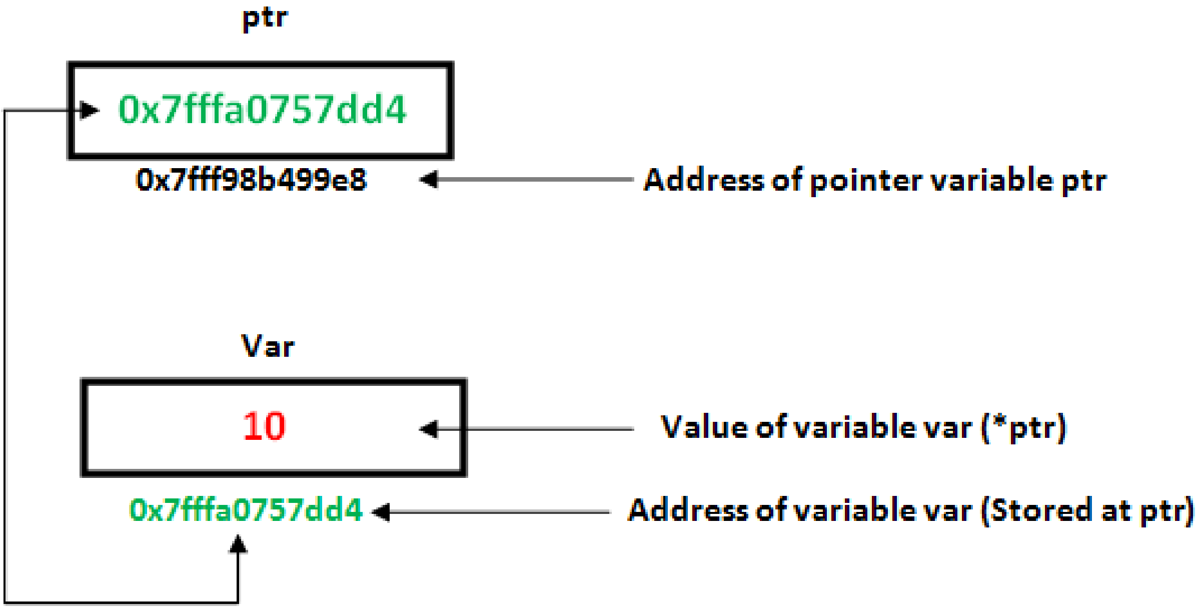
Pointers: a special type of variables that holds the **memory address** of another variables.


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (Stored at ptr)

In C++ pointers of **all types** have the same size that depends on the size of a memory address of the CPU

- For 64-bit CPU the size of a pointer is 64-bit
- For 32-bit CPU the size of a pointer is 32-bit

To declare a pointer: type * pointerName;

For example:-

```
int * iptr;
string * sptr;
//size is always 8 (bytes) on my machine
cout << sizeof(iptr);//output: 8
cout << sizeof(sptr);//output: 8
```

pointers takes the memory address of another variable. To access the variable's memory address we use the **&** symbol before the variable name:-

```
int x = 5;
//ptr now holds the memory address of x
int * ptr = &x;
```

each pointer can only hold a memory address of a variable of the **same type as the pointer.**

```
int x = 5;
string str = "hello world";

string * ptr;
ptr = &x; //illegal: x not of the same type
ptr = &str; //legal: str is of the same type
```

To access the variable pointed to by a pointer we do **dereferencing**:-

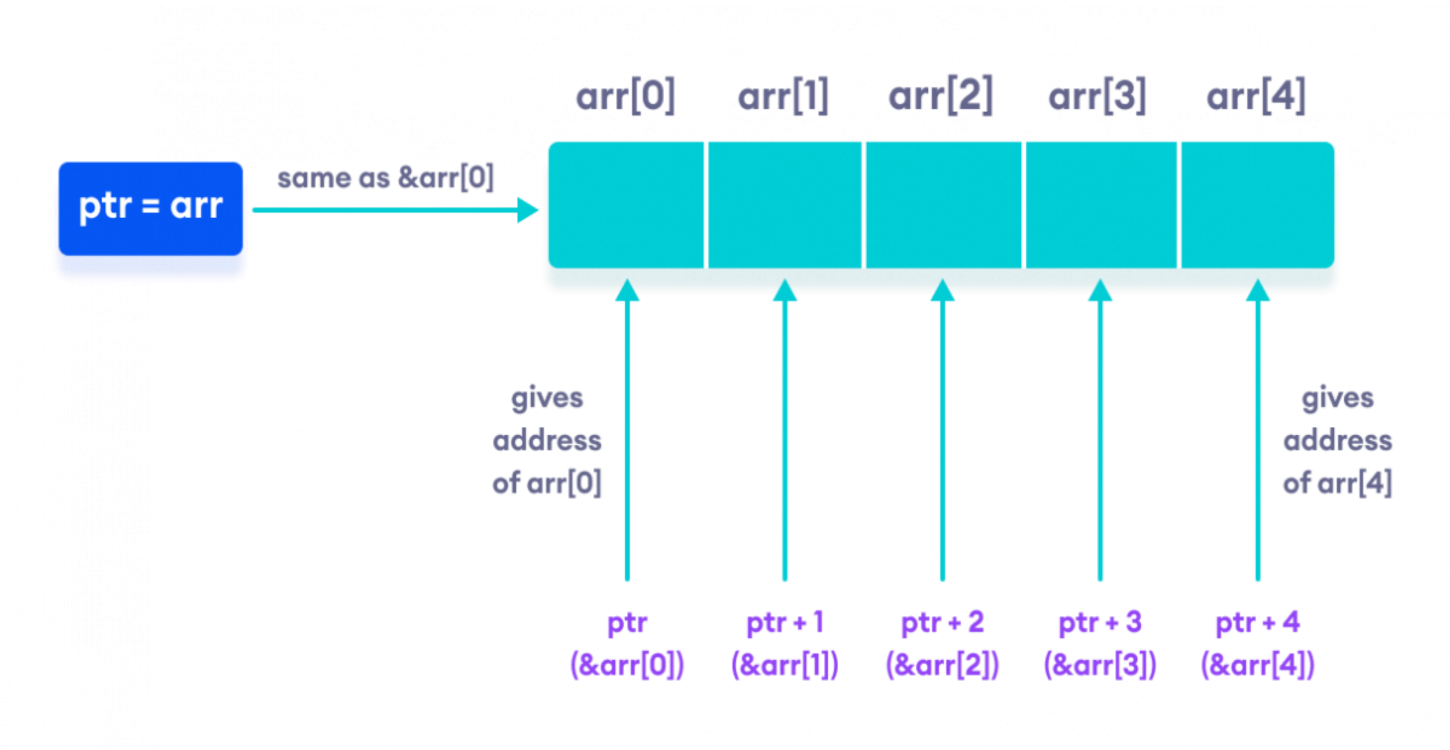
Dereferencing is done with ***** before the pointer's name:-

```
int x = 5;
int * ptr = &x;
(*ptr)++;//x is now 6
int y = (*ptr) + 3;
cout << y; //output: 9

string str = "hello ";
string * sp = &str;
(*sp).append("world");
cout << (*sp).length(); //ouptut:11
```

Arrays and Pointers:-

Arrays are actually a special type of pointers, They hold the memory address of the first element in the array.



For example:-

```
int arr[] = {1, 2, 3, 4, 5};
//arr holds the memory address of arr[0]
int * ptr = arr;
cout << *ptr;//ouput: 1
```

this also work:-

```
//added 2 to the memory address
//and dereferenced to get arr[2]
cout << *(ptr+2);//output: 3
cout << ptr[2];//same as above
```

pointers can be used like arrays and vise versa.

pointers and Functions:-

Pointers can be passed to and returned by functions,

they can also be passed instead of arrays:-

```
void print_arr(int arr[], int SIZE){
    for(int i = 0; i < SIZE; i++)
        cout << arr[i] << ' ';
    cout << endl;
}

void print_ptr(int * ptr, int SIZE){
    for(int i = 0; i < SIZE; i++)
        cout << *(ptr + i) << ' ';
    cout << endl;
}
```

Both of the functions above are equivalent as both arrays and pointers are the same thing:-

```
print_arr(arr, 5);
print_arr(ptr, 5);//this works
print_arr(&arr[0], 5);//this also works

print_ptr(ptr, 5);
print_ptr(arr, 5);//this works
print_ptr(&(*ptr), 5);//this also works
print_ptr(&ptr[0], 5);//also this works
```

all of the statements above are equivalent and each of them outputs: 1 2 3 4 5

We can use **const** on a pointer parameter to make it immutable just like passing const arrays.

For example:-

```
void add5(const int * ptr){
    *ptr = *ptr + 5;//this is illegal
}

void increment(const int * ptr, int SIZE){
    for(int i = 0; i < SIZE; i++)
        *(ptr+i)++;//this is illegal
}
```

Both functions above are illegal because they change the variables they point to.

Null Pointers:-

Any pointer that is not initialized at declaration may have the value **0.**

0 or **NULL** is a special value meaning this pointer doesn’t point to anything.

Trying to dereference a pointer with a value **NULL** will result in a **runtime error** and the program will **crash.**

```
int * ptr = NULL;
cout << *ptr;//this will cause a runtime error
```

when declaring a pointer without initializing, then it’s recommended to initialize it with **NULL**