
Week 4:Constructors

By, Mahrukh Khan

Constructor

- It is a special member function of a class that is executed whenever we create new objects of that class .
- It is called constructor because it constructs the values of data members of the class.
- used to initialize the objects of its class at the time of creation.
- There is no need to write any statement to invoke the constructor function, as it is called automatically.
- A constructor that accepts no parameters and that is not defined by a programmer is called the default constructor.

Rules

- Always have a name same as the class name.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and they cannot return values.
- Constructors can have default arguments.
- There can be multiple constructors in a class.
- Normally, constructors are declared public.

Constructor

A **constructor** is a member function of a class that has the same name as the class. A constructor is called automatically when an object of the class is declared. Constructors are used to initialize objects. A constructor must have the same name as the class of which it is a member.

Types Of Constructors

- Default Constructor
- User Define constructor
- Null constructor
- Parameterized Constructors
- Copy Constructor

Default Constructor

A constructor that is defined by the compiler in the absence of user define constructor. A default constructor contains an empty body.

```
#include <iostream>
using namespace std;
class Employee
{
    private:
    int id;
};
int main(void)
{
    Employee e1;
    Employee e2;
    return 0;
}
```

C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\New folder\construct

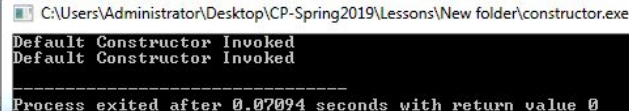
Process exited after 0.02319 seconds with return value
Press any key to continue . . .

User-Defined Constructor

A constructor that is defined by the programmer explicitly for the initializing the data members. It May have an empty body.

```
#include <iostream>
using namespace std;
class Employee
{
public:
    Employee()
    {
        cout<<"Default Constructor Invoked"<<endl;
    }
};

int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```



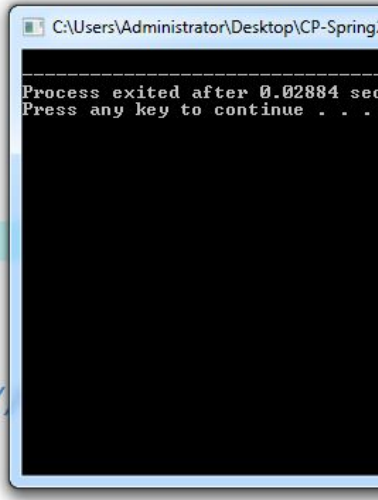
```
C:\Users\Administrator\Desktop\CP-Spring2019\Lessons\New folder\constructor.exe
Default Constructor Invoked
Default Constructor Invoked
-----
Process exited after 0.07094 seconds with return value 0
```

NULL Constructor

A constructor that does nothing. A user defined constructor that has an empty body. Normally written to avoid the call of default constructor.

```
using namespace std;
class Employee
{
    private:
    int id;
    public:
    Employee()
    {
    }
};

int main(void)
{
    Employee e1;
    Employee e2;
    return 0;
}
```

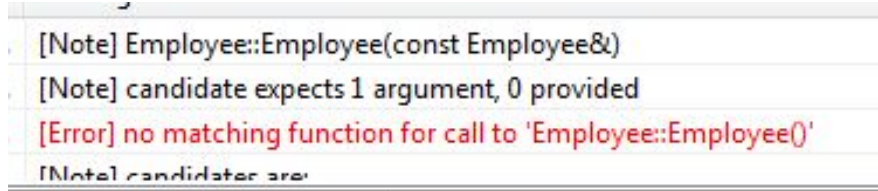


Parameterized Constructor

The constructors that can take arguments just like any other member function.

```
class Employee
{
    private:
    int id;
    public:
    Employee(int idd)
    {
        id=idd;
    }
};

int main(void)
{
    Employee e1;
    Employee e2;
    return 0;
}
```

A screenshot of a compiler error message. The message is displayed in a light gray box with a white background. It contains the following text: [Note] Employee::Employee(const Employee&), [Note] candidate expects 1 argument, 0 provided, [Error] no matching function for call to 'Employee::Employee()', and [Note] candidates are:.

```
[Note] Employee::Employee(const Employee&)\n[Note] candidate expects 1 argument, 0 provided\n[Error] no matching function for call to 'Employee::Employee()'\n[Note] candidates are:
```


Example

```
using namespace std;
class Employee
{
    private:
    int id;
    public:
    Employee(int idd)
    {
        id=idd;
    }
};
int main(void)
{
    Employee e1(2);
    return 0;
}
```

“this” pointer

Every object in C++ has access to its own address through an important pointer called this pointer. The this pointer is an implicit parameter to all member functions. Arrow operator is used to access a member function or member variable of an object through a pointer,

```
class Employee
{
    private:
    int id;
    public:
    Employee(int id)
    {
        this->id=id;
    }
};
```

```
int main(void)
{
    Employee e1(2);
    return 0;
}
```

Example

Consider a class `Movie` that contains information about a movie. The class has the following attributes:

- The movie name
- The MPAA rating (for example, G, PG, PG-13, R)
- The number of people that have rated this movie as a 1 (Terrible)
- The number of people that have rated this movie as a 2 (Bad)
- The number of people that have rated this movie as a 3 (OK)
- The number of people that have rated this movie as a 4 (Good)
- The number of people that have rated this movie as a 5 (Great)

Example

Implement the class with accessor and mutator functions for the movie name and MPAA rating. Write a function `addRating` that takes an integer as an input parameter. The function should verify that the parameter is a number between 1 and 5, and if so, increment the number of people rating the movie that match the input parameter. For example, if 3 is the input parameter, then the number of people that rated the movie as a 3 should be incremented by 1. Write another function, `getAverage`, that returns the average value for all of the movie ratings. Finally, add a constructor that allows the programmer to create the object with a specified name and MPAA rating. The number of people rating the movie should be set to 0 in the constructor.

Test the class by writing a `main` function that creates at least two movie objects, adds at least five ratings for each movie, and outputs the movie name, MPAA rating, and average rating for each movie object.

Copy Constructor

- A constructor that creates a new object with the reference of an existing one. It is a constructor that copies the values of one object into the data members of the other object.
- The copy constructor is used to –
 - Initialize one object from another of the same type.
 - Copy an object to pass it as an argument to a function.
 - Copy an object to return it from a function.
- We cannot pass the argument by value to a copy constructor.
- If we don't define our own copy constructor, the C++ compiler creates a default copy constructor for each class which does a member-wise copy between objects. The compiler created copy constructor works fine in general. We need to define our own copy constructor only if an object has pointers or any runtime allocation of the resource like file handle, a network connection..etc.

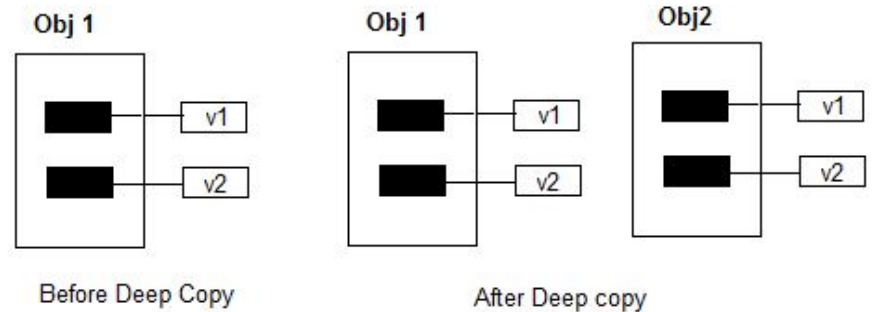
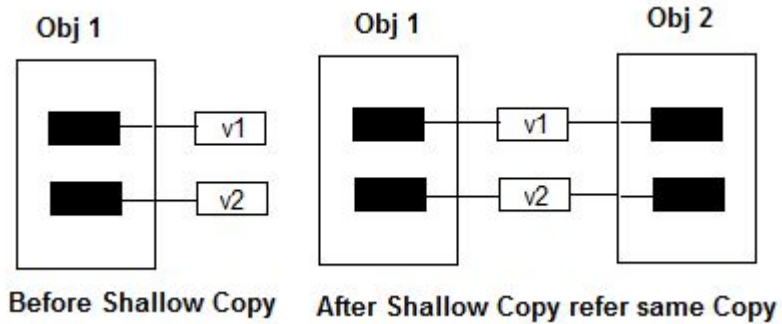
Copy Constructor

- *Default constructor does only shallow copy.*
- When we make a copy constructor private in a class, objects of that class become non-copyable.
- Syntax:

Class_Name (const class_name &object_Name)

- If you **pass** an object by reference, no **copying** happens. If you assign or **pass** an object by value, the object is **copied**. But by default this is a **shallow copy**. The distinction between **shallow copy** and deep **copy** is how they deal with members of the object which are pointers to another object.
-

Shallow Copy/Deep Copy



Example

```
class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p2) {x = p2.x; y = p2.y; }

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

```
int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1;    // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```

```
p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15
-----
Process exited after 0.09953 seconds with
Press any key to continue . . .
```


Constructor outside Class

```
class BankAccount
{
public:
    BankAccount(int dollars, int cents, double rate);
    //Initializes the account balance to $dollars.cents and
    //initializes the interest rate to rate percent.

    void set(int dollars, int cents, double rate);
    void set(int dollars, double rate);
    void update();

    double get_balance();
    double get_rate();
    void output(ostream& outs);
private:
    double balance;
    double interest_rate;
    double fraction(double percent);
};

BankAccount::BankAccount(int dollars, int cents, double rate)
{
    if ((dollars < 0) || (cents < 0) || (rate < 0))
    {
        cout << "Illegal values for money or interest rate.\n";
        exit(1);
    }
    balance = dollars + 0.01*cents;
    interest_rate = rate;
}
```

Function Overloading

C++ allows you to give two or more different definitions to the same function name, which means you can reuse names that have strong intuitive appeal across a variety of situations.

```
double ave(double n1, double n2)
{
    return ((n1 + n2)/2.0);
}
```

```
double ave3(double n1, double n2, double n3)
{
    return ((n1 + n2 + n3)/3.0);
}
```

Function Overloading

Overloading a Function Name

If you have two or more function definitions for the same function name, that is called **overloading**. When you overload a function name, the function definitions must have different numbers of formal parameters or some formal parameters of different types. When there is a function call, the compiler uses the function definition whose number of formal parameters and types of formal parameters match the arguments in the function call.

```
int main( )  
{  
    using namespace std;  
    cout << "The average of 2.0, 2.5, and 3.0 is "  
        << ave(2.0, 2.5, 3.0) << endl;  
  
    cout << "The average of 4.5 and 5.5 is "  
        << ave(4.5, 5.5) << endl;  
  
    return 0;  
}
```

two arguments



Overloading

Class Addition

```
{  
    int add(int a, int b)  
    {  
        return a+b;  
    }  
    int add(int a, int b, int c)  
    {  
        return a+b+c;  
    }  
}
```

Constructor Overloading

In C++, We can have more than one constructor in a class with same name, as long as each has a different list of arguments. This concept is known as Constructor Overloading and is quite similar to function overloading.

- Overloaded constructors essentially have the same name (name of the class) and different number of arguments.
- A constructor is called depending upon the number and type of arguments passed.
- While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

Member Initializers

This feature allows you to set default values for member variables. When an object is created the member variables are automatically initialized to the specified values.

```
class Coordinate
{
    public:
        Coordinate();
        Coordinate(int x);
        Coordinate(int x, int y);
        int getX();
        int getY();
    private:
        int x=1;
        int y=2;
};
```

```
Coordinate::Coordinate()
{ }
Coordinate::Coordinate(int xval) : x(xval)
{ }
Coordinate::Coordinate(int xval, int yval) : x(xval), y(yval)
{ }
int Coordinate::getX()
{
    return x;
}
int Coordinate::getY()
{
    return y;
}
```

Destructor

- A destructor is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.
- A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.
- A destructor takes no arguments and has no return type.

```
class X {  
public:  
    // Constructor for class X  
    X();  
    // Destructor for class X  
    ~X();  
};
```

Example

```
class Line {
public:
    void setLength( double len );
    double getLength( void );
    Line();    // This is the constructor declaration
    ~Line();   // This is the destructor: declaration

private:
    double length;
};

// Member functions definitions including constructor
Line::Line(void) {
    cout << "Object is being created" << endl;
}
Line::~~Line(void) {
    cout << "Object is being deleted" << endl;
}
void Line::setLength( double len ) {
    length = len;
}
double Line::getLength( void ) {
    return length;
}
```

```
int main() {
    Line line;

    // set line length
    line.setLength(6.0);
    cout << "Length of line : " << line.getLength() << endl;

    return 0;
}
```

```
Object is being created
Length of line : 6
Object is being deleted
```


Homework:Reading Assignment

- Difference Between Copy Constructor and Assignment Operator in C++