Pointers

Points to be covered about pointers in C++ are as follows:

- Definition of pointers
- Why we need pointers
- Declaration of pointers
- Initialization of pointers
- Accessing the value of pointers
- Accessing the address of the pointers
- Pointer arithmetic

Definition of pointers

Pointers is a normal variable BUT it stores addresses instead of the common data types.

since Pointers is a normal variable, then it have these two things:

1. Memory location: It refers to the memory address where the pointer itself is stored.
2. Content: It refers to the address the pointer holds (address of another variable).

Think of a pointer as a map that guides you to a treasure. The map's location is where you hold it, and its content is the **address** of the treasure.

Why we need pointers

Pointers allow us to access variables or data indirectly. Instead of directly accessing the value of a variable or data, we use a pointer to access its memory address and indirectly retrieve or modify its value.

For example :

1. Iterating over an array

2. Passing arguments to functions by reference, enabling modification of original data inside the function.
3. Dynamic memory allocation.

Declaration

To declare a pointer variable it is just like any variable declaration

`dataType variableName;`

but for pointers we add one more thing, the `*` sign, then we have

`dataType *variableName;`

For example:

```
int *ptr;
```

This declares a pointer variable named `ptr` that can point to an integer value.

NOTE: the data type is not for the pointer itself but it is what we are going to point to.

Initialization

Pointer variables should always be initialized before we use it. We can initialize a pointer variable with the address of another variable using the & (Reference operator) operator. For example:

```
int num = 10;
int *ptr = &num;
```

This initializes a pointer variable named `ptr` with the address of the variable `num`.

Now `ptr` points to the address of `num`.

Accessing the value

since pointer holds a different data than usual, so we can access its content we use the dereference operator `*` . For example:

```
int num = 10;
int *ptr = &num;
cout << "Value: " << *ptr << "\n"; // 10
cout << "Address: " << ptr << "\n"; // the memory address of num (notice the *)
```

This will output the value of the variable `num`, which is 10.

Note: don't get confused with the * sign we just add it in the declaration to specify that this variable is a pointer and then we use it for dereferencing.

Pointer arithmetic

We can perform arithmetic operations such as addition and subtraction on pointers. For example:

```
int arr[5] = {1, 2, 3, 4, 5};
int *ptr = &arr[2];
cout << *(ptr + 2) << endl; // 5
cout << *(ptr - 2) << endl; // 1
```

The first line outputs the value 5, which is the 5th element of the array.

The second line outputs the value 1, which is the 1st element of the array.

Examples:

1. Direct Access:

```
int num = 10;
int* ptr = &num;

// Direct variable access
cout << "Value of num: " << num << endl; // 10

// Accessing variable through pointer
cout << "Value of num using pointer: " << *ptr << endl; // 10
```

2. Modifying data:

```
int num = 10;
int *ptr = &num;
num = 20;
cout << "Modified num: " << num << endl; // 20
```

```
*ptr = 30;
cout << "Modified num using pointer: " << num << endl; // 30
```

3. Array Access:

```
int arr[5] = {1, 2, 3, 4, 5};
int* ptr = arr;

// Direct array access
cout << "Value at index 2: " << arr[2] << endl;

// Array access through pointer
cout << "Value at index 2 using pointer: " << *(ptr + 2) << endl;

// Also can be done as normal array
cout << "Value at index 2 using pointer: " << ptr[2] << endl;
```

4. Swap Function(call by reference):

```
void swap(int *num1, int *num2)
{
    int temp = *num1;
    *num1 = *num2;
    *num2 = temp;
}
```