

Constructors and Destructors

Constructors are the special type of class functions which initializes the object automatically when the object is created, the compiler identifies that the given member function is a constructor by its name and return type. The constructor has the same name as the class and it does not have any return type. Constructor initializes the values to object members after the memory is allocated to the object. The constructors are always public and they are used to make initializations at the time of object creation. The objects constructor is automatically called whenever the object is created statically or dynamically.

Example for Constructor inside the class:

```
class student
{
private:
    string name;
    int rollno;
public:
    student() // constructor defined inside the class
    {
        // body of the constructor
    }
};
```

Constructor can be defined either inside the class or outside the class using class name and scope resolution '::' operator.

Example for Constructor outside the class:

```
class student
```

```
{  
Private:  
    string name;  
    int rollno;  
public:  
    student(); // constructor declaration inside the class  
};  
  
student :: student() // constructor defined outside the class  
{  
    // body of the constructor  
}
```

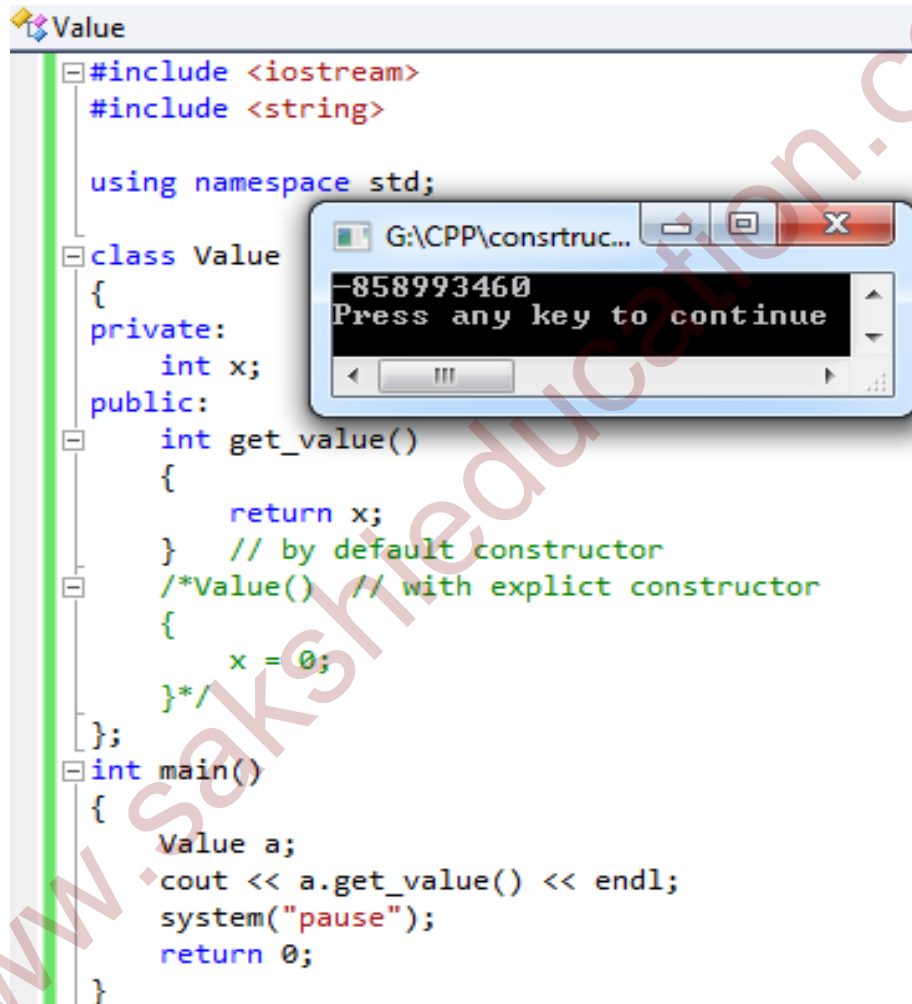
Actually the compiler has a default constructor then why do we need to define a constructor for a given class and some other important points related to constructor are as follows.

- We define a constructor when we want to initialize the data members of the class with some default values or with some arguments passed to the class or when we want to carry out any work when an object is created.
- As soon as we define a constructor the default constructor created by compiler is destroyed, so while defining a constructor make sure that the default constructor also defined.
- If we do not specify any constructor the compiler will create a default constructor with no arguments and empty body loop.
- Constructors have no return type.
- Constructors function name is same as that of class name.

The following examples show default constructor and explicit used defined constructor. If the class does not have any used defined explicit constructor, then the compiler will automatically create an empty default constructor which does not initialize any of the class member variables,

if we use it to allocate an object of a class, then the member variables of the class will not be initialized and they will hold garbage values. It's good to provide at least one constructor in the class which will prevent the compiler from creating an empty default constructor and ensuring that users don't have to instantiate objects of the class that have uninitialized members.

Example with default constructor:



```
#include <iostream>
#include <string>

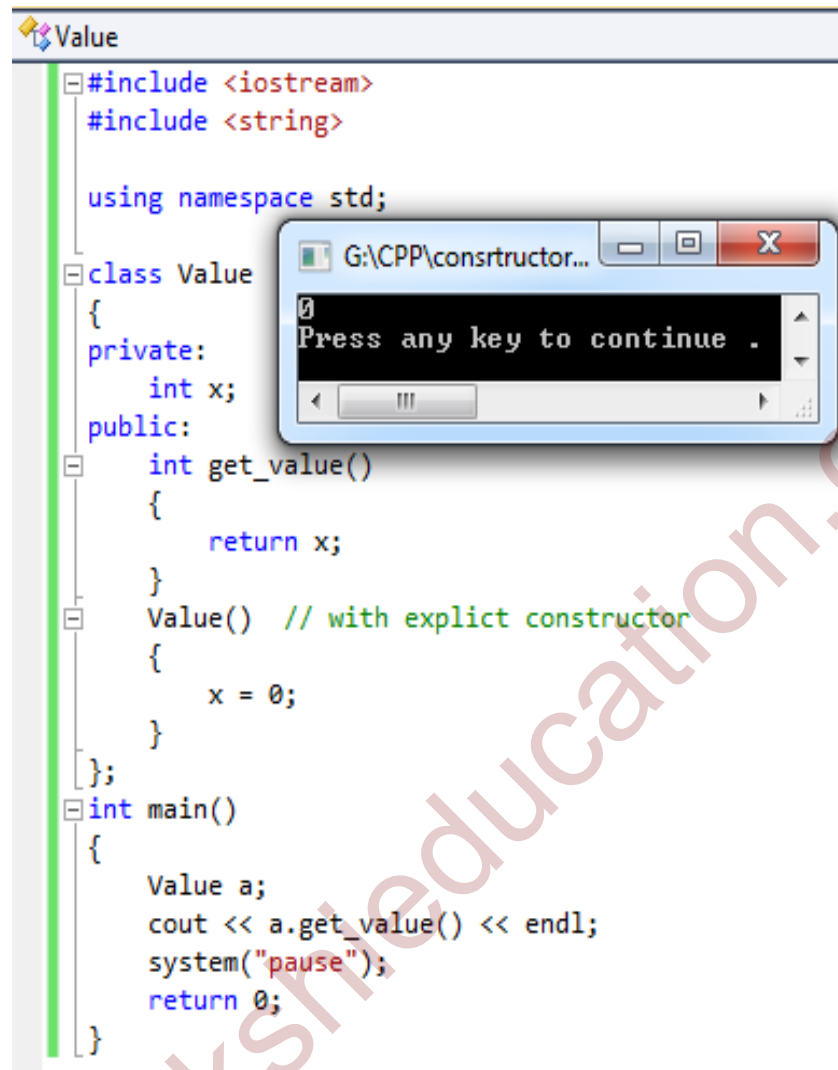
using namespace std;

class Value
{
private:
    int x;
public:
    int get_value()
    {
        return x;
    } // by default constructor
    /*Value() // with explicit constructor
    {
        x = 0;
    }*/
};

int main()
{
    Value a;
    cout << a.get_value() << endl;
    system("pause");
    return 0;
}
```

Figure1: Example with default constructor which holds garbage value

Example with explicit constructor:



```
#include <iostream>
#include <string>

using namespace std;

class Value
{
private:
    int x;
public:
    int get_value()
    {
        return x;
    }
    Value() // with explicit constructor
    {
        x = 0;
    }
};

int main()
{
    Value a;
    cout << a.get_value() << endl;
    system("pause");
    return 0;
}
```

Figure2: Example with explicit constructor with initialized values

Types of constructors

The constructors are classified into three types and they are

1. Default constructor
2. Parameterized constructor
3. Copy constructor

Default constructor

The default constructor does not take any arguments. The default constructor is so important for initialization of object members and even if we do not define any constructor explicitly, the compiler will provide the default constructor.

Parameterized constructor

The parameterized constructor is used to provide different values to data members of different objects by passing the appropriate values as arguments.

Copy constructor

Copy constructors are the special type of constructors which takes an object as argument and are used to copy the values of data members of one object into the other object.

Example:

```
int main ()
{
    student s1, s2, s3;
    student s4(s3); or student s4 = s3; // it copies the content of s3 to s4
}
```

Example program for all constructors:

The following example program shows all the three types of constructors.

```
#include <iostream>
#include <string>

using namespace std;

class Value
{
private:
    int x;

public:
```

```
int get_value()
{
    return x;
}

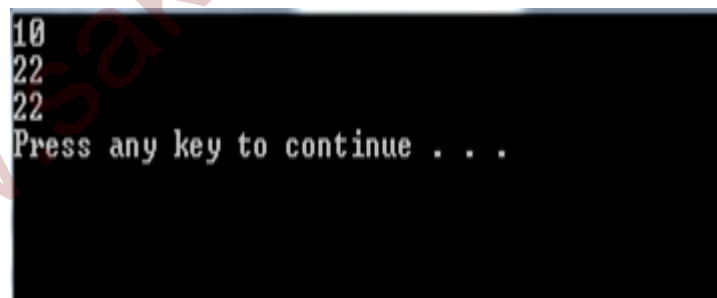
Value() // default constructor
{
    x = 10;
}

Value(int i) // parameterized constructor
{
    x = i;
}

Value(Value &b) // copy constructor
{
    x = b.x;
}

};

int main()
{
    Value a; // default constructor set value of x to 10
    Value b (22); // parameterized constructor set value of x to 22
    Value c = b; // copy constructor will copy the values of constructor b to
    constructor c
    cout << a.get_value() << endl; // print the value -- 10
    cout << b.get_value() << endl; // print the value -- 22
    cout << c.get_value() << endl ;// print the value -- 22
    system("pause");
    return 0;
}
```



```
10
22
22
Press any key to continue . . .
```

Figure3: Output of the program

Constructor Overloading

The constructors can be overloaded like function overloading. The overloaded constructors have the same name of the class but different number of arguments passed.

Depending upon the number and type of argument passed the specific constructor is called. Since the constructor is called when object is created, so argument to the constructor also passed while creating object.

Example program for constructor overloading:

The following program shows the constructor without arguments and with different number of arguments.

```
#include <iostream>
#include <string>

using namespace std;

class student{

private:
    string name;
    int roll_no;

public:
    student () // Constructor without argument
    {
        name = "student1";
        cout << "Name: " << name << endl;
        roll_no = 7;
        cout << "Roll_no: " << roll_no << endl;
    }

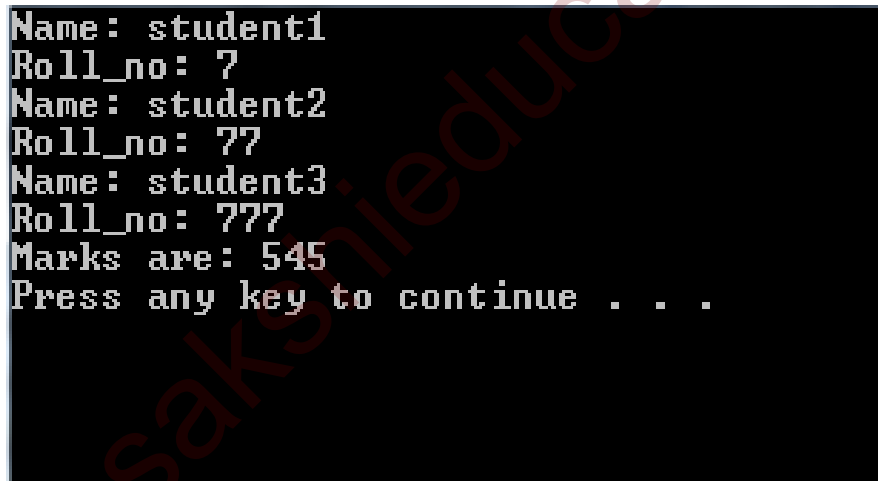
    student (string x, int y) // Constructor with two arguments
    {
        name = x;
        cout << "Name: " << name << endl;
        roll_no = y;
        cout << "Roll_no: " << roll_no << endl;
    }

    student (string x, int y, int z) // Constructor with three arguments
    {
        int marks;
        name = x;
        cout << "Name: " << name << endl;
        roll_no = y;
        cout << "Roll_no: " << roll_no << endl;
        marks = z;
        cout << "Marks are: " << marks << endl;
    }

};
```

```
int main()
{
    student s1; // no arguments
    student s2("student2", 77); // two arguments
    student s3("student3",777, 545); // three arguments
    system("pause");
    return 0;
}
```

In the above program the object s1 has no arguments, so here the constructor with no argument is invoked which initializes the member name to “student1” and roll_no to “7”. For the object s2 we passed two arguments, so it will call the constructor with two arguments which initializes the member name to “student2” and roll_no to “77”. Similarly for object s3 we passed three arguments so it will call the constructor with three arguments which initializes the member name “student3”, roll_no to “777” and marks to “545” as shown in the below output screen.



```
Name: student1
Roll_no: 7
Name: student2
Roll_no: 77
Name: student3
Roll_no: 777
Marks are: 545
Press any key to continue . . .
```

Figure4: Output of the program

Destructors

The destructor is a special type of class function which destroys the object as soon as the scope of the object ends. The destructor is called automatically by the compiler when the object goes out of the scope. The destructor is looks like similar to constructor. For destructor the class

name is used for the name of destructor with a tilde '~' symbol as prefix to it. The destructors will never have any arguments and no return type.

Syntax:

```
class student
```

```
{
```

```
    public:
```

```
    ~student(); // Destructor
```

```
};
```

Important points about destructor:

- The destructor function is called automatically when object goes out of the scope:
 - When the function call ends.
 - When the program ends.
 - When a block containing local variables ends.
 - When a delete operator is called.
- There is only one destructor in a class with class name preceded by '~' tilde operator with no arguments and return type.
- If we do not create user defined destructor in class then the compiler creates a default destructor. This works fine unless we have dynamically allocated memory or pointer in class. When a class contains a pointer to memory allocated in class then we should create a destructor to release memory before the instance is destroyed otherwise it create memory leak.

Example program for destructor:

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

```
class destructor
{
    public:

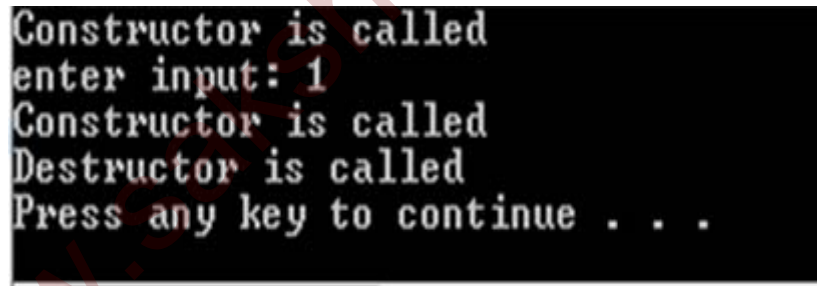
        destructor()
        {
            cout << "Constructor is called" << endl;
        }

        ~destructor()
        {
            cout << "Destructor is called" << endl;
        }
};

int main()
{
    destructor temp1; // constructor is called
    int x;
    cout << "enter input: ";
    cin >> x;

    if(x == true)
    {
        destructor temp2; // constructor is called
    } // destructor is called for temp2

    system("pause");
    return 0;
} // destructor is called for temp1
```



```
Constructor is called
enter input: 1
Constructor is called
Destructor is called
Press any key to continue . . .
```

Figure5: Output of the program