C Programming

Day2

by Manjiri Deshpande

Dr. D. Y. Patil Pratishthan's

Institute for Advanced Computing and Software Development

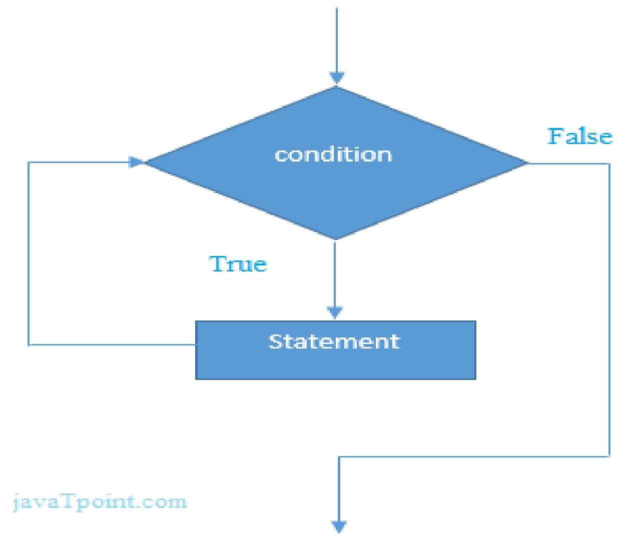
C Loops

- Loops are used when a task needs to be repeated a number of times
- For example
- Printing numbers from 1 to 100;
- Printing even numbers from 1 to 50
- Printing details of all bank customers
- Sequentially searching a record in a file
- Loops can be categorized as
- 1 Pre tested loop
- 2 Post tested loop

while loop in C

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

```
while(condition)
{
//code to be executed
}
```



Properties of while loop

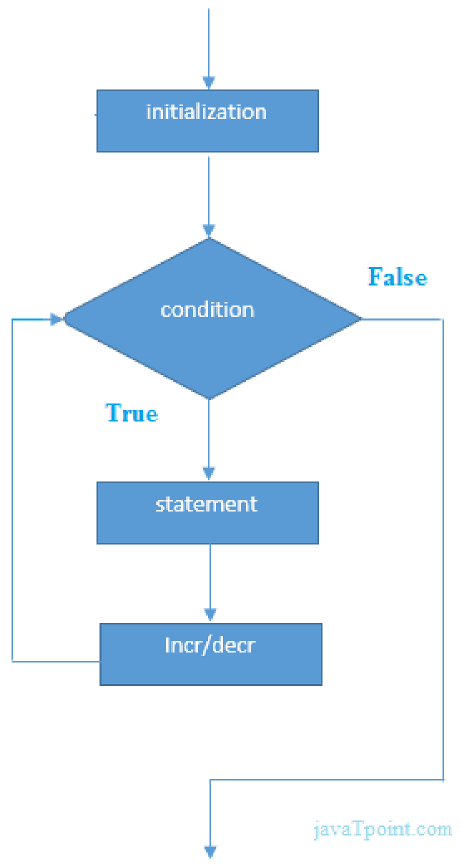
- A conditional expression is used to check the condition. The statements defined inside the while loop will repeatedly execute until the given condition fails.
- The condition will be true if it returns 0. The condition will be false if it returns any non-zero number.
- In while loop, the condition expression is compulsory.
- Running a while loop without a body is possible.
- We can have more than one conditional expression in while loop.
- If the loop body contains only one statement, then the braces are optional.

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.
- 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.

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

Flowchart:-


initialization
condition
False
True
statement
Incr/decr
javaTpoint.com

for loop in C

- Properties of Expression 1 i=1
- The expression represents the initialization of the loop variable.
- We can initialize more than one variable in Expression 1.
- Expression 1 is optional.
- In C, we can not declare the variables in Expression 1. However, It can be an exception in some compilers.

for loop in C

- Properties of Expression 2
- Expression 2 is a conditional expression. It checks for a specific condition to be satisfied. If it is not, the loop is terminated.
- Expression 2 can have more than one condition. However, the loop will iterate until the last condition becomes false. Other conditions will be treated as statements.
- Expression 2 is optional.
- Expression 2 can perform the task of expression 1 and expression 3. That is, we can initialize the variable as well as update the loop variable in expression 2 itself.
- We can pass zero or non-zero value in expression 2. However, in C, any non-zero value is true, and zero is false by default.

for loop in C

- Properties of Expression 3
- Expression 3 is used to update the loop variable.
- We can update more than one variable at the same time.
- Expression 3 is optional.
- Loop body
- The braces {} are used to define the scope of the loop. However, if the loop contains only one statement, then we don't need to use braces. A loop without a body is possible. The braces work as a block separator, i.e., the value variable declared inside for loop is valid

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.
- Syntax:-

```
do{
//code to be executed
}while(condition);
```

Nested Loops in C

- C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.
- Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define '**while**' loop inside a '**for**' loop.

Nested Loops in C

- C supports nesting of loops in C. **Nesting of loops** is the feature in C that allows the looping of statements inside another loop. Let's observe an example of nesting loops in C.
- Any number of loops can be defined inside another loop, i.e., there is no restriction for defining any number of loops. The nesting level can be defined at n times. You can define any type of loop inside another loop; for example, you can define '**while**' loop inside a '**for**' loop.

Syntax of Nested loop

```
Outer_loop
{
    Inner_loop
   {
         // inner loop statements.
   }
       // outer loop statements.
}
```

Nested for loop

```
for(int i=1;i<=n;i++)  // outer loop
  {
      for(int j=1;j<=10;j++)  // inner loop
      {
          printf("%d\t",(i*j)); // printing the value.
      }
      printf("\n");
```

Nested while loop

```
while(condition)
{
    while(condition)
    {
        // inner loop statements.
    }
// outer loop statements.
}
```

Nested do while loop

```
do
{
   do
  {
      // inner loop statements.
   }while(condition);
// outer loop statements.
}while(condition);
```

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

C continue statement

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

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

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 statement 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:

```
label:
//some part of the code;
goto label;
```

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.***

Three program elements involved in using a Function:-

- There are three aspects of a C function.
- ***Function declaration /*****Function prototype:-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.***

Syntax:-

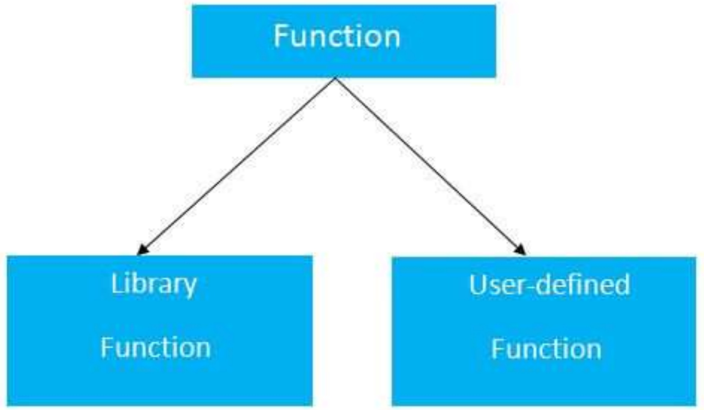
C function aspects	Syntax	
1	Function declaration	return_type function_name (argument list);
2	Function call	function_name (argument_list)
3	Function definition	return_type function_name (argument list) {function body;}

```
return_type function_name(data_type p
arameter...)
{
//code to be executed
}
```

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 c [illegible] gram 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.

Example without return value:

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

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

with return value:

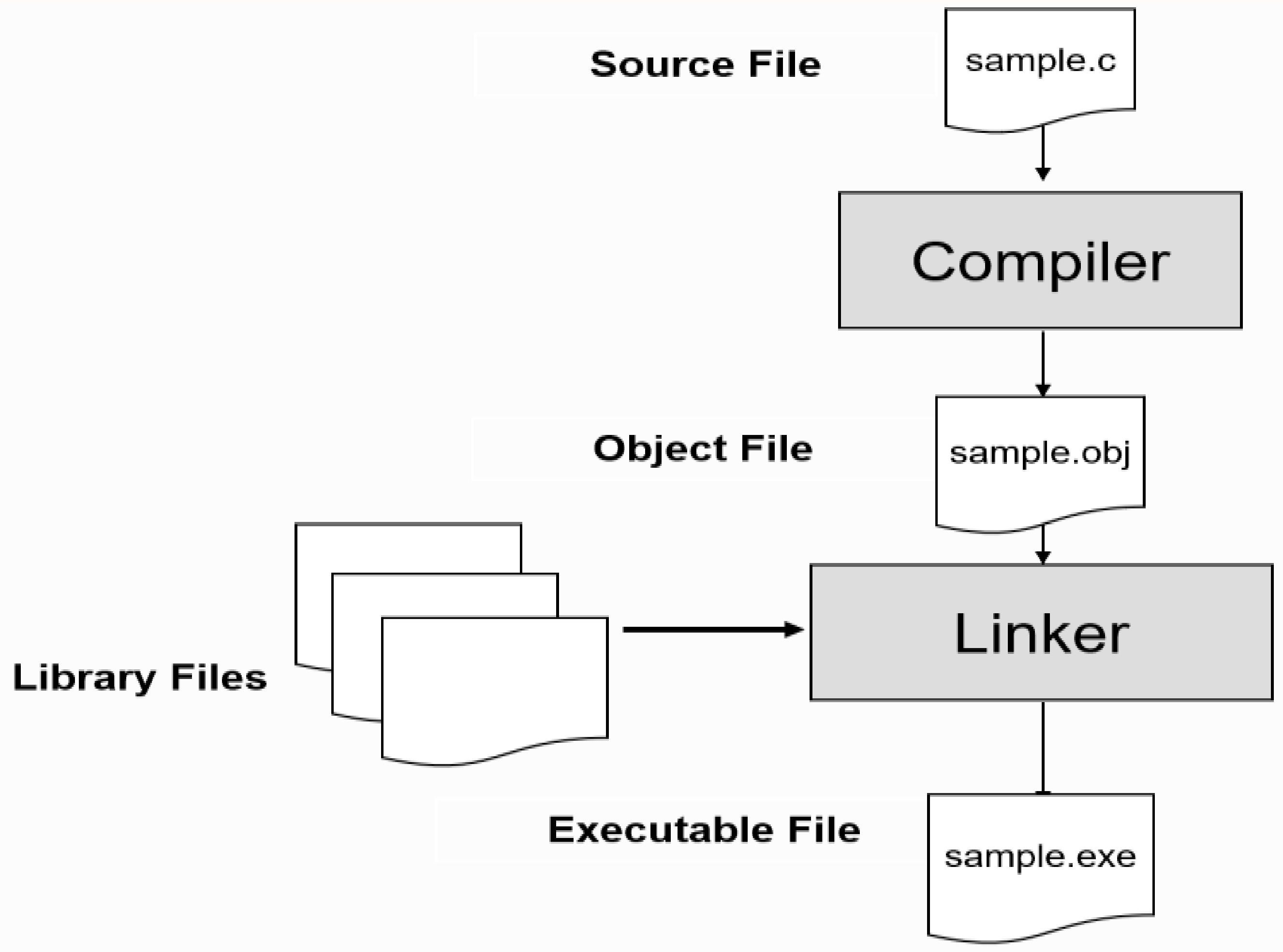
```
int get(){
return 10;
}

OR
float get(){
return 10.2;
}
```

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

Programming Environment



Types Of Error

- ***Syntax Error:-***

Statement missing

Unknown symbol

Logical Errors:-

Counter not incremented in the while loop

Wrong Expression

Runtime Errors:-

Division by zero

DMA failed

Linker Errors:-

Function definition missing

C Pointers

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

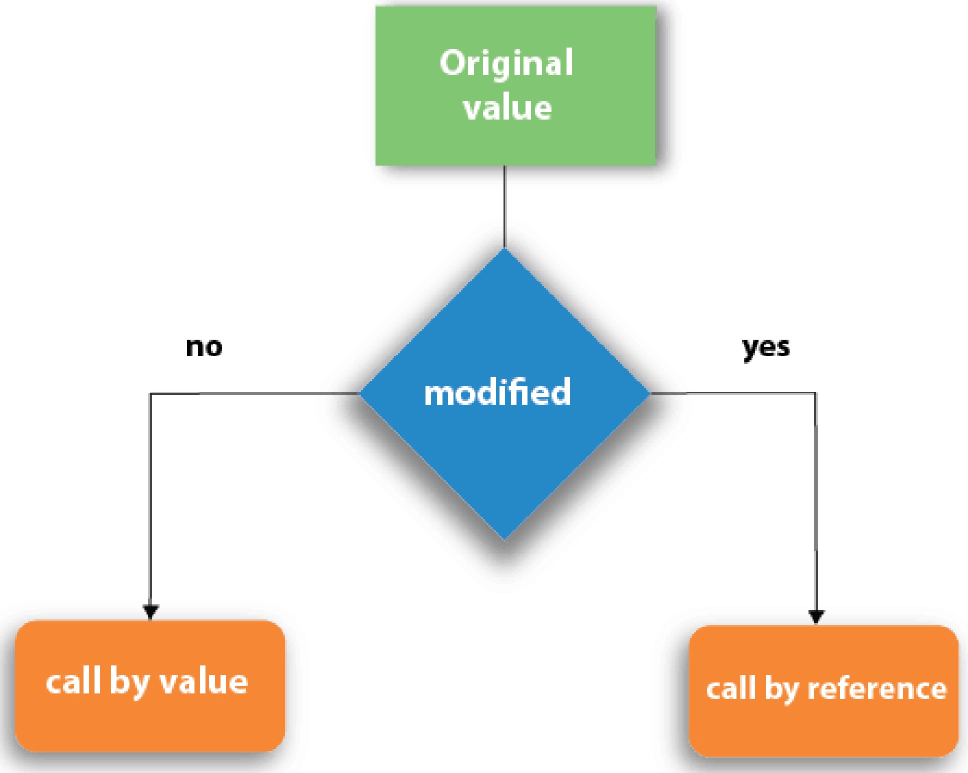
```
int n = 10;
int* p = &n; // Variable p of type pointer is pointing to the address of the varia
ble n of type integer
```

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

```
int *a;//pointer to int
char *c;//pointer to char
```

Call by value and Call by Address in C

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



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.

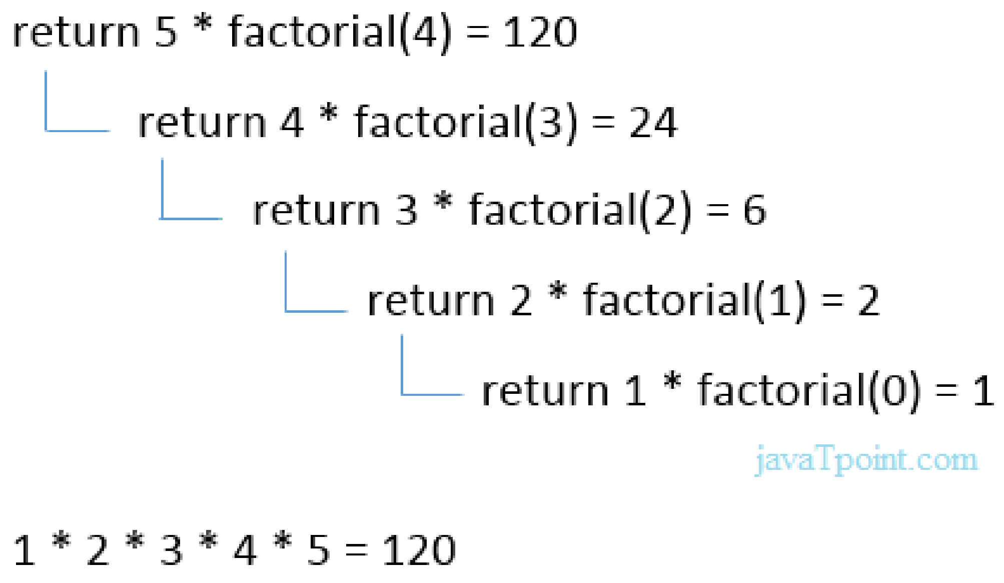
Call by Address in C

- In call by address, 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 address, 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.

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.

Recursion in C



```
return 5 * factorial(4) = 120
  └── return 4 * factorial(3) = 24
        └── return 3 * factorial(2) = 6
              └── return 2 * factorial(1) = 2
                    └── return 1 * factorial(0) = 1
```

1 * 2 * 3 * 4 * 5 = 120

Fig: Recursion

Storage Classes in C

- Storage classes in C are used to determine the lifetime, visibility, memory location, and initial value of a variable. There are four types of storage classes in C
- Automatic
- External
- Static
- Register

Automatic(auto);

- Automatic variables are allocated memory automatically at runtime.
- The visibility of the automatic variables is limited to the block in which they are defined.
- The scope of the automatic variables is limited to the block in which they are defined.The automatic variables are initialized to garbage by default.
- The memory assigned to automatic variables gets freed upon exiting from the block.
- The keyword used for defining automatic variables is auto.
- Every local variable is automatic in C by default.

Static

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

Register

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

Why Arrays?

- A variable can store only one value at a time.
- For example
- To store marks of 30 students in a class 30 variables would be required.
- Not a feasible solution if number of students increases .
- For example int m1,m2,m3,m4,m5---------------m30;
- Arrays provide the solution since one variable can be declared to hold many elements of similar type.

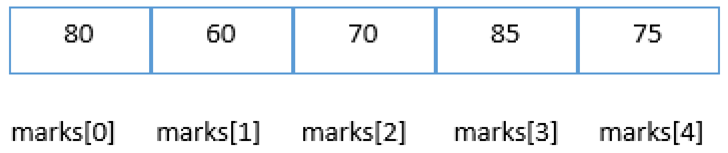
What is an Array?

- An array is a finite set of homogenous elements stored at contiguous memory locations.

- Declaration syntax:-
- data type arrayname[size of array];
- E.g.
- int arr[5];

Array Initialization

```
int marks[5]={80,60,70,85,75};
Or
int marks[]={80,60,70,85,75};
```



Initialization of Array

Properties of Array

- The array contains the following properties.
- 1. The elements are stored at contiguous memory locations.
- 2.marks[0] is the first elements of array and marks[4] is the last element.
- 3.The index of an array starts at 0.Therefore ,the index of the last element of an array will be 1 less than its size.
- 4.Every individual elements of an array can be accessed using the index.

Very Important about Array:-

- ***The name of an array is address of the array.It is also called base address.***
- ***Thus marks is the base address of the array or the address of the first element of the array.***
- ***The address of the first element can also be represented as (marks+0)***
- ***Next element will have address(marks+1) and so on…***

Representation of an Array

Address		Value	
marks	100	*marks	80
(marks+0)	100	*(marks+0)	80
(marks+1)	104	*(marks+1)	60
(marks+2)	108	*(marks+2)	70

Pointer Arithmetic

- Like A arithmetic operations are performed on ordinary variables, they can also be performed on pointers.
- But to perform these operations the pointers must point to elements of the same array, it is to be kept in mind that a pointer is a variable but an array name not a variable ,hence the assignment,increment,decrement operations can be done on pointers but not array names.
- The following operations are not allowed
- 1.addition of two pointers
- 2.multiplication of a pointer with number
- 3.dividing a pointer with a number.

Pointer Arithmetic

- Int arr[5]={10,20,30,40,50};
- Int *ptr1;
- Int *ptr2;
- Ptr1=arr;
- Ptr2=ptr1; //both pointer pointing to same element;
- Ptr1=ptr1+2//moves the pointer two location forward
- Ptr2=ptr2-2//moves the pointer two location backward
- If(ptr1==ptr2)//whether they pointing to same location

Size Of Array & Pointer

```
int arr[] = { 10, 20, 30, 40, 50, 60 };
 int* ptr = arr;

 // sizof(int) * (number of element in arr[]) is printed
 printf("Size of arr[] %ld\n", sizeof(arr));

 // sizeof a pointer is printed which is same for all
 // type of pointers (char *, void *, etc)
 printf("Size of ptr %ld", sizeof(ptr));
```

Passing Array to Function in C

```
Simple array
Void disp(int[]);
int arr[5] = {10,20,30,40,50};
----------
-----------

             disp(arr);

              -------------

void disp(int a[])
{
```

```
By using pointer
void disp(int *);
int arr[5] = {10,20,30,40,50};

----------------
------------------------

             disp(arr);



void disp(int *a)
```