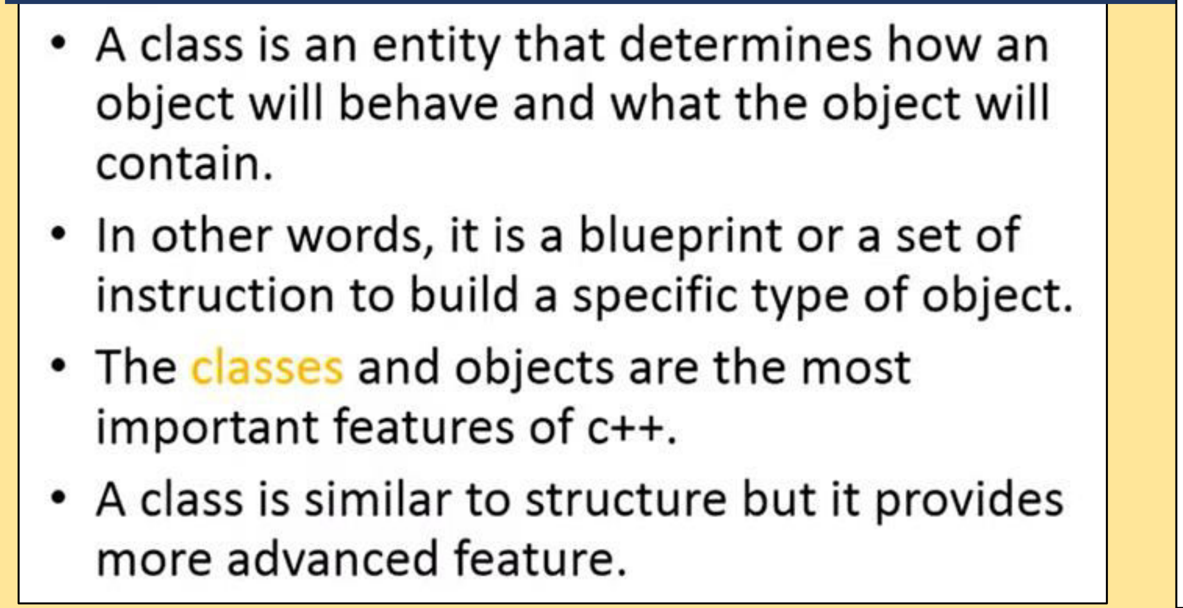
CSE 1203 (Object-oriented Programming)

Chapter 2

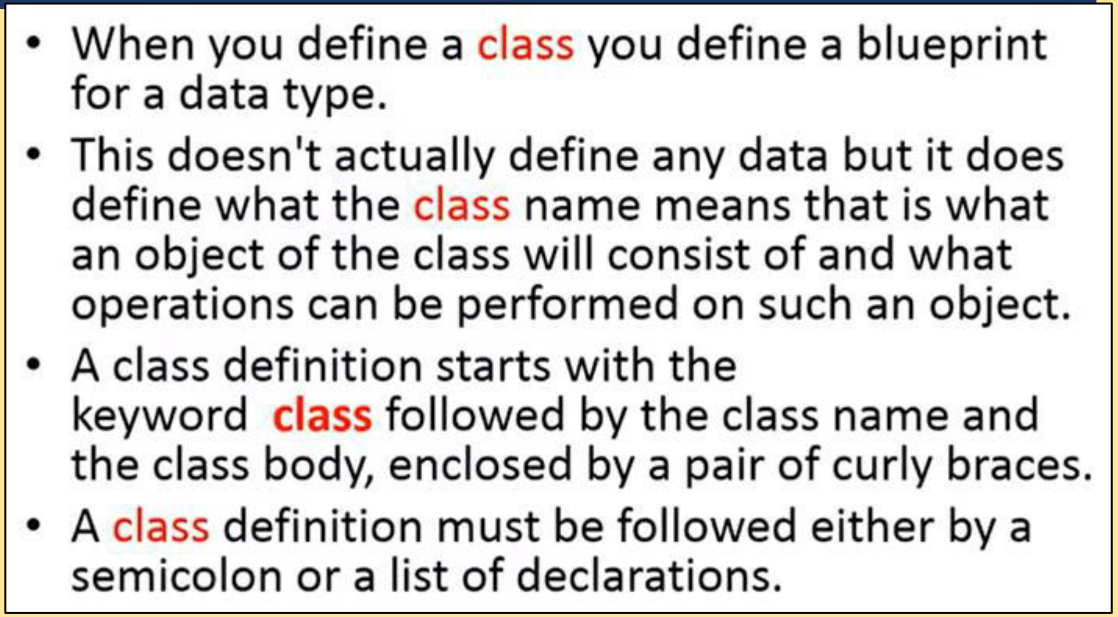
Classes & Objects in OOP

Class in OOP

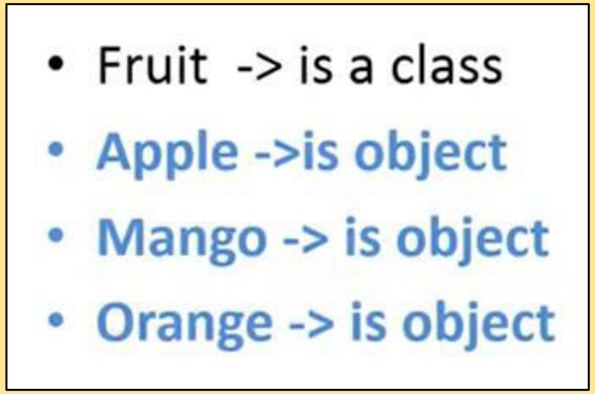
What is a class?



- A class is an entity that determines how an object will behave and what the object will contain.
- In other words, it is a blueprint or a set of instruction to build a specific type of object.
- The classes and objects are the most important features of c++.
- A class is similar to structure but it provides more advanced feature.



- When you define a class you define a blueprint for a data type.
- This doesn't actually define any data but it does define what the class name means that is what an object of the class will consist of and what operations can be performed on such an object.
- A class definition starts with the keyword **class** followed by the class name and the class body, enclosed by a pair of curly braces.
- A class definition must be followed either by a semicolon or a list of declarations.



- Fruit -> is a class
- **Apple ->is object**
- **Mango -> is object**
- **Orange -> is object**

Vehicle -> is a class

Car -> is object

bus -> is object

Truck -> is object

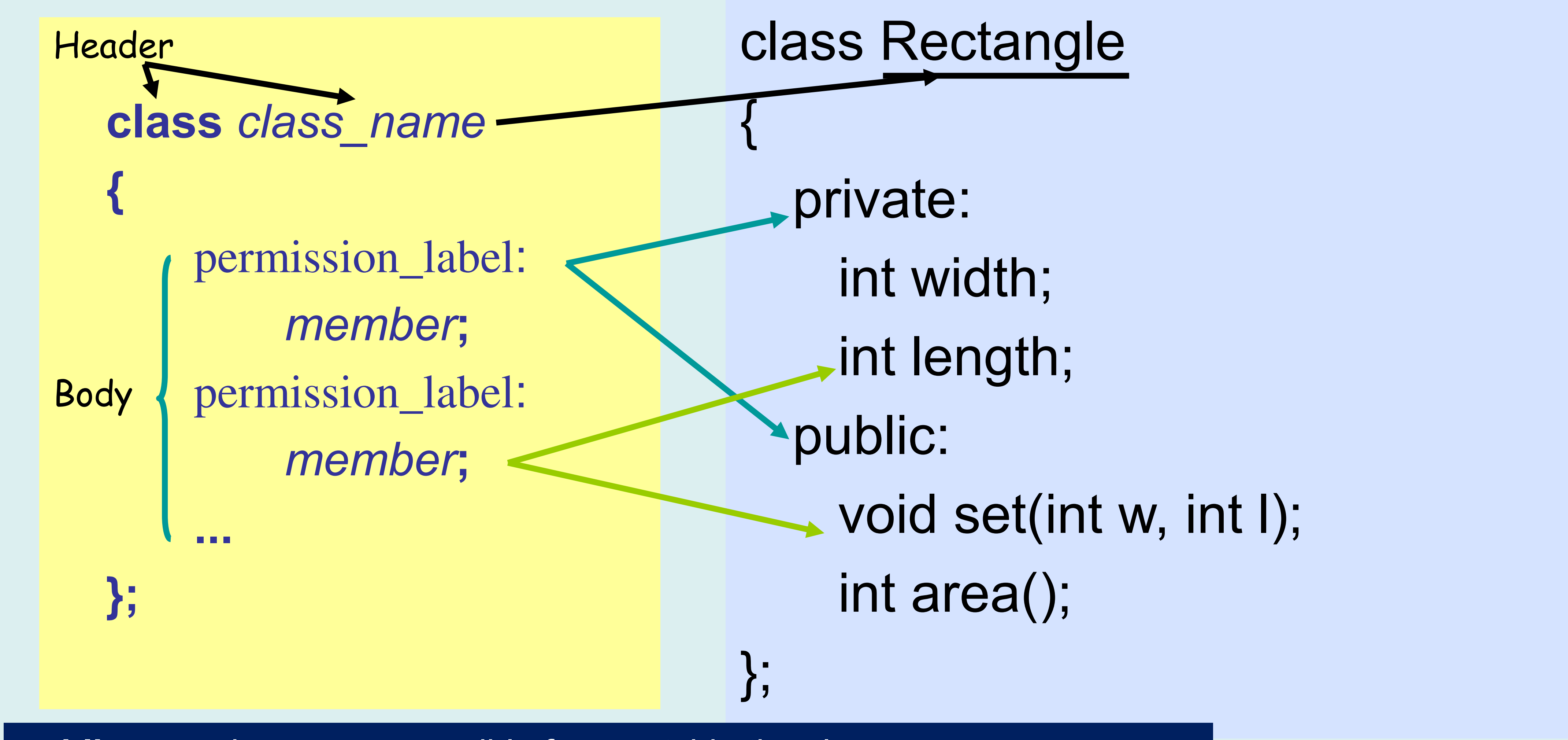
University -> is a class

RUET -> is object

BUET -> is object

Cambridge -> is object

Define a Class Type



Header

Body

```
class class_name
{
    permission_label:
        member;
    permission_label:
        member;
    ...
};
```

```
class Rectangle
{
  private:
    int width;
    int length;
  public:
    void set(int w, int l);
    int area();
};
```

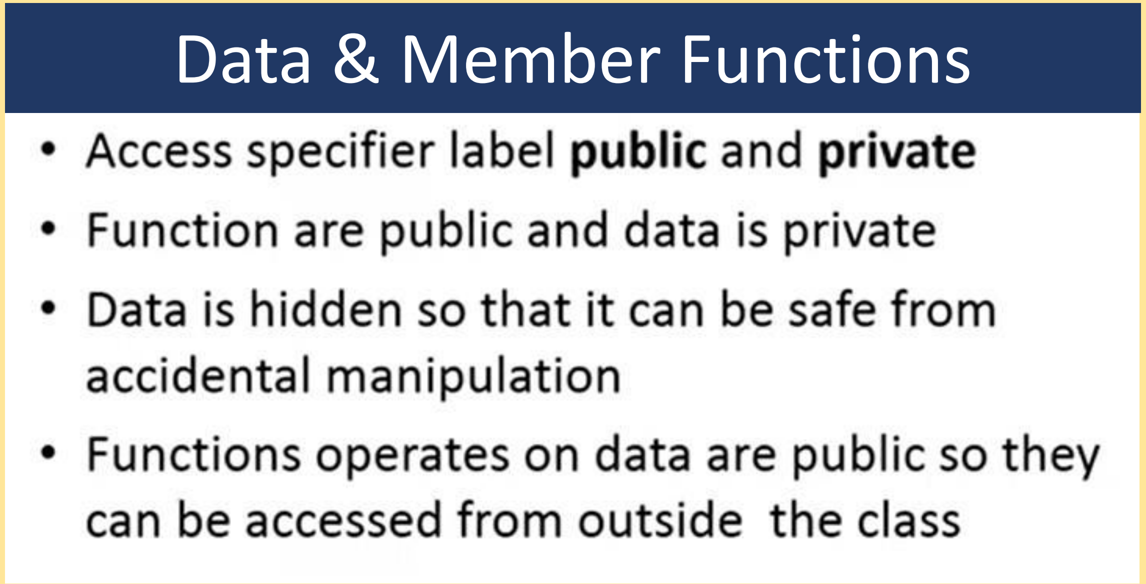
public - members are accessible from outside the class.
private - members cannot be accessed (or viewed) from outside the class.

Class in OOP: C++ example

```
#include <iostream>

using namespace std;

class Rectangle{
 private:
    int height;
    int width;
 public:
    void SetData(int h,int w){
     height=h;
     width=w;
    }
    void Display(){
     cout<<"Height="<<height<<" Weight="<<width;
    }
};
int main()
{
   Rectangle r1,r2;
   r1.SetData(3,4);
   r1.Display();
   return 0;
}
```

Question:

1. What happens if private keyword is replaced by public?
2. What happens if public keyword is replaced by private?



Data & Member Functions

- Access specifier label **public** and **private**
- Function are public and data is private
- Data is hidden so that it can be safe from accidental manipulation
- Functions operates on data are public so they can be accessed from outside the class

Class in OOP: Data & Member Function

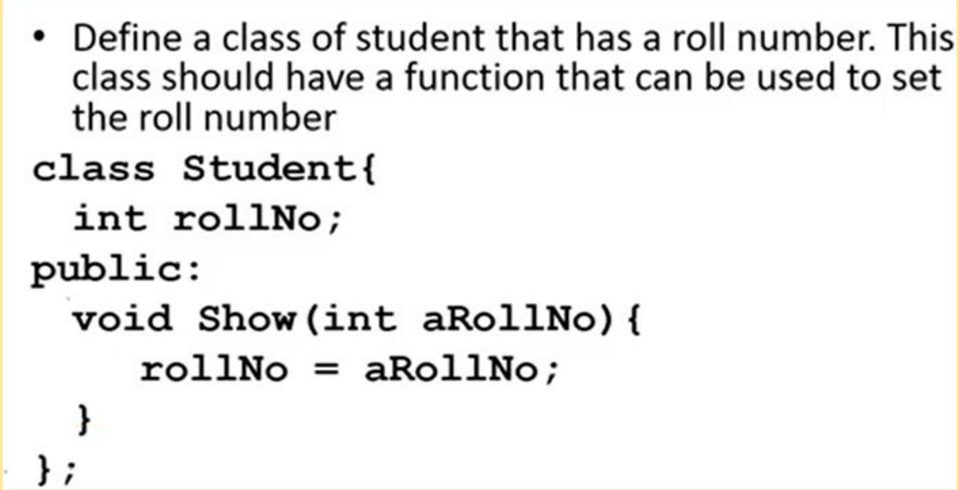
Data & Member Functions

- Member functions are the functions that operate on the data encapsulated in the class
- Public member functions are the interface to the class
- Define member function inside the class definition

OR

- Define member function outside the class definition

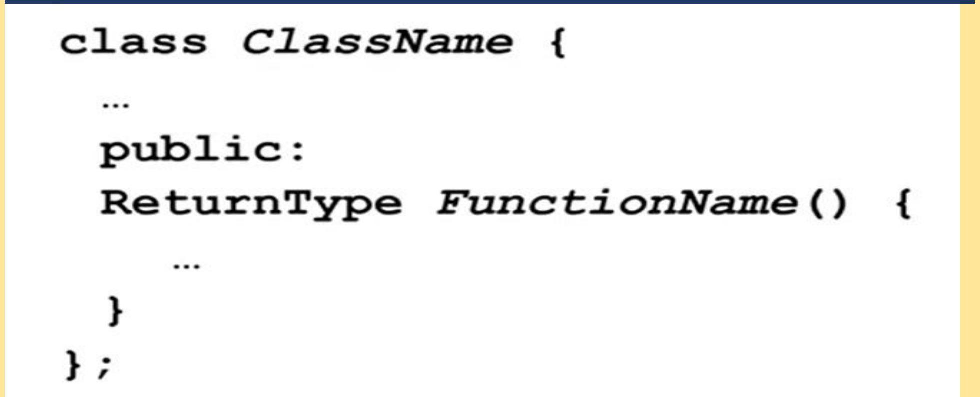
Example



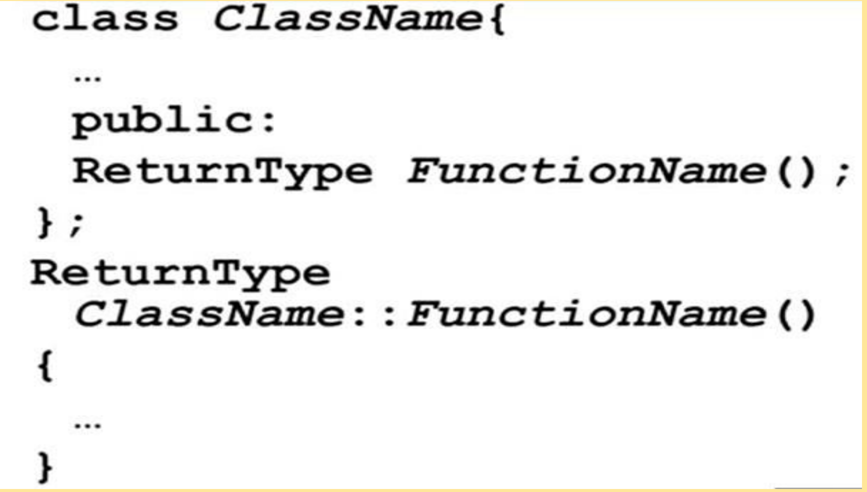
- Define a class of student that has a roll number. This class should have a function that can be used to set the roll number

```
class Student{
  int rollNo;
public:
  void Show(int aRollNo){
     rollNo = aRollNo;
  }
};
```

Member Functions inside the Class



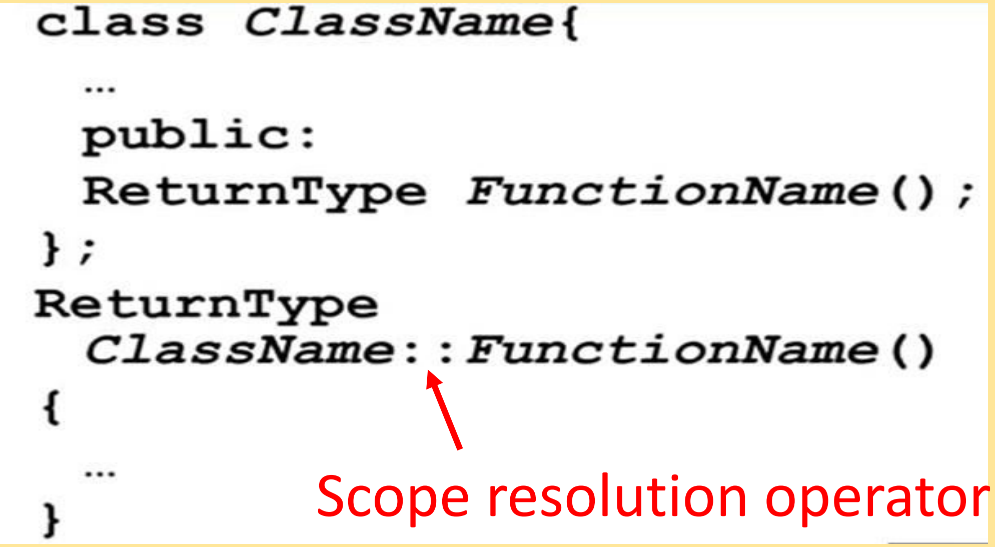
```
class ClassName {
  …
  public:
  ReturnType FunctionName() {
     …
  }
};
```



```
class ClassName{
  …
  public:
  ReturnType FunctionName();
};
ReturnType
  ClassName::FunctionName()
{
  …
}
```

Class in OOP: Data & Member Function

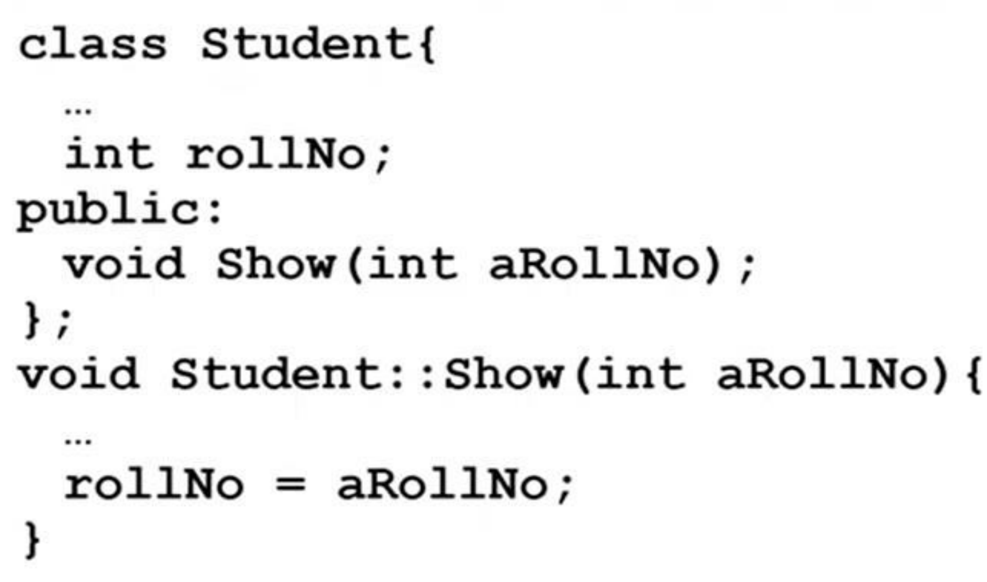
Member Functions outside the Class



```
class ClassName{
  …
  public:
  ReturnType FunctionName();
};
ReturnType
  ClassName::FunctionName()
{
  …
}
```

Scope resolution operator

Example

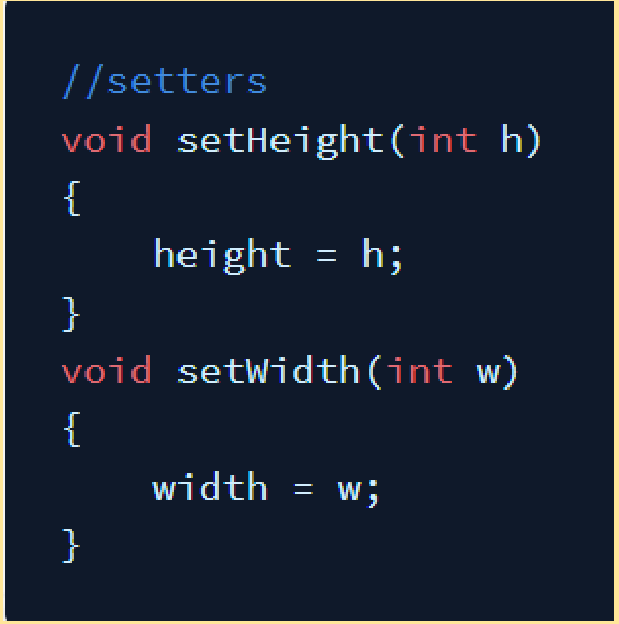


```
class Student{
  …
  int rollNo;
public:
  void Show(int aRollNo);
};
void Student::Show(int aRollNo){
  …
  rollNo = aRollNo;
}
```

Class in OOP: Setters & Getters

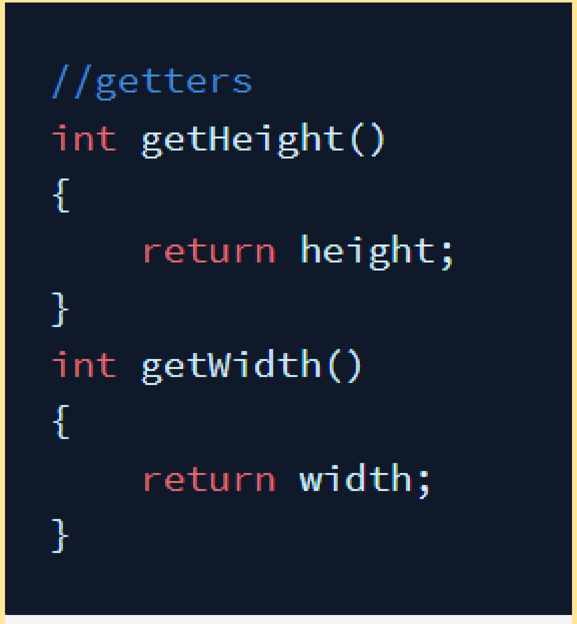
- **Setters** allow for a private variable to be modified.
- **Getters** give public access to private data.

Example: setters()



```cpp
//setters
void setHeight(int h)
{
    height = h;
}
void setWidth(int w)
{
    width = w;
}
```

Example: getters()

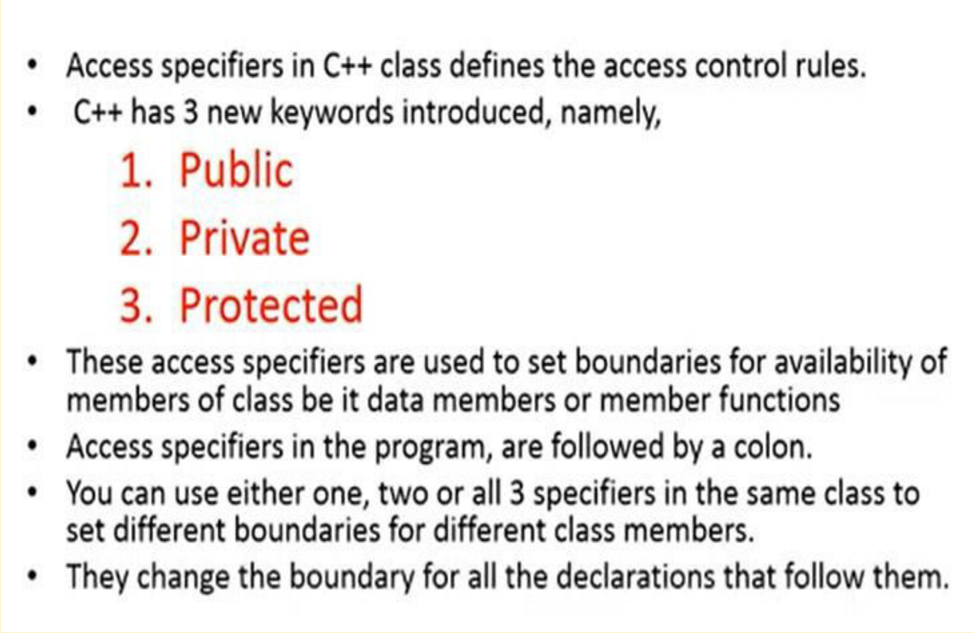


```cpp
//getters
int getHeight()
{
    return height;
}
int getWidth()
{
    return width;
}
```

Write a complete C++ program using these setters and getters

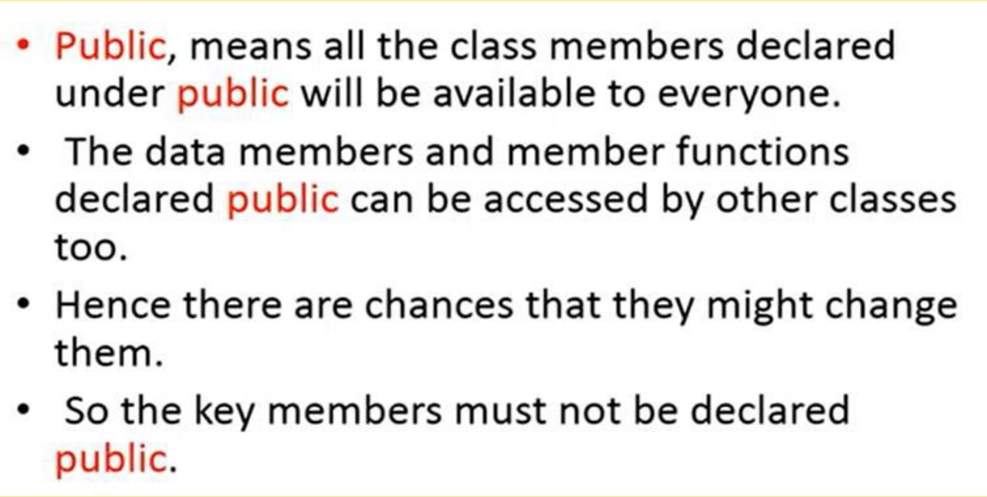
Class in OOP: Access Control

Access Specifiesrs



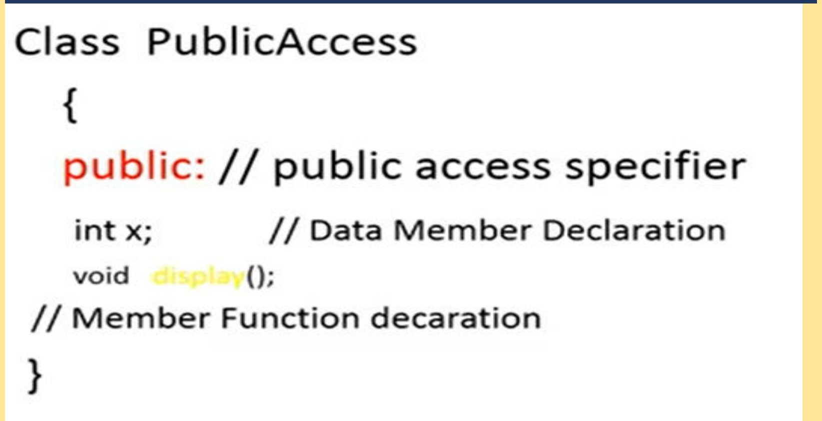
- Access specifiers in C++ class defines the access control rules.
- C++ has 3 new keywords introduced, namely,
  1. Public
  2. Private
  3. Protected
- These access specifiers are used to set boundaries for availability of members of class be it data members or member functions
- Access specifiers in the program, are followed by a colon.
- You can use either one, two or all 3 specifiers in the same class to set different boundaries for different class members.
- They change the boundary for all the declarations that follow them.

public specifier



- Public, means all the class members declared under public will be available to everyone.
- The data members and member functions declared public can be accessed by other classes too.
- Hence there are chances that they might change them.
- So the key members must not be declared public.

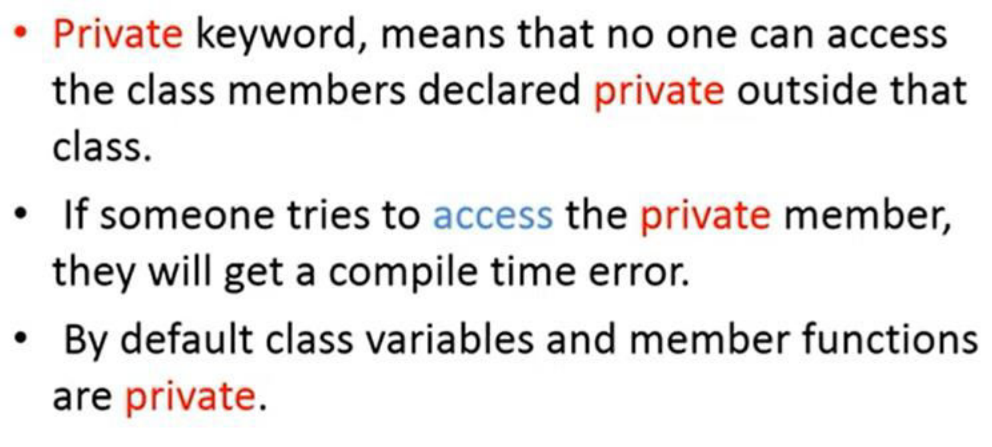
public specifier: Example



```
Class  PublicAccess
   {
   public: // public access specifier
    int x;          // Data Member Declaration
    void  display();
 // Member Function decaration
 }
```

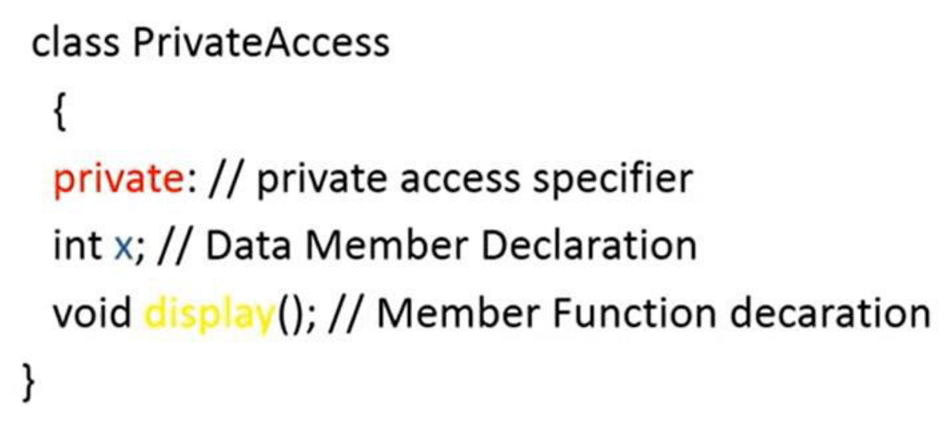
Class in OOP: Access Control

private specifier



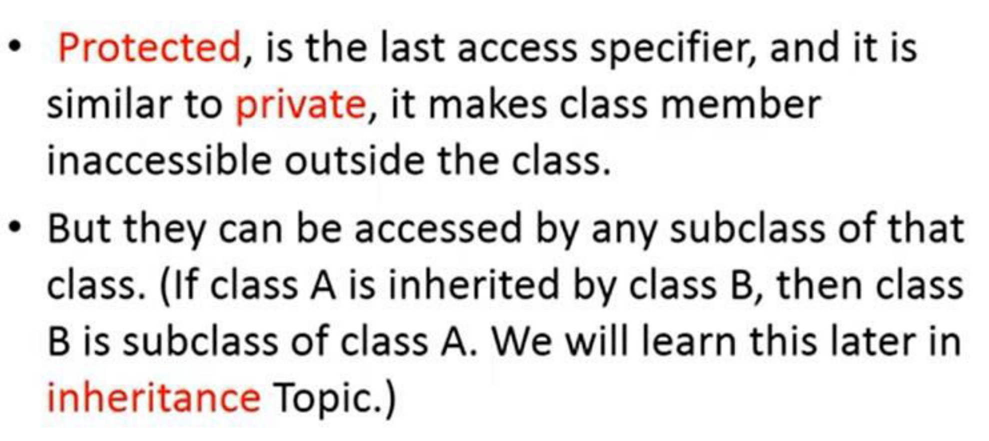
- Private keyword, means that no one can access the class members declared private outside that class.
- If someone tries to access the private member, they will get a compile time error.
- By default class variables and member functions are private.

private specifier: Example



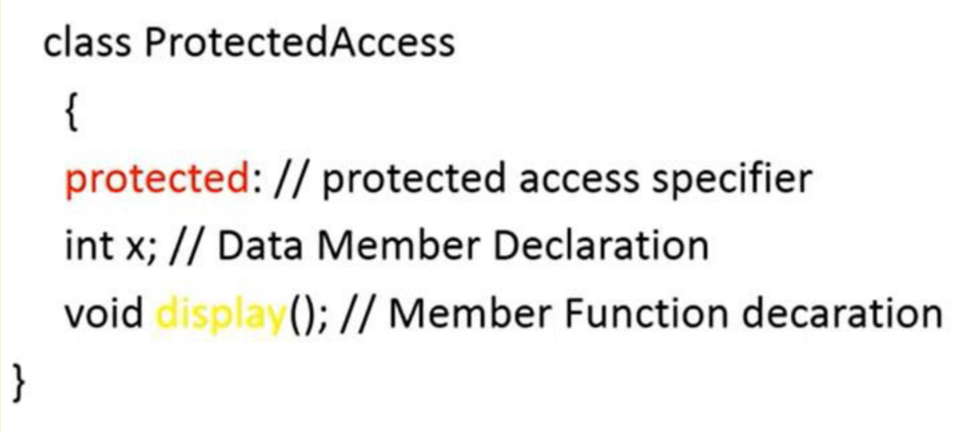
```
class PrivateAccess
 {
 private: // private access specifier
 int x; // Data Member Declaration
 void display(); // Member Function decaration
}
```

protected specifier



- Protected, is the last access specifier, and it is similar to private, it makes class member inaccessible outside the class.
- But they can be accessed by any subclass of that class. (If class A is inherited by class B, then class B is subclass of class A. We will learn this later in inheritance Topic.)

protected specifier: Example



```
class ProtectedAccess
 {
 protected: // protected access specifier
 int x; // Data Member Declaration
 void display(); // Member Function decaration
}
```

Class in OOP: Constructors

What is a constructor?

- C++ requires a construct call for each object it has created
- Constructors are special class functions which performs initialization of every object.
- The Compiler calls the Constructor whenever an object is created.
- If there is no constructor, the compiler provides a **default constructor** that is, a constructor with no parameters
- Name of constructor function is same as name of class
- constructor has no return type

Example

```
Class Test{
            Private:
                  int n;
          Public:
                 Test(){
                 n = 0;
                  }
};
```

Constructor: Initializaton

- One of the most common tasks a constructor carries out is initializing data members
- In the Test class the constructor must initialize the n member to 0
- The initialization takes place following the member function declarator but before the function body.
- Initialization in constructor's function body

```
      Test()
      { n = 0; }
```

- It's preceded by a colon. The value is placed in parentheses following the member data.

```
Test() : n(0)
{  }
```

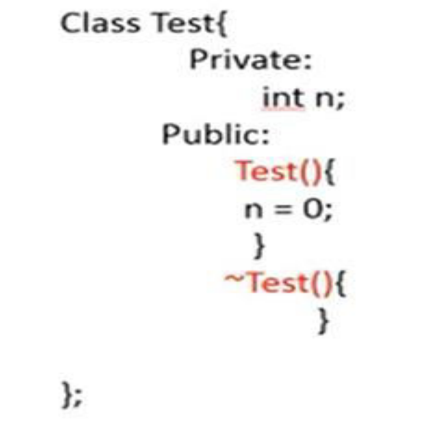
- If multiple members must be initialized, they're separated by commas.
  - Test() : n1(7), n2(8), n3(4) ←initializer list { }

Class in OOP: Destructors

What is a destructor?

- Destructor is a function called implicitly when an object is destroyed
- The name of the destructor for a class is the **tilde character (~)** followed by the class name
- No arguments and no return type for a destructor
- The most common use of destructors is to deallocate memory that was allocated for the object by the constructor

Example



```
Class Test{
        Private:
              int n;
      Public:
           Test(){
            n = 0;
            }
          ~Test(){
                }

};
```

Destructor: C++ Program

```
class Test{
  public:
      Test(){
       cout<<"Constructor is called"<<endl;
      }
      ~Test(){
       cout<<"Destructor is called"<<endl;
      }
};
void CreateObj(){
 Test t1,t2;
 cout<<"Inside the CreateObj()"<<endl;
}
int main()
{
    CreateObj();
    return 0;
}
```

Class in OOP: Class in Different File

What is a destructor?

```
#include <iostream>
#include "reg.h“

using namespace std;

int main(){
  Circle c1;
  c1.SetRadius(3);
  cout<<c1.GetArea();
}
```

reg.h

```
class Circle{
  private:
      int r;
  public:
      void SetRadius(int x){
        r=x;
      }
      float GetArea(){
       return(3.14*r*r);
      }
};
```

Class in OOP: static Data Member

What is a static data member?

Static data members are class members that are declared using static keywords. A static member has certain special characteristics which are as follows:

- Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.
- It is initialized before any object of this class is created, even before the main starts.
- It is visible only within the class, but its lifetime is the entire program.
- The value must be initialized outside the class
- The getter function must be static

Syntax:

static data_type data_member_name;

```
// This program counts no. of objects
#include <iostream>
using namespace std;
class Base{
 int x;
 static int y;
 public:
 Base(int X){
 x=X;
 y++;
 }
 static int getY(){ return y;}
};
int Base::y=0;
int main()
{
 Base c1(10),c2(20);
 cout<<Base::getY();
 return 0;
}
```

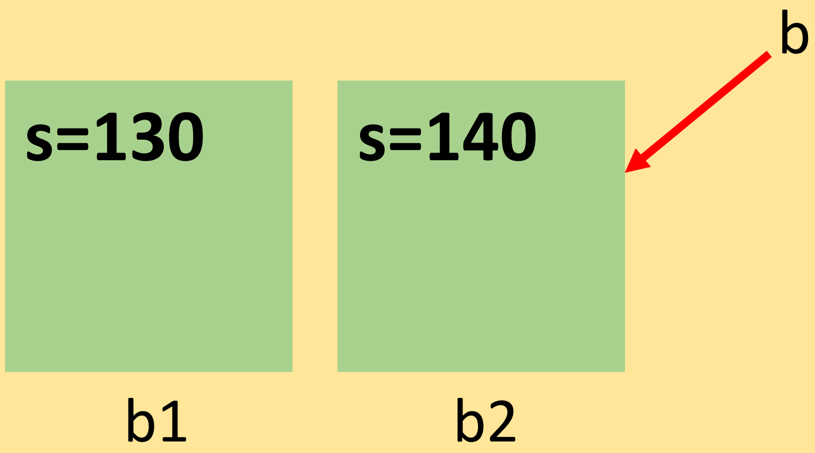
Class in OOP: Passing Object as Parameter

Example

```
#include <iostream>
using namespace std;

class Ball{
 private:
 int s;
 public:
  Ball(){}
  Ball(int x){
  s=x;
 }
  void AvgSpeed(Ball *b){
   cout<<(s+b->s)/2;
  }
};
```

Contd..

```
int main()
{
 Ball b1(130),b2(140);
 b1.AvgSpeed(&b2);
 return 0;
}
```



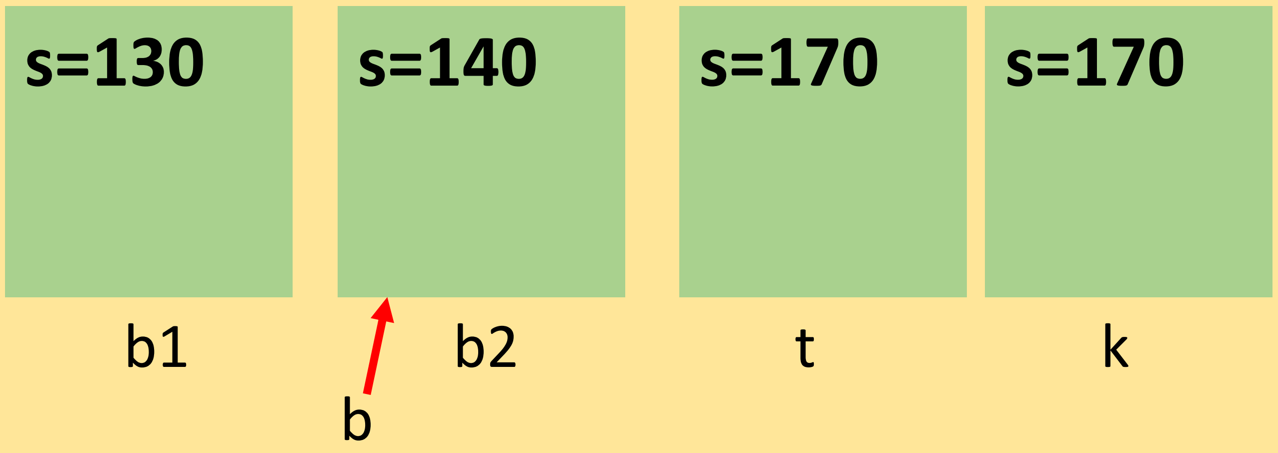
Class in OOP: Object as Function Return

Example

```
#include <iostream>
using namespace std;
class Ball{
 private:
 float s;
 public:
  Ball(){}
  Ball(int x){
  s=x;
 }
 float GetSpeed (){
  return s;
  }
  Ball AvgSpeed(Ball *b){
   Ball t;
   t.s=s+b->s;
   return(t);
  }
};
```

Contd..

```
int main()
{
 Ball b1(130),b2(140);
 Ball k;
 k=b1.AvgSpeed(&b2);
 cout<<k.GetSpeed()/2;
 return 0;
}
```



s=130 | s=140 | s=170 | s=170

b1 | b2 | t | k

b

Class in OOP: Copy Constructor

What is a Copy Constructor?

A **copy constructor** is a member function that initializes an object using another object of the same class. In simple terms, a constructor which creates an object by initializing it with an object of the same class, which has been created previously is known as a **copy constructor**.
Copy constructor is used to initialize the members of a newly created object by copying the members of an already existing object.

Copy constructor takes a reference to an object of the same class as an argument.

```
Sample(Sample &t) {
  id=t.id;
}
```

In Sample class

```
int main() {
 Sample s1(100);
 Sample s2(s1);
}
```

In main()(

```
int id
```

Sample Class

Class in OOP: Copy Constructor Example

Example

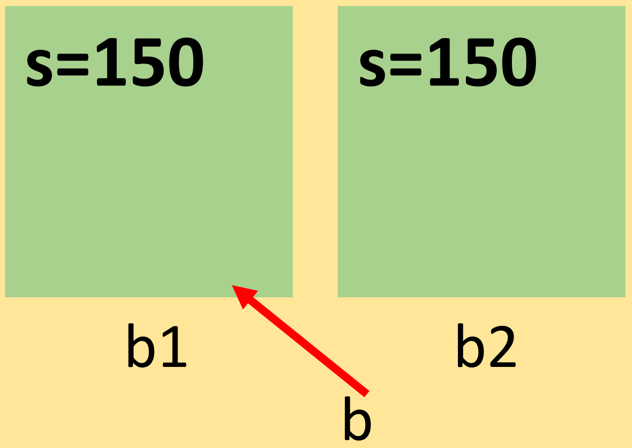
```
#include <iostream>
using namespace std;

class Ball{
 private:
 float s;
 public:
  Ball(){}
  Ball(int x){
  s=x;
 }
 Ball(Ball &b){
   s=b.s;
  }
 float GetSpeed (){
  return s;
  }
};
```

Copy constructor

Contd..

```
int main()
{
 Ball b1(150);
 Ball b2(b1);
 cout<<b2.GetSpeed();
 return 0;
}
```

Copy constructor



Class in OOP: const Member Function

const member function

declaration

return_type func_name (*para_list*) const;

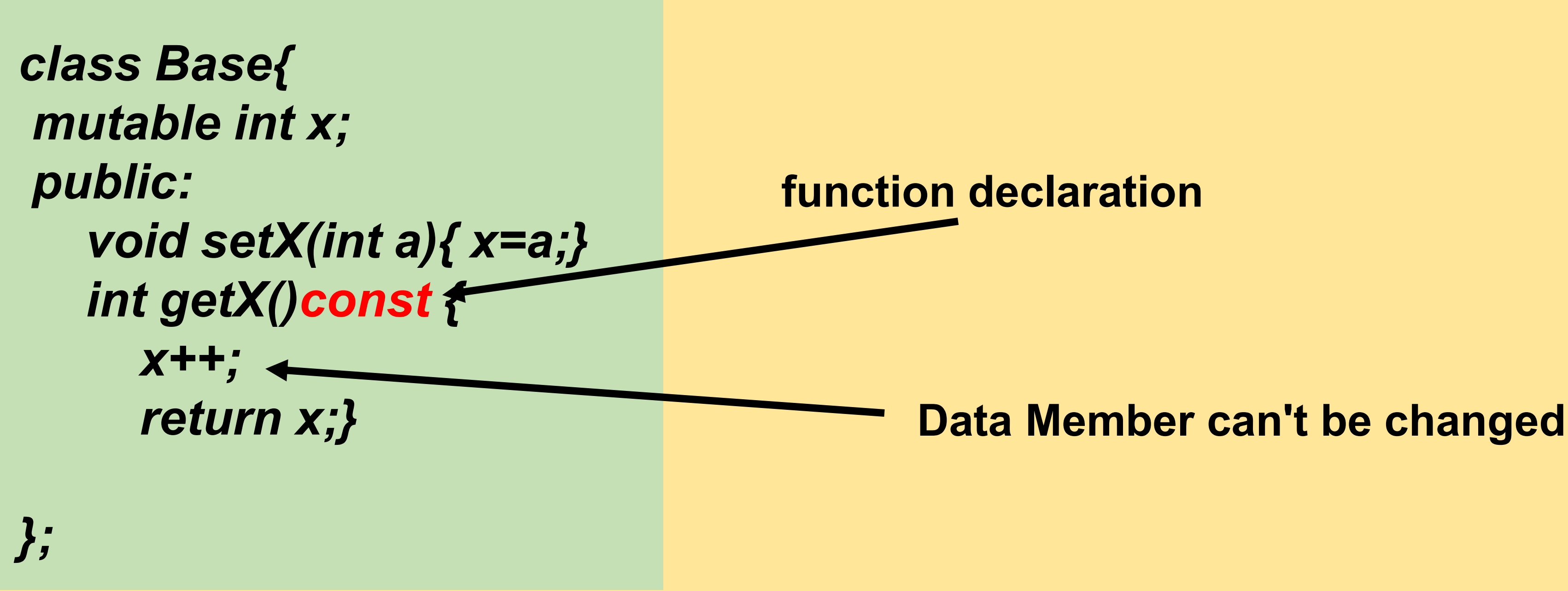
definition

return_type func_name (*para_list*) const { … }

return_type class_name :: *func_name* (*para_list*) const { … }

- Makes no modification about the data members (safe function)
- It is illegal for a const member function to modify a class data member

Class in OOP: const Member Function



```
class Base{
 mutable int x;
 public:
     void setX(int a){ x=a;}
     int getX()const {
         x++;
         return x;}

};
```

function declaration

Data Member can't be changed

To make const function executable →making data member mutable

Class in OOP: static Member Function

- **static** member function
  - Static member function can contain only static data member
  - Non-static Static member function can contain both static and non-static data member
  - Static function can run using

    <classname>::<static function()

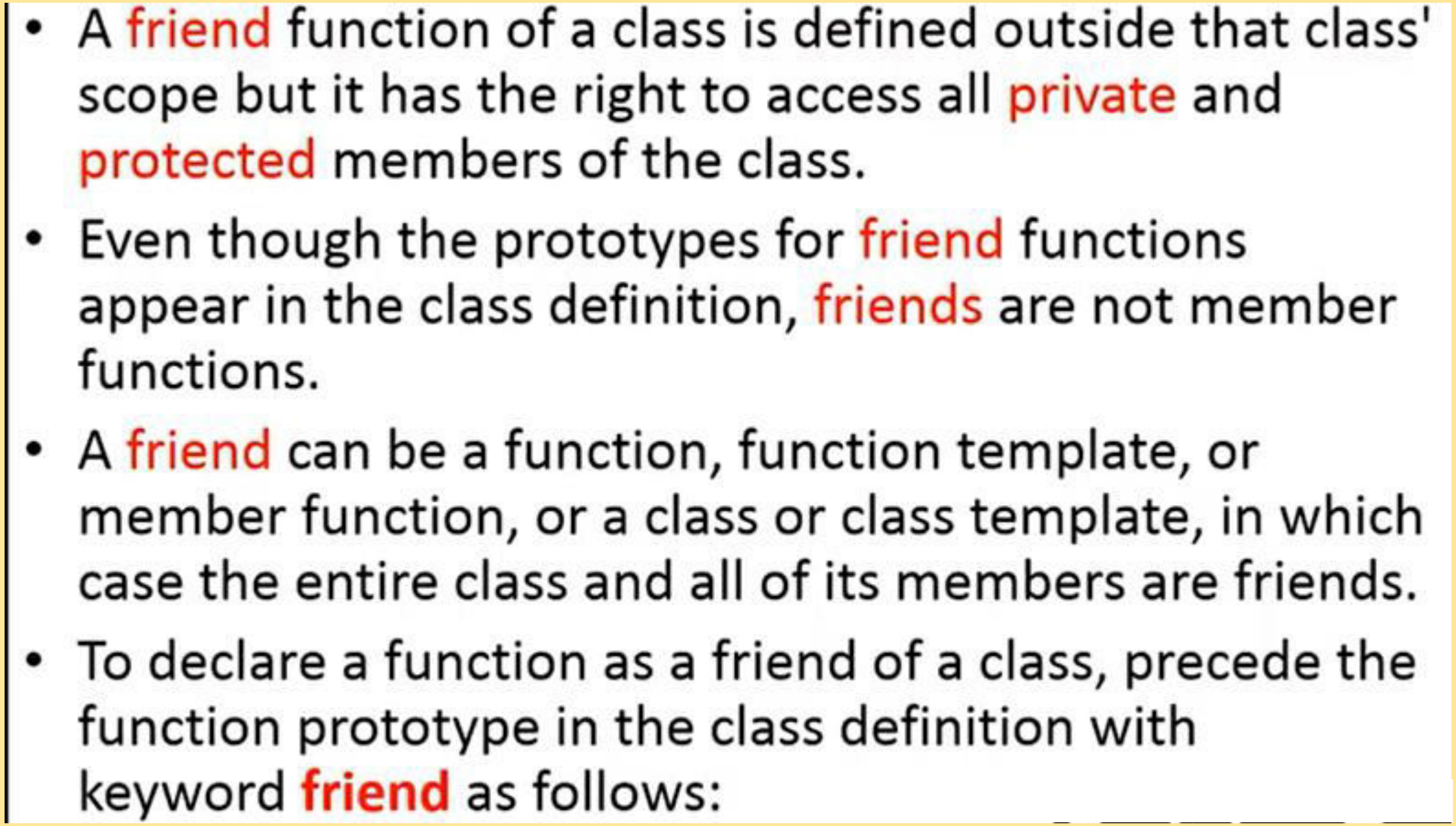
Class in OOP: static Member Function

```
class Base{
 int x;
 static int y;
 public:
 Base(int X){
  x=X;
  y++;
 }
 static int getY(){ return y;}
};

int Base::y=0;
```

```
int main()
{
  Base c1(10),c2(20);

  cout<<Base::getY();
    return 0;
}
```

Static method can be called by class name with scope resolution

Class in OOP: friend Member Function



- A friend function of a class is defined outside that class' scope but it has the right to access all private and protected members of the class.
- Even though the prototypes for friend functions appear in the class definition, friends are not member functions.
- A friend can be a function, function template, or member function, or a class or class template, in which case the entire class and all of its members are friends.
- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword **friend** as follows:

Class in OOP: friend Function Example_01

```
#include <iostream>
using namespace std;

class Test{
 private:
     int n;
 public:
     Test(int x){
      n=x;
     }
    friend void show(Test t);
};
void show(Test t){
 cout<<"n="<<t.n;
 }
```

```
int main()
{
 Test p(10);
 show(p);
 return 0;
}
```

Class in OOP: friend Function Example_02

```
#include <iostream>
using namespace std;

class B; //forwrad declaration
class A{
 private:
    int n;
 public:
    A(int x){
     n=x;
    }
   friend void Add(A,B);
};
```

```
class B{
 private:
    int n;
 public:
    B(int x){
     n=x;
    }
   friend void Add(A,B);
};
void Add(A a,B b){
 cout<<"Sum="<<a.n+b.n;
 }
```

```
int main()
{
 A oa(10);
 B ob(20);
 Add(oa,ob);
 return 0;
}
```

Here private members of different classes are added using friend function

Introduction to OOP

End