Object Oriented Concept and Programming
Unit –2

–Madhavi Dave

Classes and objects

introduction

- The most important feature if c++ is class.
- A class is an idea of structure used in c.
- It is a new way to creating and implementing user-defined data type.

C structures revisited

- **Example is**:

```
struct student
{
     int Roll_no;
     int marks;
};
```

- Create structure variable:

```
     Struct student s1;
```

Limitation of C structure

- U can not perform all operation directly on structure variables.
- Like

  struct student s1, s2, s3;

  then

  S3=s1+s2 is not allowed
- They do not permit data hiding. Structure members are public and directly accessed by the structure variable.
- In c++ class is there to overcome these limitations.

Specifying Class

- A class is a way to bind data and its associated function together.
- Class specification has two parts:

  - Class declaration
  - Class function definition

Class declaration

- General format of class declaration is:

```
class class_name
{
      private:
                 variable declaration;
                 function declaration;
      protected:
                  variable declaration;
                 function declaration;
      public:
                  variable declaration;
                 function declaration;

}
```

The member of class declare as private, protected or public. By default it is private. They are called access specifies.

Cont..

- Example:

```
class student
{
      int Roll_no;
      int marks;
    public:
      void setdata( );
      void display( );
};
```

Creating objects

- Syntax is:

  class_name object_name

- Example:

  student s1,s2;

S1 and s2 are object of class type student. The necessary memory is allocated to an object at this time.

Access Class Members

- Syntax is:

  Object_name.memberfunction(actual argument)

  Object_name.datamember

- Example:

  S1.getdata(arguments if any)

Note: Only public data member and functions are accessed by object.

Private, protected, public

- **Private**: Members declared as private can be accessed only by the member function of that class. All members of class are private default.

- **Protected**: Members declared as protected can be accessed in the same class as well as all the other class derived from this class.

- **Public**: Members declared as public can be accessed by any other function/class in the program.

Cont..(example)

```
class student
        {
                int Roll_no;
                int marks;
            public:
                int count;
                void setdata( );
                void display( );
        };
```

- Now assume s1 is object of class student. State following are valid/invalid:

statement	valid/invalid
s1.marks=50;	invalid
s1.count=1;	valid
s1.setdat( );	valid

Defining member Function

- Member function can be defined in two ways:
  - Inside the class definition
  - Outside the class definition

The function perform same task, does not matter where it is defined outside/inside a function

Inside the Class Definition

```
class student
{
          int Roll_no;
          int marks;
    public:
          void setdata(int r, int m)
          {
                    Roll_no=r;
                    marks=m;
          }
          void display();
};
void main()
{
          student s1;
          int roll, mark;
          cout<<“enter roll no and marks”<<“\n”;
          cin||roll||mark;
          s1.setdata(roll,mark);
}
```

outside the Class Definition

- General format is:

```
return_type class_name :: function_name(argument declaration)
{
   function body
}
```

outside the Class Definition

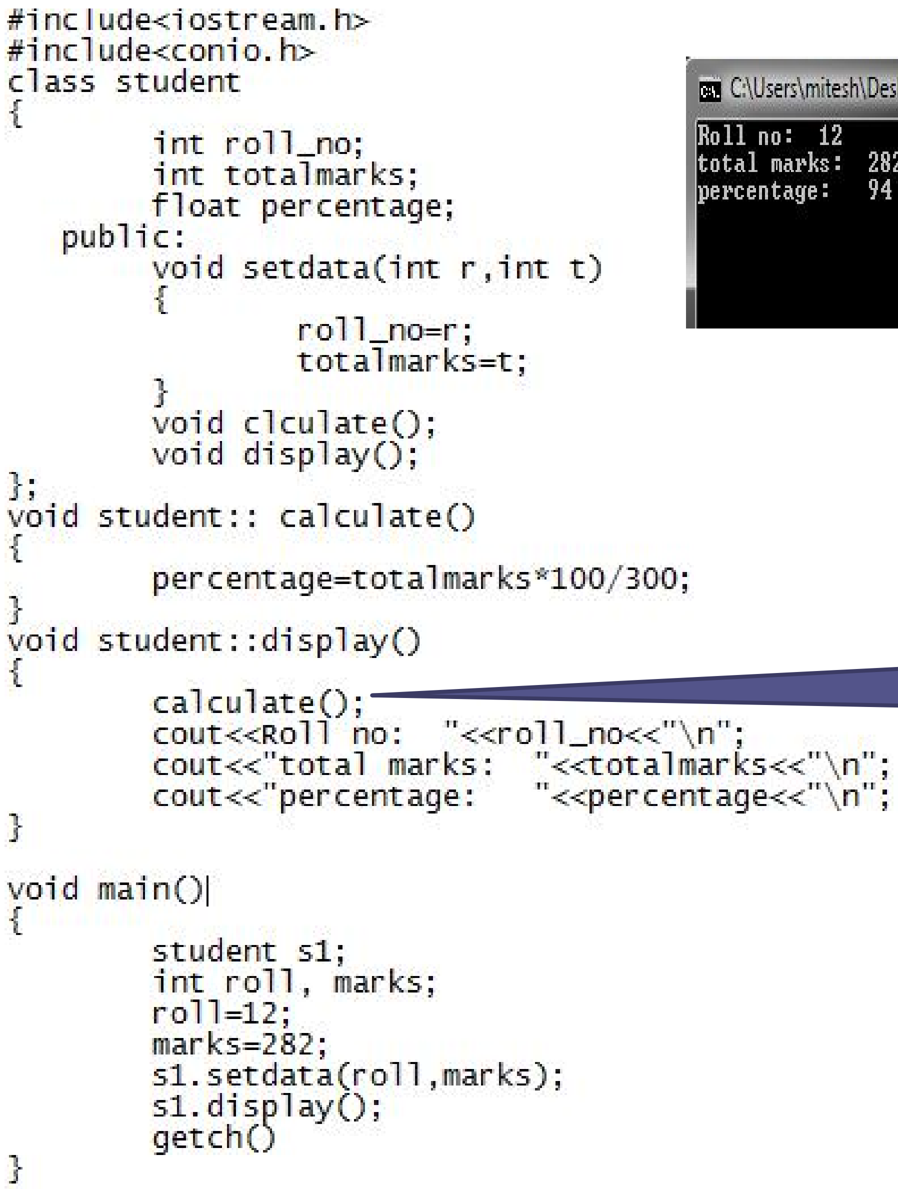
```
class student
{
            int Roll_no;
            int marks;
    public:
            void setdata(int r, int m)
            {
                        Roll_no=r;
                        marks=m;
            }
            void display();
};
void student :: display()
{
            cout<<“roll no is:   ”<<Roll_no<<“\n”;
            cout<<“marks is:   ”<<marks
}
void main()
{
            student s1;
            int roll, mark;
            cout<<“enter roll no and marks”<<“\n”;
            cin||roll||mark;
            s1.setdata(roll,mark);
            s1.dispaly();
}
```

```
Output:
enter roll no and marks
12 50
roll no is:   12
marks is:     50
```

Nesting of Member Function

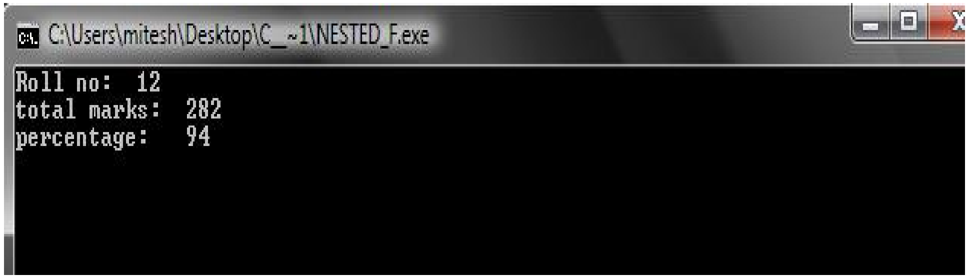
- A member function can be called from another member function of the same class. This is called *nesting of member function*.

Cont..



```
#include<iostream.h>
#include<conio.h>
class student
{
        int roll_no;
        int totalmarks;
        float percentage;
   public:
        void setdata(int r,int t)
        {
                roll_no=r;
                totalmarks=t;
        }
        void clculate();
        void display();
};
void student:: calculate()
{
        percentage=totalmarks*100/300;
}
void student::display()
{
        calculate();
        cout<<Roll no:   "<<roll_no<<"\n";
        cout<<"total marks:   "<<totalmarks<<"\n";
        cout<<"percentage:    "<<percentage<<"\n";
}

void main()
{
        student s1;
        int roll, marks;
        roll=12;
        marks=282;
        s1.setdata(roll,marks);
        s1.display();
        getch()
}
```



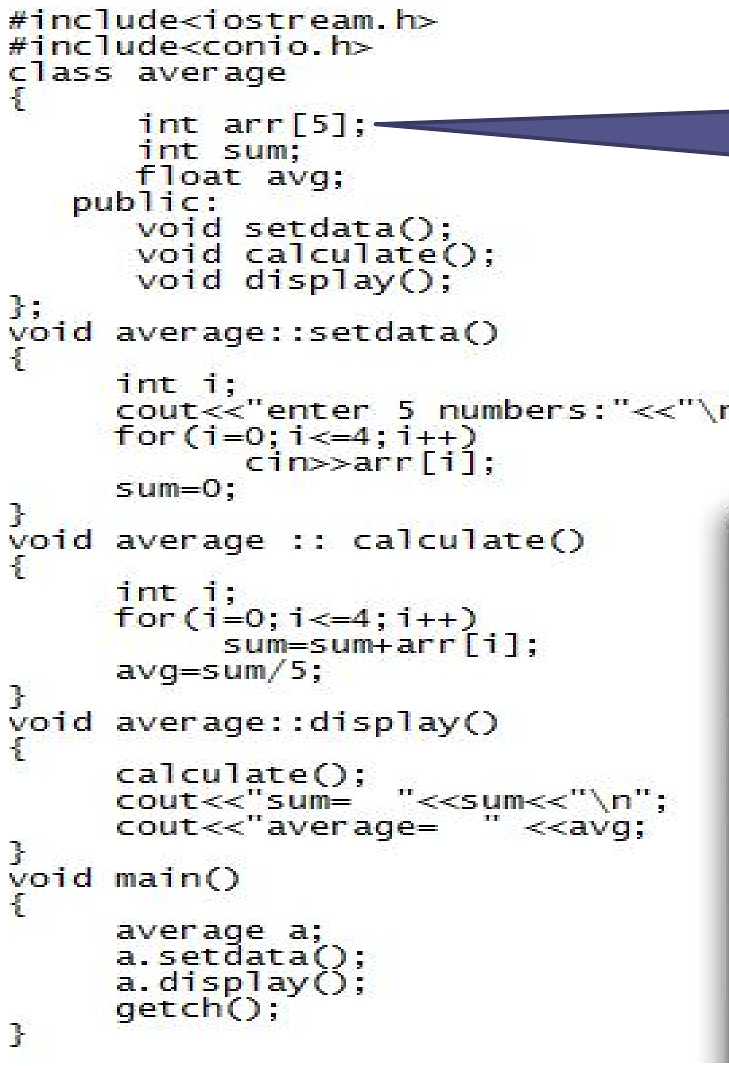
```
Roll no:  12
total marks:  282
percentage:   94
```

calculate( ) is called from display function

Private member function

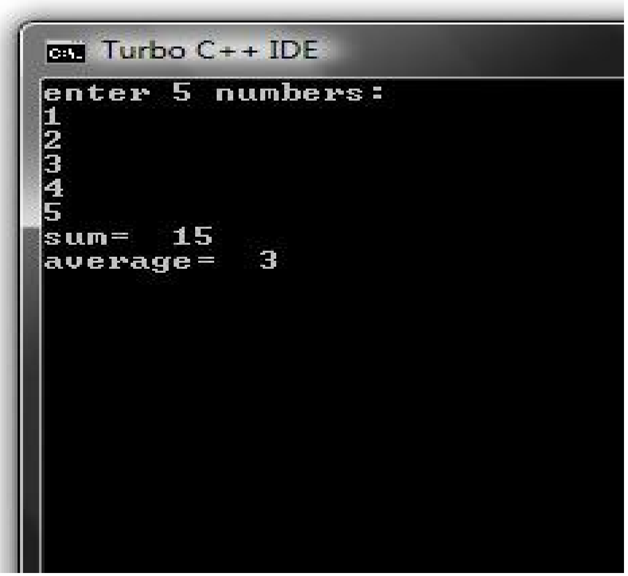
- Member function can also be declared as private. It is required when the function are to be hidden from outside world.
- Private member function can only access from member function of same class.
- Neither the object nor any external function can access the private member function.

Arrays within class



```
#include<iostream.h>
#include<conio.h>
class average
{
        int arr[5];
        int sum;
        float avg;
    public:
        void setdata();
        void calculate();
        void display();
};
void average::setdata()
{
        int i;
        cout<<"enter 5 numbers:"<<"\n";
        for(i=0;i<=4;i++)
              cin>>arr[i];
        sum=0;
}
void average :: calculate()
{
        int i;
        for(i=0;i<=4;i++)
             sum=sum+arr[i];
        avg=sum/5;
}
void average::display()
{
        calculate();
        cout<<"sum=   "<<sum<<"\n";
        cout<<"average=   "  <<avg;
}
void main()
{
        average a;
        a.setdata();
        a.display();
        getch();
}
```

arr" is a int type array which can store 5 element. Elements are scanned in setdata function



```
enter 5 numbers:
1
2
3
4
5
sum=  15
average=   3
```

Array of objects

- Syntax is:

  ```
  class_name array_name[size];
  ```

- Example:

  ```
  student std[5];
  ```

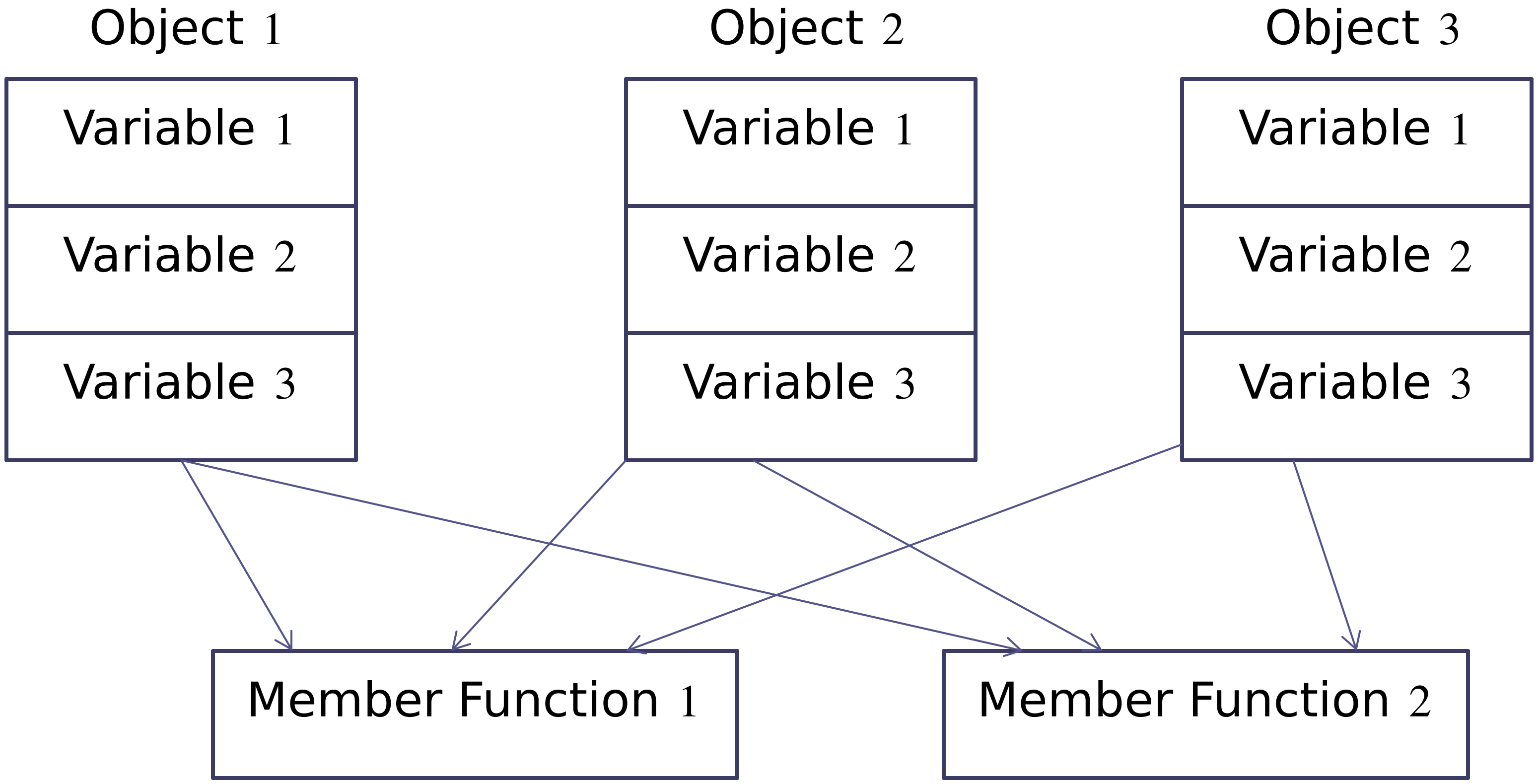
  Will create array of 5 object names std.

- How to access member function

  ```
  std[0].setdata();
  std[2].display();
  ```

Memory Allocation for Objects

- Memory allocation pattern for objects:



Cont..

- Memory is allocated when object are created not when class are specified.
- That is true for data member only, not for the member function. For that rule is slightly different.
- All objects of same class use the same member function.
- The member function are created and place in memory only once–when they are defined.
- No separate memory is allocated for member function when object is created.
- For data member separate memory is allocated for each object because data member may have a different value for each member.

Static Data Member

- Data members are made common to all objects of a class by declaring them static. That data members are called static data members.

- Only one copy of static variables is maintained by the class and it is shared by all the objects of that class.

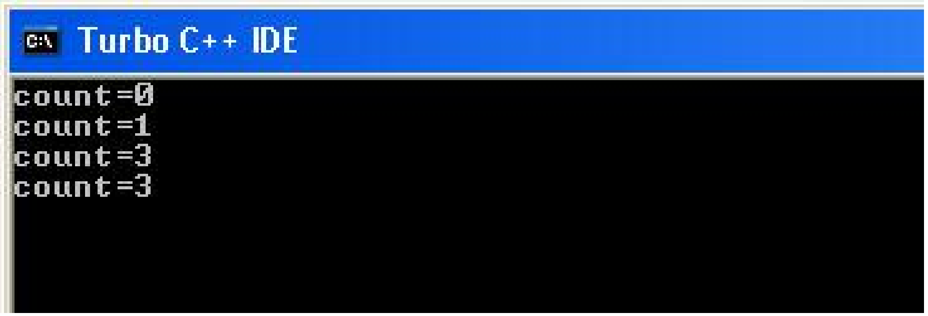
- It is generally used when we want to maintain common value to the entire class.

- They are initialized to ZERO.

Cont..

```
#include<iostream.h>
#include<conio.h>
class static_data
{
        int data;
        static int count;
    public:
        void inc()
        {
                count++;

        }
        void disp()
        {
                cout<<"count="<<count<<"\n";
        }
};
int static_data::count=0;
void main()
{
        class static_data s1,s2,s3;
        clrscr();
        s1.disp();
        s1.inc();
        s1.disp();
        s2.inc();
        s3.inc();
        s2.disp();
        s3.disp();
        getch();
}
```



```
count=0
count=1
count=3
count=3
```

Static Member Function

- Like static data member, static member function are associated with a class, not with any particular object of the class.
- So they are invoked using class name, like

  Class_name :: function_name

  OR

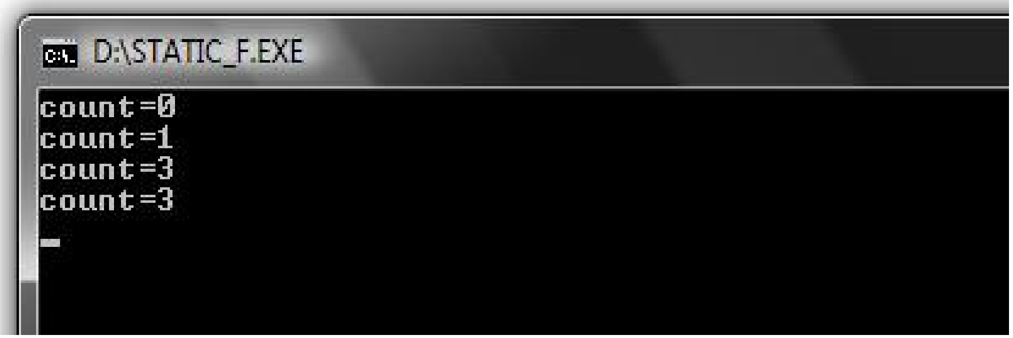
  object.function_name

- A function declares as a static can access only static member function of that class.
- They can not be declare as *const* or *volatile*.

Cont…

```
#include<iostream.h>
#include<conio.h>
class static_data
{
        int data;
        static int count;
    public:
        void inc()
        {
                count++;

        }
        static void disp()
        {
                cout<<"count="<<count<<"\n";   //only static data can be access
        }
};
int static_data::count=0;
void main()
{
        class static_data s1,s2,s3;
        clrscr();
        s1.disp();
        s1.inc();
        s1.disp();
        s2.inc();
        s3.inc();
        s2.disp();
        static_data::disp();
        getch();
}
```



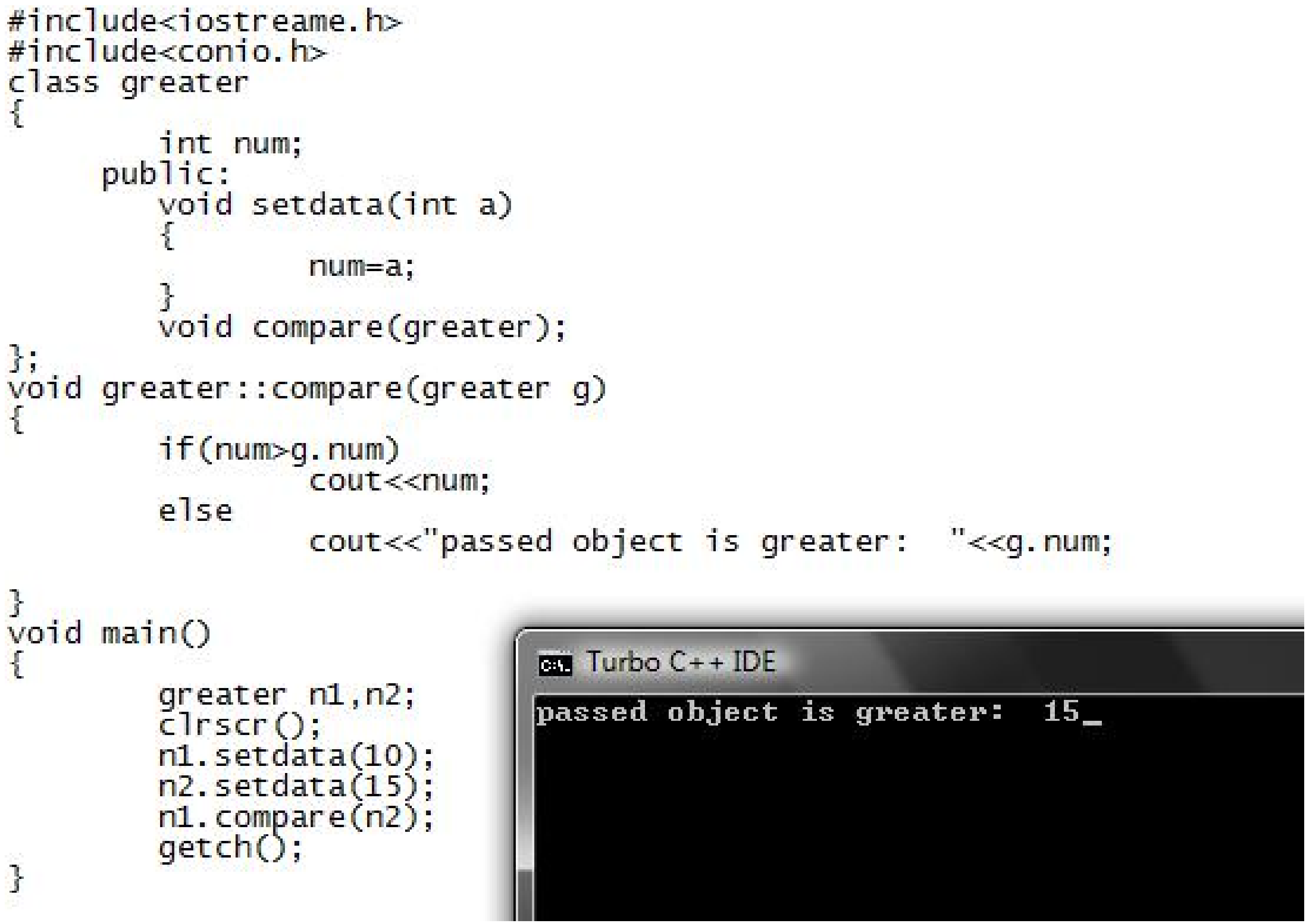
D:\STATIC_F.EXE

```
count=0
count=1
count=3
count=3
_
```

Object as Function Argument

- Object can be passed to a function as argument, like any other data type.
- It can be passed in two ways.
  - By value
  - By reference

Cont..



```
#include<iostreame.h>
#include<conio.h>
class greater
{
        int num;
     public:
        void setdata(int a)
        {
                num=a;
        }
        void compare(greater);
};
void greater::compare(greater g)
{
        if(num>g.num)
                cout<<num;
        else
                cout<<"passed object is greater:  "<<g.num;

}
void main()
{
        greater n1,n2;
        clrscr();
        n1.setdata(10);
        n2.setdata(15);
        n1.compare(n2);
        getch();
}
```

```
passed object is greater:  15_
```

Returning object

- WAP that adds two complex numbers A and B to produce third number C and displays all the three numbers.

Constant Member Function

- Ex:

```
void mul(int,int) const;
double get_bal( ) const;
```

The complier will generate error message if function try to alter the data values.

Pointers to members

It is possible to take the address of a member of a class and assign it to a pointer.

```
Class A
{
   private:
          int m;
   public:
          void show( );
}
```

Pointer to the member m as follows:

```
int  A::*  ip = &A :: m;
```

A::*	pointer to member of A class
&A::m	address of the m member of A class

Cont..

- ip can be used to access m inside member function:

```
    cout<<a.*ip;
    cout<<a.m;
```

a is object of class A.

Pointer to object

```
ap=&a;
cout<<ap   *ip;
cout<<ap   m;
```

ap is pointer to object a.

Pointer to member function

General syntax is:

Ret_type (class_name::*ptr)(arg list)=&class_name :: fun_name

And call using following syntax:

(obj_name.*ptr to member fun)(arg list);

(ptr to object->*ptr to member fun)(arg list);

Example

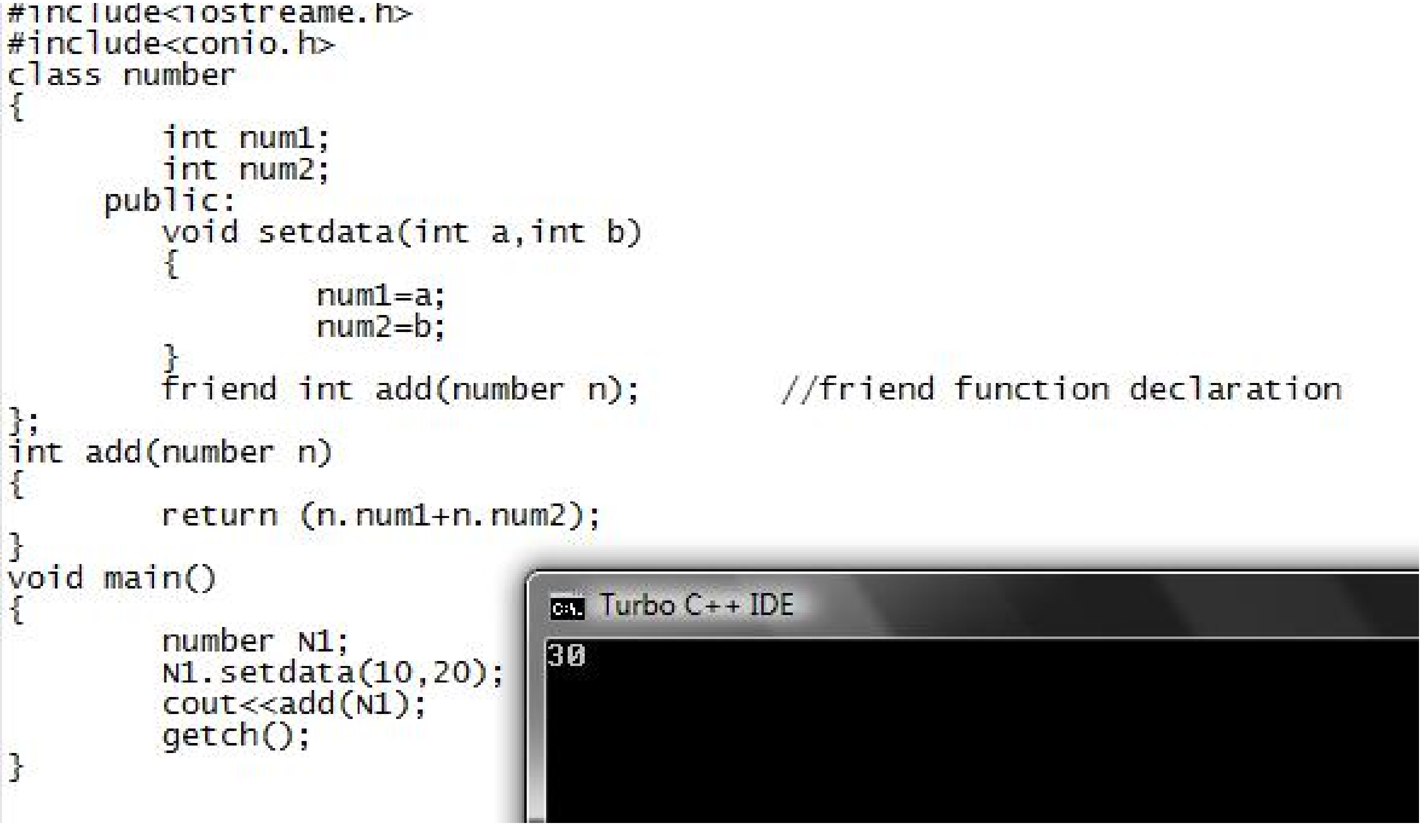
```
Class M
{
        int x;
          int y;
  public:
        void set_xy(int a,int b)
        {
                 x=a;
                 y=b;
        }

}
```

FRIEND function

- The private data members of a class can be accessed only by its member function.
- But if non-member function want to access these data, then it is possible with friend function only.
- A function can be made friend of a class by using the keyword *friend.*

Cont…



```
#include<iostreame.h>
#include<conio.h>
class number
{
        int num1;
        int num2;
     public:
        void setdata(int a,int b)
        {
                num1=a;
                num2=b;
        }
        friend int add(number n);        //friend function declaration
};
int add(number n)
{
        return (n.num1+n.num2);
}
void main()
{
        number N1;
        N1.setdata(10,20);
        cout<<add(N1);
        getch();
}
```

```
30
```

Cont….

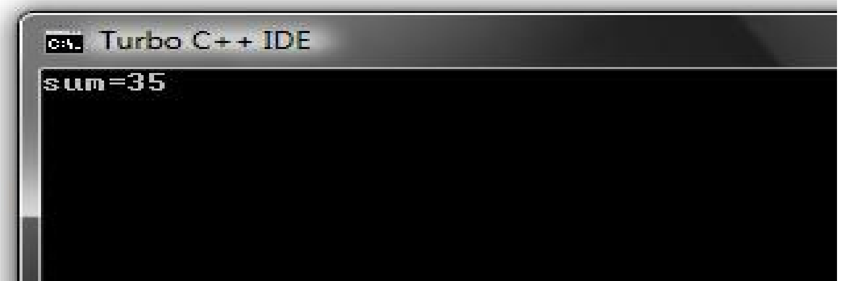
- Member function of one class can be a friend of another class.

- Class name is used as the qualifier for the member function.

Cont..

```
#include<iostreame.h>
#include<conio.h>
class B;                              //forward declaration
class A
{

        int a;
     public:
        void setdata()
        {
                a=10;
        }
        void disp()
        {
                cout<<"a="<<a;
        }
        void add(B object_B);
};
class B
{
        int b;
     public:
        void setdata()
        {
                b=25;
        }
        void disp()
        {
                cout<<"b="<<b;
        }
        friend void A::add(B obj);
};
void A::add(B obj)
{
        int sum;
        sum=a+obj.b;
        cout<<"sum="<<sum;
}
void main()
{
        A oa;
        B ob;
        clrscr();
        oa.setdata();
        ob.setdata(ob);
        oa.add();
        getch();
}
```



```
sum=35
```

Cont..

- An entire class can be made friend of another class. This has the effect of making every member function of the class a friend.
- Ex:

```
class B
{
                int b;
    public:
                void setdata( );
                void dispdata( );
};
class A
{
                friend class B;
                int a;
    public:
                void setdata( );
                void disp( );
}
```

Characteristics of friend function

- A friend function does not belong to the class to which it is declared friend.
- A friend function is invoked just like any other c++ function(without using object), as it is not a part of the class.
- It can not access data members directly like member functions. It has to use the object name along with the dot operator.
- It can be declared private, public, protected without altering the meaning.
- usually., it has object as argument.

Constructors and destructors

introduction

- In all program we have written setdata() function to set values to the private variables of the class.
- And this function must be invoked explicitly by the object.
- These functions cannot be used to initialize the member variables at the time of creation of object
- So, concept of *constructor* and *destructor* came into existence.

Constructors

- It is a special member function whose main task is to allocate the memory and initialize the objects of the class.
- It has the same name as class name.
- Constructor is invoked whenever the object of the class is created.
- As it constructs the values of data members of the class, it is called constructor.
- Types of constructor:
  - Default constructor
  - Parameterized constructor
  - Copy constructor
  - Dynamic constructor

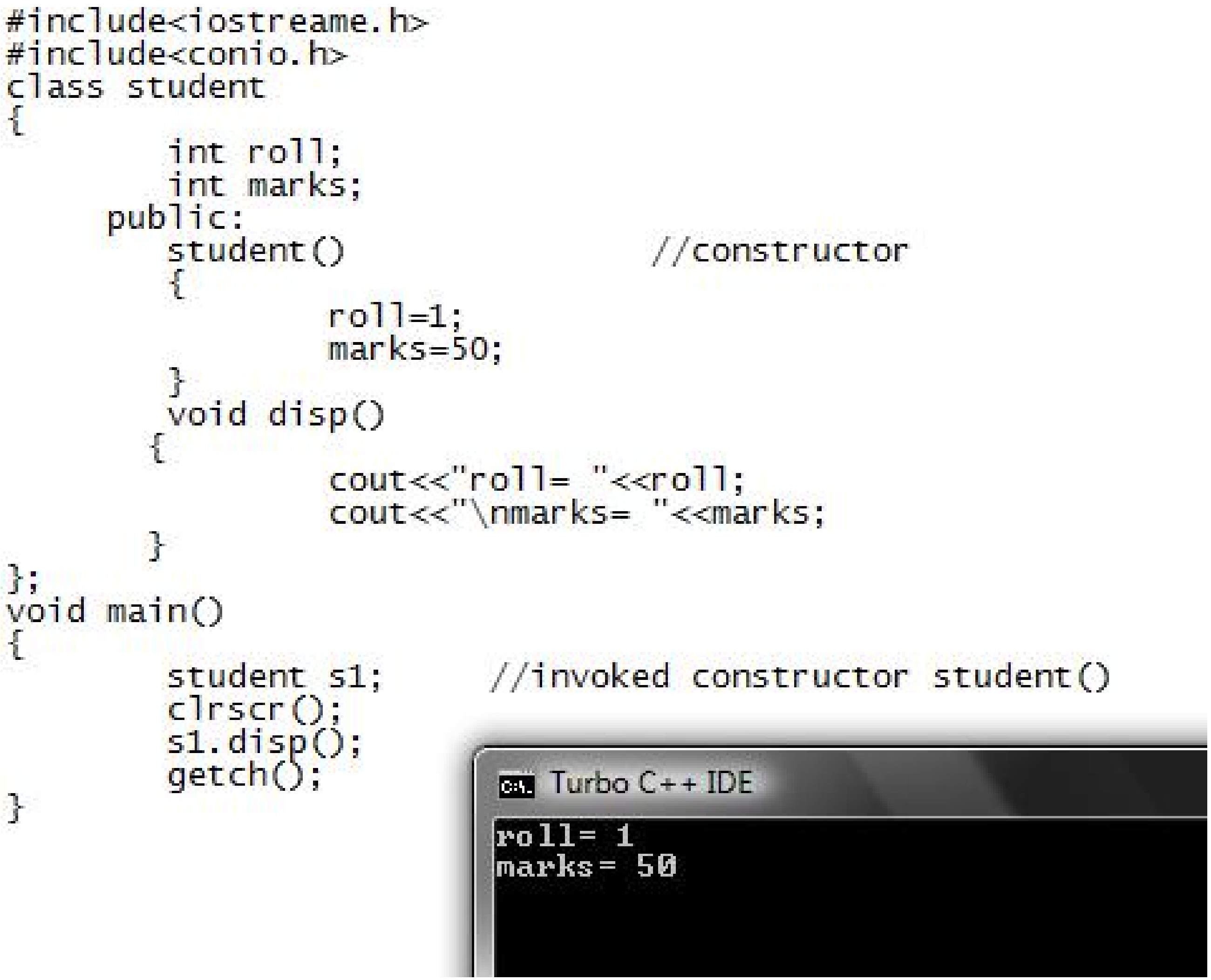
Constructor Characteristics

- They should be declared in the public section.
- They are invoked automatically when the objects are created.
- They do not have return types, not even void and therefore they cannot return values.
- They cannot be inherited, though a derived class can call the base class constructor.
- They can have default arguments.
- Constructors cannot be virtual.
- We cannot refer to their addresses.
- They make implicit calls to the operators new and delete when memory allocation is required.

Default constructor

- A constructor without argument is called "*Default Constructor*".
- Default constructor is called at run time when object is created.

Cont..



```
#include<iostreame.h>
#include<conio.h>
class student
{
        int roll;
        int marks;
     public:
        student()                    //constructor
        {
                roll=1;
                marks=50;
        }
        void disp()
       {
                cout<<"roll= "<<roll;
                cout<<"\nmarks= "<<marks;
       }
};
void main()
{
        student s1;     //invoked constructor student()
        clrscr();
        s1.disp();
        getch();
}
```

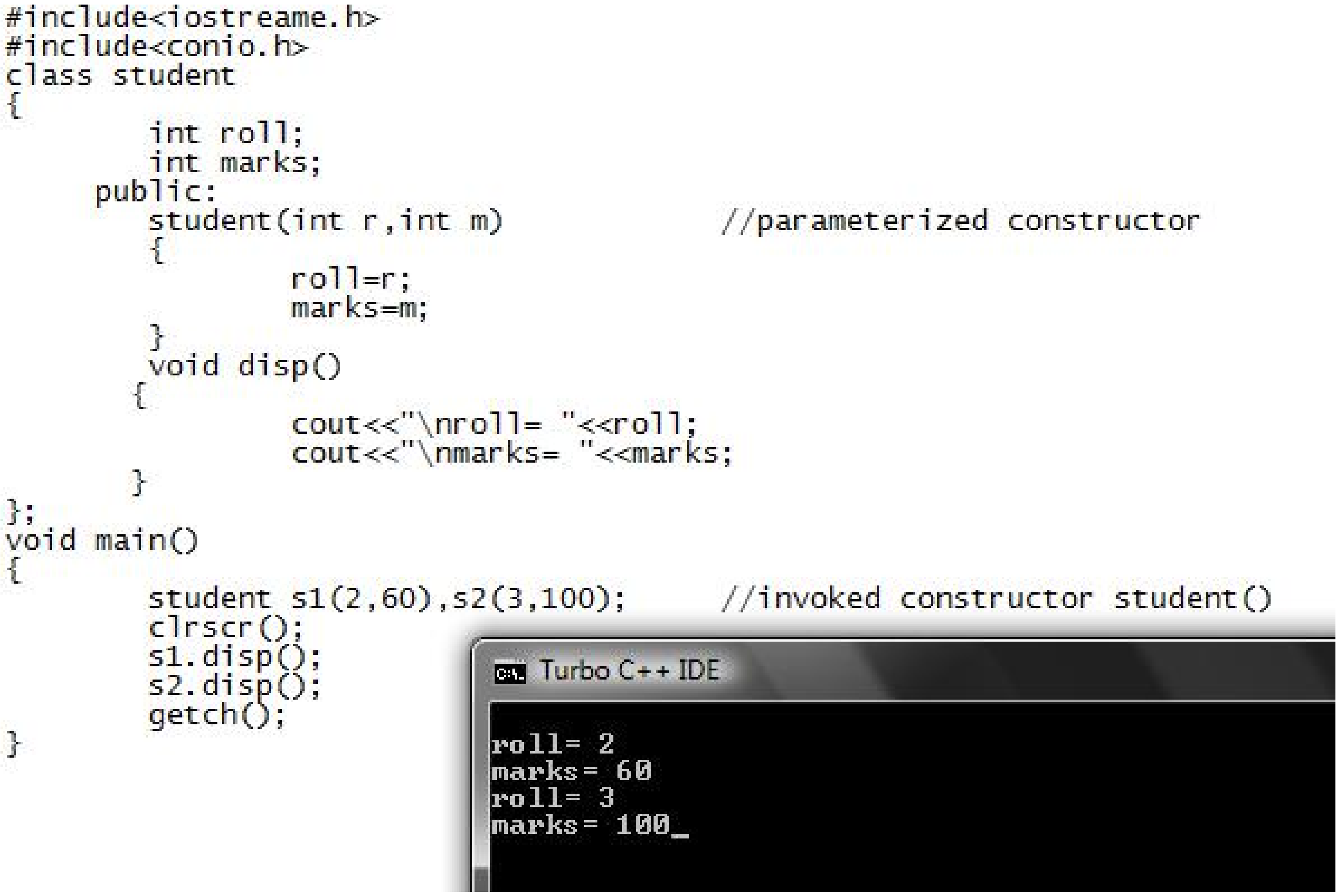
Turbo C++ IDE

```
roll= 1
marks= 50
```

Parameterized Constructor

- Constructor with take parameters are called “*parameterized constructor*”.
- For invoking this constructor, appropriate parameters should be passed while creating object.
- There are two way to call constructor:
  - Implicitly
  - Explicitly

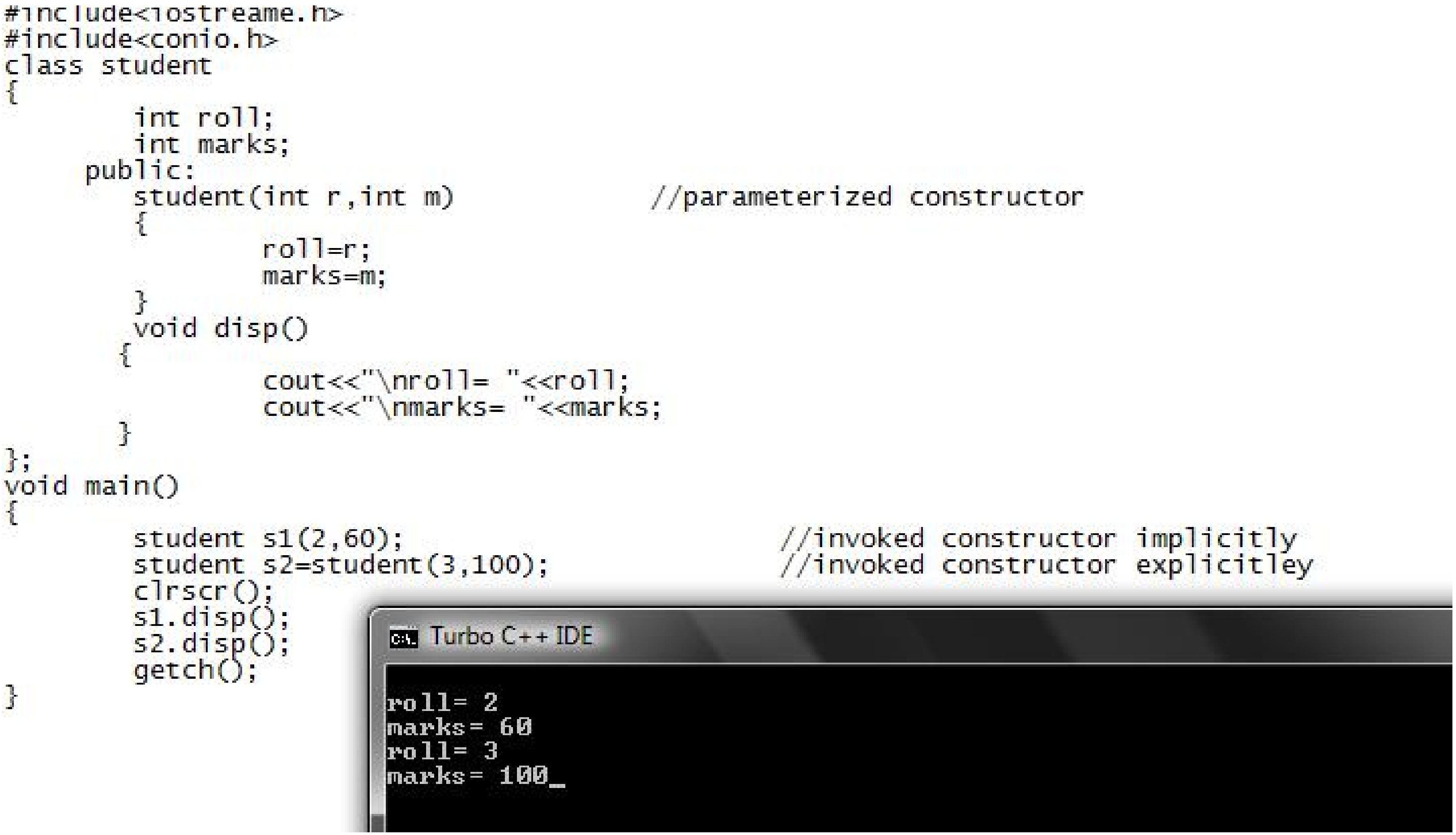
Cont..(Implicitly called)



```
#include<iostreame.h>
#include<conio.h>
class student
{
        int roll;
        int marks;
     public:
        student(int r,int m)            //parameterized constructor
        {
                roll=r;
                marks=m;
        }
        void disp()
       {
                cout<<"\nroll= "<<roll;
                cout<<"\nmarks= "<<marks;
       }
};
void main()
{
        student s1(2,60),s2(3,100);     //invoked constructor student()
        clrscr();
        s1.disp();
        s2.disp();
        getch();
}
```

```
roll= 2
marks= 60
roll= 3
marks= 100_
```

Cont..(explicitly called)



```
#include<iostreame.h>
#include<conio.h>
class student
{
        int roll;
        int marks;
     public:
        student(int r,int m)            //parameterized constructor
        {
                roll=r;
                marks=m;
        }
        void disp()
       {
                cout<<"\nroll= "<<roll;
                cout<<"\nmarks= "<<marks;
       }
};
void main()
{
        student s1(2,60);                               //invoked constructor implicitly
        student s2=student(3,100);                      //invoked constructor explicitley
        clrscr();
        s1.disp();
        s2.disp();
        getch();
}
```

```
roll= 2
marks= 60
roll= 3
marks= 100_
```

Constructor with default arguments

- The default arguments can be passed in the constructor while declaring.
- Eg. Complex(float num, float num2=0);
- The constructor is called either with or without the arguments while creating object.

Copy constructor

- It is used to make copies of the objects.
- It is generally used to initialize an object from another object.
- Eg. Integer I1(I2) or Interger I1 = I2;
- That is, it is a constructor of class *class_name* that takes a reference object of the same class as a argument.

Cont..

- Copy constructor can be invoked by:

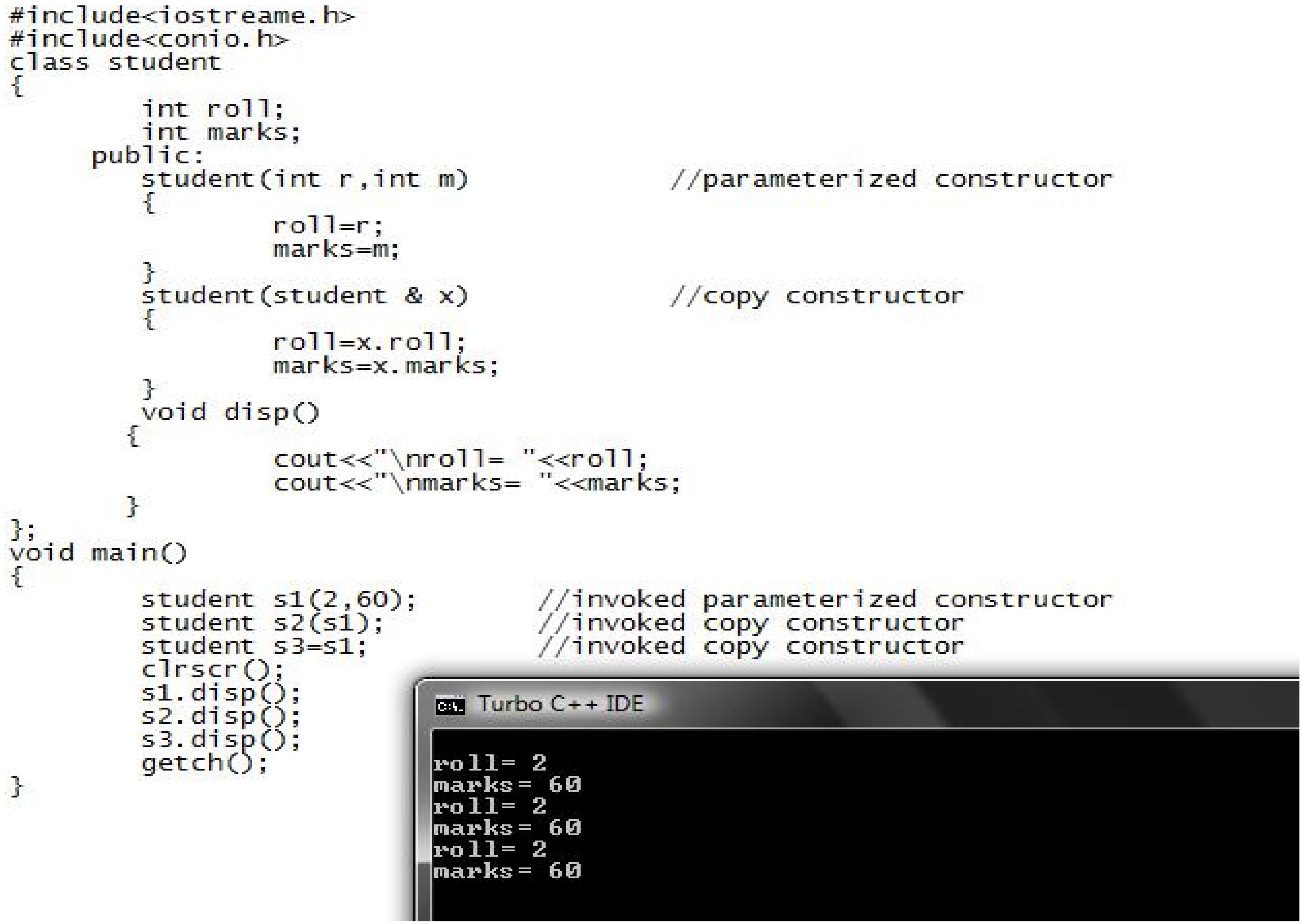
  Class_name object1(object2)

  OR

  Class_name object1=object2

- The process of initializing through a copy constructor is known as "*copy constructor*".

Cont..



```
#include<iostreame.h>
#include<conio.h>
class student
{
        int roll;
        int marks;
     public:
        student(int r,int m)            //parameterized constructor
        {
                roll=r;
                marks=m;
        }
        student(student & x)            //copy constructor
        {
                roll=x.roll;
                marks=x.marks;
        }
        void disp()
       {
                cout<<"\nroll= "<<roll;
                cout<<"\nmarks= "<<marks;
       }
};
void main()
{
        student s1(2,60);       //invoked parameterized constructor
        student s2(s1);         //invoked copy constructor
        student s3=s1;          //invoked copy constructor
        clrscr();
        s1.disp();
        s2.disp();
        s3.disp();
        getch();
}
```

```
roll= 2
marks= 60
roll= 2
marks= 60
roll= 2
marks= 60
```

Dynamic Constructor

- Object can be created run time. So, memory is allocated run time only. This is called “***dynamic constructor***”.

- That can be achieved by ***new*** operator and using pointer.

Cont..

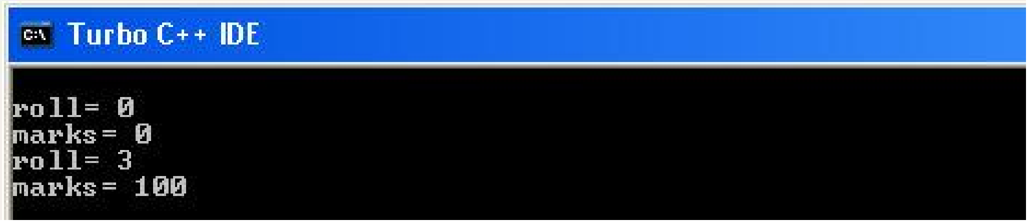
- Example:

student *sptr; // does not call any constructor and no memory is allocated.

sptr=new student( ); // memory is allocated and default constructor is called.

Cont…

```
#include<iostreame.h>
#include<conio.h>
class student
{
        int roll;
        int marks;
     public:
        student()                       //default constructor
        {
                roll=0;
                marks=0;
        }
        student(int r,int m)            //parameterized constructor
        {
                roll=r;
                marks=m;
        }
        void disp()
       {
                cout<<"\nroll= "<<roll;
                cout<<"\nmarks= "<<marks;
       }
};
void main()
{
        student *s1;                    //Memory is not allocated
        s1=new student;                 //invoke the constructor without argument
                                        //and memory is allocated
        student *s2=new student(3,100); //invoked parameterized constructor
        clrscr();
        s1->disp();
        s2->disp();
        getch();
}
```



```
roll= 0
marks= 0
roll= 3
marks= 100
```

Const Object

- We can create and use constant objects using const keyword before object declaration.
- Eg. const matrix X(m,n)
- Now here X is constant object and the values of m and n cannot be modified.
- Whenever const object try to invoke non–const member functions, the compiler gives error.

Destructor

- It is used to destroy the objects that have been created by constructor. Its name is same as class name but preceded by tilde sign.
- Eg ~integer( ) {}
- It never takes arguments not does return anything.
- It is called implicitly when program is exited or from a block.