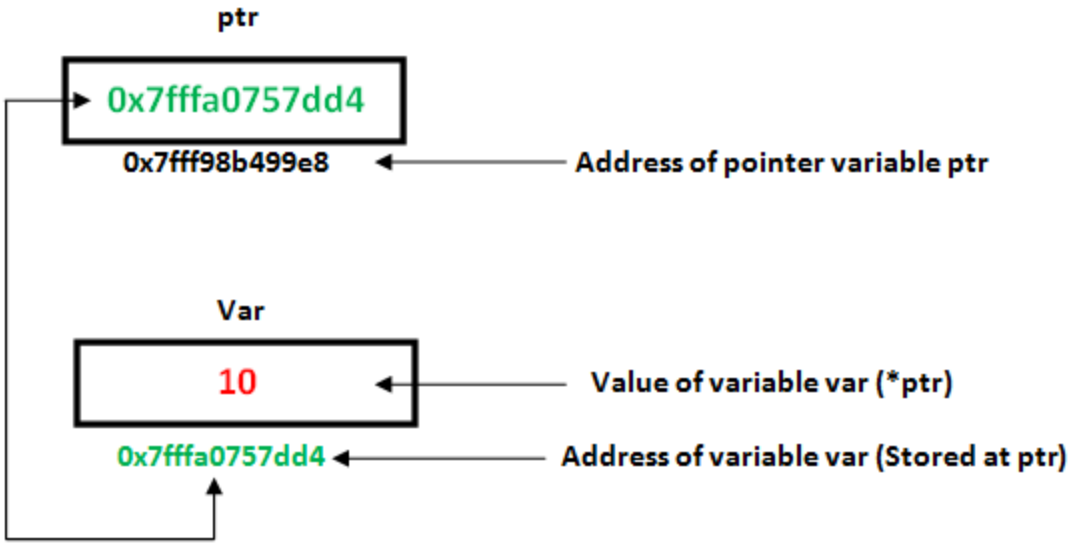
Pointers

Pointers are symbolic representation of addresses. It's general declaration in C++ has the format:

```
datatype *var_name;
int *ptr;   //ptr can point to an address which holds int data
```



How to use a pointer?

- Define a pointer variable
- Assigning the address of a variable to a pointer using unary operator (&) which returns the address of that variable.
- Accessing the value stored in the address using unary operator (*) which returns the value of the variable located at the address specified by its operand.

Example :

```
// C++ program to illustrate Pointers in C++

#include <bits/stdc++.h>
using namespace std;
void geeks()
{
    int var = 20;

        //declare pointer variable
    int *ptr;

    //note that data type of ptr and var must be same
    ptr = &var;

    // assign the address of a variable to a pointer
    cout << "Value at ptr = " << ptr << "\n";
    cout << "Value at var = " << var << "\n";
    cout << "Value at *ptr = " << *ptr << "\n";
}
//Driver program
int main()
{
    geeks();
}
```

Output:

```
Value at ptr = 0x7ffcb9e9ea4c

Value at var = 20

Value at *ptr = 20
```

Array Name as Pointers

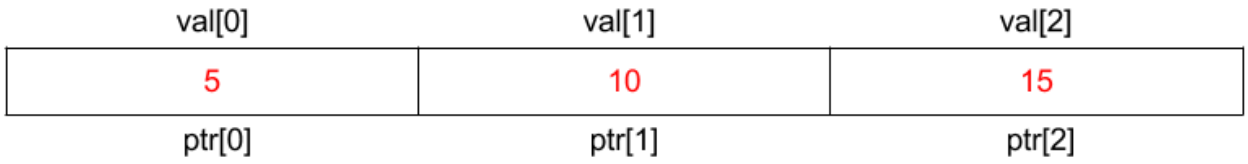
An array name contains the address of first element of the array which acts like constant pointer. It means, the address stored in array name can't be changed. For example, if we have an array named val then **val** and **&val[0]** can be used interchangeably.

```
// C program to illustrate call-by-methods
#include <stdio.h>
```

```
void geeks()
{
    //Declare an array
    int val[3] = { 5, 10, 20 };

    //declare pointer variable
    int *ptr;

    //Assign the address of val[0] to ptr
    // We can use ptr=&val[0];(both are same)
    ptr = val ;
    printf("Elements of the array are: ");
    printf("%d %d %d", ptr[0], ptr[1], ptr[2]);
}
//Driver program
int main()
{
    geeks();
}
```



Operation on pointers

A limited set of arithmetic operations can be performed on pointers which are:

- incremented (++)
- decremented (—)
- an integer may be added to a pointer (+ or +=)
- an integer may be subtracted from a pointer (– or -=)
- difference between two pointers (p1-p2)

Pointers to pointers

In C++, we can create a pointer to a pointer that in turn may point to data or other pointer. The syntax simply requires the unary operator (*) for each level of indirection while declaring the pointer.

```
char a;
char *b;
char ** c;
a = 'g';
b = &a;
c = &b;
```

Here b points to a char that stores ‘g’ and c points to the pointer b.

Void Pointers

This is a special type of pointer available in C++ which represents 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 it can point to any data type. There is payoff for this flexibility. 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.

Invalid pointers

A pointer should point to a valid address but not necessarily to valid elements (like for arrays). These are called invalid pointers. Uninitialized pointers are also invalid pointers.

```
int *ptr1;
int arr[10];
int *ptr2 = arr+20;
```