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.

Consider the following example to define a pointer which stores the address of an integer.

1. int n = 50; //normal

   int*p = &n; // Variable p of type pointer is pointing to the address of the variable n of type integ er.

   Int a=50;

   Int *b=&a;

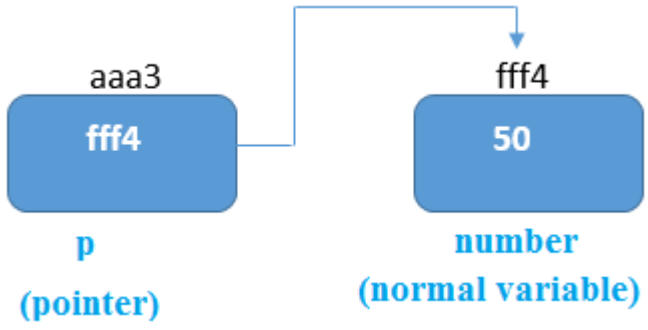
Declaring a pointer

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

1. int *a;//pointer to int
2. char *c;//pointer to char

Pointer Example

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



Int a=50; // address is (fff4)

Int b=a; //will store the value 50 in b

int *c=&a; // will store the address (fff4)

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.

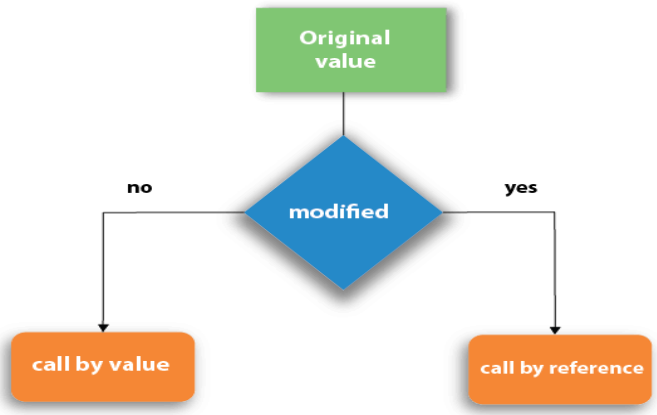
```
#include<stdio.h>
void main()
{
int number=50;    //fff4
int *p;                  //vvv2
p=&number;       // vvv2=fff4
//stores the address of number variable
printf("Address of p variable is %x \n",p);
// p contains the address of the number therefore printing p gives the address of number.
printf("Value 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.

}
```

Output

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

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.

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

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

Output

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

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

```
1. #include <stdio.h>
2. void swap(int , int); //prototype of the function
3. int main()
4. {
5.     int a = 10;
6.     int b = 20;
7.     printf("Before swapping the values in main a = %d, b = %d\n",a,b); // printing the value of a an
   d b in main
8.     swap(a,b);
9.     printf("After swapping values in main a = %d, b = %d\n",a,b); // The value of actual parameter
   s do not change by changing the formal parameters in call by value, a = 10, b = 20
10. }
11.
12. void swap (int a, int b)
13. {
14.     int temp;
15.     temp = a;
16.     a=b;
17.     b=temp;
18.     printf("After swapping values in function a = %d, b = %d\n",a,b); // Formal parameters, a = 20,
    b = 10
19. }
```

Output

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

Call by reference in C

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

Consider the following example for the call by reference.

```
1. #include<stdio.h>
2. void change(int *num) {
3.     printf("Before adding value inside function num=%d \n",*num);
4.     (*num) += 100;
5.     printf("After adding value inside function num=%d \n", *num);
6. }
```

```
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;
}
```

Output

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

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

```
#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 an
d b in main
    swap(&a,&b);
    printf("After swapping values in main a = %d, b = %d\n",a,b); // The values of actual paramete
rs do change in call by reference, 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
}
```

Output

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

Recursion

It is the process of repeating items in a self-similar way. In programming languages, if a program allows you to call a function inside the same function, then it is called a recursive call of the function.

```
void recursion() {
   recursion(); /* function calls itself */
}

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

The C programming language supports recursion, i.e., a function to call itself. But while using recursion, programmers need to be careful to define an exit condition from the function, otherwise it will go into an infinite loop.

Recursive functions are very useful to solve many mathematical problems, such as calculating the factorial of a number, generating Fibonacci series, etc.

```
#include<stdio.h>
#include<conio.h>
void count_to_ten(int cnt)
{
  Printf(" ==%d== ",cnt);
  If(cnt < 10)
  {
     count_to_ten(cnt+1);
  }
}
void main()
{
  Clrscr();
count_to_ten(0);
getch();
}
```

OutPut

==0==

==1==

==2==

==3==

..

..

==10==