker rahe hai uska hi sign ata hai .**

**But divide(/) ke case me jesa padte aa rahe hai vesa hi ayega like + and + = + , - and - = - , + and - = - , - and + = - .**

**Hamesa if ke bad else ka hona jaruri hai .**

**C programing me if() me 0 and 1 de sakte hai.**

**Logical operator**

**When we want to use more then one condition so we use Logical operators .**

**There are 3 type of logical operator:-**

**Logical AND = &&**

**Logical OR = ||**

**Logical Not = !**

**C1 C2 (C1 && C2)**  **(C1 || C2) !C1 !C2**

**0 0 0 0 1 1**

**0 1 0 1 1 0**

**1 0 0 1 0 1**

**1 1 1 1 0 0**

**There are 4 type of condition statement**

* **If statement :- When we have one condition.**

**Ex :- If() {**

**}**

**It check every if block.**

* **If else statement :- when we have only two condition.**

**Ex :- if(){**

**}**

**else{**

**}**

**Ager If(){} ki condition false hogi to else{} ko check karega.**

* **Nested if statement :- when we have more than on condition inside if statement.**

**Ex= if(){**

**If(){**

**}else{**

**}**

**}**

**else{**

**}**

* **else if ladder:- when we have multipal conditions and we want that only one true condtion should be exequte.**

**Ex :- if(){**

**}**

**else if() {**

**}**

**else if() {**

**}**

**else if() {**

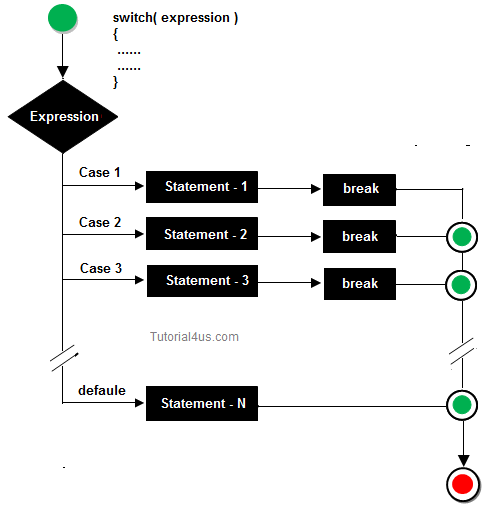
**}**

**Ager ek bhi condition true hogi to agge check nhi karta hai.**

**Switch Case**

* **It has 3 keyword case,break,default.**
* **Switch() ke ander jo bhi number dege uss vale case ko hi compiler execute karega. Ager switch ki condition case me nhi hai to kuch bhi output nhi ayega.**
* **Program me default() and break ka use karna compulsory nhi hai .**
* **Ager default() use nhi kiya to case match na hone pr kuch output nhi ayega yadi kiya hai to default() execute hota hai.**
* **Ager break ka use nhi kiya to jis case block se execute hoga use agge ke sare case print hoge.**
* **default() ka use hole program me kahi bhi ker sakte hai.**
* **Ham n number of times break ka use ker sakte hai.**
* **Hame only 1 bar hi default() ka use ker sakte hai pure switch() case me .same case ka use nhi ker sakte hai.**
* **Ham case ko khali nhi chod sakte hai like case :**

**Syntex:-**



**goto statement**

**it is use to jump on any particular statement .**

**Syntax:-**

**goto y;**

**x:**

**---**

**y:**

# continue statement

**Q what is continue statement ?**

**Ans:- 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:**

1. //loop statements
2. **continue**;
3. //some lines of the code which is to be skipped

**Conditional Operator/ Ternary Operator(? :)**

**It is also work like if else but when we have only one condition so we should use it because we can write all conditions in one line.**

**When condition is true so the code of after ? will execute if false the code of after : will execute .**

**Any random number is called garbage value.**

**LOOPS**

**Loop is used to execute the block of code several times according to the condition given in the loop. It means it executes the same code multiple times so it saves code and also helps to traverse the elements of an array**.

**There are 3 types of loop**

* **while loop**
* **do while loop**
* **for loop**

**int I; => ( declear )**

**i=10; => ( initialization)**

**i++; => ( inciment )**

**step 1:- intilization.**

**Step 2:- check condition.**

**Step 3:-if condition is false then loop end .**

**Step 4:- if condition is true execute body and increment /decriment**

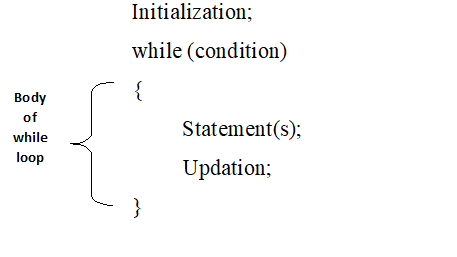
**Step 5 :- again condition check if it is true it will excute until condition will not be false.**

**while Loop**

**Q when we use while loop ?**

**Ans :-The while loop is used to repeat a section of code an unknown number of times until a specific condition is met. For example, say we want to know how many times a given number can be divided by 2 before it is less than or equal to 1**.

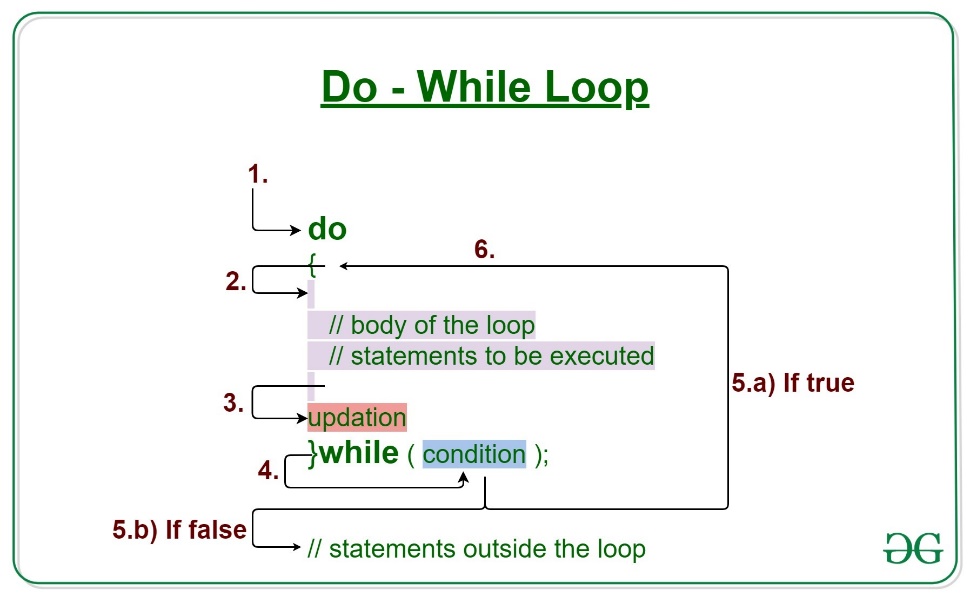
**Syntex of while Loop**



**do while Loop**

**Q when we use do while loop ?**

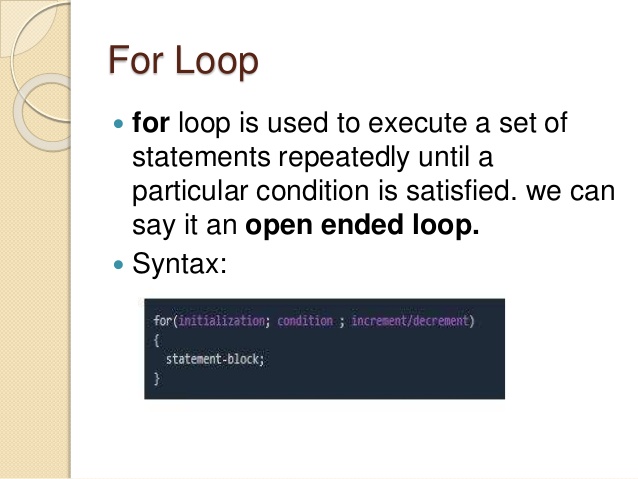
**Ans** :-Because **do while loops** check the condition after the block is executed, the control structure is often also known as a post-test **loop**. Contrast with the **while loop**, which tests the condition before the code within the block is executed, the **do**-**while loop** is an exit-condition **loop**.



**for Loop**

**Q why we use for loop ?**

**Ans** :- A "For" **Loop** is **used** to repeat a specific block of code a known number of times. For example, if **we** want to check the grade of every student in the class, **we loop** from 1 to that number. When the number of times is not known before hand, **we use** a "While" **loop**.



**Functions**

**Q What is function ?**

**Ans :-** **A function is a group of statements that together perform a task.**

**Function ki starting small letter se hi hote hai.**

**In c there are 2 type of functions**

**1:-predefine**

**2;-userdefine**

**Predefine :- asa function jo c programing ne pahle se diya hoaa hai use predefine dunction kahte hai**

**Ex:- getch() , printf() , clrscr() , scanf()…etc**

**Userdefine :- asa function jo programmer creat karta hai apne requairment ke hisab se.**

**Ex:- sum() , add()**

**Function Declearation**

**Syntex:**

**Return\_type Function\_name(perameter\_list)**

**Step1:- return\_type can be :- int , void , float**

**Step 2:- Function definition**

**Syntax**

**Return\_type function\_name(parameters){**

**}**

**Step 3:- function\_calling**

**Advantage of function:-**

* **time saving**
* **space less**
* **edit simply**

**category of function:-**

* **function with no parameter and no return type**

**Ex:-void sum();**

* **function with parameter and no return type**

**Ex:-void sum(100);**

* **function with no parameter and return type**

**Ex :- int sum();**

* **function with parameter and return type**

**Ex :- int sum(100);**

**retun keyword value ko calling method pr return karta hai .**

**Array**

**Q what is aray?**

**Ans:- array is a collection of homogenous type of data. We can store n number of values in one variable with the help of array.**

**Array can store only same type of data**

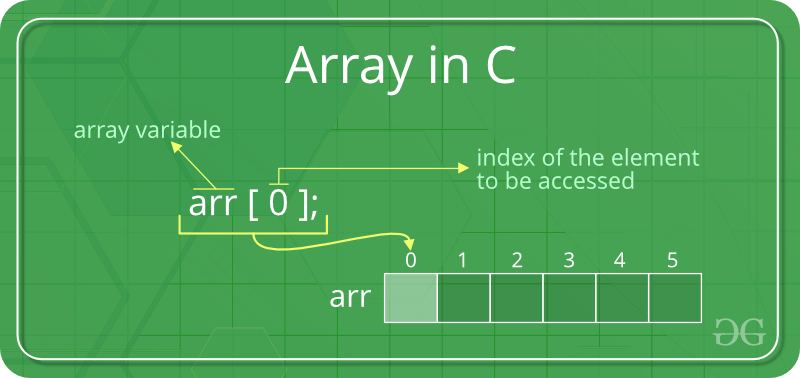
**Like only if we declear variable with int data type so it will store only int type of data not string or float.**

**Array indexing always starts with 0.**

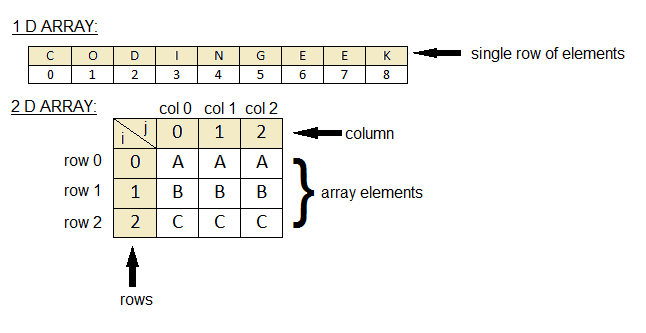
**There are two type of array .**

* One dimensional **array**.
* Multi dimensional **array**
* One dimensional **array** is collection of n number of value/elements.

**1d array declearation int x [5]={ 10 , 2 , 6 , 45 , 5}**

* 

**List of 1d array is called 2d array**



**It goes continues with every one like list of 2d array is called 3d array , list of 3d array is called 4d aray………**

**Yadi hame data ko Row and Column ke form me print karvana hai to ham 2d array ka use karge.**

**2 array declearation int x[ 2 ] [ 3 ]= { { 1 , 5 , 6 } , { 6 , 4 , 8 }}**

* **Array me job hi data store hota hai vo continues memory location me ja ker store hota hai**

**Pointer**

**Pointer variable same data type ke normal variable ke address ko hold/store karta hai .**

**Value**

Address of normal variable.

**Pointer variable normal variable**

**& => assign address**

**%u => print address**

**int x = 10;**

**int \*p = &x;**

**&x**

**x**

**10**

**&x &p**

**\*p**

**X = 10;**

**&x = &x;**

**P = &x;**

**&p = &p;**

**\*p = 10;**

**Double pointer**

**Int x=10; \*\*q**

**Int \*p=&x; Int \*\*q=&p; \*q**

**\*p**

**&p**

**&x**

**10**

**x p q**

**&x &p &q**

**Very Impotant:-**

**X=10; &x = &x; &p = &p &q = &q**

**\*p=10; p = &x; q = &p**

**\*\*q=10; \*q = &x**

**Q what is double pointer ?**

**Ans** **:-** A **pointer** is used to store the address of variables. So, when we define a **pointer** to **pointer**, the first **pointer** is used to store the address of the second **pointer**. Thus it is known as **double pointers**

**Character**

**All keys on keyboard are called character. It represent only inside {‘ ’}**

**ASCII :- keyboard m jitney bhi keys hai uske corresponding har ek key ki integer value hote hai jise ASCII value kahte hai .**

**Full form of ASCII :-** **American Standard Code of Information Interchange.**

**A to Z** = **65 to 90** (Capital character)

**a to z** = **97 to 122** (Small character)

**0 to 9** = **48 to 57** ( Numbers )

**String :- collection of character is called String.**

**OR**

**Array of character is called String.**

**Ex;- “Manish”**

**String is always terminated by null character (null character -> \0)**

**In C language String can create by 2 type.**

* **By using character array.**
* **By using String literal.**

**String literal me compiler already null character add ker deta hai String ke end me.**

* **Some important String functions :-**

|  |  |  |
| --- | --- | --- |
| **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. |

C Math

**C Programming allows us to perform mathematical operations through the functions defined in <math.h> header file. The <math.h> header file contains various methods for performing mathematical operations such as sqrt(), pow(), ceil(), floor() etc.**

C Math Functions

**There are various methods in math.h header file. The commonly used functions of math.h header file are given below**.

|  |  |  |
| --- | --- | --- |
| **No.** | **Function** | **Description** |
| 1) | ceil(number) | rounds up the given number. It returns the  integer value which is greater than or equal to  given number. |
| 2) | floor(number) | rounds down the given number. It returns the  integer value which is less than or equal to  given number. |
| 3) | sqrt(number) | returns the square root of given number. |
| 4) | pow(base, exponent) | returns the power of given number. |
| 5) | abs(number) | returns the absolute value of given number. |

**Structure**

**Q what is Structure ?**

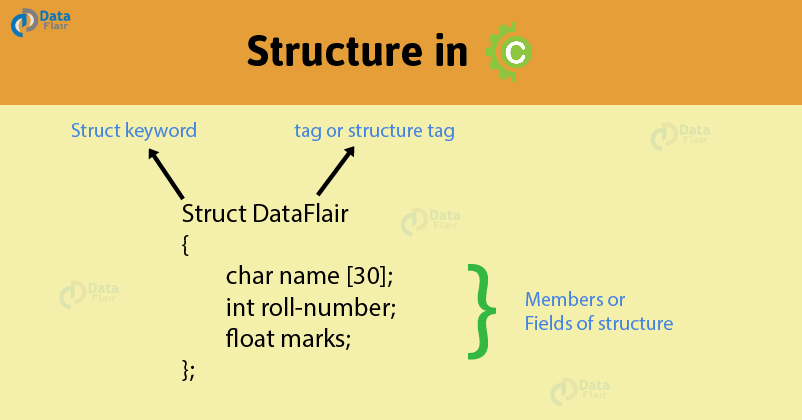
**Ans :- Collection of different type of data type is called Structure .**

**OR**

**Collection of homogeneous type of data type is called Structure.**

* **Structure is created by using struct keyword.**
* **Structure always terminated by ‘ ; ’**

**Structure Syntex :-**



**Structure ke ander ka data hamesa Structure ke variable (object) se hi access hota hai.**

* **We can make more than one structure in one program.**

**Union**

**Q what is union ?**

**Ans :- Union is an user defined datatype in C programming language. It is a collection of variables of different datatypes in the same memory location. We can define a union with many members, but at a given point of time only one member can contain a value .**

* **C unions are used to save memory.**
* **Union ko union keyword se create karte hai .**
* **Structure me separate data type ki separate memory create hote hai . but union me only largest data type ki memory create hoti hai.**

**Ex :-**



# Dynamic memory allocation in C

**The concept of dynamic memory allocation in c language enables the C programmer to allocate memory at runtime. Dynamic memory allocation in c language is possible by 4 functions of stdlib.h header file.**

1. **malloc()**
2. **calloc()**
3. **realloc()**
4. **free()**

**Before learning above functions, let's understand the difference between static memory allocation and dynamic memory allocation.**

|  |  |
| --- | --- |
| **static memory allocation** | **dynamic memory allocation** |
| **memory is allocated at compile time.** | memory is allocated at  runtime. |
| **memory can't be increased while executing program.** | memory can be increased while executing program. |
| **used in array.** | used in linked list. |

**methods used for dynamic memory allocation.**

|  |  |
| --- | --- |
| **malloc()** | **allocates single block of requested memory**. |
| **calloc()** | **allocates multiple block of requested memory**. |
| **realloc()** | **reallocates the memory occupied by malloc() or calloc() functions.** |
| **free()** | **frees the dynamically allocated memory.** |

## malloc() function in C

**The malloc() function allocates single block of requested memory.**

**It doesn't initialize memory at execution time, so it has garbage value initially.**

**It returns NULL if memory is not sufficient.**

**The syntax of malloc() function is given below:**

**ptr=(cast-type\*)malloc(byte-size)**

* **calloc() function in C**

**The calloc() function allocates multiple block of requested memory.**

**It initially initialize all bytes to zero.**

**It returns NULL if memory is not sufficient.**

**The syntax of calloc() function is given below:**

**ptr=(cast-type\*)calloc(number, byte-size)**

* **realloc() function in C**

**If memory is not sufficient for malloc() or calloc(), you can reallocate the memory by realloc() function. In short, it changes the memory size**.

**syntax of realloc() function**:

**ptr=realloc(ptr, new-size)**

**free() function in C**

**The memory occupied by malloc() or calloc() functions must be released by calling free() function. Otherwise, it will consume memory until program exit.**

**syntax of free() function.**

**free(ptr)**

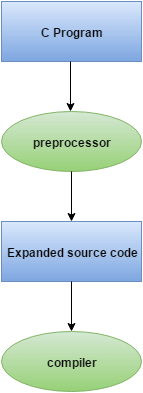
**Pre-processor**

**Program ke compile hone ke pahle ek process hoti hai use pre-processor kahte hai.**

**Q What is pre-processor in c ?**

**Ans :- Pre-processor is a micro processor that is used by compiler to transform your code before compilation. It is called micro pre-processor because it allows us to add macros.**

#### **Note: Pro-processor direcives are executed before compilation.**



**All preprocessor directives starts with hash # symbol.**

**list of pre-processor directives.**

* **#include**
* **#define**
* **#undef**
* **#ifdef**
* **#ifndef**
* **#if**
* **#else**
* **#elif**
* **#endif**
* **#error**
* **#pragma**

**Storage Classes**

**Q What is storage class ?**

**Ans :- Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable.**

**There are four types of storage classes in C**

* **Automatic**
* **External**
* **Static**
* **Register**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Storage Classes** | **Storage Place** | **Default Value** | **Scope** | **Lifetime** |
| **auto** | RAM | Garbage Value | Local | Within function |
| **extern** | RAM | Zero | Global | Till end of the main program |
| **static** | RAM | Zero | Local | Till the program is terminate |
| **register** | Register | Garbage Value | Local | Within the function |

## Automatic

* **Automatic variables are allocated memory automatically at runtime.**
* **The visibility of the automatic variables is limited to the block in which they are defined.**

**The scope of the automatic variables is limited to the block in which they are defined.**

* **The automatic variables are initialized to garbage by default.**
* **The memory assigned to automatic variables gets freed upon exiting from the block.**
* **The keyword used for defining automatic variables is auto.**
* **Every local variable is automatic in C by default.**

**Syntex :-**

1. **#include <stdio.h>**
2. **int main()**
3. **{**
4. **int a; //auto**
5. **char b;**
6. **float c;**
7. **printf("%d %c %f",a,b,c); // printing initial default value of automatic variables a, b, and c.**
8. **return 0;**
9. **}**

**Output:**

**garbage garbage garbage**

## Static

* **The variables defined as static specifier can hold their value between the multiple function calls.**
* **Static local variables are visible only to the function or the block in which they are defined.**
* **A same static variable can be declared many times but can be assigned at only one time.**
* **Default initial value of the static integral variable is 0 otherwise null.**
* **The visibility of the static global variable is limited to the file in which it has declared.**
* **The keyword used to define static variable is static.**

**Syntex :-**

1. **#include<stdio.h>**
2. **static char c;**
3. **static int i;**
4. **static float f;**
5. **static char s[100];**
6. **void main ()**
7. **{**
8. **printf("%d %d %f %s",c,i,f); // the initial default value of c, i, and f will be printed.**
9. **}**

**Output:**

**0 0 0.000000 (null)**

## Register

* **The variables defined as the register is allocated the memory into the CPU registers depending upon the size of the memory remaining in the CPU.**
* **We can not dereference the register variables, i.e., we can not use &operator for the register variable.**
* **The access time of the register variables is faster than the automatic variables.**
* **The initial default value of the register local variables is 0 and register default value is garbage**
* **The register keyword is used for the variable which should be stored in the CPU register. However, it is compiler?s choice whether or not; the variables can be stored in the register.**
* **We can store pointers into the register, i.e., a register can store the address of a variable.**
* **Static variables can not be stored into the register since we can not use more than one storage specifier for the same variable.**

**Syntex :-**

1. **#include <stdio.h>**
2. **int main()**
3. **{**
4. **register int a; // variable a is allocated memory in the CPU register. The initial default value of a is 0.**
5. **printf("%d",a);**
6. **}**

**Output:**

**0**

## External

* **The external storage class is used to tell the compiler that the variable defined as extern is declared with an external linkage elsewhere in the program.**
* **The variables declared as extern are not allocated any memory. It is only declaration and intended to specify that the variable is declared elsewhere in the program.**
* **The default initial value of external integral type is 0 otherwise null.**
* **We can only initialize the extern variable globally, i.e., we can not initialize the external variable within any block or method.**
* **An external variable can be declared many times but can be initialized at only once.**
* **If a variable is declared as external then the compiler searches for that variable to be initialized somewhere in the program which may be extern or static. If it is not, then the compiler will show an error.**

**Syntex :-**

1. **#include <stdio.h>**
2. **int a;**
3. **int main()**
4. **{**
5. **extern int a; // variable a is defined globally, the memory will not be allocated to a**
6. **printf("%d",a);**
7. **}**

**Output**

**0**

# Recursion in C

# Q What is recursion ?

**Ans :-** **Recursion is the process which comes into existence when a function calls a copy of itself to work on a smaller problem. Any function which calls itself is called recursive function, and such function calls are called recursive calls. Recursion involves several numbers of recursive calls. However, it is important to impose a termination condition of recursion. Recursion code is shorter than iterative code however it is difficult to understand.**

**Recursion cannot be applied to all the problem, but it is more useful for the tasks that can be defined in terms of similar subtasks. For Example, recursion may be applied to sorting, searching, and traversal problems.**

**Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.**

**In the following example, recursion is used to calculate the factorial of a number**.

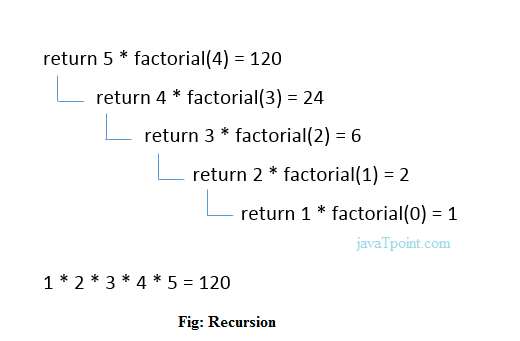
1. **#include <stdio.h>**
2. **int fact (int);**
3. **int main()**
4. **{**
5. **int n,f;**
6. **printf("Enter the number whose factorial you want to calculate?");**
7. **scanf("%d",&n);**
8. **f = fact(n);**
9. **printf("factorial = %d",f);**
10. **}**
11. **int fact(int n)**
12. **{**
13. **if (n==0)**
14. **{**
15. **return 0;**
16. **}**
17. **else if ( n == 1)**
18. **{**
19. **return 1;**
20. **}**
21. **else**
22. **{**
23. **return n\*fact(n-1);**
24. **}**
25. **}**

#### **Output**

**Enter the number whose factorial you want to calculate?5**

**factorial = 120**

**We can understand the above program of the recursive method call by the figure given below:**



## Recursive Function

**A recursive function performs the tasks by dividing it into the subtasks. There is a termination condition defined in the function which is satisfied by some specific subtask. After this, the recursion stops and the final result is returned from the function.**

**The case at which the function doesn't recur is called the base case whereas the instances where the function keeps calling itself to perform a subtask, is called the recursive case. All the recursive functions can be written using this format.**

**Pseudocode for writing any recursive function is given below.**

1. **if (test\_for\_base)**
2. **{**
3. **return some\_value;**
4. **}**
5. **else if (test\_for\_another\_base)**
6. **{**
7. **return some\_another\_value;**
8. **}**
9. **else**
10. **{**
11. **// Statements;**
12. **recursive call;**
13. **}**

## Example of recursion in C

**Let's see an example to find the nth term of the Fibonacci series.**

1. **#include<stdio.h>**
2. **int fibonacci(int);**
3. **void main ()**
4. **{**
5. **int n,f;**
6. **printf("Enter the value of n?");**
7. **scanf("%d",&n);**
8. **f = fibonacci(n);**
9. **printf("%d",f);**
10. **}**
11. **int fibonacci (int n)**
12. **{**
13. **if (n==0)**
14. **{**
15. **return 0;**
16. **}**
17. **else if (n == 1)**
18. **{**
19. **return 1;**
20. **}**
21. **else**
22. **{**
23. **return fibonacci(n-1)+fibonacci(n-2);**
24. **}**
25. **}**

#### **Output**

**Enter the value of n?12**

**144**

## Memory allocation of Recursive method

**Each recursive call creates a new copy of that method in the memory. Once some data is returned by the method, the copy is removed from the memory. Since all the variables and other stuff declared inside function get stored in the stack, therefore a separate stack is maintained at each recursive call. Once the value is returned from the corresponding function, the stack gets destroyed. Recursion involves so much complexity in resolving and tracking the values at each recursive call. Therefore we need to maintain the stack and track the values of the variables defined in the stack.**

**Let us consider the following example to understand the memory allocation of the recursive functions**.

1. **int display (int n)**
2. **{**
3. **if(n == 0)**
4. **return 0; // terminating condition**
5. **else**
6. **{**
7. **printf("%d",n);**
8. **return display(n-1); // recursive call**
9. **}**
10. **}**

**Explanation**

**Let us examine this recursive function for n = 4. First, all the stacks are maintained which prints the corresponding value of n until n becomes 0, Once the termination condition is reached, the stacks get destroyed one by one by returning 0 to its calling stack. Consider the following image for more information regarding the stack trace for the recursive functions.**

