INTRODUCTION TO C

```
1. #include<stdio.h>
2. int main() {
3. printf("Hello C Programming\n");
4. return 0;
5. }
```

C programming is a general-purpose, procedural, imperative computer programming language developed in 1972 by Dennis M. Ritchie at bell laboratories of AT&T (American Telephone & Telegraph), located in the U.S.A.

It was developed to overcome the problems of previous languages such as B, BCPL, etc.

Initially, C language was developed to be used in **UNIX operating system**. It inherits many features of previous languages such as B and BCPL.

Let's see the programming languages that were developed before C language.

Language	**Year**	**Developed by**
Algol	1960	International Group
BCPL	1967	Martin Richard
B	1970	Ken Thompson
Traditional C	1972	Dennis Ritchie

Why to Learn C Programming?

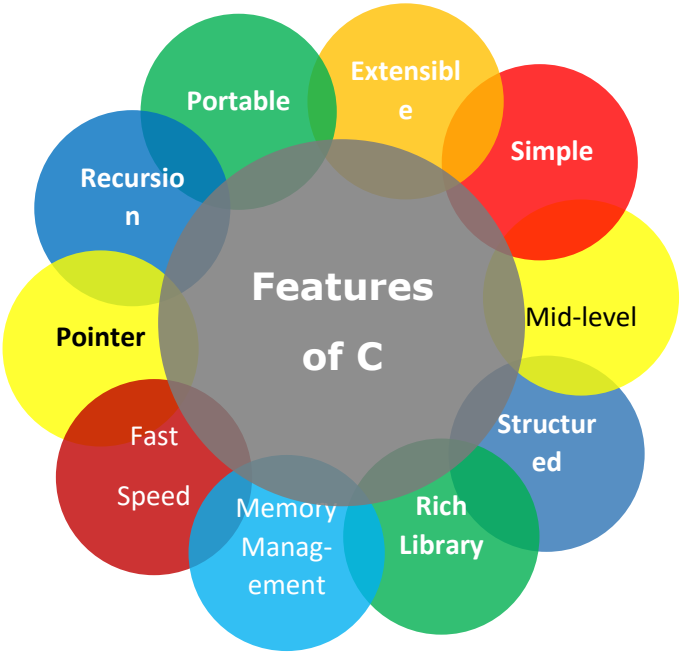
C programming language is a MUST for students and working professionals to become a great Software Engineer specially when they are working in Software Development Domain. I will list down some of the key advantages of learning C Programming:

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Features of C Language

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

- Simple
- Machine Independent or Portable
- Mid-level programming language
- structured programming language
- Rich Library
- Memory Management
- Fast Speed
- Pointers
- Recursion
- Extensible



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.

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.

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.

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.

Rich Library

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

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.

Speed

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

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.

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.

Extensible

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

First C Program

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

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

```
1. #include <stdio.h>
2. int main(){
3. printf("Hello C Language");
4. return 0;
5. }
```

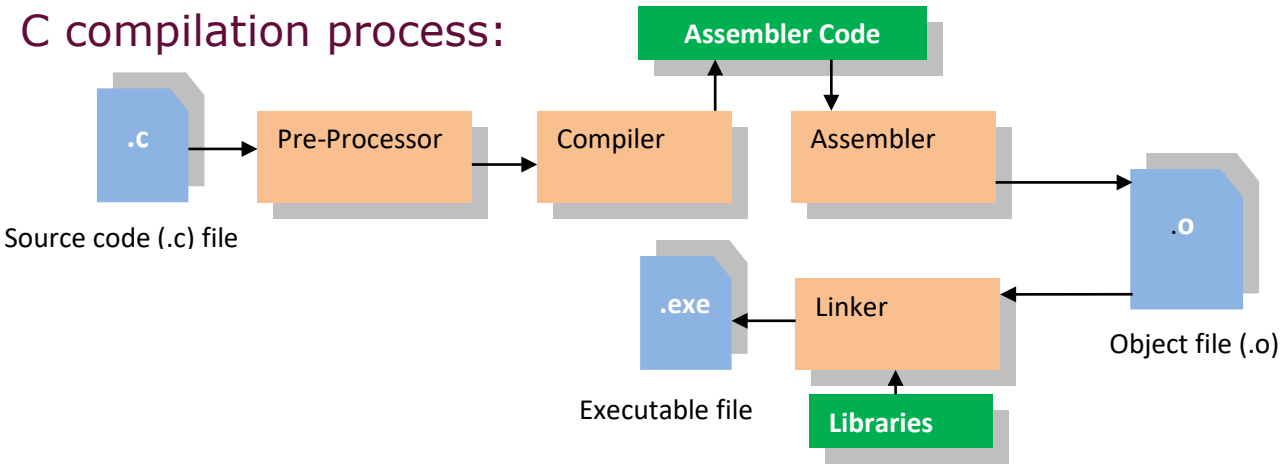
#include<stdio.h> includes the standard input output library functions. The printf() function is defined in stdio.h .

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

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

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

C compilation process:



C - Basic Syntax

You have seen the basic structure of a C program, so it will be easy to understand other basic building blocks of the C programming language.

Tokens in C

A C program consists of various tokens and a token is either a keyword, an identifier, a constant, a string literal, or a symbol. For example, the following C statement consists of five tokens –

```
printf("Hello world");
```

The individual tokens are –

```
printf

(

"Hello world"

);
```

Semicolons

In a C program, the semicolon is a statement terminator. That is, each individual statement must be ended with a semicolon. It indicates the end of one logical entity.

Given below are two different statements –

```
printf("Hello world");

return 0;
```

Keywords

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

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

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

We will learn about all the C language keywords later.

Data Types in C

A data type is a classification of data which tells the compiler or interpreter how the programer intends to use the data. In another word you can say that it defines the size (BYTE) and the range of a variable.

Classification of the data types in C language

1.Pre-define data types
2.User define data types

Pre-define data types in C:

Already define by the C standard, these are int, char, float, double, …etc. Using the sizeof operator you can get the size of (in bytes) data types. These data types are dependent on the platform so C standard also introduces the fixed size of data type like uint8_t, uint16_t uint32_t …etc. These are defined in stdint.h header file for more detail you check this header file.

User-defined data types in C:

The C language also provides flexibility to the programmer to create own data types. A user-defined data type is created by the users using the tags **struct**, **union** or **enum**.

In C language, different data types have the different ranges. The range varies from compiler to compiler. In below table, I have listed some data types with there ranges and format specifier as per the 32-bit GCC compiler.

Data Type	Memory (bytes)	Range	Format Specifier
short int	2	-32,768 to 32,767	%hd
unsigned short int	2	0 to 65,535	%hu
unsigned int	4	0 to 4,294,967,295	%u
int	4	-2,147,483,648 to 2,147,483,647	%d
long int	4	-2,147,483,648 to 2,147,483,647	%ld
unsigned long int	4	0 to 4,294,967,295	%lu
long long int	8	-(2^63) to (2^63)-1	%lld
unsigned long long int	8	0 to 18,446,744,073,709,551,615	%llu
signed char	1	-128 to 127	%c
unsigned char	1	0 to 255	%c
float	4	-	%f
double	8	-	%lf
long double	12	-	%Lf

C Operators

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

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

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

Precedence of Operators in C

The precedence of operator species that which operator will be evaluated first and next. The associability specifies the operator direction to be evaluated; it may be left to right or right to left.

Let's understand the precedence by the example given below:

Int value=10+20*10;

The value variable will contain **210** because * (multiplicative operator) is evaluated before + (additive operator).

The precedence and associativity of C operators is given below:

CATEGORY	OPERATOR	ASSOCIATIVITY
Postfix	() [] ->.++ --	Left to right
Unary	= - ! ~ ++ -- (type)* & sizeof	Right to left
Multiplicative	*%*	Left to right
Additive	+-	Left to right
Shift	<< >>	Left to right
Relational	<<= >>=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right

Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -+ *= /= %=>>=<<=&^=|=	Right to left
Comma	,	Left to right

Comments in C

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

1. Single Line Comments
2. Multi-Line Comments

Single Line Comments

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

```
1. #include<stdio.h>
2. int main(){
3.     //printing information
4.     printf("Hello C");
5. return 0;
6. }
```

Output:

Hello C

Even you can place the comment after the statement. For example:

```
printf("Hello C");//printing information
```

Mult Line Comments

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

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

Let's see an example of a multi-Line comment in C.

```
#include<stdio.h>
int main(){
    /*printing information
      Multi-Line Comment*/
    printf("Hello C");
return 0;
}
```

Output:

```
Hello C
```

C Format Specifier

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

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

Format specifier	Type
%c	Character
%d	Signed integer
%e or %E	Scientific notation of floats
%f	Float values
%g or %G	Similar as %e or %E
%hi	Signed integer (short)
%hu	Unsigned Integer (short)
%i	integer
%l or %ld or %li	Long
%lf	Double
%Lf	Long double
%lu	Unsigned int or unsigned long
%lli or %lld	Long long
%llu	Unsigned long long
%o	Octal representation
%p	Pointer
%s	String
%u	Unsigned int
%x or %X	Hexadecimal representation
%n	Prints nothing
%%	Prints % character

Let see few examples to understand format specifiers in C:

1. Format specifier (character): %c

```c
#include <stdio.h>

int main()
{
    char data = 'A';

    printf("%c\n", data);

    return 0;
}
```

Output: A

```c
#include <stdio.h>

int main()
{
    int data = 65;

    printf("%c\n", data);

    return 0;
}
```

Output: A

In both codes, you can see %c convert data in character and printf function print it on the console.

2. Format specifiers (integer): %d, %i, %u

```c
#include <stdio.h>

int main()
{
    int data = 65;

    printf("%d\n", data);
    printf("%u\n", data);
    printf("%i\n", data);

    return 0;
}
```

Output:

65
65
65

Difference between %d and %i format specifier in C

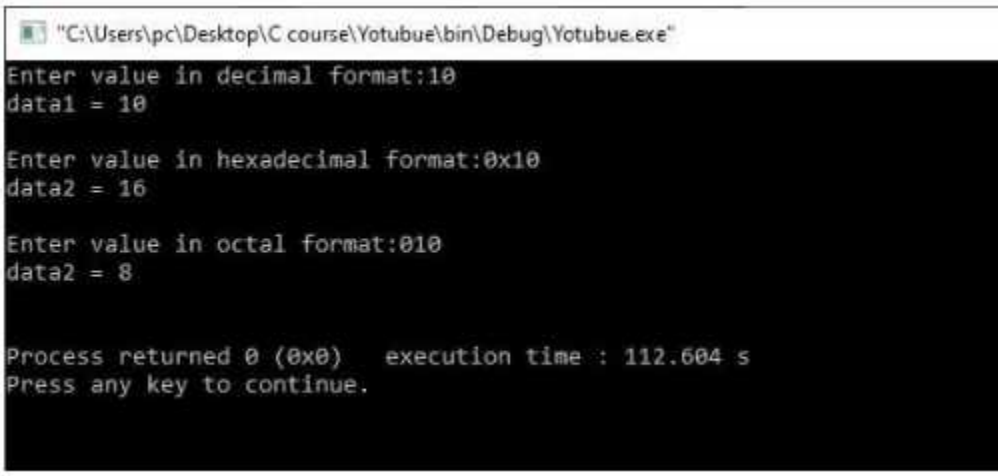
When you are printing using the printf function, there is no specific difference between the %i and %d format specifiers. But both format specifiers behave differently with scanf function.

The %d format specifier takes the integer number as decimal but %i format specifier takes the integer number as decimal, hexadecimal or octal type. it means the %i automatically identified the base of the input integer number.

Note: You must put '0x' for hexadecimal number and '0' for octal number while entering the input number.

```c
#include <stdio.h>

int main()
{
    int data1, data2, data3;

    printf("Enter value in decimal format:");
    scanf("%d",&data1);
    printf("data1 = %i\n\n", data1);

    printf("Enter value in hexadecimal format:");
    scanf("%i",&data2);
    printf("data2 = %i\n\n", data2);

    printf("Enter value in octal format:");
    scanf("%i",&data3);
    printf("data2 = %i\n\n", data3);

    return 0;
}
```

Output:



3. Format specifiers (float) : %f, %e or %E

```c
#include <stdio.h>

int main()
{
    float data = 6.27;

    printf("%f\n", data);
    printf("%e\n", data);

    return 0;
}
```

Output:

6.270000
6.270000e+000

Use of special elements with %f

```c
#include <stdio.h>

int main()
{
    float data = 6.276240;

    printf("%f\n", data);
    printf("%0.2f\n", data);
    printf("%0.4f\n", data);

    return 0;
}
```

Output:

6.276240
6.28
6.2762

You can see, how we can control the precision of float by placing elements with a format specifier. Here %.2f and %.4f will restrict the values up to two and four decimal values.

4. Format specifiers (octal number): %o

```c
#include <stdio.h>
int main()
{
    int data = 65;
    printf("%o\n", data);

    return 0;
}
```

Output:

101

5. Format specifier (Hexadecimal number): %x, %X

```c
#include <stdio.h>
int main()
{
    int data = 11;
    printf("%x\n", data);

    return 0;
}
```

Output: b

6.Format specifier (character array or string): %s

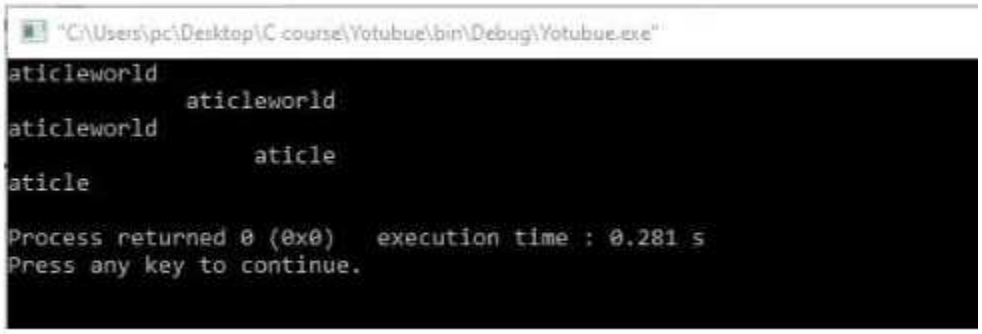
```
#include <stdio.h>
int main()
{
    char blogName[] = "aticleworld";

    printf("%s\n", blogName);

    return 0;
}
```

Output:
aticleworld

Use of special elements with %s

```
#include <stdio.h>
int main()
{
    char blogName[] = "aticleworld";

    printf("%s\n", blogName);
    printf("%24s\n", blogName);
    printf("%-24s\n", blogName);
    printf("%24.6s\n", blogName);
    printf("%-24.6s\n", blogName);

    return 0;
}
```

Output:



In below code, you can see how – and + is used for left and right alignment. The value after the decimal represents precision.

Escape Sequence in C

An escape sequence in C language is a sequence of characters that doesn't represent itself when used inside string literal or character.

It is composed of two or more characters starting with backslash \. For example: \n represents new line.

List of Escape Sequences in C

Escape Sequence	Meaning
\a	Alarm or Beep
\b	Backspace
\f	Form feed
\n	New Line
\r	Carriage Return
\t	Tab (Horizontal)
\v	Vertical Tab
\\	Backslash
\’	Single Quote
\”	Double Quote

\?	Question Mark
\nnn	Octal nuber
\xhh	hexadecimal number
\0	null

Escape Sequence Example

```
1. #include<stdio.h>
2. int main(){
3.     int number=50;
4.    printf("You\nare\nlearning\n\'c\' language\n\"Do you know C language\""
   );
5. return 0;
6. }
```

Output:

```
You
are
learning
'c' language
"Do you know C language"
```

C if else Statement

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

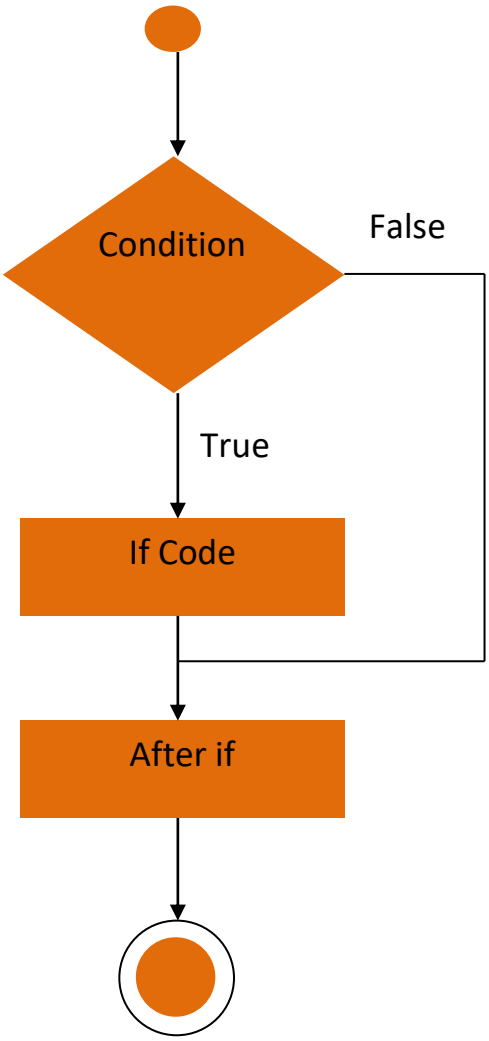
There are the following variants of if statement in C language.

- If statement
- If-else statement
- If else-if ladder

- Nested if

If Statement

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




```
1. if(expression){
2. //code to be executed
3. }
```

Let's see a simple example of C language if statement.

```
1. #include<stdio.h>
2. int main(){
3. int number=0;
4. printf("Enter a number:");
5. scanf("%d",&number);
6. if(number%2==0){
7. printf("%d is even number",number);
8. }
9. return 0;
10.     }
```

Output

```
Enter a number:4
4 is even number
enter a number:5
```

Program to find the largest number of the three.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int a, b, c;
5.      printf("Enter three numbers?");
6.     scanf("%d %d %d",&a,&b,&c);
7.     if(a>b && a>c)
8.     {
9.         printf("%d is largest",a);
10.          }
11.          if(b>a  && b > c)
12.          {
13.              printf("%d is largest",b);
14.          }
15.          if(c>a && c>b)
16.          {
17.              printf("%d is largest",c);
18.          }
19.          if(a == b && a == c)
20.          {
21.              printf("All are equal");
22.          }
23.      }
```

Output

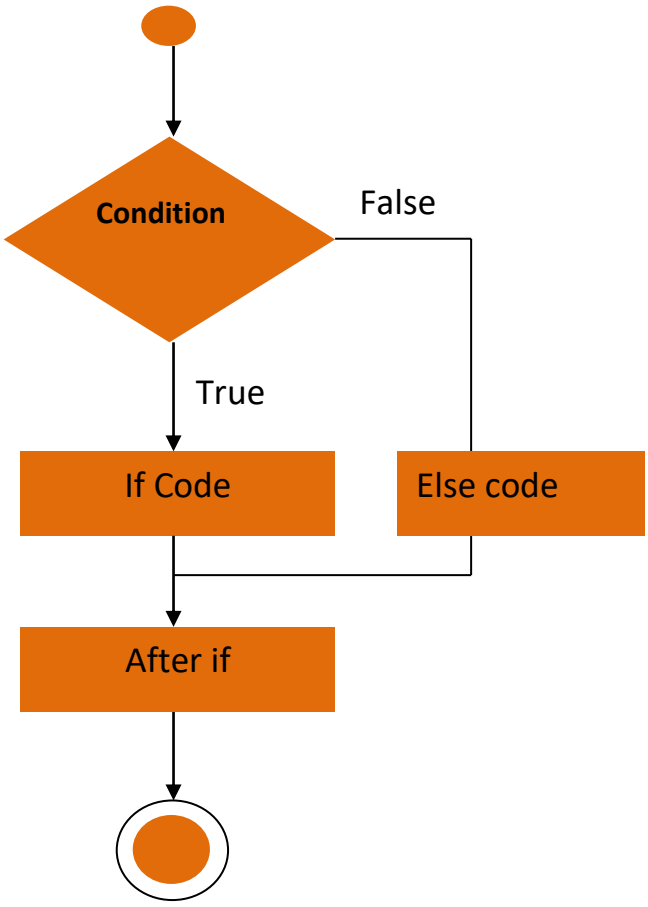
```
Enter three numbers?
12 23 34
```

34 is largest

If-else Statement

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

```
1. if(expression){
2. //code to be executed if condition is true
3. }else{
4. //code to be executed if condition is false
5. }
```



Let's see the simple example to check whether a number is even or odd using if-else statement in C language.

```
1. #include<stdio.h>
2. int main(){
3. int number=0;
```

```
4. printf("enter a number:");
5. scanf("%d",&number);
6. if(number%2==0){
7. printf("%d is even number",number);
8. }
9. else{
10.        printf("%d is odd number",number);
11.        }
12.        return 0;
13.        }
```

Output:

```
enter a number:4
4 is even number
enter a number:5
5 is odd number
```

Program to check whether a person is eligible to vote or not.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int age;
5.     printf("Enter your age?");
6.     scanf("%d",&age);
7.     if(age>=18)
8.     {
9.         printf("You are eligible to vote...");
```

```
10.         }
11.         else
12.         {
13.             printf("Sorry ... you can't vote");
14.         }
15.     }
```

Output:

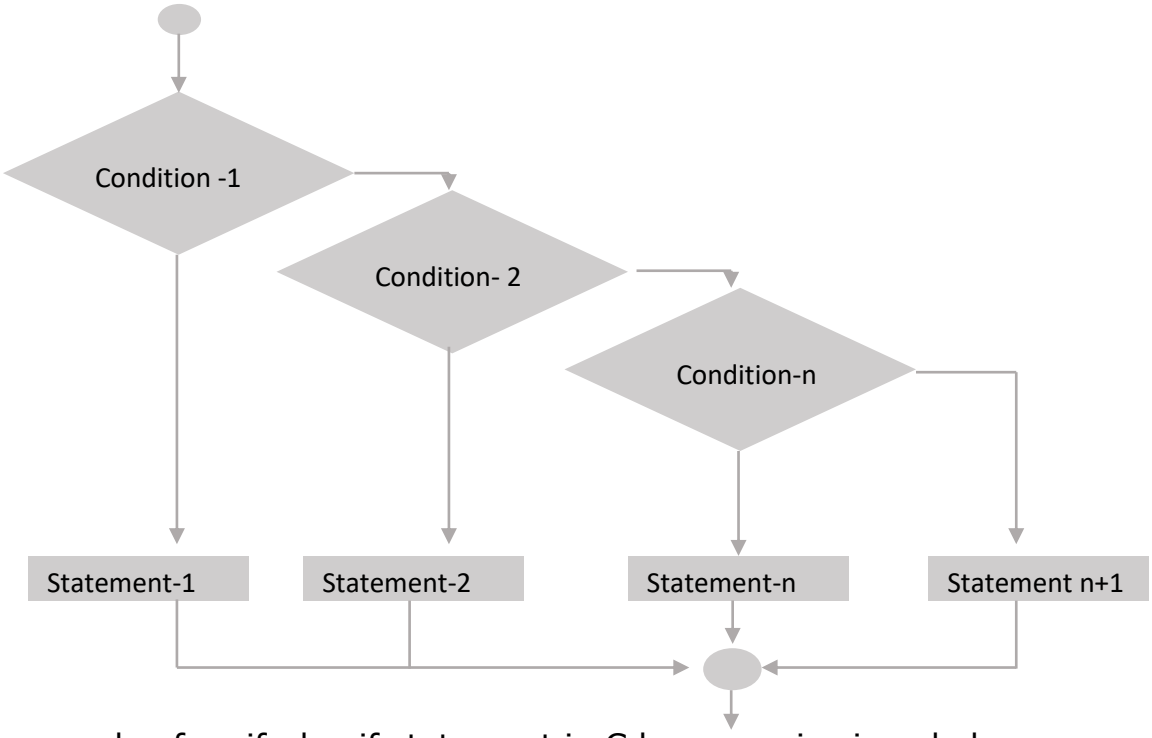
```
Enter your age?18
You are eligible to vote...
Enter your age?13
Sorry ... you can't vote
```

If else-if ladder Statement

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

```
1. if(condition1){
2. //code to be executed if condition1 is true
3. }else if(condition2){
4. //code to be executed if condition2 is true
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
```

```
...
    else{
    //code to be executed if all the conditions are false
    }
```



The example of an if-else-if statement in C language is given below.

```
#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");
```

```
11.       }
12.       else if(number==100){
13.       printf("number is equal to 100");
14.       }
15.       else{
16.       printf("number is not equal to 10, 50 or 100");
17.       }
18.       return 0;
19.       }
```

Output:

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Program to calculate the grade of the student according to the specified marks.

```
1. #include <stdio.h>
2. int main()
3. {
4.     int marks;
5.     printf("Enter your marks?");
6.     scanf("%d",&marks);
7.     if(marks > 85 && marks <= 100)
8.     {
9.         printf("Congrats ! you scored grade A ...");
10.          }
```

```
11.         else if (marks > 60 && marks <= 85)
12.         {
13.             printf("You scored grade B + ...");
14.         }
15.         else if (marks > 40 && marks <= 60)
16.         {
17.             printf("You scored grade B ...");
18.         }
19.         else if (marks > 30 && marks <= 40)
20.         {
21.             printf("You scored grade C ...");
22.         }
23.         else
24.         {
25.             printf("Sorry you are fail ...");
26.         }
27.     }
```

Output:
Enter your marks?10
Sorry you are fail ...
Enter your marks?40
You scored grade C ...
Enter your marks?90
Congrats ! you scored grade A ...

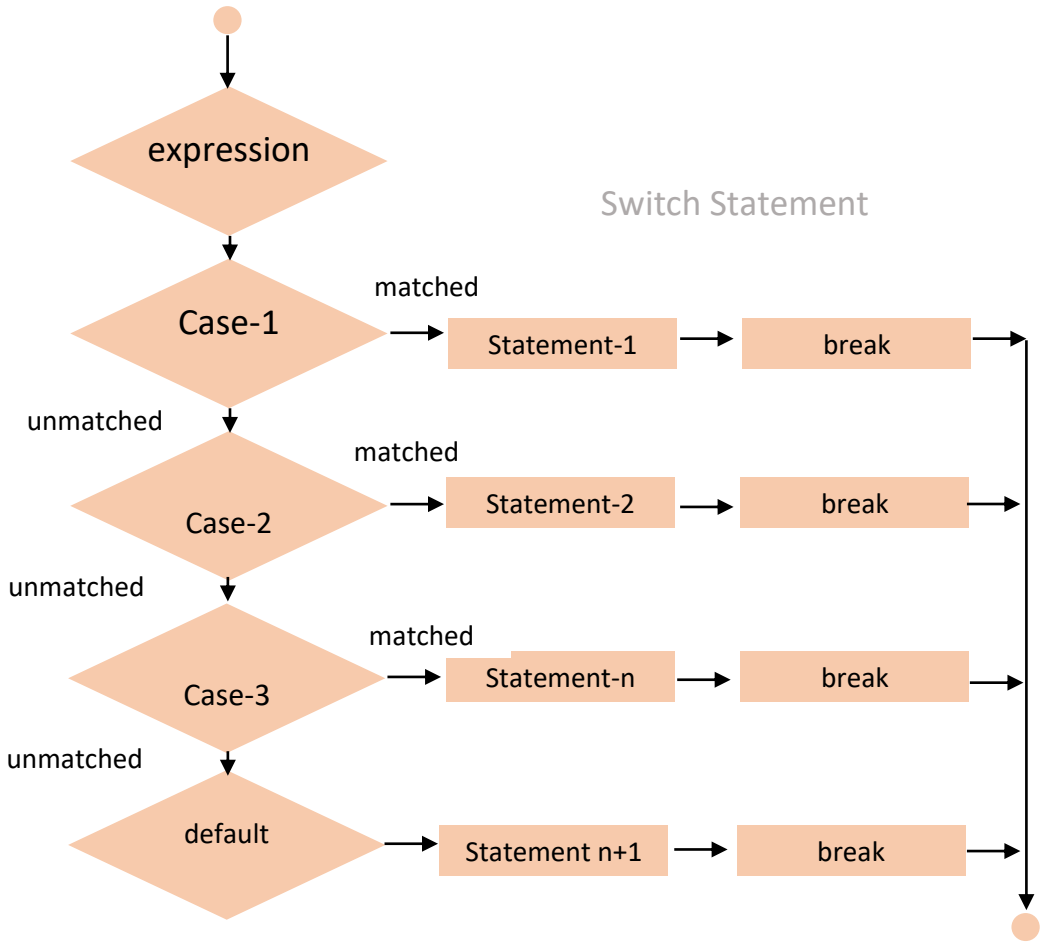
C Switch Statement

The switch statement in C is an alternate to if-else-if ladder statement which allows us to execute multiple operations for the different possibles values of

a single variable called switch variable. Here, We can define various statements in the multiple cases for the different values of a single variable.

The syntax of switch statement in c language is given below:

```
1. switch(expression){
2. case value1:
3.  //code to be executed;
4.  break;  //optional
5. case value2:
6.  //code to be executed;
7.  break;  //optional
8. ......
9.
10.      default:
11.       code to be executed if all cases are not matched;
12.      }
```



Let's see a simple example of c language switch statement.

```
1. #include<stdio.h>
2. int main(){
3. int number=0;
4. printf("enter a number:");
5. scanf("%d",&number);
6. switch(number){
7. case 10:
8. printf("number is equals to 10");
9. break;
10.     case 50:
11.     printf("number is equal to 50");
12.     break;
13.     case 100:
14.     printf("number is equal to 100");
15.     break;
16.     default:
17.     printf("number is not equal to 10, 50 or 100");
18.     }
19.     return 0;
```

```
20.       }
```

Output:

```
enter a number:4
number is not equal to 10, 50 or 100
enter a number:50
number is equal to 50
```

Switch case example 2

```
1. #include <stdio.h>
2. int main()
3. {
4.     int x = 10, y = 5;
5.     switch(x>y && x+y>0)
6.     {
7.        case 1:
8.        printf("hi");
9.        break;
10.             case 0:
11.             printf("bye");
12.             break;
13.             default:
14.             printf(" Hello bye ");
15.          }
16.
17.       }
```

Output:

```
hi
```

C Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies. There are three types of loops used in the C language. In this part of the tutorial, we are going to learn all the aspects of C loops.

Why use loops in C language?

The looping simplifies the complex problems into the easy ones. It enables us to alter the flow of the program so that instead of writing the same code again and again, we can repeat the same code for a finite number of times. For example, if we need to print the first 10 natural numbers then, instead of using the printf statement 10 times, we can print inside a loop which runs up to 10 iterations.

Advantage of loops in C

1) It provides code reusability.

2) Using loops, we do not need to write the same code again and again.

3) Using loops, we can traverse over the elements of data structures (array or linked lists).

Types of C Loops

There are three types of loops in C language that is given below:

1. do while
2. while
3. for

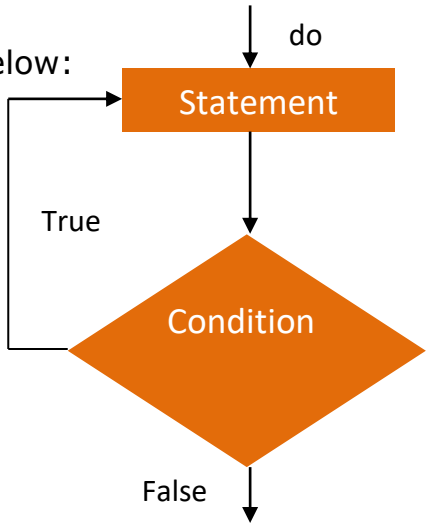
do while loop in C

The do while loop is a post tested loop. Using the do-while loop, we can repeat the execution of several parts of the statements. The do-while loop is

mainly used in the case where we need to execute the loop at least once. The do-while loop is mostly used in menu-driven programs where the termination condition depends upon the end user.

do while loop syntax

The syntax of the C language do-while loop is given below:




```
1. do{
2. //code to be executed
3. }while(condition);
```

Example 1

```
1. #include<stdio.h>
2. #include<stdlib.h>
3. void main ()
4. {
5.     char c;
6.     int choice,dummy;
7.     do{
8.     printf("\n1. Print Hello\n2. Print Sau prakashani \n3. Exit\n");
9.     scanf("%d",&choice);
10.          switch(choice)
11.          {
12.             case 1 :
13.             printf("Hello");
14.             break;
15.             case 2:
16.             printf("Sau prakashani");
17.             break;
18.             case 3:
19.             exit(0);
```

```
20.             break;
21.             default:
22.             printf("please enter valid choice");
23.         }
24.         printf("do you want to enter more?");
25.         scanf("%d",&dummy);
26.         scanf("%c",&c);
27.         }while(c=='y');
28.     }
```

Output

```
1. Print Hello
2. Print Sau prakashani
3. Exit
1
Hello
do you want to enter more?
y

1. Print Hello
2. Print Sau prakashani
3. Exit
2
Sau prakashani
do you want to enter more?
n
```

Example 2:

```
1. #include<stdio.h>
2. int main(){
3. int i=1;
4. do{
```

```
5. printf("%d \n",i);
6. i++;
7. }while(i<=10);
8. return 0;
9. }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Program to print table for the given number using do while loop

```
1. #include<stdio.h>
2. int main(){
3. int i=1,number=0;
4. printf("Enter a number: ");
5. scanf("%d",&number);
6. do{
7. printf("%d \n",(number*i));
8. i++;
9. }while(i<=10);
10.     return 0;
11.     }
```

Output:

```
Enter a number: 5
5
10
15
20
25
30
35
40
45
50
```

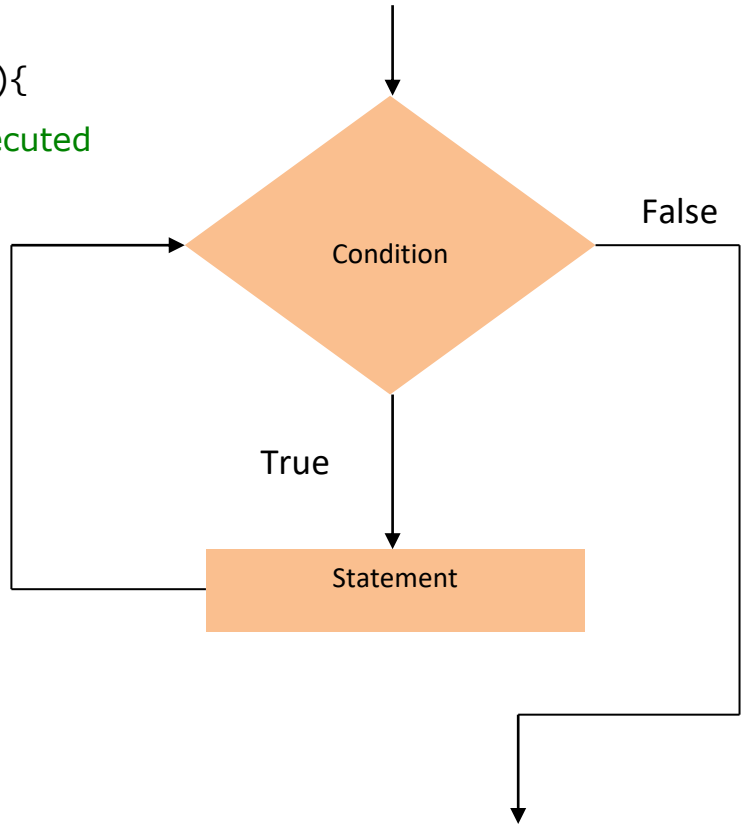
while loop in C

While loop is also known as a pre-tested loop. In general, a while loop allows a part of the code to be executed multiple times depending upon a given Boolean condition. It can be viewed as a repeating if statement. The while loop is mostly used in the case where the number of iterations is not known in advance.

Syntax of while loop in C language

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

```
1. while(condition){
2. //code to be executed
3. }
```



Example of the while loop in C language

Let's see the simple program of while loop that prints table of 1.

```
1. #include<stdio.h>
2. int main(){
3. int i=1;
4. while(i<=10){
5. printf("%d \n",i);
6. i++;
7. }
8. return 0;
9. }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Program to print table for the given number using while loop in C

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

```
3. int i=1,number=0,b=9;
4. printf("Enter a number: ");
5. scanf("%d",&number);
6. while(i<=10){
7. printf("%d \n",(number*i));
8. i++;
9. }
10.     return 0;
11.     }
```

Output:

```
Enter a number: 50
50
100
150
200
250
300
350
400
450
500
```

for loop in C

The **for loop** in C language is used to iterate the statements or a part of the program several times. It is frequently used to traverse the data structures like the array and linked list.

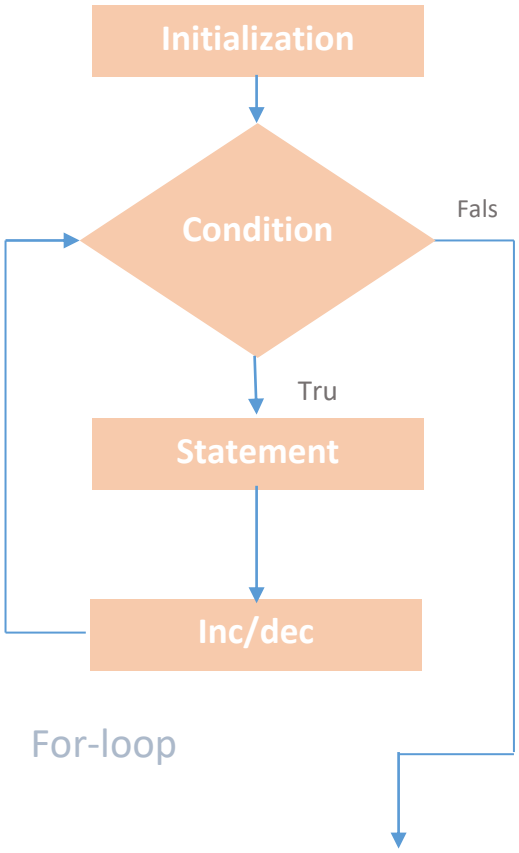
Syntax of for loop in C

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

```
1. for(Expression 1; Expression 2; Expression 3){
2. //code to be executed
3. }
```

Example:

```
1. #include<stdio.h>
2. int main(){
3. int i=0;
4. for(i=1;i<=10;i++){
5. printf("%d \n",i);
6. }
7. return 0;
8. }
```



For-loop

Output:

```
1
2
3
4
5
6
7
8
9
10
```

Example:

```
1. #include<stdio.h>
2. int main(){
3. int i=1,number=0;
4. printf("Enter a number: ");
5. scanf("%d",&number);
6. for(i=1;i<=10;i++){
```

```
7.  printf("%d \n",(number*i));
8.  }
9.  return 0;
10.       }
```

Output:

```
Enter a number: 2
2
4
6
8
10
12
14
16
18
20
```

Example:

```
1.  #include <stdio.h>
2.  int main()
3.  {
4.     int a,b,c;
5.     for(a=0,b=12,c=23;a<2;a++)
6.     {
7.        printf("%d ",a+b+c);
8.     }
9.  }
```

Output:

```
35 36
```

Example:

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i=1;
5.     for(;i<5;i++)
6.     {
7.         printf("%d ",i);
8.     }
9. }
```

Output:

1 2 3 4

Example:

```
1. #include <stdio.h>
2. int main()
3. {
4.     int i,j,k;
5.     for(i=0,j=0,k=0;i<4,k<8,j<10;i++)
6.     {
7.         printf("%d %d %d\n",i,j,k);
8.         j+=2;
9.         k+=3;
10.            }
11.        }
```

Output:

```
0 0 0
1 2 3
2 4 6
3 6 9
4 8 12
```

Example:

```
1. #include<stdio.h>
2. int main()
3. {
4.     int i;
5.     for(i=0;;i++)
6.     {
7.         printf("%d",i);
8.     }
9. }
```

Output:

infinite loop

C break statement

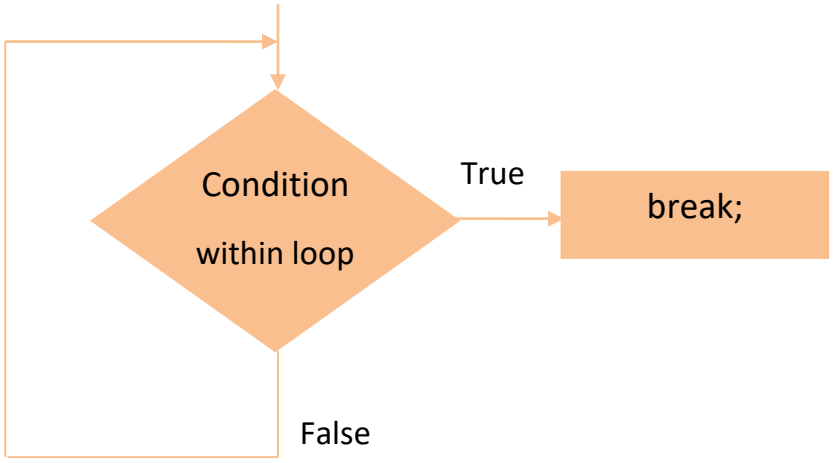
The break is a keyword in C which is used to bring the program control out of the loop. The break statement is used inside loops or switch statement. The break statement breaks the loop one by one, i.e., in the case of nested loops, it breaks the inner loop first and then proceeds to outer loops. The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

1. With switch case
2. With loop

Syntax:

```
1. //loop or switch case
2. break;
```

Example:




```
1. #include<stdio.h>
2. #include<stdlib.h>
3. void main ()
4. {
5.     int i;
6.     for(i = 0; i<10; i++)
7.     {
8.         printf("%d ",i);
9.         if(i == 5)
10.             break;
11.         }
12.         printf("came outside of loop i = %d",i);
13.
14.     }
```

Output:

0 1 2 3 4 5 came outside of loop i = 5

Example:

```
1. #include<stdio.h>
2. int main(){
3. int i=1,j=1;//initializing a local variable
4. for(i=1;i<=3;i++){
5. for(j=1;j<=3;j++){
6. printf("%d &d\n",i,j);
```

```
7. if(i==2 && j==2){
8. break;//will break loop of j only
9. }
10.       }//end of for loop
11.       return 0;
12.       }
```

Output:

```
1 1
1 2
1 3
2 1
2 2
3 1
3 2
3 3
```

C continue statement

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

Syntax:

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

Example:

```
1. #include<stdio.h>
2. void main ()
3. {
4.     int i = 0;
5.     while(i!=10)
```

```
6.    {
7.        printf("%d", i);
8.        continue;
9.        i++;
10.           }
11.       }
```

Output:

infinite loop

```
1. #include<stdio.h>
2. int main(){
3. int i=1;//initializing a local variable
4. //starting a loop from 1 to 10
5. for(i=1;i<=10;i++){
6. if(i==5){//if value of i is equal to 5, it will continue the loop
7. continue;
8. }
9. printf("%d \n",i);
10.       }//end of for loop
11.       return 0;
12.       }
```

Output:

```
1
2
3
4
6
7
8
9
10
```

C Functions

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

Advantage of functions in C

There are the following advantages of C functions.

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.
- However, Function calling is always a overhead in a C program.

Function Aspects

There are three aspects of a C function.

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

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

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

SN	c function aspect	syntax
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

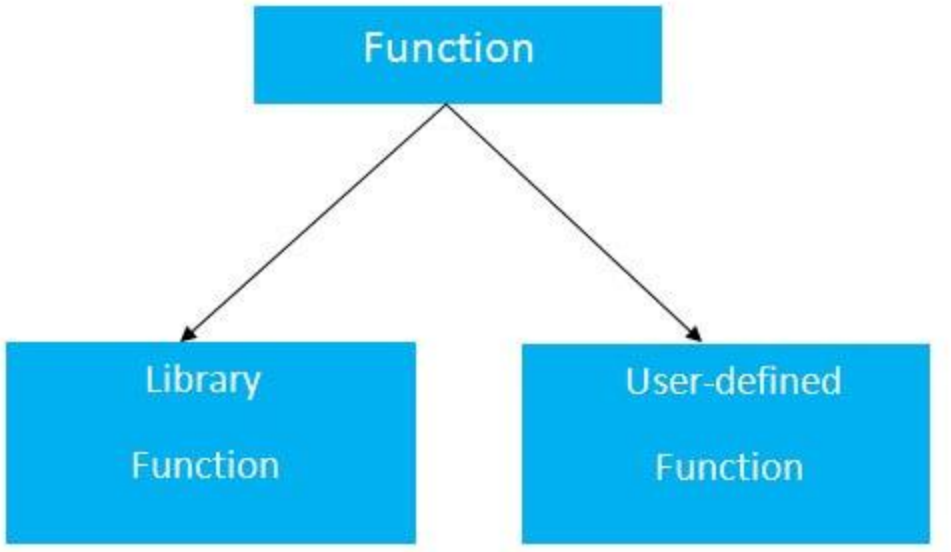
The syntax of creating function in c language is given below:

```
1. return_type function_name(data_type parameter...){
2. //code to be executed
3. }
```

Types of Functions

There are two types of functions in C programming:

1. **Library Functions**: are the functions which are declared in the C header files such as scanf(), printf(), gets(), puts(), ceil(), floor() etc.
2. **User-defined functions**: are the functions which are created by the C programmer, so that he/she can use it many times. It reduces the complexity of a big program and optimizes the code.



Return Value

A C function may or may not return a value from the function. If you don't have to return any value from the function, use void for the return type.

Let's see a simple example of C function that doesn't return any value from the function.

Example without return value:

```
1. void hello(){
2. printf("hello c");
3. }
```

In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

```
1. float get(){
2. return 10.2;
3. }
```

Now, you need to call the function, to get the value of the function.

Different aspects of function calling

A function may or may not accept any argument. It may or may not return any value. Based on these facts, There are four different aspects of function calls.

- function without arguments and without return value
- function without arguments and with return value
- function with arguments and without return value
- function with arguments and with return value

Example for Function without argument and return value

Example 1

```
1. #include<stdio.h>
2. void printName();
3. void main ()
4. {
5.     printf("Hello ");
6.     printName();
7. }
8. void printName()
9. {
10.          printf("Sau prakashani");
11.      }
```

Output:

```
Hello Sau prakashani
```

Example 2

```
1. #include<stdio.h>
2. void sum();
3. void main()
```

```
{
    printf("\nGoing to calculate the sum of two numbers:");
    sum();
}
void sum()
{
        int a,b;
        printf("\nEnter two numbers");
        scanf("%d %d",&a,&b);
        printf("The sum is %d",a+b);
    }
```

Output:

Going to calculate the sum of two numbers:

Enter two numbers 10
24

The sum is 34

Example: program to calculate the area of the square

```
#include<stdio.h>
int sum();
void main()
{
    printf("Going to calculate the area of the square\n");
    float area = square();
    printf("The area of the square: %f\n",area);
}
int square()
    {
        float side;
        printf("Enter the length of the side in meters: ");
        scanf("%f",&side);
        return side * side;
    }
```

Output:

```
Going to calculate the area of the square
Enter the length of the side in meters: 10
The area of the square: 100.000000
```

C Library Functions

Library functions are the inbuilt function in C that are grouped and placed at a common place called the library. Such functions are used to perform some specific operations. For example, printf is a library function used to print on the console. The library functions are created by the designers of compilers. All C standard library functions are defined inside the different header files saved with the extension **.h**. We need to include these header files in our program to make use of the library functions defined in such header files. For example, To use the library functions such as printf/scanf we need to include stdio.h in our program which is a header file that contains all the library functions regarding standard input/output.

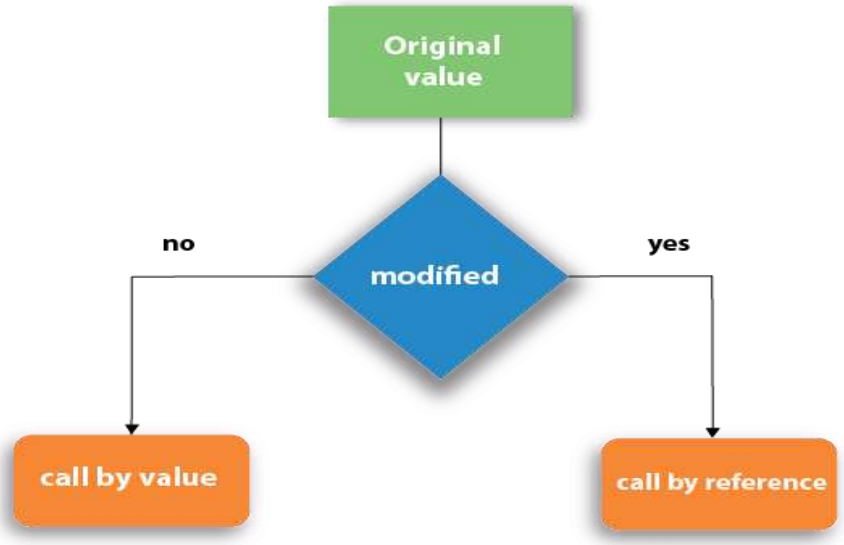
The list of mostly used header files is given in the following table.

SN	Header file	Description
1	stdio.h	This is a standard input/output header file. It contains all the library functions regarding standard input/output.
2	conio.h	This is a console input/output header file.
3	string.h	It contains all string related library functions like gets(), puts(),etc.
4	Stdlib.h	This header file contains all the general library functions like malloc(), calloc(), exit(), etc
5	math.h	This header file contains all the math operations related functions like sqrt(), pow(), etc.
6	time.h	This header file contains all the time-related functions.
7	ctype.h	This header file contains all character handling functions.
8	stdarg.h	Variable argument functions are defined in this header file.

9	signal.h	All the signal handling functions are defined in this header file.
10	setjmp.h	This file contains all the jump functions
11	locale.h	This file contains locale functions.
12	errno.h	This file contains error handling functions.
13	assert.h	This file contains diagnostics functions.

Call by value and Call by reference in C

There are two methods to pass the data into the function in C language, i.e., call by value and call by reference.



Let's understand call by value and call by reference in c language one by one.

Call by value in C

- In call by value method, the value of the actual parameters is copied into the formal parameters. In other words, we can say that the value of the variable is used in the function call in the call by value method.

- In call by value method, we can not modify the value of the actual parameter by the formal parameter.
- In call by value, different memory is allocated for actual and formal parameters since the value of the actual parameter is copied into the formal parameter.
- The actual parameter is the argument which is used in the function call whereas formal parameter is the argument which is used in the function definition.

Let's try to understand the concept of call by value in c language by the example given below:

```
1. #include<stdio.h>
2. void change(int num) {
3.     printf("Before adding value inside function num=%d \n",num);
4.     num=num+100;
5.     printf("After adding value inside function num=%d \n", num);
6. }
7. int main() {
8.     int x=100;
9.     printf("Before function call x=%d \n", x);
10.          change(x);//passing value in function
11.          printf("After function call x=%d \n", x);
12.       return 0;
13.       }
```

Output:

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=100
```

Call by Value Example: Swapping the values of the two variables

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

```
2. void swap(int , int); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // pr
   inting the value of a and b in main
8.     swap(a,b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The val
   ue of actual parameters do not change by changing the formal parameters in
    call by value, a = 10, b = 20
10.        }
11.        void swap (int a, int b)
12.        {
13.            int temp;
14.            temp = a;
15.            a=b;
16.            b=temp;
17.            printf("After swapping values in function a = %d, b = %d\n",a,b); /
   / Formal parameters, a = 20, b = 10
18.}
```

Output

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 10, b = 20
```

Call by reference in C

- In call by reference, the address of the variable is passed into the function call as the actual parameter.
- The value of the actual parameters can be modified by changing the formal parameters since the address of the actual parameters is passed.
- In call by reference, the memory allocation is similar for both formal parameters and actual parameters. All the operations in the function

are performed on the value stored at the address of the actual parameters, and the modified value gets stored at the same address.

Consider the following example for the call by reference.

```
1. #include<stdio.h>
2. void change(int *num) {
3.     printf("Before adding value inside function num=%d \n",*num);
4.     (*num) += 100;
5.     printf("After adding value inside function num=%d \n", *num);
6. }
7. int main() {
8.     int x=100;
9.     printf("Before function call x=%d \n", x);
10.          change(&x);//passing reference in function
11.          printf("After function call x=%d \n", x);
12.       return 0;
13.       }
```

Output:

```
Before function call x=100
Before adding value inside function num=100
After adding value inside function num=200
After function call x=200
```

Call by reference Example: Swapping the values of the two variables

```
1. #include <stdio.h>
2. void swap(int *, int *); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // pr
   inting the value of a and b in main
```

```
8.     swap(&a,&b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The val
   ues of actual parameters do change in call by reference, a = 10, b = 20
10.    }
11.    void swap (int *a, int *b)
12.    {
13.       int temp;
14.       temp = *a;
15.       *a=*b;
16.       *b=temp;
17.       printf("After swapping values in function a = %d, b = %d\n",*a,*b)
   ; // Formal parameters, a = 20, b = 10
18.    }
```

Output:

```
Before swapping the values in main a = 10, b = 20
After swapping values in function a = 20, b = 10
After swapping values in main a = 20, b = 10
```

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.

Generally, iterative solutions are more efficient than recursion since function call is always overhead. Any problem that can be solved recursively, can also be solved iteratively. However, some problems are best suited to be solved

by the recursion, for example, tower of Hanoi, Fibonacci series, factorial finding, etc.

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

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

Output:

```
Enter the number whose factorial you want to calculate?5
factorial = 120
```

Example of recursion in C

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

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

Output:

```
Enter the value of n?12
144
```

C Array

An array is defined as the collection of similar type of data items stored at contiguous memory locations. Arrays are the derived data type in C programming language which can store the primitive type of data such as int, char, double, float, etc. It also has the capability to store the collection of derived data types, such as pointers, structure, etc. The array is the simplest data structure where each data element can be randomly accessed by using its index number.

C array is beneficial if you have to store similar elements. For example, if we want to store the marks of a student in 6 subjects, then we don't need to define different variables for the marks in the different subject. Instead of that, we can define an array which can store the marks in each subject at the contiguous memory locations.

By using the array, we can access the elements easily. Only a few lines of code are required to access the elements of the array.

Properties of Array

The array contains the following properties.

- Each element of an array is of same data type and carries the same size, i.e., int = 4 bytes.
- Elements of the array are stored at contiguous memory locations where the first element is stored at the smallest memory location.
- Elements of the array can be randomly accessed since we can calculate the address of each element of the array with the given base address and the size of the data element.

Advantage of C Array

1) Code Optimization: Less code to the access the data.

2) Ease of traversing: By using the for loop, we can retrieve the elements of an array easily.

3) Ease of sorting: To sort the elements of the array, we need a few lines of code only.

4) Random Access: We can access any element randomly using the array.

Disadvantage of C Array

1) Fixed Size: Whatever size, we define at the time of declaration of the array, we can't exceed the limit. So, it doesn't grow the size dynamically like LinkedList which we will learn later.

Declaration of C Array

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

data_type array_name[array_size];

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

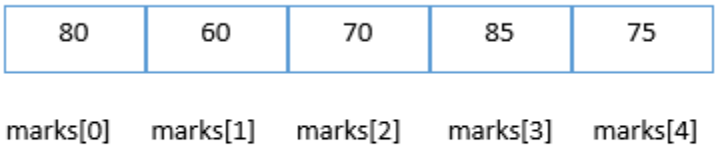
```
int marks[5];
```

Here, int is the *data_type*, marks are the *array_name*, and 5 is the *array_size*.

Initialization of C Array

The simplest way to initialize an array is by using the index of each element. We can initialize each element of the array by using the index. Consider the following example.

```
1. marks[0]=80;//initialization of array
2. marks[1]=60;
3. marks[2]=70;
4. marks[3]=85;
5. marks[4]=75;
```



Initialization of Array

C array example

```
1. #include<stdio.h>
2. int main(){
3. int i=0;
4. int marks[5];//declaration of array
5. marks[0]=80;//initialization of array
6. marks[1]=60;
7. marks[2]=70;
8. marks[3]=85;
9. marks[4]=75;
10.//traversal of array
11.for(i=0;i<5;i++){
12.printf("%d \n",marks[i]);
13.}//end of for loop
14.return 0;
15.}
```

Output

```
80
60
70
85
75
```

C Array: Declaration with Initialization

We can initialize the c array at the time of declaration. Let's see the code.

```
int marks[5]={20,30,40,50,60};
```

In such case, there is **no requirement to define the size**. So it may also be written as the following code.

```
int marks[]={20,30,40,50,60};
```

Let's see the C program to declare and initialize the array in C.

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

```
3. int i=0;
4. int marks[5]={20,30,40,50,60};//declaration and initialization of array
5.  //traversal of array
6. for(i=0;i<5;i++){
7. printf("%d \n",marks[i]);
8. }
9. return 0;
10.       }
```

Output

```
20
30
40
50
60
```

C Array Example: Sorting an array

In the following program, we are using bubble sort method to sort the array in ascending order.

```
1. #include<stdio.h>
2. void main ()
3. {
4.     int i, j,temp;
5.     int a[10] = { 10, 9, 7, 101, 23, 44, 12, 78, 34, 23};
6.     for(i = 0; i<10; i++)
7.     {
8.         for(j = i+1; j<10; j++)
9.         {
10.                 if(a[j] > a[i])
11.                 {
12.                     temp = a[i];
13.                     a[i] = a[j];
```

```
                a[j] = temp;
            }
        }
    }
    printf("Printing Sorted Element List ...\n");
    for(i = 0; i<10; i++)
    {
        printf("%d\n",a[i]);
    }
}
```

Program to print the largest and second largest element of the array.

```
#include<stdio.h>
void main ()
{
    int arr[100],i,n,largest,sec_largest;
    printf("Enter the size of the array?");
    scanf("%d",&n);
    printf("Enter the elements of the array?");
    for(i = 0; i<n; i++)
    {
        scanf("%d",&arr[i]);
    }
    largest = arr[0];
    sec_largest = arr[1];
    for(i=0;i<n;i++)
    {
        if(arr[i]>largest)
        {
            sec_largest = largest;
            largest = arr[i];
        }
        else if (arr[i]>sec_largest && arr[i]!=largest)
```

```
22.             {
23.                 sec_largest=arr[i];
24.             }
25.         }
26.         printf("largest = %d, second largest = %d",largest,sec_largest);
27.
28.}
```

Two Dimensional Array in C

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

Declaration of two dimensional Array in C

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

data_type array_name[rows][columns];

Consider the following example.

```
int twodimen[4][3];
```

Here, 4 is the number of rows, and 3 is the number of columns.

Initialization of 2D Array in C

In the 1D array, we don't need to specify the size of the array if the declaration and initialization are being done simultaneously. However, this will not work with 2D arrays. We will have to define at least the second dimension of the array. The two-dimensional array can be declared and defined in the following way.

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

Two-dimensional array example in C

```
1. #include<stdio.h>
2. int main(){
3. int i=0,j=0;
4. int arr[4][3]={{1,2,3},{2,3,4},{3,4,5},{4,5,6}};
5. //traversing 2D array
6. for(i=0;i<4;i++){
7.  for(j=0;j<3;j++){
8.    printf("arr[%d] [%d] = %d \n",i,j,arr[i][j]);
9.  }//end of j
10.       }//end of i
11.       return 0;
12.       }
```

Output

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

C 2D array example: Storing elements in a matrix and printing it.

```
1. #include<stdio.h>
2. void main ()
3. {
```

```
4.    int arr[3][3],i,j;
5.    for (i=0;i<3;i++)
6.    {
7.       for (j=0;j<3;j++)
8.       {
9.          printf("Enter a[%d][%d]: ",i,j);
10.                scanf("%d",&arr[i][j]);
11.           }
12.        }
13.        printf("\n printing the elements ....\n");
14.        for(i=0;i<3;i++)
15.        {
16.           printf("\n");
17.           for (j=0;j<3;j++)
18.           {
19.              printf("%d\t",arr[i][j]);
20.           }
21.        }
22.     }
```

Output:

```
Enter a[0][0]: 56
Enter a[0][1]: 10
Enter a[0][2]: 30
Enter a[1][0]: 34
Enter a[1][1]: 21
Enter a[1][2]: 34

Enter a[2][0]: 45
Enter a[2][1]: 56
Enter a[2][2]: 78

 printing the elements ....

56      10      30
34      21      34
45      56      78
```

Store Numbers and Calculate Average Using Arrays

```
#include<stdio.h>
int main() {
    int n, i;
    float num[100], sum = 0.0, avg;

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

    while (n > 100 || n < 1) {
        printf("Error! number should in range of (1 to 100).\n");
        printf("Enter the number again: ");
        scanf("%d", &n);
    }
    for (i = 0; i < n; ++i) {
        printf("%d. Enter number: ", i + 1);
        scanf("%f", &num[i]);
        sum += num[i];
    }

    avg = sum / n;
    printf("Average = %.2f", avg);
    return 0;
}
```

Output

```
Enter the numbers of elements: 6
```

1. Enter number: 45.3
2. Enter number: 67.5
3. Enter number: -45.6
4. Enter number: 20.34
5. Enter number: 33
6. Enter number: 45.6

Average = 27.69

Find the Largest Element in an array

```
#include<stdio.h>
int main() {
    int i, n;
    float arr[100];
    printf("Enter the number of elements (1 to 100): ");
    scanf("%d", &n);

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

    // storing the largest number to arr[0]
    for(i = 1; i < n; ++i) {
        if (arr[0] < arr[i])
            arr[0] = arr[i];
    }
```

```
    printf("Largest element = %.2f", arr[0]);


    return 0;
}
```

Output

```
Enter the number of elements (1 to 100): 5
Enter number1: 34.5
Enter number2: 2.4
Enter number3: -35.5
Enter number4: 38.7
Enter number5: 24.5
Largest element = 38.70
```

This program takes n number of elements from the user and stores it in arr[].

C Pointers

Pointers (pointer variables) are special variables that are used to store addresses rather than values.

Pointer Syntax

Here is how we can declare pointers.

```
int* p;
```

Here, we have declared a pointer p of int type.

You can also declare pointers in these ways.

```
int *p1;
```

```
int * p2;
```

Let's take another example of declaring pointers.

```
int* p1, p2;
```

Here, we have declared a pointer p1 and a normal variable p2.

Address in C

If you have a variable var in your program, &var will give you its address in the memory.

We have used address numerous times while using the scanf() function.

```
scanf("%d", &var);
```

Here, the value entered by the user is stored in the address of var variable. Let's take a working example.

```
#include<stdio.h>
int main()
{
  int var = 5;
  printf("var: %d\n", var);

  // Notice the use of & before var
  printf("address of var: %p", &var);
  return 0;
}
```

Output:

```
var: 5
address of var: 2686778
```

Note: You will probably get a different address when you run the above code.

Assigning addresses to Pointers

Let's take an example.

```
int* pc, c;

c = 5;

pc = &c;
```

Here, 5 is assigned to the c variable. And, the address of c is assigned to the pc pointer.

Get Value of Thing Pointed by Pointers

To get the value of the thing pointed by the pointers, we use the * operator. For example:

```
int* pc, c;

c = 5;

pc = &c;

printf("%d", *pc);   // Output: 5
```

Here, the address of c is assigned to the pc pointer. To get the value stored in that address, we used *pc.

Note: In the above example, pc is a pointer, not *pc. You cannot and should not do something like *pc = &c;

By the way, * is called the dereference operator (when working with pointers). It operates on a pointer and gives the value stored in that pointer.

Changing Value Pointed by Pointers

Let's take an example.

```
int* pc, c;

c = 5;

pc = &c;

c = 1;

printf("%d", c);    // Output: 1

printf("%d", *pc);  // Ouptut: 1
```

We have assigned the address of c to the pc pointer.

Then, we changed the value of c to 1. Since pc and the address of c is the same, *pc gives us 1.

Let's take another example.

```
int* pc, c;

c = 5;

pc = &c;

*pc = 1;

printf("%d", *pc);  // Ouptut: 1

printf("%d", c);    // Output: 1
```

We have assigned the address of c to the pc pointer.

Then, we changed *pc to 1 using *pc = 1;. Since pc and the address of c is the same, c will be equal to 1.

Let's take one more example.

```
int* pc, c, d;

c = 5;
```

```
d = -15;


pc = &c; printf("%d", *pc); // Output: 5

pc = &d; printf("%d", *pc); // Ouptut: -15
```

Initially, the address of c is assigned to the pc pointer using pc = &c;. Since c is 5, *pc gives us 5.

Then, the address of d is assigned to the pc pointer using pc = &d;. Since d is -15, *pc gives us -15.

Example: Working of Pointers

Let's take a working example.

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

  c = 22;
  printf("Address of c: %p\n", &c);
  printf("Value of c: %d\n\n", c);  // 22

  pc = &c;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 22

  c = 11;
  printf("Address of pointer pc: %p\n", pc);
  printf("Content of pointer pc: %d\n\n", *pc); // 11
```

```
    *pc = 2;
    printf("Address of c: %p\n", &c);
    printf("Value of c: %d\n\n", c); // 2
    return 0;
}
```

Output

```
Address of c: 2686784
Value of c: 22

Address of pointer pc: 2686784
Content of pointer pc: 22

Address of pointer pc: 2686784
Content of pointer pc: 11

Address of c: 2686784
Value of c: 2
```

Relationship Between Arrays and Pointers

An array is a block of sequential data. Let's write a program to print addresses of array elements.

```
#include<stdio.h>
int main() {
    int x[4];
    int i;
```

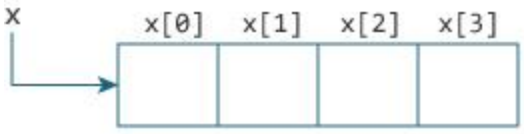
```
    for(i = 0; i < 4; ++i) {
        printf("&x[%d] = %p\n", i, &x[i]);
    }
    printf("Address of array x: %p", x);
    return 0;
}
```

Output

```
&x[0] = 1450734448
&x[1] = 1450734452
&x[2] = 1450734456
&x[3] = 1450734460
Address of array x: 1450734448
```

There is a difference of 4 bytes between two consecutive elements of array x. It is because the size of int is 4 bytes (on our compiler).

Notice that, the address of &x[0] and x is the same. It's because the variable name x points to the first element of the array.



From the above example, it is clear that &x[0] is equivalent to x. And, x[0] is equivalent to *x.

Similarly,

&x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).

&x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).

...

Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).

Example 1: Pointers and Arrays

```
#include<stdio.h>
int main() {
  int i, x[6], sum = 0;
  printf("Enter 6 numbers: ");
  for(i = 0; i < 6; ++i) {
  // Equivalent to scanf("%d", &x[i]);
     scanf("%d", x+i);
  // Equivalent to sum += x[i]
     sum += *(x+i);
  }
  printf("Sum = %d", sum);
  return 0;
}
```

When you run the program, the output will be:

```
Enter 6 numbers:  2
 3
 4
 4
 12
 4
Sum = 29
```

Here, we have declared an array x of 6 elements. To access elements of the array, we have used pointers.

Example 2: Arrays and Pointers

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

```
  int x[5] = {1, 2, 3, 4, 5};
  int* ptr;
  // ptr is assigned the address of the third element
  ptr = &x[2];
  printf("*ptr = %d \n", *ptr);   // 3
  printf("*(ptr+1) = %d \n", *(ptr+1)); // 4
  printf("*(ptr-1) = %d", *(ptr-1));  // 2
  return 0;
}
```

When you run the program, the output will be:

```
*ptr = 3
*(ptr+1) = 4
*(ptr-1) = 2
```

In this example, &x[2], the address of the third element, is assigned to the ptr pointer. Hence, 3 was displayed when we printed *ptr.

And, printing *(ptr+1) gives us the fourth element. Similarly, printing *(ptr-1) gives us the second element.

Example: Passing Pointers to Functions

```
#include<stdio.h>
void addOne(int* ptr) {
  (*ptr)++; // adding 1 to *ptr
}
int main()
{
  int* p, i = 10;
  p = &i;
```

```
  addOne(p);
  printf("%d", *p); // 11
  return 0;
}
```

Here, the value stored at p, *p, is 10 initially.

We then passed the pointer p to the addOne() function. The ptr pointer gets this address in the addOne() function.

Inside the function, we increased the value stored at ptr by 1 using (*ptr)++;. Since ptr and p pointers both have the same address, *p inside main() is also 11.

C Strings

In C programming, a string is a sequence of characters terminated with a null character \0. For example:

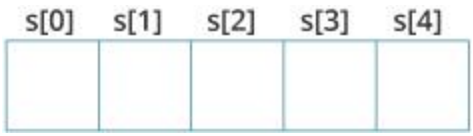
```
char c[] = "c string";
```

When the compiler encounters a sequence of characters enclosed in the double quotation marks, it appends a null character \0 at the end by default.

How to declare a string?

Here's how you can declare strings:

```
char s[5];
```

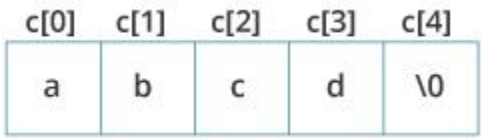


Here, we have declared a string of 5 characters.

How to initialize strings?

You can initialize strings in a number of ways.

```
char c[] = "abcd";
char c[50] = "abcd";
char c[] = {'a', 'b', 'c', 'd', '\0'};
char c[5] = {'a', 'b', 'c', 'd', '\0'};
```



Let's take another example:

```
char c[5] = "abcde";
```

Here, we are trying to assign 6 characters (the last character is '\0') to a char array having 5 characters. This is bad and you should never do this.

Read String from the user

You can use the scanf() function to read a string.

The scanf() function reads the sequence of characters until it encounters whitespace (space, newline, tab etc.).

Example 1: scanf() to read a string

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

```
{
    char name[20];
    printf("Enter name: ");
    scanf("%s", name);
    printf("Your name is %s.", name);
    return 0;
}
```

Output:

Enter name: Dennis Ritchie

Your name is Dennis.

How to read a line of text?

You can use the fgets() function to read a line of string. And, you can use puts() to display the string.

Example 2: fgets() and puts()

```
#include<stdio.h>
int main()
{
    char name[30];
    printf("Enter name: ");
    fgets(name, sizeof(name), stdin);  // read string
    printf("Name: ");
    puts(name);    // display string
    return 0;
}
```

Output

```
Enter name: Tom Hanks
Name: Tom Hanks
```

Here, we have used fgets() function to read a string from the user.

```
fgets(name, sizeof(name), stdlin); // read string
```

The sizeof(name) results to 30. Hence, we can take a maximum of 30 characters as input which is the size of the name string.

To print the string, we have used puts(name);.

Example 3: Passing string to a Function

```
#include<stdio.h>
void displayString(char str[]);
int main()
{
   char str[50];
   printf("Enter string: ");
   fgets(str, sizeof(str), stdin);
   displayString(str);     // Passing string to a function.
   return 0;
}
void displayString(char str[])
{
   printf("String Output: ");
   puts(str);
}
```

Example 4: Strings and Pointers

```
#include<stdio.h>
int main(void) {
  char name[] = "Harry Potter";
  printf("%c", *name);     // Output: H
  printf("%c", *(name+1));   // Output: a
  printf("%c", *(name+7));   // Output: o
  char *namePtr;
  namePtr = name;
  printf("%c", *namePtr);     // Output: H
  printf("%c", *(namePtr+1));   // Output: a
  printf("%c", *(namePtr+7));   // Output: o
}
```

Commonly Used String Functions

strlen() - calculates the length of a string

strcpy() - copies a string to another

strcmp() - compares two strings

strcat() - concatenates two strings

String Manipulations In C Programming Using Library Functions

You need to often manipulate strings according to the need of a problem. Most, if not all, of the time string manipulation can be done manually but, this makes programming complex and large.

To solve this, C supports a large number of string handling functions in the standard library "string.h".

Few commonly used string handling functions are discussed below:

Function	Work of Function
strlen()	computes string's length
strcpy()	copies a string to another
strcat()	concatenates(joins) two strings
strcmp()	compares two strings
strlwr()	converts string to lowercase
strupr()	converts string to uppercase

Strings handling functions are defined under "string.h" header file.

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

Note: You have to include the code below to run string handling functions.

Example: gets() and puts()

```
#include<stdio.h>
int main()
{
   char name[30];
   printf("Enter name: ");
   gets(name);     //Function to read string from user.
   printf("Name: ");
```

```
    puts(name);    //Function to display string.
    return 0;
}
```

Note: Though, gets() and puts() function handle strings, both these functions are defined in "stdio.h" header file.