

POINTERS

Introduction

Definition

Pointer is a variable that stores/hold address of another variable of same data type .It is also known as locator or indicator that points to an address of a value. A pointer is a derived data type in C:

Benefit of using pointers

- ❑ Pointers are more efficient in handling Array and Structure.
- ❑ Pointer allows references to function and thereby helps in passing of function as arguments to other function.
- ❑ It reduces length and the program execution time.
- ❑ It allows C to support dynamic memory management.

Declaration of Pointer

```
data_type* pointer_variable_name;
```

```
int* p;
```

Note: void type pointer works with all data types, but isn't used often.

Initialization of Pointer variable

Pointer Initialization is the process of assigning address of a variable to pointer variable.

Pointer variable contains address of variable of same data type

```
int a = 10 ;
```

```
int *ptr ; //pointer declaration
```

```
ptr = &a ; //pointer initialization
```

or,

```
int *ptr = &a ; //initialization and declaration together
```

Note:Pointer variable always points to same type of data.

```
float a;
```

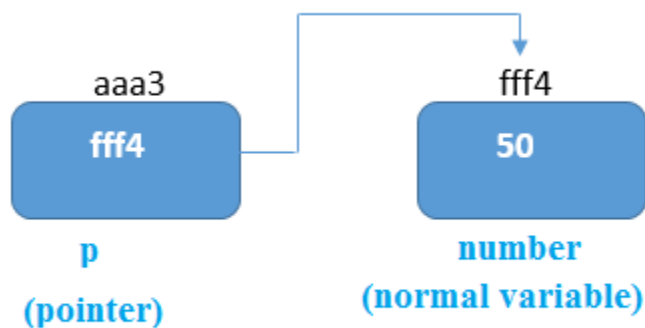
```
int *ptr;
```

```
ptr = &a; //ERROR, type mismatch
```

Above statement defines, p as pointer variable of type int.

Pointer Example

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



javatpoint.com

As you can see in the above figure, pointer variable stores the address of number variable, i.e., fff4. The value of number variable is 50. But the address of pointer variable p is aaa3.

By the help of * (**indirection operator**), we can print the value of pointer variable p. Let's see the pointer example as explained for the above figure.

```
#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 a 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;
}
```

Output

```
Address of number variable is fff4
Address of p variable is fff4
Value of a variable is 50
```

Reference operator (&) and Dereference operator (*)

& is called reference operator. It gives you the address of a variable. There is another operator that gets you the value from the address, it is called a dereference operator (*).

Symbols used in pointer

Operator	Name	Uses and Meaning of Operator
*	Asterisk	Declares a pointer in a program. Returns the referenced variable's value.
&	Ampersand	Returns a variable's address

Dereferencing of Pointer

Once a pointer has been assigned the address of a variable. To access the value of variable, pointer is dereferenced, using the indirection operator *.

```
int a,*p;
a = 10;
p = &a;
printf("%d",*p); //this will print the value of a.
printf("%d",*&a); //this will also print the value of a.
printf("%u",&a); //this will print the address of a.
printf("%u",p); //this will also print the address of a.
```

```
printf("%u",&p); //this will also print the address of p.
```

KEY POINTS TO REMEMBER ABOUT POINTERS IN C:

- ❑ Normal variable stores the value whereas pointer variable stores the address of the variable.
- ❑ The content of the C pointer always be a whole number i.e. address.
- ❑ Always C pointer is initialized to null, i.e. `int *p = null`.
- ❑ The value of null pointer is 0.
- ❑ `&` symbol is used to get the address of the variable.
- ❑ `*` symbol is used to get the value of the variable that the pointer is pointing to.
- ❑ If a pointer in C is assigned to NULL, it means it is pointing to nothing.
- ❑ The size of any pointer is 2 byte (for 16 bit compiler).

Example:

```
#include <stdio.h>
#include <conio.h>
void main(){
int number=50;
int *p;
clrscr();
p=&number;//stores the address of number variable
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);
getch();
}
```

Output

Address of number variable is fff4

Address of p variable is fff4

Value of p variable is 50

Example:

```
#include <stdio.h>
int main()
{
int *ptr, q;
q = 50;
/* address of q is assigned to ptr */
ptr = &q;
/* display q's value using ptr variable */
printf("%d", *ptr);
return 0;
}
```

Output

50

Example:

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

```

int main()
{
int var =10;
int *p;
p= &var;
printf ( "\n Address of var is: %u", &var);
printf ( "\n Address of var is: %u", p);
printf ( "\n Address of pointer p is: %u", &p);
/* Note I have used %u for p's value as it should be an address*/
printf( "\n Value of pointer p is: %u", p);
printf ( "\n Value of var is: %d", var);
printf ( "\n Value of var is: %d", *p);
printf ( "\n Value of var is: %d", *( &var));
}

```

Output:

```

Address of var is: 00XBBA77
Address of var is: 00XBBA77
Address of pointer p is: 77221111
Value of pointer p is: 00XBBA77
Value of var is: 10
Value of var is: 10
Value of var is: 10

```

NULL Pointer

A pointer that is not assigned any value but NULL is known as NULL pointer. If you don't have any address to be specified in the pointer at the time of declaration, you can assign NULL value.

Or

It is always a good practice to assign a NULL value to a pointer variable in case you do not have an exact address to be assigned. This is done at the time of variable declaration. A pointer that is assigned NULL is called a null pointer.

```
int *p=NULL;
```

Note: The NULL pointer is a constant with a value of zero defined in several standard libraries/ in most the libraries, the value of pointer is 0 (zero)

Example:

The value of ptr is 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 first pointer is used to store the address of a variable whereas the second pointer is used to store the address of the first pointer. Let's understand it by the diagram given below.



The syntax of declaring a double pointer is given below.

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

Consider the following example.

```
#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 stored at the address contained by p i.e. 10 will be printed
```

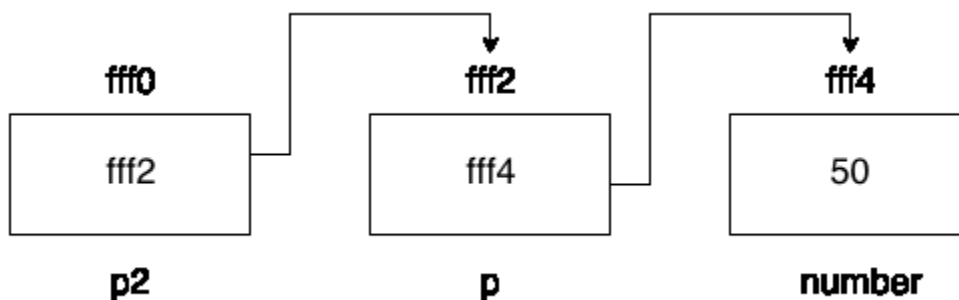
```
    printf("value stored at pp: %d\n",**pp); // value stored at the address contained by the pointer stored at pp
```

```
}
```

Output

```
address of a: d26a8734
address of p: d26a8738
value stored at p: 10
value stored at pp: 10
```

C double pointer example



As you can see in the above figure, p2 contains the address of p (fff2), and p contains the address of number variable (fff4)

```

#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;
}

```

Output

```

Address of number variable is fff4
Address of p variable is fff4
Value of *p variable is 50
Address of p2 variable is fff2
Value of **p variable is 50

```

Arrays and Pointers

Pointer

- Address of a variable in memory
- Allows us to indirectly access variables
- in other words, we can talk about its address rather than its value

Arrays:

Array is a group of elements that share a common name, and that are different from one another by their positions within the array. Under one name, a collection of elements of the same type is stored in memory.

- can be of any data type, e.g., integer, character, long integer, float, double, etc.
- even collection of arrays
- Arrays of structure, pointer , union etc. are also allowed

Advantages:

- For ease of access to any element of an array
- Passing a group of elements to a function

How arrays are stored in memory?

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
Nums	54	6	23	45	32	78	89
	1200	1204	1208	1212	1216	1220	1224

An array of one-dimensional elements consists of a series of individual variables of the array data type, each stored one after the other in the computer's memory. Each element has an address, just as all other variables do. An array's address is its first element, which corresponds to the first byte of memory it occupies.

Here we declare an array of seven integers like this:

```
int nums[7] = {54, 6, 23, 45, 32, 78, 89};
```

Let's assume that the first element of the array scores has the address 1200. This means that &nums[0] would be 1000. Since an int occupies 4 bytes, the address of the second element of the array, &nums[1] would be 1204, the address of the third element of the array, &nums[2] would be 1208, and so on.

What about a two-dimensional array declared like this?

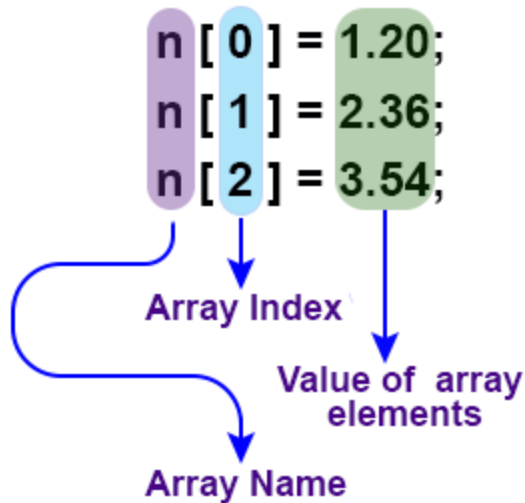
```
int nums [2][4] = {{1, 2, 3, 4}, {5, 6, 7, 8}};
```

Visualize the said two-dimensional array as a table of rows and columns:

column					
		[0]	[1]	[2]	[3]
row	[0]	1	2	3	4
	[1]	5	6	7	8
numbers					

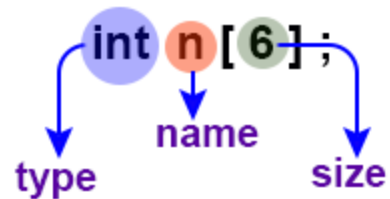
C Array: Syntax and Declaration

C syntax:



Declaration:

```
int n[6];
```

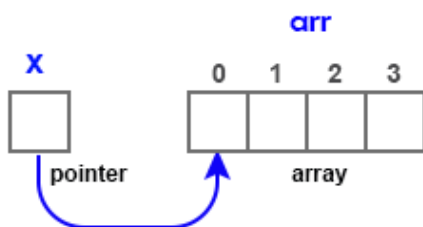


Sets aside memory for the array

Pointers and arrays in C:

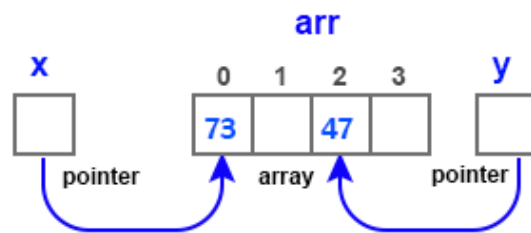
- A pointer can point at any array

```
int arr[4], *x, *y;  
x = &arr[0];
```



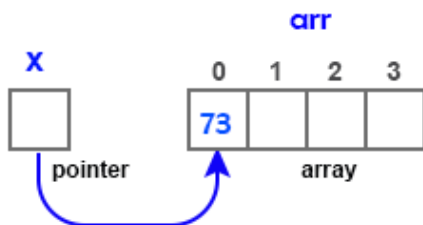
- Addition works

```
y = x + 2;  
*y = 47;
```



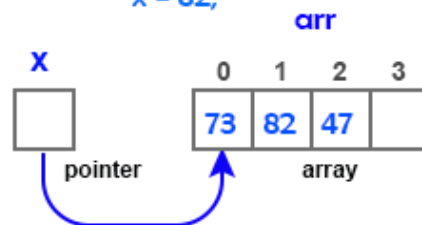
- Pointer arithmetic can be used to access array elements

```
*x = 73;
```



- So does subtraction

```
x = &arr[3];  
x = x - 2;  
*x = 82;
```



Relationship between Arrays and Pointers

Following 3 for loops are equivalent:

Code:

```
#include <stdio.h>  
int main(){  
int i,* ptr, sum =0;
```



```
int nums[5]={1,2,3,4,5};
for(ptr = nums; ptr <& nums[5];++ptr)
    sum +=* ptr;
printf("Sum = %d ", sum);// Sum = 15
}
```

Output:

```
Sum = 15
```

Code:

```
#include <stdio.h>
int main(){
int i,* ptr, sum =0;
int nums[5]={1,2,3,4,5};
for(i =0; i < 5;++i)
    sum +=*(nums+i);
printf("Sum = %d ", sum);// Sum = 15
}
```

Output:

```
Sum = 15
```

Code:

```
#include <stdio.h>

int main(){
int i,* ptr, sum =0;
int nums[5]={1,2,3,4,5};
    ptr = nums;
for(i =0; i < 5;++i)
    sum += ptr[i];
printf("Sum = %d ", sum);// Sum = 15
}
```

Output:

```
Sum = 15
```

Example: Arrays and Pointers

Following example print i) array elements using the array notation and ii) by dereferencing the array pointers:

Code:

```
#include <stdio.h>

int nums[] = {0, 5, 87, 32, 4, 5};
int *ptr;
int main(void)
{
int i;
```

```
ptr = &nums[0]; /* pointer to the first element of the array */
printf("Print array elements using the array notation\n");
printf("and by dereferencing the array pointers:\n");
for (i = 0; i < 6; i++)
{
    printf("\nnums[%d] = %d ", i , nums[i]);
    printf("\nptr + %d = %d\n",i, *(ptr + i));
} return 0;
}
```

Output:

Print array elements using the array notation
and by dereferencing the array pointers:

```
nums[0] = 0
ptr + 0 = 0

nums[1] = 5
ptr + 1 = 5

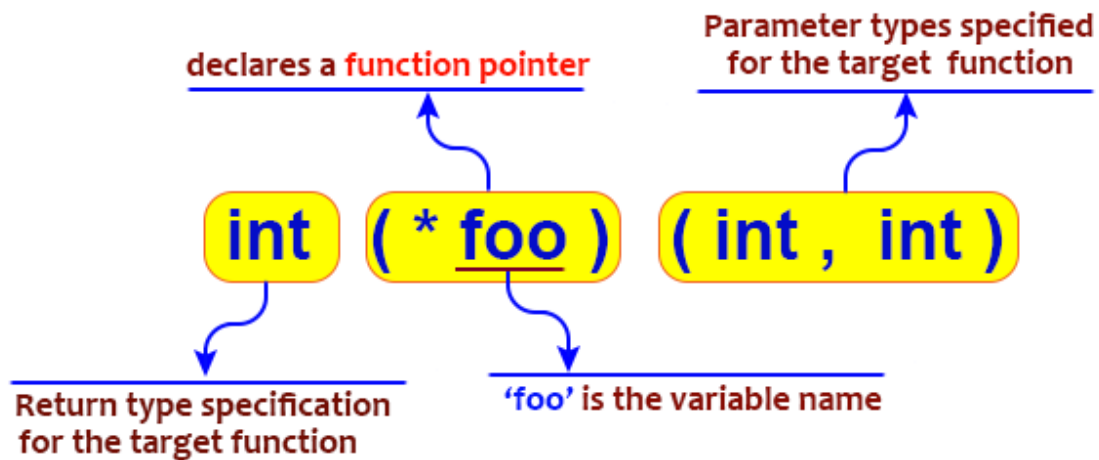
nums[2] = 87
ptr + 2 = 87

nums[3] = 32
ptr + 3 = 32
nums[4] = 4
ptr + 4 = 4

nums[5] = 5
ptr + 5 = 5
```

Pointers and functions

The C language makes extensive use of pointers, as we have seen. They can be used to provide indirect references to primitive types, to create dynamically sized arrays, to create instances of structs on demand, and to manipulate string data, among other things. Pointers can also be used to create references to functions. In other words, a function pointer is a variable that contains the address of a function.



Functions as pointers

- Memory stores the function code
- The address or start of a function is referred to as a "function pointer"
- Since function pointers do not allocate or deallocate memory, they are "different" from other pointers
- It is possible to pass function pointers as arguments to other functions or as returns from other functions

Why use function pointers?

- Efficiency
- Elegance
- Runtime binding

Function pointer declarations

Unlike a function declaration, a function pointer declaration wraps the function name in parentheses and precedes it with an asterisk. Here is an example:

- `int function(int x, int y); /* a function taking two int arguments and returning an int */`
- `int (*pointer)(int x, int y); /* a pointer to such a function */`

Pointers as Arguments

By passing a pointer into a function, the function may read or change memory outside of its activation record.

Example: Pointers and functions

In the following example, function passes the value of `p` and the address of `q`, which are used to initialize the variables `x` and `ptry` in the function test.

Code:

```
#include <stdio.h>

void test( int x, int *ptry ) {
    x = 200;
    *ptry = 200;
    return;
}

int main( void ) {
    int p = 10, q = 20;
    printf( "Initial value of p = %d and q = %d\n", p, q );
    test( p, &q );
    printf( "After passes to test function: p = %d, q = %d\n", p, q );
    return 0;
}
```

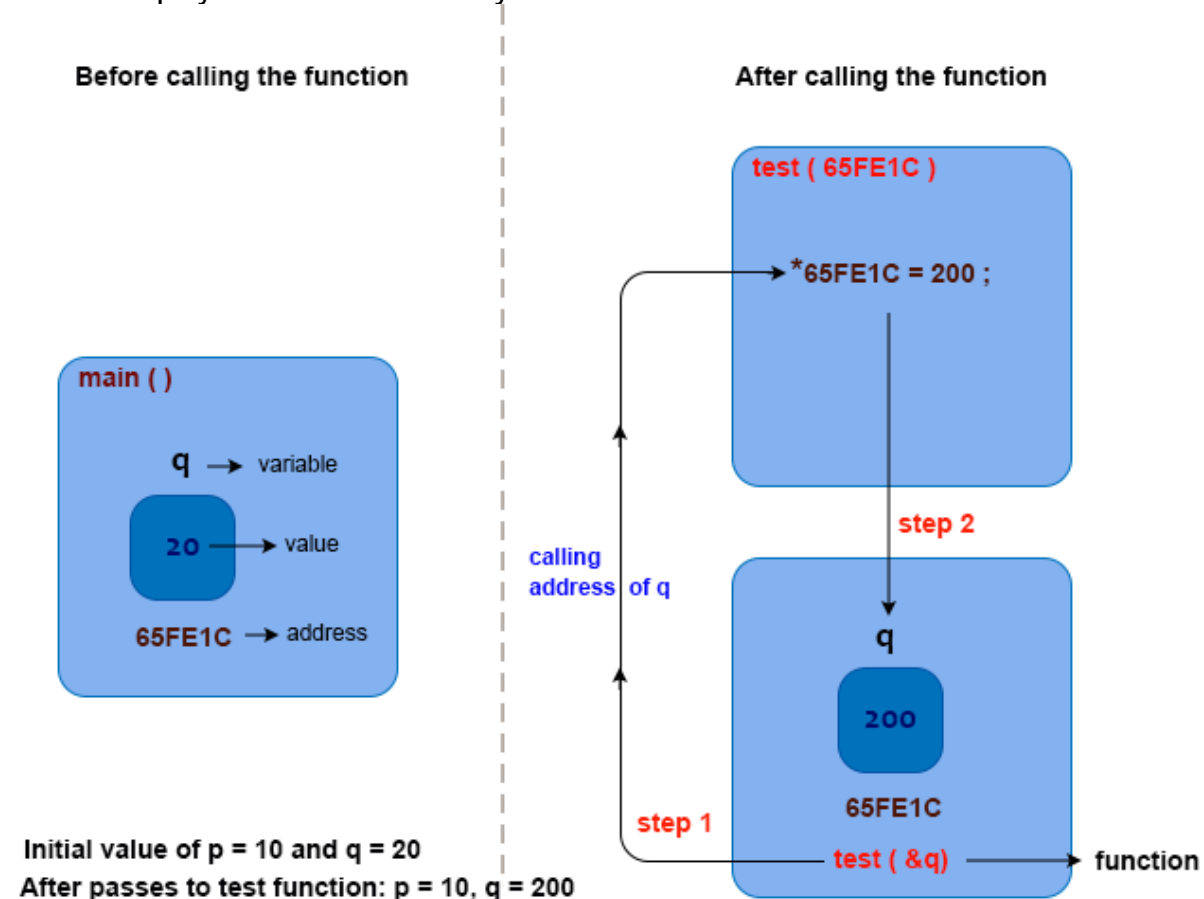
In the said code 'p' remains at 10 as there is no return value from the function test, but q change its value from 20 to 200 in the function test() as bptr holds the address of a variable q that is stored in main function. So when we change the v

Output:

Initial value of p = 10 and q = 20

After passes to test function: p = 10, q = 200

value of *ptry it will automatically reference in the main function.



Example: Swapping two values using pointers and functions

Code:

```
#include<stdio.h>
void swap(int*p,int*q){
int temp_val;
    temp_val =*p;
    *p =*q;
    *q = temp_val;
}

int main()
{
int x =45;
int y =65;
printf("Initial value of x = %d and y = %d\n", x, y );
swap(&x,&y);
printf("After swapping values of x = %d and y = %d\n", x, y );
return 0;
}
```

Output:

```
Initial value of x = 45 and y = 65
After swapping said values x = 65 and y = 45
```

Command Line Arguments in C

The arguments passed from command line are called command line arguments. These arguments are handled by main() function.

To support command line argument, you need to change the structure of main() function as given below.

```
int main(int argc, char *argv[] )
```

Here, **argc** counts the number of arguments. It counts the file name as the first argument.

The **argv[]** contains the total number of arguments. The first argument is the file name always.

Example

Let's see the example of command line arguments where we are passing one argument with file name.

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

```
void main(int argc, char *argv[] ) {
```

```
printf("Program name is: %s\n", argv[0]);

if(argc < 2){
    printf("No argument passed through command line.\n");
}
else{
    printf("First argument is: %s\n", argv[1]);
}
}
```

Run this program as follows in Linux:

```
./program hello
```

Run this program as follows in Windows from command line:

```
program.exe hello
```

Output:

```
Program name is: program
First argument is: hello
```

If you pass many arguments, it will print only one.

```
./program hello c how r u
```

Output:

```
Program name is: program
First argument is: hello
```

But if you pass many arguments within double quote, all arguments will be treated as a single argument only.

```
./program "hello c how r u"
```

Output:

```
Program name is: program
First argument is: hello c how r u
```

You can write your program to print all the arguments. In this program, we are printing only argv[1], that is why it is printing only one argument.

Passing Pointers to Functions in C

- Pointers in C
- Functions in C

Passing the pointers to the function means the memory location of the variables is passed to the parameters in the function, and then the operations are performed. The function definition accepts these addresses using pointers, addresses are stored using pointers.

Arguments Passing without pointer

When we pass arguments without pointers the changes made by the function would be done to the local variables of the function.

Below is the C program to pass arguments to function without a pointer:

```
// C program to swap two values
// without passing pointer to
// swap function.
#include <stdio.h>

void swap(int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
}

// Driver code
int main()
{
    int a = 10, b = 20;
    swap(a, b);
    printf("Values after swap function are: %d, %d",
           a, b);
    return 0;
}
```

Output

Values after swap function are: 10, 20

Arguments Passing with pointers

A pointer to a function is passed in this example. As an argument, a pointer is passed instead of a variable and its address is passed instead of its value. As a result, any change made by the function using the pointer is permanently stored at the address of the passed variable. In C, this is referred to as call by reference.

Below is the C program to pass arguments to function with pointers:

```
// C program to swap two values
// without passing pointer to
// swap function.
#include <stdio.h>
```

```

void swap(int* a, int* b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
}

// Driver code
int main()
{
    int a = 10, b = 20;
    printf("Values before swap function are: %d, %d\n",
           a, b);
    swap(&a, &b);
    printf("Values after swap function are: %d, %d",
           a, b);
    return 0;
}

```

Output

Values before swap function are: 10, 20

Values after swap function are: 20, 10

C Function Pointer

As we know that we can create a pointer of any data type such as int, char, float, we can also create a pointer pointing to a function. The code of a function always resides in memory, which means that the function has some address. We can get the address of memory by using the function pointer.

Let's see a simple example.

```

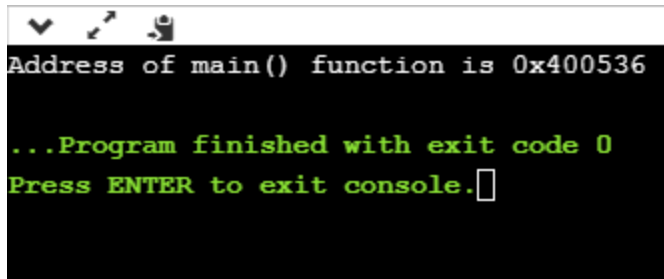
#include <stdio.h>

int main()
{
    printf("Address of main() function is %p",main);
    return 0;
}

```

The above code prints the address of **main()** function.

Output

A screenshot of a terminal window with a black background and green text. At the top, it shows the address of the main() function as 0x400536. Below that, it says "...Program finished with exit code 0" and "Press ENTER to exit console." with a cursor at the end.

```
Address of main() function is 0x400536

...Program finished with exit code 0
Press ENTER to exit console.█
```

In the above output, we observe that the `main()` function has some address. Therefore, we conclude that every function has some address.

Declaration of a function pointer

Till now, we have seen that the functions have addresses, so we can create pointers that can contain these addresses, and hence can point them.

Syntax of function pointer

return type (*ptr_name)(type1, type2...);

For example:

int (*ip) (**int**);

In the above declaration, `*ip` is a pointer that points to a function which returns an `int` value and accepts an integer value as an argument.

float (*fp) (**float**);

In the above declaration, `*fp` is a pointer that points to a function that returns a `float` value and accepts a `float` value as an argument.

We can observe that the declaration of a function is similar to the declaration of a function pointer except that the pointer is preceded by a `'*'`. So, in the above declaration, `fp` is declared as a function rather than a pointer.

Till now, we have learnt how to declare the function pointer. Our next step is to assign the address of a function to the function pointer.

float (*fp) (**int** , **int**); // Declaration of a function pointer.

float func(**int** , **int**); // Declaration of function.

fp = func; // Assigning address of func to the fp pointer.

In the above declaration, `'fp'` pointer contains the address of the `'func'` function.

Note: Declaration of a function is necessary before assigning the address of a function to the function pointer.

Calling a function through a function pointer

We already know how to call a function in the usual way. Now, we will see how to call a function using a function pointer.

Suppose we declare a function as given below:

```
float func(int , int);    // Declaration of a function.
```

Calling an above function using a usual way is given below:

```
result = func(a , b);    // Calling a function using usual ways.
```

Calling a function using a function pointer is given below:

```
result = (*fp)( a , b);    // Calling a function using function pointer.
```

Or

```
result = fp(a , b);        // Calling a function using function pointer, and indirection operator can be removed.
```

The effect of calling a function by its name or function pointer is the same. If we are using the function pointer, we can omit the indirection operator as we did in the second case. Still, we use the indirection operator as it makes it clear to the user that we are using a function pointer.

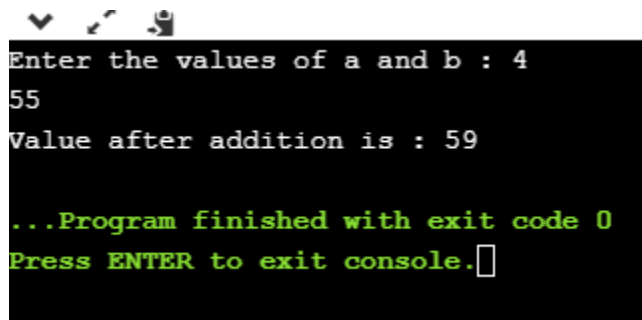
```
#include <stdio.h>
int add(int,int);
int main()
{
    int a,b;
    int (*ip)(int,int);
    int result;
    printf("Enter the values of a and b : ");
    scanf("%d %d",&a,&b);
    ip=add;
    result=(*ip)(a,b);
    printf("Value after addition is : %d",result);
    return 0;
```

```

}
int add(int a,int b)
{return a+b;}

```

OUTPUT:



```

Enter the values of a and b : 4
55
Value after addition is : 59

...Program finished with exit code 0
Press ENTER to exit console.

```

Dynamic Memory Allocation programs/examples in C programming language

Dynamic Memory Allocation in C Programming Language - C language provides features to manual management of memory, by using this feature we can manage memory at run time, whenever we require memory allocation or reallocation at run time by using Dynamic Memory Allocation functions we can create amount of required memory.

There are following functions:

- ❑ malloc - It is used to allocate specified number of bytes (memory blocks).
- ❑ calloc It is used to allocate specified number of bytes (memory blocks) and initialize all memory with 0.
- ❑ realloc It is used to reallocate the dynamically allocated memory to increase or decrease amount of the memory.
- ❑ free It is used to release dynamically allocated memory.

In this section, we will learn programming using these function, here bunch of programs which are using Dynamic Memory Allocation.

Dynamic Memory Allocation Examples using C programs

C program to create memory for int, char and float variable at run time.

In this program we will create memory for int, char and float variables at run time using malloc() function and before exiting the program we will release the memory allocated at run time by using free() function.

```

/*C program to create memory for int,
char and float variable at run time.*/
#include <stdio.h>
#include <stdlib.h>
int main()
{
int *iVar;

```

```
char *cVar;
float *fVar;
/*allocating memory dynamically*/
iVar=(int*)malloc(1*sizeof(int));
cVar=(char*)malloc(1*sizeof(char));
fVar=(float*)malloc(1*sizeof(float));
printf("Enter integer value: ");
scanf("%d",&iVar);
printf("Enter character value: ");
scanf(" %c",&cVar);
printf("Enter float value: ");
scanf("%f",&fVar);
printf("Inputted value are: %d, %c, %.2f\n",&iVar,&cVar,&fVar);
/*free allocated memory*/
free(iVar);
free(cVar);
free(fVar);
return 0;
}
```

Output:

Enter integer value: 100

Enter character value: x

Enter float value: 123.45

Inputted value are: 100, x, 123.45