CSE-284: Object Oriented Programming

Class and Objects

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

Topics to be Covered

- What is OOP ?
- Class and Objects
- Member Data
- Member Functions
- Access Modifiers

OOP

- Object Oriented programming (OOP) is a programming paradigm that relies on the concept of **classes** and **objects**.
- It is used to structure a software program into simple, reusable pieces of code blueprints (usually called classes), which are used to create individual instances of objects.
- There are many object-oriented programming languages including JavaScript, C++, Java, and Python.

- A wonderful Resource on OOP Basics

Class

- A class is an abstract blueprint used to create more specific, concrete objects.
- Classes often represent broad categories, like Car or Dog that share attributes.
- These classes define what attributes an instance of this type will have, like color, but not the value of those attributes for a specific object.
- Classes can also contain functions, called methods/member functions available only to objects of that type.
- These functions are defined within/outside the class and perform some action helpful to that specific type of object.

Class

- For example, our Car class may have a method repaint that changes the color attribute of our car.
- This function is only helpful to objects of type Car, so we declare it within the Car class thus making it a method.

- Class templates are used as a blueprint to create individual **objects**.
- These represent specific examples of the abstract class, like myCar or goldenRetriever.
- Each object can have unique values to the properties defined in the class.

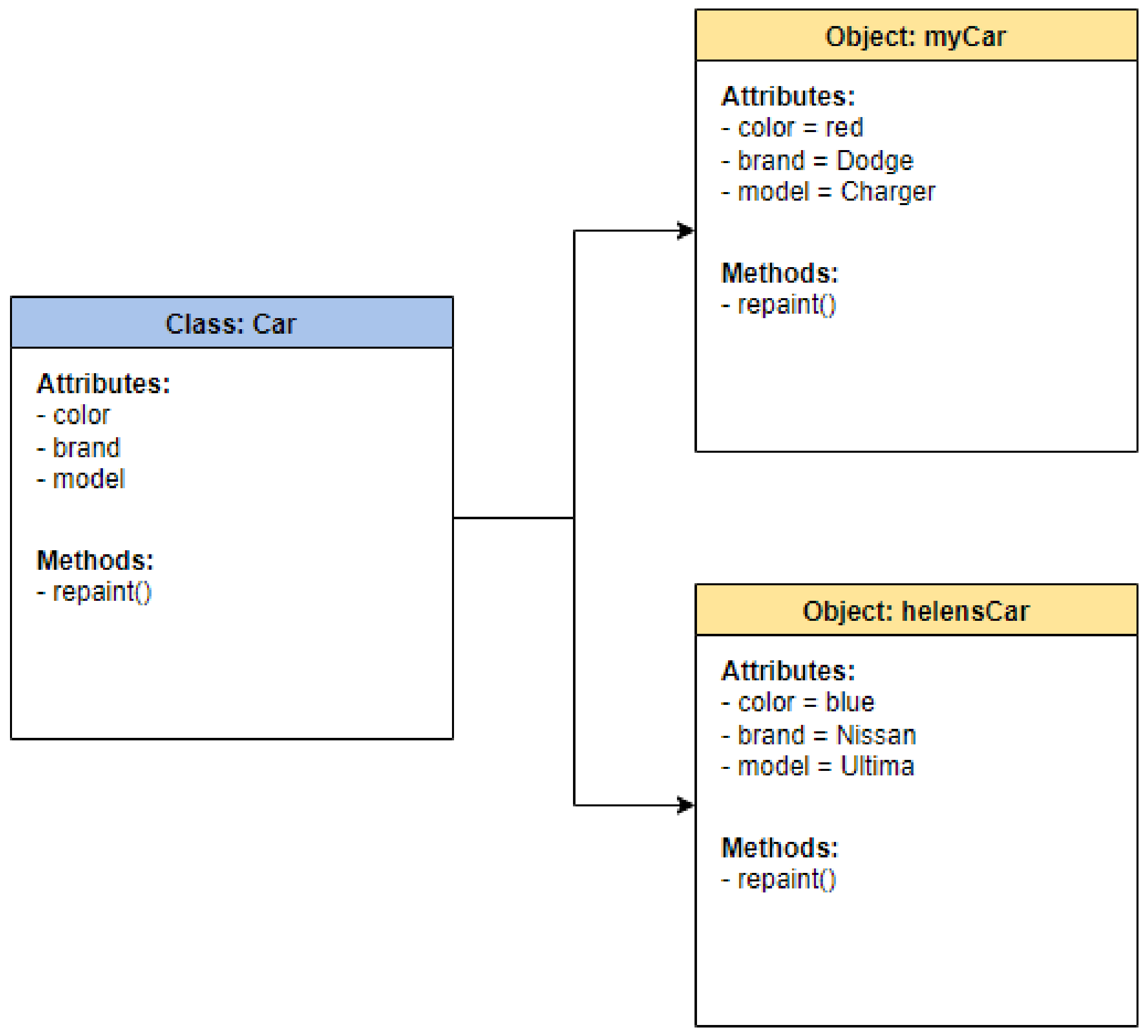
Class

- For example, say we created a class, Car, to contain all the properties a car must have, color, brand, and model. We then create an instance of a Car type object, myCar to represent my specific car.

- We could then set the value of the properties defined in the class to describe my car, without affecting other objects or the class template.

- We can then reuse this class to represent any number of cars.

Class


Class: Car
Attributes:
- color
- brand
- model
Methods:
- repaint()
Object: myCar
Attributes:
- color = red
- brand = Dodge
- model = Charger
Methods:
- repaint()
Object: helensCar
Attributes:
- color = blue
- brand = Nissan
- model = Ultima
Methods:
- repaint()

Class

- In a nutshell, classes are essentially user defined data types.
- Classes are where we create a blueprint for the structure of methods and attributes. Individual objects are instantiated, or created from this blueprint.

- Of course OOP includes objects! Objects are instances of classes created with specific data.

- Methods/Member Function represent behaviors. Methods perform actions; methods might return information about an object, or update an object’s data. The method’s code is defined in the class definition.

C++ Class Definitions

- 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.

```
class Box {
private:
      // data members
public:
      // member functions
};
```

C++ Class Definitions

- The keyword public / private / protected determines the access attributes of the members of the class that follow it.
- A private member cannot be directly accessed by the object and in order to access the data members, you have you have to define member functions which generally have public access specifier.
- You can also the data members and member functions as public, private or protected.

Define C++ Objects

- A class provides the blueprints for objects, so basically an object is created from a class.
- We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box:

```
Box Box1;    // Declare Box1 of type Box
Box Box2;    // Declare Box2 of type Box
```

Both of the objects Box1 and Box2 will have their own copy of data members.

Access Modifiers

- Keywords: private and public
- You may have noticed two keywords: private and public in the above example.
- The private keyword makes data and functions private. Private data and functions can be accessed only from inside the same class.
- The public keyword makes data and functions public. Public data and functions can be accessed out of the class.
- If you try to access private data from outside of the class, compiler throws error. This feature in OOP is known as data hiding.

Sample Viva-Voce Questions

1. What is a class?
2. What is an object?
3. What is an inline function?
4. What are/is the operator/operators used to access the class members?
5. Which access specifier/s can help to achieve data hiding in C++?
6. When a class member is defined outside the class, which operator can be used to associate the function definition to a particular class?
7. What are access modifiers?
8. What is the difference between structure and a class?
9. Write down two main difference between structure and a class.
10. What is the default access modifier in a class? p[illegible]
11. Where a class member function can be defined?
12. Can we declare a member function private?
13. What is the difference between public and private data members?
14. Do class declarations end with a semicolon? Do class method definitions?
15. Write a code to show different way of declaring a member function.

THANK YOU

CSE-284: Object Oriented Programming

Constructor and Destructor

Reference:
Simple-Snippets.com

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

Topics to be Covered

- What is Constructor ?
- Types of Constructor
- What is Destructor ?

Constructor

1. What is a 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) is created. It is special member function of the class.
- A class constructor is a special member function of a class that is executed whenever we create new objects of that class.
- Every time an instance of a class is created the constructor method is called.
- Constructors can be very useful for setting initial values for certain member variables.

Constructor

2. How constructors are different from a normal member function?

- 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).

Constructor

2. Types of Constructors?

- Default Constructors:
- Default constructor is the constructor which doesn't take any argument. It has no parameters.
- Parameterized Constructors:
- 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.
- Copy Constructor: A copy constructor is a member function which initializes an object another object of the same class.

Destructor

1. What is Destructor?

- Destructor is a member function which destructs or deletes an object.
- A destructor is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class.

2. When is destructor called?

- The function ends
- The program ends.
- A block containing local variables
- A delete operator is called.

Destructor

3. 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 anything(not even void).

4. Can there be more than one destructor in a class?

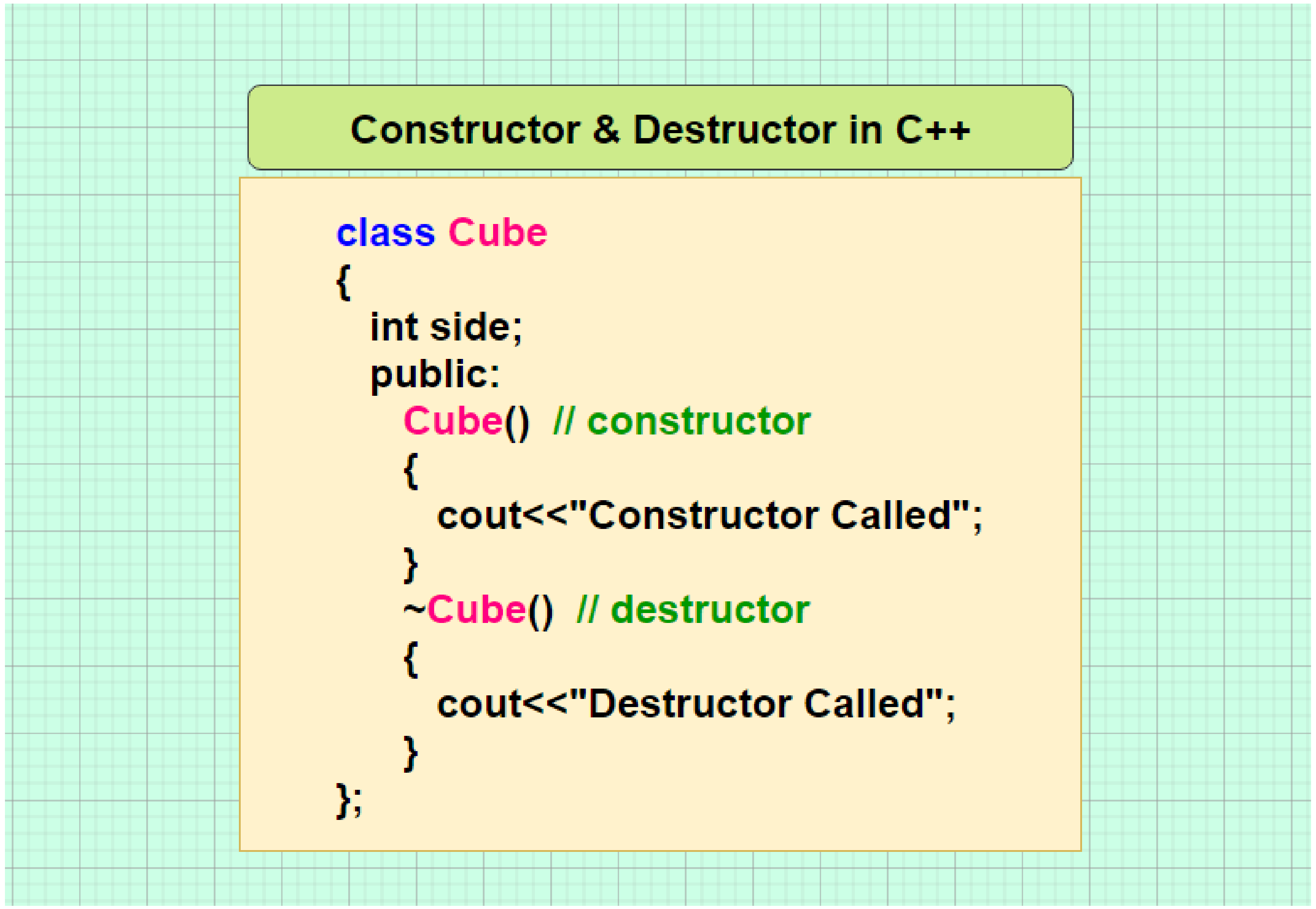
- No, there can only one destructor in a class

Destructor

3. When do we need to write a user-defined destructor?

- if we do not write our own destructor in class, compiler creates a default destructor for us.
- The default destructor works fine unless we have dynamically allocated memory or pointer in class.
- *When a class contains a pointer to memory allocated in class, we should write a destructor to release memory before the class instance is destroyed. This must be done to avoid memory leak.*

Example



Constructor & Destructor in C++

```
class Cube
{
  int side;
  public:
    Cube()  // constructor
    {
      cout<<"Constructor Called";
    }
    ~Cube()  // destructor
    {
      cout<<"Destructor Called";
    }
};
```

THANK YOU

Static Data Member and Function Overloading

Reference:
Simple-Snippets.com

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

Topics to be Covered

- What is Static Data Member ?
- What is Static Member Function ?
- What do you mean by Function Overloading ?

Static Data Member

1. What is a Static Data Member?

- Normally, when we instantiate objects of a class each object gets its own copy of all normal member variables.
- When we declare a member of a class as static it means no matter how many objects of the class are created, there is only one copy of the static member.
- That means one single copy of that data member is shared between all objects of that class.
- All static data is initialized to zero when the first object is created, if no other initialization is present.

Static Data Member

- Static data members can be initialized outside the class using the scope resolution operator (SRO) (::) to identify which class it belongs to.
- Since Static data members are class level variables, we do not require the object to access these variables
- they can be accessed simply by using the class name and scope resolution(::) operator with the variable name to access its value.
- A very common use of static data members is to keep a count of total objects of a particular class (program counter kind of application)

C++ Static Data Member

Declaration:

static data_type member_name;

Definition:

data_type class_name :: member_name =value;

If you are calling a static data member within a member function, member function should be declared as static (i.e. a static member function can access the static data members)

Static Member Function

1. What is a Static Member Function?

- By declaring a function member as static, you make it independent of any particular object of the class.
- A static member function can be called even if no objects of the class exist
- The static functions are accessed using only the class name and the scope resolution operator (::)
- *A static member function can only access static data member, other static member functions and any other functions from outside the class*.

Function Overloading

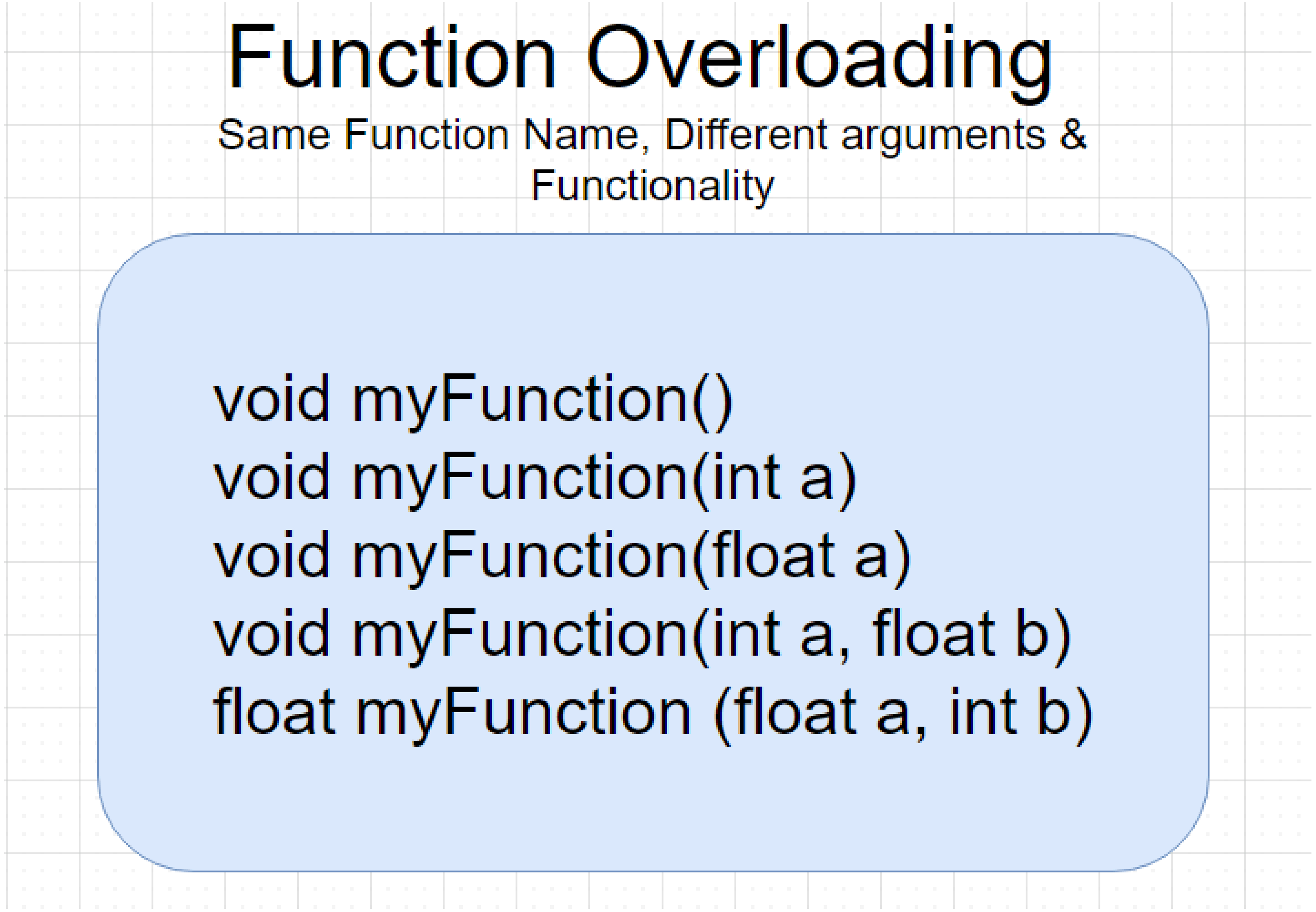
1. What is a Function Overloading in C++?

- Function Overloading in C++ is the mechanism by which a programmer can specify multiple definitions of the same function(same name) by changing:
  - Number of arguments passed to the function
  - Data type of arguments passed to the function

Function Overloading

- Essentially Function overloading means same function name but different number or type of arguments with different functionality.
- *Function overloading is also a type of Static or Compile time Polymorphism*.
- Overloading is a form of polymorphism. It allows the programmer to write functions to do conceptually the same thing on different types of data without changing the name.
- This allows consistency in notation, which is good both for reading and for writing code.
- This also allows the same method name to be reused across multiple implementations.

Function Overloading



Function Overloading

Same Function Name, Different arguments & Functionality

```
void myFunction()
void myFunction(int a)
void myFunction(float a)
void myFunction(int a, float b)
float myFunction (float a, int b)
```

Image Reference: Simple-Snippets.com

Thank You

CSE-284: Object Oriented Programming

Inheritance in C++

Reference:
TrytoProgram.com

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

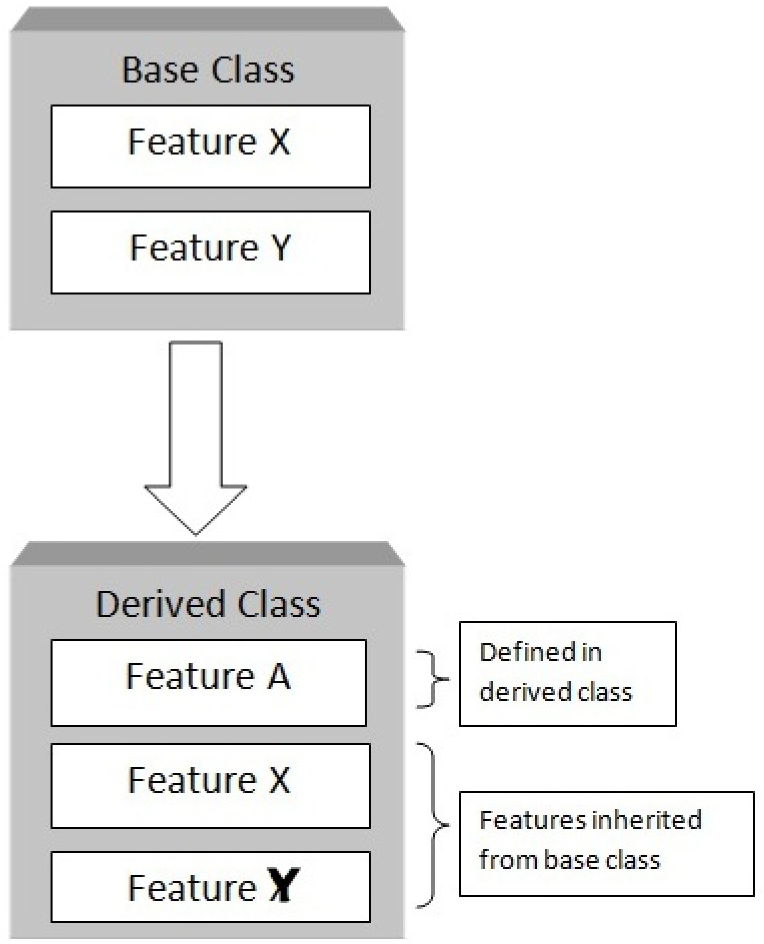
Topics to be Covered

- What is Inheritance?
- Types of Inheritance
- What are the Access specifiers?

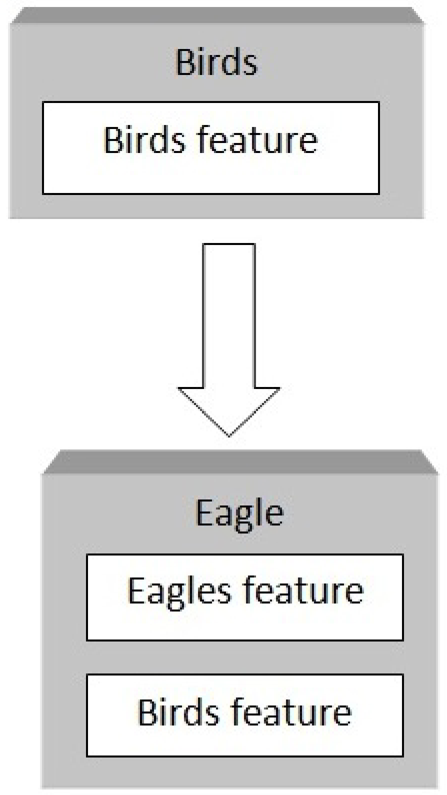
Inheritance

- C++ Inheritance is one of the powerful features of C++.
- Inheritance is the technique of building new classes called derived classes from the existing class called a base class by inheriting the features of the base class.
- Apart from inheriting the properties of the base class, an extra new feature can be added to the derived class.
- A base class is called parental class and the derived class is called a descendant class as it inherits the feature of the parental class.

Inheritance



Example of inheritance



Types of Inheritance

- Single inheritance
- Multilevel inheritance
- Hierarchical inheritance
- Multiple inheritance
- Hybrid inheritance

C++ Inheritance Syntax or Format

```
class base_class_name
{
   ...................
};

class derived_class_name : acess_specifier base_class_name
{
   ...........
} ;
```

Access Specifiers

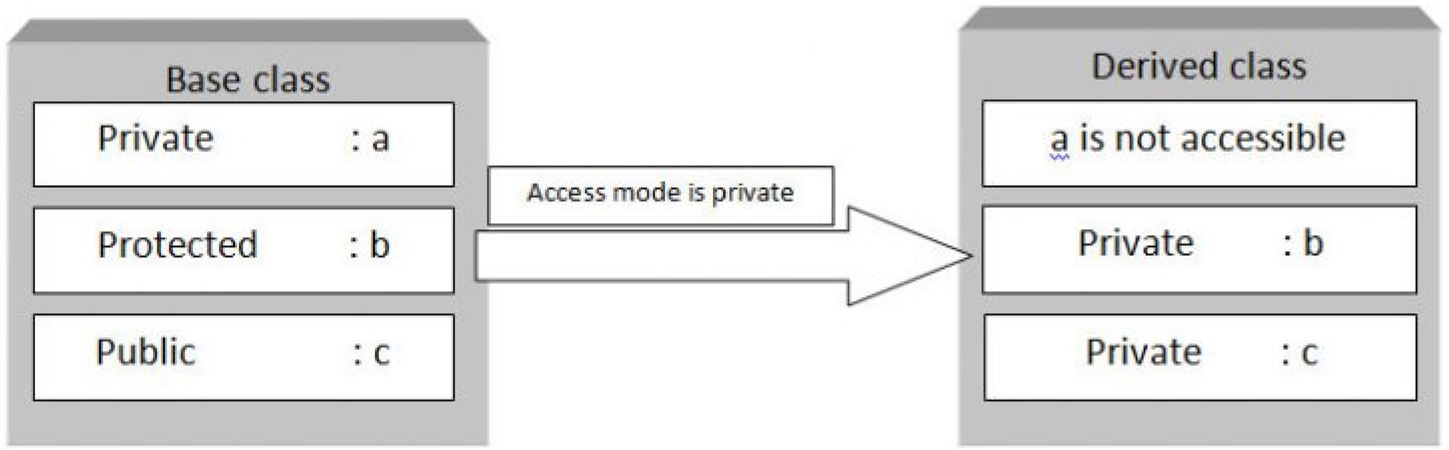
- C++ access specifiers are used for determining or setting the boundary for the availability of class members (data members and member functions) beyond that class.
- For example, the class members are grouped into sections, private, protected and public. These keywords are called access specifiers which define the accessibility or visibility level of class members.
- By default the class members are private. So if the visibility labels are missing then by default all the class members are private.

- *In inheritance, it is important to know when a member function in the base class can be used by the objects of the derived class. This is called accessibility and the access specifiers are used to determine this*

Private Access Specifiers

- If private access specifier is used while creating a class, then the public and protected data members of the base class become the private member of the derived class and private member of base class remains private.

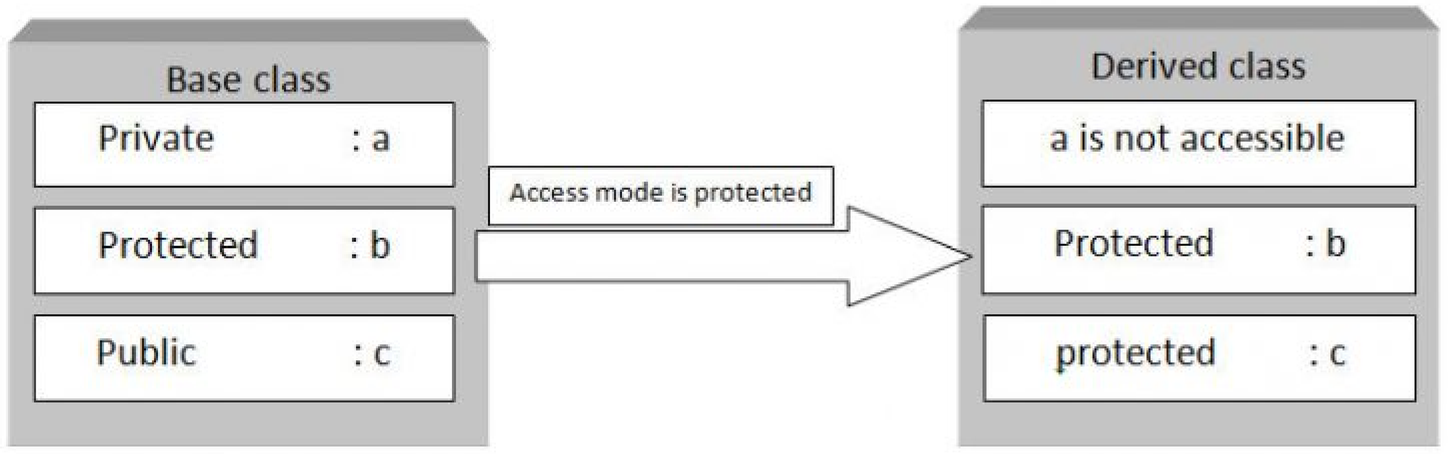
- *In this case, the members of the base class can be used only within the derived class and cannot be accessed through the object of derived class whereas they can be accessed by creating a function in the derived class.*



Protected Access Specifiers

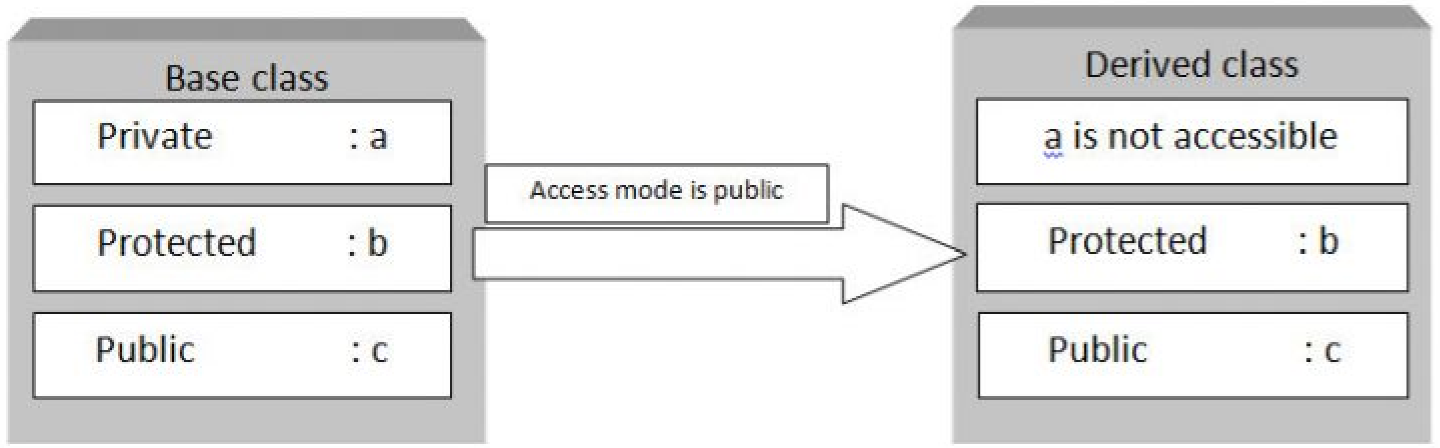
- If protected access specifier is used while deriving class then the public and protected data members of the base class becomes the protected member of the derived class and private member of the base class are inaccessible.

- In this case, the members of the base class can be used only within the derived class as protected members except for the private members.



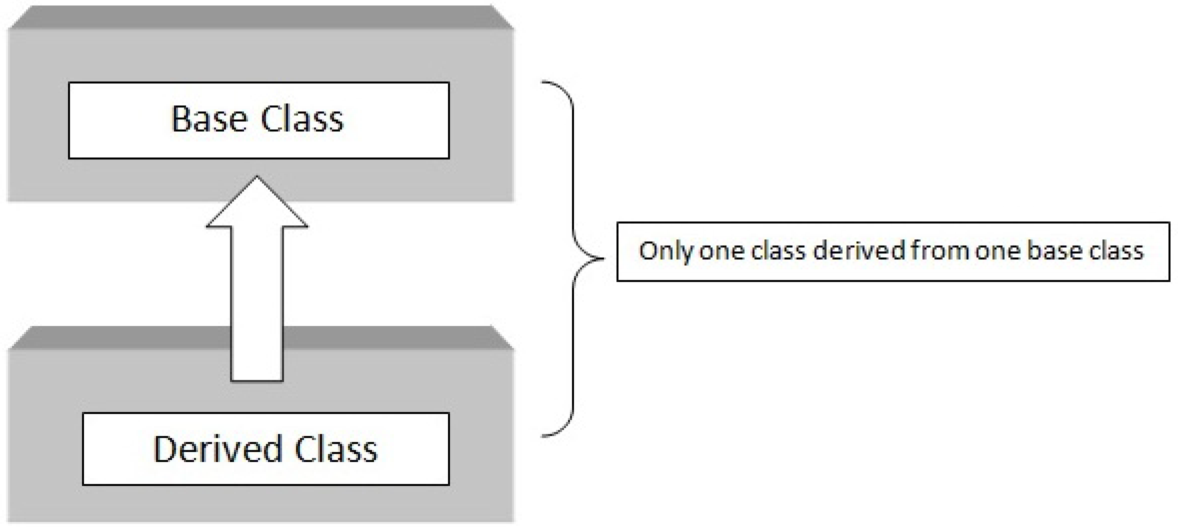
Public Access Specifiers

- If public access specifier is used while deriving class then the public data members of the base class becomes the public member of the derived class and protected members becomes the protected in the derived class but the private members of the base class are inaccessible.



Single level Inheritance

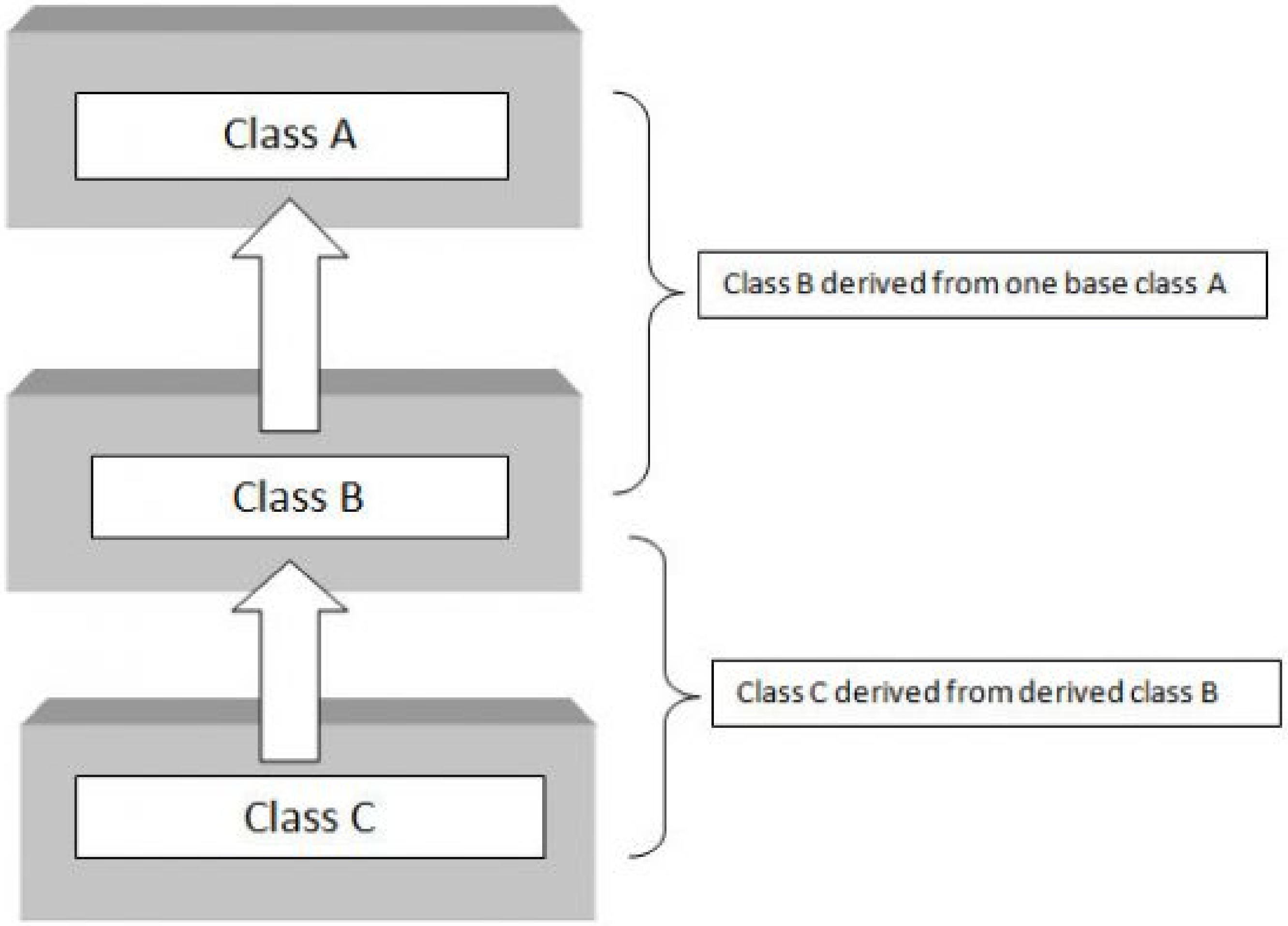
- Single inheritance only one class can be derived from the base class.
- Based on the visibility mode used or access specifier used while deriving, the properties of the base class are derived.
- Access specifier can be private, protected or public.



Multi level Inheritance

- If a class is derived from another derived class then it is called multilevel inheritance.
- So in C++ multilevel inheritance, *a class has more than one parent class*.
- For example, if we take animals as a base class then mammals are the derived class which has features of animals and then humans are the also derived class that is derived from sub-class mammals which inherit all the features of mammals.

Multi level Inheritance


Class A
Class B
Class C
Class B derived from one base class A
Class C derived from derived class B

Multi level Inheritance

C++ Programming Multilevel Inheritance Syntax

```
class A // base class
{
     ...........
};
class B : access_specifier A // derived class
{
     ...........
 } ;
 class C : access_specifier B // derived from derived class
B
 {
     ...........
 } ;
```

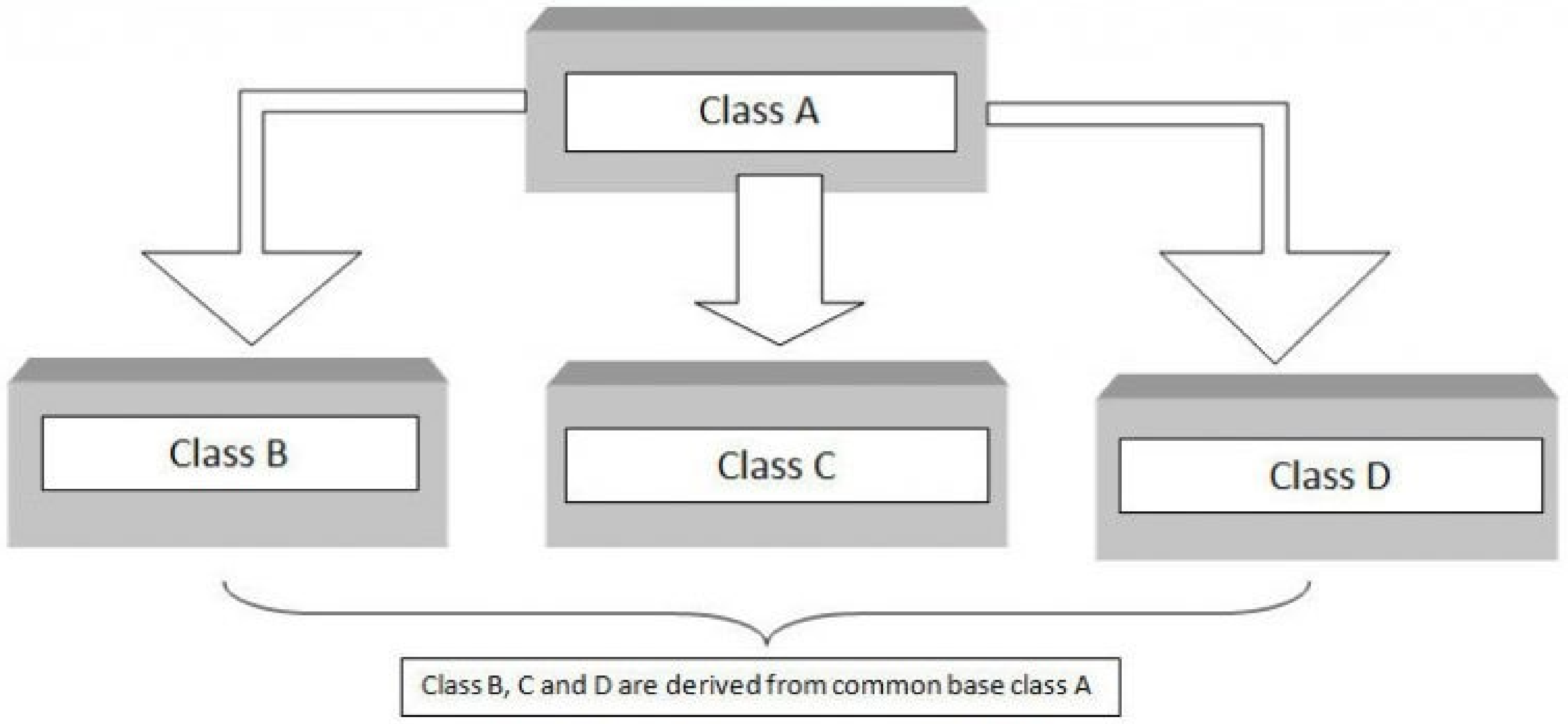
Hierarchical Inheritance

- When several classes are derived from common base class it is called hierarchical inheritance.

- In C++ hierarchical inheritance, the feature of the base class is inherited onto more than one sub-class.

- For example, a car is a common class from which Audi, Ferrari, Maruti etc. can be derived.

Hierarchical Inheritance

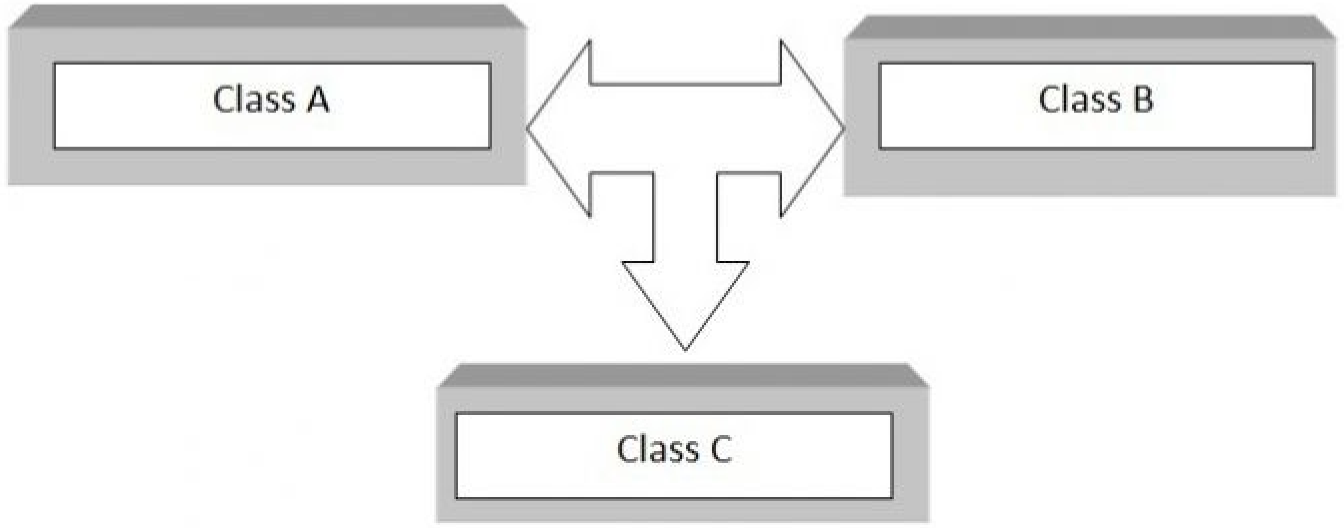


Multiple Inheritance

- If a class is derived from two or more base classes then it is called multiple inheritance.
- In C++ multiple inheritance a derived class has more than one base class.
- *How does multiple inheritance differ from multilevel inheritance?*
- Though but multiple and multilevel sounds like same but they differ hugely in meaning. In multilevel inheritance, we have multiple parent classes whereas in in multiple inheritance we have multiple base classes.
- To put it in simple words, in multilevel inheritance, a class is derived from a class which is also derived from another base class. And these levels of inheritance can be extended. On the contrary, in multiple inheritance, a class is derived from two different base classes.

Multiple Inheritance

- For example
- Multilevel inheritance : Inheritance of characters by a child from father and father inheriting characters from his father (grandfather)
- Multiple inheritance : Inheritance of characters by a child from mother and father



THANK YOU

CSE-284: Object Oriented Programming

Friend Function in C++

Reference:

Section.io

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

Topics to be Covered

- What is Friend function ?
- Difference between member function and friend function
- What is Friend Class ?
- Why do we need friend function ?

Friend Function

- Data hiding is a fundamental concept of object-oriented programming. It restricts the access of private members from outside of the class.
- Similarly, protected members can only be accessed by derived classes and are inaccessible from outside.
- However, there is a feature in C++ called friend functions that break this rule and allow us to access member functions from outside the class.
- ***A friend function can access the private and protected data of a class.***
- They are defined globally outside the class scope.
- *Friend functions are not member functions of the class.*

Why Friend Function ?

- A friend function is a function that is declared outside a class but is capable of accessing the private and protected members of the class.

- *There could be situations in programming wherein we want two classes to share their members.*
- *These members may be data members, class functions.*
- *In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data members of the class.*

Special features of friend functions?

- A friend function does not fall within the scope of the class for which it was declared as a friend. Hence, functionality is not limited to one class.
- A friend function can be declared in the private or public part of a class without changing its meaning.
- Friend functions are not called using objects of the class because they are not within the class's scope.
- *Friend functions can use objects of the class as arguments.*

```
class className{
   // Other Declarations
   friend returnType functionName(arg list);
};
```

Friend Class

- A friend class can have access to the data members and functions of another class in which it is declared as a friend.
- They are used in situations where we want a certain class to have access to another class's private and protected members.
- Friend functions are used to work as a link between the classes.

```
class S; //forward declaration

class P{
  // Other Declarations
  friend class S;
};

class S{
  // Declarations → P এরমত
};
```

Friend Class

- In the illustration above, class S is a friend of class P.
- As a result class S can access the private data members of class P.
- However, this does not mean that class P can access private data members of class S.

- *Friend functions should be used for restricted purposes only. Having excessive friend functions and classes can reduce the object-oriented programming feature of encapsulation in a program.*
- *Friendship is not reciprocal. If class X is Y's friend, then Y does not automatically become X's friend.*
- *Friendship cannot be inherited.*

Sample Questions

1) What is a friend function?
2) What is a friend class?
3) Write a code to show how to define a friend function/class.
4) When one should use friend function and friend class.
5) Does structure support friend function/class.
6) If Dog is a friend of Boy, is Boy a friend of Dog?
7) If Dog is a friend of Boy, and Terrier derives from Dog, is Terrier a friend of Boy?

Sample Questions

1) if Dog is a friend of Boy and Boy is a friend of House, is Dog a friend of House?
2) Where must the declaration of a friend function appear?
3) What is role of static keyword on class member variable?
4) Explain the static member function.
5) Will you be able to access a non static data member by a static member function?
6) What are differences between a normal member function and a static member function.
7) Whether non static method can use static members?
8) Can static member variables be private ?

THANK YOU

Operator Overloading in C++

Reference:

TrytoProgram.com

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

CUET

Topics to be Covered

- What is Operator overloading ?
- What is unary and binary operator overloading
- Why do we need operator overloading
- Operator overloading using friend function

Operator Overloading

- Operator overloading is an important concept in C++.
- It is a type of polymorphism in which an operator is overloaded to give user defined meaning to it.
- Overloaded operator is used to perform operation on user-defined data type.
- For example '+' operator can be overloaded to perform addition on various data types, like for Integer, String(concatenation) or even a user defined class object.
- Overloaded operators are functions with special names the keyword *operator* followed by the symbol for the operator being defined.
- Like any other function, an overloaded operator has a return type and a parameter list.

Operator Overloading

- *Operator functions are same as normal functions.*
- *The only differences are, name of an operator function is always operator keyword followed by symbol of operator and operator functions are called when the corresponding operator is used.*

Syntax of operator overloading

```
class className {
    public:
       returnType operator symbol (arguments) {
       }
    };
```

Implementing Operator Overloading

- *Operator overloading can be done by implementing a function which can be :*
  - *Member Function*
  - *Friend Function*
- *Operator overloading function can be a member function if the Left operand is an Object of that class, but if the Left operand is different, then Operator overloading function must be a non-member function.*
- *Operator overloading function can be made friend function if it needs access to the private and protected members of class.*

Implementing Operator Overloading

- *Operator that cannot be overloaded are as follows:*

1) *Scope operator (::)*
2) *Sizeof*
3) *member selector(.)*
4) *member pointer selector(*)*
5) *ternary operator(?:)*

Thank You

CSE-284: Object Oriented Programming

Function Overriding in C++

Reference:

TrytoProgram.com

Eftekhar Hossain
Lecturer
Dept. of ETE, CUET

Topics to be Covered

- What is Function Overriding?
- When do we need to override a function?

Function Overriding

- The process of re-defining the superclass non-private method in the subclass with the same signature is called ***Function Overriding*** in C++.

- The same signature means the name and the parameters should be the same.

- The implementation of the subclass overrides (i.e. replaces) the implementation of the superclass method.

- The overriding method is always going to be executed from the current class object.

Function Overriding

```
class Parent {
    public:
       void display (){

        cout<<“Display of Parent”;
       }
    };
```

```
class Child: Public Parent {
    public:
       void display (){

       }
    };
```

```
Parent P;
P.display(); “Display of Parent”

Child C;
C.display(); “Display of Parent”
```

Function Overriding

```
class Parent {
    public:
       void display (){

         cout<<“Display of Parent”;
       }
    };
```

```
class Child: Public Parent {
    public:
       void display (){
     cout<<“Display of Child”;

       }
    };
```

```
Parent P;
P.display(); “Display of Parent”

Child C;
C.display(); “Display of Child”
```

Implementing Operator Overloading

- Suppose we have two car design companies that are X and Y.
- So, the features available in X will also be available in Y.
- But Y company has redefined those features.
- So, writing functions once again is called function overriding.

- Suppose in 'X's car there is a key that you have to use to lock the car or open the car but in 'Y's car has the same feature that is you can open the door of the car with a keyless entry. So, you don't have to unlock it using the key. So, the same feature is available that is opening and closing of doors of the car but it is redefined in the 'Y' car.

When do we need to override a function in C++?

- If the superclass function logic is not fulfilling the sub-class business requirements, then the subclass needs to override that function with the required business logic.

- Usually, in most real-time applications, the superclass functions are implemented with generic logic which is common for all the next-level sub-classes.

Thank You