# C++ Constructors

A constructor is a special type of member function that initialises an object automatically when it is created.

Compiler identifies a given member function is a constructor by its name and the return type.

Constructor has the same name as that of the class and it does not have any return type. Also, the constructor is always public.

... .. ...

class temporary

{

private:

int x;

float y;

public:

// Constructor

temporary(): x(5), y(5.5)

{

// Body of constructor

}

... .. ...

};

int main()

{

Temporary t1;

... .. ...

}

Above program shows a constructor is defined without a return type and the same name as the class.

## How constructor works?

In the above pseudo code, temporary() is a constructor.

When an object of class temporary is created, the constructor is called automatically, and xis initialized to 5 and y is initialized to 5.5.

You can also initialise the data members inside the constructor's body as below. However, this method is not preferred.

temporary()

{

x = 5;

y = 5.5;

}

// This method is not preferred.

## Use of Constructor in C++

Suppose you are working on 100's of Person objects and the default value of a data member age is 0. Initialising all objects manually will be a very tedious task.

Instead, you can define a constructor that initialises age to 0. Then, all you have to do is create a Person object and the constructor will automatically initialise the age.

These situations arise frequently while handling array of objects.

Also, if you want to execute some code immediately after an object is created, you can place the code inside the body of the constructor.

### Example 1: Constructor in C++

**Calculate the area of a rectangle and display it.**

#include <iostream>

using namespace std;

class Area

{

private:

int length;

int breadth;

public:

// Constructor

Area(): length(5), breadth(2){ }

void GetLength()

{

cout << "Enter length and breadth respectively: ";

cin >> length >> breadth;

}

int AreaCalculation() { return (length \* breadth); }

void DisplayArea(int temp)

{

cout << "Area: " << temp;

}

};

int main()

{

Area A1, A2;

int temp;

A1.GetLength();

temp = A1.AreaCalculation();

A1.DisplayArea(temp);

cout << endl << "Default Area when value is not taken from user" << endl;

temp = A2.AreaCalculation();

A2.DisplayArea(temp);

return 0;

}

In this program, class Area is created to handle area related functionalities. It has two data members length and breadth.

A constructor is defined which initialises length to 5 and breadth to 2.

We also have three additional member functions GetLength(), AreaCalculation() and DisplayArea() to get length from the user, calculate the area and display the area respectively.

When, objects A1 and A2 are created, the length and breadth of both objects are initialized to 5 and 2 respectively, because of the constructor.

Then, the member function GetLength() is invoked which takes the value of length and breadth from the user for object A1. This changes the length and breadth of the object A1.

Then, the area for the object A1 is calculated and stored in variable temp by calling AreaCalculation() function and finally, it is displayed.

For object A2, no data is asked from the user. So, the length and breadth remains 5 and 2 respectively.

Then, the area for A2 is calculated and displayed which is 10.

**Output**

Enter length and breadth respectively: 6

7

Area: 42

Default Area when value is not taken from user

Area: 10

## Constructor Overloading

Constructor can be overloaded in a similar way as function overloading.

Overloaded constructors have the same name (name of the class) but different number of arguments.

Depending upon the number and type of arguments passed, specific constructor is called.

Since, there are multiple constructors present, argument to the constructor should also be passed while creating an object.

## Example 2: Constructor overloading

// Source Code to demonstrate the working of overloaded constructors

#include <iostream>

using namespace std;

class Area

{

private:

int length;

int breadth;

public:

// Constructor with no arguments

Area(): length(5), breadth(2) { }

// Constructor with two arguments

Area(int l, int b): length(l), breadth(b){ }

void GetLength()

{

cout << "Enter length and breadth respectively: ";

cin >> length >> breadth;

}

int AreaCalculation() { return length \* breadth; }

void DisplayArea(int temp)

{

cout << "Area: " << temp << endl;

}

};

int main()

{

Area A1, A2(2, 1);

int temp;

cout << "Default Area when no argument is passed." << endl;

temp = A1.AreaCalculation();

A1.DisplayArea(temp);

cout << "Area when (2,1) is passed as argument." << endl;

temp = A2.AreaCalculation();

A2.DisplayArea(temp);

return 0;

}

For object A1, no argument is passed while creating the object.

Thus, the constructor with no argument is invoked which initialises length to 5 and breadthto 2. Hence, area of the object A1 will be 10.

For object A2, 2 and 1 are passed as arguments while creating the object.

Thus, the constructor with two arguments is invoked which initialises length to l (2 in this case) and breadth to b (1 in this case). Hence, area of the object A2 will be 2.

**Output**

Default Area when no argument is passed.

Area: 10

Area when (2,1) is passed as argument.

Area: 2

## Default Copy Constructor

An object can be initialized with another object of same type. This is same as copying the contents of a class to another class.

In the above program, if you want to initialise an object A3 so that it contains same values as A2, this can be performed as:

....

int main()

{

Area A1, A2(2, 1);

// Copies the content of A2 to A3

Area A3(A2);

OR,

Area A3 = A2;

}

You might think, you need to create a new constructor to perform this task. But, no additional constructor is needed. This is because the copy constructor is already built into all classes by default.