Object Oriented Programming with C++ Day 3

Compiled by
M S Anand

Department of Computer Science

OOP with C++

Introduction

Text Book(s):

1. "C++ Primer", Stanley Lippman, Josee Lajoie, Barbara E Moo, Addison-Wesley Professional, 5th Edition..

Friend functions

What is Friend Function?
A friend function in C++ is a function that is declared outside (or inside) a class and 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 or function templates. In such cases, we make the desired function, a friend to both these classes which will allow accessing private and protected data of members of the class.

Generally, non-member functions cannot access the private members of a particular class. Once declared as a friend function, the function is able to access the private and the protected members of these classes.

Friend functions

Friend functions in C++ have the following types

1. Function with no argument and no return value
2. Function with no argument but with return value
3. Function with argument but no return value
4. Function with argument and return value

Friend functions

Declaration of a friend function in C++

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

In the above declaration, the keyword **friend** precedes the function.

We can define the friend function anywhere in the program like a normal C++ function. A class's function definition does not use either the keyword **friend or scope resolution operator.**

Friend function is called as function_name(class_name) and member function is called as class_name. function_name.

Friend functions

Use of Friend function in C++
As discussed, we require friend functions whenever we have to access the private or protected members of a class. This is only the case when we do not want to use the objects of that class to access these private or protected members.

To understand this better, let us consider two classes: Tokyo and Rio. We might require a function, metro(), to access both these classes without any restrictions. Without the friend function, we will require the object of these classes to access all the members. Friend functions in c++ help us avoid the scenario where the function has to be a member of either of these classes for access.

OOP with C++

Friend functions

Characteristics of Friend Function in C++

1. The function is not in the 'scope' of the class to which it has been declared a friend.
2. Friend functionality is not restricted to only one class
3. Friend functions can be a member of a class or a function that is declared outside the scope of class.
4. <u>It cannot be invoked using the object as it is not in the scope of that class</u>.
5. We can invoke it like any normal function of the class.
6. Friend functions <u>have objects as arguments</u>.
7. It cannot access the member names directly and has to use dot membership operator and use an object name with the member name.
8. We can declare it either in the 'public' or the 'private' part.

Friend functions

Implementing Friend Functions
Friend Functions can be implemented in two ways:
A method of another class:
We declare a friend class when we want to access the non-public data members of a particular class.
A Global function:
A ‘global friend function’ allows you to access all the private and protected members of the global class declaration.
A simple example of a C++ friend function used to print the length of the box: friend1.cpp

One more example: Friend2.cpp
In the above example, max () function is friendly to both class A and B, i.e., the max () function can access the private members of two classes.

Friend class

A class cannot access the private members of another class. Similarly, a class cannot access its protected members . We need a friend class in this case.
A friend class is used when we need to access private and protected members of the class in which it has been declared as a friend. It is also possible to declare only one member function of another class to be a friend.

Declaration of friend class

```
class class_name
{
     friend class friend_class;// declaring friend class
};
class friend_class
{
};
```

Friend class

All functions in friend_class are friend functions of class_name.

Sample program Friend3.cpp

Implementing a global function

Sample program: Friend4.cpp

Friend class

<u>Quick summary</u>

Friend Class is a class that can access both private and protected variables of the class in which it is declared as a friend, just like a friend function. Classes declared as friends to any other class will have all the member functions as friend functions to the friend class. Friend functions are used to link both these classes.

OOP with C++

Friend class

Friend Class in C++ Syntax:

```
class One{
<few lines of code here>
friend class Two;
};
class Two{
<few lines of code>
};
```

Note : Unless and until we declare, class friendship is neither mutual nor inherited.

Friend class

What does it mean?

1. If class A is a friend of class B, then class B is not a friend of class A.

2. Also, if class A is a friend of class B, and then class B is a friend of class C, class A is not a friend of class C.

3. If Base class is a friend of class X, subclass Derived is not a friend of class X; and if class X is a friend of class Base, class X is not a friend of subclass Derived.

Operator functions

Operator functions
An operator function can be either a nonstatic member function, or a nonmember function with at least one parameter that has class, reference to class, enumeration, or reference to enumeration type.
You cannot change the precedence, grouping, or the number of operands of an operator.

What is an operator function?
A function which defines additional tasks to an operator or which gives special meaning to an operator is called an operator function.

Operator functions

The general form of operator function is:

```
return-type class-name :: operator op (argument list)
{
            // Function body
}
```

return-type is the value returned by the specified operation and op is the operator being overloaded. **operator** is a keyword.

<u>Operator functions must be either member functions (non-static) or friend functions</u>

Operator overloading

Operator overloading is a compile-time polymorphism. It is an idea of giving special meaning to an existing operator in C++ without changing its original meaning.
In C++, we can make operators work for user-defined classes. This means C++ has the ability to provide the operators with a special meaning for a data type, this ability is known as operator overloading. For example, we can overload an operator '+' in a class like String so that we can concatenate two strings by just using +.

The main idea behind "Operator overloading" is to use C++ operators with class variables or class objects. Redefining the meaning of operators really does not change their original meaning; instead, they have been given additional meaning along with their existing ones.

An example program OverloadPlus.cpp

Operator overloading

What is the difference between operator functions and normal functions?

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

Another example: OperatorPlus2.cpp

Operator overloading

Can we overload all operators?
Almost all operators can be overloaded except a few.

<u>Operators that can be overloaded</u>
Unary operators
Binary operators
Special operators ([ ], (), etc)

Operator overloading

Which operators Cannot be overloaded?

Conditional [?:], sizeof, scope(::), Member selector(.), member pointer selector(.*) and the casting operators.

- We can only overload the operators that exist and cannot create new operators or rename existing operators.
- At least one of the operands in overloaded operators must be user-defined, which means we cannot overload the minus operator to work with one integer and one double. However, you could overload the minus operator to work with an integer and a mystring.
- It is not possible to change the number of operands of an operator supports.
- All operators keep their default precedence and associations (what they use for), which cannot be changed.
- Only built-in operators can be overloaded.

Operator overloading

The process of selecting the most suitable overloaded function or operator is called overload resolution.

Another example: MultiplyOverload.cpp

One more PrefixDecrement.cpp

OOP with C++

Operator overloading

The subscript operator [] is normally used to access array elements. This operator can be overloaded to enhance the existing functionality of C++ arrays.

The program IndexOverload.cpp

Conversion functions

Note

If you make a function that takes a reference to a object of your class, and you pass it another type other than your object the constructor of your class will convert that type to an object of your class.

Conversion functions

Conversion functions

Conversion functions convert a value from one datatype to another. User-defined data types are designed by the user to suit their requirements, the compiler does not support automatic type conversions for such data types; therefore, the users need to design the conversion routines by themselves if required.

There can be 3 types of situations that may come in the data conversion between incompatible data types:

Conversion of primitive data type to user-defined type: To perform this conversion, the idea is to use the constructor to perform type conversion during the object creation.

Sample program: Conversion1.cpp

Conversion functions

Conversion of class object to primitive data type: In this conversion, the **from** type is a class object and the **to** type is primitive data type. The normal form of an overloaded casting operator function, also known as a conversion function. Below is the syntax for the same:

Syntax:

```
operator typename()
{
        // Code
}
```

Now, this function converts a **user-defined data type** to a **primitive data type**. For Example, the operator **double()** converts a class object to type double, the operator **int()** converts a class type object to type int, and so on. Below is the program to illustrate the same:

Conversion2.cpp

Conversion functions

Conversion of one class type to another class type: In this type, one class type is converted into another class type. It can be done in 2 ways :

.

1.Using constructor
2.Using Overloading casting operator

1.Using constructor :

In the Destination class we use the constructor method
//Objects of different types ObjectX=ObjectY;
Here ObjectX is Destination object and ObjectY is source object

Sample: Conversion3.cpp.

Conversion functions

2.Using Overloading casting operator
// Objects of different types objectX = objectY;
Here we use Overloading casting operator in source class i.e. overloading destination class in source class

See the below example in which we have two classes Time and Minute respectively and will convert one class Time to another Minute class.
In the example, minute class is destination class and time class is source class so we need to overload the destination class in the source class
Here we should not tell the return type but we return the overloaded class object
i.e. returning value without specifying return type

Sample code: Conversion4.cpp

Operator overloading

Insertion and Extraction operator

In C++, stream insertion operator “<<” is used for output and extraction operator “>>” is used for input.

We must know the following things before we start overloading these operators.

1) cout is an object of ostream class and cin is an object of istream class

2) These operators must be overloaded as a global function. And if we want to allow them to access private data members of the class, we must make them friend.

Sample program: InOutOverload.cpp

Static members

Static members
We can define class members static using **static** keyword. 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.

A static member is shared by all objects of the class. All static data is initialized to zero when the first object is created, if no other initialization is present. We can't put it in the class definition but it can be initialized outside the class as done in the following example by redeclaring the static variable, using the scope resolution operator **::** to identify which class it belongs to.

Sample Program StaticMember.cpp

Static members

Static Function Members

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 and 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.

Static member functions have a class scope and they do not have access to the **this** pointer of the class. You could use a static member function to determine whether some objects of the class have been created or not.

Sample program – StaticFunction.cpp

One more program: StaticMember2.cpp

Project initiation

THANK YOU

M S Anand
Department of Computer Science Engineering

anandms@yahoo.com