**-------------------------------- Pointers ------------------------------------**

**Pointers:** Pointer is a variable that holds the address of another variable. How to create a pointer using:

Address of operator (&)

Dereference operator (\*)

**Why do we used the pointers?**

* To allocate new objects on the heap
* To pass functions to other functions
* To iterate over elements in arrays or other data structures.

**Advantage:**

* Provide direct access to memory
* Provide a way to return more than one value to the functions
* Reduces the storage space and time of the program
* Provides an alternate way to return array elements
* Allows us to perform dynamic memory allocation and deallocation.
* Build complex data structures like linked list, stack, queue, graph, tree etc
* Allows us to resize the dynamically allocated memory block
* Addresses of objects can be extracted using pointers

**Disadvantage:**

* Uninitialized pointers might cause segmentation fault.
* Dynamically allocated block needs to be freed explicitly.  Otherwise, it would lead to memory leak.
* Pointers are slower than normal variables
* If pointers are updated with incorrect values, it might lead to memory corruption.

**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:

int var = 10;

int\* ptr = &var;

cout << ptr << var << \*ptr;

ptr = address, var = value, \*ptr = value

**References and Pointers:**

There are 3 ways to pass arguments to a function:

1. call by value

2. call by reference with pointer argument

3. call by reference with reference argument

**Array Name as pointers**

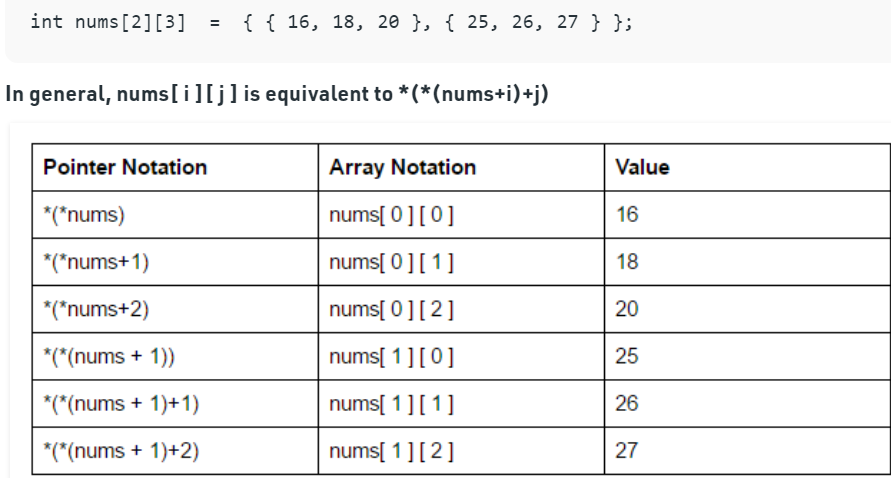
An array name contains the address of first element of the array which acts like constant pointer.

It means, the address stored in array name can’t be changed.

**Pointer Expression and pointer arithmetic**

* Increment (++)
* Decrement (--)
* Add pointer (+ or +=)
* Subtracted pointer (- or -=)
* Difference between two pointer (p1 – p2)

**Advanced Pointer Notation**



**Pointers and String Literals**

String literals are arrays containing null-terminated character sequences. String literals are arrays of type character plus terminating null-character, with each of the elements being of type const char (as characters of string can’t be modified).

Ex: const char\* p = “string”;

If we consider the array:

s (1800) t (1801) r (1802)

char x = \*(p + 1);

char y = p[3];

Both x and y will be same

**Pointers to Pointers**

We can create a pointer to a pointer that in turn may point to data or other pointer. Ex:

char a;

char \*b;

char \*\* c;

a = ’g’;

b = &a;

c = &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.

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.

**NULL Pointers**

Null pointer is a pointer which point nowhere and not just an invalid address. Following 2 methods to assign a pointer as NULL

Int \*ptr1 = 0;

Int \*ptr2 = NULL;