【C++】Day27

Class	C++
Date	@December 29, 2021
Material	
Series Number	
Summary	static Class Member

【Ch7】Classes

7.5.6 Literal Classes

The parameters and return type of a `constexpr` function must be literal types.

Classes that are literal types may have function members that are `constexpr`. Such members must meet all the requirements of a `constexpr` function. These member functions are implicitly const.

An aggregate class whose data members are all of literal type is a literal class.

A nonaggregate class, that meets the following restrictions, is also a literal class

- The data members all must have literal type.
- The class must have at least one constexpr constructor
- If a data member has an in-class initializer, the initializer for a member of buit-in type must be a constant expression, or if the member has class type, the initializer must use the member's own constexpr constructor
- The class must use default definition for its destructor, which is the member that destroys objects of the class type

7.6 static Class Members

Declaring static Members

We say a member is associated with the class by adding the keyword `static` to its declaration. Like any other member, static members can be public or private.

See the following code for an example:

```
class Account {
public:
  void calculate() {
    amount += amount * interestRate;
  }
  static double rate() { return interestRate; }
  static void rate(double);

private:
  string owner;
  double amount;
  static double interestRate;
  static double initRate();
};
```

The `static` members of a class exist outside any object. Objects do not contain data associated with static data members.

Thus, each Account object will contain two members, `owner` and `amount`. There is only one `interestRate` object that will be shared by all the `Account` objects.

Using a Class static Member

We can access a static member directly through the scope operator:

```
double r;
r = Account::rate(); //acccess a static member using the scope operator
```

Even though static members are not part of the object of its class, we can use object, reference, or pointer of the class type to access a static member:

```
Account ac1;
Account *ac2 = &ac1;
//equivalent ways to call the static member rate function
```

```
r = ac1.rate(); //through an Account object or reference
r = ac2->rate(); //through a pointer to an Account object
```

Note: As with any class member, when we refer to a class static member outside the class body, we must specify the class in which the member is defined. The static keyword, however, is used only on the declaration inside the class body.

We must define and initialize each static data member outside the class body.

We define a static data member similarly to how we define class member functions outside the class. We name the object's type, followed by the name of the class, the scope operator, and the member's own name:

```
//define and initialize a static class member
double Account::interestRate = initRate();
```

Tip: The best way to ensure that the object is defined exactly once is to put the definition of static data members in the same file that contains the definitions of the class noninline member functions.

In-Class Initialization of static Data Members

We can provide in-class initializers for static members that have const integral type and must do so for static members that are `constexprs` of literal type. The initializers must be constant expressions.

```
class Account {
//same as before
private:
  static constexpr int period = 30; //period is a constant expression
  double daily_tbl[period];
};
```

If an initializer is provided inside the class, the member's definition must not specify an initial value:

```
//definition of a static member with no initializers
constexpr int Account::period; //initializer provided in the class definition
```

Best Practices: Even if a const static data member is initialized in the class body, that member ordinarily should be defined outside the class definition

static Members Can Be Used in Ways Ordinary Members Can't

A static data member can have the same type as the class type of which it is a member:

```
class Bar {
public:
  //...
private:
  static Bar mem1; //ok: static member can have incomplete type
  Bar *mem2; //ok: pointer mem can have incomplete type
  Bar mem3; //error: data members must have complete type
};
```

Another difference is that we can use a static member as a default argument:

```
class Screen {
public:
  Screen &clear(char = bkground);
private:
  static const char bkground;
};
```