Scope and Access Control

Chapter 8

Objectives

- **Use C++ scoping facilities.**
- **Introduce constants into your programs through enumeration types and through the *const* keyword.**
- **Define "static members" and use them in your code.**
- **Control access to member data and functions through public, private, and protected access specifiers.**
- **Define "friend" function and explain how a friend function differs from a member function.**
- **Gain experience through code walk-throughs and lab exercises.**

  - The example programs are in the **chapter directory.**
  - Labs located in Labs/Lab8

Scoping in C++

- ***Scope* refers to the visibility of variables, symbolic constants and functions.**
  - **C++ has a richer set of facilities than C does for controlling access to variables.**
- **C++ supports:**
  - Block scope
  - Function scope
  - File scope
  - Global scope
  - Class scope
- **A *scope resolution operator* :: can make visible names that would otherwise be hidden.**

Block and Function Scope

- **A variable may be defined within a *block* and not be accessible from outside that block:**

```
int x, y;

if (x < y)
{
    int temp = x;
    x = y;
    y = temp;
}
temp = y; // illegal (temp out of scope)
```

Block and Function Scope (continued)

- **A local variable defined inside a function has function scope and is not accessible outside that function.**
- **An automatic variable part of the activation record of the function will not even exist after the function is exited.**
- **A local static variable will exist when the function is not active, but cannot be accessed from outside the function.**

File and Global Scope

- **A variable defined outside any function and with the keyword *static* has *file scope*. It can be accessed from any function in the file but not from outside the file.**
- **If the keyword *static* is omitted, the variable has *global scope* and can be accessed from any file where the variable is declared with the *extern* keyword.**
- **File scope is used in C to provide support for data hiding, but in C++ the more powerful capability of *access control* using *public* and *private* is available.**

Class Scope

- **The definition of a class can be distributed across multiple files, with necessary visibility of member data and functions achieved through use of the *class scope* operator ::**

```
#include "IntStack.h"

 void IntStack::Push(int x) {

         stack[top++] = x;

}
```

- **The data members stack and top are accessible directly without having to go through an invoking object, because they are within the scope of the class IntStack.**

Access Control

- **Access to data and function members of a class are governed by access specifiers:**
  - ***private***
  - ***protected***
  - ***public***

Protected access will be explained later when we discuss inheritance

```
class Employee{
public:
    Money CalculatePay();
    String GetName();
protected:
    float hours_worked;
private:
    String name;
};
```

Enumeration Types

- **An enumeration type declares a set of symbolic constants:**

  enum {SUN, MON, TUE, WED, THU, FRI, SAT};

- **Unlike *const*, there is no addressable storage associated with an enumerator:**

  &SUN // error

- **By default, the first enumerator is assigned value zero, and subsequent enumerators are assigned a value one greater than predecessor.**
- **Enumerators can be explicitly assigned integer values:**

  enum {STACKSIZE = 5};

- **An optional tag name can be used subsequently for declaring a variable to be of enumeration type:**

  enum Bit {OFF, ON};

  OR

  typedef enum {OFF, ON} Bit;
  Bit flag;
  flag = ON;

Enumeration Types and Class Scope

- **Within a class definition, an integer constant can be defined and initialized.**

```
class IntStack {
public:
    enum {STACKSIZE = 5};
    ...
private:
    int stack[STACKSIZE];
    int top;
};
```

```
int IntStack::IsFull(){
    return (top == STACKSIZE};
}
```

Enumeration Types and Class Scope (continued)

- **From outside the class, use the class scope operator.**

```
cout << "size of stack is “  << IntStack::STACKSIZE << endl;
```

- **Review the example program <u>EnumStack.</u>**

Enum Class

- **enum classes (C++ 11) are strongly typed and strongly scoped.**
  - do not allow implicit conversion to int.
  - do not compare enumerators from different enumerations.

```
// Declaration
enum class Color{ Black, Red, Green, Blue, White};

// Initialization
Color r = Color::Red;
```

- **Review the example program ClassEnum.**

:: for Global Data

- **The scope operator :: provides a solution to the problem of accessing hidden global data.**

```
int size = 5;
int main()
{
   int size = 10;              // hides global size
   cout << size << endl;       // local version
   cout << ::size << endl;     // global version
   return 0;
}
```

Static Class Members

- **Sometimes it is necessary for all objects of a class to have access to a particular variable.**
  - For example, in a user interface all child windows belonging to a certain Windows class may display their text in a common color.
- **A *static data member* is a single shared object accessible to all objects of a class.**
- **A data member is made static by prefixing its declaration with the keyword *static*.**
- **Global data is shared by all objects of a class, but static data members have advantages:**
  - Information hiding can be enforced. A static member can be made private, global data cannot
  - A static member is not entered into the program's global name space, avoiding possibility of an accidental name conflict

Initialization of Static Member

- **An ordinary data member is initialized in the constructor for the class.**
- **Static member data does not exist on a per-object basis, so it should not be initialized in a constructor.**
- **Static member data is defined and initialized in a code file, using class scope operator :: to access the data member:**
  - COLORREF Background::m_color = COLOR_RED;

  - Note if you do not define a static data member, you will get a link error.

Static Function Class Members

- **A member function which accesses only the static data members of a class may also be declared as static.**

```
class Background {
public:
    ...
    static SetColor(COLORREF color);
    static COLORREF GetColor();
    ...
private:
    static COLORREF m_color;
};
...
// in file using ChildWnd class
Background::SetColor(COLOR_GREEN);
```

Demo

- **Folder Instances contains a simple class that tracks the count of instances.**
- **Study the code to understand the logic and use of static members.**
- **Build and run the program.**

Friend Functions

- **Friend functions are an alternative to member functions for accessing private class data.**
- **Friend functions are:**
  - Declared in a class.
  - Have access to private data of the class.
  - Use the keyword friend.
  - Are not bound to an invoking object.

Invoking Member and Friend Functions

- **Member functions use invoking object and "dot" notation:**

```
Vector v1;
Vector v2;
...
cout << v1.DotProduct(v2) << endl;
```

- **Friend function in a case like this will have more natural notation:**

```
// declared inside the class
friend int DotProduct (const Vector& v1, const Vector& v2);

// invocation
cout << DotProduct(v1, v2) << endl;
```

Implementing a Friend Function

- **The code for a friend function does not use the class scope operator ::.**
  - Remember that a friend function is not a member function!

```
int DotProduct(const Vector& v1,
               const Vector& v2) {
   int sum = 0;
   for (int i = 0; i < Vector::size; i++)
      sum += v1.vec[i] * v2.vec[i];
   return sum;
}
```

Efficiency and Friend Functions

- **A friend function may be more efficient.**
  - It can gain access to member data directly, without going through access functions.
- **Friend functions break strict information hiding:**
  - Access to a class private data is permitted to a function outside the class.

```
/*
An entire class can be declared a friend
*/


class Window;
// Window can access all private,
protected, and public members of Screen
class Screen {
    friend class Window;
    . . .
};
```

Friend Demo

- **Folder Vector demonstrates friend functions, static members, and overloaded operators.**
- **Examine, build and run the program.**

Summary

- C++ supports scoping at the block, function, file, global and class level.
- The scope operator :: can be used to specify class scope and to gain access to global names that are hidden.
- Constants can be introduced into C++ programs through enumeration types and by the *const* prefix.
- Access to class members is governed by the keywords *public*, *protected*, and *private*.
- A static data member is a single shared object accessible to all members of its class.
- Friend functions have access to private data of a class and do not have an invoking object.