Secure Coding

6 NAME CONTROL

- 6.1 Namespace
- 6.2 Scoping
- 6.3 Static
- 6.4 Constants

6.1 Namespace

- C++ provides namespaces to prevent name conflicts.

For example, if each of two libraries has an identifier cout and an application tried to use both libraries, a conflict would result.

6.1 Namespace

We will routinely use the namespace std, which covers the standard C ++definitions, declarations, and so on for the standard C ++ library.

6.1 Namespace

Example:

```
namespace mfc
{
    int inflag;
}
namespace owl
{
    int inflag;
}
```

6.1 Namespace

A namespace can be used to disambiguate a name that otherwise would cause a conflict.

Example:

```
mfc::inflag=3;       //mfc's inflag
owl::inflag=-123;    //owl's inflag
```

6.1 Namespace

Use a using declaration for a shorthand.

Example:

```
using mfc::inflag;//using declaration for mfc::inflag
inflag=3;       //mfc::inflag
owl::inflag=-123;//namespace name needed
```

6.1 Namespace

Use a using directive for a shorthand.

Example:

```
using namespace mfc;//using directive
inflag=21;      //mfc's inflag
owl::inflag=-123;    //full name needed
```

6.1 Namespace

A namespace includes not only variables, but also functions.

Example:

```
namespace mfc
{
    int inflag;
    void g(int i);
}
```

6.2 Scoping

Scoping rules tell you where a variable is valid, where it is created, and where it gets destroyed .

The scope of a variable extends from the point where it is defined to the first closing brace that matches the closest opening brace before the variable was defined.

6.2 Scoping

Example: This example shows when variables are visible and when they are unavailable (that is, when they go out of scope).

scoping example1

6.2 Scoping

There is a significant difference between C and C++ when defining variables.

Both languages require that variables be defined before they are used, but C forces you to define all the variables at the beginning of a scope.

C++ (not C) allows you to define variables anywhere in a scope, so you can define a variable right before you use it.

6.2 Scoping

Example:

scoping example2

6.3 Static
Static Variables

- **Static Variables**

Normally, variables defined local to a function disappear at the end of the function scope.

When you call the function again, storage for the variables is created anew and the values are re-initialized.

6.3 Static
Static Variables

If you want a value to be extant throughout the life of a program, you can define a function's local variable to be **static** and give it an initial value.

The initialization is performed only the first time the function is called, and the data retains its value between function calls.

This way, a function can "remember" some piece of information between function calls.

6.3 Static
Static Variables

Example:

```
#include <iostream>
using namespace std;
void func( )
{
  static int i = 0;
  cout << "i = " << ++i << endl;
}
int main( )
{
  for(int x = 0; x < 10; x++)
     func( );
  return 0;
}
```

Output:

```
i = 1
i = 2
i = 3
i = 4
i = 5
i = 6
i = 7
i = 8
i = 9
i = 10
```

6.3 Static
Static Variables

Each time func( ) is called in the for loop, it prints a different value. If the keyword static is not used, the value printed will always be ‘1’.

6.3 Static
Static Variables

Why a global variable isn't used instead?

The beauty of a static variable is that it is unavailable outside the scope of the function, so it can't be inadvertently changed. This localizes errors.

6.3 Static
Static Variables

Example:

```
#include<iostream>
using namespace std;
void fun( );
void main( )
{
    fun( );
    fun( );
}
```

```
void fun( )
{
    static int a=1;
    int i=5;
    a++;
    i++;
    cout<<"i="<<i<<",a="<<a<<endl;
}
```

Output:

i=6, a=2

i=6, a=3

6.3 Static
Static Variables

- **global variables and classes**

Example:

```
#include<iostream>
using namespace std;
int global;
void f( )
{  global=5;}
void g( )
{  cout<<global<<endl;}
```

```
int main( )
{
    f( );
    g( );
    return 0;
}
```

The global variable is dangerous!

6.3 Static
Static Variables

Solution:

Example:

```
#include<iostream>
using namespace std;
class Application
{
public:
    void f( );
    void g( );
private:
    int global;
};
```

```
void Application::f( )
{  global=5;}

void Application::g( )
{  cout<<global<<endl;}

int main( )
{
   Application  MyApp;
   MyApp.f( );
   MyApp.g( );
   return 0;
}
```

6.3 Static
Static Variables

Example:

```
#include<iostream>
using namespace std;
class Clock
{
public:    //
        Clock( );
        void SetTime(int NewH, int NewM, int NewS);
        void ShowTime( );
        ~Clock( ){ }
private:   //
        int Hour, Minute, Second;
};
```

6.3 Static
Static Variables

```
Clock::Clock( )
{
    Hour=0;
    Minute=0;
    Second=0;
}
void Clock::SetTime(int NewH, int NewM, int NewS)
{
    Hour=NewH;
    Minute=NewM;
    Second=NewS;
}
void Clock::ShowTime( )
{   cout<<Hour<<":"<<Minute<<":"<<Second<<endl;}
```

6.3 Static
Static Variables

```
Clock globClock;
void main( )
{
    cout<<"First time output:"<<endl;
    globClock.ShowTime();
    globClock.SetTime(8,30,30);
    Clock myClock(globClock);
    cout<<"Second time output:"<<endl;
    myClock.ShowTime( );
}
```

Output:

First time output:

0:0:0

Second time output:

8:30:30

What's the difference of static and global variables?

6.3 Static
Static Members in C++

- **Static Data Members**

This is accomplished with static data members inside a class. There is a single piece of storage for a static data member, regardless of how many objects of that class you create.

All objects share the same static storage space for that data member, so it is a way for them to "communicate" with each other.

6.3 Static
Static Members in C++

The static data belongs to the class; its name is scoped inside the class and it can be public, private, or protected.

6.3 Static
Static Members in C++

Remarks:

A) All the objects own one copy of the static data members in a class.

B) Static data members must be initialized outside the class.

Example:

```
#include <iostream>
using namespace std;
class Point
{
public:
        Point(int xx=0, int yy=0) {X=xx; Y=yy; countP++; }
        Point(Point &p);
        int GetX( ) {return X;}
        int GetY( ) {return Y;}
        void GetC( ) {cout<<" Object id="<<countP<<endl;}
private:
        int X,Y;
        static int countP;
};
```

```
Point::Point(Point &p)
{
        X=p.X;
        Y=p.Y;
        countP++;
}
int Point::countP=0; // initialized outside the class Point
void main( )
{
        Point A(4,5);
        cout<<"Point A,"<<A.GetX( )<<","<<A.GetY( );
        A.GetC( );
        Point B(A);
        cout<<"Point B,"<<B.GetX( )<<","<<B.GetY( );
        B.GetC( );
}
```

Output:
Object id=1
Object id=2

6.3 Static
Static Members in C++

- **Static Methods**

Like static data members, static methods work for the class as a whole rather than for a particular object of a class.

Instead of making a global function that lives in and "pollutes" the global or local namespace, you bring the function inside the class.

6.3 Static
Static Members in C++

Remarks:

A) Static methods can only access the static member values and static member methods of the same class.

B) Static methods can be accessed by the class name and scope resolution.

6.3 Static
Static Members in C++

Example:

```
#include<iostream>
using namespace std;
class Application
{
  public:
     static void f( );
     static void g( );
  private:
     static int global;
};
int Application::global=0;
void Application::f( )
{  global=5;}
void Application::g( )
{  cout<<global<<endl;}
int main( )
{
      Application::f( );
      Application::g( );
       return 0;
}
```

6.3 Static
Static Members in C++

Example :

```
class A
{
public:
    static void f(A a);
private:
    int x;
};
```

```
void A::f(A a)
{
    cout<<x;     //wrong
    cout<<a.x;  //right
}
```

Example :

```
#include <iostream>
using namespace std;
class Point
{
public:
        Point(int xx=0, int yy=0) {X=xx;Y=yy;countP++;}
        Point(Point &p);
        int GetX( ) {return X;}
        int GetY( ) {return Y;}
        static void GetC( ) {cout<<" Object id="<<countP<<endl;}
private:
        int X,Y;
        static int countP;
}
```

```
Point::Point(Point &p)
{           X=p.X;
            Y=p.Y;
            countP++;
}
int Point::countP=0;
void main( )
{
            Point A(4,5);
            cout<<"Point A,"<<A.GetX( )<<","<<A.GetY( );
            A.GetC( );              //using object
            Point B(A);
            cout<<"Point B,"<<B.GetX( )<<","<<B.GetY( );
            Point::GetC( );     //using class name and scope resolution
}
```

6.4 Constants
Constant Variables

C++ introduces the concept of a named constant that is just like a variable, except that its value cannot be changed.

The modifier const tells the compiler that a name represents a constant. Any data type, built-in or user-defined, may be defined as const.

6.4 Constants
Constant Variables

Example:

```
const int x = 10;
```

Remark:

In C++, a const must always have an initialization value.

6.4 Constants
Constant Members in C++

- **Constant Reference**

Why we use constant reference?

Example:

```
#include<iostream>
using namespace std;
void display(const double& r)
// Constant reference cant not be modified
{   cout<<r<<endl;   }
```

```
int main( )
{
  double d(9.5);
  display(d);
  return 0;
}
```

r can't be modified!

6.4 Constants
Constant Members in C++

- **Constant Objects**

Example:

```
class A
{
 public:
    A(int i,int j) {x=i; y=j;}
                  ...
 private:
    int x,y;
};
A const a(3,4); //a is a const object, can not be modified
```

6.4 Constants
Constant Members in C++

Remark:

Constant objects can only access the constant methods.

6.4 Constants
Constant Members in C++

- **Constant Methods**

Class methods can be made const. If you declare a method const, you tell the compiler the method can be called for a const object.

A method that is not specifically declared const is treated as one that will modify data members in an object, and the compiler will not allow you to call it for a const object.

6.4 Constants
Constant Members in C++

Constant methods syntax:

```
type-of-return function-name(argument-list) const
{
   //body
}
```

6.4 Constants
Constant Members in C++

Remarks:

A) Using the key word const to define a constant member method.

B) Constant member method can not modify the data members.

C) The keyword const can be used to make a function reloading.

6.4 Constants
Constant Members in C++

Example:

```
#include<iostream>
using namespace std;
class R
{
public:
    R(int r1, int r2){R1=r1;R2=r2;}
    void print( );
    void print( ) const;
private:
    int R1,R2;
};
```

6.4 Constants
Constant Members in C++

```
void R::print( )
{    cout<<R1<<":"<<R2<<endl;  }
void R::print( ) const
{    cout<<R1<<";"<<R2<<endl;  }
void main( )
{
    R a(5,4);
    a.print( );  // void print()
   const R b(20,52);
    b.print( );  // void print() const
}
```

6.4 Constants
Constant Members in C++

- **Constant Data Members**

Example:

```
#include<iostream>
using namespace std;
class A
{
public:
    A(int i);
    void print( );
    const int& r;
private:
    const int a;
    static const int b;   //static constant data member
};
```

6.4 Constants
Constant Members in C++

```
const int A::b=10;    // initialized outside the class
A::A(int i):a(i),r(a) { }
void A::print( )
{   cout<<a<<":"<<b<<":"<<r<<endl; }
void main( )
{
   A a1(100),a2(0);  //r and b are initialized
   a1.print( );
   a2.print( );
}
```

Must be initialized in the initializer list

Output:

100:10:100

0:10:0

Summarize

- Namespace
- Scoping
- Static Data Members
- Static Methods
- Constant Objects
- Constant Methods
- Constant Data Members

- **initialization**

  static variable

  static object

- **constructor**

  global static object

  static object

- **initialization**

  static

  const

  static const

  const data member

  static data member

  static const data member

  static data member in nested class

- **this**

  member function

  static member function