Friend Class and Function in C++

Friend Class

- **Friend Class** A 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

```
#include <iostream>
using namespace std;
class Box
{
    private:
        int length;
    public:
        Box(): length() { }
        friend int printLength(Box); ///friend function
};
int printLength(Box b)
{
    b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Addition of members of two different classes using friend Function

```
#include <iostream>
using namespace std;
class B;
class A
{
    private:
        int numA;
    public:
        A(): numA(12) { }
        /// friend function declaration
        friend int add(A, B);
};
class B
{
    private:
        int numB;
    public:
        B(): numB(1) { }
        /// friend function declaration
        friend int add(A, B);
};
/// Function add() is the friend function of classes A and B
/// that accesses the member variables numA and numB
int add(A objectA, B objectB)
{
    return (objectA.numA + objectB.numB);
}
int main()
{
    A objectA;
    B objectB;
    cout<<"Summation is: "<< add(objectA, objectB)<< endl;
    return 0;
}
```

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 "= 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.
- A class is made abstract by declaring at least one of its functions as **pure virtual** function. A pure virtual function is specified by placing "= 0" in its declaration as follows −

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 called **getArea()** −

```
#include <iostream>
using namespace std;
/// Base class
class Shape {
   public:
      /// pure virtual function
      ///providing interface framework.
      virtual int getArea() = 0;
      void setWidth(int w) {
         width = w;
      }

      void setHeight(int h) {
         height = h;
      }

   protected:
      int width;
      int height;
};

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

```
class Triangle: public Shape {
   public:
      int getArea() {
         return (width * height)/2;
      }
};

int main(void) {
   Rectangle Rect;
   Triangle  Tri;

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

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

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

   /// Print the area of the object.
   cout << "Total Triangle area: " << Tri.getArea() << endl;
   return 0;
}
```

Thank You