

Constructors in C++

Constructor in C++ is a special method that is invoked automatically at the time of object creation. It is used to initialize the data members of new objects generally. The constructor in C++ has the same name as the class or structure. Constructor is invoked at the time of object creation. It constructs the values i.e. provides data for the object which is why it is known as constructors.

Constructor does not have a return value, hence they do not have a return type.

The prototype of Constructors is as follows:

```
<class-name> (list-of-parameters);
```

Constructors can be defined inside or outside the class declaration:-

The syntax for defining the constructor within the class:

```
<class-name> (list-of-parameters) { //  
constructor definition }
```

The syntax for defining the constructor outside the class:

```
<class-name>: :<class-name> (list-of-  
parameters){ // constructor definition}
```

Example

C++

```
// defining the constructor within the class
```

```
#include <iostream>
```

```
using namespace std;
```

```
class student {
```

```
    int rno;
```

```
    char name[10];
```

```
    double fee;
```

```
public:
```

```
student()
```

```
{
```

```
    cout << "Enter the RollNo:";
```

```
    cin >> rno;
```

```
    cout << "Enter the Name:";
```

```
    cin >> name;
```

```
    cout << "Enter the Fee:";
```

```
    cin >> fee;
```

```
}
```

```
void display()
```

```
{  
  
    cout << endl << rno << "\t" << name <<  
    "\t" << fee;  
  
}  
  
};
```

```
int main()
```

```
{
```

```
    student s; // constructor gets called  
    automatically when
```

```
        // we create the object of the  
class
```

```
s.display();
```

```
return 0;
```

```
}
```

OutputEnter the RollNo:Enter the
Name:Enter the Fee: 0 6.95303e-310

Example

C++

```
// defining the constructor outside the  
class
```

```
#include <iostream>

using namespace std;

class student {

    int rno;

    char name[50];

    double fee;

public:

    student();

    void display();

};
```

```
student::student()
```

```
{
```

```
    cout << "Enter the RollNo:";
```

```
    cin >> rno;
```

```
    cout << "Enter the Name:";
```

```
    cin >> name;
```

```
    cout << "Enter the Fee:";
```



```
cin >> fee;
```

```
}
```

```
void student::display()
```

```
{
```

```
    cout << endl << rno << "\t" << name <<  
    "\t" << fee;
```

```
}
```

```
int main()
```

```
{
```

```
student s;  
  
s.display();
```

```
return 0;
```

```
}
```

Output:

```
Enter the RollNo: 30 Enter the Name: ram  
Enter the Fee: 20000 30 ram 20000
```

How constructors are different from a normal member function?

C++

```
#include <iostream>
```

```
using namespace std;
```

```
class Line {
```

```
    public:
```

```
        void setLength( double len );
```

```
        double getLength( void );
```

```
        Line( double len ); //This is the  
constructor
```

```
    private:
```

```
        double length;
```

```
};
```

```
//Member function definition including  
constructor
```

```
Line::Line( double len ) {
```

```
    cout<<"Object is being created , length  
="<< len <<endl;
```

```
    length = len;
```

```
}
```

```
void Line::setLength( double len ) {
```

```
    length = len;
```

```
}
```

```
double Line::getLength( void ) {  
  
    return length;  
  
}
```

```
//Main function for the program
```

```
int main() {
```

```
    Line line(10.0);
```

```
    //get initially set length
```

```
    cout<<"Length of line : " <<  
line.getLength() << endl;
```

```
    //set line length again
```

```
    line.setLength(6.0);
```

```
cout<<"Length of line :" <<  
line.getLength() << endl;
```

```
return 0;
```

```
}
```

- A constructor is different from normal functions in following ways: Constructor has same name as the class itself
- Default Constructors don't have input argument however, Copy and Parameterized Constructors have input arguments
- Constructors don't have return type

- A constructor is automatically called when an object is created.
- It must be placed in public section of class.
- If we do not specify a constructor, C++ compiler generates a default constructor for object (expects no parameters and has an empty body).

Characteristics of the constructor:

- The name of the constructor is the same as its class name.
- Constructors are mostly declared in the public section of the class though it can be declared in the private section

of the class.

- Constructors do not return values; hence they do not have a return type.
- A constructor gets called automatically when we create the object of the class.
- Constructors can be overloaded.
- Constructor can not be declared virtual.

Types of Constructors

1. Default Constructors: Default constructor is the constructor which doesn't take any argument. It has no parameters. It is also called a zero-argument constructor.

// Cpp program to illustrate the


```
// concept of Constructors
```

```
#include <iostream>
```

```
using namespace std;
```

```
class construct {
```

```
public:
```

```
    int a, b;
```

```
    // Default Constructor
```

```
    construct()
```

```
{
```

```
    a = 10;
```

```
    b = 20;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    // Default constructor called  
    automatically
```

```
    // when the object is created
```

```
    construct c;  
  
    cout << "a: " << c.a << endl << "b: " << c.b;  
  
    return 1;  
  
}
```

Output: 10 b: 20

Note: Even if we do not define any constructor explicitly, the compiler will automatically provide a default constructor implicitly.

C++

// Example

```
#include<iostream>
```

```
using namespace std;
```

```
class student
```

```
{
```

```
    int rno;
```

```
    char name[50];
```

```
    double fee;
```

```
public:
```

```
    student()
```

```
        // Explicit Default
```

```
constructor
```

```
{
```

```
    cout<<"Enter the RollNo:";
```

```
cin>>rno;
```

```
cout<<"Enter the Name:";
```

```
cin>>name;
```

```
cout<<"Enter the Fee:";
```

```
cin>>fee;
```

```
}
```

```
void display()
```

```
{
```

```
    cout<<endl<<rno<<"\t"<<name<<"\t"<<f
```

```
ee;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    student s;
```

```
    s.display();
```

```
    return 0;
```

```
}
```

. Parameterized Constructors: It is possible to pass arguments to constructors. Typically, these arguments

help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.

Note: when the parameterized constructor is defined and no default constructor is defined explicitly, the compiler will not implicitly call the default constructor and hence creating a simple object as

Student s; Will flash an error

CPP

// CPP program to illustrate

// parameterized constructors

```
#include <iostream>
```

```
using namespace std;
```

```
class Point {
```

```
private:
```

```
    int x, y;
```

```
public:
```

```
    // Parameterized Constructor
```

```
    Point(int x1, int y1)
```

```
{
```



```
x = x1;
```

```
y = y1;
```

```
}
```

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

```
int getY() { return y; }
```

```
};
```

```
int main()
```

```
{
```

```
// Constructor called
```

```
Point p1(10, 15);
```

```
// Access values assigned by  
constructor
```

```
cout << "p1.x = " << p1.getX()
```

```
<< ", p1.y = " << p1.getY();
```

```
return 0;
```

```
}
```

Outputp1.x = 10, p1.y = 15

C++

// Example

```
#include<iostream>
```

```
#include<string.h>
```

```
using namespace std;
```

```
class student
```

```
{
```

```
    int rno;
```

```
    char name[50];
```

```
double fee;
```

```
public:
```

```
student(int, char[], double);
```

```
void display();
```

```
};
```

```
student::student(int no, char n[], double f)
```

```
{
```

```
rno=no;
```

```
strcpy(name,n);
```

```
fee=f;
```

```
}
```

```
void student::display()
```

```
{
```

```
    cout<<endl<<rno<<"\t"<<name<<"\t"<<fee
```

```
;
```

```
}
```

```
int main()
{
    student s(1001,"Ram",10000);

    s.display();

    return 0;
}
```

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly.

```
Example e = Example(0, 50); // Explicit call
Example e(0, 50); // Implicit call
```

- **Uses of Parameterized constructor:**
- It is used to initialize the various data elements of different objects with different values when they are created.
- It is used to overload constructors.
- **Can we have more than one constructor in a class?**

Yes, It is called Constructor Overloading.

3. Copy Constructor:

A copy constructor is a member function that initializes an object using another object of the same class. A detailed article on Copy Constructor.

Whenever we define one or more non-default constructors(with parameters) for a class, a default constructor(without parameters) should also be explicitly defined as the compiler will not provide a default constructor in this case. However, it is not necessary but it's considered to be the best practice to always define a default constructor.

Copy constructor takes a reference to an object of the same class as an argument.

```
Sample(Sample &t) { id=t.id; }
```

CPP

```
// Illustration
```

```
#include <iostream>
```



```
using namespace std;
```

```
class point {
```

```
private:
```

```
    double x, y;
```

```
public:
```

```
    // Non-default Constructor &
```

```
    // default Constructor
```

```
    point(double px, double py) { x = px, y =  
py; }
```

```
};
```

```
int main(void)
```

```
{
```

```
    // Define an array of size
```

```
    // 10 & of type point
```

```
    // This line will cause error
```

```
    point a[10];
```

```
    // Remove above line and program
```

```
// will compile without error
```

```
point b = point(5, 6);
```

```
}
```

```
// Example: Explicit copy constructor
```

```
#include <iostream>
```

```
using namespace std;
```

```
class Sample
```

```
{
```

```
int id;
```

```
public:
```

```
void init(int x)
```

```
{
```

```
    id=x;
```

```
}
```

```
    Sample(){} //default constructor with  
empty body
```

```
    Sample(Sample &t) //copy constructor
```

```
{
```

```
id=t.id;
```

```
}
```

```
void display()
```

```
{
```

```
    cout<<endl<<"ID="<<id;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
    Sample obj1;
```

```
    obj1.init(10);
```

```
obj1.display();
```

Sample obj2(obj1); //or obj2=obj1;
copy constructor called

```
obj2.display();
```

```
return 0;
```

```
}
```

OutputID=10 ID=10

```
#include<iostream>
```

```
#include<string.h>
```

```
using namespace std;
```

```
class student
```

```
{
```

```
    int rno;
```

```
    char name[50];
```

```
    double fee;
```

```
public:
```

```
    student(int,char[],double);
```

```
    student(student &t)    //copy
```

```
constructor
```

```
{
```

```
    rno=t.rno;
```

```
strcpy(name,t.name);
```

```
fee=t.fee;
```

```
}
```

```
void display();
```

```
};
```



```
student::student(int no,char n[],double f)
```

```
{
```

```
    rno=no;
```

```
    strcpy(name,n);
```

```
    fee=f;
```

```
}
```

```
void student::display()
```

```
{
```

```
    cout<<endl<<rno<<"\t"<<name<<"\t"<<f
```

```
ee;
```

```
}
```

```
int main()
```

```
{
```

```
    student s(1001,"Manjeet",10000);
```

```
    s.display();
```

```
    student manjeet(s); //copy constructor  
called
```

```
    manjeet.display();
```

```
return 0;
```

```
}
```

```
#include<iostream>
```

```
#include<string.h>
```

```
using namespace std;
```

```
class student
```

```
{
```

```
int rno;
```

```
char name[50];
```

```
double fee;
```

```
public:
```

```
student(int,char[],double);
```

```
student(student &t) //copy  
constructor
```

```
{
```

```
    rno=t.rno;
```

```
    strcpy(name,t.name);
```

```
    fee=t.fee;
```

```
}
```

```
void display();
```

```
};
```

```
student::student(int no,char n[],double f)
```

```
{
```

```
    rno=no;
```

```
    strcpy(name,n);
```

```
    fee=f;
```

```
}
```

```
void student::display()
```

```
{
```

```
    cout<<endl<<rno<<"\t"<<name<<"\t"<<f
```

```
ee;
```

```
}
```

```
int main()
```

```
{
```

```
    student s(1001,"Manjeet",10000);
```

```
    s.display();
```

```
    student manjeet(s); //copy constructor  
called
```

```
    manjeet.display();
```

```
    return 0;
```

```
}
```

```
#include <iostream>
```

```
using namespace std;
```

```
int count = 0;
```

```
class Test {
```

public:

Test()

{

count++;

cout << "\n No. of Object created:\t"

<< count;

}

~Test()

{

cout << "\n No. of Object destroyed:\t"


```
<< count;
```

```
    --count;
```

```
}
```

```
};
```

```
main()
```

```
{
```

```
    Test t, t1, t2, t3;
```

```
    return 0;
```

```
}
```

Output No. of Object created: 1 No. of

Object created: 2 No. of Object created: 3
No. of Object created: 4 No. of Object
destroyed: 4 No. of Object destroyed: 3 No.
of Object destroyed: 2 No. of Object
destroyed: 1

Characteristics of a destructor:-

1. Destructor is invoked automatically by the compiler when its corresponding constructor goes out of scope and releases the memory space that is no longer required by the program.
2. Destructor neither requires any argument nor returns any value therefore it cannot be overloaded.
3. Destructor cannot be declared as static and const;
4. Destructor should be declared in the public section of the program.
5. Destructor is called in the reverse order

of its constructor invocation.

Q: What are the functions that are generated by the compiler by default, if we do not provide them explicitly?

Ans: The functions that are generated by the compiler by default if we do not provide them explicitly are:

- I. Default constructor
- II. Copy constructor
- III. Assignment operator
- IV. Destructor

```
#include <iostream>
```

```
using namespace std;
```

```
int count = 0;
```

```
class Test {
```

```
public:
```

```
    Test()
```

```
{
```

```
    count++;
```

```
    cout << "\n No. of Object created:\t"  
<< count;
```

```
}
```

```
~Test()
```

```
{
```

```
    cout << "\n No. of Object destroyed:\t"  
<< count;
```

```
    --count;
```

```
}
```

```
};
```

```
main()
```

```
{
```

```
    Test t, t1, t2, t3;
```

```
    return 0;
```

```
}
```

Output No. of Object created: 1 No. of Object created: 2 No. of Object created: 3 No. of Object created: 4 No. of Object destroyed: 4 No. of Object destroyed: 3 No. of Object destroyed: 2 No. of Object destroyed: 1

Characteristics of a destructor:-

1. Destructor is invoked automatically by the compiler when its corresponding constructor goes out of scope and releases the memory space that is no longer required by the program.
2. Destructor neither requires any argument nor returns any value therefore it cannot be overloaded.
3. Destructor cannot be declared as static and const;
4. Destructor should be declared in the public section of the program.

5. Destructor is called in the reverse order of its constructor invocation.

Q: What are the functions that are generated by the compiler by default, if we do not provide them explicitly?

Ans: The functions that are generated by the compiler by default if we do not provide them explicitly are:

- I. Default constructor
- II. Copy constructor
- III. Assignment operator
- IV. Destructor