What is C?

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language, despite being old.

C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

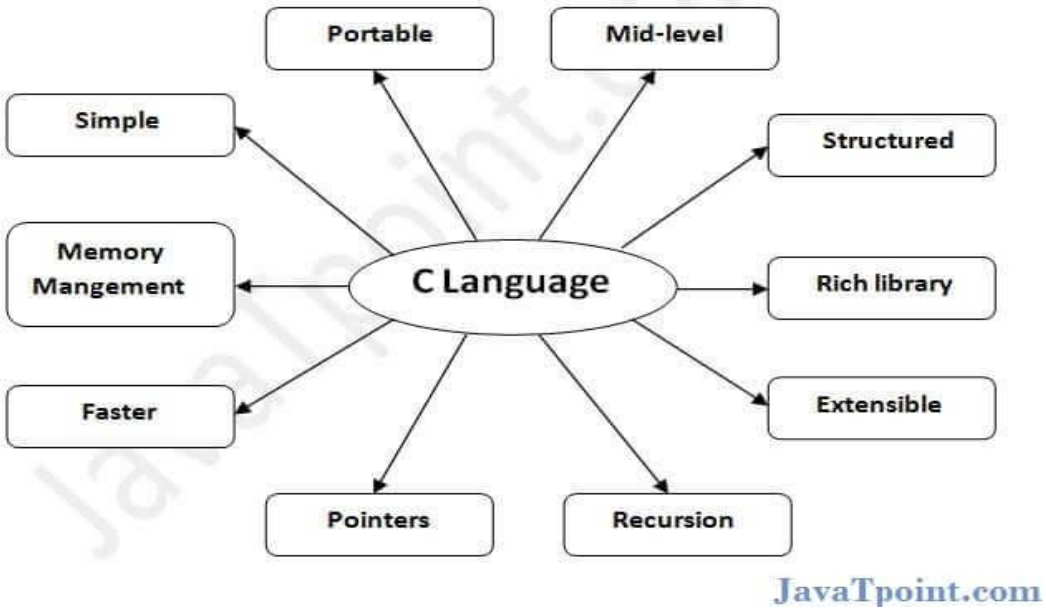
Why Learn C?

- **It is one of the most popular programming language in the world**
- **If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar**
- **C is very fast, compared to other programming languages, like Java and Python**
- **C is very versatile; it can be used in both applications and technologies**

Difference between C and C++

- **C++ was developed as an extension of C, and both languages have almost the same syntax**
- **The main difference between C and C++ is that C++ support classes and objects, while C does no**

Features of C Language



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**

1) Simple

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) Machine Independent or Portable

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

3) 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.

4) 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.

5) Rich Library

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

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) Speed

The compilation and execution time of C language is fast since there are lesser inbuilt functions and hence the lesser overhead.

8) 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.

9) 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.

10) Extensible

C language is extensible because it can easily adopt new features.

C Keywords and Identifiers

In this tutorial, you will learn about keywords; reserved words in C programming that are part of the syntax. Also, you will learn about identifiers and how to name them.

Character set

A character set is a set of alphabets, letters and some special characters that are valid in C language.

Alphabets

```
Uppercase: A B C ................................... X Y Z

Lowercase: a b c ..................................... x y z
```

C accepts both lowercase and uppercase alphabets as variables and functions.

Digits

```
0 1 2 3 4 5 6 7 8 9
```

Special Characters

Special Characters in C Programming

,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"

^	!	*	/	|
-	\	~	+	

White space Characters

Blank space, newline, horizontal tab, carriage return and form feed.

C Keywords

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier. For example:

```
int money;
```

Here, `int` is a keyword that indicates `money` is a variable of type `int` (integer).

As C is a case sensitive language, all keywords must be written in lowercase.

Here is a list of all keywords allowed in ANSI C.

C Keywords

`auto`	`double`	`int`	`struct`
`break`	`else`	`long`	`switch`
`case`	`enum`	`register`	`typedef`
`char`	`extern`	`return`	`union`

`continue`	`for`	`signed`	`void`
`do`	`if`	`static`	`while`
`default`	`goto`	`sizeof`	`volatile`
`const`	`float`	`short`	`unsigned`

All these keywords, their syntax, and application will be discussed in their respective topics. However, if you want a brief overview of these keywords without going further, visit List of all keywords in C programming.

C Identifiers

Identifier refers to name given to entities such as variables, functions, structures etc.

Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program. For example:

```
int money;
double accountBalance;
```

Here, `money` and `accountBalance` are identifiers.

Also remember, identifier names must be different from keywords. You cannot use `int` as an identifier because `int` is a keyword.

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.
3. You cannot use keywords like `int`, `while` etc. as identifiers.
4. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

You can choose any name as an identifier if you follow the above rule, however, give meaningful names to identifiers that make sense.

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.

Let's see the syntax to declare a variable:

The example of declaring the variable is given below:

```
1. int a;
2. float b;
3. 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:

```
1. int a=10,b=20;//declaring 2 variable of integer type
2. float f=20.8;
3. 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:

```
1. int a;
2. int _ab;
3. int a30;
```

Invalid variable names:

```
1. int 2;
2. int a b;
3. int long;
```

String Literals

A string literal is a sequence of characters enclosed in double-quote marks.

For example:

```
"good"                  //string constant
""                     //null string constant
"      "               //string constant of six white space
"x"                    //string constant having a single character.
"Earth is round\n"         //prints string with a newline
```

Constants

If you want to define a variable whose value cannot be changed, you can use the `const` keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

Notice, we have added keyword `const`.

Here, `PI` is a symbolic constant; its value cannot be changed.

```
const double PI = 3.14;
PI = 2.9; //Error
```

In C programming, data types are declarations for variables. This determines the type and size of data associated with variables. For example,

```
int myVar;
```

Here, `myVar` is a variablename of `int` (integer) datatype. The size of `int` is 4 bytes.

Data Types

Here's a table containing commonly used types in C programming for quick access.

Type	Size (bytes)	Format Specifier
`int`	at least 2, usually 4	`%d`, `%i`
`char`	1	`%c`
`float`	4	`%f`
`double`	8	`%lf`
`short int`	2 usually	`%hd`
`unsigned int`	at least 2, usually 4	`%u`
`long int`	at least 4, usually 8	`%ld`, `%li`
`long long int`	at least 8	`%lld`, `%lli`
`unsigned long int`	at least 4	`%lu`
`unsigned long long int`	at least 8	`%llu`
`signed char`	1	`%c`
`unsigned char`	1	`%c`
`long double`	at least 10, usually 12 or 16	`%Lf`

int

Integers are whole numbers that can have both zero, positive and negative values but no decimal values. For example, `0`, `-5`, `10`

We can use `int` for declaring an integer variable.

```
int id;
```

Here, `id` is a variable of type integer.

You can declare multiple variables at once in C programming. For example,

```
int id, age;
```

The size of `int` is usually 4 bytes (32 bits). And, it can take 2^{32} distinct states from `-2147483648` to `2147483647`.

float and double

`float` and `double` are used to hold real numbers.

```
float salary;
double price;
```

In C, floating-point numbers can also be represented in exponential. For example,

```
float normalizationFactor = 22.442e2;
```

What's the difference between `float` and `double`?

The size of `float` (single precision float data type) is 4 bytes. And the size of `double` (double precision float data type) is 8 bytes.

char

Keyword `char` is used for declaring character type variables. For example,

```
char test = 'h';
```

The size of the character variable is 1 byte.

void

`void` is an incomplete type. It means "nothing" or "no type". You can think of void as absent.

For example, if a function is not returning anything, its return type should be `void`.

Note that, you cannot create variables of `void` type.

short and long

If you need to use a large number, you can use a type specifier `long`. Here's how:

```
long a;
long long b;
long double c;
```

Here variables `a` and `b` can store integer values. And, `c` can store a floating-point number.

If you are sure, only a small integer (`[-32,767, +32,767]` range) will be used, you can use `short`.

```
short d;
```

You can always check the size of a variable using the `sizeof()` operator.

```
#include <stdio.h>
int main() {
  short a;
  long b;
  long long c;
  long double d;

  printf("size of short = %d bytes\n", sizeof(a));
  printf("size of long = %d bytes\n", sizeof(b));
  printf("size of long long = %d bytes\n", sizeof(c));
  printf("size of long double= %d bytes\n", sizeof(d));
  return 0;
}
Run Co
```

signed and unsigned

In C, `signed` and `unsigned` are type modifiers. You can alter the data storage of a data type by using them:

- `signed` - allows for storage of both positive and negative numbers
- `unsigned` - allows for storage of only positive numbers

For example,

```
// valid codes
unsigned int x = 35;
int y = -35;  // signed int
int z = 36;  // signed int

// invalid code: unsigned int cannot hold negative integers
unsigned int num = -35;
```

Here, the variables `x` and `num` can hold only zero and positive values because we have used the `unsigned` modifier.

Considering the size of `int` is 4 bytes, variable `y` can hold values from `-`2^{31} to $2^{31}-1$, whereas variable `x` can hold values from `0` to $2^{32}-1$.

Derived Data Types

Data types that are derived from fundamental data types are derived types. For example: arrays, pointers, function types, structures, etc.

Local Variable

A variable that is declared inside the function or block is called a local variable.

It must be declared at the start of the block.

```
// C program to declare and print local variable inside a
// function.
#include <stdio.h>

void function()
{
    int x = 10; // local variable
    printf("%d", x);
}

int main() { function(); }
```

```
#include <stdio.h>

int main() {
   // Write C code here
   int x=10; //local variable
   printf("%d",x);

   return 0;
}
```

Global Variable

A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.

It must be declared at the start of the block.

```
// C program to demonstrate use of global variable
#include <stdio.h>

int x = 20; // global variable

void function1() { printf("Function 1: %d\n", x); }

void function2() { printf("Function 2: %d\n", x); }

int main()
{

    function1();
    function2();
    return 0;
}
```

```
#include <stdio.h>
int x=10; //global variable
int main() {
   // Write C code here
   //int x=10; //local variable
   printf("%d",x);

   return 0;
}
```

Static Variable

A variable that is declared with the static keyword is called static variable.

It retains its value between multiple function calls.

```
1. void function1(){
2. int x=10;//local variable
3. static int y=10;//static variable
4. x=x+1;
5. y=y+1;
6. printf("%d,%d",x,y);
7. }
```

If you call this function many times, the local variable will print the same value for each function call, e.g, 11,11,11 and so on. But the static variable will print the incremented value in each function call, e.g. 11, 12, 13 and so on.

The default value of static variables is zero.

Syntax of Static Variable in C

```
static data_type variable_name = initial_value;
```

As its lifetime is till the end of the program, it can retain its value for multiple function calls as shown in the example.

Example of Static Variable in C

Output

```
First Call
    Local: 30
```

```
// C program to demonstrate use of static variable
#include <stdio.h>
void function()
{
    int x = 20; // local variable
    static int y = 30; // static variable
    x = x + 10;
    y = y + 10;
    printf("\tLocal: %d\n\tStatic: %d\n", x, y);
}

int main()
{
    printf("First Call\n");
    function();
    printf("Second Call\n");
    function();
    printf("Third Call\n");
    function();
    return 0;
}
```

```
    Static: 40
Second Call
    Local: 30
    Static: 50
Third Call
    Local: 30
    Static: 60
```

In the above example, we can see that the local variable will always print the same value whenever the function will be called whereas the static variable will print the incremented value in each function call.

Automatic Variable in C

All the local variables are automatic variables by default. They are also known as auto variables.
Their scope is local and their lifetime is till the end of the block. If we need, we can use the auto keyword to define the auto variables.
The default value of the auto variables is a garbage value.

Syntax of Auto Variable in C

```
auto data_type variable_name;
        or
data_type variable_name;    (in local scope)
```

Example of auto Variable in C

```
// C program to demonstrate use of automatic variable
#include <stdio.h>
void function()
{
    int x = 10; // local variable (also automatic)
    auto int y = 20; // automatic variable
    printf("Auto Variable: %d", y);
}
int main()
{
    function();
    return 0;
}
```

External Variables in C
External variables in C can be shared between multiple C files. We can declare an external variable using the extern keyword.
Their scope is global and they exist between multiple C files.

Syntax of Extern Variables in C

```
extern data_type variable_name;
```

Example of Extern Variable in C

```
----------myfile.h------------
extern int x=10;  //external variable (also global)



----------program1.c----------
#include "myfile.h"
#include <stdio.h>
void printValue(){
printf("Global variable: %d", x);
}
```

In the above example, x is an external variable that is used in multiple C files.

Register Variables in C

Register variables in C are those variables that are stored in the CPU register instead of the conventional storage place like RAM. Their scope is local and exists till the end of the block or a function.

These variables are declared using the register keyword.
The default value of register variables is a garbage value.

Syntax of Register Variables in C

```
register data_type variable_name = initial_value;
```

Example of Register Variables in C

```
// C program to demonstrate the definition of register
// variable
#include <stdio.h>


int main()
{
    //    register variable
    register int var = 22;


    printf("Value of Register Variable: %d\n", var);
    return 0;
}
```

Output

```
Value of Register Variable: 22
```

NOTE: We cannot get the address of the register variable using addressof (&) operator because they are stored in the CPU register. The compiler will throw an error if we try to get the address of register variable.

Constant Variable in C

Till now we have only seen the variables whose values can be modified any number of times. But C language also provides us a way to make the value of a variable immutable. We can do that by defining the variable as constant.

A **constant variable** in C is a read-only variable whose value cannot be modified once it is defined. We can declare a constant variable using the const keyword.

Syntax of Const Variable in C

```
const data_type variable_name = value;
```

Note: We have to always initialize the const variable at the definition as we cannot modify its value after defining.

Example of Const Variable in C

```
// C Program to Demonstrate constant variable
#include <stdio.h>
int main()
{
    // variable
    int not_constant;

    // constant variable;
    const int constant = 20;

    // changing values
    not_constant = 40;
    constant = 22;

    return 0;
```

```
}
```

Save file

C code and save the file as `myfirstprogram.c`

Header files

- #include <stdio.h> includes the standard input output library functions.
- #include <conio.h> includes the standard input output library functions. It contains some useful console functions.

printf () and scanf() in C

- The printf() and scanf() functions are used for input and output in C language. Both functions are inbuilt library functions, defined in stdio.h (header file).

printf() output

- `printf()` is a function used to output/print text to the screen. In our example it will output "Hello World!".
- `printf()` is one of the main output function. The function sends formatted output to the screen.

main()

- Int(data type)(void null) main() The main() function is the entry point of every program in c language.
- All valid C programs must contain the `main()` function. The code execution begins from the start of the `main()` function.

return 0

- 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.
- The `return 0;` statement inside the `main()` function is the "Exit status" of the program.

Example 1: C Output

#include<stdio.h>
int main()
{
// Displays the string inside quotations
printf("C Programming");
return 0;
}
	#include<stdio.h>
	#include<conio.h>
	void main()
	{
	// Displays the string inside quotations
	printf("C Programming");
	getch();
	}

scanf() Input

- In C programming, `scanf()` is one of the commonly used function to take input from the user. The `scanf()` function reads formatted input from the standard input such as keyboards.

Int

```
#include <stdio.h>

// Driver code

int main()

{

int a, b;

printf("Enter first number: ");

scanf("%d", &a);

printf("Enter second number: ");

scanf("%d", &b);

printf("A : %d \t B : %d" , a , b);

return 0;

}
```

OUPTPUT

Enter first number: 8

Enter second number: 9

A : 8 B : 9

Float decimal value show

```
#include <stdio.h>
// Driver code
int main()
{
   float a, b;

    printf("Enter first number: ");
    scanf("%f", &a);

    printf("Enter second number: ");
    scanf("%f", &b);

    printf("A : %f \t B : %f" ,
        a , b);
```

OUPTPUT

Enter first number: 6

Enter second number: 7

A : 6.000000 B : 7.000000

```
    return 0;
}
```

Char one charchter show

```
#include <stdio.h>
// Driver code
int main()
{
   char a;

    printf("Enter name: ");
    scanf("%c", &a);

    printf("%c" , a);

   return 0;
}
```

OUPTPUT

Enter name: hhhhhh

h

Full name show string

```
#include <stdio.h>
// Driver code
int main()
{
   char a[20];

    printf("Enter name: ");
```

OUPTPUT

Enter name: ravi

ravi

```
    scanf("%s", &a);

    printf("%s" , a);

   return 0;
}
```

comments

In programming, comments are hints that a programmer can add to make their code easier to read and understand.

Types of Comments

There are two ways to add comments in C:

1. `//` **- Single Line Comment**
2. `/*...*/` **- Multi-line Comment**

`//` - Single Line Comment	`/*...*/` - Multi-line Comment
#include <stdio.h>	#include <stdio.h>
int main() {	int main() {
// print Hello World to the screen	// print Hello World to the screen
printf("Hello World");	printf("Hello World");

// printf("Hello World"); not using this code	/*printf("Hello World");*/
return 0;	not using code this comment use/*wk.*/
}	return 0;
	}

C Programming Operators

Arithmetic Operators in C

+	-	*	/	%

// Working of arithmetic operators direct value put	// Working of arithmetic operators direct value put
#include <stdio.h>	#include <stdio.h>
int main()	int main()
{	{
int a = 9,b = 4, c;	int a ,b,c;
	//addition
c = a+b;	printf("enter the no=\n");
printf("a+b = %d \n",c);	
c = a-b;	scanf("%d%d",&a,&b);//input value
printf("a-b = %d \n",c);	
c = a*b;	c = a+b;
printf("a*b = %d \n",c);	printf("totalsum= %d\n",c);//ouput print
c = a/b;	
printf("a/b = %d \n",c);	return 0;

c = a%b;	}
printf("Remainder when a divided by b = %d \n",c);	
return 0;	
}	

Scanf input value

// Working of arithmetic operators(+)	// Working of arithmetic operators(*)
#include <stdio.h>	#include <stdio.h>
int main()	int main()
{	{
int a ,b,c;	int a ,b,c;
//addition	//addition
printf("enter the no a=");	printf("enter the no a=");
scanf("%d",&a);//input value	scanf("%d",&a);//input value
printf("enter the no b=");	printf("enter the no b=");
scanf("%d",&b);	scanf("%d",&b);
c = a+b;	c = a*b;
printf("totalsum= %d\n",c);//ouput print	printf("totalsum= %d\n",c);//ouput print
return 0;	return 0;
}	}

// Working of arithmetic operators(-)	// Working of arithmetic operators(-)
#include <stdio.h>	#include <stdio.h>
int main()	int main()
{	{
int a ,b,c;	int a ,b,c;

//addition	//addition
printf("enter the no a=");	printf("enter the no a=");
scanf("%d",&a);//input value	scanf("%d",&a);//input value
printf("enter the no b=");	printf("enter the no b=");
scanf("%d",&b);	scanf("%d",&b);
c = a-b;	c = a/b;
printf("totalsum= %d\n",c);//ouput print	printf("totalsum= %d\n",c);//ouput printf
return 0;	return 0;
}	}

// Working of arithmetic operators(%)
#include <stdio.h>
int main()
{
int a ,b,c;

```
//addition
printf("enter the no a=");
scanf("%d",&a);//input value
printf("enter the no b=");
scanf("%d",&b);
c = a%b;
printf("reminder= %d\n",c);//ouput printf

return 0;
}
```

C Assignment Operators

=	*=	+=	-=	/=	%=

// Working of(+=)assignment operators	ouput	Sovle sum
#include <stdio.h>	enter the no a=4	A=a+b
int main()	enter the no b=2	a=4+2
{	answer is a=11	a=6

int a ,b;		
printf("enter the no a=");		
scanf("%d",&a);//input value		
printf("enter the no b=");		
scanf("%d",&b);//input value		
a +=b;		
printf("answer is a=%d",a);		
return 0;		
}		

// Working of(*=)assignment operators	// Working of(-=)assignment operators
#include <stdio.h>	#include <stdio.h>
int main()	int main()
{	{
int a ,b;	int a ,b;
printf("enter the no a=");	printf("enter the no a=");
scanf("%d",&a);//input value	scanf("%d",&a);//input value
printf("enter the no b=");	printf("enter the no b=");
scanf("%d",&b);//input value	scanf("%d",&b);//input value
a *=b;	a -=b;
printf("answer is a=%d",a);	printf("answer is a=%d",a);
return 0;	return 0;
}	}

// Working of(/=)assignment operators	// Working of(%=)assignment operators
#include <stdio.h>	#include <stdio.h>
int main()	int main()
{	{
int a ,b;	int a ,b;
printf("enter the no a=");	printf("enter the no a=");
scanf("%d",&a);//input value	scanf("%d",&a);//input value
printf("enter the no b=");	printf("enter the no b=");
scanf("%d",&b);//input value	scanf("%d",&b);//input value
a /=b;	a %=b;
printf("answer is a=%d",a);	printf("answer is a=%d",a);
return 0;	return 0;
}	}

C Relational Operators

A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Relational operators are used in decision making and loops.

Operator	Meaning of Operator	Example
==	Equal to	`5 == 3` is evaluated to 0
>	Greater than	`5 > 3` is evaluated to 1
<	Less than	`5 < 3` is evaluated to 0
!=	Not equal to	`5 != 3` is evaluated to 1
>=	Greater than or equal to	`5 >= 3` is evaluated to 1
<=	Less than or equal to	`5 <= 3` is evaluated to 0

```
// Working of ==assignment operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a==b;
   printf("enter the no c=%d",c);
   return 0;
}
```

```
// Working of  > Greater than Relational Operators  operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a>b;
   printf("enter the no c=%d",c);
   return 0;
}
```

```
// Working of  Less than than Relational Operators  operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a<b;
   printf("enter the no c=%d",c);
   return 0;
}
```

```
// Working of  Less than than Relational Operators  operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a!=b;
   printf("enter the no c=%d",c);
   return 0;
}
```

```
// Working of  greater than equal than Relational Operators operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a>=b;
   printf("enter the no c=%d",c);
   return 0;
}
```

```
// Working of  Less than than Relational Operators  operators
#include <stdio.h>
int main()
{
   int a ,b,c;

   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
   c=a<=b;
   printf("enter the no c=%d",c);
   return 0;
}
```

C Logical Operators

An expression containing logical operator returns either 0 or 1 depending upon whether expression results true or false. Logical operators are true =1 AND false 0.

Operator	Meaning	Example
&&	Returns true if both statements are true	If c = 5 and d = 2 then, expression `((c==5) && (d>5))` equals to 0.
||	Returns true if one of the statements is true	If c = 5 and d = 2 then, expression `((c==5) || (d>5))` equals to 1.
!	Reverse the result, returns false if the result is true	If c = 5 then, expression `!(c==5)` equals to 0.

&& operator both are true result is 1	|| operator one are true result is 1

#include <stdio.h>
int main() {
int a,b,c,result;
printf("enter no a=");
scanf("%d",&a);
printf("enter no b=");
scanf("%d",&b);
printf("enter no c=");
scanf("%d",&c);
result= (a==b) && (c>b);
printf("%d",result);
return 0;
}
	#include <stdio.h>
	int main() {
	int a,b,c,result;
	printf("enter no a=");
	scanf("%d",&a);
	printf("enter no b=");
	scanf("%d",&b);
	printf("enter no c=");
	scanf("%d",&c);
	result= (a ==b) || (c<b);
	printf("%d",result);
	return 0;
	}

!not operator	true

#include <stdio.h>
 	
int main() {
  int a,b,result;
  printf("enter no a=");
  scanf("%d",&a);
  printf("enter no b=");
  scanf("%d",&b);
 	
  result= !(a==b);
  printf("%d",result);
 	
  return 0;
}
	#include <stdio.h>
	
	int main() {
	  int a,b,result;
	  printf("enter no a=");
	  scanf("%d",&a);
	  printf("enter no b=");
	  scanf("%d",&b);
	
	  result= !(a !=b);
	  printf("%d",result);
	
	  return 0;
	}

#include <stdio.h>	enter no a=50//output
	enter no b=20
int main() {	1
int a,b,result;	True ans
printf("enter no a=");	
scanf("%d",&a);	
printf("enter no b=");	
scanf("%d",&b);	
result= !(a<b);	
printf("%d",result);	
return 0;	
}	

Not operator	output
#include <stdio.h>	enter no a=10
	enter no b=5
int main() {	enter no c=20
int a,b, c,result;	0
printf("enter no a=");	enter no a=10
scanf("%d",&a);	
printf("enter no b=");	
scanf("%d",&b);	
printf("enter no c=");	
scanf("%d",&c);	
result= !((a>b) &&(a<c));	
printf("%d",result);	
return 0;	
}	

C Increment and Decrement Operators

C programming has two operators increment `++` and decrement `--` to change the value of an operand (constant or variable) by 1.
Increment `++` increases the value by 1 whereas decrement `--` decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

The operators `++` and `--` are used as prefixes. These two operators can also be used as postfixes like `a++` and `a–`

Increment operator और Decrement operator को prefix और postfix forms के साथ सही तरह से समझने के लिए आपको बस नीचे दिए गये कुछ rules याद रखने हैं बस.

Prefix Increment Operator: पहले variable में increment होगा उसके बाद variable use होगा.

Postfix Increment Operator: पहले variable use होगा उसके बाद variable में increment होगा.

Prefix Decrement Operator: पहले variable में decrement होगा उसके बाद variable use होगा.

Postfix Decrement Operator: पहले variable use होगा उसके बाद variable में decrement होगा.

```
a = 5
++a;          // a becomes 6
a++;          // a becomes 7
--a;          // a becomes 6
a--;          // a becomes 5
```

#include <stdio.h>	// Online C compiler to run C program online
int main() {	#include <stdio.h>
int a;	int main() {
	int a;
// 5 is displayed	
// Then, a is increased to 6.	// 5 is displayed
printf("enter the no=");	// Then, a is increased to 6.
scanf("%d",&a);	printf("enter the no=");
printf("%d\n", a++);// postfix a++	scanf("%d",&a);
	printf("%d\n", a--);// postfix a--
// a is increased to 6	
// Then, it is displayed.	// a is increased to 6
printf("%d\n", ++a);// prefix ++a	// Then, it is displayed.
	printf("%d\n", --a);// prefix --a
return 0;	
}	return 0;

Other Operators

Comma Operator

Comma operators are used to link related expressions together. For example:

```
int a, c = 5, d;
```

The sizeof operator

The `sizeof` is a unary operator that returns the size of data (constants, variables, array, structure, etc).

Example 6: sizeof Operator

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    double c;
    char d;
    printf("Size of int=%lu bytes\n",sizeof(a));
    printf("Size of float=%lu bytes\n",sizeof(b));
    printf("Size of double=%lu bytes\n",sizeof(c));
    printf("Size of char=%lu byte\n",sizeof(d));
    return 0;
}
```

C Bitwise Operators

During computation, mathematical operations like: addition, subtraction, multiplication, division, etc are converted to bit-level which makes processing faster and saves power.

Bitwise operators are used in C programming to perform bit-level operations.

Operators	Meaning of operators
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	Bitwise complement
<<	Shift left
>>	Shift right

Bitwise AND Operator &

The output of bitwise AND is 1 if the corresponding bits of two operands is 1. If either bit of an operand is 0, the result of corresponding bit is evaluated to 0.

In C Programming, the bitwise AND operator is denoted by &.

Let us suppose the bitwise AND operation of two integers 12 and 25.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bit Operation of 12 and 25
  00001100
& 00011001
  ________
  00001000  = 8 (In decimal)
```

```c
#include <stdio.h>
int main()
{
   int a ,b,c;
   //addition
   printf("enter the no=");
   scanf("%d",&a);//input value
   printf("enter the no=");
   scanf("%d",&b);//input value
   c = a&b;
printf("ans= %d\n",c);//ouput print
return 0;
}
```

Bitwise OR Operator |

The output of bitwise OR is 1 if at least one corresponding bit of two operands is 1. In C Programming, bitwise OR operator is denoted by |.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise OR Operation of 12 and 25
  00001100
| 00011001
  ________
  00011101  = 29 (In decimal)
```

```
#include <stdio.h>
int main()
{
   int a ,b,c;
   //addition
   printf("enter the no=");
   scanf("%d",&a);//input value
   printf("enter the no=");
   scanf("%d",&b);//input value


   c = a|b;
printf("ans= %d\n",c);//ouput print
return 0;
}
```

Bitwise XOR (exclusive OR) Operator ^

The result of bitwise XOR operator is 1 if the corresponding bits of two operands are opposite. It is denoted by ^.

```
12 = 00001100 (In Binary)
25 = 00011001 (In Binary)

Bitwise XOR Operation of 12 and 25
  00001100
^ 00011001
  ________
  00010101  = 21 (In decimal)
```

```
#include <stdio.h>
int main()
{
   int a ,b,c;
   //addition
   printf("enter the no=");
   scanf("%d",&a);//input value
   printf("enter the no=");
   scanf("%d",&b);//input value
   c = a^b;
printf("ans= %d\n",c);//ouput print
return 0;
}
```

Bitwise Complement Operator ~

Bitwise complement operator is a unary operator (works on only one operand). It changes 1 to 0 and 0 to 1. It is denoted by `~`.

```
35 = 00100011 (In Binary)

Bitwise complement Operation of 35
~ 00100011
  ________
  11011100  = 220 (In decimal)
```

Twist in Bitwise Complement Operator in C Programming

The bitwise complement of 35 (`~35`) is -36 instead of 220, but why?

For any integer `n`, bitwise complement of `n` will be `-(n + 1)`. To understand this, you should have the knowledge of 2's complement.

2's Complement

Two's complement is an operation on binary numbers. The 2's complement of a number is equal to the complement of that number plus 1. For example:

```
 Decimal         Binary           2's complement
   0            00000000          -(11111111+1) = -00000000 = -
0(decimal)
   1            00000001          -(11111110+1) = -11111111 = -
256(decimal)
   12           00001100          -(11110011+1) = -11110100 = -
244(decimal)
   220          11011100          -(00100011+1) = -00100100 = -
36(decimal)

Note: Overflow is ignored while computing 2's complement.
```

The bitwise complement of 35 is 220 (in decimal). The 2's complement of 220 is -36. Hence, the output is `-36` instead of 220.

Bitwise Complement of Any Number N is -(N+1). Here's how:

```
bitwise complement of N = ~N (represented in 2's complement form)
2'complement of ~N= -(~(~N)+1) = -(N+1)
```

Example 4: Bitwise complement

```
#include <stdio.h>
int main() {
   printf("Output = %d\n", ~35);
   printf("Output = %d\n", ~-12);
   return 0;
}
```

Shift Operators in C programming

There are two shift operators in C programming:

- **Right shift operator**
- **Left shift operator.**

Right Shift Operator

Right shift operator shifts all bits towards right by certain number of specified bits. It is denoted by `>>`.

```
212 = 11010100 (In binary)
212 >> 2 = 00110101 (In binary) [Right shift by two bits]
212 >> 7 = 00000001 (In binary)
212 >> 8 = 00000000
212 >> 0 = 11010100 (No Shift)
```

Left Shift Operator

Left shift operator shifts all bits towards left by a certain number of specified bits. The bit positions that have been vacated by the left shift operator are filled with 0. The symbol of the left shift operator is `<<`.

```
212 = 11010100 (In binary)
212<<1 = 110101000 (In binary) [Left shift by one bit]
212<<0 = 11010100 (Shift by 0)
212<<4 = 110101000000 (In binary) =3392(In decimal)
```

Example #5: Shift Operators

```c
#include <stdio.h>

int main() {

    int num=212, i;
```

```c
    for (i = 0; i <= 2; ++i) {
        printf("Right shift by %d: %d\n", i, num >> i);

    }
    printf("\n");

    for (i = 0; i <= 2; ++i) {
        printf("Left shift by %d: %d\n", i, num << i);

    }

    return 0;
}
```

Run C

```
Right Shift by 0: 212
Right Shift by 1: 106
Right Shift by 2: 53

Left Shift by 0: 212
Left Shift by 1: 424
Left Shift by 2: 848
```

The conditional operator

The conditional operator in C is kind of similar to the if-else statement as it follows the same algorithm as of if-else statement but the conditional operator takes less space and helps to write the if-else statements in the shortest way possible. It is also known as the ternary operator in C as it operates on three operands.

Syntax of Conditional/Ternary Operator in C

The conditional operator can be in the form

```
variable = Expression1 ? Expression2 : Expression3;
```

Or the syntax can also be in this form

```
variable = (condition) ? Expression2 : Expression3;
```

```
//Program to Store the greatest of the two Numbers using the ternary conditional operator
#include <stdio.h>
int main()
{
   int a,b;
   //addition
   printf("enter the no a=");
   scanf("%d",&a);//input value
   printf("enter the no b=");
   scanf("%d",&b);//input value
(a > b) ? printf(" a is greater than b that is %d > %d",a, b): printf("a is not greater than b that is %d > %d",
                a, b);

    return 0;
}
```

SOME EXAMPLE IN OPERATOR

```
//find item  bill
#include <stdio.h>
int main() {

   char x[10];
   int price,quantity,bill;
   float discount,netbill;

   printf("\n enter item name=");
   scanf("%s",&x);
    printf("\n enter price=");
   scanf("%d",&price);
   printf("\n enter quantity=");
   scanf("%d",&quantity);
   bill=price*quantity;
   printf("totalbill=%d",bill);
   discount=(float)bill/100*10;
   printf("\ndiscount=%f",discount);
   netbill=bill-discount;
   printf("\n netbill=%f", netbill);

   return 0;
}
```

//FIND RESULT

```
#include <stdio.h>
int main() {

   char x[10];
   int math,pun,hindi,total;
   float per;

   printf("\n enter student name=");
   scanf("%s",&x);
    printf("\n enter math=");
   scanf("%d",&math);
   printf("\n enter pun=");
   scanf("%d",&pun);
   printf("\n enter hindi=");
   scanf("%d",&hindi);
   total=math+pun+hindi;
   printf("total=%d",total);
   per=(float)total/300*100;
   printf("\npercentage=%f",per);
   return 0;
}
```

```
#include <stdio.h>
int main() {

    char x[10];
    int math,pun,hindi,total,finalresult;
    float per;

    printf("\n enter student name=");
    scanf("%s",&x);
     printf("\n enter math=");
    scanf("%d",&math);
    printf("\n enter pun=");
    scanf("%d",&pun);
    printf("\n enter hindi=");
    scanf("%d",&hindi);
    total=math+pun+hindi;
    printf("total=%d",total);
    per=(float)total/300*100;
    printf("\npercentage=%f",per);
    finalresult=per>33;
    printf("\n33abovepass",finalresult);
    return 0;
}
```

//Find Rectangle

```
//find  rectangle
#include <stdio.h>
int main() {

int length,breadth,total;
printf("enter length=");
scanf("%d",&length);
printf("enter length=");
scanf("%d",&breadth);
total=length*breadth;
printf("\nrectangle is : %d",total);
printf("\nPerimeter of rectangle is : %d", 2 * (length+breadth));
    return 0;
}
```

//Find Tempratue

```
//find  tempratue
#include <stdio.h>
int main() {

float f,c;
printf("enter temprature=");
scanf("%f",&f);
c=(f-32)*5/9;
printf("\n the temprature is : %.2f",c);//.2f are used in last decimal number

decrease

      return 0;
}
```

//Find Average

```
#include <stdio.h>
int main() {
float score,over,avg;
printf("\n enter score=");
scanf("%f",&score);
printf("\n enter over=");
scanf("%f",&over);
avg=score/over;
printf("\n the average is : %.1f",avg);
//printf("\n the average is : %.2f",avg);
//printf("\n the average is : %f",avg);
//%.2f are used in last decimal number decrease 2 means decimal no last
two or one possible use 2 and 1 keep
    return 0;
}
```

The Idea Is Swapping

Assign X To A Temp Variable: Temp = X
Assign Y To X: X = Y
Assign Temp To Y: Y = Temp
Example:

X = 100, Y = 200
After Line 1: Temp = X Temp = 100
After Line 2: X = Y X = 200
After Line 3 : Y = Temp Y = 100

Using Temporary Variable
Below Is The C Program To Swap Two Numbers Using A Temporary Variable:

// C program to swap two variables	Ouput
#include <stdio.h>	Enter Value of x 5
	Enter Value of y 4
// Driver code	After Swapping: x = 4, y = 5
int main()	Enter Value of x 5
{	Enter Value of y 4
int x, y;	After Swapping: x = 4, y = 5
printf("Enter Value of x ");	
scanf("%d", &x);	
printf("\nEnter Value of y ");	
scanf("%d", &y);	
int temp = x;	
x = y;	
y = temp;	
printf("\nAfter Swapping: x = %d, y = %d",	
x, y);	
return 0;	
}	

```
#include<stdio.h>
int main() {
  double first, second, temp;
  printf("Enter first number: ");
  scanf("%lf", &first);
  printf("Enter second number: ");
  scanf("%lf", &second);
  // value of first is assigned to temp
  temp = first;
  // value of second is assigned to first
  first = second;
  // value of temp (initial value of first) is assigned to second


  second = temp;
  // %.2lf displays number up to 2 decimal points
  printf("\nAfter swapping, first number = %.2lf\n", first);
  printf("After swapping, second number = %.2lf", second);
  return 0;
}
```

//Find Interset Bill

```
//find interset  bill
#include <stdio.h>
int main() {

    char x[10];
    float amount,rate,time;
    int   interest,totalamount ;

    printf("\n entename=");
    scanf("%s",&x);
     printf("\n enter amount=");
    scanf("%f",&amount);
    printf("\n enter rate=");
    scanf("%f",&rate);
     printf("\n enter time=");
    scanf("%f",&time);
    interest=amount*rate*time/100;
    printf("totalbill=%d", interest);

    totalamount=amount+interest;
    printf("\n netbill=%d", totalamount);


    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int a;
    float b;
    printf("Enter integer and then a float: ");

    // Taking multiple inputs
    scanf("%d%f", &a, &b);
    printf("You entered %d and %f", a, b);
    return 0;
}
```

//Float and Double Input/Output

```
#include <stdio.h>
int main()
{
    float num1;
    double num2;
    printf("Enter a number: ");
    scanf("%f", &num1);
    printf("Enter another number: ");
    scanf("%lf", &num2);
    printf("num1 = %f\n", num1);
    printf("num2 = %lf", num2);
    return 0;
}
```

Types of Control Statements

There are four types of control statements in C:

- Decision making statements (if, if-else)
- Selection statements (switch-case)
- Iteration statements (for, while, do-while)
- Jump statements (break, continue, goto)

If= The if Statement

Use the `if` statement to specify a block of code to be executed if a condition is `true`.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
}
```

```
// Program to display a marks if

#include <stdio.h>
int main() {
   int marks;
   printf("Enter an integer: ");
   scanf("%d", &marks);

   // true if number is less than 0
   if (marks>=33) {
      printf("33 above  marks in english");
   }

   return 0;
}
```

```
// Program to display a number if

#include <stdio.h>
int main() {
    int a,b;

    printf("Enter an integer a: ");
    scanf("%d", &a);
    printf("Enter an integer b: ");
    scanf("%d", &b);


    // true if number is less than 0
    if (a >=b) {
        printf(" a  greater%d>=%d",a,b);
    }


    return 0;
}

```

//A GREATER THAN

```
#include <stdio.h>

int main()
{

    int a,b;
    // if statement with true condition
    printf("enter the no a=");
    scanf("%d",&a);
     printf("enter the no b=");
    scanf("%d",&b);
    if (a < b) {
        printf("a is less than b");
    }

    // if statement with false condition
    if (a > b) {
        printf("a is greater than b");
    }

    return 0;
}
```

// Program to display a number if it is negative

```
#include <stdio.h>
int main() {
    int number;
    printf("Enter an integer: ");
    scanf("%d", &number);
    // true if number is less than 0
    if (number < 0) {
        printf("You entered %d.\n", number);
    }
    printf("The if statement is easy.");
    return 0;
}
```

```
#include <stdio.h>
int main()
{
   int a, b, c;
    printf("Enter three numbers?");
   scanf("%d %d %d",&a,&b,&c);
   if(a>b && a>c)
   {
      printf("%d is largest",a);
   }
   if(b>a  && b > c)
   {
      printf("%d is largest",b);
   }
   if(c>a && c>b)
   {
      printf("%d is largest",c);
   }
   if(a == b && a == c)
   {
      printf("All are equal");
   }
}
```

THE ELSE STATEMENT

Use the `else` statement to specify a block of code to be executed if the condition is `false`.

Syntax

```
if (condition) {
  // block of code to be executed if the condition is true
} else {
  // block of code to be executed if the condition is false
}
```

/// Check whether an integer is odd or even

```
#include <stdio.h>
int main() {
   int number;
   printf("Enter an integer: ");
   scanf("%d", &number);
   // True if the remainder is 0
   if  (number%2 == 0) {
      printf("%d is an even integer.",number);
   }
   else {
      printf("%d is an odd integer.",number);
   }
   return 0;
}
```

```
#include <stdio.h>
int main()
{
  int num,rem;
  printf("Enter an number=");
  scanf("%d", &num);
  rem=num%2;//divede reminder 0 that means true even
  if(rem==0){
    printf("%d Even\n",num);
}
  else{
    printf("%d=Odd",num);
}
  return 0;
}
```

//VOTER SHEET

```
#include <stdio.h>
int main()
{
  int age;
  printf("Enter an integer\n");
  scanf("%d", &age);

  if (age>=18){
    printf("can vote");
  }
  else{
    printf("can not vote");
  }
  return 0;
}
```

```
#include <stdio.h>
int main() {
   char x[10];
   int math,pun,hindi,total;
   float per;
   printf("\n enter student name=");
   scanf("%s",&x);
    printf("\n enter math=");
   scanf("%d",&math);
   printf("\n enter pun=");
   scanf("%d",&pun);
   printf("\n enter hindi=");
   scanf("%d",&hindi);
   total=math+pun+hindi;
   printf("total=%d",total);
   per=(float)total/300*100;
   printf("\npercentage=%f\n",per);
   if (per>=33){
   printf("pass");
  }
  else{
   printf("fail");
  }
   return 0;
}
```

```
//enter letter upercase or not in if else in c
#include <stdio.h>
int main() {
   // Write C code here
   char ch;
   /* Input character from user */
   printf("Enter any character: ");
   scanf("%c", &ch);
  if(ch >= 'A' && ch <= 'Z')
   {
      printf("%c is uppercase alphabet.", ch);
   }
   else{
   printf("letter is not uppercase");
   }
return 0;
}
```

The else if Statement

Use the `else if` statement to specify a new condition if the first condition is `false`.

C if...else if Ladder

The `if...else` statement executes two different codes depending upon whether the test expression is true or false. Sometimes, a choice has to be made from more than 2 possibilities.
The if...else if ladder allows you to check between multiple test expressions and execute different statements.

Syntax

```
if (condition1) {
  // block of code to be executed if condition1 is true
} else if (condition2) {
  // block of code to be executed if the condition1 is
false and condition2 is true
} else {
  // block of code to be executed if the condition1 is
false and condition2 is false
}
```

```
// Program to relate two integers using =, > or < symbol
#include <stdio.h>
int main() {
   int number1, number2;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);
   //checks if the two integers are equal.
   if(number1 == number2) {
      printf("Result: %d = %d",number1,number2);
   }
   //checks if number1 is greater than number2.
   else if (number1 > number2) {
      printf("Result: %d > %d", number1, number2);
   }
   //checks if both test expressions are false
   else {
      printf("Result: %d < %d",number1, number2);
   }

   return 0;
}
```

```
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number==10){
printf("number is equals to 10");
}
else if(number==50){
printf("number is equal to 50");
}
else if(number==100){
printf("number is equal to 100");
}
else{
printf("number is not equal to 10, 50 or 100");
}
return 0;
}
```

```
/*** C program to check whether a character is uppercase or lowercase */
#include <stdio.h>
int main() {
   char character;
   printf("Enter a character: ");
   scanf(" %c", &character);
   if (character >= 'A' && character <= 'Z') {
      printf("The character %c is uppercase.\n", character);
   }
   else if (character >= 'a' && character <= 'z') {
      printf("The character %c is lowercase.\n", character);
   }
   else {
      printf("The input  %c is invalid.\n", character);
   }
   return 0;
}
```

```
#include <stdio.h>
int main() {
   char x[10];
   int math,pun,hindi,total;
   float per;
   printf("\n enter student name=");
   scanf("%s",&x);
    printf("\n enter math=");
   scanf("%d",&math);
   printf("\n enter pun=");
   scanf("%d",&pun);
   printf("\n enter hindi=");
   scanf("%d",&hindi);
   total=math+pun+hindi;
   printf("total=%d",total);
   per=(float)total/300*100;
   printf("\npercentage=%f\n",per);
if( per> 85 &&  per <= 100)
   {
      printf("Congrats ! you scored grade A ...");
   }
   else if ( per > 60 &&  per <= 85)
   {
      printf("You scored grade B + ...");
   }
   else if ( per > 40 &&  per <= 60)
   {
      printf("You scored grade B ...");
   }
   else if ( per > 30 &&  per <= 40)
   {
      printf("You scored grade C ...");
   }
   else
   {
      printf("Sorry you are fail ...");
   }
   return 0;
}
```

```
#include <stdio.h>
int main()
{
   int marks;
   printf("Enter your marks?");
   scanf("%d",&marks);
   if(marks > 85 && marks <= 100)
   {
      printf("Congrats ! you scored grade A ...");
   }
   else if (marks > 60 && marks <= 85)
   {
      printf("You scored grade B + ...");
   }
   else if (marks > 40 && marks <= 60)
   {
      printf("You scored grade B ...");
   }
   else if (marks > 30 && marks <= 40)
   {
      printf("You scored grade C ...");
   }
   else
   {
      printf("Sorry you are fail ...");
   }
return 0;
}
```

Example 4: Nested if statement...else

This program given below relates two integers using either `<`, `>` and `=` similar to the `if...else` ladder's example. However, we will use a nested `if...else` statement to solve this problem.

```
#include <stdio.h>
int main() {
   int number1, number2;
   printf("Enter two integers: ");
   scanf("%d %d", &number1, &number2);

   if (number1 >= number2) {
     if (number1 == number2) {
       printf("Result: %d = %d",number1,number2);
     }
     else {
       printf("Result: %d > %d", number1, number2);
     }
   }
   else {
       printf("Result: %d < %d",number1, number2);
   }

   return 0;
}
```

```
#include <stdio.h>
int main() {
  // variables to store the three numbers
  int a, b, c;

  //take input from the user
  scanf("%d %d %d", &a, &b, &c);

  //if else condition to check whether the first number is greater than the second
  if (a > b) {

    //nested if else condition to check if a>c
    if (a > c) {
      //a is greatest
      printf("%d a =is greatest a", a);
    } else {
      //c is the greatest
      printf("%d  c greatest than a", c);
    }

  } else {

    //nested if else condition to check if b>c
    if (b > c) {
      //b is greatest
      printf("%d b is greatest ", b);
    } else {
      //c is the greatest
      printf("%d  c greatest b", c);
    }
  }
  return 0;
}
```

LOOP

In programming, a loop is used to repeat a block of code until the specified condition is met.

C programming has three types of loops:

1. **for loop**
2. **while loop**
3. **do...while loop**

For Loop

The syntax of the `for` loop is:

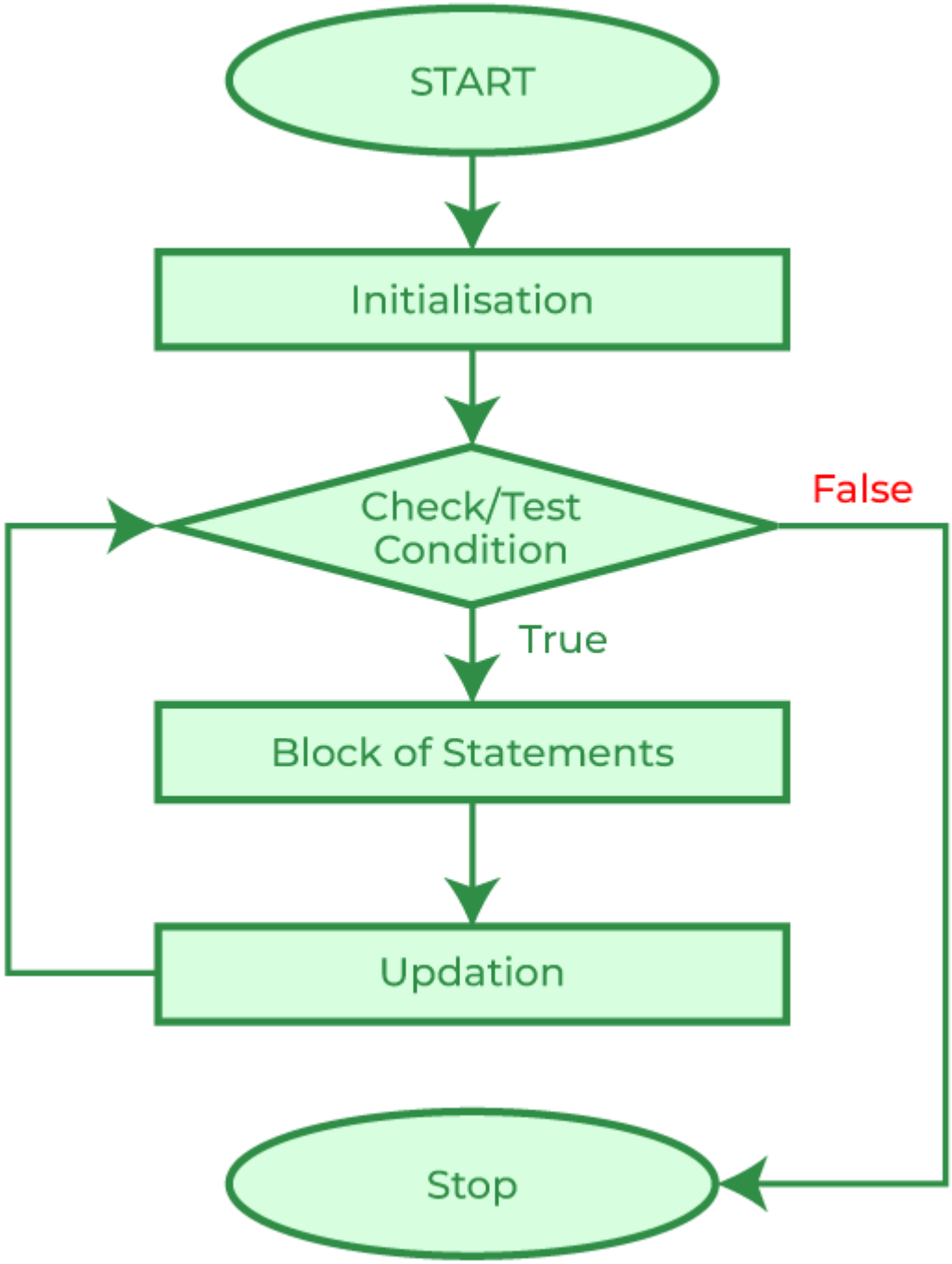
```
for (initializationStatement; testExpression; updateStatement)
{
    // statements inside the body of loop
}
```

How for loop works?

- **The initialization statement is executed only once.**
- **Then, the test expression is evaluated. If the test expression is evaluated to false, the `for` loop is terminated.**
- **However, if the test expression is evaluated to true, statements inside the body of the `for` loop are executed, and the update expression is updated.**
- **Again the test expression is evaluated.**

This process goes on until the test expression is false. When the test expression is false, the loop terminates.

for loop Flowchart



```
// Print numbers from 1 to 10
#include <stdio.h>

int main() {
  int i;

  for (i = 1; i <= 11; ++i)
  {
    printf("%d\n ", i);
  }
  return 0;
}
```

```
// Print numbers from 1 to 10 scanf using

#include <stdio.h>

int main() {

  int i,num;

  printf("enter no=");

  scanf("%d",&num);

  for (i = 1; i < num; i++)

  {

    printf("%d\n",i);//I anser

  }

  return 0;

}
```

```
// Print numbers  scanf using
#include<stdio.h>
int main(){
        int i,num;
         printf("enter no i=");
         scanf("%d",&i);
         printf("enter no num=");
         scanf("%d",&num);
         for(i;i<=num;i++){
                 printf("%d\n",i);
         }

         return 0;
}
```

```
//Even no
#include <stdio.h>

int main() {
  int i,num;

  printf("enter no=");
  scanf("%d",&num);

  for (i = 2; i < num; i=i+2)
  {
    printf("%d\n",i);
  }
  return 0;
}
```

```
//Odd no
#include <stdio.h>
int main() {
  int i,num;
  printf("enter no=");
  scanf("%d",&num);
  for (i = 1; i < num; i=i+2)
  {
    printf("%d\n",i);
  }
  return 0;
}

```

i is initialized to 1.
The test expression i < 11 is evaluated. Since 1 less than 11 is true, the body of for loop is executed. This will print the 1 (value of i) on the screen.
The update statement ++i is executed. Now, the value of i will be 2. Again, the test expression is evaluated to true, and the body of for loop is executed. This will print 2 (value of i) on the screen.
Again, the update statement ++i is executed and the test expression i < 11 is evaluated. This process goes on until i becomes 11.
When i becomes 11, i < 11 will be false, and the for loop terminates.

```
//table display count no
#include<stdio.h>
int main(){
int i=1,number=0;
printf("Enter a number: ");
scanf("%d",&number);
for(i=1;i<=10;i++){
printf("%d*%d=%d \n",number,i,(number*i));
}
return 0;
}
```

/ /Print numbers character

```
#include <stdio.h>

int main() {
  int i,num;

  printf("enter no=");
  scanf("%d",&num);

  for (i = 1; i < num; i++)
  {
    printf("hello\n");
  }
  return 0;
}
```

Infinitive for loop in C

To make a for loop infinite, we need not give any expression in the syntax. Instead of that, we need to provide two semicolons to validate the syntax of the for loop. This will work as an infinite for loop.

```
#include<stdio.h>
void main ()
{
   for(;;)
   {
       printf("welcome to javatpoint");
   }
}
```

//Print for loop without curly braces

```
#include <stdio.h>
int main()
{
        int i;
        // for loop without curly braces
        for (i = 1; i <= 10; i++)
            printf("%d ", i);
        printf("\nThis statement executes after for loop end!!!!"); // Statement print only once

        return 0;
}
```

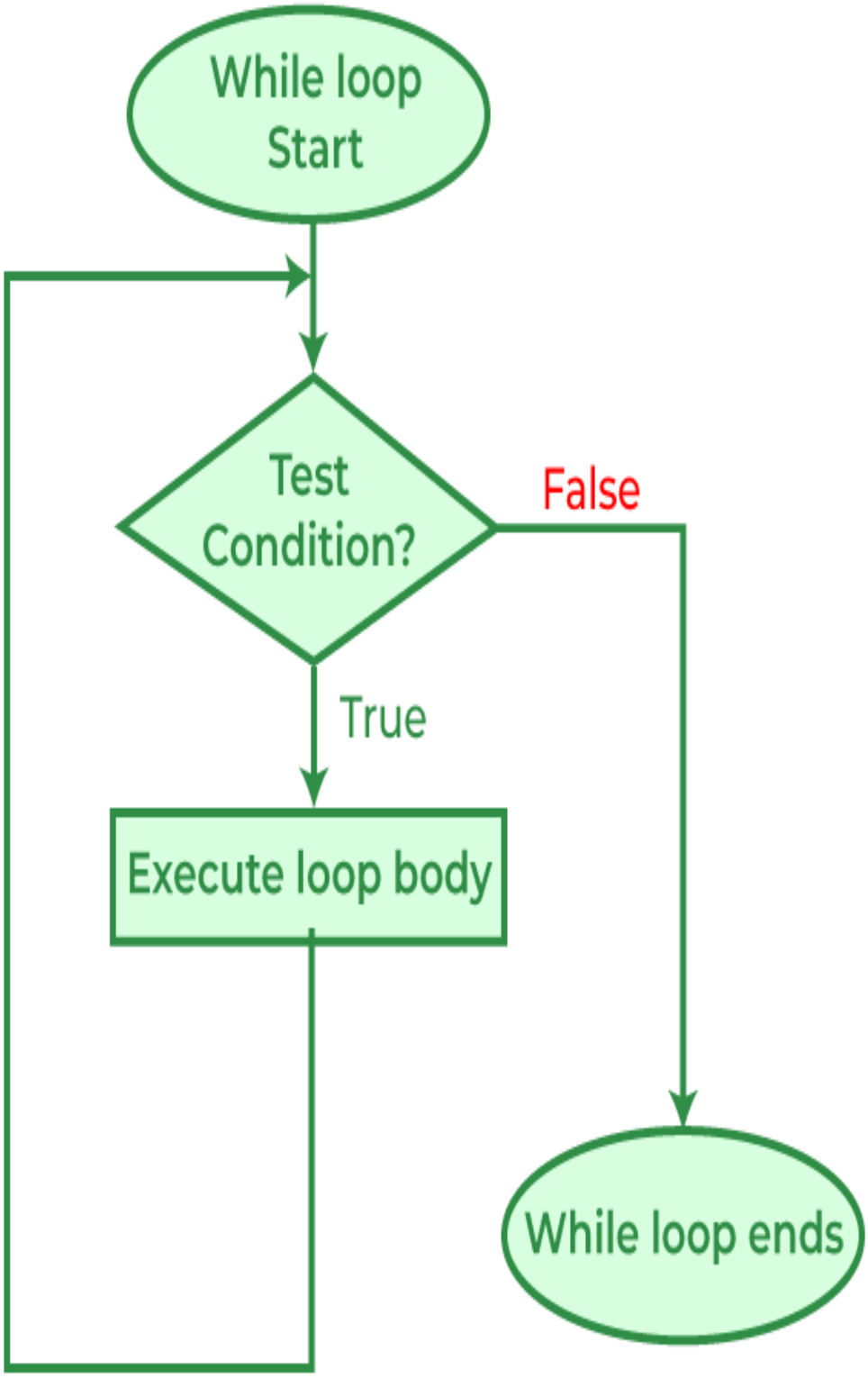
while loop

The syntax of the `while` loop is:

```
while (testExpression) {
  // the body of the loop
}
```

How while loop works?

- The `while` loop evaluates the `testExpression` inside the parentheses `()`.
- If `testExpression` is true, statements inside the body of `while` loop are executed. Then, `testExpression` is evaluated again.
- The process goes on until `testExpression` is evaluated to false.
- If `testExpression` is false, the loop terminates (ends).


While loop Start
Test Condition?
False
True
Execute loop body
While loop ends

```
// Print numbers no

#include <stdio.h>
int main() {
  int i=0,no;
  printf("enter no= ");
  scanf("%d",&no);//input user no

  while (i <= no) {
    printf("%d\n", i);//output i
    i++;
  }

  return 0;
}
```

// Print numbers no, I,and no print

```
#include <stdio.h>
int main() {
  int i,no;
  printf("enter i= ");
  scanf("%d",&i);//input user no
  printf("enter no= ");
  scanf("%d",&no);//input user no
  while (i <= no) {
    printf("%d\n", i);//output i
    i++;
  }

  return 0;
}
```

```
// Print numbers from 1 to 5
#include <stdio.h>
int main() {
  int i = 1;

  while (i <= 5) {
    printf("%d\n", i);
    ++i;
  }
  return 0;
}
```

```
// Print numbers msg
#include <stdio.h>
int main() {
  int i=0,no;
  printf("enter no= ");
  scanf("%d",&no);//input user no
  while (i <= no) {
    printf("hello\n", i);//output i
    i++;
  }
  return 0;
}
```

do...while loop

The `do..while` loop is similar to the `while` loop with one important difference. The body of `do...while` loop is executed at least once. Only then, the test expression is evaluated.

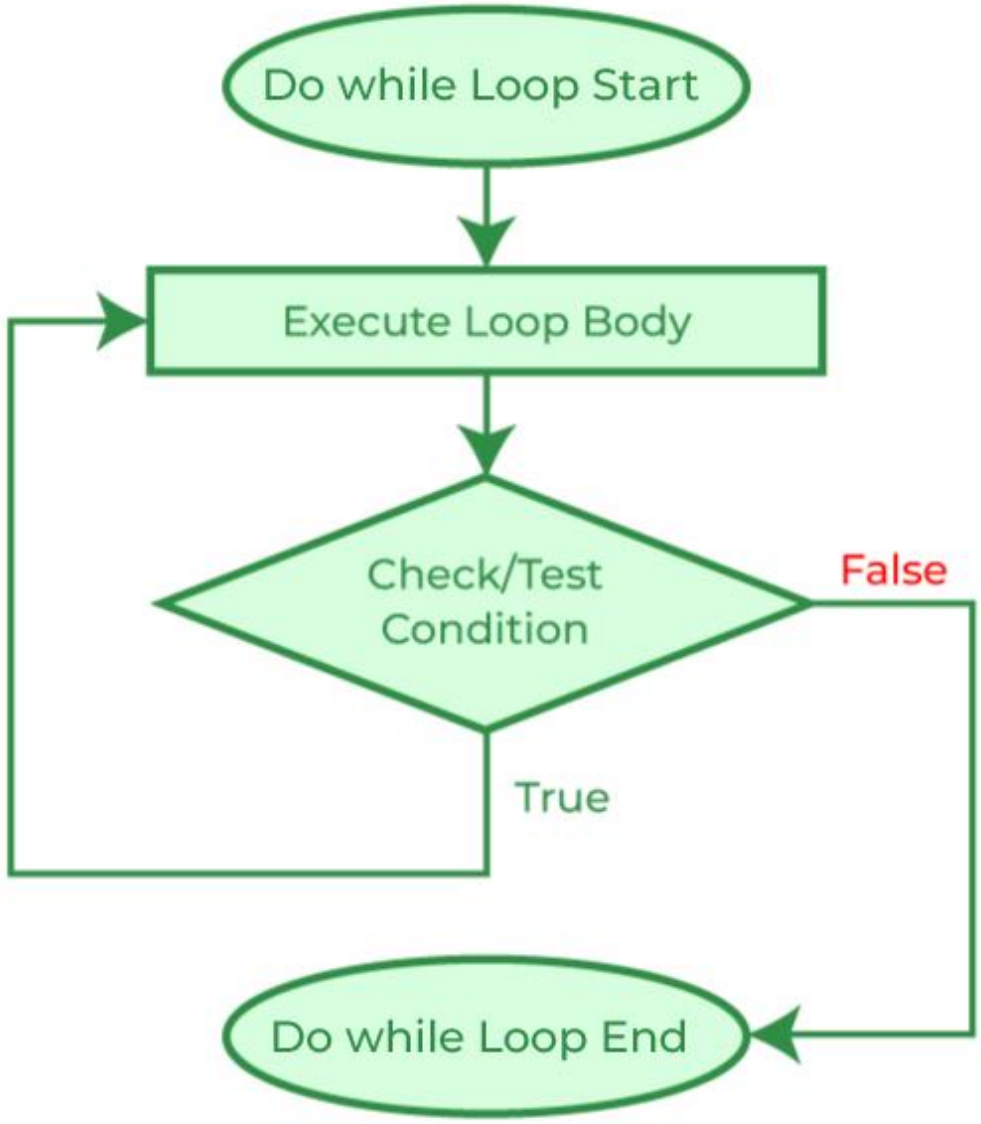
The syntax of the `do...while` loop is:

```
do {
  // the body of the loop
}
while (testExpression);
```

How do...while loop works?

- The body of `do...while` loop is executed once. Only then, the `testExpression` is evaluated.
- If `testExpression` is true, the body of the loop is executed again and `testExpression` is evaluated once more.
- This process goes on until `testExpression` becomes false.
- If `testExpression` is false, the loop ends.

Flowchart of do...while Loop



// C Program to demonstrate the use of do...while loop iand no print

```
#include <stdio.h>
int main()
{
// loop variable declaration and initialization
int i,no;
// do while loop
printf("enter no=");
scanf("%d",&i);
printf("enter no=");
scanf("%d",&no);//input no
do {
printf("%d\n",i);//output i
i++;
} while (i <= no);
return 0;
}
```

```
// C Program to demonstrate the use of do...while loop
#include <stdio.h>

int main()
{

// loop variable declaration and initialization
int i = 0;
// do while loop
do {
printf("%d\n",i);
i++;
} while (i < 3);
return 0;
}
```

```
// C Program to demonstrate the use of do...while loop
#include <stdio.h>
int main()
{
// loop variable declaration and initialization
int i=0,no;
// do while loop
printf("enter no=");
scanf("%d",&no);//input no
do {
printf("%d\n",i);//output i
i++;
} while (i <= no);
return 0;
}
```

```
// C Program to demonstrate the use of do...while loop
#include <stdio.h>

int main()
{

    // loop variable declaration and initialization
    int i=0,no;
    // do while loop
    printf("enter no=");
    scanf("%d",&no);//input no
    do {
        printf("hi\n",i);//output i
        i++;
    } while (i <= no);

    return 0;
}
```

```
// C Program to print multiplication table using do...while
// loop
#include <stdio.h>

int main()
{
    int N = 5, i = 1;

    do {
        printf("%d\tx\t%d\t=\t%d\n", N, i, N * i);
    } while (i++ < 10);
    return 0;
}
```

```
// C Program to print multiplication table using do...while
// loop
#include <stdio.h>
int main()
{
int i , n ;
printf("enter tsble no =");
scanf("%d",&n);//input n number
        do {
printf("%d\tx\t%d\t=\t%d\n",i,n,i*n); //i=0;n=4//(i)0 x(no)6=0
        } while (i++ < 10);//increment i++//
        return 0;
}
```

C break

The break statement ends the loop immediately when it is encountered. Its syntax is:

```
break;
```

The break statement is almost always used with `if...else` statement inside the loop.

```
// C Program to demonstrate break statement with for loop
#include <stdio.h>
int main()
{
   // using break inside for loop to terminate after 2
   // iteration
   printf("break in for loop\n");
   for (int i = 1; i < 5; i++) {
      if (i == 3) {
         break;
      }
      else {
         printf("%d ", i);
      }
   }

   return 0;
}
```

```
// C Program to demonstrate break statement with for loop
#include <stdio.h>

int main()
{
 int n;
   // using break inside for loop to terminate after 2
   // iteration

   printf("break in for loop\n");
    scanf("%d",&n);
   for (int i = 1; i < n; i++) {
      if (i == 4) {
         break;
      }
      else {
         printf("%d ", i);
      }
   }

   return 0;
}
```

C continue

The `continue` statement skips the current iteration of the loop and continues with the next iteration. Its syntax is:

```
continue;
```

```
// C program to explain the use
// of continue statement with for loop

#include <stdio.h>

int main()
{
// for loop to print 1 to 8
  for (int i = 1; i <= 8; i++) {
     // when i = 4, the iteration will be skipped and for
      // will not be printed
      if (i == 4) {
         continue;
      }
      printf("%d ", i);
   }
   printf("\n");

   return 0;
}
```

```
// C program to explain the use
// of continue statement with for loop

#include <stdio.h>

int main()
{
    int n;
    // for loop to print 1 to 8
      printf("continue in for loop\n");
     scanf("%d",&n);

    for (int i = 1; i <= n; i++) {
        // when i = 4, the iteration will be skipped and for
        // will not be printed
        if (i == 4) {
            continue;
        }
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

```
#include <stdio.h>

int main()
{
  int i,n;
    // continue  loop to print 1 to 8
       printf("continue in for loop\n");
     scanf("%d",&n);
// while loop to print 1 to 8
    while (i < n) {
        // when i = 4, the iteration will be skipped and for
        // will not be printed
        i++;
        if (i == 4) {
            continue;
        }
        printf("%d ", i);
    }
    return 0;
}
```

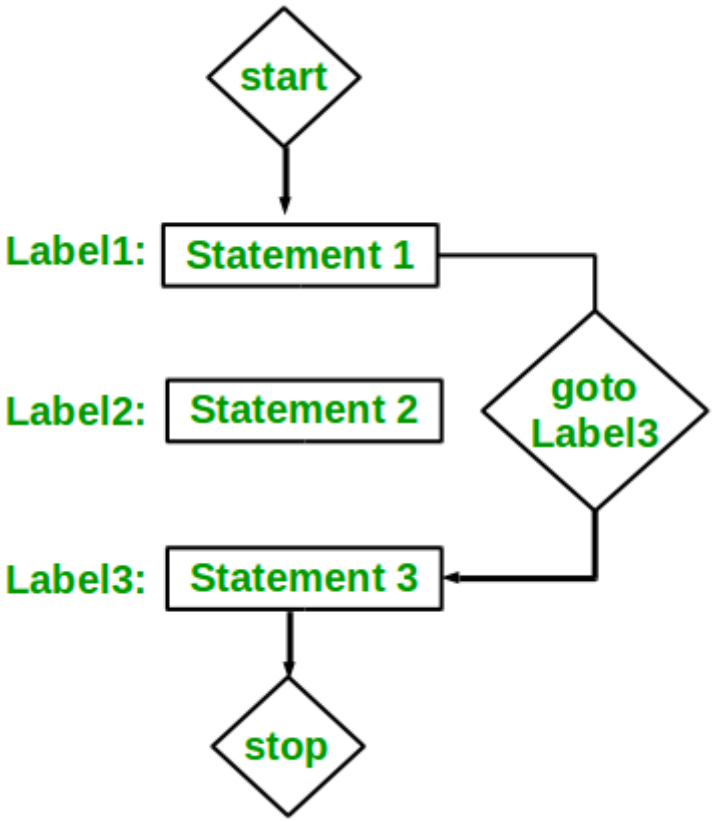
```
// C program to demonstrate difference between
// continue and break
#include <stdio.h>

int main()
{

    printf("The loop with break produces output as: \n");

    for (int i = 1; i <= 7; i++) {

        // Program comes out of loop when
        // i becomes multiple of 3.
        if (i == 3)
            break;
        else
            printf("%d ", i);
    }

    printf("\nThe loop with continue produces output as: \n");
    for (int i = 1; i <= 7; i++) {

        // The loop prints all values except
        // those that are multiple of 3.
        if (i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

he **C goto statement** is a jump statement which is sometimes also referred to as an **unconditional jump** statement. The goto statement can be used to jump from anywhere to anywhere within a function.

Syntax:

```
Syntax1       |   Syntax2
-----------------------------
goto label;   |    label:
.             |    .
.             |    .
.             |    .
label:        |    goto label;
```

In the above syntax, the first line tells the compiler to go to or jump to the statement marked as a label. Here, the label is a user-defined identifier that indicates the target statement. The statement immediately followed after 'label:' is the destination statement. The 'label:' can also appear before the 'goto label;' statement in the above syntax.



```
// C program to print numbers
// from 1 to 10 using goto statement
#include <stdio.h>

// function to print numbers from 1 to 10
void printNumbers()
{
    int n = 1;
label:
    printf("%d ", n);
    n++;
    if (n <= 10)
        goto label;
}

// Driver program to test above function
int main()
{
    printNumbers();
    return 0;
}
```

```
// C program to check if a number is
// even or not using goto statement
#include <stdio.h>

// function to check even or not
void checkEvenOrNot(int num)
{
    if (num % 2 == 0)
        // jump to even
        goto even;
    else
        // jump to odd
        goto odd;

even:
    printf("%d is even", num);
    // return if even
    return;
odd:
    printf("%d is odd", num);
}

int main()
{
    int num = 26;
    checkEvenOrNot(num);
    return 0;
}
```

```
switch (expression)
{
    case constant1:
      // statements
      break;

    case constant2:
      // statements
      break;
    .
    .
    .
    default:
      // default statements
}
```

How does the switch statement work?
The `expression` is evaluated once and compared with the values of each `case` label.

```
// Program to create a simple calculator
#include <stdio.h>

int main() {
   char operation;
   double n1, n2;

   printf("Enter an operator (+, -, *, /): ");
   scanf("%c", &operation);
   printf("Enter two operands: ");
   scanf("%lf %lf",&n1, &n2);

   switch(operation)
   {
      case '+':
         printf("%.1lf + %.1lf = %.1lf",n1, n2, n1+n2);
         break;

      case '-':
         printf("%.1lf - %.1lf = %.1lf",n1, n2, n1-n2);
         break;

      case '*':
         printf("%.1lf * %.1lf = %.1lf",n1, n2, n1*n2);
         break;
      case '/':
         printf("%.1lf / %.1lf = %.1lf",n1, n2, n1/n2);
         break;
      // operator doesn't match any case constant +, -, *, /
      default:
         printf("Error! operator is not correct");
   }
   return 0;
}
```

If there is a match, the corresponding statements after the matching label are executed. For example, if the value of the expression is equal to `constant2`, statements after `case constant2:` are executed until `break` is encountered.
If there is no match, the default statements are executed.
Notes:
If we do not use the `break` statement, all statements after the matching label are also executed.
The `default` clause inside the `switch` statement is optional.

```
// C program to print the day using switch
#include <stdio.h>
// Driver Code
int main()
{
	int n,i;
	char ch;
	printf("The day  day=");
	scanf("%d",&n);
	for(i=0;i<2*n;i+=1)
	{
	scanf("%c",&ch);
	switch (ch) {
	case 'M':
		printf("Monday");
		break;
	case 'T':
		printf("Tuesday");
		break;
	case 'W':
		printf("Wednesday");
		break;
	case 'H':
		printf("Thursday");
		break;
	case 'F':
		printf("Friday");
		break;
	case 'S':
		printf("Satursday");
		break;
	case 'N':
		printf("Sunday");
		break;
	//default:
		//printf("Invalid Input");
	}
}
	return 0;
}
```

```
// C program to print the day using switch
#include <stdio.h>
// Driver Code
int main()
{
	int day;
	printf("The day number day=");
	scanf("%d",&day);
	printf("The day is ", day);
	switch (day) {
	case 1:
		printf("Monday");
		break;
	case 2:
		printf("Tuesday");
		break;
	case 3:
		printf("Wednesday");
		break;
	case 4:
		printf("Thursday");
		break;
	case 5:
		printf("Thursday");
		break;
	case 6:
		printf("Thursday");
		break;
	case 7:
		printf("Thursday");
		break;
	default:
		printf("Invalid Input");
		break;
	}
	return 0;
}
```

Nested loop

```
for(initialization; condition; increment/decrement)
{
   // statement of outer loop

   for(initialization; condition; increment/decrement)
   {
        // statement of inner loop
   }

   // statement of outer loop
}
```

For loop

```
#include <stdio.h>
int main()
{
   int i,j;
   for(i=1;i<=3;i++)
   {
      printf("\nOuter Loop : i = %d\n",i); /* Outer loop */

      for(j=1;j<=4;j++)/* Inner loop */
      {
         printf("Inner Loop : i = %d, j = %d\n",i,j);
      }
   }

   return 0;
```

```
#include <stdio.h>
int main()
{
   int i,j;


   for(i=1;i<=4;i++)
   {
      for(j=1;j<=5;j++)
      {
         printf("%d ",j);
      }
      printf("\n");
   }

   return 0;
}
```

```
//pattern scanf
#include<stdio.h>
int main(){
int i,j,t;
printf("Enter a number: ");
scanf("%d",&t);
for(i=1;i<=t;i++){
for(j=1;j<=i;j++)
{

printf("%d",j);   // (“*”)
}

printf("\n");
}
return 0;
}

```

Print English alphabets

```
#include <stdio.h>
int main() {
  char c;

  for (c = 'A'; c <='Z'; c++)
  {
    printf("%c\n",c);
  }
  return 0;
}

```

```
//c program to display all alphabets from a-z using ASCII VALUE//

#include <stdio.h>
int main() {

  int i;

  //ASCII value of a=97//
  for (i=97; i <=122; i++)
  {
    printf("%c\n",i);
  }
  return 0;
}
```

```
#include<stdio.h>
int main(){
    int i,j,num2;
    printf("enter i=");
    scanf("%d",&i);

    printf("enter num2=");
    scanf("%d",&num2);
    for(i;i<=num2;i++){
        for(j=1;j<=i;j++){
            printf("%d",j);
        }
            printf("\n");
    }

    return 0;
}

```

While loop

```
#include <stdio.h>
int main()
{
   int i=1,j;
   while (i <= 5)
   {
      j=1;
      while (j <= i )
      {
         printf("%d ",j);
         j++;
      }
      printf("\n");
      i++;
   }
   return 0;
}
```

Do while loop

```
#include <stdio.h>
int main()
{
   int i=1,j;
   do
   {
      j=1;
      do
      {
         printf("*");
         j++;
      }while(j <= i);
      i++;
      printf("\n");
   }while(i <= 5);
   return 0;
}
```

Odd/even no

```
#include <stdio.h>

int main() {
  int i,num,even=0,odd=0;

  printf("enter no=");
  scanf("%d",&num);
  printf("\n even no");
  while(i<=num)
{
        if(i%2==0)
        {
                printf("\n%d",i);
                even++;
        }
        i++;

}
  printf("\n odd no=");
  i=1;

  while(i<=num)
  {
        if(i%2==1)
        {
                printf("\n%d",i);
                odd++;
          }
          i++;
  }
  printf("\n total even no:%d",even);
   printf("\n total even no:%d",odd);

  return 0;
}
```

ARRAY

An array is a variable that can store multiple values. For example, if you want to store 100 integers, you can create an array for it.

```
int data[100];
```

How to declare an array?

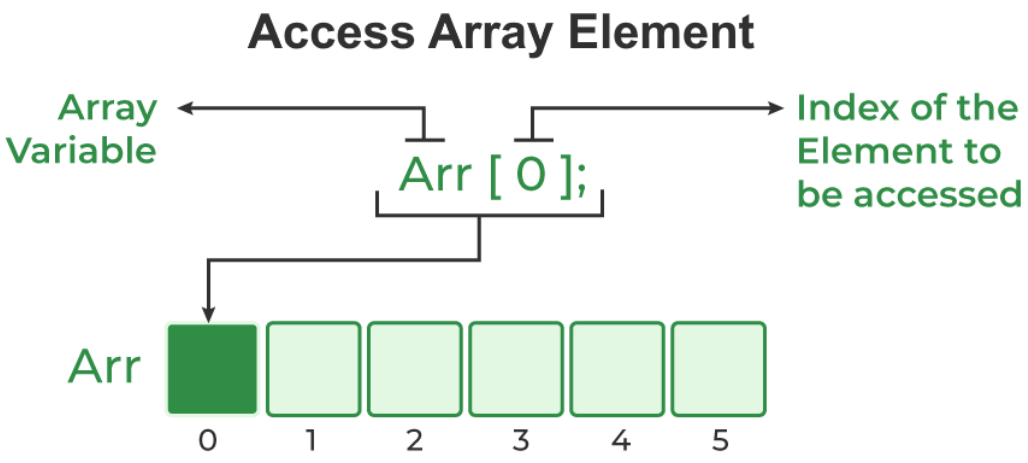
```
dataType arrayName[arraySize];
```

For example,

```
float mark[5];
```

Here, we declared an array, `mark`, of floating-point type. And its size is 5. Meaning, it can hold 5 floating-point values.

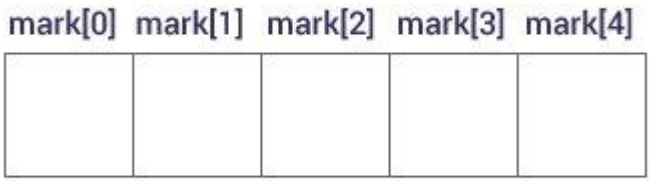
It's important to note that the size and type of an array cannot be changed once it is declared.



Access Array Elements

You can access elements of an array by indices.

Suppose you declared an array `mark` as above. The first element is `mark[0]`, the second element is `mark[1]` and so on.



Declare an Array

Few keynotes:

- Arrays have 0 as the first index, not 1. In this example, `mark[0]` is the first element.
- If the size of an array is `n`, to access the last element, the `n-1` index is used. In this example, `mark[4]`
- Suppose the starting address of `mark[0]` is **2120d**. Then, the address of the `mark[1]` will be **2124d**. Similarly, the address of `mark[2]` will be **2128d** and so on.
  This is because the size of a `float` is 4 bytes.

How to initialize an array?

It is possible to initialize an array during declaration. For example,

```
int mark[5] = {19, 10, 8, 17, 9};
```

You can also initialize an array like this.

```
int mark[] = {19, 10, 8, 17, 9};
```

Here, we haven't specified the size. However, the compiler knows its size is 5 as we are initializing it with 5 elements.

Initialize an Array

```
mark[0] is equal to 19
mark[1] is equal to 10
mark[2] is equal to 8
mark[3] is equal to 17
mark[4] is equal to 9
```

```
#include <stdio.h>
int main()
{
    int num[5],i;
    for(i=0;i<5;i++)
    {
        printf("Enter %d Number : ",(i+1));
        scanf("%d",&num[i]);
    }
    printf("Array Elements : ");
    for(i=0;i<5;i++)
    {
        printf("%d ",num[i]);
    }

    return 0;
}
```

```
#include <stdio.h>
int main()
{
    int num[5],i;

    for(i=0;i<5;i++)
    {
        //printf("Enter %d Number : ",(i+1));
        printf("enter no=");
        scanf("%d",&num[i]);
    }

    printf("Array Elements : ");
    for(i=0;i<5;i++)
    {
        printf("%d ",num[i]);
    }

    return 0;
}
```

```
// Program to take 5 values from the user and store them in an array
// Print the elements stored in the array
#include <stdio.h>
int main() {

  int i,values[5];

  printf("Enter 5 integers: ");

  // taking input and storing it in an array
  for( i = 0; i < 5; ++i) {
     scanf("%d", &values[i]);
  }

  printf("Displaying integers: ");

  // printing elements of an array
  for( i = 0; i < 5; ++i) {
     printf("%d\n", values[i]);
  }
  return 0;
}
```

// Program to find the average of n numbers using arrays

```
#include <stdio.h>

int main() {

  int marks[10], i, n, sum = 0;
  double average;

  printf("Enter number of elements: ");
  scanf("%d", &n);

  for(i=0; i < n; ++i) {
    printf("Enter number%d: ",i+1);
    scanf("%d", &marks[i]);

    // adding integers entered by the user to the sum variable
    sum += marks[i];
  }

  // explicitly convert sum to double
  // then calculate average
  average = (double) sum / n;

  printf("Average = %.2lf", average);

  return 0;
}
```

```
#include <stdio.h>
int main()
{
    int a[2],b[2],c[2],i;
    for(i=0;i<2;i++)
    {
        printf("Enter Number : a= ");
        scanf("%d",&a[i]);
         printf("Enter Number : b= ");
        scanf("%d",&b[i]);

        c[i]=a[i]+b[i];

    }

    for(i=0;i<2;i++)
    {
        printf(" total=%d \n",c[i]);
    }

    return 0;
}
```

```
//array of function
#include<stdio.h>
void sumn();/*function declartion*/
int main()
{
        sumn();/*function call*/
        return 0;
}
void sumn()/*function declartion*/
{

    int a[2],b[2],c[2],i;
    for(i=0;i<2;i++)
    {
        printf("Enter Number : a= ");
        scanf("%d",&a[i]);
         printf("Enter Number : b= ");
        scanf("%d",&b[i]);

        c[i]=a[i]+b[i];

    }

    for(i=0;i<2;i++)
    {
        printf(" total=%d \n",c[i]);
    }

}
```

Reverse order use a program

```
#include <stdio.h>
int main()
{
   int num[5],i;
   for(i=0;i<5;i++) //use this operator ++i order increase operator
   {
      printf("Enter %d Number : ",(i+1));
      printf("enter no=");
      scanf("%d",&num[i]);
   }
   printf("Array Elements : ");
   for(i=4;i>=0;i--)//use this operator --i reverse order decrease operator
   {
      printf("%d ",num[i]);
   }
   return 0;
}
```

```
Enter 1 Number : enter no=44
Enter 2 Number : enter no=55
Enter 3 Number : enter no=11
Enter 4 Number : enter no=33
Enter 5 Number : enter no=22
Array Elements : 22 33 11 55 44
--------------------------------
```

Result in aray

```
#include <stdio.h>
int main()
{
      char x[2];
      int i; // I is using run in loop//
    float pun[2],hindi[2],math[2],eng[2],total[2],per[2];

    for(i=0;i<2;i++)
    {
        //printf("Enter %d Number : ",(i+1));
        printf("enter name=");
        scanf("%s",&x[i]);
        printf("enter number pun=");
        scanf("%f",&pun[i]);
        printf("enter number hindi=");
        scanf("%f",&hindi[i]);
        printf("enter number math=");
        scanf("%f",&math[i]);
        printf("enter number eng=");
        scanf("%f",&eng[i]);
        printf("\n-----------**********----------\n");
        total[i]=pun[i]+hindi[i]+math[i]+eng[i];
       per[i]=total[i]/400*100;
    }
    for(i=0;i<2;i++)
    {
        //printf("%d ",num[i]);
         printf(" \n total is %f\n ",total[i]);
           printf(" \n per is %f\n ", per[i]);
    }
    return 0;
}
```

```
enter name=nisha
enter number pun=56
enter number hindi=77
enter number math=88
enter number eng=99

------------**************-----------
enter name=jot
enter number pun=67
enter number hindi=88
enter number math=99
enter number eng=55

------------**************-----------

 total is 320.000000

 per is 80.000000

 total is 309.000000

 per is 77.250000
```

1-d-One Dimensional Array in C

The One-dimensional arrays, also known as 1-D arrays in C are those arrays that have only one dimension.

Syntax of 1D Array in C

```
array_name [size];
```

```
#include<stdio.h>
int main()
{
    int arr[5], i;

    for(i = 0; i < 5; i++)
    {
        printf("Enter a[%d]: ", i);
        scanf("%d", &arr[i]);
    }
    printf("\nPrinting elements of the array: \n\n");
    for(i = 0; i < 5; i++)
    {
        printf("%d ", arr[i]);
    }
    // signal to operating system program ran fine
    return 0;
}
```

```
#include <stdio.h>
int main()
{
   int num[5],i;

   for(i=0;i<5;i++)
   {
      printf("Enter %d Number : ",(i+1));
      scanf("%d",&num[i]);
   }

   printf("Array Elements : ");
   for(i=0;i<5;i++)
   {
      printf("%d ",num[i]);
   }

   return 0;
}
```

Two-Dimensional Array In C

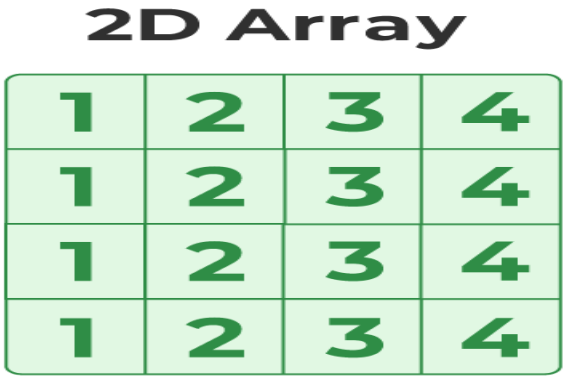
A Two-Dimensional array or 2D array in C is an array that has exactly two dimensions. They can be visualized in the form of rows and columns organized in a two-dimensional plane.

Syntax of 2D Array in C

```
array_name[size1] [size2];
```

Here,

- **size1:** Size of the first dimension.
- **size2:** Size of the second dimension.



```
#include <stdio.h>
int main()
{
    int i,j,arr[3][5];

    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("Enter %d row %d column element : ",i,j);
            scanf("%d",&arr[i][j]);
        }
    }

    printf("\n2D Array Elements :\n");
    for(i=0;i<3;i++)
    {
        for(j=0;j<5;j++)
        {
            printf("%d ",arr[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

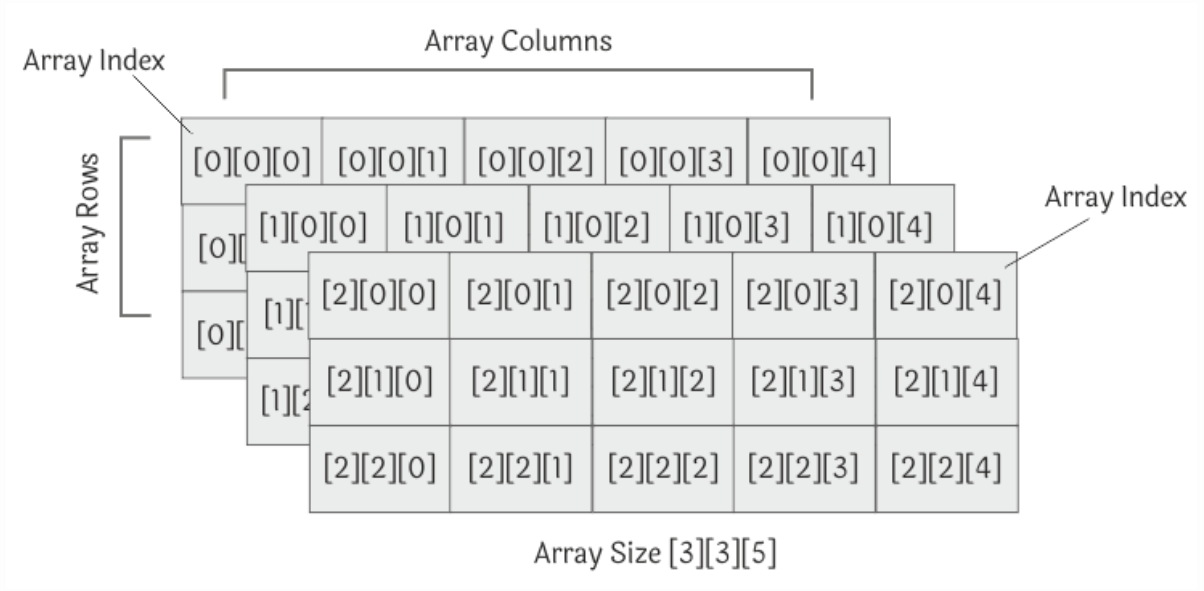
Three-Dimensional Array in C

Another popular form of a multi-dimensional array is Three Dimensional Array or 3D Array. A 3D array has exactly three dimensions. It can be visualized as a collection of 2D arrays stacked on top of each other to create the third dimension.

Syntax of 3D Array in C

```
array_name [size1] [size2] [size3];
```

```
data_type array_name[tables][rows][columns];
```



```
#include <stdio.h>
int main()
{
   int t,i,j,arr[2][3][5];

   for(t=0;t<2;t++)
   {
      for(i=0;i<3;i++)
      {
         for(j=0;j<5;j++)
         {
            printf("Enter %d table %d row %d column element : ",(t+1),i,j);
            scanf("%d",&arr[t][i][j]);
         }
      }
   }

   for(t=0;t<2;t++)
   {
      printf("\n%d Table Array Elements :\n",(t+1));
      for(i=0;i<3;i++)
      {
         for(j=0;j<5;j++)
         {
            printf("%d ",arr[t][i][j]);
         }
         printf("\n");
      }
   }

   return 0;
}
```

Structure

Structures (also called structs) are a way to group several related variables into one place. Each variable in the structure is known as a member of the structure.

Unlike an array, a structure can contain many different data types (int, float, char, etc.).

Create a Structure

You can create a structure by using the `struct` keyword and declare each of its members inside curly braces:

```
struct MyStructure {   // Structure declaration
  int myNum;           // Member (int variable)
  char myLetter;       // Member (char variable)
}; // End the structure with a semicolon
```

To access the structure, you must create a variable of it.

Use the `struct` keyword inside the `main()` method, followed by the name of the structure and then the name of the structure variable:

Create a struct variable with the name "s1":

```
struct myStructure {
  int myNum;
  char myLetter;
};

int main() {
  struct myStructure s1;
  return 0;
}
```

Access Structure Members

To access members of a structure, use the dot syntax (`.`):

Store record of student

```
#include <stdio.h>
struct students
{
    int roll;
    char name[20];
    float cgpa;
};

int main()
{
    struct students stu1; //stu1 variable name

    printf("Enter Student Roll No. : ");
    scanf("%d",&stu1.roll);

    printf("Enter Student Name : ");
    scanf("%s",stu1.name);

    printf("Enter Student CGPA : ");
    scanf("%f",&stu1.cgpa);

    printf("\nStudent Roll No. : %d",stu1.roll);
    printf("\nStudent Name : %s",stu1.name);
    printf("\nStudent CGPA : %f",stu1.cgpa);

    return 0;
}
```

//Array of use in struct

```
#include <stdio.h>
struct students
{
    int roll;
    char name[20];
    float cgpa;
};

int main()
{
    struct students stu[3];
    int i;

    for(i=0;i<3;i++)
    {
        printf("\nEnter Student Roll No. : ");
        scanf("%d",&stu[i].roll);

        printf("Enter Student Name : ");
        scanf("%s",stu[i].name);

        printf("Enter Student CGPA : ");
        scanf("%f",&stu[i].cgpa);
    }

    for(i=0;i<3;i++)
    {
        printf("\nStudent Roll No. : %d, Name : %s,
CGPA : %f",stu[i].roll,stu[i].name,stu[i].cgpa);
    }

    return 0;
}
```

```
#include<stdio.h>
struct student
{
	char name[50];
	int m1,m2,m3,m4,m5,sum;
	float avg;
};
int main()
{
	int n,i;
	scanf("%d",&n);
	struct student s[n];
	for(i=0;i<n;i++)//n is use this  no student data fill input number 4//
	{
		printf("enter name=");
		scanf("%s",s[i].name);
		printf("enter name= m1,m2,m3,m4,m5");
		scanf("%d %d %d %d
%d",&s[i].m1,&s[i].m2,&s[i].m3,&s[i].m4,&s[i].m5);
		s[i].sum=s[i].m1+s[i].m2+s[i].m3+s[i].m4+s[i].m5;
		s[i].avg=s[i].sum/5;
	}
	for(i=0;i<n;i++)
	{
		printf("Name: %s\nSum:
%d\nAverage:%f\n",s[i].name,s[i].sum,s[i].avg);
	}
	return 0;

}
```

union

```
#include<stdio.h>
Union  student
{
        char name[50];
        int m1,m2,m3,m4,m5,sum;
        float avg;
};
int main()
{
        int n,i;
        scanf("%d",&n);
        struct student s[n];
        for(i=0;i<n;i++)
        {
                printf("enter name=");
                scanf("%s",s[i].name);
                printf("enter name= m1,m2,m3,m4,m5");
                scanf("%d %d %d %d
%d",&s[i].m1,&s[i].m2,&s[i].m3,&s[i].m4,&s[i].m5);
                s[i].sum=s[i].m1+s[i].m2+s[i].m3+s[i].m4+s[i].m5;
                s[i].avg=s[i].sum/5;
        }
        for(i=0;i<n;i++)
        {
                printf("Name: %s\nSum:
%d\nAverage:%f\n",s[i].name,s[i].sum,s[i].avg);
        }
        return 0;

}
```

FUNCTION

A function is a block of code that performs a specific task.

Suppose, you need to create a program to create a circle and color it. You can create two functions to solve this problem:

- create a circle function

```
//no parameter and no return value
#include<stdio.h>
void sumn();/*function declartion*/
int main()
{
    sumn();/*function call*/
    return 0;
}
void sumn()/* no parameter ,no return value*/
{

    int a,b,res;
    printf("enter value of a=");
    scanf("%d",&a);
    printf("enter value of a=");
    scanf("%d",&b);
    res=a+b;
    printf("%d+%d=%d",a,b,res);
```

```
}
```

- create a color function

Dividing a complex problem into smaller chunks makes our program easy to understand and reuse.

```
#include<stdio.h>

void sumn();/*function declartion*/

int main()
{

        sumn();/*function call*/

        return 0;
}

void sumn()/* no parameter ,no return value*/

{

        int a,b,res;
        printf("enter value of  two numbers to add=");
        scanf("%d%d",&a,&b);
        res=a+b;
        printf("%d+%d=%d",a,b,res);
}
```

```c
#include <stdio.h>
int addNumbers(int a, int b);          // function prototype

int main()
{
    int n1,n2,sum;

    printf("Enters two numbers: ");
    scanf("%d %d",&n1,&n2);

    sum = addNumbers(n1, n2);         // function call
    printf("sum = %d",sum);

    return 0;
}

int addNumbers(int a, int b)          // function definition
{
    int result;
    result = a+b;
    return result;                   // return statement
}
```

//with parameter and no return value

```
#include<stdio.h>

void sumn(int,int);/*function declartion*/

int main()
{

int a,b;
printf("entter value of a and  b :");
scanf("%d%d",&a,&b);

        sumn(a,b);
        return 0;
}

void sumn(int x,int y)/* two parameter ,no return value*/

{

        int res;
        res=x+y;
        printf("%d+%d=%d",x,y,res);
}

```

//with parameter and return value

```
#include<stdio.h>

int sumn(int,int);/*function declartion*/

int main()
{

int a,b,c;
printf("entter value of a and  b :");
scanf("%d%d",&a,&b);

      c = sumn(a,b);/*function call*/
      printf("%d+%d=%d",a,b,c);
      return 0;
}

int sumn(int x,int y)/* value passes of type int */

{

      int total;
      total=x+y;
      return(total);/* data type of value returned same as return type*/
}
```

//with no parameter but return value

```
#include <stdio.h>
int main()
{
    int ans;

    ans=sum(); /*function call-no value passes*/

    printf("Total : %d",ans);

    return 0;
}


int sum()
{
    int num1,num2,sum;

    printf("Enter First Number : ");
    scanf("%d",&num1);
    printf("Enter Second Number : ");
    scanf("%d",&num2);

    sum=num1+num2;

    return sum;
}
```

```
//function call with array //
#include<stdio.h>
void sumn();/*function declartion*/
int main()
{
      sumn();/*function call*/
      return 0;
}
void sumn()/*function declartion*/
{

   int a[2],b[2],c[2],i;
   for(i=0;i<2;i++)
   {
       printf("Enter Number : a= ");
       scanf("%d",&a[i]);
        printf("Enter Number : b= ");
       scanf("%d",&b[i]);

       c[i]=a[i]+b[i];

   }

   for(i=0;i<2;i++)
   {
       printf(" total=%d \n",c[i]);
   }
}
```

Recursion in C

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.

```
#include<stdio.h>
void odd();
void even();
int n=1;

void odd()
{
    if(n <= 10)
    {
        printf("%d ", n+1);
        n++;
        even();
    }
    return;
}

void even()
{
    if(n <= 10)
    {
        printf("%d ", n-1);
        n++;
        odd();
    }
    return;
}

int main()
{
    odd();
}
```

In this C program, we have functions named odd() and even(). A variable n is assigned with a value 1 as we have to take values from 1 to 10.

Now inside the odd() function, we have an if statement which states that if the value of n is less than or equals 10 add 1 to it and print. Then the value of n is incremented by 1(it becomes even), and the even() function is called.

Now inside the even() function, we again have an if statement which states that if the value of n is less than or equals 10 subtract 1 from it and print.

Then the value of n is incremented by 1(it becomes odd, and the odd() function is called. This indirect recursion goes on until the if condition inside both the functions becomes unsatisfied. At last, we have the main() function inside, which we call the odd() function as the first number handle is 1, which is odd.

Now let's simulate this program using **stack** and the concept called **activation record** by which we could track program logic in respect of program stack.

In the following images:

- Act means "Activation Record"
- o means **odd()**
- e means **even()**
- m means **main()**

```
#include <stdio.h>
void myprint(int num);
int main()
{
    int n;

    printf("Enter a Number : ");
    scanf("%d",&n);

    myprint(n);

    return 0;
}

void myprint(int num)
{
    printf("%d ",num);

    if(num==1)
    {
        return;
    }
    else
    {
        myprint(num-1);
    }
}
```

जैसा की आप पहले पढ़ चुके हैं की main function को system द्वारा अपने आप call किया जाता है यानी stack memory में सबसे पहले main function को space allocate किया जाता है.

Step 1: main function में से हमने myprint function को call किया है. myprint function call होते है उसे भी stack memory में space allocate हो जाएगा.

function calling के समय हमने agrument के तौर पर value 5 को pass किया है जो myprint function के variable num में store हो जाएगी.

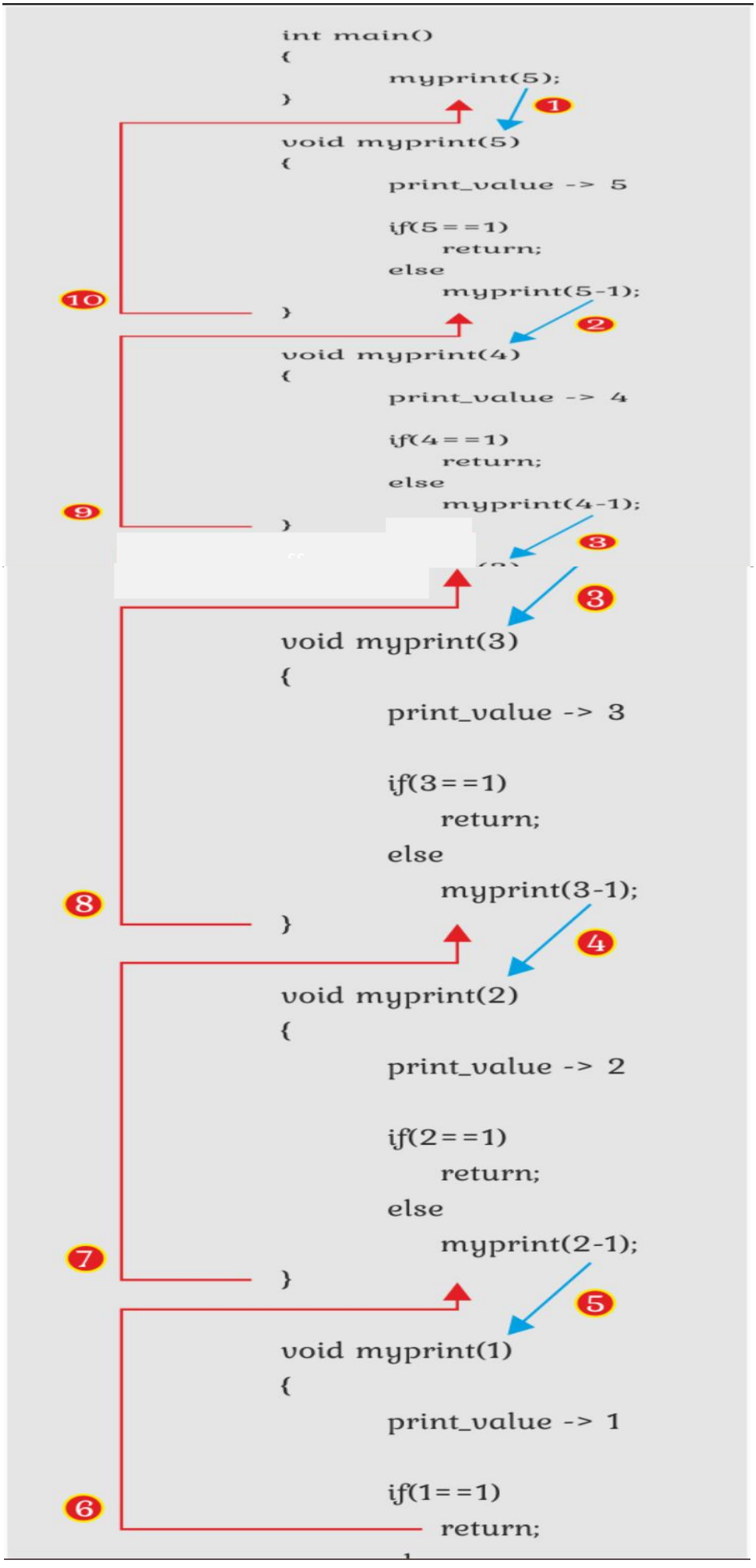
इसके बाद हमने variable num को print कराया है और उसके बाद if statement की condition (5==1) false हो जाएगी जिसकी वजह से program का control else में चला जाएगा.

जैसा की आप पहले पढ़ चुके हैं की main function को system द्वारा अपने आप call किया जाता है यानी stack memory में सबसे पहले main function को space allocate किया जाता है.

Step 1: main function में से हमने myprint function को call किया है. myprint function call होते है उसे भी stack memory में space allocate हो जाएगा.

function calling के समय हमने agrument के तौर पर value 5 को pass किया है जो myprint function के variable num में store हो जाएगी.

इसके बाद हमने variable num को print कराया है और उसके बाद if statement की condition (5==1) false हो जाएगी जिसकी वजह से program का control else में चला जाएगा.


int main()
{
myprint(5);
}
1
void myprint(5)
{
print_value -> 5
if(5==1)
return;
else
myprint(5-1);
10
}
2
void myprint(4)
{
print_value -> 4
if(4==1)
return;
else
myprint(4-1);
9
}
3
3
void myprint(3)
{
print_value -> 3
if(3==1)
return;
else
myprint(3-1);
8
}
4
void myprint(2)
{
print_value -> 2
if(2==1)
return;
else
myprint(2-1);
7
}
5
void myprint(1)
{
print_value -> 1
if(1==1)
6
return;

Pointer

What is a Pointer in C?

A pointer is defined as a derived data type that can store the address of other C variables or a memory location. We can access and manipulate the data stored in that memory location using pointers.

As the pointers store the memory addresses, their size is independent of the type of data they are pointing to. This size of pointers only depends on the system architecture.

Syntax

The syntax of pointers is similar to the variable declaration in C, but we use the **(*) dereferencing operator** in the pointer declaration.

```
datatype * ptr;
```

where

- **ptr** is the name of the pointer.
- **datatype** is the type of data it is pointing to.

The above syntax is used to define a pointer to a variable. We can also define pointers to functions, structures, etc.

How to Use Pointers?

The use of pointers can be divided into three steps:

1. **Pointer Declaration**
2. **Pointer Initialization**
3. **Dereferencing**

1. Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the **(*) dereference operator** before its name.

Example

```
int *ptr;
```

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

2. Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the **(&) addressof operator** to get the memory address of a variable and then store it in the pointer variable.

Example

```
int var = 10;
int * ptr;
ptr = &var;
```

We can also declare and initialize the pointer in a single step. This method is called **pointer definition** as the pointer is declared and initialized at the same time.

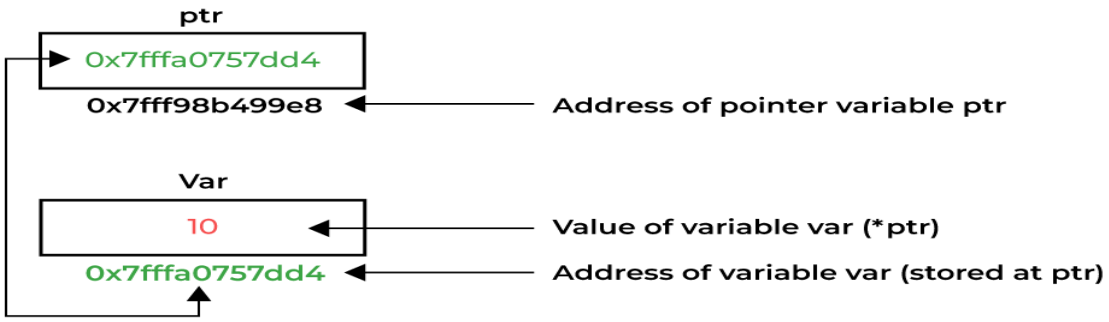
Example

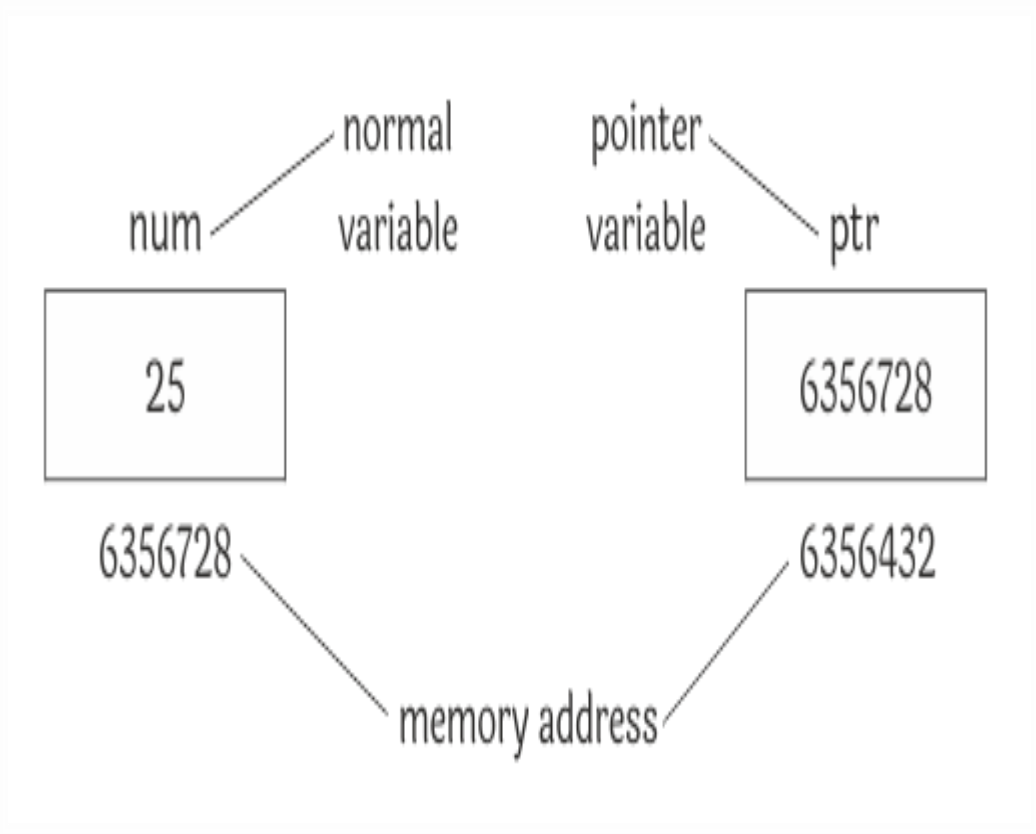
```
int *ptr = &var;
```

Note: *It is recommended that the pointers should always be initialized to some value before starting using it. Otherwise, it may lead to number of errors.*

3. Dereferencing

Dereferencing a pointer is the process of accessing the value stored in the memory address specified in the pointer. We use the same **(*) dereferencing operator** that we used in the pointer declaration.





```
#include <stdio.h>
int main()
{
   int num, *ptr;

   num=25;

   ptr = &num;

   printf("Value of Num : %d\n",num);
   printf("Address of Num : %d\n",&num);

   printf("Value of Ptr : %d\n",ptr);
   printf("Address of Ptr : %d\n",&ptr);
   return 0;
}
```

```
// C program to illustrate Pointers
#include <stdio.h>

void nn()
{
        int var = 10;

        // declare pointer variable
        int* ptr;

        // note that data type of ptr and var must be same
        ptr = &var;

        // assign the address of a variable to a pointer
        printf("Value at ptr = %p \n", ptr);
        printf("Value at var = %d \n", var);
        printf("Value at *ptr = %d \n", *ptr);
}

// Driver program
int main()
{
        nn();
        return 0;
}
```

```
#include <stdio.h>
int main()
{
   int num, *ptr;

   num=20;

   ptr = &num;

   printf("Using Num Variable\n");
   printf("Value of Num : %d\n",num);
   printf("Address of Num : %d\n",&num);

   printf("\n\nUsing Pointer \n");
   printf("Value of Num : %d\n",*ptr);
   printf("Address of Num : %d\n",ptr);

   return 0;
}
```

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.

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

```
#include <stdio.h>
int main()
{
   char str[20];

   printf("Enter Your Name : ");
   scanf("%s",str);

   printf("Your Name : %s",str);

   return 0;
}
```

C gets() and puts() functions

The gets() and puts() are declared in the header file stdio.h. Both the functions are involved in the input/output operations of the strings.

C gets() function

The gets() function enables the user to enter some characters followed by the enter key. All the characters entered by the user get stored in a character array. The null character is added to the array to make it a string. The gets() allows the user to enter the space-separated strings. It returns the string entered by the user.

Reading string using gets()
#include<stdio.h>
int main ()
{
char s[30];
printf("Enter the string? ");
gets(s);
printf("You entered %s",s);
return 0;
}

The gets() function is risky to use since it doesn't perform any array bound checking and keep reading the characters until the new line (enter) is encountered. It suffers from buffer overflow, which can be avoided by using fgets(). The fgets() makes sure that not more than the maximum limit of characters are read. Consider the following example.

```
#include<stdio.h>
int  main()
{
   char str[20];
   printf("Enter the string? ");
   fgets(str, 20, stdin);
   printf("%s", str);

   return 0;
}
```

C puts() function

The puts() function is very much similar to printf() function. The puts() function is used to print the string on the console which is previously read by using gets() or scanf() function. The puts() function returns an integer value representing the number of characters being printed on the console. Since, it prints an additional newline character with the string, which moves the cursor to the new line on the console, the integer value returned by puts() will always be equal to the number of characters present in the string plus 1.

Declaration

1. `int puts(char[])`

Let's see an example to read a string using gets() and print it on the console using puts().

```
#include<stdio.h>
#include <string.h>
int main(){
char name[50];
printf("Enter your name: ");
gets(name); //reads string from user
printf("Your name is: ");
puts(name);  //displays string
return 0;
}
```

String input output using gets() and puts() functions

जब आप users से multi-word strings input लेना चाहते हो तो उसके लिए हम gets() function का use करते हैं.

gets() function की ऐसी string input ले सकते हो जिसमें space भी मौजूद है. gets() function तब तक string input लेता है जब तक आप keyboard enter key (newline) press नहीं कर देते.

puts() function की help से आप किसी भी string को screen पर print करा सकते हो. puts() function में आप सिर्फ string variable name लिखते हैं उसके अलावा आप format specifier या कोई और text message नहीं लिखते.

मुझे puts() function कुछ खास पसंद नहीं इसलिए मैं हमेशा printf() function का ही use करता हूँ लेकिन सिर्फ आपको समझाने के लिए मैंने नीचे example में puts function का use किया है.

```
#include <stdio.h>
int main()
{
   char str[20];

   printf("Enter Your Name : ");
   gets(str);

   printf("Your Name : ");
   puts(str);

   return 0;
}
```

C String Functions

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

No.	Function	Description
1)	strlen(string_name)	returns the length of string name.
2)	strcpy(destination, source)	copies the contents of source string to destination string.
3)	strcat(first_string, second_string)	concats or joins first string with second string. The result of the string is stored in first string.
4)	strcmp(first_string, second_string)	compares the first string with second string. If both strings are same, it returns 0.
5)	strrev(string)	returns reverse string.
6)	strlwr(string)	returns string characters in lowercase.
7)	strupr(string)	returns string characters in uppercase.

C String Length: strlen() function

The strlen() function returns the length of the given string. It doesn't count null character '\0'.

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


int main() {
    // Write C code here

    int length;
    char n[5];
    printf("\nenter name=");
    scanf("%s",&n);

    length=strlen(n);
   printf("\n lenth of string %d",length);



    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[20];
   int len;

   printf("Enter Your String : ");
   gets(str);

   len=strlen(str);

   printf("String Length : %d",len);

   return 0;
}
```

C Copy String: strcpy()

The strcpy(destination, source) function copies the source string in destination.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[20],dstr[20];

   printf("Enter Your String : ");
   gets(str);

   strcpy(dstr,str);

   printf("Str value : %s",str);
   printf("\nDStr value : %s",dstr);

   return 0;
}
```

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

main()
{
    char source[] = "C Program";
    char destination[50];
    strcpy(destination, source);
    printf("Source string: %s\n", source);
    printf("Destination string: %s\n", destination);
    return 0;
}
```

C String Concatenation: strcat()

The strcat(first_string, second_string) function concatenates two strings and result is returned to first_string.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   printf("\nStr1 value Before Concatenate : %s",str1);
   printf("\nStr2 value Before Concatenate : %s",str2);

   strcat(str1,str2);

   printf("\n\nStr1 value After Concatenate : %s",str1);
   printf("\nStr2 value After Concatenate : %s",str2);

   return 0;
}
```

C Compare String: strcmp()

The strcmp(first_string, second_string) function compares two string and returns 0 if both strings are equal.

Here, we are using *gets()* function which reads string from the console.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   if((strcmp(str1,str2))==0)
   {
      printf("str1 and str2 are equal");
   }
   else
   {
      printf("str1 and str2 are not equal");
   }

   return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   if((strcmpi(str1,str2))==0)
   {
      printf("str1 and str2 are equal");
   }
   else
   {
      printf("str1 and str2 are not equal");
   }

   return 0;
}
```

Strcmpi()

Explanation:

जैसा की output 3 में देख सकते हो की strcmp() function का case sensitive होने की वजह से strings 'Karan" और 'KARAN" same नहीं हैं.

अगर आप चाहते हो की दोनों strings को compare करते वक्त case sensitivity check नहीं की जाए यानी छोटा a और बड़ा A दो अलग-अलग characters नहीं माने जाए तो इसके लिए हम strcmpi() function का use करते हैं.

strcmpi() function in C Programming

strcmpi() function का use भी strcmp() function की तरह 2 string को compare करना है. बस फर्क ये की strcmpi() **case insensitive** function है.

जिसका मतलब ये होता है की इस function के लिए छोटा a और बड़ा A दो अलग-अलग characters नहीं हैं.

strcmpi() function syntax:

```
int strcmpi(string1, string2);
```

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   if((strcmpi(str1,str2))==0)
   {
      printf("str1 and str2 are equal");
   }
   else
   {
      printf("str1 and str2 are not equal");
   }

   return 0;
}
```

Output 1:

```
Enter Your First String : Karan
Enter Your Second String : Kiran
str1 and str2 are not equal
```

Output 2:

```
Enter Your First String : Karan
Enter Your Second String : Karan
str1 and str2 are equal
```

Output 3:

```
Enter Your First String : Karan
Enter Your Second String : KARAN
str1 and str2 are equal
```

Strncmp()

strncmp() function in C Programming

strncmp() function की help से हम 2 strings के शुरू के n characters को compare करते हैं अगर दोनों strings n characters same होंगे तो ये function 0 value return करता हैं.

अगर दोनों strings के n characters same नहीं होंगे तब ये function कोई भी negative या positive value return करता है.

इसके अलावा सबसे जरूरी बात strncmp() **case sensitive** function हैं. जिसका मतलब ये होता है की इस function के लिए छोटा a और बड़ा A दो अलग-अलग characters हैं.

strncmp() function syntax:

```
int strncmp(string1, string2, int n);
```

strncmp() function का standard syntax ऊपर दिए गए syntax से थोड़ा अलग होता है. मैंने यहाँ original syntax को simplify form में लिखा है जिससे की आप strncmp() function को आसानी से समझ सकें.

Argument: जब आप strncmp() function को call करते हैं तब आप उन दो strings को arguments के तौर पर pass करते हैं जिन्हें आप compare करना चाहते हो.

इन दो strings के अलावा हम एक int value (n) भी pass करते हैं जिसका use इस काम के लिए किया जाता है की उन दोनों strings के कितने characters को compare करना है.

Return Value: strncmp() function arguments के तौर पर pass की गयी दोनों strings के n characters same होंगे तो ये function 0 value return करेगा.

strncmp() function example program:

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   if((strncmp(str1,str2,3))==0)
   {
      printf("str1 and str2 are equal");
   }
   else
   {
      printf("str1 and str2 are not equal");
   }

   return 0;
}
```

```
Enter Your First String : Karan
Enter Your Second String : Kiran
str1 and str2 are not equal
```

Output 2:

```
Enter Your First String : Karan
Enter Your Second String : Karim
str1 and str2 are equal
```

Output 3:

```
Enter Your First String : Karan
Enter Your Second String : KARAN
str1 and str2 are not equal
```

Explanation:

strnicmp() function in C Programming

strnicmp() function का use भी strncmp() function की तरह 2 strings के शुरू के n characters compare करता है. बस फर्क ये की strnicmp() **case insensitive** function है.

जिसका मतलब ये होता है की इस function के लिए छोटा a और बड़ा A दो अलग-अलग characters नहीं हैं.

strnicmp() function syntax:

```
int strnicmp(string1, string2,int n);
```

strnicmp() function का standard syntax ऊपर दिए गए syntax से थोड़ा अलग होता है. मैंने यहाँ original syntax को simplify form में लिखा है जिससे की आप strnicmp() function को आसानी से समझ सकें.

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str1[20],str2[20];

   printf("Enter Your First String : ");
   gets(str1);

   printf("Enter Your Second String : ");
   gets(str2);

   if((strnicmp(str1,str2,3))==0)
   {
      printf("str1 and str2 are equal");
   }
   else
   {
      printf("str1 and str2 are not equal");
   }

   return 0;
}
```

```
}
```

Output 1:

```
Enter Your First String : Karan
Enter Your Second String : Kiran
str1 and str2 are not equal
```

Output 2:

```
Enter Your First String : Karan
Enter Your Second String : Karim
str1 and str2 are equal
```

Output 3:

```
Enter Your First String : Karan
Enter Your Second String : KARAN
str1 and str2 are equal
```

C Reverse String: strrev()

The strrev(string) function returns reverse of the given string. Let's see a simple example of strrev() function.

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


int main() {
    // Write C code here

    char str[5];
    printf("\nenter name=");
    scanf("%s",&str);

    printf("Before reverse function call : %s",str);

   strrev(str);

    printf("After reverse function call : %s",str);
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[20];
    printf("Enter Your String : ");
    gets(str);
    printf("Before Reverse : %s",str);
    strrev(str);
    printf("\nAfter Reverse : %s",str);
    return 0;
}
```

Output 1:

```
Enter Your String : ABC
Before Reverse : ABC
After Reverse : CBA
```

Output 2:

```
Enter Your String : Hello World
Before Reverse : Hello World
After Reverse : dlroW olleH
```

C String Uppercase: strupr()

The strupr(string) function returns string characters in uppercase. Let's see a simple example of strupr() function.

```
#include <stdio.h>
#include <string.h>
int main() {
    // Write C code here
    char str[5];
    printf("\nenter name=");
    scanf("%s",&str);
    printf("Before strupr function call : %s",str);
    strupr(str);
    printf("After strupr function call : %s",str);
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
   char str[20];

   printf("Enter Your String : ");
   gets(str);

   printf("Before strupr function call : %s",str);

   strupr(str);

   printf("After strupr function call : %s",str);

   return 0;
}
```

```
Enter string: javatpoint
String is: javatpoint
Upper String is: JAVATPOINT
```

C String Lowercase: strlwr()

The strlwr(string) function returns string characters in lowercase. Let's see a simple example of strlwr() function.

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


int main() {
    // Write C code here

    char str[5];
    printf("\nenter name=");
    scanf("%s",&str);

    printf("Before lowercase function call : %s\n",str);

    strlwr(str);

    printf("After lowercase  function call : %s",str);
    return 0;
}
```

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[20];

    printf("Enter Your String : ");
    gets(str);

    printf("Before strlwr function call : %s\n",str);

    strlwr(str);

    printf("After strlwr function call : %s",str);

    return 0;
}
```

Output:

```
Enter string: JAVATpoint
String is: JAVATpoint
Lower String is: javatpoint
```

//Ascii Code

```
#include <stdio.h>

int main() {
   // Write C code here


   char n;
   printf("\nenter name=");
   scanf("%c",&n);


  printf("\n asci= %d",n);


   return 0;
}
```

Additional program

```
//password create
#include<stdio.h>
int main(){

	int password,repeatpassword,conformpassword;

	printf("enter password=");
	scanf("%d",&password);

	printf("enter repeatpassword=");
	scanf("%d",&repeatpassword);

	printf("enter conformpasswor=");
	scanf("%d",&conformpassword);
if((password==repeatpassword)&&(password==conformpas
sword))
{
		printf("correct");
	}

	else{

		printf("incorrect");
	}
	return 0;
}
```

//password

```
#include<stdio.h>
int main(){

	int password,conformpassword;

	printf("enter password=");
	scanf("%d",&password);



	printf("enter conformpasswor=");
	scanf("%d",&conformpassword);
	if(password==conformpassword)
	{
		printf("correct");
	}

	else{

		printf("incorrect");

	}

	return 0;
}
```