

# C++ LAUNCHPAD



## Lecture-6

# Pointers

- Address Of Operator
- Pointers
- Dereferencing Pointers

Prateek Narang

Doubts ?

# Initialization of Array.

# Pointers

# Address of Operator (&)

To get the address of a variable, we can use **the address-of operator (&)**

```
int p = 5;
```

```
cout << &p << endl; // It will print address of  
the variable;
```

-----

```
int arr[3];
```

```
cout << arr << endl; // it will show you address  
of first element.
```

```
cout << &arr[0] << endl; // same as above.
```

# What are pointers?

- Pointers are one of the most powerful and confusing aspects of the C/C++ language.
- A pointer is a variable that holds the address of another variable.
- To declare a pointer, we use an asterisk between the data type and the variable name

# Declaring a pointer variable!

```
int *pnPtr; // a pointer to an integer value
```

```
double *pdPtr; // a pointer to a double value
```

```
int* pnPtr2; // also valid syntax
```

```
int * pnPtr3; // also valid syntax
```

```
int *pnptr1, *pnptr3 // Declaring Multiple pointers.
```

Note – The space between the type and variable name.

# Initializing the pointer variable.

- Pointer variable when declared store some arbit collection of 1s and 0s. So we can say that they are pointing to some garbage address.
- We can initialize the pointer variable to some valid address i.e. address of same type of valid memory.
- `Int x = 10; int *q= &x;`
- We should never store address of a different type in the pointer variable.



# Dereference Operator (\*)

An interesting property of pointers is that they can be used to access the variable they point to directly. This is done by preceding the pointer name with the dereference operator (\*). The operator itself can be read as "**value pointed to by**"

Therefore the value pointed by q in previous example can be accessed as

```
int r = *q;
```

```
// or
```

```
cout << *q << endl;
```

```
// or
```

```
int z = (*q) + 1;
```



## So what is \* ?

- Using \* in a declaration of variable or as a function argument - is only signifying that this variable is meant to store an address.
- Using \* in an expression can mean two things
  - As binary operator - Multiplication
  - As unary operator – Dereferencing the address.

# Assignment in pointers

- A pointer variable is assignable – it means that we can change the address which it is storing now.

```
int x = 10, y = 20;
```

```
int *ptr = &x; // Now ptr is storing address of x
```

```
ptr = &y; // Now ptr is storing address of y;
```

- Assigning value which is being pointed at by the variable.

```
*ptr = 25; // This would change the value to  
which ptr is pointing to i.e. now y would become  
25.
```

Lets write some code.



We should never de-reference any garbage address !

So Always initialize your pointer with NULL.

# Null Pointer

Sometimes it is useful to make our pointers point to nothing. This is called a null pointer. We assign a pointer a null value by setting it to address 0:

```
double *p = 0;
```

Dereferencing Null pointer always gives segmentation fault.

# Pointers and Functions

Address are also passed by value  
to a function!



# Pointers and Arrays!

- Pointers and arrays are intricately linked in the C language
- An Array is actually a pointer that points to the first element of the array! Because the array variable is a pointer, you can dereference it, which returns array element 0:
- $a[i]$  is same as  $*(a + i)$

Note – An array name is just an alias to address of first element. There is no separate storage for the variable name. It behaves as a pointer but is not actually a pointer.

# Lets see some code!



# Pointer Arithmetic

- Addition, Multiplication, Division of two addresses doesn't make any sense.
- Addition of an address by a constant integer value i.e. `ptr + 5` means address of cell which is  $5 * \text{sizeof}(*\text{ptr})$  away from `ptr`.
- Similar for subtraction.
- Again Multiplying/Dividing an address with some constant value doesn't make any sense.
- Subtracting two address of same type would give you number of elements between them.

# Pointer Arithmetic contd..

Lets look at few input/output examples to understand more clearly!

So when you are passing an array to a function, you are actually passing the address of the first element.

# Lets see some problems.

- Sum of Array
- Bubble Sort

`void sumofarray(int arr[], int N)`  
is same as  
`void sumofarray(int *arr, int N)`

What would happen if I change  
the call to  
`cout << sumofarray(arr+5, 10);`



# Pointers vs Arrays

- the sizeof operator
  - sizeof(array) returns the amount of memory used by all elements in array
  - sizeof(pointer) only returns the amount of memory used by the pointer variable itself
- the & operator
  - &array is an alias for &array[0] and returns the address of the first element in array
  - &pointer returns the address of pointer
- Pointer variable can be assigned a value whereas array variable cannot be.  

```
int a[10];  
int *p;  
p=a; /*legal*/  
a=p; /*illegal*/
```
- Arithmetic on pointer variable is allowed.  

```
p++; /*Legal*/  
a++; /*illegal*/
```

# Pointer as return value

We should never return address of a local variable from a function.

# C++ LAUNCHPAD



Thank You!

Prateek Narang