Lecture 4 and 5

constructor

- A constructor is a special member function used to initialize the objects of its class .
- Name same as the class name.
- The constructor is invoked whenever an object is created. It is called as constructor because it constructs the values of data members of the class.
- No return type for constructor , not even void
- Always it should be declared in public
- Invoked automatically when objects are created. No need to write separate statements to invoke constructor

A constructor is declared and defined as follows:

```
class integer
{
    int m,n;
    public:
    integer (void)      //constructor
    {
    m=3;
    n=2;
    }
};
```

OR

```
class integer
{
    int m,n:
    public:
    integer (void);     //constructor
};
integer::integer(void)
{
m=3;
n=2;
}
```

Types of constructor

- Default constructor
- Parameterized constructor
- Copy constructor

Default constructor

- Constructor without any argument
- If the programmer is not providing any default constructor, compiler will supply the default constructor.

```
class integer
{
        int m,n;
        public:
        integer (void)        //constructor
        {
        m=3;
        n=2;
        }
};
int main()
{  integer I1;
}
```

It is not only initializing always, if we want to read values from user, that can also be written inside the constructor so that a separate read function can be omitted. But usually it is used to initialize the data members.

```
class integer
{
    int m,n;
    public:
    integer (void)        //constructor
    {
    cin>>m>>n;
    }
};
int main()
{  integer I1;
}
```

Parameterized constructor

- Constructor with arguments

```
class integer
{
      int m,n;
      public:
      integer( int x, int y)
      {
       m=x;
       n=y;
      }
};
```

```
int main()
{
integer I1(0,100); //implicit call

integer I1 = integer(0,100); // explicit call

}
```

If a parameterized or copy constructor is used and If the programmer is not providing any default constructor, **compiler will not supply the default constructor**

- When a class with multiple constructors with multiple arguments are present, it is called constructor overloading.

- Do nothing constructor is a default constructor without any body ie not doing anything. Then why it has to be given?
  If the programmer is writing any parameterized or copy constructor, then compiler wont provide any default constructor.

Can have any no. of parametrized constructor

```
class integer
{
        int m,n;
        public:
        Integer(){}   // do nothing constructor
        integer( int x, int y)
        {
         m=x;
         n=y;
        }
        Integer(int x)
        { m=n=x;}
};
```

```
Int main()
{
   integer I1;  // default do nothing constructor
   integer I1(3,4); // calls first parameterized constructor
   integer I1(2); // calls 2nd parameterized constructor
}
```

Copy constructor- copy the value of one object to another

- Constructor can’t have argument as same type object.
- But it accepts reference to its own class as argument

```
class code
{
      int x;
      public:
      code(){cout<<“default”;}
      code(int a)
         {cout<<“parameterized”<<endl;
         x=a;}
      code(code &c1)//pass by reference
        {cout<<“copy”<<endl;
         x= c1.x;}
      void disp(){cout<<x;}
};
```

```
main()
{
 code A(10); // calls parameterized constructor
 code B(A);    // calls copy constructor
 A.disp();
 B.disp();
 code C=A;  // calls copy constructor
 C.disp();
 code D;    // default constructor
 D=A;  // wont call copy constructor
 D.disp();
code E();  // wont call anything or wont create any obj
}
```

- Pass by reference- if a variable or object is passed as argument at the time of function call and it is accepted by creating an alias with the help of reference variable.

- If the programmer is not providing the body for copy constructor, the compiler will provide the copy constructor and the values of one object will be copied to other.

When the copy constructor is called implicitly?

Copy constructor is called in 3 places

1. When an object is created from another object of same type

```
  sample I1;

  sample I2(I1);  //copy constructor is called

       or

   sample I3=I1; //copy constructor is called
```

2. When an object is passed by value as an argument to a function

In this case copy constructor will be called implicitly and programmer need not explicitly write the code for copying i.e. body of copy constructor is not required.

```
class Distance
{ int feet, inches;
public:
void add(Distance a, Distance b)
{
inches=a.inches+b.inches;
feet= a.feet+b.feet;
}
void disp(){cout<<feet<<inches;}
};

main()
{
Distance d1, d2, d3;
d3.add(d1,d2);
d3.disp();
}
```

3. When an object is returned from a function

In this case copy constructor will be called implicitly and programmer need not explicitly write the

code for copying i.e. body of copy constructor is not required.

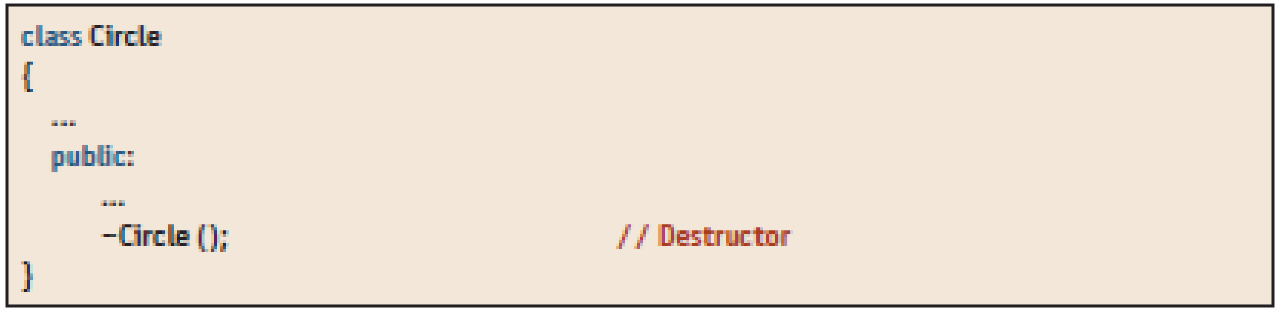
```
Class Distance
{ int feet, inches;
public:
Distance(){}
Distance(Distance &d){cout<<"copy"<<endl;}
 Distance add(Distance a, Distance b)
{
inches=a.inches+b.inches;
feet= a.feet+b.feet;
return *this;   // will discuss about this pointer later
}
void disp(){cout<<feet<<inches;}
};
```

```
main()
{
Distance d1, d2,
d3,d4;
d4=d3.add(d1,d2);
d4.disp();
}
```

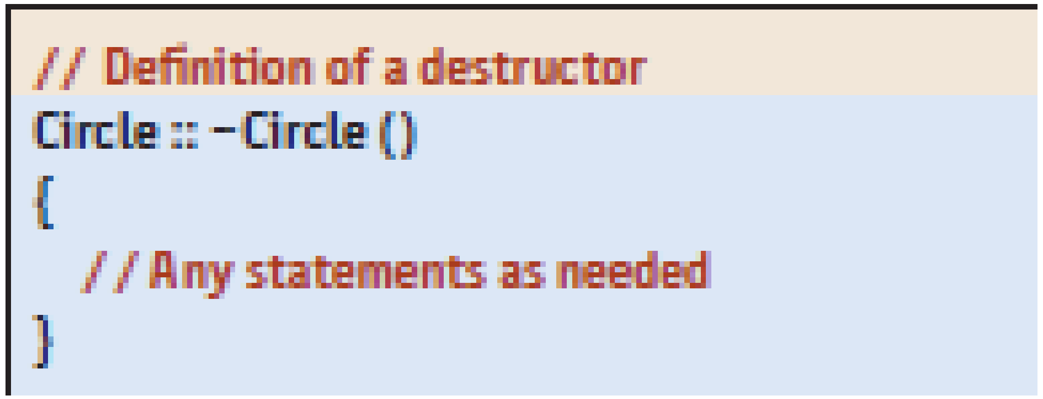
Destructor

- A destructor is a special-purpose member function with no parameter and is designed to clean up and recycle an object.
- Used to destroy the objects created .
- A destructor cannot have a return value (not even *void*) because it returns nothing.
- A destructor has no arguments
- Always access modifier is public
- A destructor is guaranteed to be automatically called and executed by the system when the object instantiated from the class goes out of scope.
- In other words, if we have instantiated five objects from the class, the destructor is automatically called five times to guarantee that all objects are cleaned up.
- Destructor can’t be overloaded

- Destructor name is same as class name preceded with tilde symbol.



```
class Circle
{
   ...
   public:
        ...
        ~Circle ();                               // Destructor
}
```



```
// Definition of a destructor
Circle :: ~Circle ()
{
   // Any statements as needed
}
```

```
int count=0;
class alpha
{
    public:
    alpha( )
    {
    count ++;
    cout<<"\n object created"<<count<<endl;
    }
    ~alpha( )
    {
    cout<<"\n object destroyed"<<count<<endl;
    count--;
    }
};
```

```
int main( )
{
    cout<<" \n enter main \n:";
    alpha A1,A2;
    {
        cout<<" \n enter block 1 :\n";
        alpha A3;
    }
    {
        cout<<" \n \n enter block2 \n";
        alpha A4;
    }
    cout<<\n exit from main \n:";

}
```

- enter main
- object created 1
- object created 2
- enter block 1
- object created 3
- object destroyed 3
- enter block 2
- object created 3
- object destroyed 3
- exit from main
- object destroyed 2
- object destroyed 1

How destructor can be called explicitly?

```
object.~destructorname();

Circle C1;
C1.~Circle();
```

Default arguments

- C++ allows us to call a function without specifying all its arguments.
- In such cases, the function assigns a default value to the parameter which does not have a matching arguments in the function call.
- Default values are specified when the function is declared or defined.(cant specify at both the places)
- Default values are added from right to left.
- float amount (float principle, int period ,float rate=0.15);
- int mul(int i, int j=3); //legal
- int mul(int i=2, int j);//illegal
- int mul(int i, int j=3, int k); //illegal
- int mul(int i, int j=3, int k=4); //legal

If there is a function with default arguments like this

- int mul(int i, int j=3, int k=4);

mul(4,5,6);   // i will get 4, j will get 5 and k will get 6

mul(4,5);   //i will get 4, j will get 5 and k will get 4

mul(4);   //i will get 4, j will get 3 and k will get 4

mul();   //error, coz cant find a corresponding fn

- Default arguments are useful in situation where some arguments always have the some value.
- For example, bank interest may retain the same interest rate for all customers for a particular period of deposit
- **float value(float p, int n, float r=0.15);**

```
void printline(char ch='*',int len=40);
int main()
{
    printline( );  // will print * 40 times
    printline('='); // will print = 40 times
    printline('+', 30); // will print + 30 times
}
void printline (char ch, int len)
{
  for(int i=1;i<=len;i++)
        cout<<ch;
}
```