Chapter 9

More on Classes and Objects

Chapter 9 Topics (Part 3)

- Static Member（静态成员）
  - Static Data Member
  - Static Member Function
- Friend（友元）
  - Friend Function
  - Friend Member
  - Friend Class
- Class Template（类模板）

What is Static Member?

Static member can be

- static data member
- static member function

Static member has only one copy shared by all the instances of a class.

Static Data Member Declaration

SYNTAX

```
static DataType StaticDataMemberName;
```

Their memory remains allocated throughout execution of the entire program.

Initializing Static Data Members

SYNTAX

```
DataType ClassName::StaticDataMemberName=InitialValue;
```

- This assignment statement must locate outside the class block and any other body blocks.
- Initializer can not be used to specify the initial values for static dada members.

Accessing Static Public Data Members

SYNTAX

```
ObjectName . StaticDataMemberName
```

OR

```
ClassName :: StaticDataMemberName
```

The static dada members can be accessed before any object is created.

Example

```
# include <iostream>
using namespace std;
class Box {
public:
  Box(int,int);
  int volume( );
  static int height;
private:
  int width;
  int length;
};
Box::Box(int w,int len)
{  width=w; length=len;}
```

```
int Box::volume( )
{return(height*width*length); }
int Box::height=10;

int main( )
{
   Box a(15,20),b(20,30);
   cout<<a . height<<endl;
   cout<<b . height<<endl;
   cout<<Box :: height<<endl;
   cout<<a.volume( )<<endl;
   return 0;
}
```

Static Member Function Declaration

SYNTAX

```
static DataType StaticMemberFunctionName(Parameter List);
```

OR

```
static DataType StaticMemberFunctionName(Parameter List)
{
                    //body block
}
```

Static member functions are mainly used to access static data members.

Accessing Static Public Member Functions

SYNTAX

```
ObjectName . StaticMemberFunctionName(Argument List)
```

OR

```
ClassName :: StaticMemberFunctionName(Argument List)
```

The static member function does not have pointer `this`.

Example

```
# include <iostream>
using namespace std;
class SmallCat {
public:
  SmallCat(double w);
  void display( );
  static void total_display( );
private:
  double weight;
  static double total_weight;
  static double total_number;
};
SmallCat :: SmallCat(double w)
{
  weight=w;
  total_weight+=w;
  total_number++;
}
```

```
void SmallCat :: display( )
{cout<<“The small cat weights ”
        <<weight<<“ pounds.”<<endl;
}
void SmallCat :: total_display( )
{cout<< total_number
        <<“ small cats total weight ”
        << total_weight<< “pounds.”<<endl;
}
double SmallCat :: total_weight=0;
double SmallCat :: total_number =0;
int main( )
{ SmallCat w1(1.8),w2(1.5);
   w1.display( );
   w2.display( );
   SmallCat :: total_display( );
   return 0;
}
```

Chapter 9 Topics (Part 3)

- Static Member
  - Static Data Member
  - Static Member Function
- Friend
  - Friend Function
  - Friend Member
  - Friend Class
- Class Template

What is Friend?

Friend can be

- external function
- member function of another class
- a whole class

Friend can access all members of a class directly from the outside.

Friend Function Declaration

SYNTAX

```
class ClassName{
   friend DataType FunctionName(Parameter List);
   ...
}
```

```
DataType FunctionName(Parameter List)
{
   ...
}
```

Example

```
# include <iostream>
using namespace std;
class Box {
public:
  Box(int,int,int);
  friend void show(Box & );
private:
  int width;
  int length;
  int height;
};
```

```
Box::Box(int w,int len,int hei)
{  width=w; length=len;height=hei;}
void show(Box &b )
{cout<<"volume:"
        <<b.height*b.width*b.length
        <<endl;}
int main( )
{  Box a(15, 20,10);
   show(a );
   return 0;
}
```

Friend Member Declaration

SYNTAX

```
class ClassName1{
   DataType FunctionName(Parameter List);
   ...
}
```

```
class ClassName2{
   friend DataType ClassName1 ::FunctionName(Parameter List);
   ...
}
```

Example

```
# include <iostream>
using namespace std;
class Box;   //pre-reference declaration
class GoldenBox{
public:
  GoldenBox(int);
  void show(Box & );
private:
   int weight;
};
class Box {
public:
  Box(int ,int ,int );
friend void GoldenBox::show(Box & );
private:
  int width;
  int length;
  int height;
};
```

```
GoldenBox::GoldenBox(int wei)
{  weight=wei;  }
 void GoldenBox ::show(Box &b )
{ cout<<"volume:"
        <<b.height*b.width*b.length
        <<",  weight:"<<weight<<endl;}
Box::Box(int w,int len,int hei)
{  width=w; length=len;height=hei;}
 int main( )
{  Box a(15, 20,10);
   GoldenBox b(100);
   b.show(a);
   return 0;
}
```

Friend Class Declaration

SYNTAX

```
class ClassName1{
   ...
}
```

```
class ClassName2{
   friend ClassName1;
   ...
}
```

Example

```
# include <iostream>
using namespace std;
class Box;   //pre-reference declaration
class GoldenBox{
public:
  GoldenBox(int);
  void show(Box & );
private:
   int weight;
};
class Box {
public:
  Box(int ,int ,int );
  friend GoldenBox;
private:
  int width;
  int length;
  int height;
};
```

```
GoldenBox::GoldenBox(int wei)
{  weight=wei;  }
 void GoldenBox ::show(Box &b )
{ cout<<"volume:"
        <<b.height*b.width*b.length
        <<",  weight:"<<weight<<endl;}
Box::Box(int w,int len,int hei)
{  width=w; length=len;height=hei;}
 int main( )
{  Box a(15, 20,10);
   GoldenBox b(100);
   b.show(a);
   return 0;
}
```

Chapter 9 Topics (Part 3)

- Static Member
  - Static Data Member
  - Static Member Function
- Friend
  - Friend Function
  - Friend Member
  - Friend Class
- Class Template

Class Template Declaration

```
template <class T>
class Box {
public:
   Box (T initial) : value(initial)  { }
   T getValue( ) { return value; }
   void setValue (T newValue);
private:
   T value;
};
template <class T>
void Box<T>:: setValue (T newValue)
{ value = newValue; }
```

Create an Instance

```
template <class T>
class Box {
public:
    Box (T initial) : value(initial)
   {       }
    T getValue( )
   { return value; }
    void setValue (T newValue)
    { value = newValue; }
private:
    T value;
};
```

```
Box<int> iBox(7);
cout << iBox.getValue( );
iBox.setValue(12);
cout << iBox.getValue( );
```

7

12

Can Be Filled with Different Arguments

```
Box<int> iBox(7);
cout << iBox.getValue( );
iBox.setValue(12);
cout << iBox.getValue( );
iBox.setValue(3.1415); // ERROR - invalid type
Box<double> dBox(2.7);
cout << dBox.getValue( );
dBox.setValue(3.1415);
cout << dBox.getValue( );
iBox = dBox; // ERROR - mismatched types
```

Multiple Type Arguments

```
template <class T1, class T2>
class MTclass{
public:
    MTclass(T1 a, T2 b)
    {i=a;j=b;}
    void show( )
    {cout<<“i=”<<i<<“j=”<<j<<endl;}
private:
    T1 i;
    T2 j;
}
```

```
MTclass<int, double> obj1(2, 3.14159);
MTclass<char, char *> obj2(‘x’, “This is a test”);
```