Lecture 5

Types of constructor:-

There three types of constructor:

(i) Default Constructor

(ii) Parameterized Constructor

(iii) Copy constructor

The constructor which has no arguments is known as default constructor.

Demonstration of default Constructor

```
#include <iostream>

#include <conio>

class Add

{

int x, y, z;

public:

Add();

void calculate(void);

void display(void);

};

Add::Add()

{
```

```
x=6;
y=5;
}
void Add :: calculate()
{
z=x+y;
}
void Add :: display()
{
cout<<z;
}
void main()
{
Add a;
a.calculate();
a.display();
getch();
}
```

Note: Here in the above program when the statement Add a; will execute (i.e. object is created), the default constructor Add () will be called automatically and value of x and y will be set to 6 and 5 respectively.

Parameterized constructor

The constructor which takes some argument is known as parameterized constructor.

Ex: - Write a program to initialize two integer variables using parameterized constructor and add them.

```
#include <iostream>
#include <conio>
class Add
{
int x, y, z;
public:
Add(int, int);
void calculate(void);
void display(void);
};
Add :: Add(int a, int b)
{
x=a;
y=b;
}
void Add :: calculate()
{
z=x+y;
}
void Add :: display()
```

```
{
cout<<z;
}
void main()
{
Add a(5, 6);
a.calculate();
a.display();
getch();
}
```

Note: Here in the above program when the statement Add a(5, 6); will be executed (i.e. object creation), the parameterized constructor Add (int, int) will be called automatically and value of x and y will be set to 5 and 6respectively.

A parameterized constructor can be called:

(i) Implicitly: Add a(5, 6);

(ii) Explicitly :Add a=Add(5, 6);

If the constructor has one argument, then we can also use object-name=value-of-argument; instead of object-name (value-of-argument); to initialize an object.

Copy Constructor

The constructor which takes reference to its own class as argument is known as copy constructor.

Ex:-Write a program to initialize two integer variables using parameterized constructor. Copy given integers into a new object and add them.

```
#include <iostream>
#include <conio>
class Add
```

```
{
int x, y, z;
public:
Add()
{
}
Add(int a, int b)
{
x=a;
y=b;

}
Add(Add &);
void calculate(void);
void display(void);
};
Add :: Add(Add &p)
{
x=p.x;
y=p.y;
cout<<"Value of x and y for new object: ”<<x<<” and "<<y<<endl;
}
void Add :: calculate()
{
z=x+y;
}
void Add :: display()
```

```
{
cout<<z;
}
void main()
{
Add a(5, 6);
Add b(a);
b.calculate();
b.display();
getch();
}
```

Note: Here in the above program when the statement Add a(5, 6); will execute (i.e. object creation), the parameterized constructor Add (int, int) will be called automatically and value of x and y will be set to 5 and 6respectively. Now when the statement Add b(a) ; will execute, the copy constructor Add(Add&) will be called and the content of object a will be copied into object b.

What is Constructor Overloading?

If a program contains more than one constructor, then constructor is said to be overloaded.

Demonstration of Destructor

```
#include <iostream>
#include <conio>
class XYZ
{
```

```
int x;
public:
XYZ( );
~XYZ( );
void display(void);
};
XYZ::XYZ( )
{
x=9;
}
XYZ:: ~XYZ( )
{
cout<<"Object is destroyed"<<endl;
}
void XYZ::display()
{
cout<<x;
}
void main()
```

```
{

XYZ xyz;

xyz.display();

getch();

}
```

Static members

A class can contain *static* members, either data or functions.

Static data members of a class are also known as "class variables", because there is only one unique value for all the objects of that same class. Their content is not different from one object of this class to another.

For example, it may be used for a variable within a class that can contain a counter with the number of objects of

that class that are currently allocated, as in the following example:

```
#include <iostream>

#include <conio>

class CDummy {

public:

static int n;

CDummy () { n++; };

~CDummy () { n--; };

};
```

```
int CDummy::n=0;

 main () {

CDummy a;

CDummy b[5];

CDummy * c = new CDummy;

cout << a.n << endl;

delete c;

cout << CDummy::n << endl;

getch();

}
```

In fact, static members have the same properties as global variables but they enjoy class scope. For that reason, and to avoid them to be declared several times, we can only include the prototype (its declaration) in the class declaration but not its definition (its initialization). In order to initialize a static data-member we must include a formal definition outside the class, in the global scope, as in the previous example:

```
int CDummy::n=0;
```

Because it is a unique variable value for all the objects of the same class, it can be referred to as a member of any object of that class or even directly by the class name (of course this is only valid for static members):

```
cout << a.n;
cout << CDummy::n;
```

These two calls included in the previous example are referring to the same variable: the static variable n within class CDummy shared by all objects of this class.

Once again, I remind you that in fact it is a global variable. The only difference is its name and possible access restrictions outside its class.

Just as we may include static data within a class, we can also include static functions. They represent the same: they are global functions that are called as if they were object members of a given class. They can only refer to static data, in no case to non-static members of the class, as well as they do not allow the use of the keyword this, since it makes reference to an object pointer and these functions in fact are not members of any object but direct members of the class.