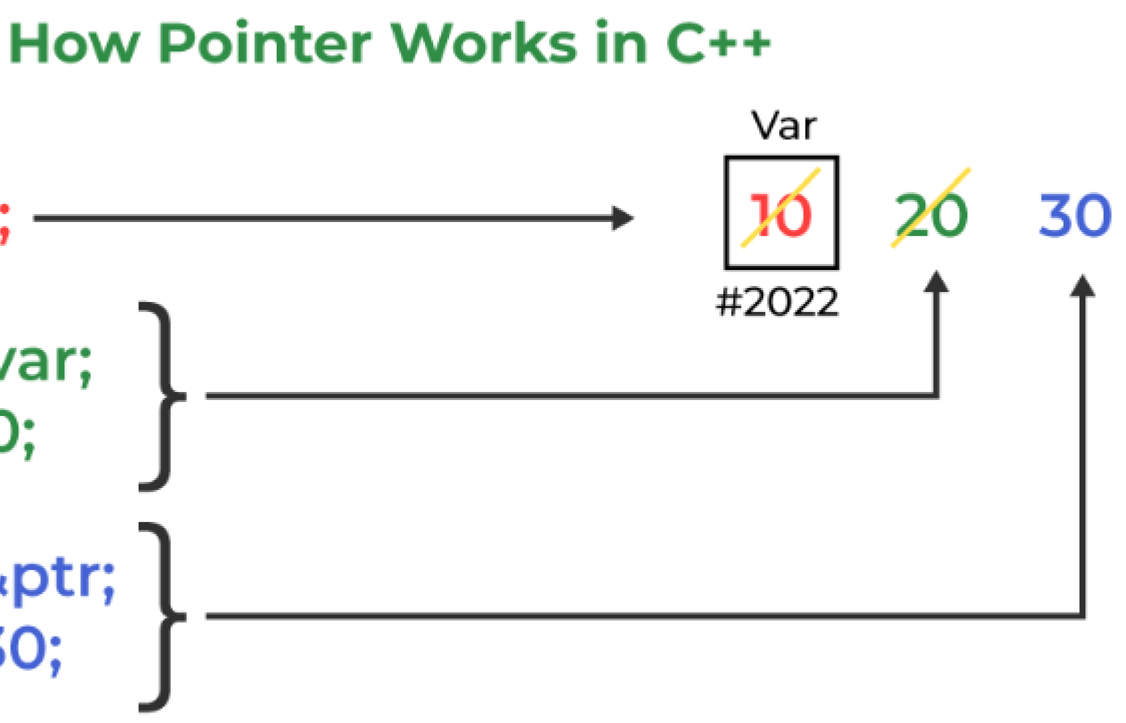
Pointers in C++:

Jaydeep Shah (radhey04ec@gmail.com)



Pointers are symbolic representations of addresses.The reason we associate data type with a pointer is that it knows how many bytes the data is stored in. When we increment a pointer, we increase the pointer by the size of the data type to which it point.

Call by value / Call by reference using poiter / Call by reference using reference variable :
There are different ways of calling the function and passing the arguments. When need to change original data or when size of data is large because of string or array or structure at that time call by reference either using pointer or reference variable method is useful.
There are two ways of using call by reference method:
1)Pass variable address during function call
2)Pass normal variable in function call but in function definition we use reference variable.

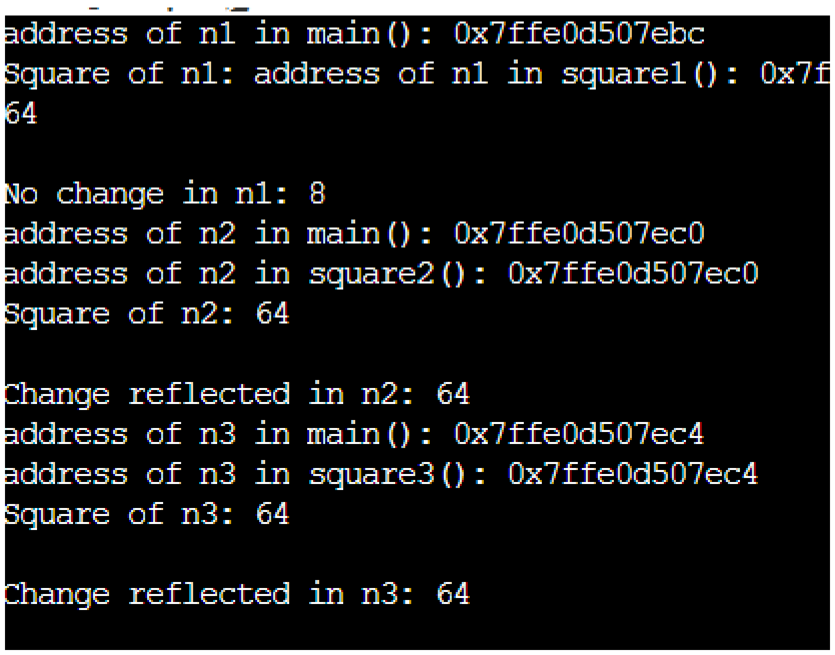
Both the method is almost same, just way of use is difference.

Example:

```
#include <bits/stdc++.h>
using namespace std;
```

```
// Pass-by-Value
int square1(int n)
{
        // Address of n in square1() is not the same as n1 in
        // main()
        cout << "address of n1 in square1(): " << &n << "\n";

        // clone modified inside the function
        n *= n;
        return n;
}
// Pass-by-Reference with Pointer Arguments
void square2(int* n)
{
        // Address of n in square2() is the same as n2 in main()
        cout << "address of n2 in square2(): " << n << "\n";

        // Explicit de-referencing to get the value pointed-to
        *n *= *n;


}
// Pass-by-Reference with Reference Arguments
void square3(int& n)
{
        // Address of n in square3() is the same as n3 in main()
        cout << "address of n3 in square3(): " << &n << "\n";

        // Implicit de-referencing (without '*')
        n *= n;


}
void geeks()
{
        // Call-by-Value
        int n1 = 8;
        cout << "address of n1 in main(): " << &n1 << "\n";
        cout << "Square of n1: " << square1(n1) << "\n";
        cout << endl << "No change in n1: " << n1 << "\n";

        // Call-by-Reference with Pointer Arguments
        int n2 = 8;
        cout << "address of n2 in main(): " << &n2 << "\n";
        square2(&n2);
        cout << "Square of n2: " << n2 << "\n";
        cout << endl << "Change reflected in n2: " << n2 << "\n";
        // Call-by-Reference with Reference Arguments
        int n3 = 8;
        cout << "address of n3 in main(): " << &n3 << "\n";
        square3(n3);
        cout << "Square of n3: " << n3 << "\n";
        cout << endl << "Change reflected in n3: " << n3 << "\n";
}

// Driver program
int main()
{
 geeks();
}
```

OUTPUT:



```
address of n1 in main(): 0x7ffe0d507ebc
Square of n1: address of n1 in square1(): 0x7f
64

No change in n1: 8
address of n2 in main(): 0x7ffe0d507ec0
address of n2 in square2(): 0x7ffe0d507ec0
Square of n2: 64

Change reflected in n2: 64
address of n3 in main(): 0x7ffe0d507ec4
address of n3 in square3(): 0x7ffe0d507ec4
Square of n3: 64

Change reflected in n3: 64
```

Void pointer:

Void Pointer

Important

This is a special type of pointer available in C++ which represents the absence of type. Void pointers are pointers that point to a value that has no type (and thus also an undetermined length and undetermined dereferencing properties). This means that void pointers have great flexibility as they can point to any data type.

>>Void pointer means point to value with no datatype, not means point to nothing.
>>These pointers cannot be directly dereferenced. They have to be first transformed into some other pointer type that points to a concrete data type before being dereferenced.

Example:
//void pointer example

```
#include <iostream>
using namespace std;


void increase_next(void *data, int size)
{
    //Is pass data character type ?
    if(size == sizeof(char))
    {
        char rec_data = *(char *)data;  //Type casting + Dereferencing
        rec_data++;
        cout << endl <<"Next value of character = " << rec_data << endl;

    }
    //Varaible integer type ?
    else if(size == sizeof(int))
    {
        int rec_data = *(int *)data;   //Type casting +Dereferencing
        rec_data++;
        cout << endl <<"Next value of Integer = " << rec_data << endl;

    }

}


int main()
{
char a = 'A';
int  b = 11;
increase_next(&a,sizeof(a));
increase_next(&b,sizeof(b));
return 0;
}
```

NULL POINTER:

Null pointer is pointer but point nowhere, Advantage is only it contains no invalid address.
Following are 2 methods to assign a pointer as NULL;

```
int *ptr1 = 0;

int *ptr2 = NULL;
```

In case, if we don’t have an address to be assigned to a pointer, then we can simply use NULL. NULL is used to represent that there is no valid memory address.

Wild pointer:

A pointer that has not been initialized to anything (not even **NULL**) is known as a **wild pointer**. The pointer may be initialized to a non-NULL garbage value that may not be a valid address.

```
dataType *pointerName;
```