

Constructors

This lesson introduces the concept of constructors, how to declare, overload and call them

We'll cover the following ^

- Introduction
- Constructor Declaration
- Constructor Definition
 - Calling a constructor
- Constructor Overloading

Introduction

A *constructor* is automatically called when an *object* of the *class* is declared.

- A *constructor* is a *member* function that is usually `public`.
- A *constructor* can be used to initialize *member* variables when an *object* is declared.

Note: A *constructor's* **name** must be the **same** as the *name* of the *class* it is declared in

A constructor cannot *return* a value.

Note: No *return* type, not even `void` can be used while declaring a *constructor*

Constructor Declaration

A constructor for the `DayofYear` class can be declared as follows:

```
class DayOfYear {
```

```

public:
    //initializes month to new_month
    //initialized day to new_day
    DayOfYear(int new_month, int new_day)
private:
    int month;
    int day;
};

```

Constructor Definition

The constructor for `DayOfYear` can be defined as follows:

```

#include <iostream>
using namespace std;

class DayOfYear
{
public:
    DayOfYear(int new_month, int new_day); //declaring constructor
    DayOfYear(); //default constructor without any parameters
    int myVar;
    void output( );
    int get_month( );
    int get_day( );

private:
    void check_date( );
    int month;
    int day;
};

int main(){
    DayOfYear birthday(11,23); //creating object and calling constructor
    DayOfYear today; //creating object and calling default constructor
    cout << "Birthday day is: " << birthday.get_day()<<endl;
    cout << "Birthday month is: " << birthday.get_month()<<endl;
    cout << "Today the day is: " << today.get_day()<<endl;
    cout << "Today month is: " << today.get_month()<<endl;
    return 0;
}

//defining constructor
DayOfYear::DayOfYear(int new_month, int new_day){ //class name and constructor name are same
    month = new_month;
    day = new_day;
}

DayOfYear::DayOfYear(){ //defining default constructor
    month = 0;
    day = 0;
}

int DayOfYear::get_month( )
{
    return month;    //returns the private variable month
}

```

```
int DayOfYear::get_day( )
{
    return day;    //returns the private variable day
}
```



Calling a constructor

As you can see above in **line 22**, the way to call a *constructor* is not like a normal *member* function.

- It is called in *object* declaration.
- It creates a `DayOfYear` object.
- Then *calls* the *constructor* to initialize *variables*.

In the example, we also *declare* and use the **default constructor** which takes no parameters and just *initializes* `month` and `day` to **0**.

As you can see in **line 23** a *default constructor* is automatically called when you create an *object*, no *parameters* are needed.

Note: It's a good practice to use default constructors even if you don't want to initialize any variables.

Constructor Overloading

Constructors can be *overloaded* by defining *constructors* with **different** *parameters* list.

Other possible constructors for `DayOfYear` class can be the following:

```
DayOfYear(); //default
DayOfYear(int newmonth, newday); //passing two int parameters
DayOfYear(double newmonth, double newday); //passing two double parameters
DayOfYear(float newday); //passing a float parameter
```



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

