CS-1201 Object-Oriented Programming

Friend Functions and Classes

Arbish Akram

Department of Computer Science
Government College University

Why Do We Need Friends?

- Encapsulation and data hiding dictate that *non-member functions should not access an object's private or protected data*, if you're not a member, you can't get in.
- A function may need to operate on objects of two different classes at once, using the private data of both.
- C++ provides the *friend* mechanism as a controlled way to relax encapsulation when it is genuinely useful.

Friend Function and Class

- A *friend* function or class is *defined outside a class's scope*, yet is granted access to that class's *private and protected members*.
- Standalone functions, entire classes, or member functions of *other classes* may all be declared friends of a class.
- Friend functions can improve performance by avoiding the overhead of extra member-function calls or copying.

Declaring a Friend

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

```
class ClassOne {
    friend void someFunction( ClassOne &, int );
    // ...
};
```

- To make all member functions of `ClassTwo` friends of `ClassOne`, place a declaration of this form inside `ClassOne`:

```
friend class ClassTwo;
```

- This single declaration grants friendship to every member function of `ClassTwo` at once.

Example 1: Friend Function

```
1  // C++ program to demonstrate the working of the friend function
2   class Distance {
3       private:
4           int meter;
5           // friend function
6           friend int addFive(Distance);
7       public:
8           Distance() : meter(0) {}
9   };
10  // friend function definition
11  int addFive(Distance d) {
12      //accessing private members from the friend function
13      d.meter += 5;
14      return d.meter;
15  }
16  int main() {
17      Distance D;
18      cout << "Distance: " << addFive(D);
19      return 0;
20  }
```

Properties of Friendship

- Friendship is granted, not taken, for class `B` to be a friend of class `A`, class `A` must explicitly declare `B` as its friend.
- Friendship is not symmetric: if `A` is a friend of `B`, you cannot infer that `B` is a friend of `A`.
- Friendship is not transitive: if `A` is a friend of `B`, and `B` is a friend of `C`, you cannot infer that `A` is a friend of `C`.
- Even someone without access to a class's source code cannot grant themselves friendship, only the class itself can declare who its friends are, so the integrity of the class is still protected.

Example: Modifying Private Data

```
1  // Friends can access private members of a class.
2  #include <iostream>
3  using namespace std;
4  class Count {
5      friend void setX( Count &, int ); // friend declaration
6      public:
7          Count() : x( 0 ) {} // constructor
8          void print() const { cout << x << endl; }
9      private:
10         int x; // data member
11 };
12 // setX can modify private data of Count
13 // because setX is declared as a friend of Count
14 void setX( Count &c, int val ) {
15     c.x = val; // allowed because setX is a friend of Count
16 }
17 int main() {
18     Count counter;
19     cout << "counter.x after instantiation: ";
20     counter.print();
21     setX( counter, 8 ); // set x using a friend function
22     cout << "counter.x after call to setX friend function: ";
23     counter.print();
24 }
```

Example 2: Friend Function

```
1  // Add members of two different classes using friend functions
2   #include <iostream>
3   using namespace std;
4   // forward declaration
5   class ClassB;
6   class ClassA {
7       public:
8           // constructor to initialize numA to 12
9           ClassA() : numA(12) {}
10      private:
11          int numA;
12           // friend function declaration
13           friend int add(ClassA, ClassB);
14  };
15  class ClassB {
16      public:
17          // constructor to initialize numB to 1
18          ClassB() : numB(1) {}
19      private:
20          int numB;
21          // friend function declaration
22          friend int add(ClassA, ClassB);
23  };
```

Example 2: Friend Function

```
1  // access members of both classes
2  int add(ClassA objectA, ClassB objectB) {
3      return (objectA.numA + objectB.numB);
4  }
5
6  int main() {
7      ClassA objectA;
8      ClassB objectB;
9      cout << "Sum: " << add(objectA, objectB);
10     return 0;
11 }
```

- `ClassA` and `ClassB` have each declared `add()` as a friend, so it can access the private data of both classes.
- The friend declaration inside `ClassA` may refer to `ClassB` even before `ClassB` is fully defined, that's what the forward declaration on the previous slide made possible.

Friend Class

- A friend class can access the private and protected members of another class using the `friend` keyword.
- All member functions of `ClassB` become friend functions of `ClassA`.
- `ClassB` can access all members of `ClassA`.
- However, `ClassA` cannot access members of `ClassB`.

```
class ClassB; // Forward declaration
class ClassA {
    friend class ClassB; // ClassB is a friend class
    // ClassA members
};
class ClassB {
    // ClassB members
};
```

Example: Friend Class

```
1  // C++ program to demonstrate the working of friend class
2  #include <iostream>
3  using namespace std;
4
5  // forward declaration
6  class ClassB;
7
8  class ClassA {
9      private:
10         int numA;
11         // friend class declaration
12         friend class ClassB;
13
14     public:
15         // constructor to initialize numA to 12
16         ClassA() : numA(12) {}
17
18 };
```

Example: Friend Class

```
1  class ClassB {
2      private:
3          int numB;
4      public:
5          // constructor to initialize numB to 1
6          ClassB() : numB(1) {}
7      // member function to add numA
8      // from ClassA and numB from ClassB
9      int add() {
10         ClassA objectA;
11         return objectA.numA + numB;
12     }
13 };
14
15 int main() {
16     ClassB objectB;
17     cout << "Sum: " << objectB.add();
18     return 0;
19 }
```

Summary

- A `friend` function or class is *not a member*, yet can access a class's private and protected members.
- Friendship must be *explicitly granted* by the class being accessed, it is never taken, and is *neither symmetric nor transitive*.