



# **Friend Class and Function in C++**





# Friend Class

- **Friend Class** *can access private and protected members of other class in which it is declared as friend.*
- It is sometimes useful to allow a particular class to access private members of other class.
- For example; a LinkedList class may be allowed to access private members of Node.

```
.....  
class B;  
class A  
{  
    ///| class B is a friend class of class A  
    friend class B;  
    .....  
}  
  
class B  
{  
    .....  
}
```





# Example of Friend Class

```
1  #include <iostream>
2  using namespace std;
3  class A
4  {
5      int x = 5;
6      friend class B; /// friend class.
7  };
8  class B
9  {
10     public:
11         void display(A &a)
12         {
13             cout<<"value of x is : "<<a.x;
14         }
15     };
16     int main()
17     {
18         A a;
19         B b;
20         b.display(a);
21         return 0;
22     }
```





## Example-02 of Friend Class

```
1  #include<iostream>
2  using namespace std;
3  class A{
4  private:
5      int x = 5;
6      friend class B;
7  };
8  class B{
9  private:
10     A obj;
11     int p = obj.x;
12 public:
13     void display()
14     {
15         cout<<p<<endl;
16     }
17 };
18 int main()
19 {
20     A a;
21     B ob;
22     ob.display();
23     return 0;
24 }
```





# Friend Function

- If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.
- By using the keyword friend compiler knows the given function is a friend function.
- For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.

```
class class_name
{
    friend data_type function_name(argument/s);    // syntax of friend function.
};
```

- In the above declaration, the friend function is preceded by the keyword friend. The function can be defined anywhere in the program like a normal C++ function. The function definition does not use either the keyword **friend** or **scope resolution operator**.





# Characteristics of Friend Function

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in the private or the public part.





# Example of Friend Function

```
1  #include <iostream>
2  using namespace std;
3  class Box
4  {
5      private:
6          int length;
7      public:
8          Box(): length() { }
9          friend int printLength(Box); ///friend function
10 };
11 int printLength(Box b)
12 {
13     b.length += 10;
14     return b.length;
15 }
16 int main()
17 {
18     Box b;
19     cout<<"Length of box: "<< printLength(b)<<endl;
20     return 0;
21 }
```





# Addition of members of two different classes using friend Function

```
1  #include <iostream>
2  using namespace std;
3  class B;
4  class A
5  {
6      private:
7          int numA;
8      public:
9          A(): numA(12) { }
10         /// friend function declaration
11         friend int add(A, B);
12     };
13     class B
14     {
15         private:
16             int numB;
17         public:
18             B(): numB(1) { }
19             /// friend function declaration
20             friend int add(A, B);
21     };
22     /// Function add() is the friend function of classes A and B
23     /// that accesses the member variables numA and numB
24     int add(A objectA, B objectB)
25     {
26         return (objectA.numA + objectB.numB);
27     }
28     int main()
29     {
30         A objectA;
31         B objectB;
32         cout<<"Summation is: "<< add(objectA, objectB)<< endl;
33         return 0;
34     }
```







# Interfaces in C++ (Abstract Classes)

➤ ***Abstract classes are the way to achieve abstraction in C++.***

Abstraction in C++ is the process to hide the internal details and showing functionality only. Abstraction can be achieved by two ways:

- ✓ Abstract class
- ✓ Interface

➤ Abstract class and interface both can have abstract methods which are necessary for abstraction.

➤ In C++ class is made abstract by declaring at least one of its functions as <strong>pure virtual function. A pure virtual function is specified by placing ***“function = 0”*** in its declaration. Its ***implementation must be provided by derived classes.***





# Interfaces

- An interface describes ***the behavior or capabilities of a C++ class without committing to a particular implementation*** of that class.
- The C++ interfaces are implemented using **abstract classes** and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation details separate from associated data.





# Interfaces

```
class Box {  
    public:  
        // pure virtual function  
        virtual double getVolume() = 0;  
  
    private:  
        double length;           /// Length of a box  
        double breadth;          /// Breadth of a box  
        double height;           /// Height of a box  
};
```



# Example of Abstract Class

- Consider the following example where parent class provides an interface to the base class to implement a function

```
1  #include <iostream>
2  using namespace std;
3  /// Base class
4  class Shape {
5  public:
6      /// pure virtual function
7      ///providing interface framework.
8      virtual int getArea() = 0;
9      void setWidth(int w) {
10         width = w;
11     }
12
13     void setHeight(int h) {
14         height = h;
15     }
16
17     protected:
18         int width;
19         int height;
20 };
21
22 /// Derived classes
23 class Rectangle: public Shape {
24 public:
25     int getArea() {
26         return (width * height);
27     }
28 };
```

```
30 class Triangle: public Shape {
31 public:
32     int getArea() {
33         return (width * height)/2;
34     }
35 };
36
37 int main(void) {
38     Rectangle Rect;
39     Triangle Tri;
40
41     Rect.setWidth(5);
42     Rect.setHeight(7);
43
44     // Print the area of the object.
45     cout << "Total Rectangle area: " << Rect.getArea() << endl;
46
47     Tri.setWidth(5);
48     Tri.setHeight(7);
49
50     /// Print the area of the object.
51     cout << "Total Triangle area: " << Tri.getArea() << endl;
52     return 0;
53 }
```





# Thank You

