Constructors

Constructors

Constructors are special class functions which performs initialization of every object.

The Compiler calls the Constructor whenever an object is created.

Constructors initialize values to object members after storage is allocated to the object.

```
class A
{
        int x;
        public:
            A(); //Constructor
};
```

The name of constructor will be same as the name of the class, and contructors never have return type.

- Constructors can be defined either inside the class definition or outside class definition using class name and scope resolution operator

```
class A
 {
        int i;
        public:
                A(); //Constructor declared
};

        A::A() // Constructor definition
{
        i=1;
}
```

Types of Constructors

Constructors are of three types :

1. Default Constructor
2. Parametrized Constructor
3. Copy Constructor

```
class_name ()
{
            Constructor Definition
 }
```

```
class Cube
 {
          int side;
          public:
         Cube()
          {
                   side=10;
          }
};
int main()
{
          Cube c;
          cout << c.side;
 }
```

A *default constructor* is a constructor that either has no parameters, or if it has parameters, *all* the parameters have default values.

```
class Cube
 {
 public:
              int side;
};

int main()
{
       Cube c;
        cout << c.side;
 }
```

A default constructor is so important for initialization of object members, that even if we do not define a constructor explicitly, the compiler will provide a default constructor implicitly.

Parameterized Constructor

These are the constructors with parameter.

Using this Constructor you can provide different values to data members of different objects, by passing the appropriate values as argument.

```
class Cube
 {
        public: int side;
         Cube(int x)
        {
                side=x;
        }
 };
 int main()
{
      Cube c1(10);
      Cube c2(20);
      Cube c3(30);
      cout << c1.side;
      cout << c2.side;
      cout << c3.side;
 }
```

```
class Line
 {
 public:
              void setLength( double len );
              double getLength( void );
              Line();  // This is the constructor
 private:
              double length;
};
Line::Line(void)
{
       cout << "Object is being created" << endl;
}
void Line::setLength( double len )
 {
              length = len;
}
double Line::getLength( void )
{
              return length;
}
```

```
// Main function for the program
int main()
{
        Line line;
        line.setLength(6.0);
        cout << "Length of line : " << line.getLength() <<endl;
        return 0;
}
```

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

```
class Line
{
public:
          void setLength( double len );
          double getLength( void );
          Line(double len);  // This is the constructor
 private:
              double length;
};        // Member functions definitions 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;
}
```

```
Object is being created, length = 10 Length of line : 10 Length of line : 6
```

Copy Constructor

Just like other member functions, constructors can also be overloaded.

Both default and parameterized constructors defined in your class you are having Overloaded Constructors, one with no parameter and other with parameter.

```
class Student
 {
            int rollno;
            string name;
            public: Student(int x)
            {
                        rollno=x;
                        name="None";
            }
Student(int x, string str)
 {
            rollno=x ;
            name=str ;
}


 };
```

```
int main()
 {
            Student A(10);
            Student B(11,"Ram");
}
```

Student S; in **main()**,

if you define any constructor explicitly, then the compiler will not provide default constructor and you will have to define it yourself.

Copy Constructor

which takes an object as argument, and is used to copy values of data members of one object into other object.

ClassName (const ClassName &old_obj);

```
class Samplecopyconstructor
{
private:  int x, y;
 public:
        Samplecopyconstructor(int x1, int y1)
        {
              x = x1; y = y1;
       }
 /* Copy constructor */
        Samplecopyconstructor (const Samplecopyconstructor &sam)
       {
              x = sam.x; y = sam.y;
       }
        void display()
       {
              cout<<x<<" "<<y<<endl;
        }
};
```

```
/* main function */
 int main()
 {
        Samplecopyconstructor obj1(10, 15); // Normal constructor
         Samplecopyconstructor obj2 = obj1; // Copy constructor
        cout<<"Normal constructor : "; obj1.display();
        cout<<"Copy constructor : "; obj2.display();
        return 0;
}
```

Destructors

Destructor is a special class function which destroys the object as soon as the scope of object ends.

The destructor is called automatically by the compiler when the object goes out of scope.

The syntax for destructor is same as that for the constructor, the class name is used for the name of destructor, with a **tilde** sign as prefix to it.

```
class A
{
                public:
                 ~A();
};
```

```
class A {
            public:
             A()
            {
                        cout << "Constructor called";
            }
             ~A()
             {
                        cout << "Destructor called";
            }
};
int main()
             {
                        A  obj1; // Constructor Called
                         int x=1
            if(x)
            {
                        A obj2; // Constructor Called
            }
// Destructor Called for obj2
 }
 // Destructor called for obj1
 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; }
```

```
// Main function for the program
int main()
{
		Line line;
		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
```

Using Initialization Lists to Initialize Fields

In case of parameterized constructor,

```
Line::Line( double len): length(len)
 {
            cout << "Object is being created, length = " << len << endl;
 }



Line::Line( double len)
{
      cout << "Object is being created, length = " << len << endl;
            length = len;
}
```

```
C::C( double a, double b, double c): X(a), Y(b), Z(c)
 {
            ....
}
```

Dynamic constructors

- Constructors too, can allot memories to objects at run time and prevent unnecessary memory wastage.

- A constructor allocating dynamic memory using new operator is called as ***dynamic constructors.***

Example:

```
class DynamicCon
{
        int * ptr;
        public:
        DynamicCon()
        {
          ptr=new int;
         *ptr=100;
        }
        DynamicCon(int v)
        {
          ptr=new int;
         *ptr=v;
        }

         int display()
        {
          return(*ptr);
        }

 };

 void main()
 {

        DynamicCon obj1, obj2(90);
        cout<<"\nThe value of obj1's ptr  is:";
        cout<<obj1.display();
        cout<<"\nThe value of obj2's ptr is:"<<endl;
        cout<<obj2.display();
 }
```

```
The value of obj1's ptr is:100
The value of obj2's ptr is: 90
```

Function Overloading

You can have multiple definitions for the same function name in the same scope.

```
class printData
{
 public: void print(int i)
{
cout << "Printing int: " << i << endl;
 }
void print(double f)
 {
cout << "Printing float: " << f << endl;
 }
void print(char* c)
 {
cout << "Printing character: " << c << endl; } };
```

```
int main(void)
 {
 printData pd; // Call print to print integer
pd.print(5); // Call print to print float
pd.print(500.263); // Call print to print character
pd.print("Hello C++");
 return 0; }
```

```
Printing int: 5
Printing float: 500.263
Printing character: Hello C++
```