Constructors

In this lesson, we explore the world of constructors and learn why they are necessary for a class.

WE'LL COVER THE FOLLOWING

What is a Constructor?

As the name suggests, the constructor is used to *construct* the object of a class. It is a special member function that outlines the steps that are performed when an instance of a class is created in the program.

> A constructor's **name** must be exactly the **same** as the name of its class.

The constructor is a special function because it **does not have a return type**. We do not even need to write `void` as the return type. It is a good practice to declare/define it as the first member function.

So, let's study the different types of constructors and use them to create class objects.

Default Constructor

The default constructor is the most basic form of a constructor. Think of it this way:

> In a default constructor, we define the default values for the data

> members of the class. Hence, the constructor creates an object in which the data members are initialized to their default values.

This will make sense when we look at the code below. Here, we have a **Date** class, with its default constructor, and we'll create an object out of it in `main()`:

```
#include <iostream>
#include <string>
using namespace std;

class Date {
  int day;
  int month;
  int year;

  public:
  // Default constructor
  Date(){
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // A simple print function
  void printDate(){
    cout << "Date: " << day << "/" << month << "/" << year << endl;
  }
};

int main(){
  // Call the Date constructor to create its object;

  Date d; // Object created with default values!
  d.printDate();
}
```

Notice that when we created a `Date` object in **line 28**, we don't treat the constructor as a function and write this:

```
d.Date()
```

We create the object just like we create an `integer` or `string` object. It's that easy!

The default constructor does not need to be explicitly defined. Even if we

The default constructor does not need to be explicitly defined. Even if we don't create it, the C++ compiler will call a default constructor and set data members to `null` or `0`.

Parameterized Constructor

The default constructor isn't all that impressive. Sure, we could use `set` functions to set the values for `day`, `month` and `year` ourselves, but this step can be avoided using a **parameterized constructor**.

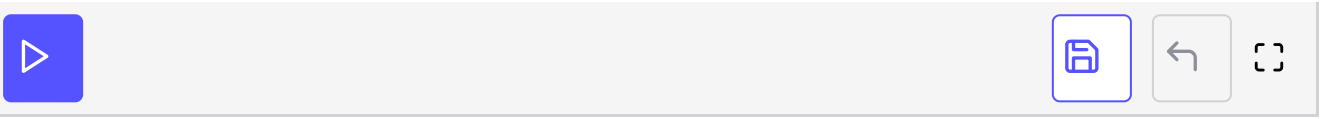
> In a parameterized constructor, we pass arguments to the constructor and set them as the values of our data members.

We are basically overriding the default constructor to accommodate our preferred values for the data members.

Let's try it out:

```
#include <iostream>
#include <string>
using namespace std;

class Date {
  int day;
  int month;
  int year;

  public:
  // Default constructor
  Date(){
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // Parameterized constructor
  Date(int d, int m, int y){
    // The arguments are used as values
    day = d;
    month = m;
    year = y;
  }

  // A simple print function
  void printDate(){
    cout << "Date: " << day << "/" << month << "/" << year << endl;
  }
};
```

```
int main(){
  // Call the Date constructor to create its object;


  Date d(1, 8, 2018); // Object created with specified values!
  d.printDate();
}
```



This is much more convenient than the default constructor!

`this` Pointer

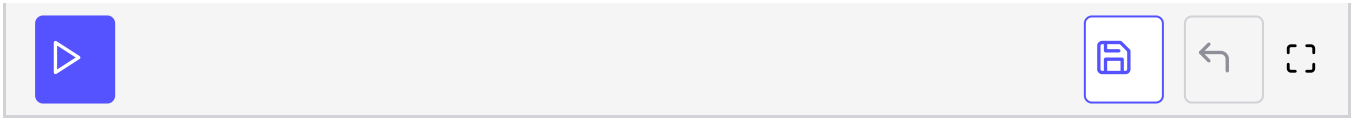
The `this` pointer exists for every class. It points to the class object itself. We use the pointer when we have an argument which has the same name as a data member. `this->memberName` specifies that we are accessing the `memberName` variable of the particular class.

> **Note**: Since `this` is a pointer, we use the `->` operator to access members instead of `.`.

Let's see it in action:

```
#include <iostream>
#include <string>
using namespace std;

class Date {
  int day;
  int month;
  int year;

  public:
  // Default constructor
  Date(){
    // We must define the default values for day, month, and year
    day = 0;
    month = 0;
    year = 0;
  }

  // Parameterized constructor
  Date(int day, int month, int year){
    // Using this pointer
    this->day = day;
    this->month = month;
    this->year = year;
  }
```

```
  // A simple print function
  void printDate(){

    cout << "Date: " << day << "/" << month << "/" << year << endl;
  }
};

int main(){
  // Call the Date constructor to create its object;

  Date d(1, 8, 2018); // Object created with specified values!
  d.printDate();
}
```



This marks the end of our discussion on *constructors*. In the next lesson, we will discuss the opposite of constructors, **destructors**.