COURSE CODE: CSE 205
COURSE TITLE: OBJECT ORIENTED PROGRAMMING

Prepared by- ***Sumaiya Afroz Mila***

Class

- Class: The building block of C++ that leads to Object Oriented programming is a Class.
- It is a user defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- A class is like a blueprint for an object/instance.
- Specifying a class-
  - Class Declaration
  - Class Function Definition – (inside class declaration, outside class declaration)

Class Declaration

- Class declaration starts with the **keyword “class”**, followed by user-defined class name
- Body of the class enclosed within braces and terminated by a semicolon
- Class body contains the **declaration** of **variables** and **functions**. These variables and functions are collectively called **class members**.(Function declaration is same as the function prototype)
- **Variables** declared inside the class are known as **data members** and **functions** are known as **member functions**.

Class Declaration Syntax

Keyword

```
class class_name
{
Access specifier: [we will learn more later]
Data member;              //variables
Function declaration();         //functions to access variable
 };
```

User defined class name

Class name ends with a semicolon

Class definition does not occupy memory

```
main(){
class_name  obj1,obj2;
}
```

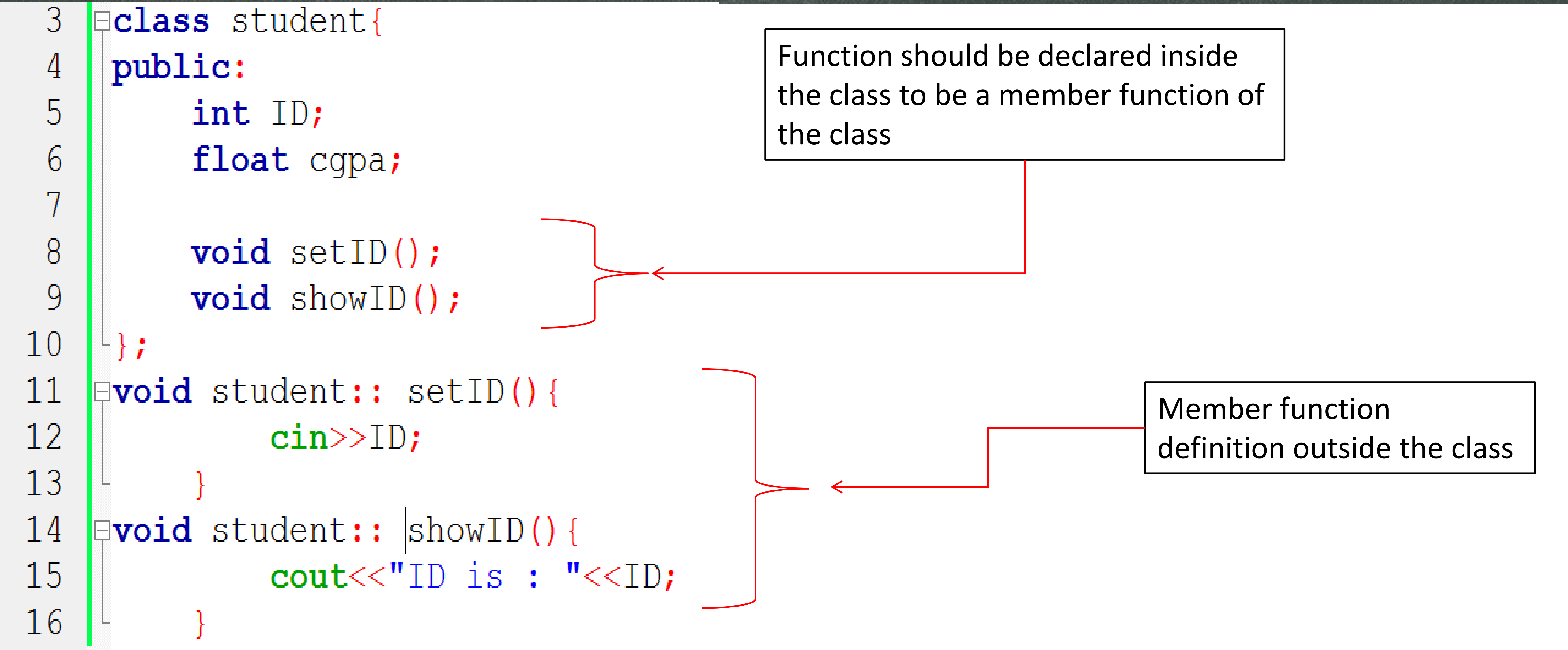
Declaration of objects of a class occupies memory

Function Definition – outside class declaration

Membership identity label.
This label tells the compiler, which class this function belongs to.

```
Return type class_name :: function_name(arguments)
{
        Function body
 };
```

Example: Function Definition – outside class declaration



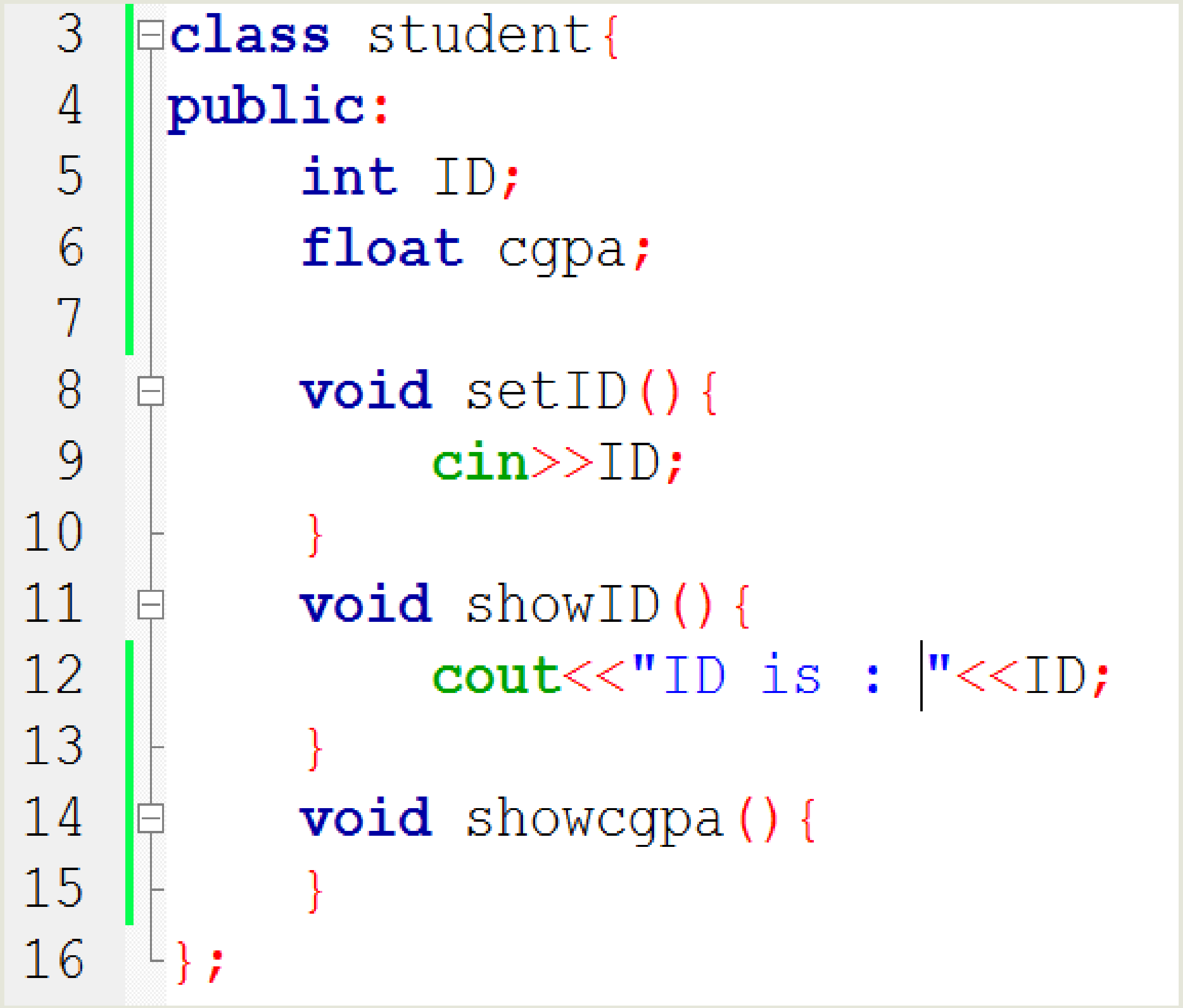
```
class student{
public:
    int ID;
    float cgpa;

    void setID();
    void showID();
};
void student:: setID(){
        cin>>ID;
    }
void student:: showID(){
        cout<<"ID is : "<<ID;
    }
```

Function should be declared inside the class to be a member function of the class

Member function definition outside the class

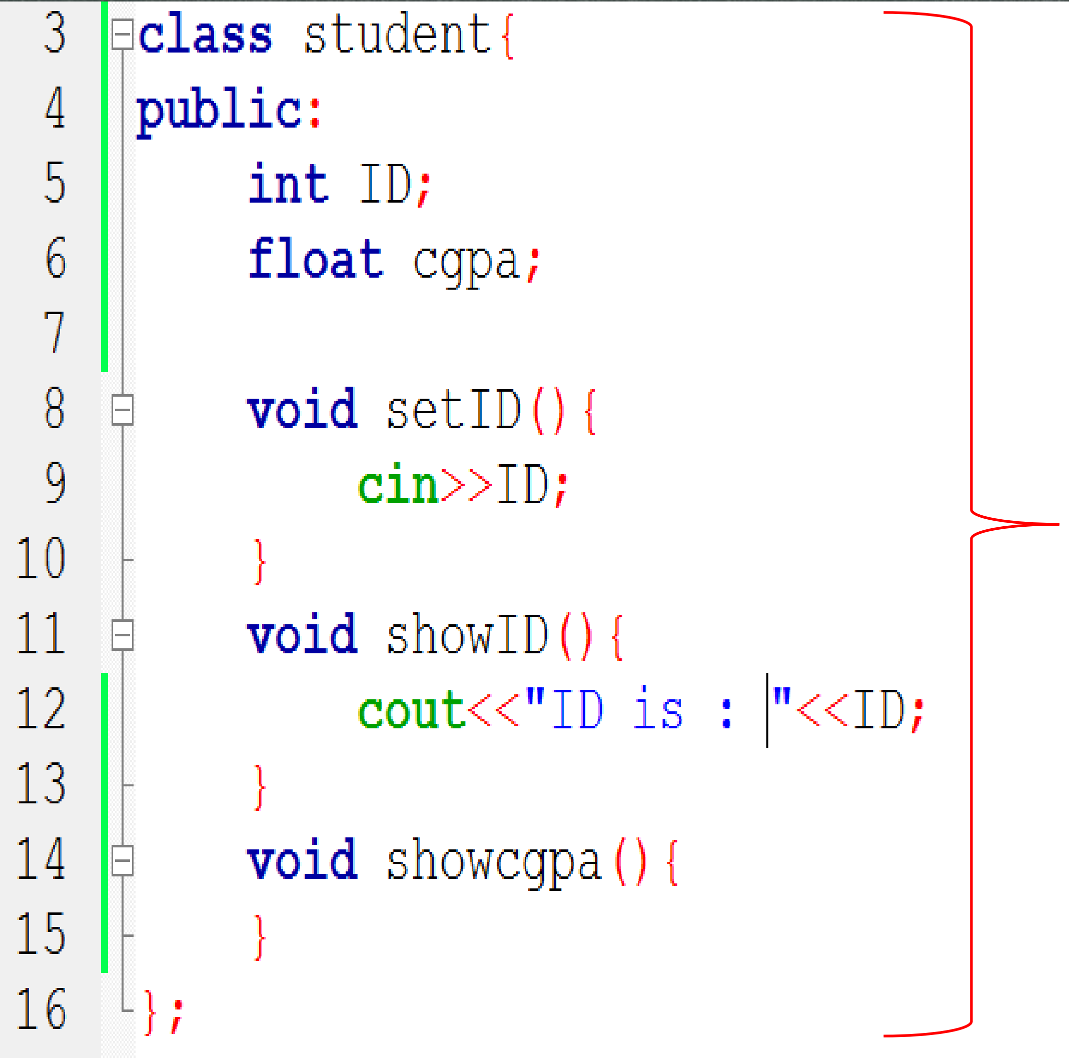
Function Definition – inside class declaration

Another method of defining member function is to replace the function declaration by the actual function definition inside the class.

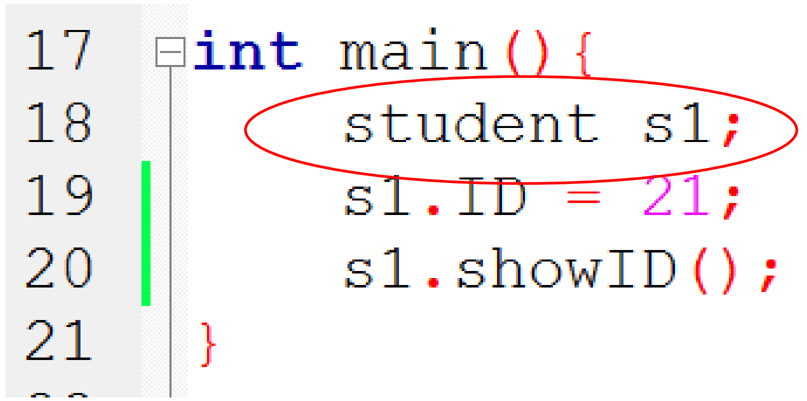


```
class student{
public:
    int ID;
    float cgpa;

    void setID(){
        cin>>ID;
    }
    void showID(){
        cout<<"ID is : "<<ID;
    }
    void showcgpa(){
    }
};
```

Let’s write our own ‘student class’



```
class student{
public:
    int ID;
    float cgpa;

    void setID(){
        cin>>ID;
    }
    void showID(){
        cout<<"ID is : "<<ID;
    }
    void showcgpa(){
    }
};
```



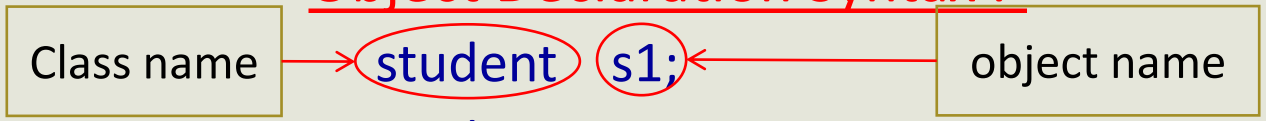
```
int main(){
    student s1;
    s1.ID = 21;
    s1.showID();
}
```

This is called ‘class declaration’. Class declaration is not a physical entity. This is just a logical representation of a student. So does not occupy memory

Creating objects of a class means creating a physical entity for that class. So creating object occupies memory

Accessing members of the class

- Objects : Instances / occurrence of the class
- <u>Object Declaration Syntax :-</u>



```
student s1;
student s2;
```

- Each instance has its own set of class variables
- Member functions are only maintained as one copy
- Accessing member variables of objects :-
  - Using object name and dot operator :

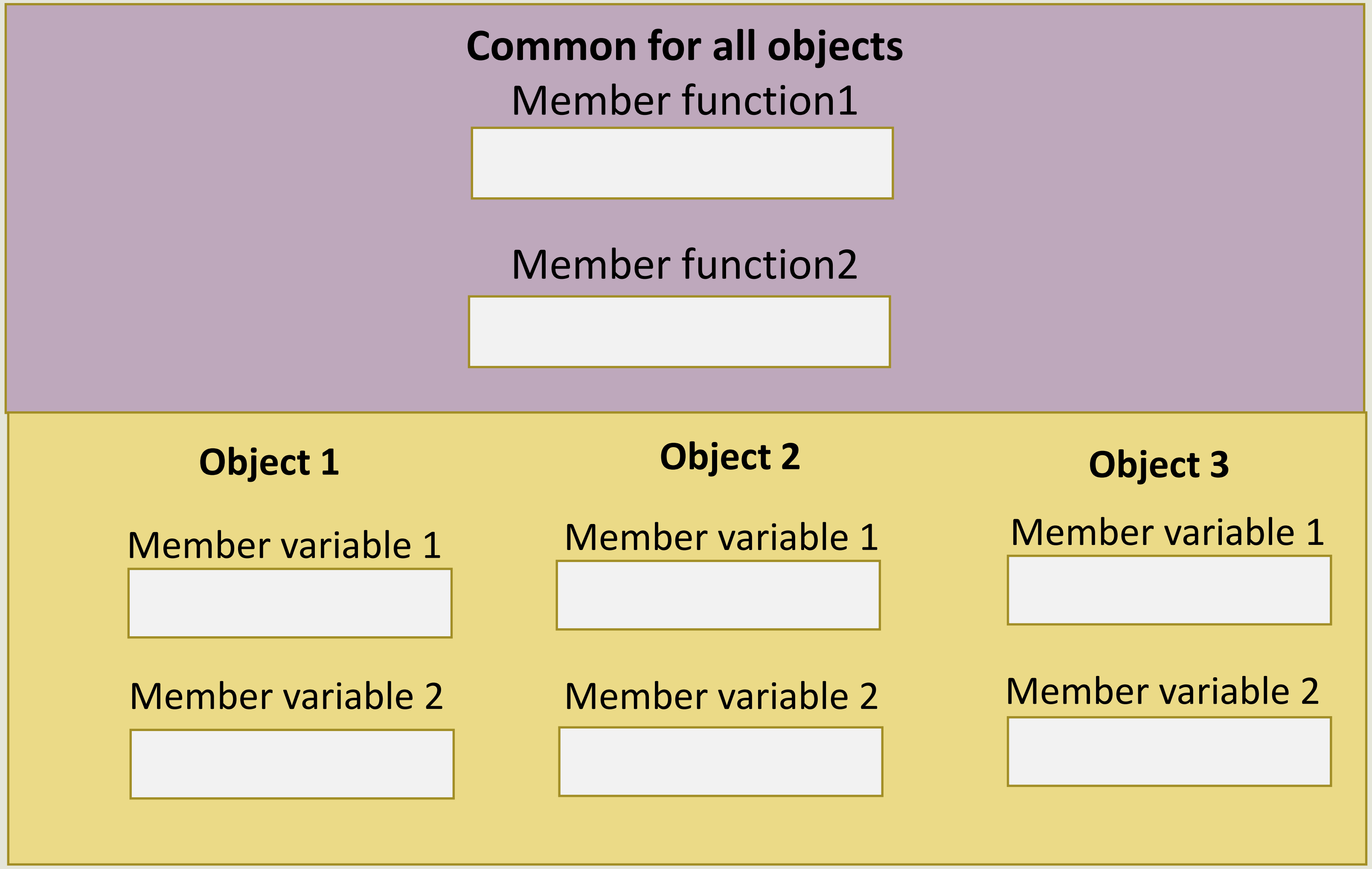
```
s1.ID= 21;
```

  - Using member functions.

```
s1.showID();
```

Memory allocation for class variables

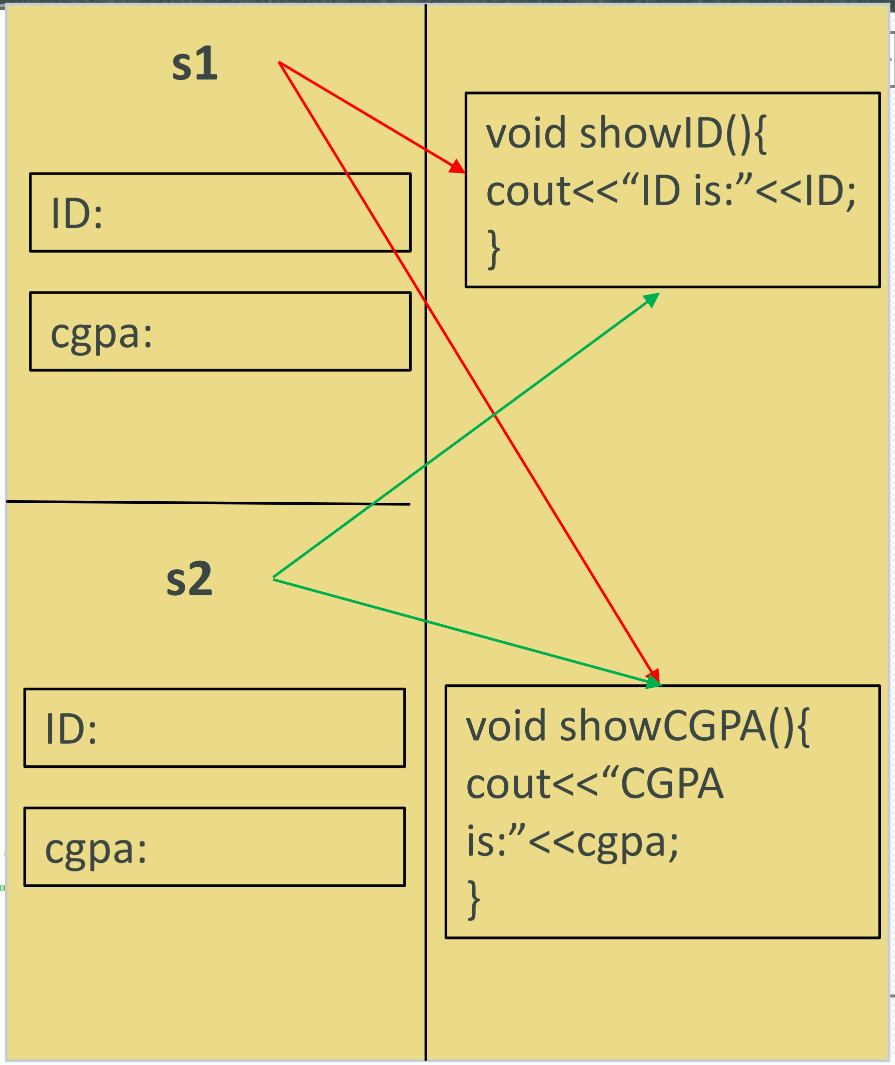
- The member functions are created and placed in the memory space only once when they are defined as a part of class declaration
- As all the objects of that class use the same member function, no separate space is allocated for func when objects are created.



- For each object, space for member variables is allocated separately.
- Separate memory locations for the objects are essential, as the member variables will hold different values for different objects.

Let’s write our own ‘student class’

```cpp
#include<iostream>
using namespace std;
class student{
public:
    int ID;
    float cgpa;

    void showID(){
        cout<<"ID is : "<<ID<<endl;
    }
    void showCGPA(){
        cout<<"CGPA is : "<<cgpa<<end
    }
};
```



```
void showID(){
cout<<“ID is:”<<ID;
}
```

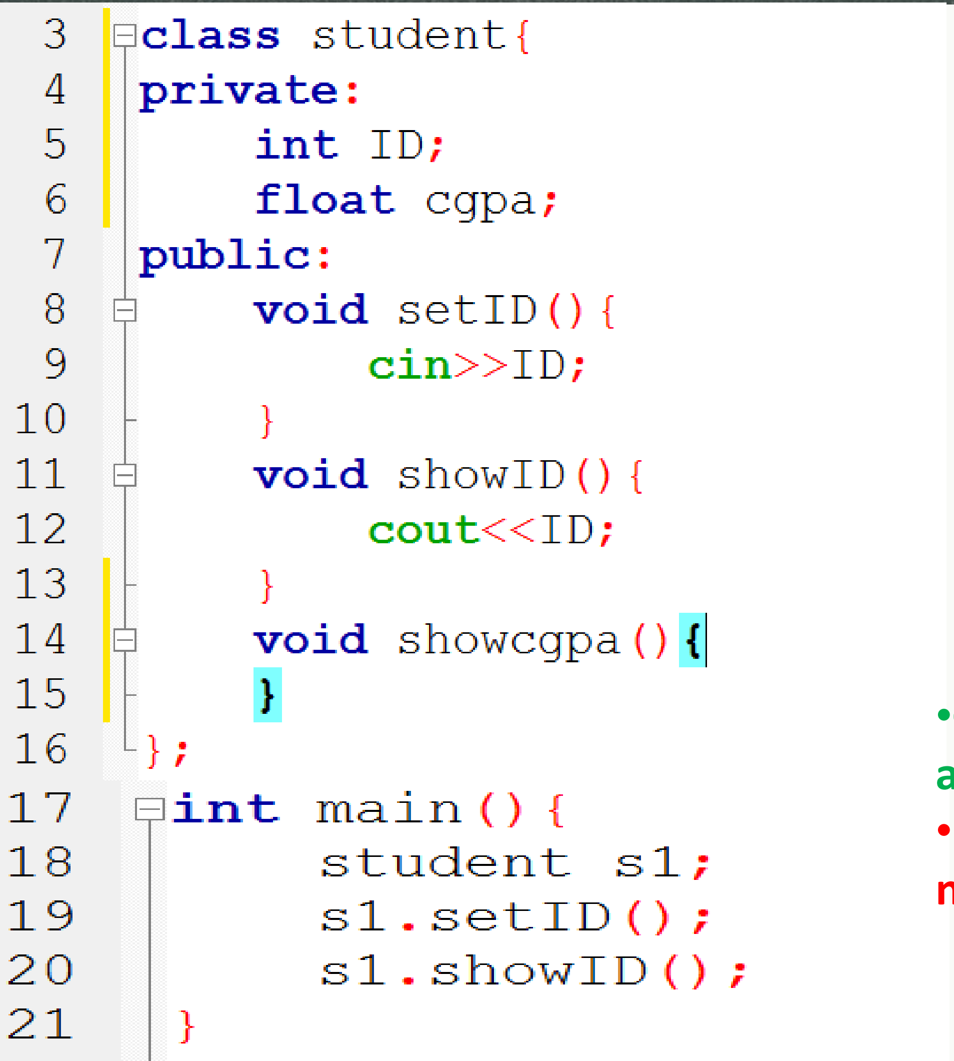
```
void showCGPA(){
cout<<“CGPA
is:”<<cgpa;
}
```

```cpp
int main(){
    student s1;
    student s2;

    s1.ID=21;
    s2.ID=25;
    s1.cgpa=2.5;
    s2.cgpa=3.5;

    s1.showID();
    s2.showID();
    s1.showCGPA();
    s2.showCGPA();
}
```

Overview: Access Specifiers

- There are three levels of access specifiers-
  - **Public** - can be accessed from everywhere of the code
  - **Private** - only accessible by the class members
  - **Protected** - similar to private. These members can not be accessed from everywhere of the code. But during inheritance derived class can access protected members. (You will learn about this more in inheritance part)

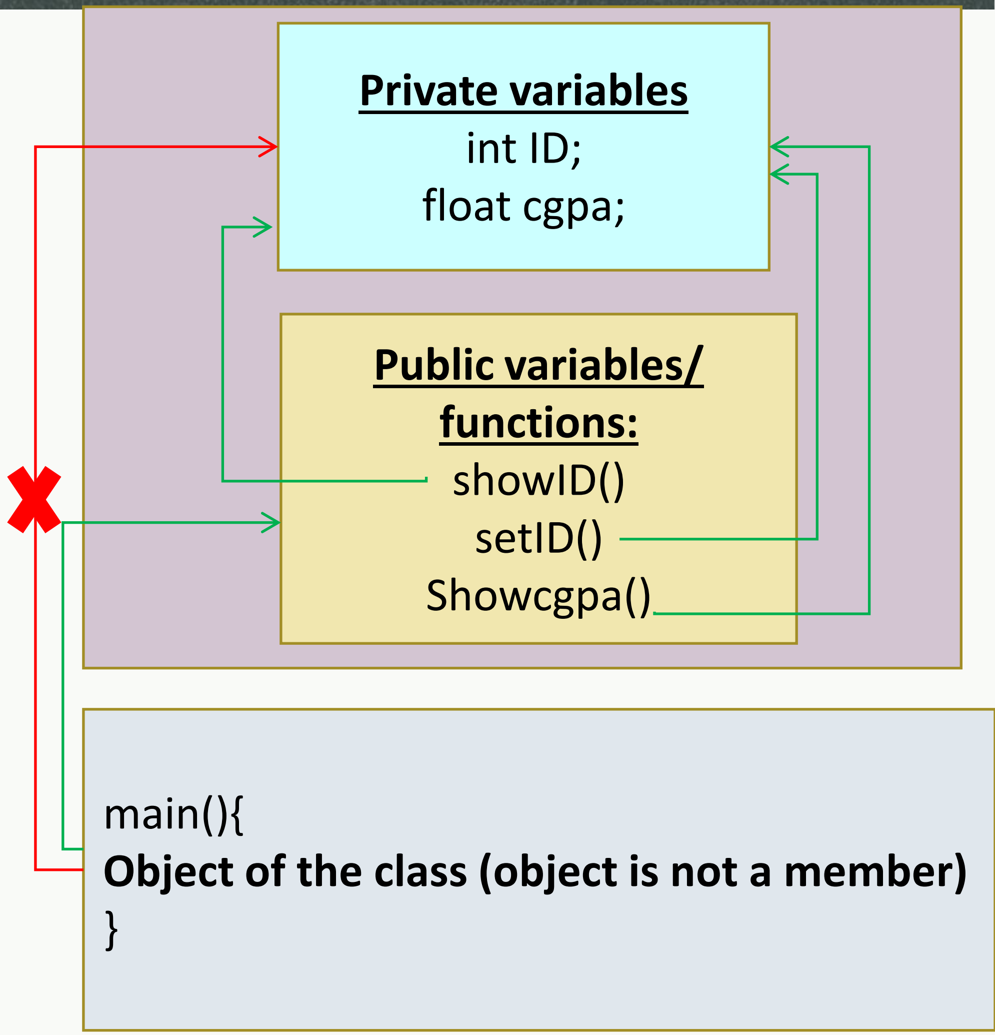
Visualizing Access Specifiers



```
class student{
private:
    int ID;
    float cgpa;
public:
    void setID(){
        cin>>ID;
    }
    void showID(){
        cout<<ID;
    }
    void showcgpa(){
    }
};
int main(){
    student s1;
    s1.setID();
    s1.showID();
}
```

- Private members are only accessible by the members of the class.
- Public members are accessible from anywhere

- **Green** arrows mean access **allowed**
- **Red** arrows mean access is **not allowed**



Private variables
int ID;
float cgpa;
Public variables/ functions:
showID()
setID()
Showcgpa()
main(){
Object of the class (object is not a member)
}

Detailed about Access Specifiers

- There are three levels of access specifiers-
  - Private
    - Usage - Default / ***private:***
    - Private vars accessible only by other member functions of the class & *friend functions*
    - *Private functions can't be called using objects !*
    - *Private member vars are not accessible from child class objects!*
  - Public
    - Usage – ***public :***
    - Accessible by other members and by any other part of the program that contain the class
    - Can set and get the value of public variables without any member function
  - Protected
    - Usage – ***protected :***
    - Protected vars can only be accessed using member functions of the class.
    - *Protected functions can't be called using objects !*
    - Inherited protected vars can be accessed using public functions in child / derived classes.

Thank you!