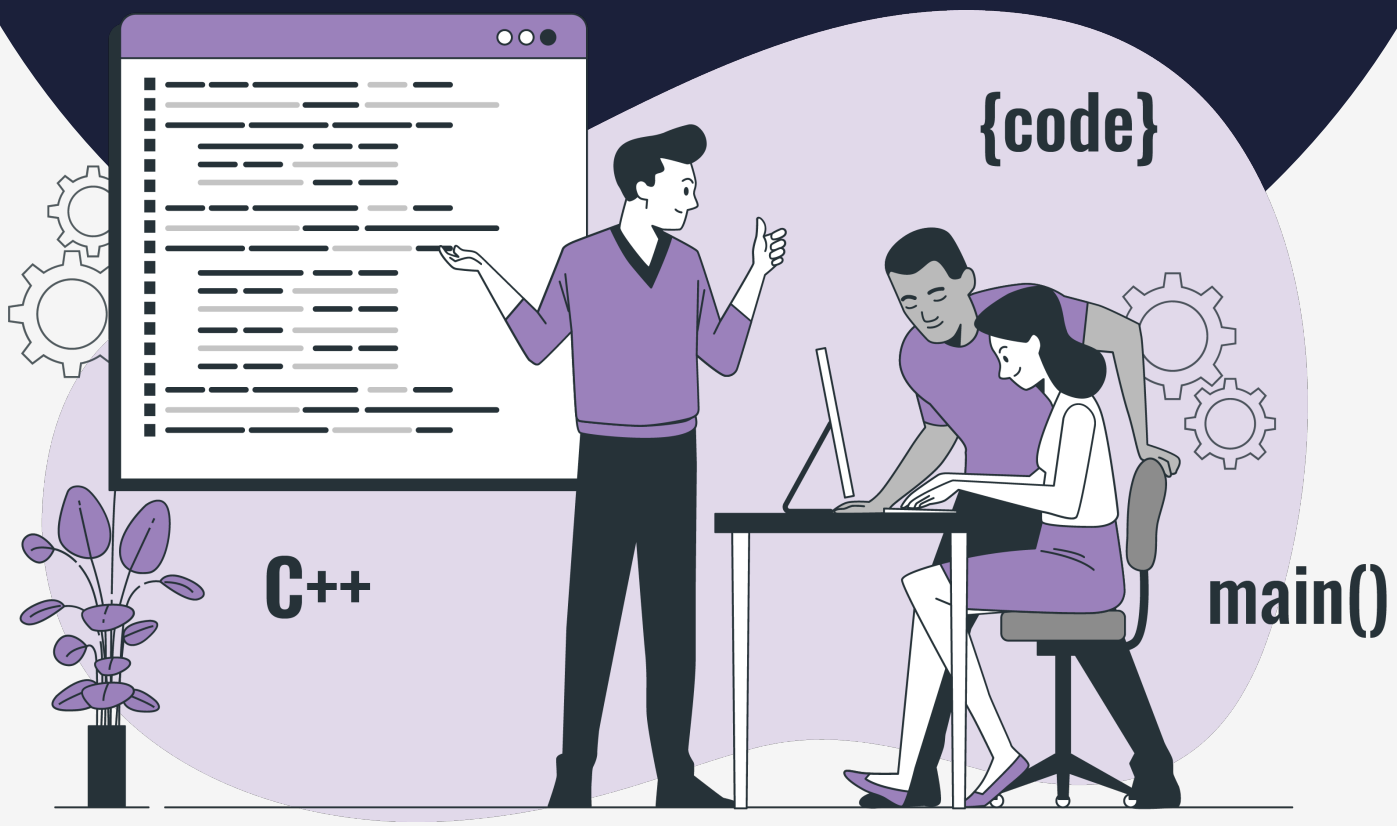
Lesson:

Pointers 1



PW SKILLS

Pre-Requisites

- Basic C++ syntax
- The concept of data storage in the memory?
- Datatypes in C++

List of Concepts Involved

- Introduction to pointers
- Storing address in a pointer
- Accessing the data through a pointer

Topic: Introduction to pointers

Let us assume that we are at a new apartment building and want to go to a particular flat, say flat number 43. We ask the watchman to tell us where it is. What do you think he would do? He will simply point to the location of the flat.
Pointers work in the same way. They are used to point to some location in the memory.
More precisely, a pointer is an object that stores the memory location of some data.
Now, imagine that the occupants of the apartment move out and some other family moves in. If we ask the watchman again where flat 43 is, will the answer change? No, right? It is because the watchman was pointing to the flat's location and not to the occupants.
Similarly, a pointer is used to point to the address of some data, so even if we change the data at that memory location, it won't affect what the pointer is pointing to.

The syntax for declaring a pointer in C++ is-

```
data-type *pointer-name;
```

1. The data type here must be the same as the data type of the data or variable, storing the data, that it is going to point to. For example, if we want to make a pointer that will point to an int variable, the data type of the pointer must also be int. The reason for this will be clear as we progress through the lesson.
2. The asterisk sign is used to indicate that the variable is a pointer.
3. pointer-name is the name that we choose to give to the pointer. It follows the general variable naming convention.

Example-

```
int *pi; // points to integer data
double *pd; // points to double data
```

Topic: Storing address in a pointer

This is done with the help of an operator called 'address-of' operator.

- **Address-of operator (&)**

When a variable is defined, it is stored in the memory at a particular location. We call this location the address of the variable. The address-of operator is used to get this address. This address can then be stored in a pointer.

For example,

```
int val = 10;
int *ptr = &val;
cout << ptr; // prints the address of val
```

Output: 0x61ff0c

In this example, the second statement defines `ptr` as a pointer to the type `int` and initializes `ptr` with the address of the int variable `val`. In other words, `ptr` points to the variable `val`.
In the output, we have the (hexadecimal) address of the pointed variable i.e. val in this case.

- Just like a normal variable can take multiple values over the course of its lifetime, a pointer can point to several different variables over its lifetime.

For instance,

```
int val = 10;
int *ptr = &val; // ptr points to val
cout << *ptr << ' ';
int val2 = 20;
ptr = &val2; // ptr points to val2
cout << *ptr;
Output: 10 20
```

In this example, we can clearly see that `ptr` was initialized with the address of `val`. Then, it was later assigned the address of `val2`. We can repeat this process several times.
In other words, as mentioned before, it is not necessary that a pointer will point to only one variable throughout its lifetime, it can point to several different variables. However, at a time, it points to only one variable i.e. it cannot point to several variables simultaneously.

- **Assigning value to a pointer**

```
int val = 10;
int *ptr = &val;
int *ptr2 = ptr; // initialize ptr2 with the same address as ptr
cout << ptr << ' ' << ptr2;
```

Output: 0x61ff0c 0x61ff0c

By assigning ptr to `ptr2` we assign the address stored in ptr to `ptr2` . Consequently, now `ptr2` points to the same variable, i.e.`val` in this case.

Topic: Accessing the data through a pointer

This is done using * operator, let us learn how ?

- **Dereferencing operator(*)**

It is used to fetch the data to which the pointer is pointing.

For example,

```
int val = 10;
int *ptr = &val;
cout << *ptr;
```

Output: 10

We can notice here that when we use `ptr` with the '*' operator, we get the value of the data that it is pointing to. In this example, the variable that `ptr` is pointing to is `val`, which has a value of 10 and hence we get 10 in the output.

- When we try to access a variable through a pointer, we get access to the variable itself. In other words, if we try to update the value of the pointer with the dereference operator, the value of the variable will change too.

For instance,

```
int val = 10;
int *ptr = &val;
cout << *ptr << ' ' << val << '\n';
*ptr = 20;
cout << *ptr << ' ' << val;
```

Output: 10 10
20 20

- As we know that the pointer is an object which points to a memory location, if we try to update the data at that location, then the change will be reflected on the dereferenced pointer too.

For example,

```
int val = 10;
int *ptr = &val;
cout << *ptr << ' ' << val << '\n';
val = 40;
cout << *ptr << ' ' << val;
```

Output: 10 10
40 40

- Since the dereference pointer gives us access to the data, it can be used in any situation where the data itself can be used.

For example,

1.
```
int val = 10;
int *ptr = &val;
int val2 = *ptr; // val2 = val
cout << val2;
```

Output: 10

2.
```
int val = 10;
int *ptr = &val;
int val2 = *ptr + *ptr; // val2 = val + val
cout << val2;
```

Output: 20

That is all for this class ! Keep learning, keep practicing !
See you in the next one !

Upcoming Class Teasers:

- Call-by-reference using pointers
- Arrays as pointers
- Pointer arithmetic
- Null pointers
- Invalid pointers
- Void pointers