C Programming

WITH #THEKAMALNAIN

THEKAMALNAIN

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

It can be defined by the following ways:
1. Mother language
2. System programming language
3. Procedure-oriented programming language
4. Structured programming language
5. Mid-level programming language

1) C as a mother language

C language is considered as the mother language of all the modern programming languages because **most of the compilers, JVMs, Kernels, etc. are written in C language**, and most of the programming languages follow C syntax, for example, C++, Java, C#, etc.

2) C as a system programming language

A system programming language is used to create system software. C language is a system programming language because it **can be used to do low-level programming (for example driver and kernel)**. It is generally used to create hardware devices, OS, drivers, kernels, etc. For example, Linux kernel is written in C.

3) C as a procedural language

A procedure is known as a function, method, routine, subroutine, etc. A procedural language **specifies a series of steps for the program to solve the problem**.
A procedural language breaks the program into functions, data structures, etc.
C is a procedural language. In C, variables and function prototypes must be declared before being used.

4) C as a structured programming language

A structured programming language is a subset of the procedural language. **Structure means to break a program into parts or blocks** so that it may be easy to understand.
In the C language, we break the program into parts using functions. It makes the program easier to understand and modify.

5) C as a mid-level programming language

C is considered as a middle-level language because it **supports the feature of both low-level and high-level languages**. C language program is converted into assembly code, it supports pointer arithmetic (low-level), but it is machine independent (a feature of high-level).
A **Low-level language** is specific to one machine, i.e., machine dependent. It is machine dependent, fast to run. But it is not easy to understand.
A **High-Level language** is not specific to one machine, i.e., machine independent. It is easy to understand.

C Program

In this tutorial, all C programs are given with C compiler so that you can quickly change the C program code.

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

History of C language is interesting to know. Here we are going to discuss a brief history of the c language.

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

Dennis Ritchie is known as the **founder of the c language**.

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

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:

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

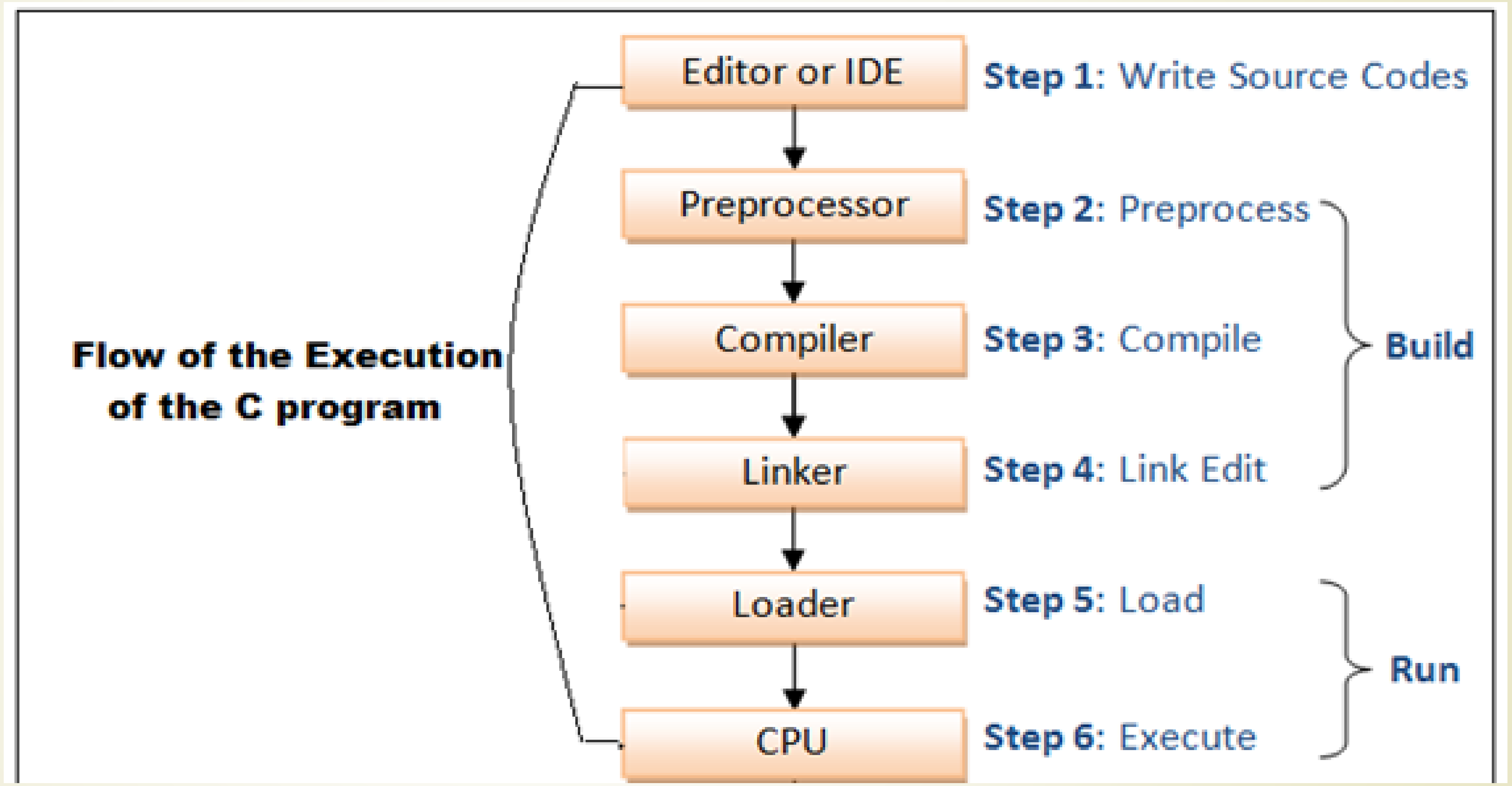
#include <stdio.h> includes the **standard input output** library functions. The printf() function is defined in stdio.h .

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

printf() The printf() function is **used to print data** on the console.
return 0 The return 0 statement, returns execution status to the OS. The 0 value is used for successful execution and 1 for unsuccessful execution

Flow of C Program

The C program follows many steps in execution. To understand the flow of C program well, let us see a simple program first.



Let's try to understand the flow of above program by the figure given below.

1) C program (source code) is sent to preprocessor first. The preprocessor is responsible to convert preprocessor directives into their respective values. The preprocessor generates an expanded source code.

2) Expanded source code is sent to compiler which compiles the code and converts it into assembly code.
3) The assembly code is sent to assembler which assembles the code and converts it into object code. Now a simple.obj file is generated.

4) The object code is sent to linker which links it to the library such as header files. Then it is converted into executable code. A simple.exe file is generated.

5) The executable code is sent to loader which loads it into memory and then it is executed. After execution, output is sent to console.

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() function

The **printf() function** is used for output. It prints the given statement to the console

The syntax of printf() function is given below:

printf("format string",argument_list);

scanf() function

The scanf() function is used for input. It reads the input data from the console.

scanf("format string",argument_list);

Program to print cube of given number

Let's see a simple example of c language that gets input from the user and prints the cube of the given number.

```
#include<stdio.h>
int main(){
int number;
printf("enter a number:");
scanf("%d",&number);
printf("cube of number is:%d ",number*number*numb
er);
return 0;
}
```

The **scanf("%d",&number)** statement reads integer number from the console and stores the given value in number variable.

The **printf("cube of number is:%d ",number*number*number)** statement prints the cube of number on the console.

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:
type variable_list;
The example of declaring the variable is given below:

int a;
float b;
char c;

```
int a=10,b=20;//declaring 2 variable of integer type
float f=20.8;
char c='A';
```

Rules for defining variables

- A variable can have alphabets, digits, and underscore.
- A variable name can start with the alphabet, and underscore only. It can't start with a digit.
- No whitespace is allowed within the variable name.
- A variable name must not be any reserved word or keyword, e.g. int, float, etc.

Valid variable names:
int a;
int _ab;
int a30;

Invalid variable names:
int 2;
int a b;
int long;

Types of Variables in C

There are many types of variables in c:

1. local variable
2. global variable
3. static variable
4. automatic variable
5. external variable

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.

```
void function1(){
int x=10;//local variable
}
```

You must have to initialize the local variable before it is used.

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.

```
int value=20;//global variable

void function1()
{
    int x=10;//local variable
}
```

Static Variable

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

It retains its value between multiple function calls.

```
void function1(){
int x=10;//local variable
static int y=10;//static variable
x=x+1;
y=y+1;
printf("%d,%d",x,y);
}
```

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.

Automatic Variable

All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using **auto keyword**.

```
void main(){
int x=10;//local variable (also automatic)
auto int y=20;//automatic variable
}
```

External Variable

We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use **extern keyword**.

myfile.h

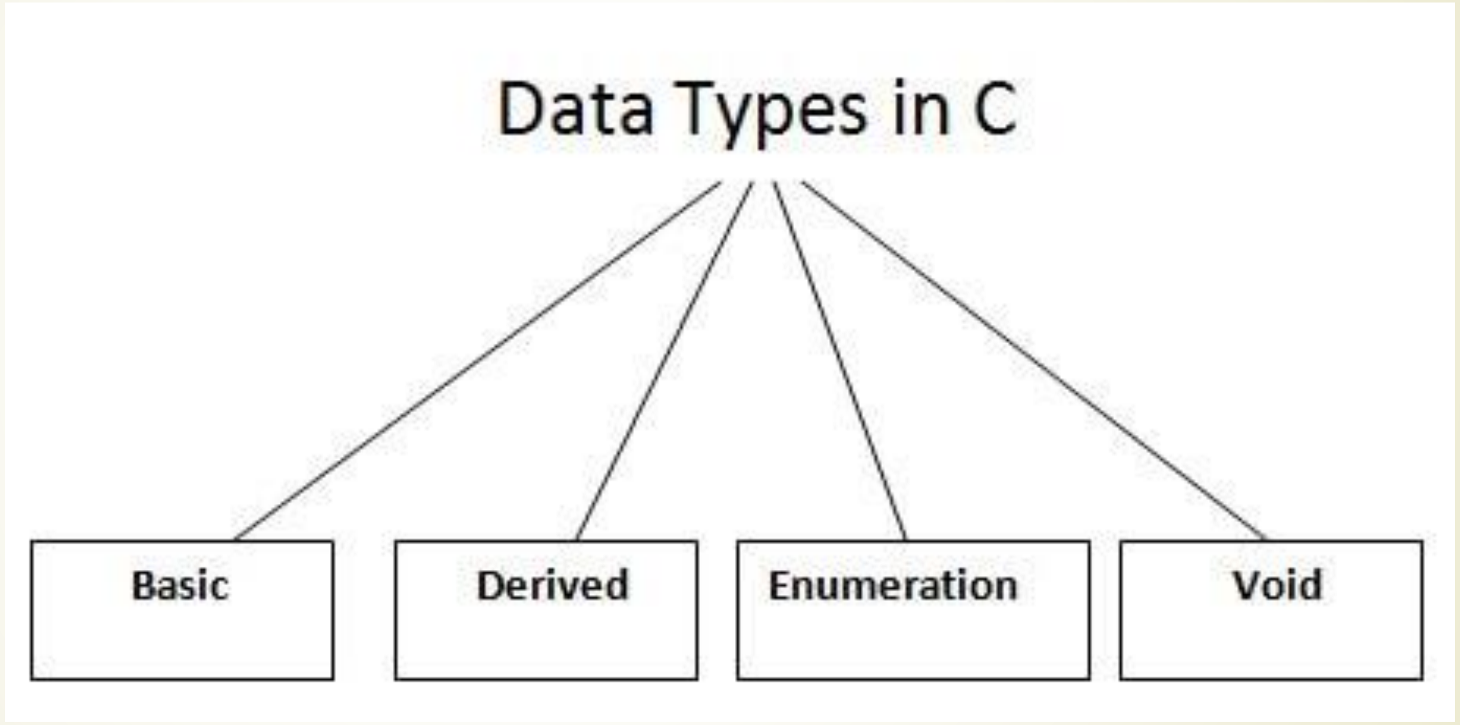
```
extern int x=10;//external variable (also global)
```

program1.c

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

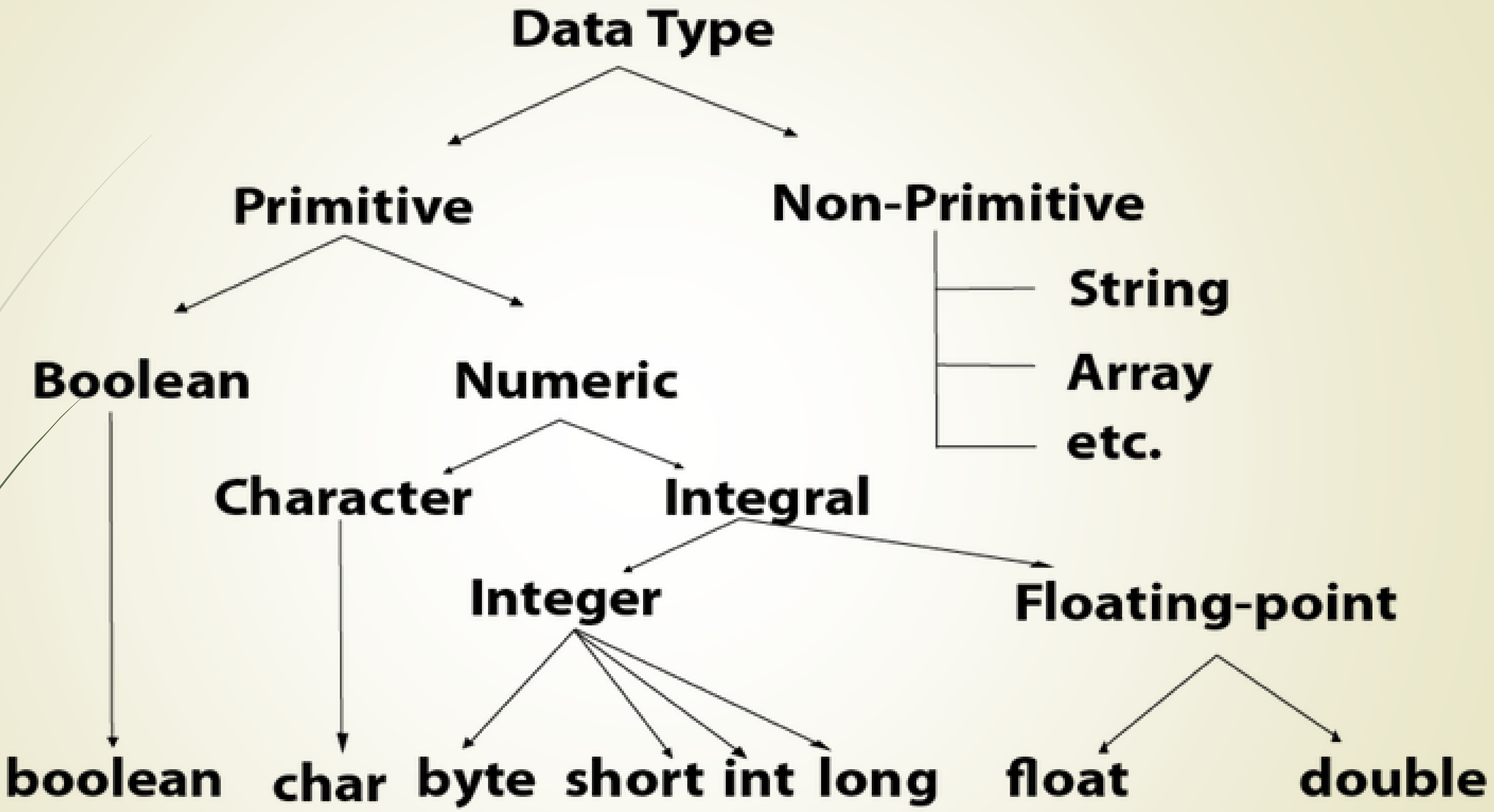
Data Types in C

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



There are the following data types in C language.

Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void


Data Type
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Keywords in C

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	continu e	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

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.

1. **Arithmetic Operators**
2. **Relational Operators**
3. **Shift Operators**
4. **Logical Operators**
5. **Bitwise Operators**
6. **Ternary or Conditional Operators**
7. **Assignment Operator**
8. **Misc Operato**

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.

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

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

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

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.

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 number
\xhh	hexadecimal number
\0	Null

Constants in C

A constant is a value or variable that can't be changed in the program, for example: 10, 20, 'a', 3.4, "c programming" etc. There are different types of constants in C programming.

Constant	Example
Decimal Constant	10, 20, 450 etc.
Real or Floating-point Constant	10.3, 20.2, 450.6 etc.
Octal Constant	021, 033, 046 etc.
Hexadecimal Constant	0x2a, 0x7b, 0xaa etc.
Character Constant	'a', 'b', 'x' etc.
String Constant	"c", "c program", "c lang" etc.

2 ways to define constant in C

There are two ways to define constant in C programming.

1. const keyword

2. #define preprocessor

1) C const keyword
The const keyword is used to define constant in C programming.
const float PI=3.14;
Now, the value of PI variable can't be changed.

```
#include<stdio.h>
int main(){
    const float PI=3.14;
    printf("The value of PI is: %f",PI);
    return 0;
}
```

C #define

The #define preprocessor directive is used to define constant or micro substitution. It can use any basic data type.

Syntax:

#define token value

Let's see an example of #define to define a constant.

```
#include <stdio.h>
#define PI 3.14
main() {
   printf("%f",PI);
}
```

C if else Statement

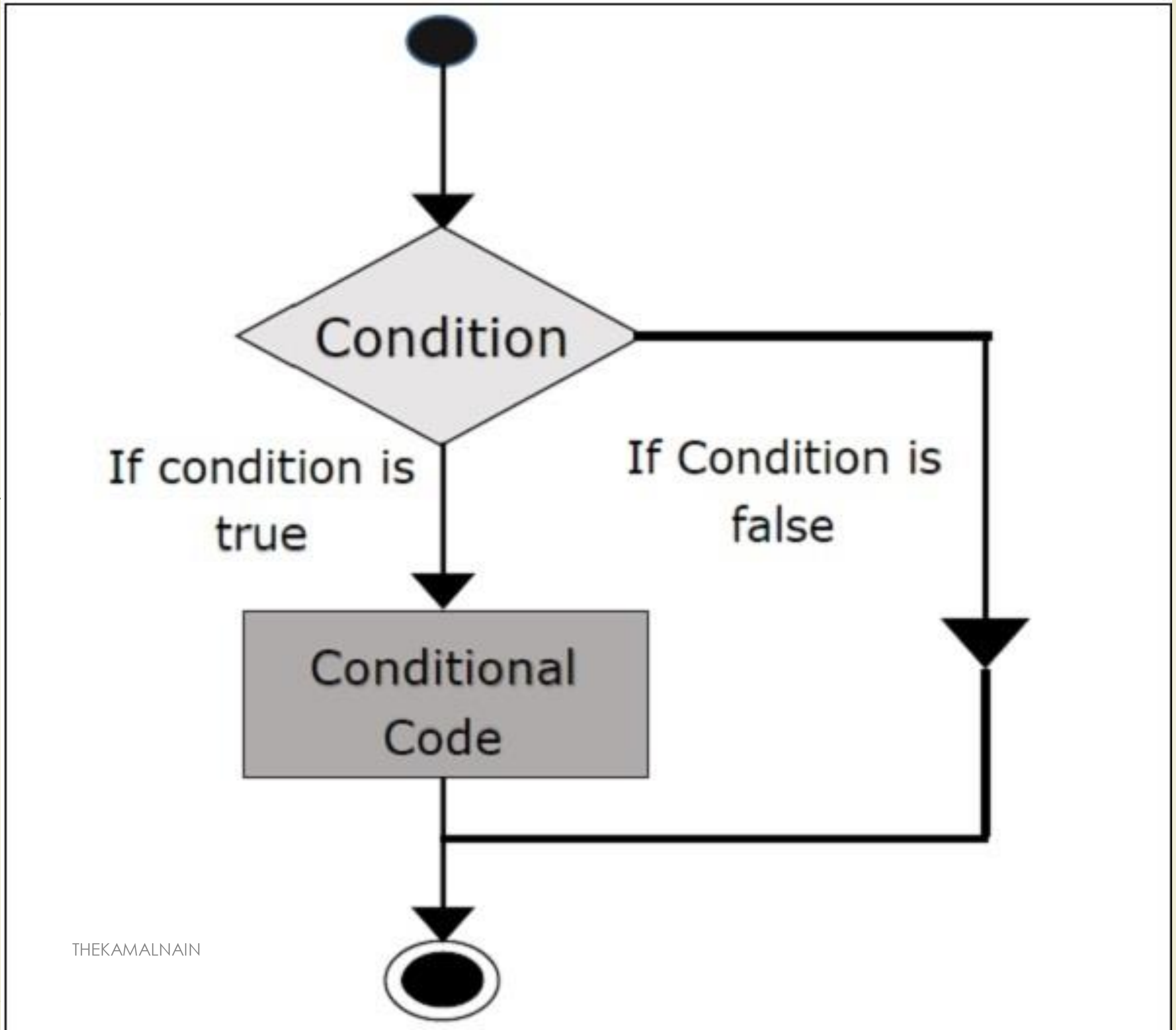
The if-else statement in C is used to perform the operations based on some specific condition. The operations specified in if block are executed if and only if the given condition is true.
There are the following variants of if statement in C language.

- 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.}
```

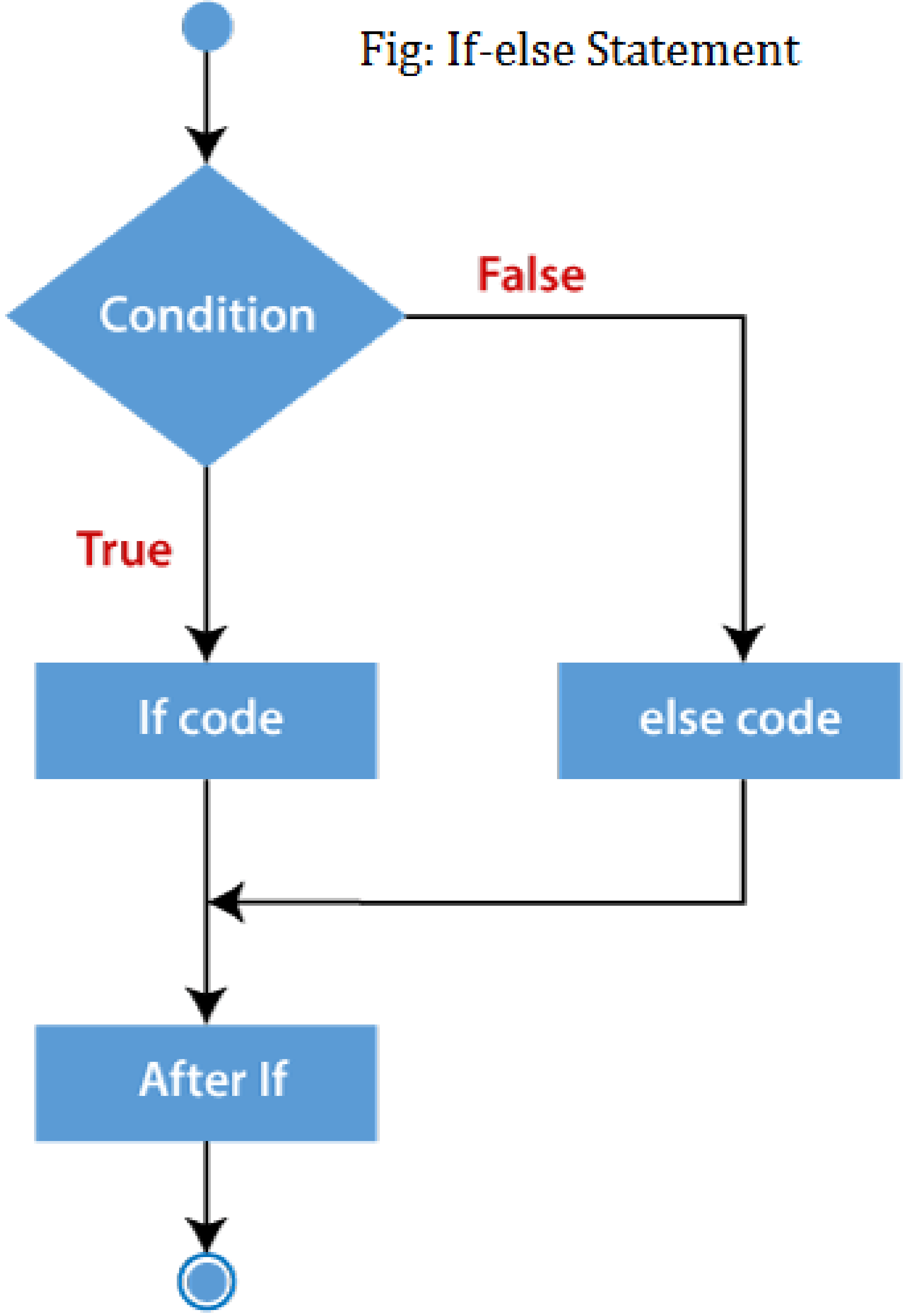

Condition
If condition is true
If Condition is false
Conditional Code

```c
#include<stdio.h>
int main(){
int number=0;
printf("Enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);

}
return 0;
}
```

If-else Statement

The if-else statement is used to perform two operations for a single condition. The if-else statement is an extension to the if statement using which, we can perform two different operations, i.e., one is for the correctness of that condition, and the other is for the incorrectness of the condition. Here, we must notice that if and else block cannot be executed simiulteneously.

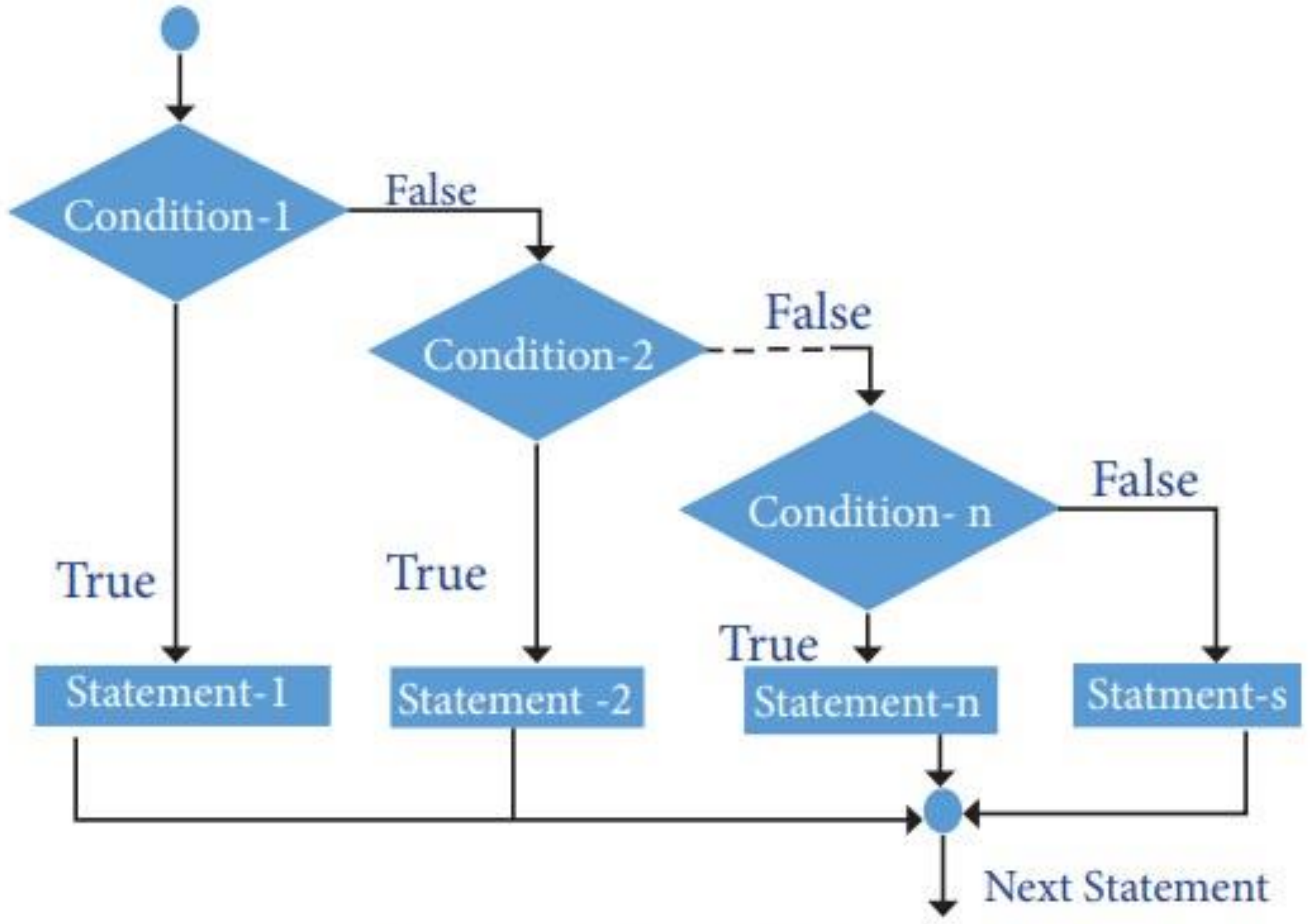
Fig: If-else Statement



```c
#include<stdio.h>
int main(){
int number=0;
printf("enter a number:");
scanf("%d",&number);
if(number%2==0){
printf("%d is even number",number);
}
else{
printf("%d is odd number",number);
}
return 0;
}
```

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.



Flowchart 10.4 if-else ladder flow chart

```
#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 Loops

The looping can be defined as repeating the same process multiple times until a specific condition satisfies

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

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

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

Do while

```
#include<stdio.h>
// int main(int argc, char const *argv[])
// {
//     int i = 1;
//     do
//     {
//         printf("hello world\n");
//         i++;
//     } while (i<10);
//     return 0;
// }
```

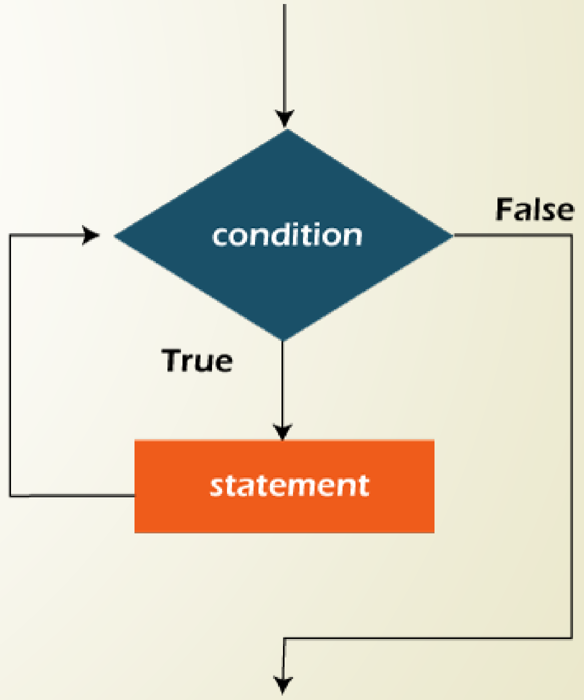
for loop in C

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

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

```
for(initialization;condition;incr/decr){
//code to be executed
}
```

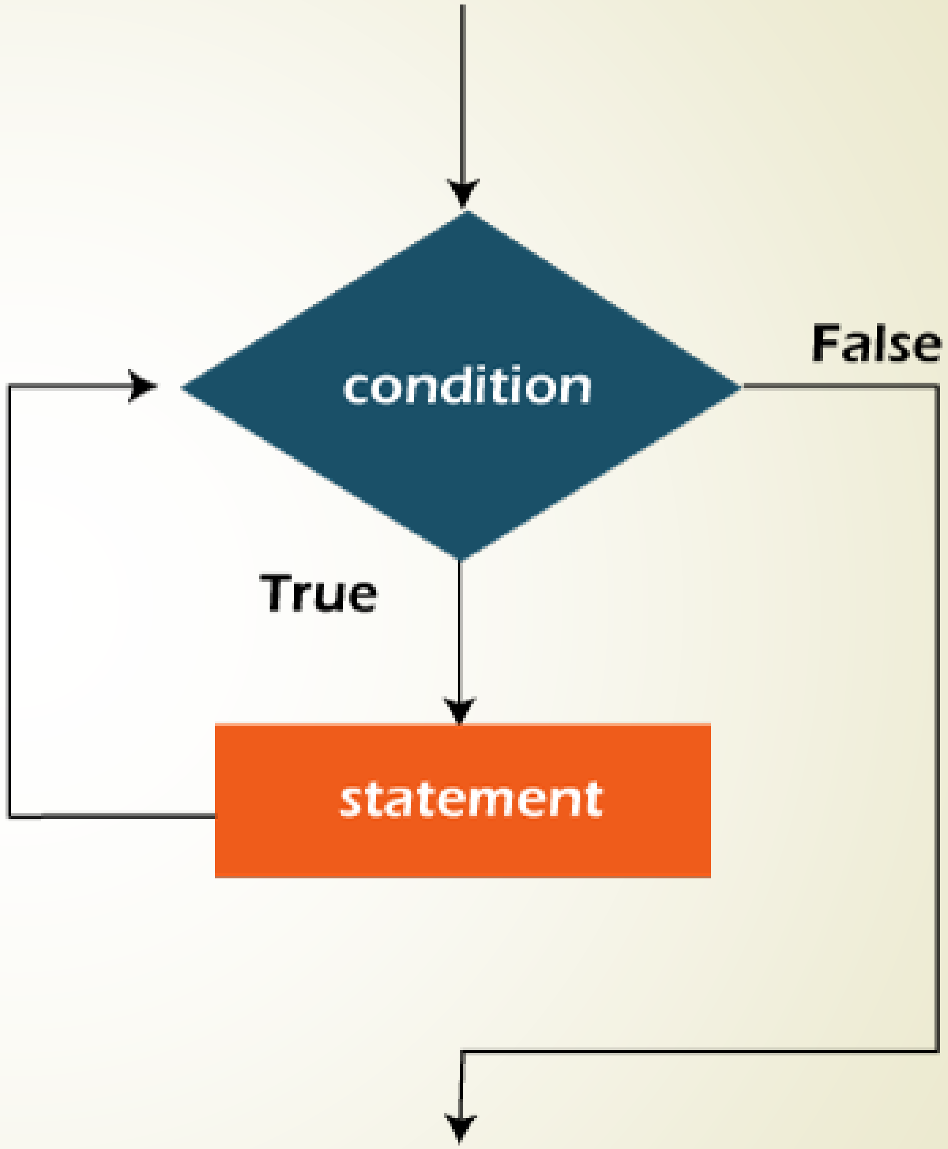
```
// #include <stdio.h>
// int main(int argc, char const *argv[])
// {
//     for(int i = 1; i < 10;i++)
//     {
//         printf("%d\n", i);
//     }
//     return 0;
// }
```



THEKAMALNAIN

//while loop

```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```



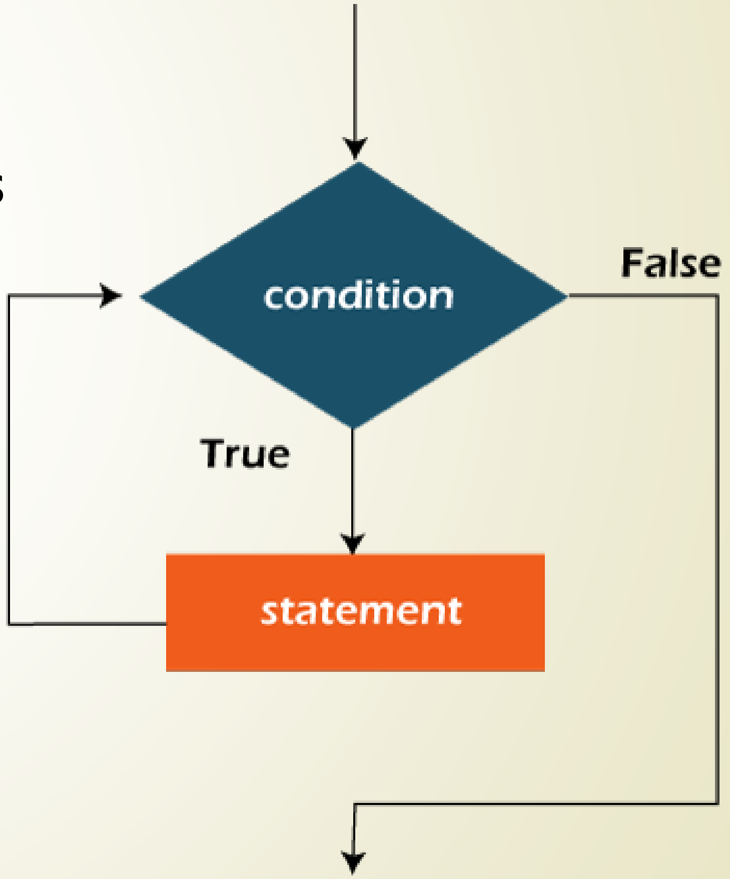
while loop in C

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.

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.}
```



```
#include<stdio.h>
int main(){
int i=1;
while(i<=10){
printf("%d \n",i);
i++;
}
return 0;
}
```

```
#include<stdio.h>
int main(){
int i=1,number=0,b=9;
printf("Enter a number: ");
scanf("%d",&number);
while(i<=10){
printf("%d \n",(number*i));
i++;
}
return 0;
}
```

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.

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

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
......

default:
 code to be executed if all cases are not matched;
}
```

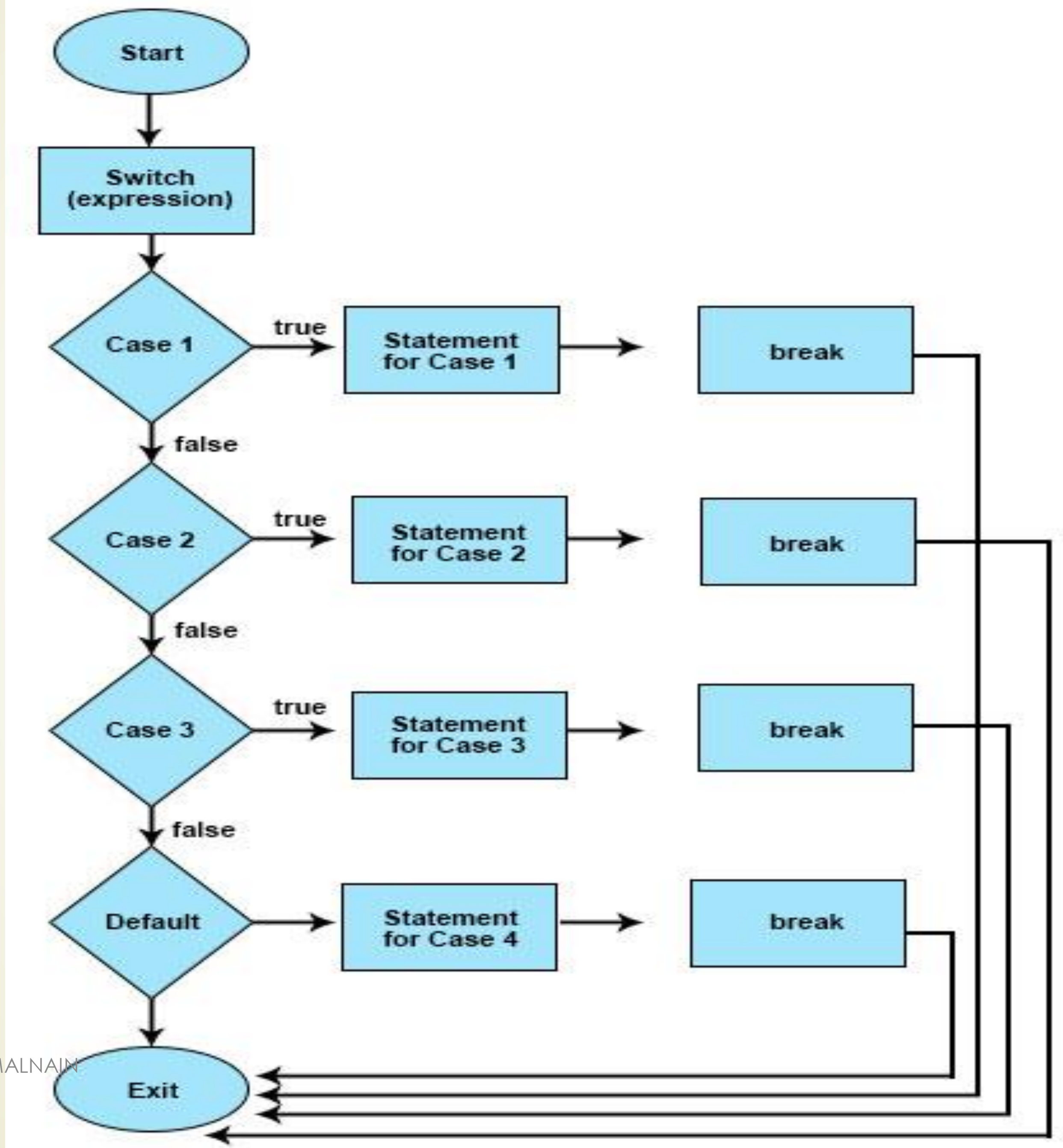
Rules for switch statement in C language

1) The *switch expression* must be of an integer or character type.
2) The *case value* must be an integer or character constant.
3) The *case value* can be used only inside the switch statement.
4) The *break statement* in switch case is not must. It is optional. If there is no break statement found in the case, all the cases will be executed present after the matched case. It is known as *fall through* the state of C switch statement.

Let's try to understand it by the examples. We are assuming that there are following variables.

1. **int** x,y,z;
2. **char** a,b;
3. **float** f;

Valid Switch	Invalid Switch	Valid Case	Invalid Case
switch(x)	switch(f)	case 3;	case 2.5;
switch(x>y)	switch(x+2.5)	case 'a';	case x;
switch(a+b-2)		case 1+2;	case x+2;
switch(func(x, y))		case 'x'>'y';	case 1,2,3;


Start
Switch (expression)
Case 1
true
Statement for Case 1
break
false
Case 2
true
Statement for Case 2
break
false
Case 3
true
Statement for Case 3
break
false
Default
Statement for Case 4
break
Exit

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

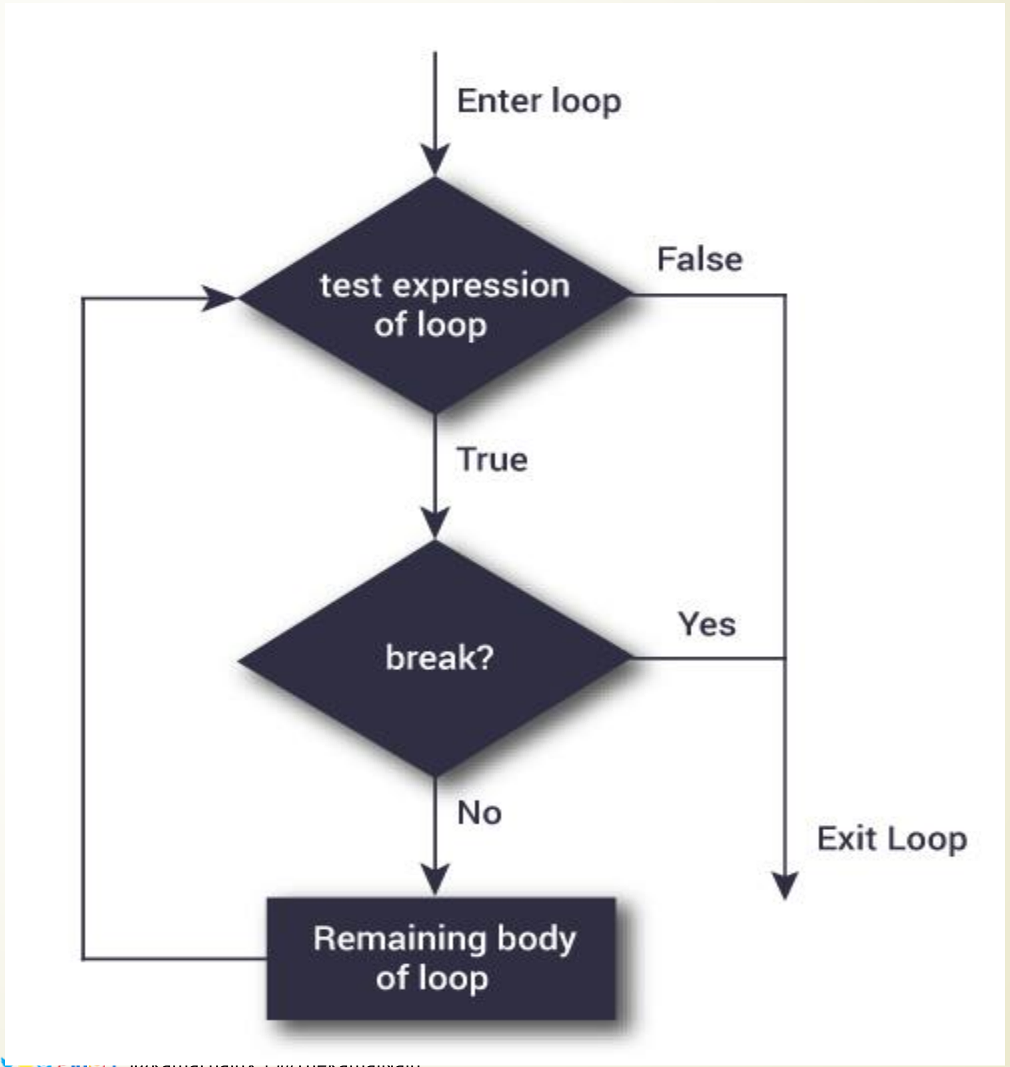
```
#include <stdio.h>
int main()
{
    int x = 10, y = 5;
    switch(x>y && x+y>0)
    {
        case 1:
        printf("hi");
        break;
        case 0:
        printf("bye");
        break;
        default:
        printf(" Hello bye ");
    }

}
```

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

Syntax:

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



```
#include<stdio.h>
#include<stdlib.h>
int main ()
{
    int i;
    for(i = 0; i<10; i++)
    {
        printf("%d ",i);
        if(i == 5)
        break;
    }
    printf("came outside of loop i = %d",i);

}
```

C break statement with the nested loop

```
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
printf("%d &d\n",i,j);
if(i==2 && j==2){
break;//will break loop of j only
}
}//end of for loop
return 0;
}
```

break statement with while loop

```
#include<stdio.h>
int main ()
{
    int i = 0;
    while(1)
    {
        printf("%d  ",i);
        i++;
        if(i == 10)
        break;
    }
    printf("came out of while loop");
}
```

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.

Syntax:

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

Continue statement example

```
#include<stdio.h>
int main(){
int i=1;//initializing a local variable
//starting a loop from 1 to 10
for(i=1;i<=10;i++){
if(i==5)
{
continue;
}
printf("%d \n",i);
}//end of for loop
return 0;
}
```

C continue statement with inner loop

```
#include<stdio.h>
int main(){
int i=1,j=1;//initializing a local variable
for(i=1;i<=3;i++){
for(j=1;j<=3;j++){
if(i==2 && j==2){
continue;//will continue loop of j only
}
printf("%d %d\n",i,j);
}
}//end of for loop
return 0;
}
```

C goto statement

The goto statement is known as jump statement in C. As the name suggests, goto is used to transfer the program control to a predefined label. The goto statment can be used to repeat some part of the code for a particular condition. It can also be used to break the multiple loops which can't be done by using a single break statement. However, using goto is avoided these days since it makes the program less readable and complicated

Syntax:

```
1.label:
2.//some part of the code;
3.goto label;
```

goto exampl

```
#include <stdio.h>
int main()
{
  int num,i=1;
  printf("Enter the number whose table you want to
print?");
  scanf("%d",&num);
  table:
  printf("%d x %d = %d\n",num,i,num*i);
  i++;
  if(i<=10)
  goto table;
}
```

```
#include <stdio.h>
int main()
{
  int i, j, k;
  for(i=0;i<10;i++)
  {
    for(j=0;j<5;j++)
    {
      for(k=0;k<3;k++)
      {
        printf("%d %d %d\n",i,j,k);
        if(j == 3)
        {
          goto out;
        }
      }
    }
  }
  out:
  printf("came out of the loop");
}
```

Type Casting in C

Typecasting allows us to convert one data type into other. In C language, we use cast operator for typecasting which is denoted by (type).

Syntax:

```
1.(type)value;
```

Note: It is always recommended to convert the lower value to higher for avoiding data loss.

Without Type Casting:

```
1.int f= 9/4;
2.printf("f : %d\n", f );//Output: 2
```

With Type Casting:

```
1.float f=(float) 9/4;
2.printf("f : %f\n", f );//Output: 2.250000
```

Type Casting example

Let's see a simple example to cast int value into the float.

```c
#include<stdio.h>
int main(){
float f= (float)9/4;
printf("f : %f\n", f );
return 0;
}
```

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*

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

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.

1. **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.}
```

If you want to return any value from the function, you need to use any data type such as int, long, char, etc. The return type depends on the value to be returned from the function.

```
1.int get(){
2.return 10;
3.}
```

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

```
#include<stdio.h>
void printName();
void main ()
{
    printf("Hello ");
    printName();
}
void printName()
{
    printf(“NIIT Noida");
}
```

```
#include<stdio.h>
void sum();
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);
}
```

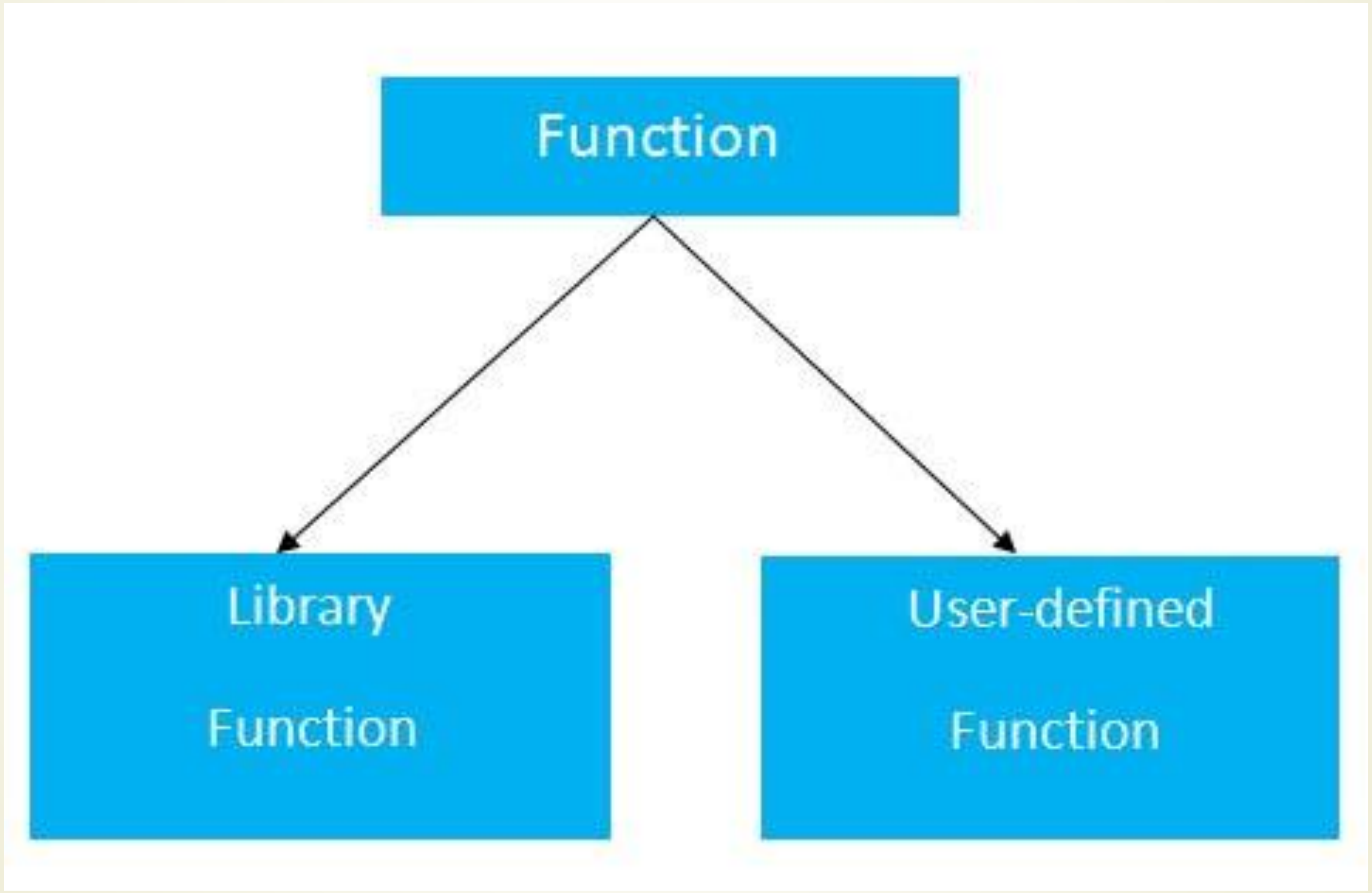
Example for Function without argument and with return value

```
#include<stdio.h>
int sum();
void main()
{
    int result;
    printf("\nGoing to calculate the sum of two numbers:");
    result = sum();
    printf("%d",result);
}
int sum()
{
    int a,b;
    printf("\nEnter two numbers");
    scanf("%d %d",&a,&b);
    return a+b;
}
```

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.

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.


Function
Library Function
User-defined Function

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:

```c
hello(){
printf("hello c");
}
```

Example with return value:

```
1.int get(){
2.return 10;
3.}
```

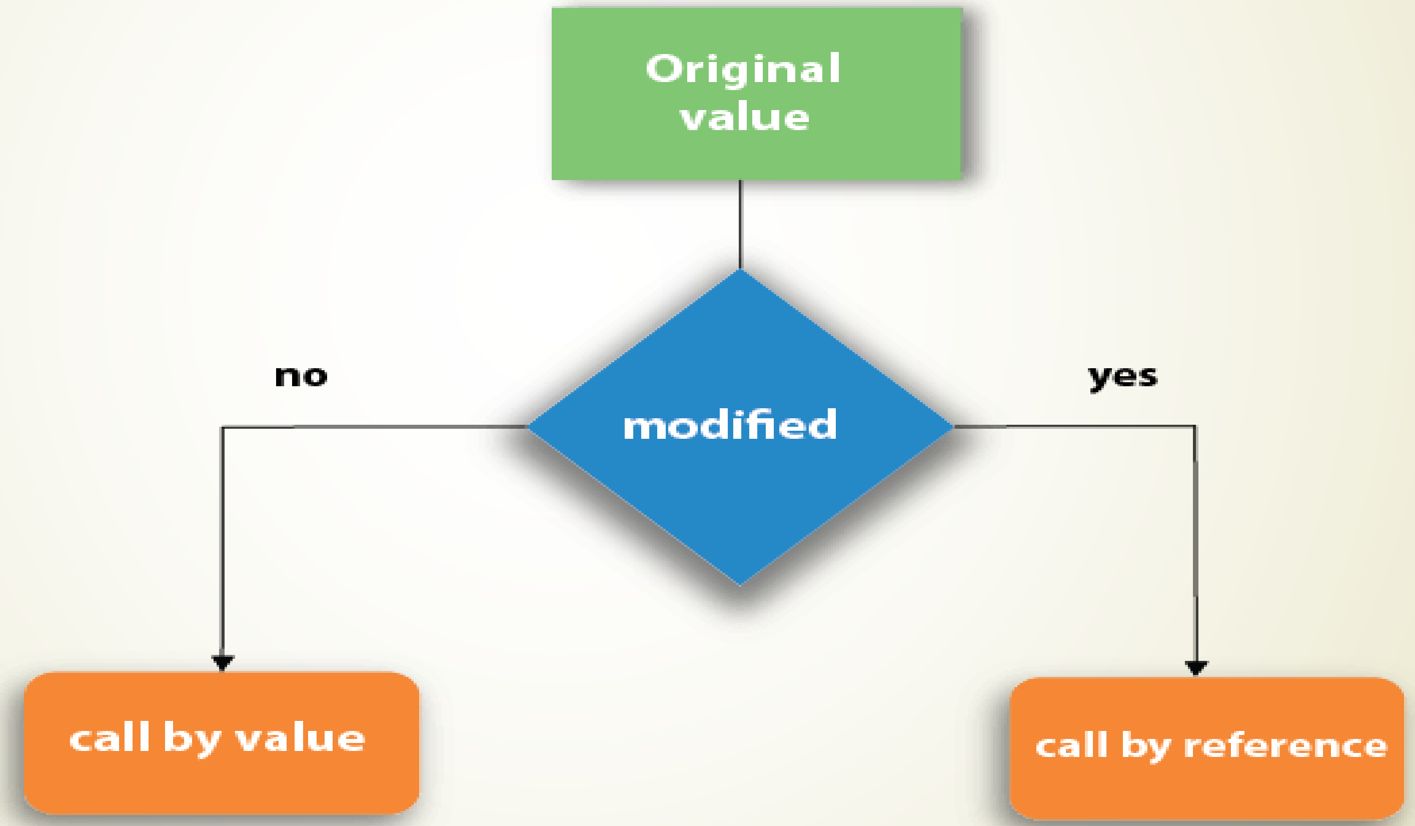
In the above example, we have to return 10 as a value, so the return type is int. If you want to return floating-point value (e.g., 10.2, 3.1, 54.5, etc), you need to use float as the return type of the method.

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

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

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*.



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.

```
#include<stdio.h>
void change(int num) {
    printf("Before adding value inside function num=%d \n",num);
    num=num+100;
    printf("After adding value inside function num=%d \n", num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(x);//passing value in function
    printf("After function call x=%d \n", x);
return 0;
}
```

```
#include <stdio.h>
void swap(int , int); //prototype of the function
int main()
{
    int a = 10;
    int b = 20;
    printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of
a and b in main
    swap(a,b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual
parameters do not change by changing the formal parameters in call by value, a = 10, b =
20
}
void swap (int a, int b)
{
    int temp;
    temp = a;
    a=b;
    b=temp;
    printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a =
20, b = 10
}
```

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.

```c
#include<stdio.h>
void change(int *num) {
    printf("Before adding value inside function num=%d \n",*num);
    (*num) += 100;
    printf("After adding value inside function num=%d \n", *num);
}
int main() {
    int x=100;
    printf("Before function call x=%d \n", x);
    change(&x);//passing reference in function
    printf("After function call x=%d \n", x);
return 0;
}
```

Difference between call by value and call by reference

No.	Call by value	Call by reference
1	A copy of the value is passed into the function	An address of value is passed into the function
2	Changes made inside the function is limited to the function only. The values of the actual parameters do not change by changing the formal parameters.	Changes made inside the function validate outside of the function also. The values of the actual parameters do change by changing the formal parameters.
3	Actual and formal arguments are created at the different memory location	Actual and formal arguments are created at the same memory location

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

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>
int fact (int);
int main()
{
    int n,f;
    printf("Enter the number whose factorial you want to calculate?");
    scanf("%d",&n);
    f = fact(n);
    printf("factorial = %d",f);
}
int fact(int n)
{
    if (n==0)
    {
        return 0;
    }
    else if ( n == 1)
    {
        return 1;
    }
    else
    {
        return n*fact(n-1);
    }  }
```

Recursive Function

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

```
#include<stdio.h>
void printFibonacci(int n){
    static int n1=0,n2=1,n3;
    if(n>0){
         n3 = n1 + n2;
         n1 = n2;
         n2 = n3;
         printf("%d ",n3);
         printFibonacci(n-1);
    }
}
int main(){
    int n;
    printf("Enter the number of elements: ");
    scanf("%d",&n);
    printf("Fibonacci Series: ");
    printf("%d %d ",0,1);
    printFibonacci(n-2);//n-2 because 2 numbers are already printed
  return 0;
 }
```

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.

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

Declaration of C Array

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

1.data_type array_name[array_size];

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

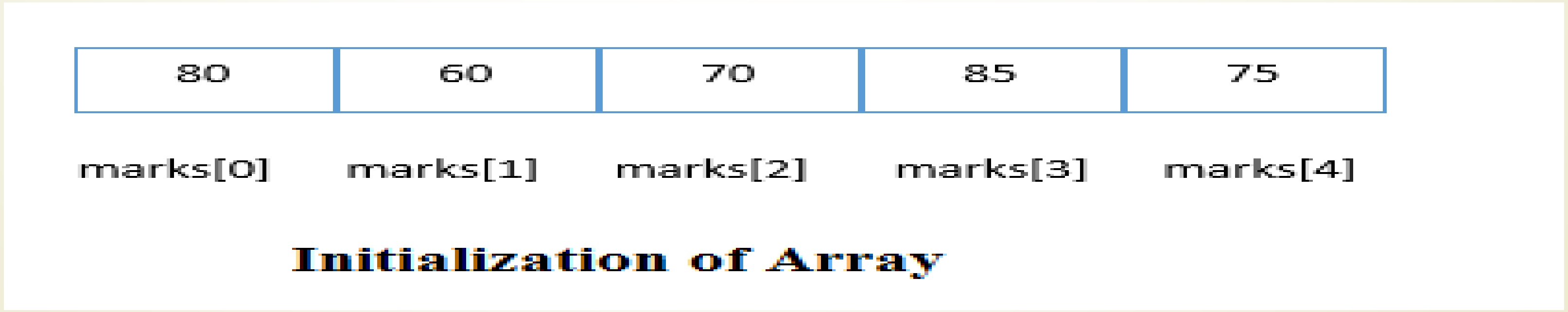
1.int marks[5];

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

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



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

C Array: Declaration with Initialization

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

```
1.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.

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

```
#include<stdio.h>
int main(){
int i=0;
int
marks[5]={20,30,40,50,60};//declaration
and initialization of array
 //traversal of array
for(i=0;i<5;i++){
printf("%d \n",marks[i]);
}
return 0;
}
```

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

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.

Declaration of two dimensional Array in C

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

```
1.data_type array_name[rows][columns];
```

Consider the following example.

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

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

Two-dimensional array example in C

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

C 2D array example:

```
#include <stdio.h>
int main ()
{
    int arr[3][3],i,j;
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("Enter a[%d][%d]: ",i,j);
            scanf("%d",&arr[i][j]);
        }
    }
    printf("\n printing the elements ....\n");
    for(i=0;i<3;i++)
    {
        printf("\n");
        for (j=0;j<3;j++)
        {
            printf("%d\t",arr[i][j]);
        }
    }
}
```

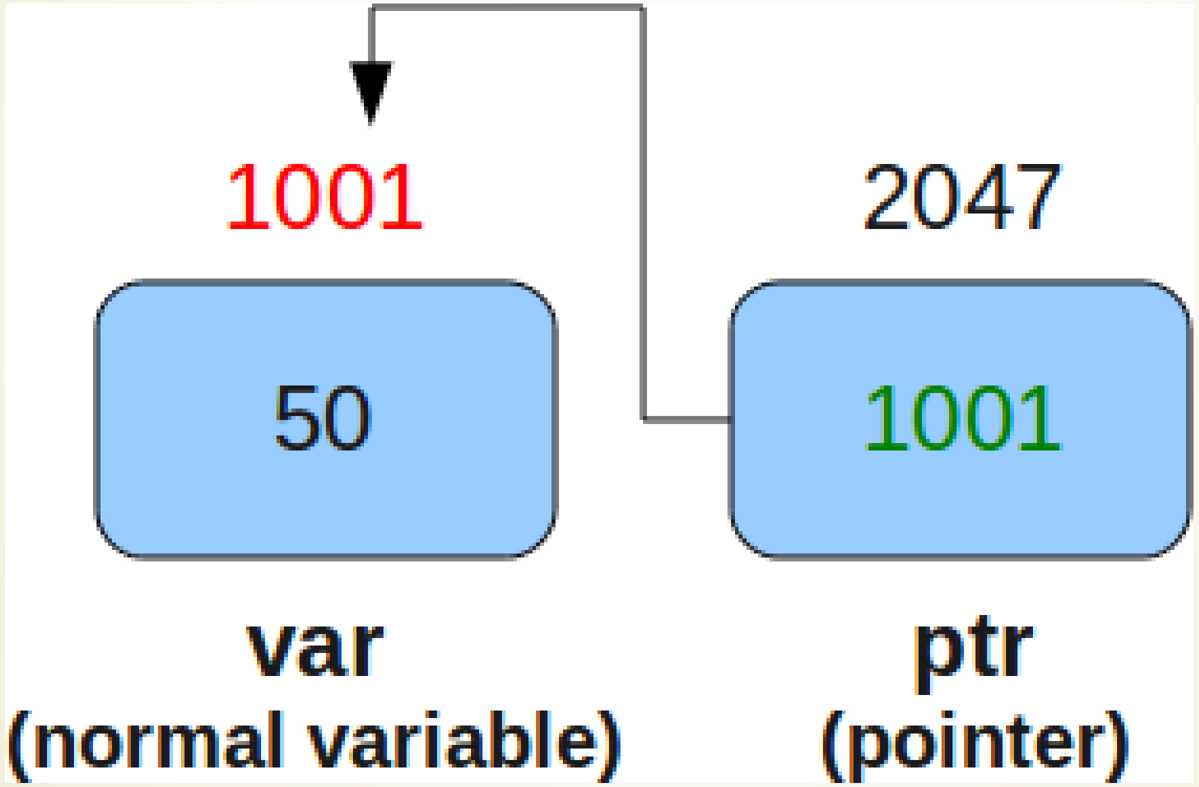
C Pointers

A **pointer** is a special type variable whose value is the address of another variable. Like any variable or constant, you must declare a pointer before you can work with it. The general form of a pointer variable declaration is

```
int    *ip;    // pointer to an integer
double *dp;    // pointer to a double
float  *fp;    // pointer to a float
char   *ch     // pointer to character
```

Pointer Example

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



```
#include<stdio.h>
int main(){
int number=50;
int *p;
p=&number;//stores the address of number variable
printf("Address of p variable is %x \n",p); // p contains the
address of the number therefore printing p gives the
address of number.
printf("Value of p variable is %d \n",*p); // As we know
that * is used to dereference a pointer therefore if we
print *p, we will get the value stored at the address
contained by p.
return 0;
}
```

```c
#include <stdio.h>

int main () {

   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Swap variaqbles :

```
#include<stdio.h>
int main()
{
        int x,y,temp,*p1,*p2;
        x=21;
        y=44;
        p1=&x;
        p2=&y;
        temp=*p1;
        *p1=*p2;
        *p2=temp;

        printf(" x is %d while y is %d",x,y);
}
```

Pointer to array

1. **int** arr[10];
2. **int** *p[10]=&arr; // Variable p of type pointer is pointing to t he address of an integer array arr.

Pointer to a function

1. **void** show (**int**);
2. **void**(*p)(**int**) = &display; // Pointer p is pointing to the addr ess of a function

Pointer to structure

```
1.struct st {
2.    int i;
3.    float f;
4.}ref;
5.struct st *p = &ref;
```

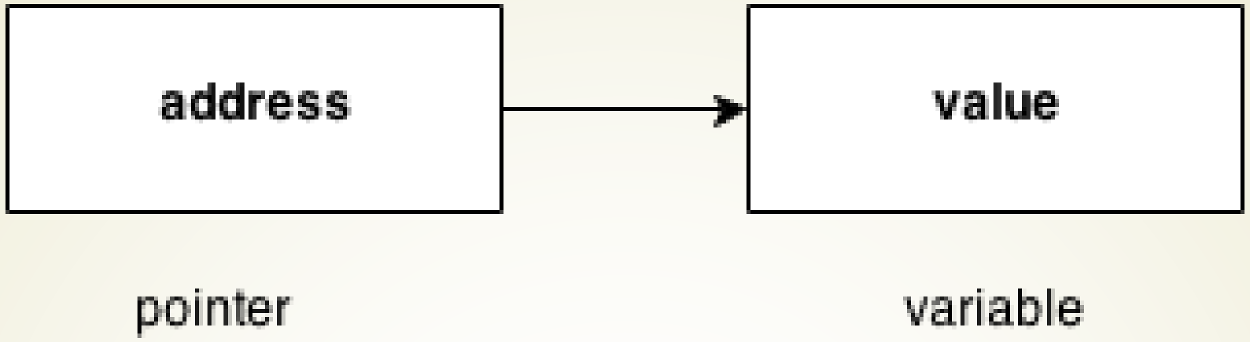
Advantage of pointer

1) Pointer **reduces the code** and **improves the performance**, it is used to retrieving strings, trees, etc. and used with arrays, structures, and functions.

2) We can **return multiple values from a function** using the pointer.

3) It makes you able to **access any memory location** in the computer's memory.

```
#include <stdio.h>

int main () {

   int  *ptr = NULL;

   printf("The value of ptr is : %x\n", ptr  );

   return 0;
}
```



```
#include<stdio.h>
int main(){
int number=50;
printf("value of number is %d, address of number is %u",number,&number);
return 0;
}
```

```
#include<stdio.h>
int main(){
int a=10,b=20,*p1=&a,*p2=&b;

printf("Before swap: *p1=%d *p2=%d",*p1,*p2);
*p1=*p1+*p2;
*p2=*p1-*p2;
*p1=*p1-*p2;
printf("\nAfter swap: *p1=%d *p2=%d",*p1,*p2);

return 0;
}
```

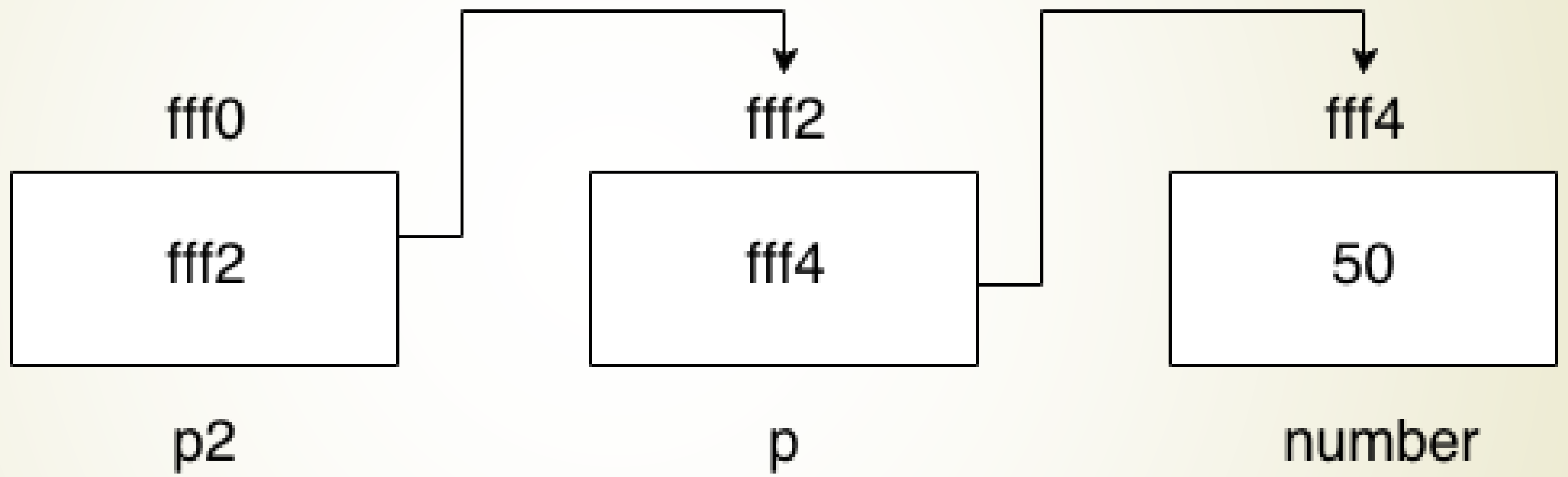
C Double Pointer (Pointer to Pointer)

As we know that, a pointer is used to store the address of a variable in C. Pointer reduces the access time of a variable. However, In C, we can also define a pointer to store the address of another pointer. Such pointer is known as a double pointer (pointer to pointer).

The syntax of declaring a double pointer is given below.

```
1.int **p; // pointer to a pointer which is pointing to an integer.
```

```
#include<stdio.h>
void main ()
{
    int a = 10;
    int *p;
    int **pp;
    p = &a; // pointer p is pointing to the address of a
    pp = &p; // pointer pp is a double pointer pointing to the address of
pointer p
    printf("address of a: %x\n",p); // Address of a will be printed
    printf("address of p: %x\n",pp); // Address of p will be printed
    printf("value stored at p: %d\n",*p); // value stoted at the address
contained by p i.e. 10 will be printed
    printf("value stored at pp: %d\n",**pp); // value stored at the
address contained by the pointer stoyred at pp
}
```


fff0
fff2
fff4
fff2
fff4
50
p2
p
number

```
#include<stdio.h>
int main(){
int number=50;
int *p;//pointer to int
int **p2;//pointer to pointer
p=&number;//stores the address of number variable
p2=&p;
printf("Address of number variable is %x
\n",&number);
printf("Address of p variable is %x \n",p);
printf("Value of *p variable is %d \n",*p);
printf("Address of p2 variable is %x \n",p2);
printf("Value of **p2 variable is %d \n",*p);
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.

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]={‘n’,’i’,’i’,’t’};

While declaring string, size is not mandatory. So we can write the above code as given below:

```
1.char ch[]={‘n’,’i’ 'i', 'n', 't', '\0'};
```

We can also define the **string by the string literal** in C language. For example:

```
1.char ch[]=“NIIT";
```

```
#include<stdio.h>
#include <string.h>
int main(){
 char ch[11]={n’,’I’ 'i', 'n', 't'};
  char ch2[11]="niit";

  printf("Char Array Value is: %s\n", ch);
  printf("String Literal Value is: %s\n", ch2);
 return 0;
}
```

Traversing String

Traversing the string is one of the most important aspects in any of the programming languages. We may need to manipulate a very large text which can be done by traversing the text.

Hence, there are two ways to traverse a string.

- By using the length of string
- By using the null character.

Using the length of string

```
#include<stdio.h>
void main ()
{
    char s[11] = "niit";
    int i = 0;
    int count = 0;
    while(i<11)
    {
        if(s[i]=='a' || s[i] == 'e' || s[i] == 'i' || s[i] == 'u' || s[i] == 'o')
        {
            count ++;
        }
        i++;
    }
    printf("The number of vowels %d",count);
}
```

Using the null character

```
#include<stdio.h>
void main ()
{
    char s[11] = “niit";
    int i = 0;
    int count = 0;
    while(s[i] != NULL)
    {
        if(s[i]=='a' | | s[i] == 'e' | | s[i] == 'i' | | s[i] == 'u' | | s[i] == 'o')
        {
            count ++;
        }
        i++;
    }
    printf("The number of vowels %d",count);
}
```

Accepting string as the input

Till now, we have used scanf to accept the input from the user.

```
#include<stdio.h>
void main ()
{
    char s[20];
    printf("Enter the string?");
    scanf("%s",s);
    printf("You entered %s",s);
}
```

It is clear from the output that, the above code will not work for space separated strings. To make this code working for the space separated strings, the minor changed required in the scanf function, i.e., instead of writing scanf("%s",s), we must write: scanf("%[^\n]s",s)

```
#include<stdio.h>
void main ()
{
    char s[20];
    printf("Enter the string?");
    scanf("%[^\n]s",s);
    printf("You entered %s",s);
}
```

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.

Declaration

1.char[] gets(**char**[]);

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

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.

Declaration

1.int puts(**char**[])

```
#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;
}
```

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(){
char ch[20]={‘n', 'i', ‘i', 't', '\0'};
   printf("Length of string is: %d",strlen(ch));
 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 ch[20]={‘n', 'i', 't', '\0'};
   char ch2[20];
   strcpy(ch2,ch);
   printf("Value of second string is: %s",ch2);
 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 ch[10]={'h', 'e', 'l', 'l', 'o', '\0'};
   char ch2[10]={'c', '\0'};
   strcat(ch,ch2);
   printf("Value of first string is: %s",ch);
 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 1st string: ");
  gets(str1);//reads string from console
  printf("Enter 2nd string: ");
  gets(str2);
  if(strcmp(str1,str2)==0)
      printf("Strings are equal");
  else
      printf("Strings are not equal");
 return 0;
}
```

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(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nReverse String is: %s",strrev(str));
 return 0;
}
```

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(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nLower String is: %s",strlwr(str));
 return 0;
}
```

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(){
  char str[20];
  printf("Enter string: ");
  gets(str);//reads string from console
  printf("String is: %s",str);
  printf("\nUpper String is: %s",strupr(str));
 return 0;
}
```

C String strstr()

The strstr() function returns pointer to the first occurrence of the matched string in the given string.

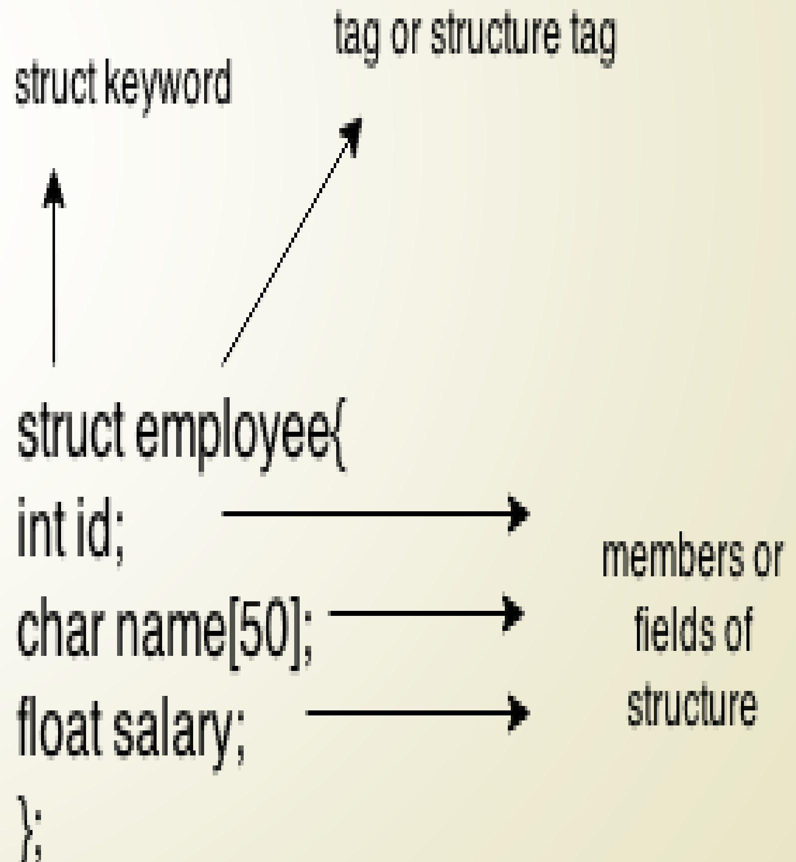
```
#include<stdio.h>
#include <string.h>
int main(){
  char str[100]="this is niit with c and c++";
  char *sub;
  sub=strstr(str,“niit");
  printf("\nSubstring is: %s",sub);
 return 0;
}
```

What is Structure

Structure in c is a user-defined data type that enables us to store the collection of different data types. Each element of a structure is called a member.

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

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .
    .
    data_type memeberN;
};
```



```
struct employee{
int id;
char name[50];
float salary;
};
```

Declaring structure variable

We can declare a variable for the structure so that we can access the member of the structure easily. There are two ways to declare structure variable:

1.By struct keyword within main() function

2.By declaring a variable at the time of defining the structure.

Accessing members of the structure

1.By . (member or dot operator)

1st way:

Let's see the example to declare the structure variable by struct keyword. It should be declared within the main function.

```
struct employee
{   int id;
    char name[50];
    float salary;
};
```

2nd way:

Let's see another way to declare variable at the time of defining the structure.

```
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2
```

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
}e1;  //declaring e1 variable for structure
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, “kamal nain");//copying string into char array
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
return 0;
}
```

```
#include<stdio.h>
#include <string.h>
struct employee
{   int id;
    char name[50];
    float salary;
}e1,e2;  //declaring e1 and e2 variables for structure
int main( )
{
   //store first employee information
   e1.id=101;
   strcpy(e1.name, “kamalnainx");//copying string into char array
   e1.salary=56000;
  //store second employee information
   e2.id=102;
   strcpy(e2.name, “thekamalnain");
   e2.salary=126000;
   //printing first employee information
   printf( "employee 1 id : %d\n", e1.id);
   printf( "employee 1 name : %s\n", e1.name);
   printf( "employee 1 salary : %f\n", e1.salary);
   //printing second employee information
   printf( "employee 2 id : %d\n", e2.id);
   printf( "employee 2 name : %s\n", e2.name);
   printf( "employee 2 salary : %f\n", e2.salary);
   return 0;
}
```

C Array of Structures

Why use an array of structures?

Consider a case, where we need to store the data of 5 students. We can store it by using the structure as given below.

```
#include<stdio.h>
struct student
{
    char name[20];
    int id;
    float marks;
};
void main()
{
    struct student s1,s2,s3;
    int dummy;
    printf("Enter the name, id, and marks of student 1 ");
    scanf("%s %d %f",s1.name,&s1.id,&s1.marks);
    scanf("%c",&dummy);
    printf("Enter the name, id, and marks of student 2 ");
    scanf("%s %d %f",s2.name,&s2.id,&s2.marks);
    scanf("%c",&dummy);
    printf("Enter the name, id, and marks of student 3 ");
    scanf("%s %d %f",s3.name,&s3.id,&s3.marks);
    scanf("%c",&dummy);
    printf("Printing the details....\n");
    printf("%s %d %f\n",s1.name,s1.id,s1.marks);
    printf("%s %d %f\n",s2.name,s2.id,s2.marks);
    printf("%s %d %f\n",s3.name,s3.id,s3.marks);
}
```

```
1#include<stdio.h>
#include <string.h>
struct student{
int rollno;
char name[10];
};
int main(){
int i;
struct student st[5];
printf("Enter Records of 5 students");
for(i=0;i<5;i++){
printf("\nEnter Rollno:");
scanf("%d",&st[i].rollno);
printf("\nEnter Name:");
scanf("%s",&st[i].name);
}
printf("\nStudent Information List:");
for(i=0;i<5;i++){
printf("\nRollno:%d, Name:%s",st[i].rollno,st[i].name);
}
  return 0;
}
```

Nested Structure in C

C provides us the feature of nesting one structure within another structure by using which, complex data types are created.

```
#include<stdio.h>
struct address
{
    char city[20];
    int pin;
    char phone[14];
};
struct employee
{
    char name[20];
    struct address add;
};
void main ()
{
    struct employee emp;
    printf("Enter employee information?\n");
    scanf("%s %s %d %s",emp.name,emp.add.city, &emp.add.pin, emp.add.phone);
    printf("Printing the employee information....\n");
    printf("name: %s\nCity: %s\nPincode: %d\nPhone:
%s",emp.name,emp.add.city,emp.add.pin,emp.add.phone);
}
```

C Nested Structure example

```
struct Employee
{
   int id;
   char name[20];
   struct Date
    {
      int dd;
      int mm;
      int yyyy;
    }doj;
}e1;
int main( )
{
   //storing employee information
   e1.id=101;
   strcpy(e1.name, “thekamalnain");//copying string into char array
   e1.doj.dd=10;
   e1.doj.mm=11;
   e1.doj.yyyy=2014;

   //printing first employee information
   printf( "employee id : %d\n", e1.id);
   printf( "employee name : %s\n", e1.name);
   printf( "employee date of joining (dd/mm/yyyy) : %d/%d/%d\n",
e1.doj.dd,e1.doj.mm,e1.doj.yyyy);
   return 0;
```

```
printf("Enter number of rows : ");
scanf("%d", &rows);
for(i=1; i<=rows; i++)
{
    /* Print trailing spaces */
    for(j=i; j<rows; j++)
    {
        printf(" ");
    }
    /* Print hollow pyramid */
    for(j=1; j<=(2*i-1); j++)
    {
        /*
         * Print star for last row (i==rows),
         * first column(j==1) and for
         * last column (j==(2*i-1)).
         */
        if(i==rows || j==1 || j==(2*i-1))
        {
            printf("*");
        }
        else
        {
            printf(" ");
        }
    }
```

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode