OOPS

Constructors

When applied to real problems, virtually every object you create will require some sort of ***initialization***. C++ allows a constructor function to be included in a class declaration. A class's constructor is called each time an object of that class is created. Thus, any initialization to be performed on an object can be done automatically by the constructor function. A constructor function has the ***same name as the class*** of which it is a part and does not have return type. Here is a short example,

```
#include <iostream>
using namespace std;
// Class Declaration
class myclass {
   int a;

  public:
   // constructor
   myclass();
   void show();
};
myclass::myclass() {
   cout << "In constructor\n";
   a = 10;
}

void myclass::show() { cout << a; }

int main() {
   // Automatic call to constructor
   myclass ob;
   ob.show();
   return 0;
}
```

Output

```
In constructor
10
```

In this simple example the constructor is called when the object is created, and the constructor initializes the private variable a to 10.

For a global object, its constructor is called once, when the program first begins execution. For local objects, the constructor is called each time the declaration statement is executed.

Destructors

The complement of a constructor is the ***destructor***. This function is called when an object is destroyed.

For example, an object that allocates memory when it is created will want to free that memory when it is destroyed. The name of a destructor is the **name of its class preceded by a** ~. For example,

```
#include <iostream>
using namespace std;
// Class Declaration
class myclass {
   int a;

  public:
   // constructor
   myclass();
   // Destructor
   ~myclass();

   void show();
};

myclass::myclass() {
   cout << "In constructor\n";
   a = 10;
}

myclass::~myclass() { cout << "In Destructor\n"; }

void myclass::show() { cout << a << "\n"; }

int main() {
   // Automatic call to constructor
   myclass ob;
   ob.show();
   return 0;
}
```

Output

```
In constructor
10
In Destructor
```

A class's destructor is called when an object is destroyed. Local objects are destroyed when they go ***out of scope***.

Global objects are destroyed when the ***program ends***. It is not possible to take the address of either a constructor or a destructor.

Constructors that take parameters. It is possible to pass one or more arguments to a constructor function. Simply add the appropriate parameters to the constructor function's declaration and definition. Then, when you declare an object, specify the arguments.

```
#include <iostream>
using namespace std;
// Class Declaration
class myclass {
   int a;

  public:
   // constructor
   myclass(int aa);
   // Destructor
   ~myclass();
   void show();
};

myclass::myclass(int aa) {
   cout << "In constructor\n";
   a = aa;
}

myclass::~myclass() { cout << "In Destructor\n"; }

void myclass::show() { cout << a << "\n"; }

int main() {
   // Automatic call to constructor
   myclass ob(100);
   ob.show();
   return 0;
}
```

Output:

```
In constructor
100
In Destructor
```

Pay particular attention to how ***ob*** is declared in ***main( )***. The value 100, specified in the parentheses following ***ob***, is the argument that is passed to myclass( )'s parameter x that is used to initialise a.

Actually, the syntax is shorthand for this longer form: ***myclass ob* = *myclass(4);***

Unlike constructor functions, ***destructors cannot have parameters***.

Although the previous example has used a constant value, you can pass an object's constructor any valid expression, including variables.