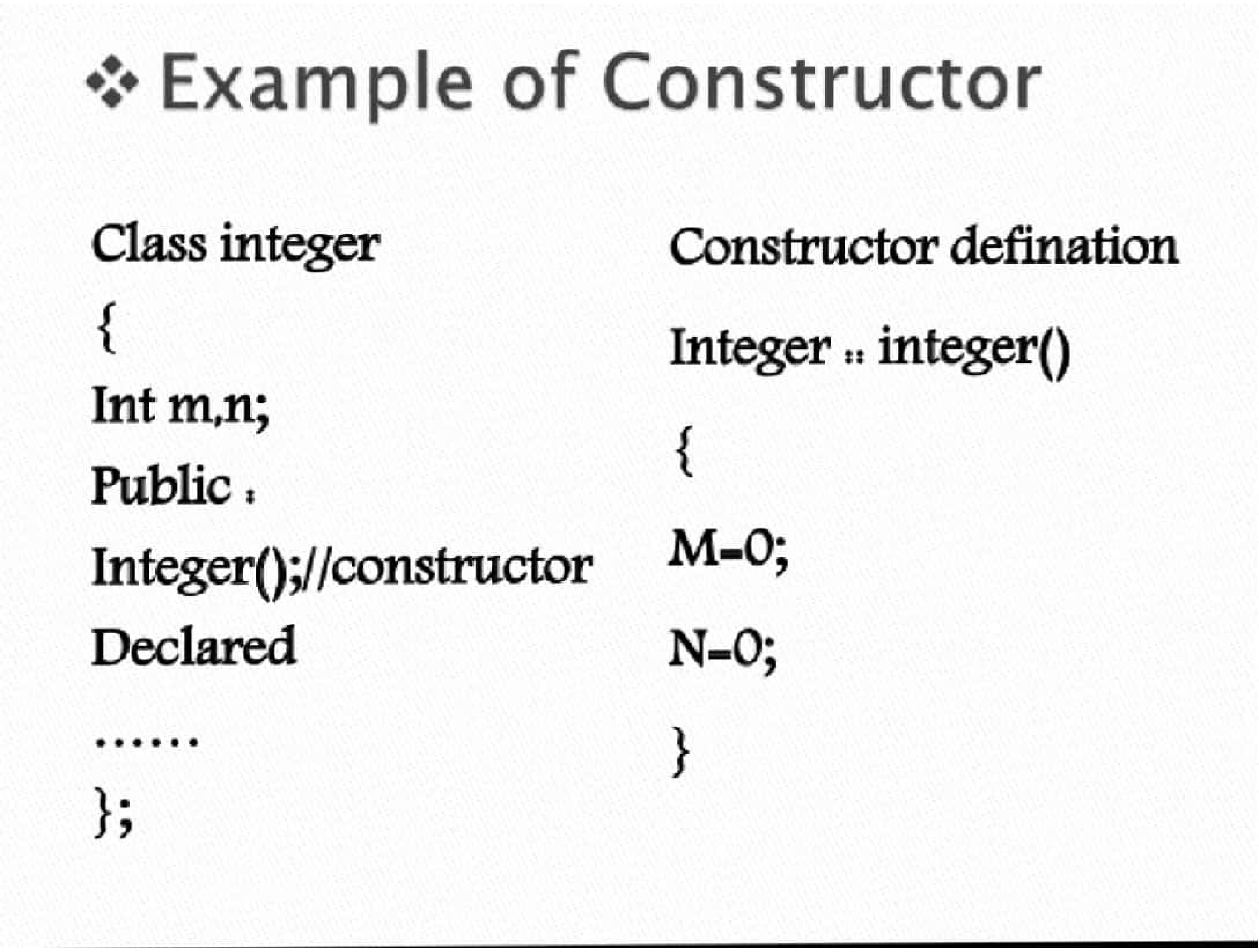
Constructor and Destructor

❖ INDEX

❖ Constructor

1. Special member function to initialize
2. The objects of its class.
3. Its name is same as the class name.
4. It is invoked whenever the object is created.



❖ Example of Constructor

```
Class integer
{
Int m,n;
Public :
Integer();//constructor
Declared
......
};
```

Constructor defination

```
Integer :: integer()
{
M=0;
N=0;
}
```

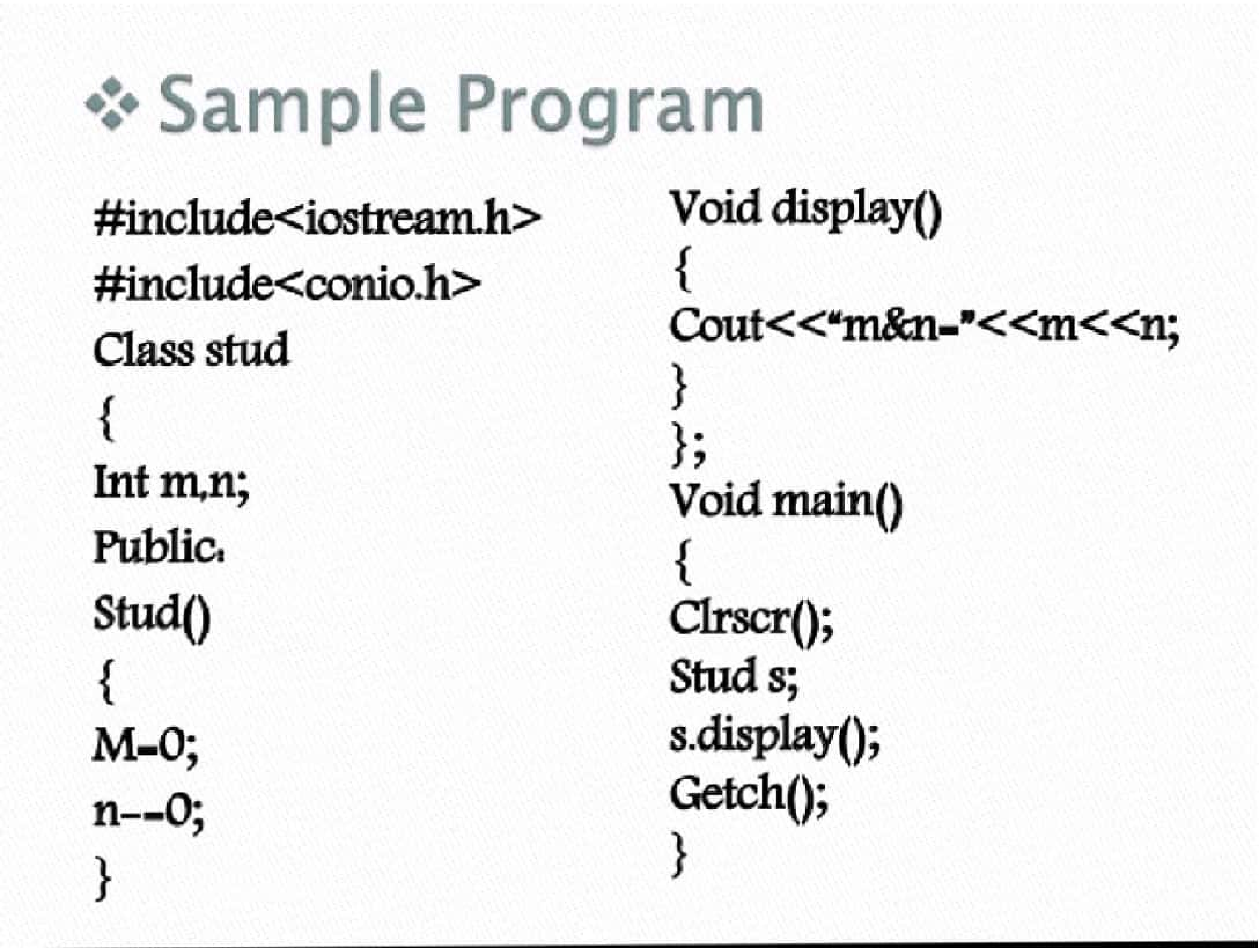
Scanned by CamScanner

❖ Characteristics of Constructor

1. They should be declared in the public section.
2. They are called automatically when the object are created.
3. They do not have return type even void
4. They have same name as the class name.

❖ Default Constructor

1. **They takes no parameters.**
2. **They are called internally by the compiler whenever the object are created.**
3. **There is no need to call it explicitly**

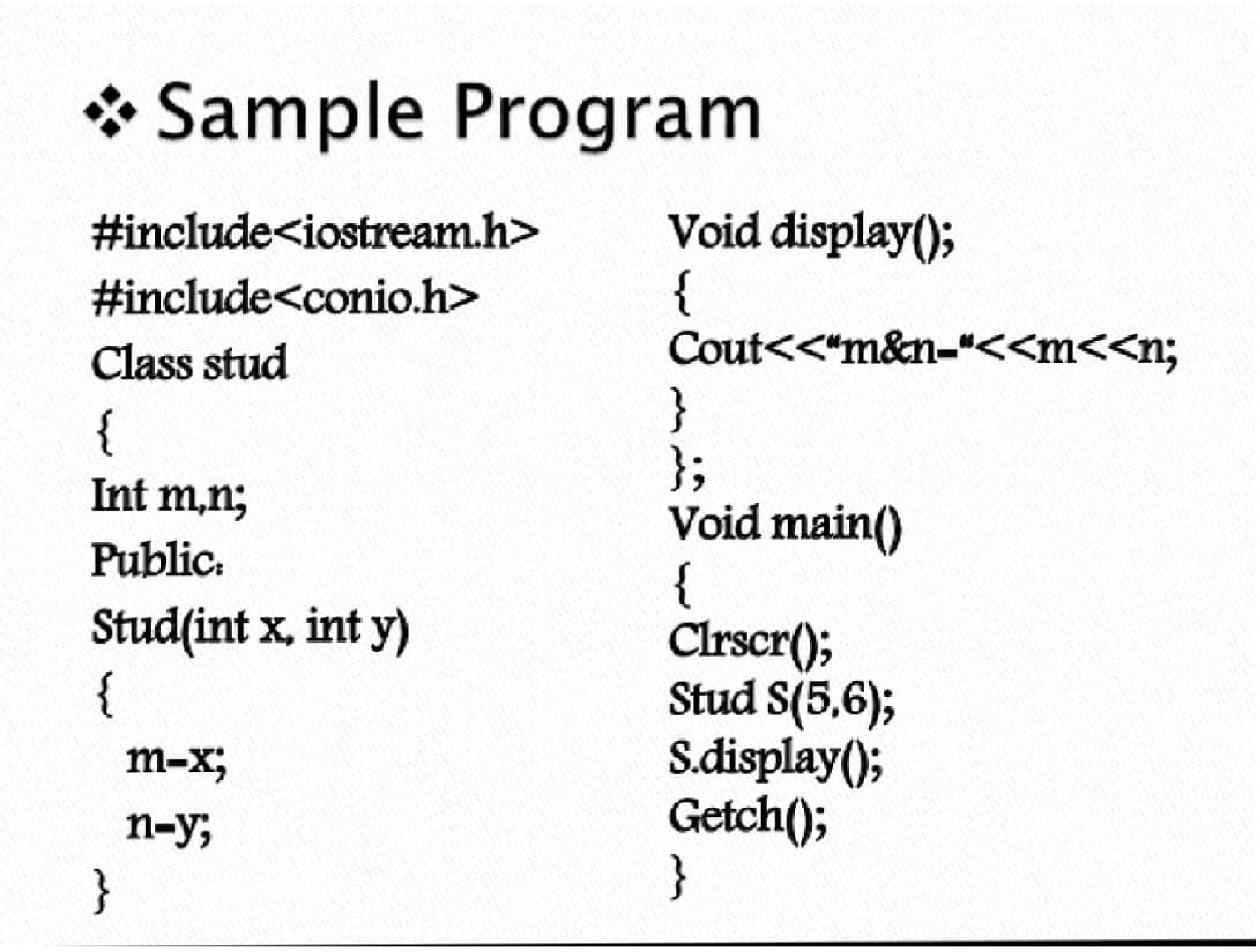


❖ Sample Program

```
#include<iostream.h>
#include<conio.h>
Class stud
{
Int m,n;
Public.
Stud()
{
M-0;
n--0;
}
Void display()
{
Cout<<"m&n-"<<m<<n;
}
};
Void main()
{
Clrscr();
Stud s;
s.display();
Getch();
}
```

❖ Parameterized Constructor

1. These are the constructor that take arguments.
2. They initialized the object data members by the value which is passed as arguments.
3. They are invoked when we pass the arguments to the object when they are being defined.
4. Example. integer int1(2,5);

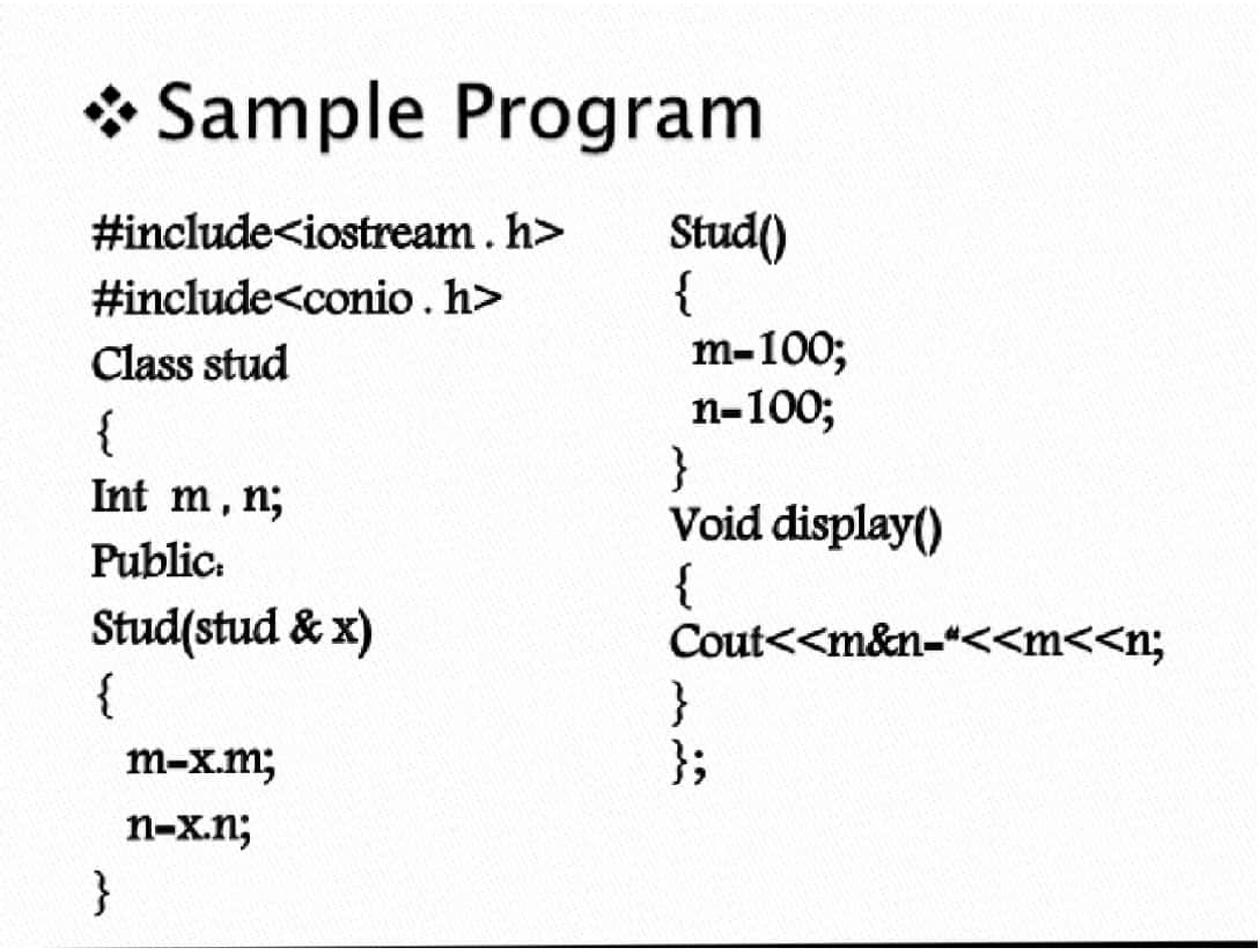


❖ Sample Program

```
#include<iostream.h>
#include<conio.h>
Class stud
{
Int m,n;
Public:
Stud(int x, int y)
{
  m-x;
  n-y;
}
Void display();
{
Cout<<"m&n-"<<m<<n;
}
};
Void main()
{
Clrscr();
Stud S(5,6);
S.display();
Getch();
}
```

❖ Copy Constructor

1. It is used to declare and initialized an object from another object.
2. It takes reference to an object of the same class as itself as an arguments.



❖ Sample Program

```
#include<iostream . h>
#include<conio . h>
Class stud
{
Int  m , n;
Public:
Stud(stud & x)
{
  m-x.m;
  n-x.n;
}
Stud()
{
 m-100;
 n-100;
}
Void display()
{
Cout<<m&n-“<<m<<n;
}
};
```

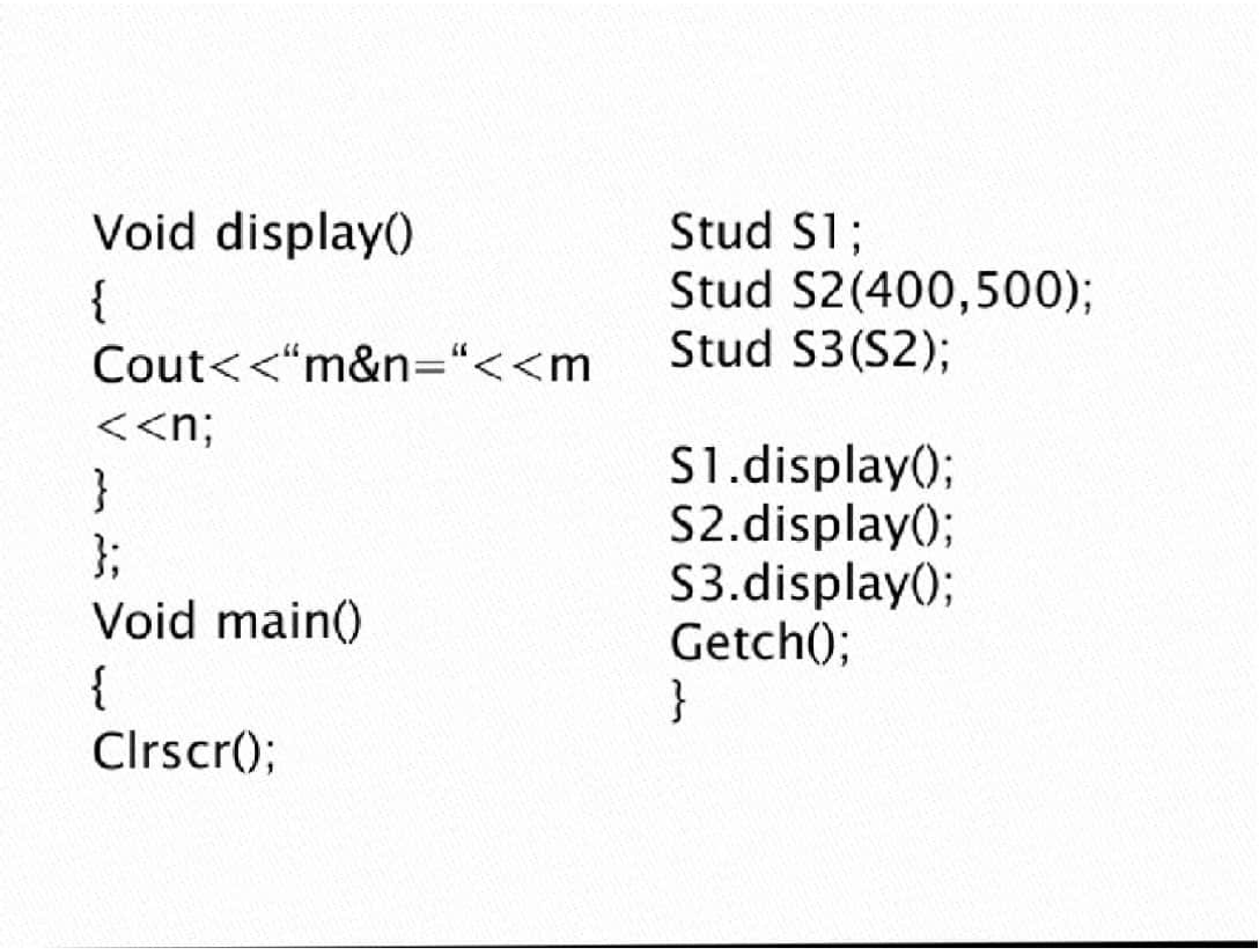
```
void main()
{
   clrscr(); integer
   int1;
   int1.display();
   integer
   int2(int1);
   int2.display();
   getch();
}
```

❖ Overloading Constructor

1. Constructor overloading is the process of defining more than one constructor in the same class.
2. C++ permits us to use multiple constructor in the same class.

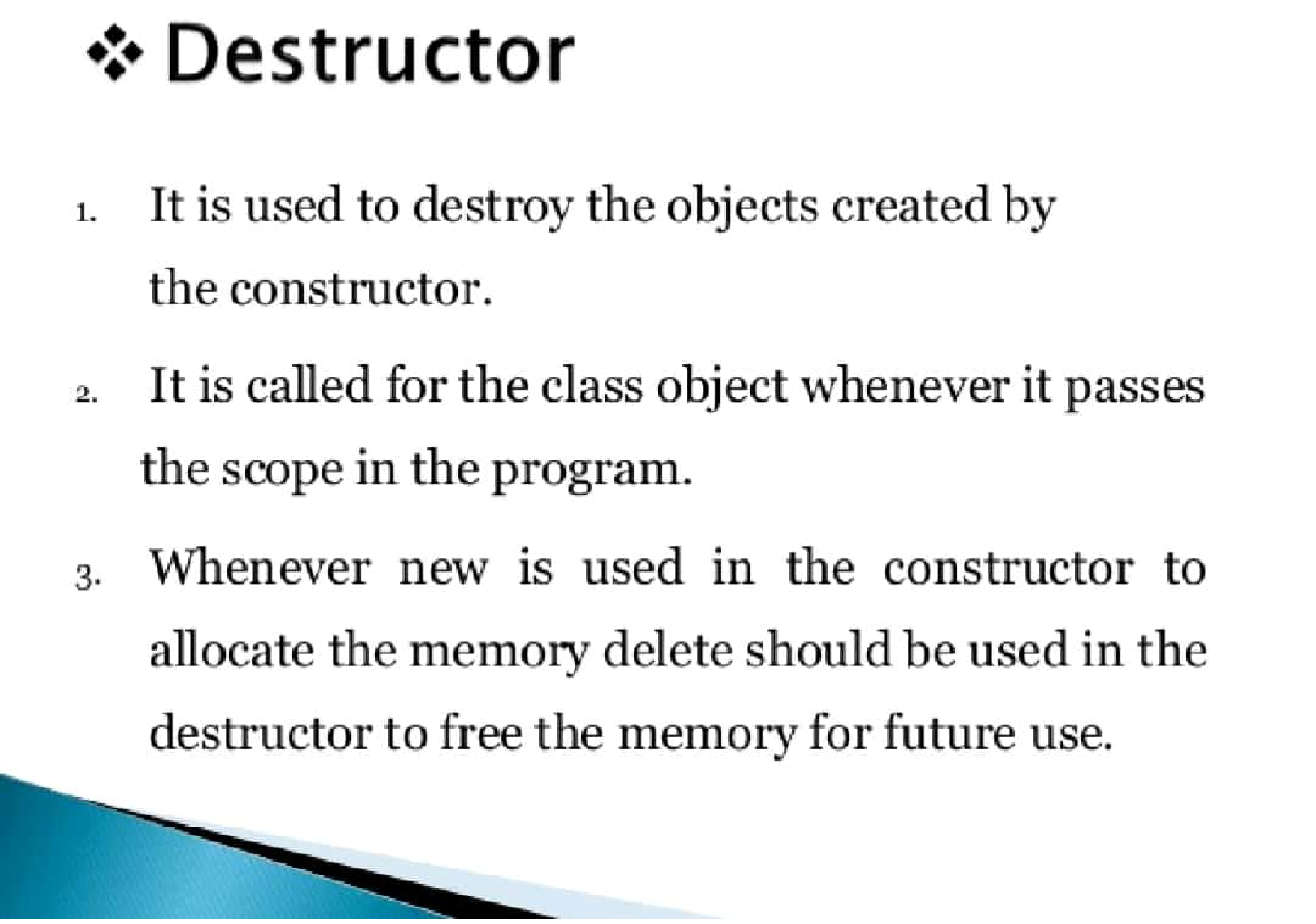
❖ Sample Program

```
#include<iostream.h>
#include<conio.h>
Class stud
{
Int m,n;
Public:
Stud(stud&x)
{
  m=x.m;
  n=x.n;
}
Stud()
{
   m=0;
   n=0;
}
Stud(int x , int y);
{
    m=x;
    n=y;
}
```



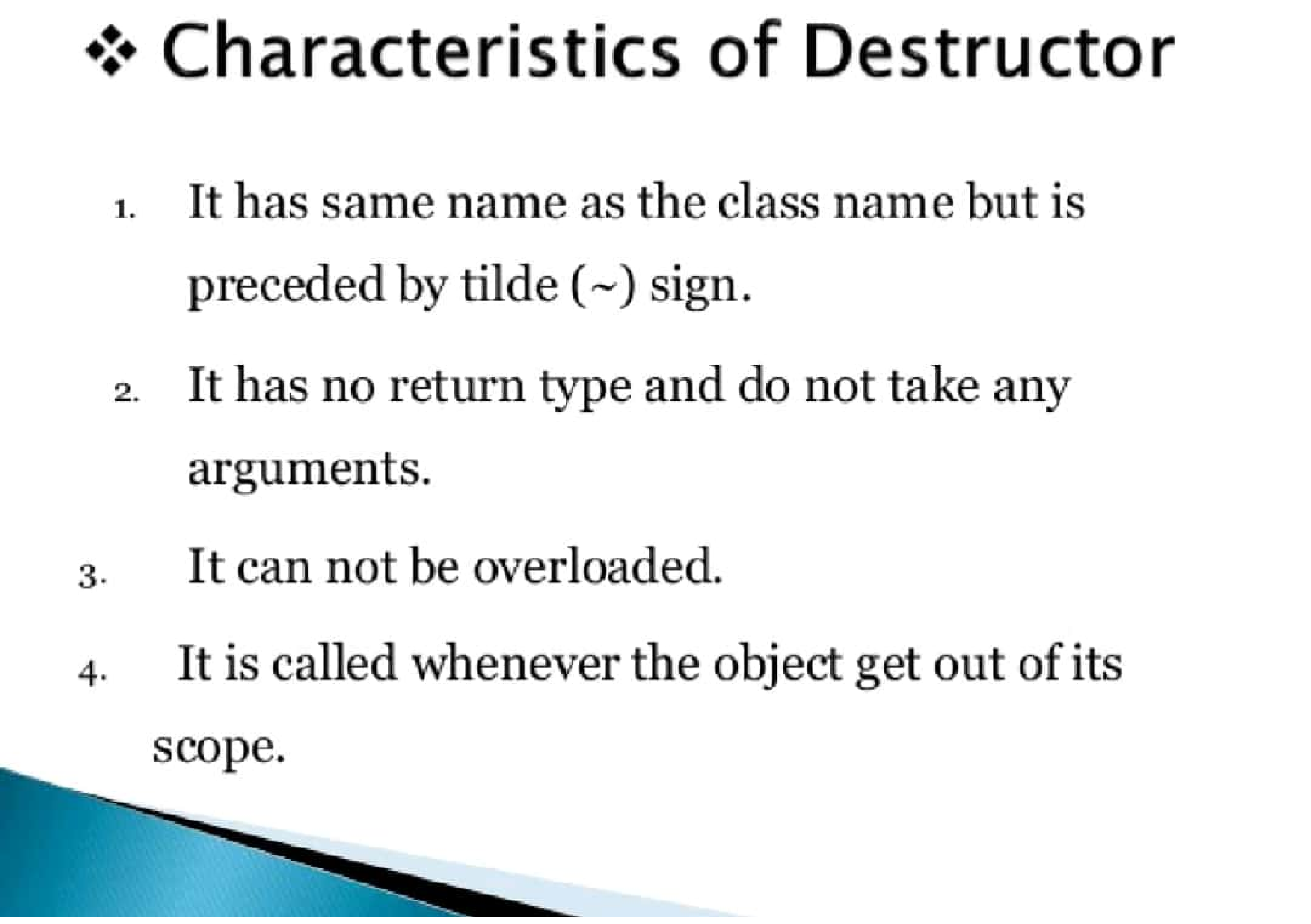
```
Void display()
{
Cout<<"m&n="<<m
<<n;
}
};
Void main()
{
Clrscr();
Stud S1;
Stud S2(400,500);
Stud S3(S2);

S1.display();
S2.display();
S3.display();
Getch();
}
```



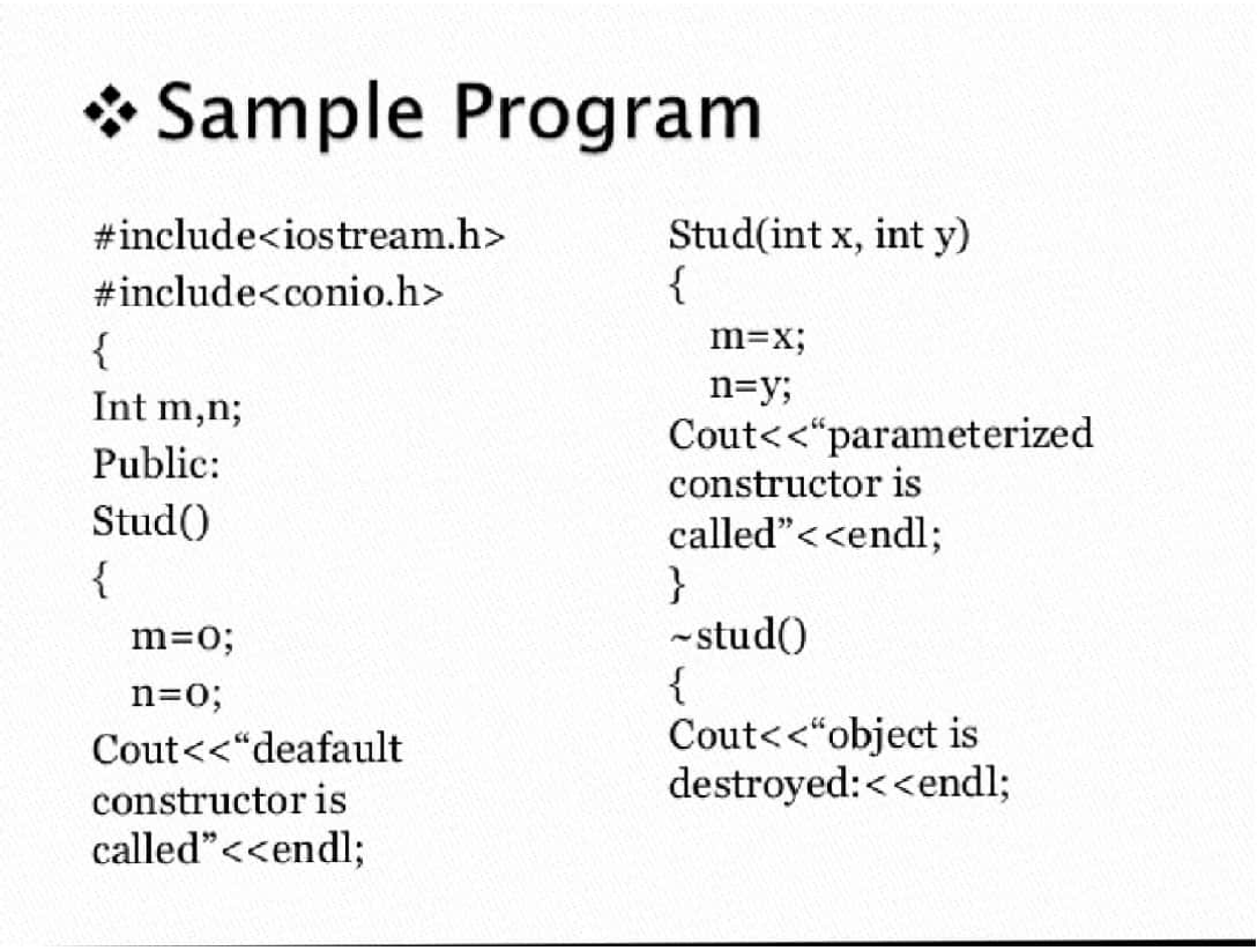
❖ Destructor

1. It is used to destroy the objects created by the constructor.
2. It is called for the class object whenever it passes the scope in the program.
3. Whenever new is used in the constructor to allocate the memory delete should be used in the destructor to free the memory for future use.



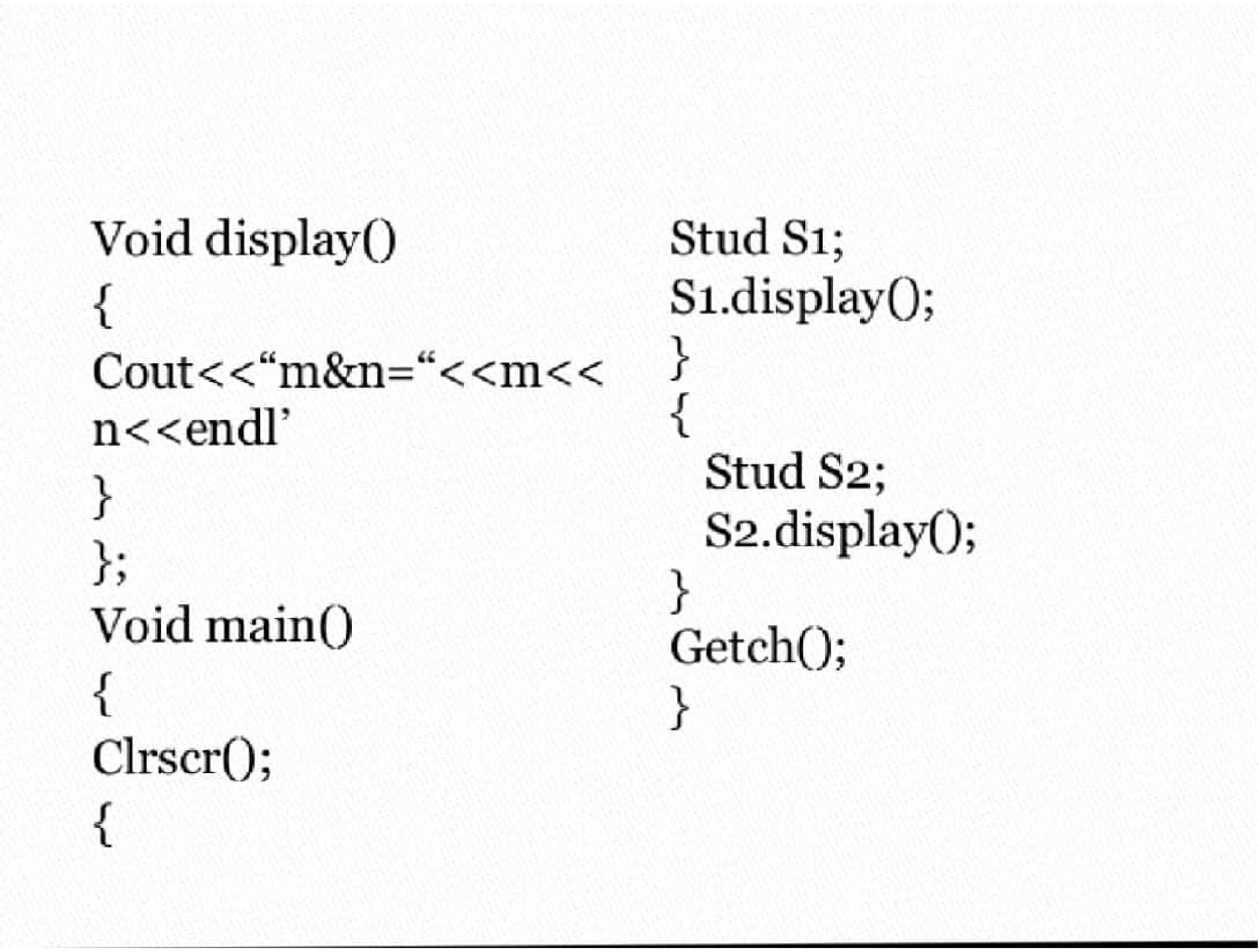
❖ Characteristics of Destructor

1. It has same name as the class name but is preceded by tilde (~) sign.
2. It has no return type and do not take any arguments.
3. It can not be overloaded.
4. It is called whenever the object get out of its scope.



❖ Sample Program

```
#include<iostream.h>
#include<conio.h>
{
Int m,n;
Public:
Stud()
{
   m=0;
   n=0;
Cout<<"deafault
constructor is
called"<<endl;
Stud(int x, int y)
{
   m=x;
   n=y;
Cout<<"parameterized
constructor is
called"<<endl;
}
~stud()
{
Cout<<"object is
destroyed:<<endl;
```



```
Void display()
{
Cout<<"m&n="<<m<<
n<<endl'
}
};
Void main()
{
Clrscr();
{
Stud S1;
S1.display();
}
{
  Stud S2;
  S2.display();
}
Getch();
}
```