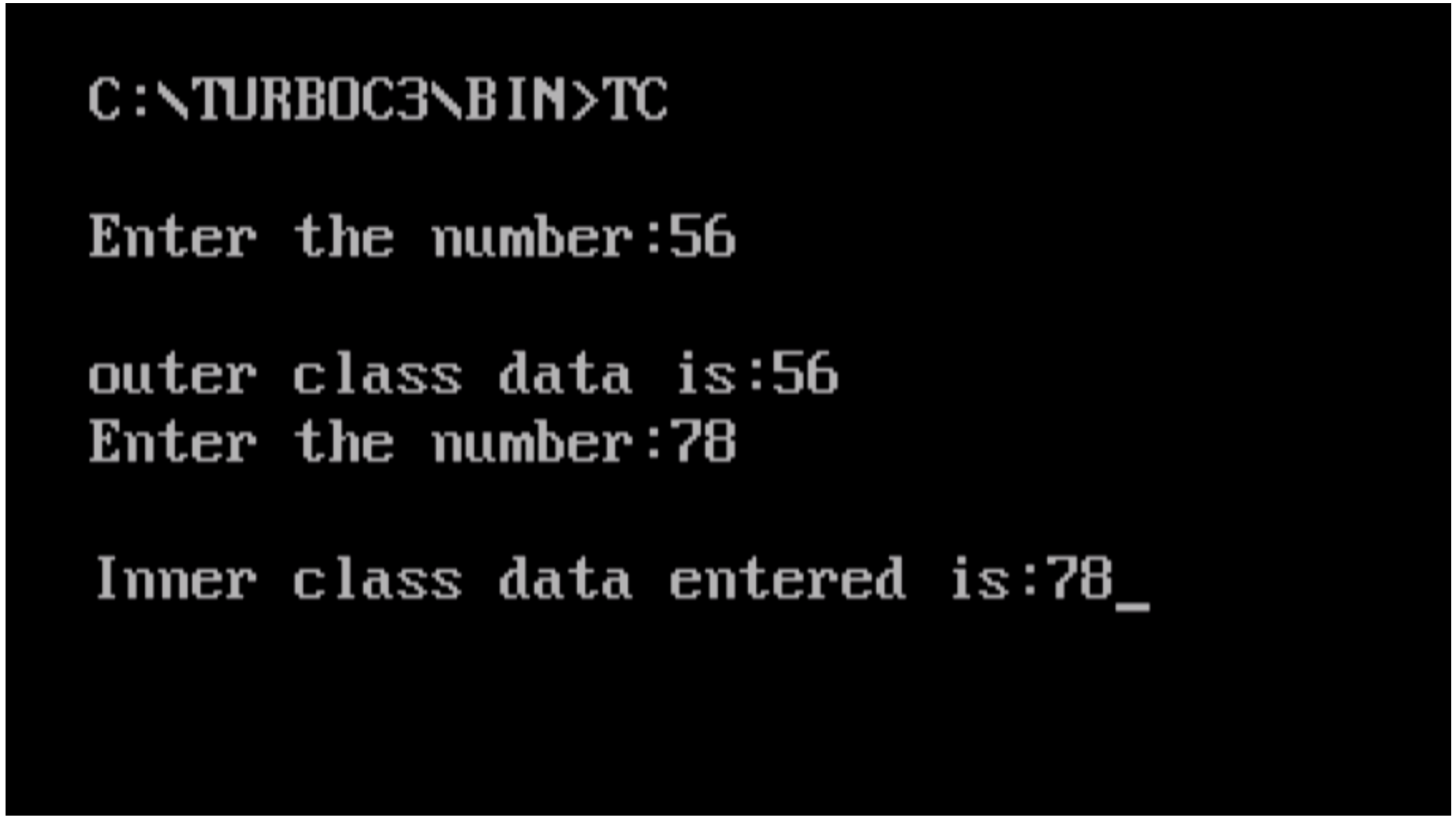
NESTED CLASS:

```
#include<iostream.h>
#include<conio.h>
class ABC
{
int data;
public:void getdata()
{               cout<<"\nEnter the number:";
                cin>>data;
                cout<<"\nouter class data is:"<<data;
}
class PQR
{ int b;
public:void getbdata()
{               cout<<"\nEnter the number:";
                cin>>b;
                cout<<"\nInner class data entered
is:"<<b;
}
};
PQR p;
void display()
{ p.getbdata();
}
};
void main()
{               ABC a;
                a.getdata();
                a.display();
getch();
}
```

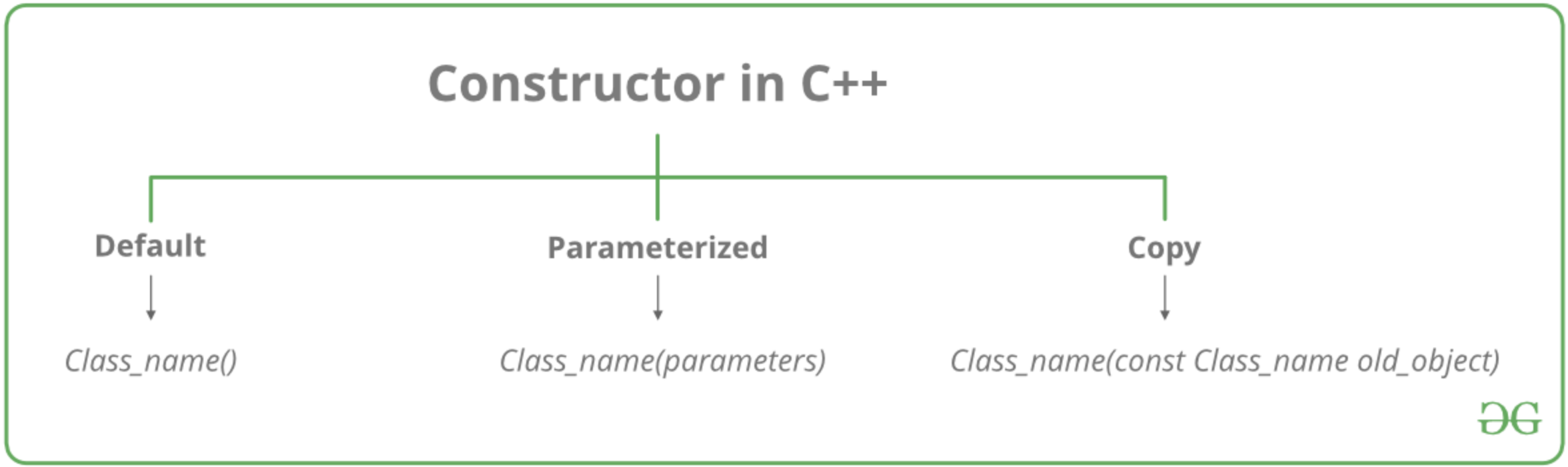

C:\TURBOC3\BIN>TC
Enter the number:56
outer class data is:56
Enter the number:78
Inner class data entered is:78_

What is constructor?

A constructor is a member function of a class which initializes objects of a class. In C++, Constructor is automatically called when object(instance of class) create. It is special member function of the class.

- Constructor has same name as the class itself
- Constructors don’t have return type
- A constructor is automatically called when an object is created.
- If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).
- It should be always declared in the public section

TYPES OF CONSTRUCTOR:



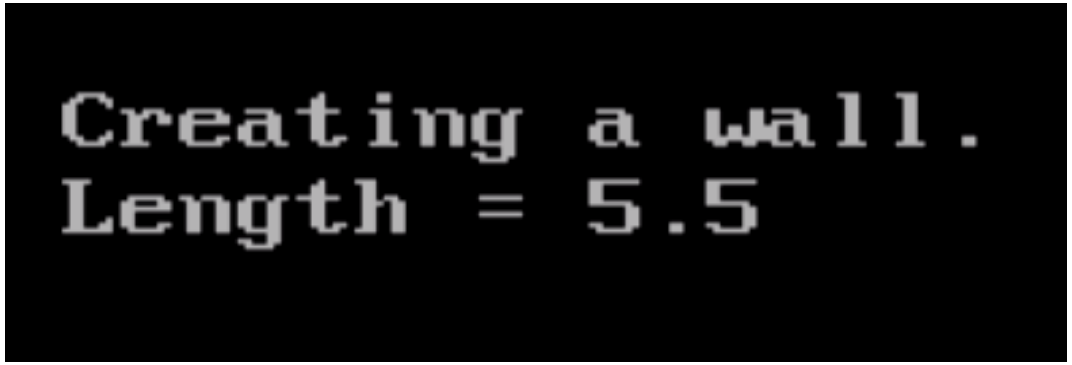
```
#include<iostream.h>
#include<conio.h>
class MyClass                          // The class
 {
  public:                             // Access specifier
    MyClass()                         // Constructor
{    cout << "Hello World!";
cout<<“This is a constructor”;
    }
};
void main() {
  MyClass myObj;    // Create an object of MyClass (this will call theconstructor)
  getch();
}
```


Hello World!
This is a constructor_

Default Constructor:

A constructor with no parameters is known as a **default constructor i.e.** doesn't take any argument.

```
#include <iostream.h>
#include<conio.h>
class  Wall {
  private:
      double length;
 public:    Wall() {
       length = 5.5;
cout << "\nCreating a wall." << endl;
       cout << "Length = " << length << endl;
    }
};
void main() {
clrscr();
   Wall wall1;
  getch();
}
```



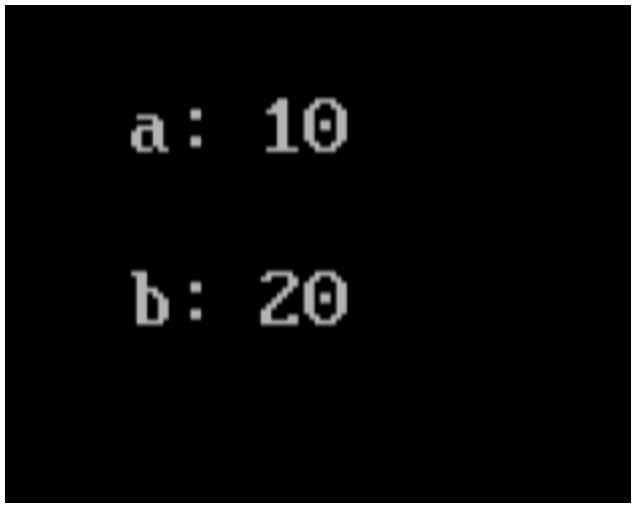
Note: If we have not defined a constructor in our class, then the C++ compiler will automatically create a default constructor with an empty code and no parameters.

```
#include <iostream.h>
#include<conio.h>
class construct {
public:
   int a, b;

   // Default Constructor
   construct()
   {
      a = 10;
      b = 20;
   }
};
  void main()
{
   clrscr();

   construct c;
   cout << "\na: " << c.a << endl
<< "\nb: " << c.b;
getch();

}
```

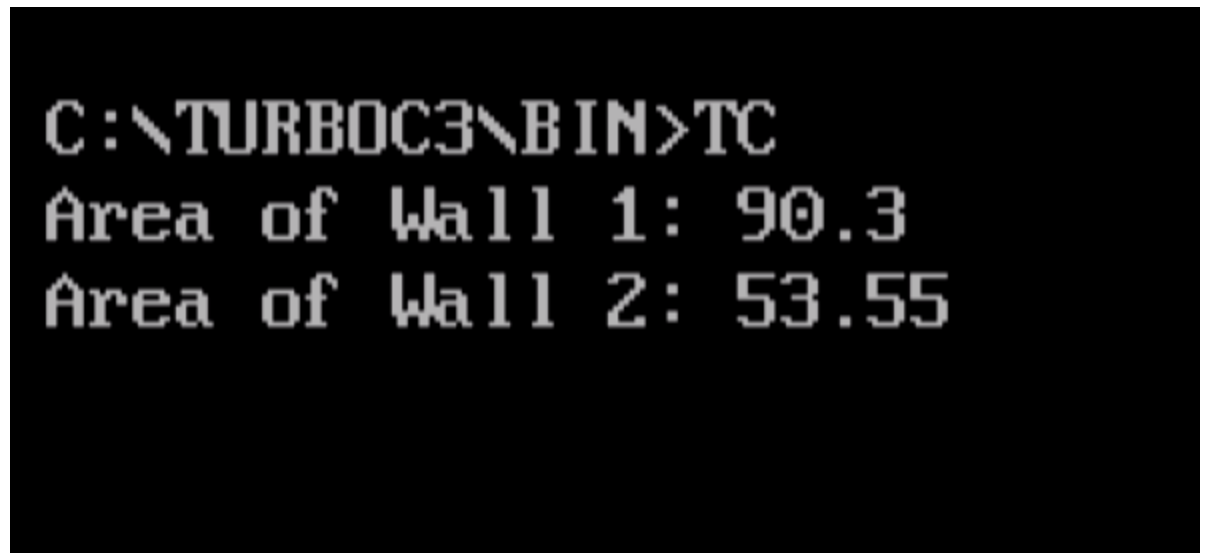

a: 10
b: 20

Parameterized Constructor:

It is possible to pass arguments to constructors. Typically, these arguments help initialize an object when it is created. To create a parameterized constructor, simply add parameters to it the way you would to any other function. When you define the constructor's body, use the parameters to initialize the object.In C++, a constructor with parameters is known as a parameterized constructor. This is the preferred method to initialize member data.

```
#include<iostream.h>
#include<conio.h>
class Wall {
  private:
   double length;
   double height;

  public:
   // create parameterized
constructor
   Wall(double len, double hgt) {
      // initialize private variables
      length = len;
      height = hgt;
   }
   double calculateArea() {
      return length * height;
   }
};
void main() {
   // create object and initialize data members
   Wall wall1(10.5, 8.6);
   Wall wall2(8.5, 6.3);
cout << "Area of Wall 1: " << wall1.calculateArea() <<
endl;
   cout << "Area of Wall 2: " << wall2.calculateArea()
<< endl;
   getch();
   }
```


C:\TURBOC3\BIN>TC
Area of Wall 1: 90.3
Area of Wall 2: 53.55

Copy Constructor:

The copy constructor in C++ is used to copy data of one object to another.
The **copy constructor** is a constructor which creates an object by initializing it with an object of the same class, which has been created previously. The copy constructor is used to −

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

A copy constructor has the following general function prototype:

```
ClassName (const ClassName &old_obj); .
```

```
#include <iostream.h>
#include<conio.h>
class Wall {
  private:
   double length;
   double height;
  public:
// parameterized constructor
   Wall(double len, double hgt) {
            // initialize private variables
            length = len;
            height = hgt;
   }
 // copy constructor with a Wall object as parameter
   Wall(Wall &obj) {
            // initialize private variables
            length = obj.length;
            height = obj.height;
   }
   double calculateArea() {
            return length * height;}
};
```

```
void main() {
clrscr();
   Wall wall1(10.5, 8.6);

   // print area of wall1
   cout << "\nArea of Room 1: " <<
wall1.calculateArea() << endl;

   // copy contents of room1 to another object
room2
   Wall wall2 = wall1;
   cout << "Area of Room 2: " <<
wall2.calculateArea() << endl;
getch();

}
```

```
Area of Room 1: 90.3
Area of Room 2: 90.3
```

Destructors:

What is destructor?

Destructor is a member function which destructs or deletes an object.

When is destructor called?

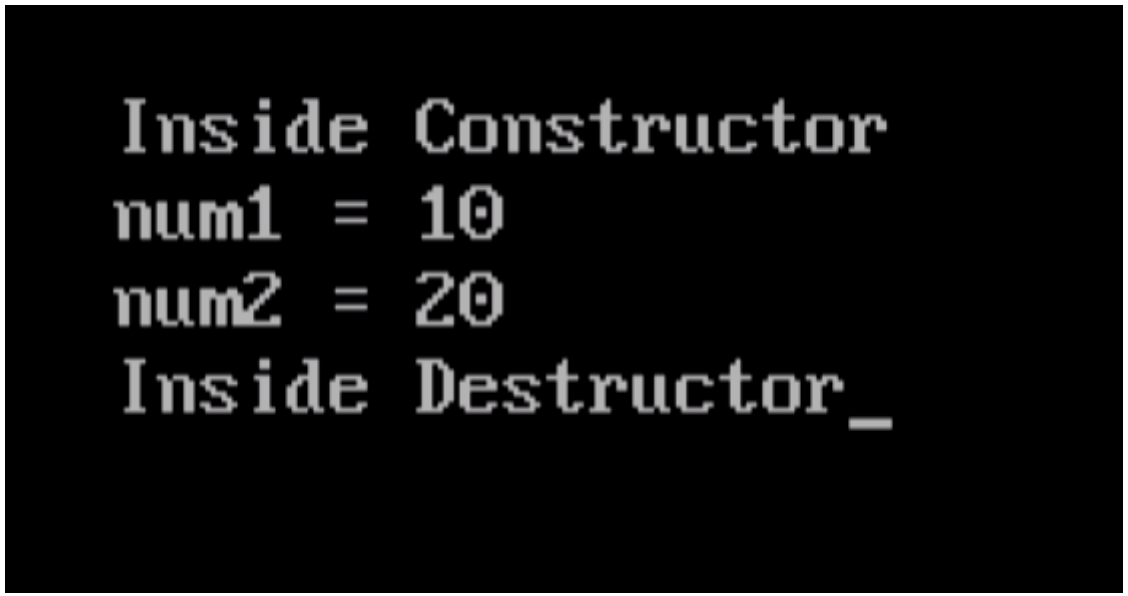
A destructor function is called automatically when the object goes out of scope:
(1) the function ends
(2) the program ends
(3) a block containing local variables ends
(4) a delete operator is called

How destructors are different from a normal member function?

Destructors have same name as the class preceded by a tilde (~)
Destructors don’t take any argument and don’t return any value

```
#include<iostream.h>
#include<conio.h>

class Demo {
  private:
  int num1, num2;
  public:
  Demo(int n1, int n2) {
    cout<<"\nInside
Constructor"<<endl;
    num1 = n1;
    num2 = n2;
  }
void display() {
    cout<<"num1 = "<< num1 <<endl;
    cout<<"num2 = "<< num2 <<endl;
  }
  ~Demo() {
    cout<<"Inside Destructor";
  }
};
void main() {
clrscr();
 { Demo obj1(10, 20);
  obj1.display();
  }
  getch();
}
```


Inside Constructor
num1 = 10
num2 = 20
Inside Destructor_