Object-Oriented Programming OOP

OUTLINES

- What is OOP
- Why OOP
- Class and object
- Create a class
- Create an object

Why Do We Need Object-Oriented Programming?

Object-Oriented Programming was developed because limitations were discovered in earlier approaches to programming.

To appreciate what OOP does, we need to understand what these limitations are and how they arose from traditional programming languages

Procedural Languages

- C, Pascal, FORTRAN, and similar languages are *procedural languages*
- A program in a procedural language is a list of instructions
- For very small programs, no other organizing principle is needed

Procedural Languages

- When programs become larger
- A procedural program is divided into functions, each function has
  - a clearly defined purpose and
  - a clearly defined interface to the other functions in the program.

Problems with Structured Programming

As programs grow ever larger and more complex many problems may occur

- **The project is too complex**
- **more programmers are added**
- **complexity increases**
- **……..**

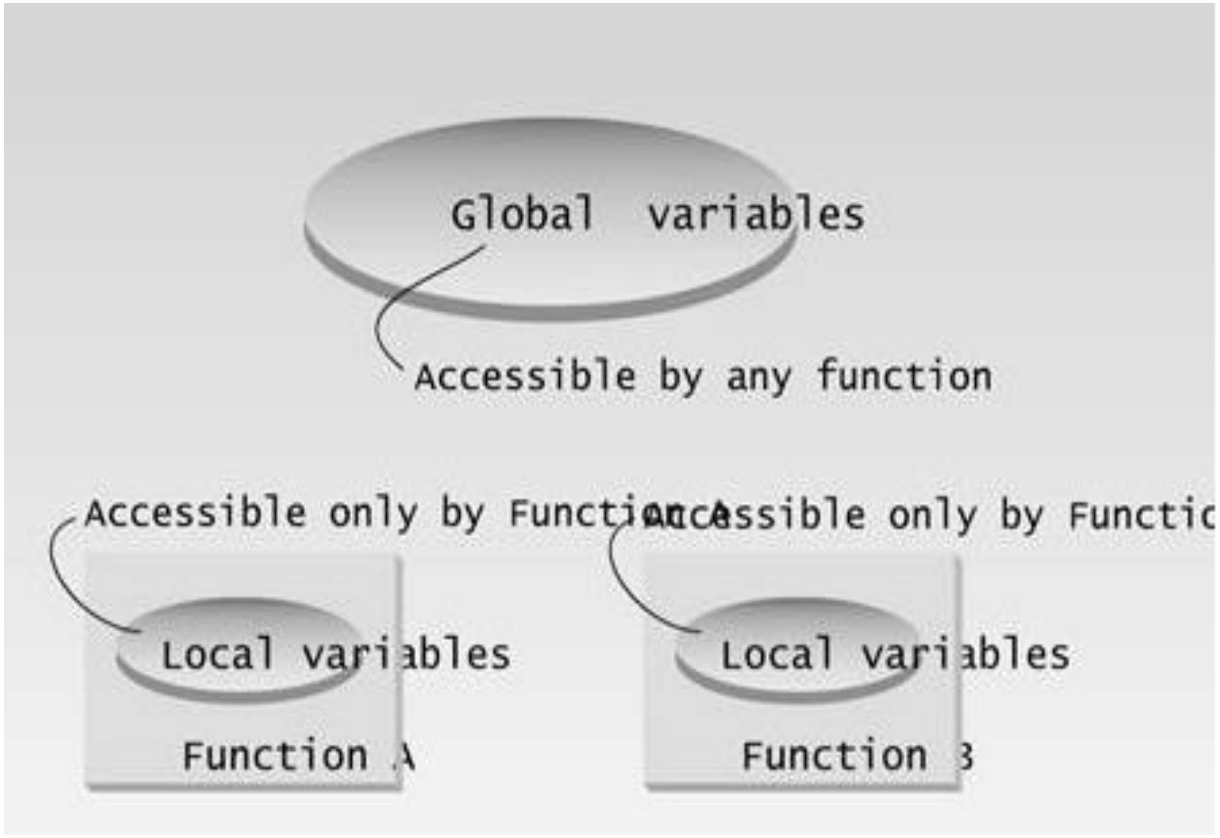
Problems with Structured Programming

There are two related problems.

- First, functions have unrestricted access to global data.
- Second, unrelated functions and data provide a poor model of the real world.

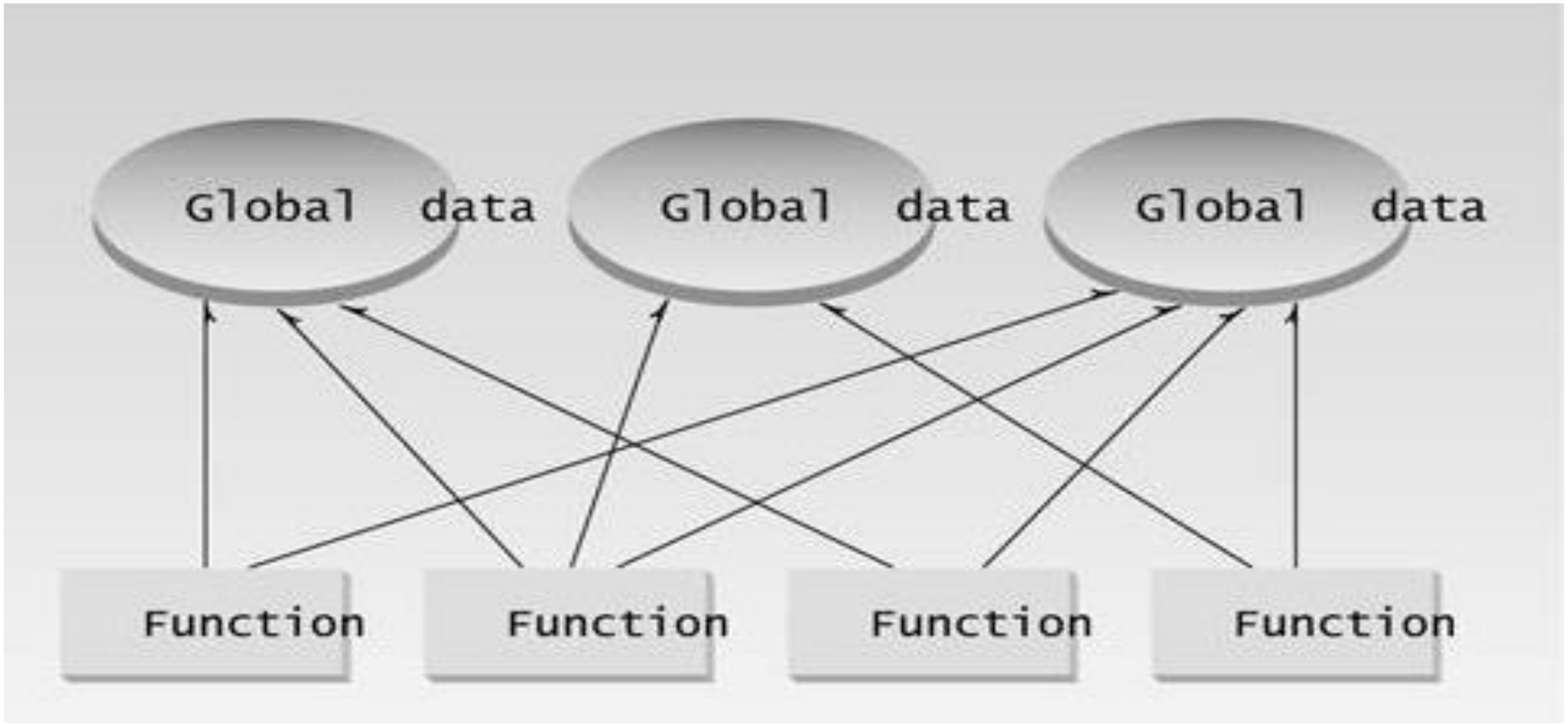
Problems with Structured Programming

- **Unrestricted Access:** Global data can be accessed by *any* function in the program


Global variables
Accessible by any function
Accessible only by Function A
Accessible only by Functic
Local variables
Function A
Local variables
Function B

Problems with Structured Programming

- **The procedural paradigm**



Problems with Structured Programming

- **Real-World Modeling**
- The third problem with the procedural paradigm is that its arrangement of separate data and functions does a poor job of modeling things in the real world

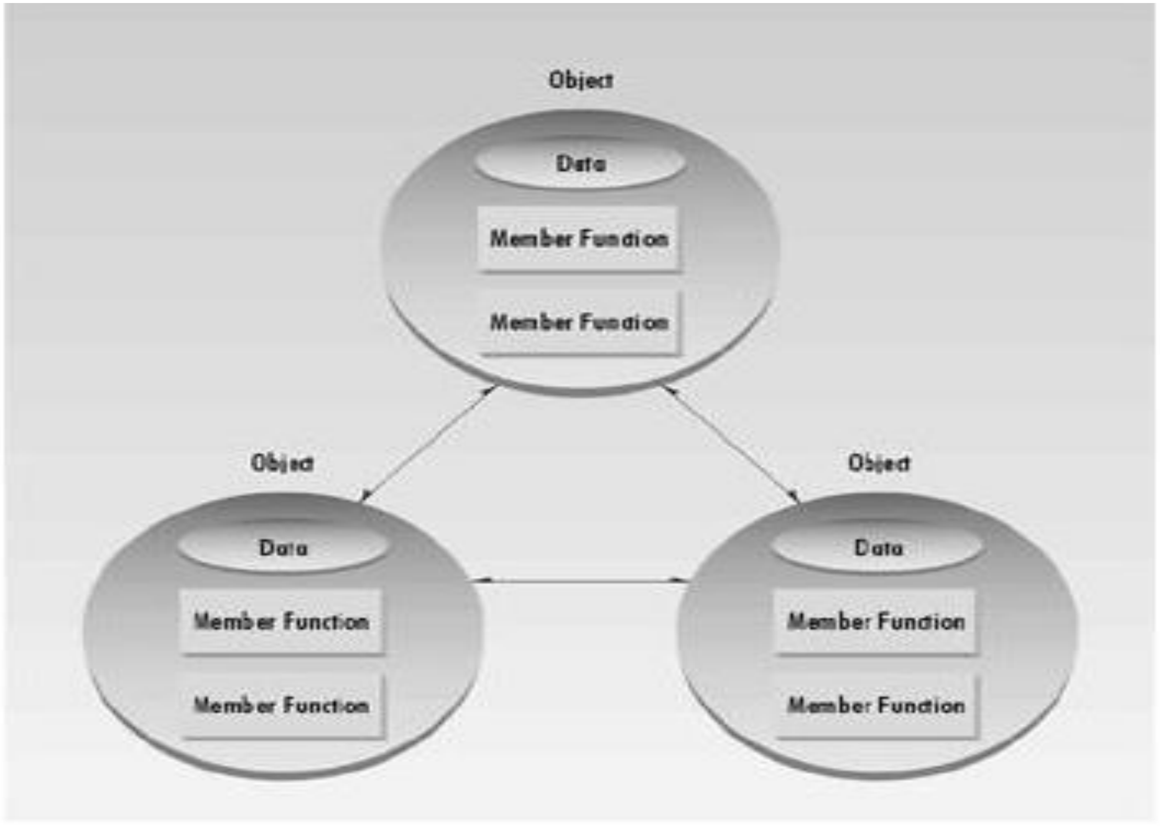
- In the physical world we deal with objects such as people and cars.
- Such objects aren't like data and they aren't like functions.
- Complex real-world objects have both *attributes* and *behavior.*

The Object-Oriented Approach

- The idea in OOP is to combine into a single unit both ***data*** and the ***functions*** *that operate on that data*.
- Such a unit is called an *object.*

- **These are often referred to as "class members".**
- **Attributes =→ variables**
- **Methods =→ functions**

- If you want to read a data item in an object, you call a member function in the object.
  - It will access the data and return the value to you.
  - You can’t access the data directly.
- The data is ***hidden***, so it is safe from accidental alteration.
- Data and its functions are said to be ***encapsulated*** into a single entity.
- ***Data encapsulation*** and ***data hiding*** are key terms in the description of object-oriented languages.


Object
Data
Member Function
Member Function
Object
Data
Member Function
Member Function
Object
Data
Member Function
Member Function

- Classes and objects are the two main aspects of object-oriented programming:
- **A class is a template for objects**
- **An object is an instance of a class**

Example: in real life, a car is an **object**.

- The car has **attributes**, such as
    - weight
    - color
- and **methods**, such as
    - drive
    - brake

Why OOP

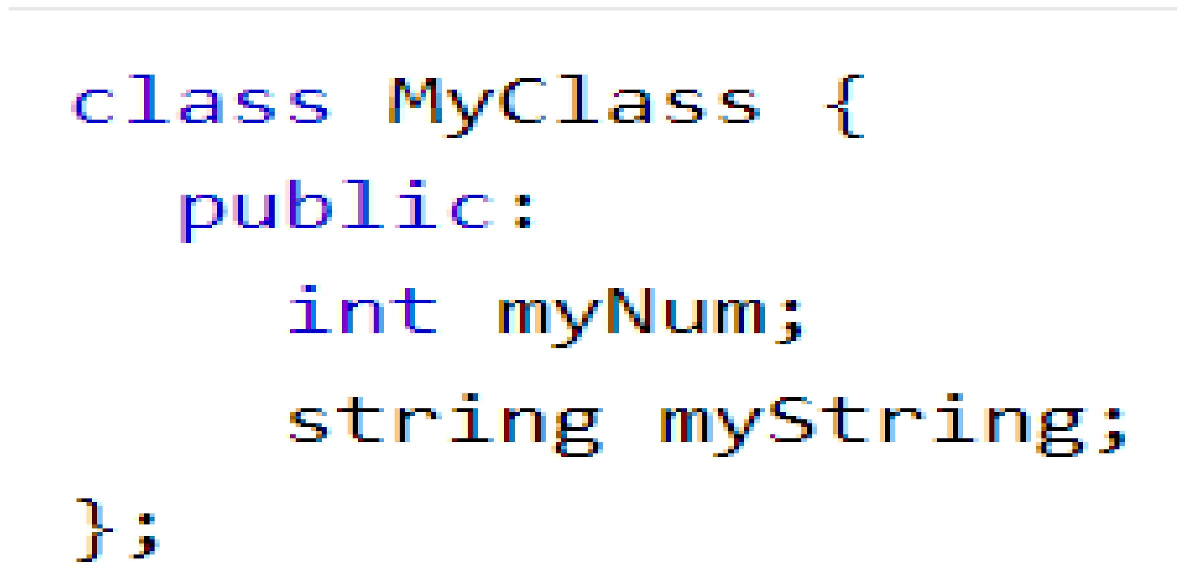
- OOP provides a clear structure for the programs
- OOP helps to make the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

- Almost all computer languages have **built-in** data types
  - int
  - char
  - float

- **A class is a user-defined data type**
  - We can define many objects of the same class

Create a Class

- **To create a class, use the class keyword**
- Example



```
class MyClass {
  public:
    int myNum;
    string myString;
};
```

```
class stud                              //declare a class
{
public:
        int  stuId;                     //class data
        string  stuName;                //class data
};
main()
{
stud student1;                                  //define an object of class stud
student1. stuId=100;                            //direct access to data
student1. stuName ="sami";                      //direct access to data
cout << student1. stuId << endl<< student1. stuName;
}
```

Class Methods

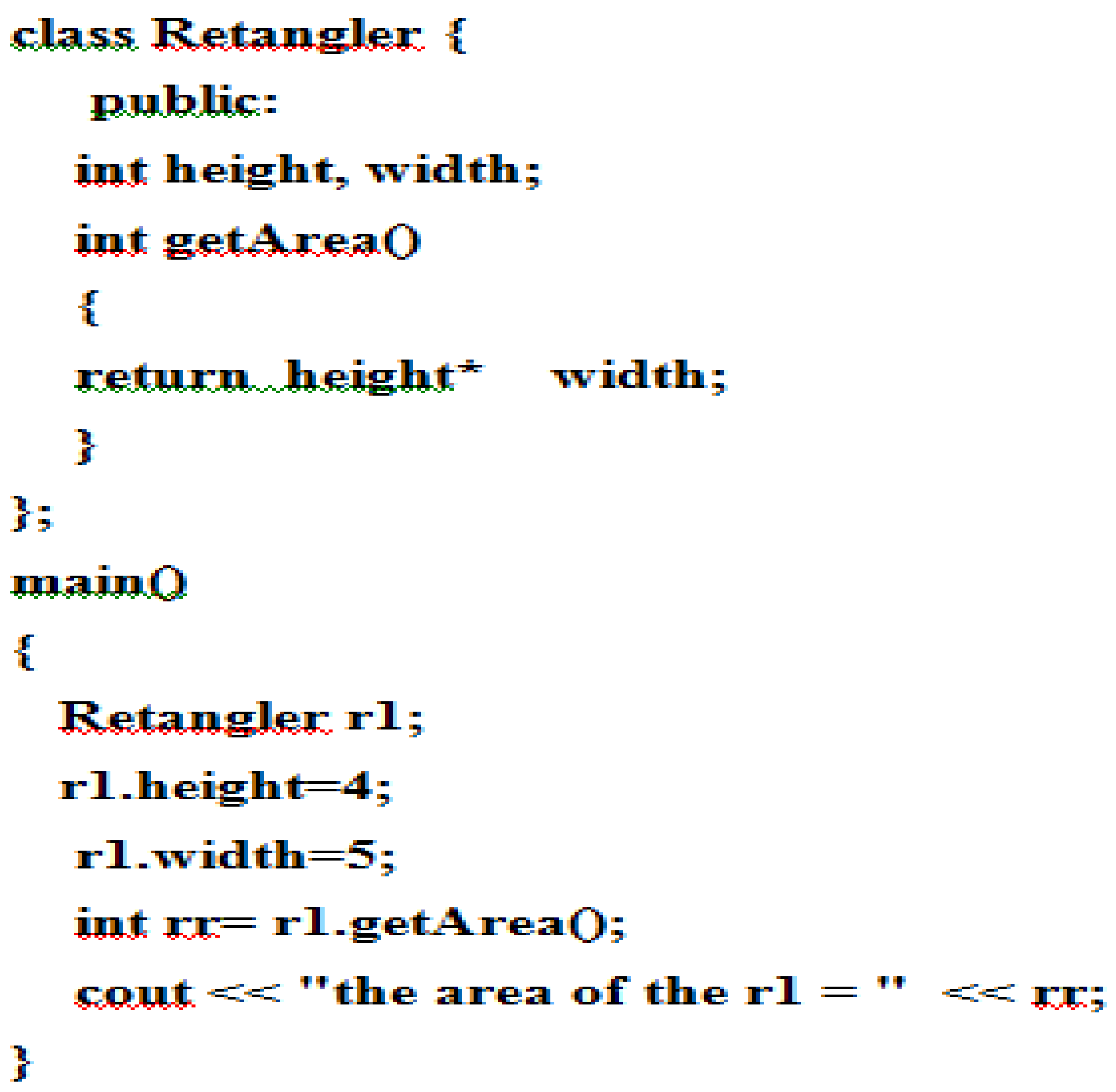
Methods are functions that belongs to the class.

- **There are two ways to define functions that belongs to a class:**
  - ✓ **Inside class definition.**
  - ✓ **Outside class definition**

Inside class definition

```
class stud                    //declare a class
{
public:
        int  stuId;           //class data
        string  stuName;      //class data
        void displayData()
        {
        cout <<  stuId << endl<<  stuName;
       }
};
main()
{
stud student1;                          //define an object of class stud
student1. stuId=100;                    //direct access to data
student1. stuName ="sami";              //direct access to data
student1. displayData ( );
}
```

Inside class definition



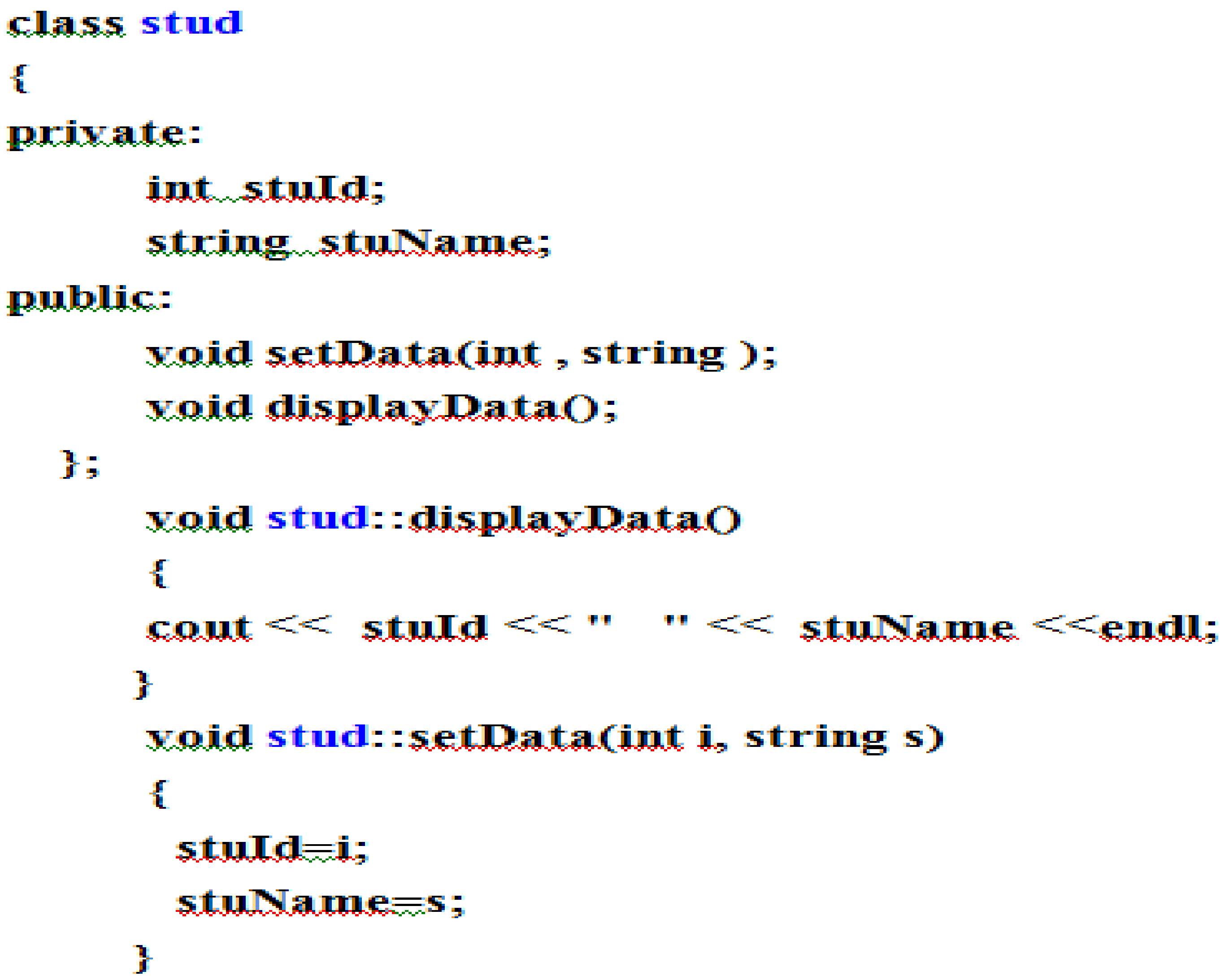
```
class Retangler {
     public:
    int height, width;
    int getArea()
    {
    return  height*    width;
    }
};
main()
{
   Retangler r1;
   r1.height=4;
    r1.width=5;
    int rr= r1.getArea();
    cout << "the area of the r1 = "  << rr;
}
```

Outside class definition

To define a function outside the class:

1. **Declare it inside the class**
2. **Define it outside of the class**.
   - This is done by **specifiying the name of the class, followed the scope resolution :: operator, followed by the name of the functio**n:

Outside class definition



```
class stud
{
private:
        int  stuId;
        string  stuName;
public:
        void setData(int , string );
        void displayData();
   };
        void stud::displayData()
        {
        cout <<  stuId << "    " <<  stuName <<endl;
        }
        void stud::setData(int i, string s)
        {
          stuId=i;
          stuName=s;
        }
```

Function Inside class

```
class Sum {
public:
   int x, y;

    int Result()
   {
      return x + y;
   }
  };
```

Function Outside class

```
class Sum {
public:
   int x, y;
   int Result();
  };
int Sum::Result()
{
   return x + y;
}
```