

C++ Object Oriented Programming

CS201- Data Structures

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.

Object

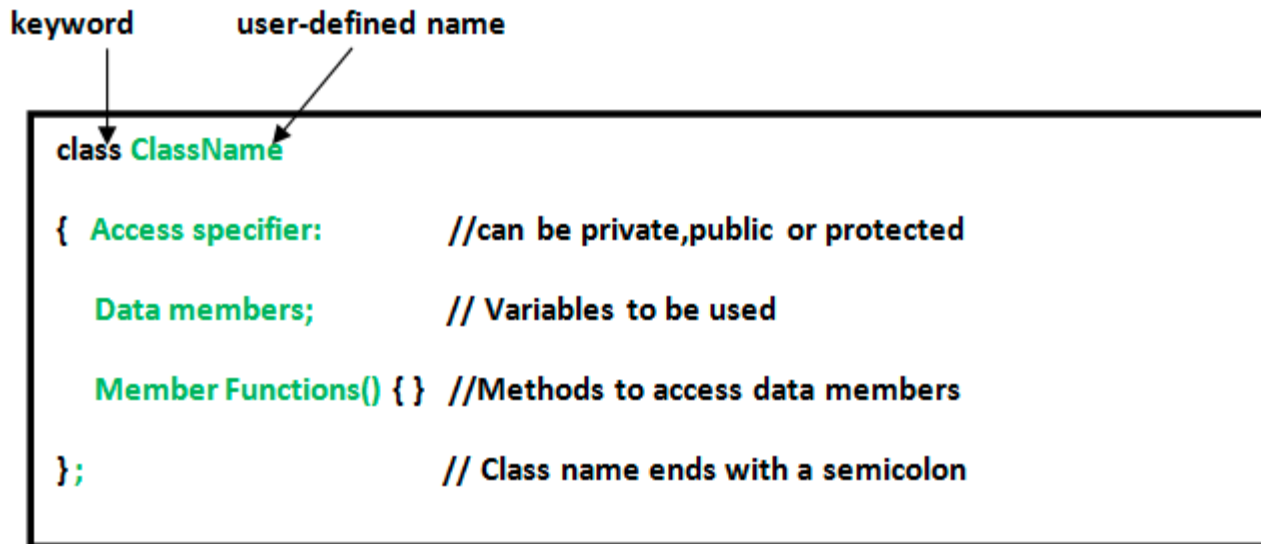
- An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

```
class person
{
    char name[20];
    int id;
public:
    void getdetails(){}
};

int main()
{
    person p1; //p1 is a object
}
```

Defining Class

- A class is defined in C++ using keyword `class` followed by the name of class. The body of class is defined inside the curly brackets and terminated by a semicolon at the end.



The diagram illustrates the syntax for defining a class in C++. It shows the keyword `class` and the user-defined name `ClassName` within a code block. The code block is enclosed in curly braces and terminated with a semicolon. The code block contains the following text:

```
class ClassName  
{ Access specifier:      //can be private,public or protected  
  Data members;          // Variables to be used  
  Member Functions() { } //Methods to access data members  
};                          // Class name ends with a semicolon
```

Annotations with arrows point to the `class` keyword and the `ClassName` identifier.

Declaring Objects

- When a class is defined, only the specification for the object is defined; no memory or storage is allocated. To use the data and access functions defined in the class, you need to create objects.
- The data members and member functions of class can be accessed using the dot('.') operator with the object.

```
ClassName ObjectName;
```

Data Members

```
// C++ program to demonstrate
// accessing of data members

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    // Access specifier
    public:

    // Data Members
    string geekname;

    // Member Functions()
    void printname()
    {
        cout << "Geekname is: " << geekname;
    }
};

int main() {

    // Declare an object of class geeks
    Geeks obj1;

    // accessing data member
    obj1.geekname = "Abhi";

    // accessing member function
    obj1.printname();
    return 0;
}
```

Output:

```
Geekname is: Abhi
```

Member Functions

- There are 2 ways to define a member function: inside or outside class definition.
- For outside the class definition use the scope resolution :: operator along with class name and function name.

```
// C++ program to demonstrate function
// declaration outside class

#include <bits/stdc++.h>
using namespace std;
class Geeks
{
    public:
    string geekname;
    int id;

    // printname is not defined inside class definition
    void printname();

    // printid is defined inside class definition
    void printid()
    {
        cout << "Geek id is: " << id;
    }
};

// Definition of printname using scope resolution operator ::
void Geeks::printname()
{
    cout << "Geekname is: " << geekname;
}

int main() {

    Geeks obj1;
    obj1.geekname = "xyz";
    obj1.id=15;

    // call printname()
    obj1.printname();
    cout << endl;

    // call printid()
    obj1.printid();
    return 0;
}
```

Output:

```
Geekname is: xyz
Geek id is: 15
```

Access Modifiers

- Access modifiers are used to implement important feature of Object Oriented Programming known as Data Hiding.
- There are 3 types of access modifiers available in C++:
 - 1. Public
 - 2. Private
 - 3. Protected

Public - Example

```
// C++ program to demonstrate public
// access modifier

#include<iostream>
using namespace std;

// class definition
class Circle
{
    public:
        double radius;

        double compute_area()
        {
            return 3.14*radius*radius;
        }
};

// main function
int main()
{
    Circle obj;

    // accessing public datamember outside class
    obj.radius = 5.5;

    cout << "Radius is:" << obj.radius << "\n";
    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

Output:

```
Radius is:5.5
Area is:94.985
```

Private - Example

```
// C++ program to demonstrate private
// access modifier

#include<iostream>
using namespace std;

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double compute_area()
        {
            // member function can access private
            // data member radius
            return 3.14*radius*radius;
        }
};

// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.radius = 1.5;

    cout << "Area is:" << obj.compute_area();
    return 0;
}
```

Output:

```
In function 'int main()':
11:16: error: 'double Circle::radius' is private
    double radius;
        ^
31:9: error: within this context
    obj.radius = 1.5;
    ^
```

Private – Example2

```
// C++ program to demonstrate private
// access modifier

#include<iostream>
using namespace std;

class Circle
{
    // private data member
    private:
        double radius;

    // public member function
    public:
        double compute_area(double r)
        {
            // member function can access private
            // data member radius
            radius = r;

            double area = 3.14*radius*radius;

            cout << "Radius is:" << radius << endl;
            cout << "Area is: " << area;
        }
};

// main function
int main()
{
    // creating object of the class
    Circle obj;

    // trying to access private data member
    // directly outside the class
    obj.compute_area(1.5);

    return 0;
}
```

Output:

```
Radius is:1.5
Area is: 7.065
```

Protected - Example

```
// C++ program to demonstrate
// protected access modifier
#include <bits/stdc++.h>
using namespace std;

// base class
class Parent
{
    // protected data members
    protected:
    int id_protected;
};

// sub class or derived class
class Child : public Parent
{
    public:
    void setId(int id)
    {
        // Child class is able to access the inherited
        // protected data members of base class

        id_protected = id;
    }

    void displayId()
    {
        cout << "id_protected is:" << id_protected << endl;
    }
};

// main function
int main() {
    Child obj1;

    // member function of derived class can
    // access the protected data members of base class

    obj1.setId(81);
    obj1.displayId();
    return 0;
}
```

Output:

id_protected is:81

Abstraction

- Data abstraction refers to, providing only essential information to the outside world and hiding their background details, i.e., to represent the needed information in program without presenting the details.
- For example, a database system hides certain details of how data is stored and created and maintained.

Abstraction - Example

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

class implementAbstraction
{
    private:
        int a, b;

    public:

        // method to set values of
        // private members
        void set(int x, int y)
        {
            a = x;
            b = y;
        }

        void display()
        {
            cout<<"a = " <<a << endl;
            cout<<"b = " << b << endl;
        }
};

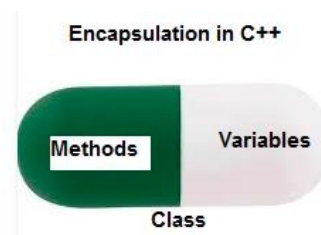
int main()
{
    implementAbstraction obj;
    obj.set(10, 20);
    obj.display();
    return 0;
}
```

Output:

```
a = 10
b = 20
```

Encapsulation

- Wrapping up(combining) of data and functions into a single unit is known as encapsulation.
- The data is not accessible to the outside world and only those functions which are wrapping in the class can access it.
- This insulation of the data from direct access by the program is called data hiding or information hiding.



Encapsulation - Example

```
// c++ program to explain
// Encapsulation

#include<iostream>
using namespace std;

class Encapsulation
{
    private:
        // data hidden from outside world
        int x;

    public:
        // function to set value of
        // variable x
        void set(int a)
        {
            x =a;
        }

        // function to return value of
        // variable x
        int get()
        {
            return x;
        }
};

// main function
int main()
{
    Encapsulation obj;

    obj.set(5);

    cout<<obj.get();
    return 0;
}
```

output:

Polymorphism

- The word polymorphism means having many forms.
- Real life example of polymorphism, a person at a same time can have different characteristic. Like a man at a same time is a father, a husband, a employee.
- The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism.

Inheritance

- Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
- Reuse the code functionality and fast implementation time.
- The existing class is called the base class, and the new class is referred to as the derived class.

Inheritance

Class Bus

fuelAmount()
capacity()
applyBrakes()

Class Car

fuelAmount()
capacity()
applyBrakes()

Class Truck

fuelAmount()
capacity()
applyBrakes()

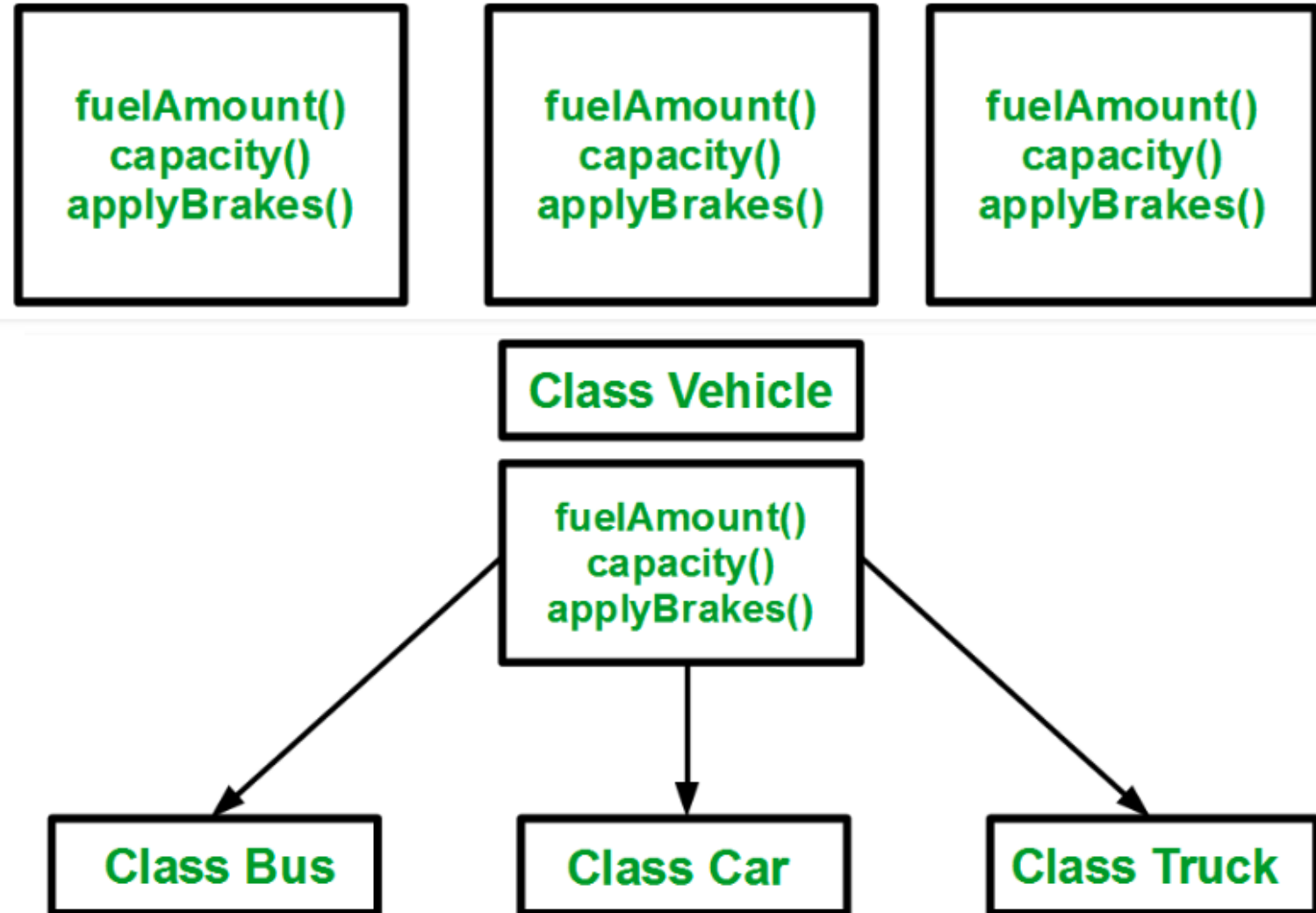
Class Vehicle

fuelAmount()
capacity()
applyBrakes()

Class Bus

Class Car

Class Truck



Inheritance - Example

```
#include <iostream>

using namespace std;

// Base class
class Shape {
public:
    void setWidth(int w) {
        width = w;
    }
    void setHeight(int h) {
        height = h;
    }

protected:
    int width;
    int height;
};

// Derived class
class Rectangle: public Shape {
public:
    int getArea() {
        return (width * height);
    }
};

int main(void) {
    Rectangle Rect;

    Rect.setWidth(5);
    Rect.setHeight(7);

    // Print the area of the object.
    cout << "Total area: " << Rect.getArea() << endl;

    return 0;
}
```

Total area: 35

Scope of Inheritance

When deriving a class from a base class, the base class may be inherited through **public**, **protected** or **private** inheritance. The type of inheritance is specified by the access-specifier as explained above.

We hardly use **protected** or **private** inheritance, but **public** inheritance is commonly used. While using different type of inheritance, following rules are applied –

- **Public Inheritance** – When deriving a class from a **public** base class, **public** members of the base class become **public** members of the derived class and **protected** members of the base class become **protected** members of the derived class. A base class's **private** members are never accessible directly from a derived class, but can be accessed through calls to the **public** and **protected** members of the base class.
- **Protected Inheritance** – When deriving from a **protected** base class, **public** and **protected** members of the base class become **protected** members of the derived class.
- **Private Inheritance** – When deriving from a **private** base class, **public** and **protected** members of the base class become **private** members of the derived class.

Constructors

- 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.
- A constructor is different from normal functions in following ways:
 - 1. Constructor has same name as the class itself
 - 2. Constructors don't have return type
 - 3. A constructor is automatically called when an object is created.
 - 4. If we do not specify a constructor, C++ compiler generates a default constructor for us (expects no parameters and has an empty body).
- Types of Constructors:
 - (1) Default Constructors:
 - (2) Parameterized Constructors:
 - (3) Copy Constructor:

Default Constructors:

1. **Default Constructors:** Default constructor is the constructor which doesn't take any argument. It has no parameters.

```
// Cpp program to illustrate the
// concept of Constructors
#include <iostream>
using namespace std;

class construct
{
public:
    int a, b;

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

int main()
{
    // Default constructor called automatically
    // when the object is created
    construct c;
    cout << "a: " << c.a << endl << "b: " << c.b;
    return 1;
}
```

```
a: 10
b: 20
```

Parameterized Constructors:

2. **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.

```
// CPP program to illustrate
// parameterized constructors
#include<iostream>
using namespace std;

class Point
{
    private:
        int x, y;
    public:
        // Parameterized Constructor
        Point(int x1, int y1)
        {
            x = x1;
            y = y1;
        }

        int getX()
        {
            return x;
        }
        int getY()
        {
            return y;
        }
};

int main()
{
    // Constructor called
    Point p1(10, 15);

    // Access values assigned by constructor
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

    return 0;
}
```

Output:

```
p1.x = 10, p1.y = 15
```


Copy Constructor:

A copy constructor is a member function which initializes an object using another object of the same class. A copy constructor has the following general function prototype:

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

Following is a simple example of copy constructor.

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

class Point
{
private:
    int x, y;
public:
    Point(int x1, int y1) { x = x1; y = y1; }

    // Copy constructor
    Point(const Point &p2) {x = p2.x; y = p2.y; }

    int getX()          { return x; }
    int getY()          { return y; }
};

int main()
{
    Point p1(10, 15); // Normal constructor is called here
    Point p2 = p1;    // Copy constructor is called here

    // Let us access values assigned by constructors
    cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
    cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();

    return 0;
}
```

Output:

```
p1.x = 10, p1.y = 15
p2.x = 10, p2.y = 15
```

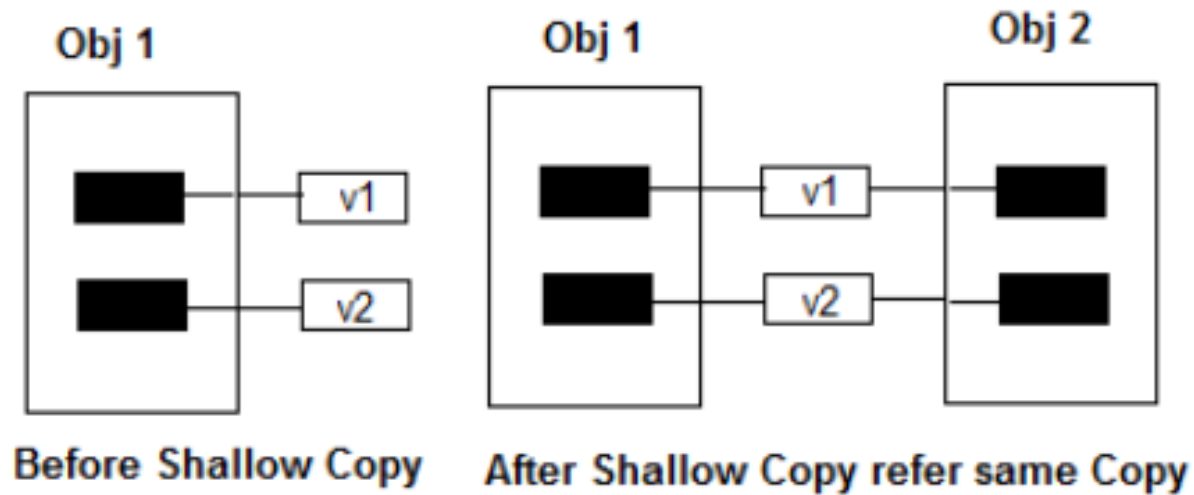
Assignment Operator (=)

- An assignment operator is used to replace the data of a previously initialized object with some other object's data.
- `Obj1 = Obj2` where Objs are of the same/different type

Shallow copy

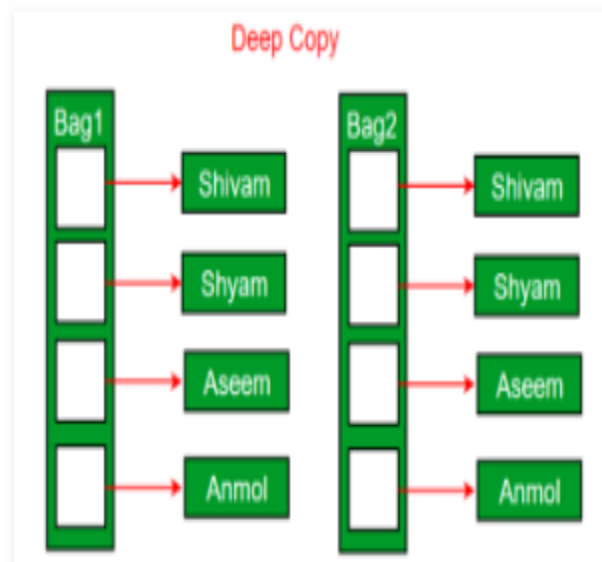
- Shallow copy is a bit-wise copy of an object. A new object is created that has an exact copy of the values in the original object.
- The default copy constructor and the assignment operators use the copying method known as member wise copy or shallow copy.
- The most common problem in Shallow copy is when a pointer to data that's owned by the object so when the object is destroyed the data it owns gets destroyed as well.

Shallow copy



Deep copy







Deep copy is possible only with user defined copy constructor. In user defined copy constructor, we make sure that pointers (or references) of copied object point to new memory locations.



Destructor

- A destructor is also a special function which is called when created object is deleted.
- Declared with (~ tilde sign).
- A destructor is a special member function that is called when the lifetime of an object ends. The purpose of the destructor is to free the resources that the object may have acquired during its lifetime.

More Concepts

Sr.No	Concept & Description
1	Class Member Functions  A member function of a class is a function that has its definition or its prototype within the class definition like any other variable.
2	Class Access Modifiers  A class member can be defined as public, private or protected. By default members would be assumed as private.
3	Constructor & Destructor  A class constructor is a special function in a class that is called when a new object of the class is created. A destructor is also a special function which is called when created object is deleted.
4	Copy Constructor  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.
5	Friend Functions  A friend function is permitted full access to private and protected members of a class.
6	Inline Functions  With an inline function, the compiler tries to expand the code in the body of the function in place of a call to the function.

Friend Functions

- 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.
- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

Friend Functions - Example

```
#include <iostream>

using namespace std;

class Box {
    double width;

    public:
        friend void printWidth( Box box );
        void setWidth( double wid );
};

// Member function definition
void Box::setWidth( double wid ) {
    width = wid;
}

// Note: printWidth() is not a member function of any class.
void printWidth( Box box ) {
    /* Because printWidth() is a friend of Box, it can
       directly access any member of this class */
    cout << "Width of box : " << box.width << endl;
}

// Main function for the program
int main() {
    Box box;

    // set box width without member function
    box.setWidth(10.0);

    // Use friend function to print the width.
    printWidth( box );

    return 0;
}
```

Width of box : 10