Pointers in C++

We will discuss **pointers** here. Playing with pointers in C++ is really fun.

Now, let's again have a look at the declaration.

int a= 44;

As we know, the variable 'a' will take some space and will store 44 in it.

Now let's go to the undiscussed part. As we all know that when we declare 'a', it is given a memory location and the value of 'a' is stored in that memory location. In the world of programming, 'a' will also have an address. So, this address is the address of that memory location in which the value of 'a' is stored.

Address of 'a' is something like 0xffff377c. It will vary for every computer as per memory given to 'a' at that time.

Now coming to the **pointer**, a pointer points to some variable, that is, it stores the address of a variable. E.g.- if 'a' has an address 0xffff377c, then the pointer to 'a' will store a value 0xffff377c in it. So, if 'b' is pointer to 'a' and the value of 'a' is 10 and address is 0xffff377c, then 'b' will have a value 0xffff377c and its address will be different.

Address in C++ is represented as **&a** read as **address of a**. Remember that all the time when we were taking the value of 'a' using 'cin', we were taking an input from the user and storing it at the address of 'a', i.e. in a.

How to Use Pointers?

You must be enjoying programming in C++, and will do even more now. Till now, you have just seen what is a pointer. So, let's introduce pointers in our program.

```
int a = 44;
int *b; /* declaration of pointer b */
b = &a;
```

int *b - This statement should mean that '*b' is an integer but then what is the significance of '*' before 'b'? It means that b points to some integer ('b' is a pointer to some integer).

Or we can say that 'b' will store the address of some integer.

(/add_c

b = &a; - As said, 'b' will store the address of some integer because it is a pointer to an integer. In this declaration, it is storing the address of 'a'. Since 'b' is a pointer and '&a' represents address, so, by declaring 'b = &a;' we are storing the address of 'a' in 'b'.

So,
***b** is the value of the variable 'b' is pointing to. Here *b is 44. As stated earlier, **'int *b;'** means that '*b' is an integer, but '*' before means that b is a pointer. So, '*b' will be the value of the variable to which 'b' is pointing. Here, 'b' is pointing to 'a' therefore, 'b' will store the address of 'a' and '*b' will be the value of the integer to which 'b' is pointing i.e. 'a'

So, in short,
int a; - 'a' is an integer.
int *b; - 'b' is a pointer to an integer.
b = &a; - 'b' is now pointing to 'a'(value of 'b' is the address of 'a').
'*b' will now represent a (value of '*b' is the value of 'a').

Let's see an example of pointers.

```
#include <iostream>

using namespace std;

int main(){
        int a = 10;
        int *p;
        p = &a;
        cout << "p = " << p << endl;
        cout << "*p = " << *p << endl;
        cout << "&p = " << &p << endl;
        cout << "*&p = " << *&p << endl;
        return 0;
}
```

Output

</>
Value of address will vary every time we run our program because every time a new memory will be allocated.

As discussed earlier, 'p' is a pointer to 'a'. Since 'a' has a value of 10, so '*p' is 10. 'p' stores the address of a. So the output **p = 0xffff377c** implies that 0xffff377c is the address of 'a'. **'&p'** represents the address of 'p' which is 0xffff3778. Now, **'*&p'** is the value of '&p' and the value of '&p' is the address of 'a'. So, it is 0xffff377c.

Passing Pointers to Function

Let's first consider an example that will swap two numbers i.e., interchange the values of two numbers.

(/add_c

```
#include <iostream>

using namespace std;

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

int main(){
        int num1, num2;
        cout << "Enter first number" << endl;
        cin >> num1;
        cout << "Enter second number" << endl;
        cin >> num2;
        swap( &num1, &num2);
        cout << "First number = " << num1 << endl;
        cout << "Second number = " << num2 << endl;
        return 0;
}
```

Output

Swapping means to interchange the values.
void swap(int *a, int *b) - It means our function 'swap' is taking two pointers as argument. So, while calling this function, we will have to pass the address of two integers (**call by reference**).
int t; t = *a; We took any integer t and gave it a value '*a'.
***a = *b** - Now, *a is *b. This means that now the values of *a and *b will be equal to that of *b.
***b = t;** - Since 't' has an initial value of '*a', therefore, '*b' will also contain that initial value of '*a'. Thus, we have interchanged the values of the two variables.

Since we have done this swapping with pointers (we have targeted on address), so, this interchanged value will also reflect outside the function and the values of 'num1' and 'num2' will also get interchanged.

In the above example, we passed the address of the two variables (num1 and num2) to the swap function. The address of num1 is stored in 'a' pointer and that of num2 in 'b' pointer. In the swap function, we declared a third variable 't' and the values of 'a' and 'b' (and thus that of num1 and num2) gets swapped.

</>
In normal function call (**call by value**), the parameters of a function are xerox copies of the arguments passed to the function. It is like we are passing xerox copies.

So altering them won't affect the real values.

But in **call by referance**, we pass the address of variables to the function. Passing address is like passing original 'x' and 'y'.

Altering the parameters will alter the real values also.

In the swapping example also, we used call by reference in which we passed the address of num1 and num2 as the arguments to the function. The function parameters 'a' and 'b' point to the address of num1 and num2 respectively. So, any change in the parameters 'a' and 'b' changes the value of num1 and num2 also.

Let's see a normal function call (passing values and not address) and try to alter the value of the variable.

(/add_c

```
#include <iostream>

using namespace std;

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

int main(){
        int num1, num2;
        cout << "Enter first number" << endl;
        cin >> num1;
        cout << "Enter second number" << endl;
        cin >> num2;
        swap( num1, num2);
        cout << "First number = " << num1 << endl;
        cout << "Second number = " << num2 << endl;
        return 0;
}
```

Output

In the above example, we are trying to interchange the values of two variables. We passed num1 and num2 as arguments in the function swap. 'a' and 'b' are the copies of 'num1' and 'num2' respectively. In the swap function, the values of 'a' and 'b' got interchanged while the values of 'num1' and 'num2' remained unchanged.

So, you have seen that the numbers got swapped inside the function but outside it, there was no change. Because by passing values to functions, copies of the values got passed and not the real values. So, outside the function there was no effect on the variables.

If you have any confusion left, then just go through the above two codes again and you will understand it. If not, then you can always raise your doubts in the discussion section (/discussion/).

Pointers to Array

We also have pointers to array which we will see in the next topic Array.

Before moving to the next topic, practice a lot of problems on pointers so that you have a strong grip over it.

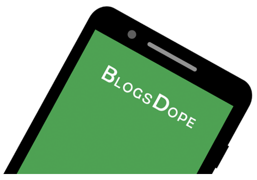
You are only good as the chances you take.

-Al Pacino

Prev(/cpp-scope-of-variables/) (/cpp-array/) Next

Like Follow

Download Our App.



(/add_c