C++ CLASS CONSTRUCTOR AND DESTRUCTOR

The Class Constructor:

A class **constructor** is a special member function of a class that is executed whenever we create new objects of that class.

A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Following example explains the concept of constructor:

```
#include <iostream>

using namespace std;

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

   private:
      double length;
};

// Member functions definitions including constructor
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;

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

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

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

Parameterized Constructor:

A default constructor does not have any parameter, but if you need, a constructor can have parameters. This helps you to assign initial value to an object at the time of its creation as shown in the following example:

```
#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 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;
}
```

When the above code is compiled and executed, it produces the following result:

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

Using Initialization Lists to Initialize Fields:

In case of parameterized constructor, you can use following syntax to initialize the fields:

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

Above syntax is equal to the following syntax:

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

If for a class C, you have multiple fields X, Y, Z, etc., to be initialized, then use can use same syntax and separate the fields by comma as follows:

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

The Class 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.

Following example explains the concept of destructor:

```
#include <iostream>

using namespace std;

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;

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

   return 0;
}
```

When the above code is compiled and executed, it produces the following result:

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