Chapter 10

Classes: A Deeper Look, Part 2

OBJECTIVES

- To specify **const** (constant) objects and const member functions.
- To create objects **composed** of other objects.
- To use **friend** functions and friend classes.
- To use the **this** pointer.
- To create and destroy objects dynamically with operators **new** and **delete**, respectively.
- To use **static** data members and member functions.

Topics

- **10.1 Introduction**
- 10.2 const (Constant) Objects and const Member Functions
- 10.3 Composition: Objects as Members of Classes
- 10.4 friend Functions and friend Classes
- 10.5 Using the this Pointer
- 10.6 Dynamic Memory Management with Operators new and delete
- 10.7 static Class Members

10.1 Introduction

- const对象, 常成员函数, 常数据成员
- 类的组合(composition): 一个类以其他类的对象作为成员
- 友元(friend)函数: 非成员函数如何访问类的私有成员?
- this指针: 所有非static函数的隐含实参
- 动态内存管理: new和delete运算符
- 静态static的类成员

Topics

- 10.1 Introduction
- **10.2 const (Constant) Objects and const Member Functions**
- 10.3 Composition: Objects as Members of Classes
- 10.4 friend Functions and friend Classes
- 10.5 Using the this Pointer
- 10.6 Dynamic Memory Management with Operators new and delete
- 10.7 static Class Members

10.2 const (Constant) Objects and const Member Functions

```
const Time noon( 12, 0, 0 );
```

- 构造后不能更改数据成员
- 措施: 仅能调用noon对象的const member function (常成员函数)

OBJECT	Member Function	Access
const	const	√
const	non-const	X
non-const	const	√
non-const	non-const	√

10.2 const (Constant) Objects and const Member Functions

- 常成员函数
- • Prototype

  void printUniversal() const;

- • Definition

  void Time::printUniversal() const { … }

- 实现代码要求:
  - • 不能修改本(this)object (数据成员)
  - • 不能调用其它non-const成员函数

10.2 const (Constant) Objects and const Member Functions

```
1.  void Time::test(Time &another) const
2.  {
3.      minute = 20;
4.      // 不能修改object, this->minute=20
5.      printStandard();
6.      // 不能调用non-const成员函数
7.      another.minute = 20;
8.      // OK, 非同一object
9.      another.setHour(6);
10.     // OK
11. }
```

```
1. const Time noon;
2. Time not_noon;
3. noon.test(not_noot);
```

10.2 const (Constant) Objects and const Member Functions

- 注意点:
  - 成员函数是否为常成员函数, 不仅取决于它不修改对象、不调用non-const成员函数, 而且必须显式地声明为const!
  - 构造函数/析构函数不能、也不需要声明为const! 对象的常量特性体现在初始化(构造)后、析构之前.
- 建议:
  - 所有不更改object的成员函数均声明为const成员函数

程序解读（P322）

10.2 const (Constant) Objects and const Member Functions

- 常数据成员

  constructor initializer list

  (构造函数初始化列表)

  Increment::Increment(int i) : increment(i){..}

- 所有类数据成员都可以用构造函数初始化列表进行初始化, 而以下情况只能如此:
  - **const data member**
  - **reference data member 引用类型的数据成员**

```
1.  class Increment
2.  {
3.  public:
4.      Increment( int c = 0, int i = 1 ); // default constructor
5.
6.      // function addIncrement definition
7.      void addIncrement()
8.      {
9.          count += increment;
10.     } // end function addIncrement
11.
12.     void print() const; // prints count and increment
13. private:
14.     int count;
15.     const int increment; // const data member
16.     int &refCount;
17. }; // end class Increment
```

```
1.  // constructor
2.  Increment::Increment( int c, int i )
3.     : count( c ), // initializer for non-const member
4.       increment( i ), // required initializer for const member
5.       refCount ( count ) // required initializer for reference member
6.  {
7.     cout << "Now count = " << count << ", increment = " << increment
8.          << " and refCount= " << refCount<< endl;
9.     refCount = 99;
10. }
11.
12. // print count and increment values
13. void Increment::print() const
14. {
15.    cout << "count = " << count << ", increment = " << increment << endl;
16. }
```

```
Increment value( 10, 5 );
value.print();
```

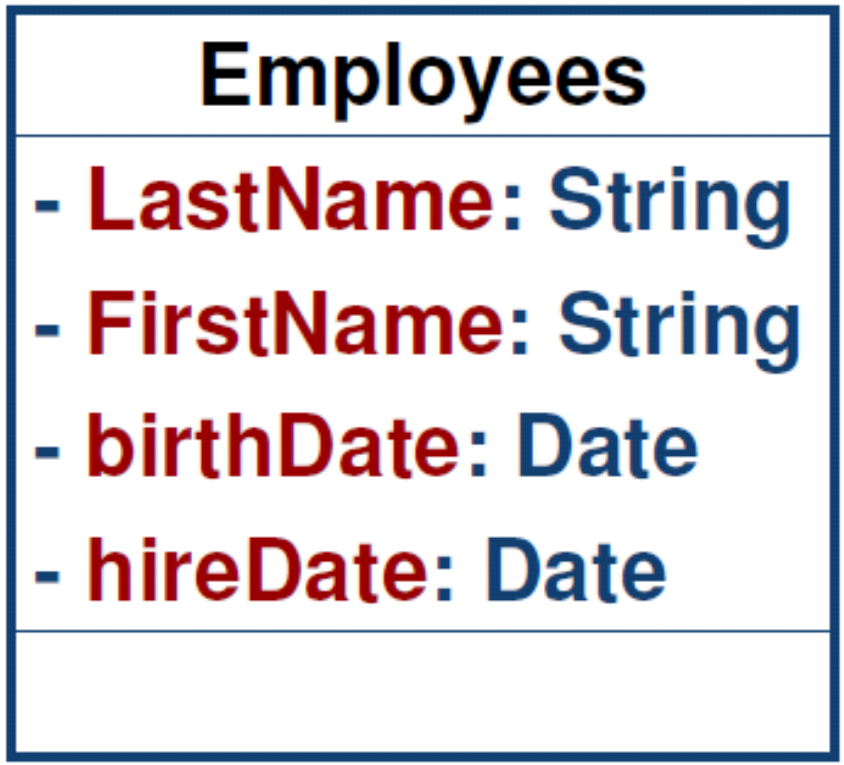
```
Now count = 10, increment = 5 and refCount = 10
 count = 99, increment = 5
```

Topics

- 10.1 Introduction
- 10.2 const (Constant) Objects and const Member Functions
- **10.3 Composition: Objects as Members of Classes**
- 10.4 friend Functions and friend Classes
- 10.5 Using the this Pointer
- 10.6 Dynamic Memory Management with Operators new and delete
- 10.7 static Class Members

10.3 Composition: Objects as Members of Classes

- **Composition (组合, *has-a*): A class has objects of other class as members. (vs *is-a*)**
- **Host Object (宿主对象) vs Contained Object (被包含对象)**



10.3 Composition: Objects as Members of Classes

```
// P72 fig03_05.cpp
15  class GradeBook
16  {

19      void setCourseName ( string name )
20      {
21          courseName = name;
22      }

38  private:
39      string courseName;
40  };
```

osit ers

```
#include <iostream>
using namespace std;
class Test{
public:
    Test( int a ){ num = a; }
private:
    int num;
};
class Test2{
public:
    Test2( int a, int b ) { num = b; }
private:
    Test t;
    int num;
};
int main()
{
    Test2 test( 10, 20 );
    return 0;
}
```

error C2512: 'Test' : no appropriate default constructor available

```
#include <iostream>
using namespace std;
class Test{
public:
    Test( int a ){ num = a; }
private:
    int num;
};
class Test2{
public:
    Test2( int a, int b ): t(a) { num = b; }
private:
    Test t;
    int num;
};
int main()
{
    Test2 test( 10, 20 );
    return 0;
}
```

使用构造函数初始化列表，对类成员t进行初始化！

10.3 Composition: Objects as Members of Classes

- 结论:
- 成员对象若有缺省构造函数, 则可不使用初始化列表, 随后可通过public的set函数对成员对象进行赋值
- 若无缺省构造函数, 则必须使用初始化列表

- 如果成员对象没有显式通过成员初始化列表中初始化, 则自动隐含调用其缺省构造函数 (default constructor).

10.3 Composition: Objects as Members of Classes

- 成员对象的构造和析构顺序(Employee类 manager对象包含Date类的birthDate和hireDate)
- 成员对象的构造先于宿主对象

  构造: birthDate和hireDate manager
- 成员对象之间按照类定义中的声明顺序构造 (**not** in the order they are listed in the constructor's member **initializer list**)

  构造: birthDate hireDate
- 成员对象的析构后于宿主对象

  析构: manager hireDate birthDate

10.3 Composition: Objects as Members of Classes

main栈区

```
birth构造: Date object constructor for date 7/24/1949
Hire构造: Date object constructor for date 3/12/1988
birthDate缺省拷贝构造: 无输出
hireDate缺省拷贝构造: 无输出
manager构造: Employee object constructor: Bob Blue
lastDayOff构造: Date object constructor for date 1/1/1994
lastDayOff析构: Date object destructor for date 1/1/1994
manager析构: Employee object destructor: Blue, Bob
hireDate析构: Date object destructor for date 3/12/1988
birthDate析构: Date object destructor for date 7/24/1949
hire析构: Date object destructor for date 3/12/1988
birth析构: Date object destructor for date 7/24/1949
```

```
int main()
{    Date birth(7,24,1949);
     Date hire(3,12,1988);
     Employee manager("Bob","Blue",birth,hire);
     Date lastDayOff(1,1,1994);
}
```

程序解读（P326）

Topics

- 10.1 Introduction
- 10.2 const (Constant) Objects and const Member Functions
- 10.3 Composition: Objects as Members of Classes
- **10.4 friend Functions and friend Classes**
- 10.5 Using the this Pointer
- 10.6 Dynamic Memory Management with Operators new and delete
- 10.7 static Class Members

10.4 friend Functions and friend Classes

- 类外部的函数和类, 如何访问该类的非公有成员?
- **Standalone functions or entire classes may be declared to be friends of another class, and hence has the right to access the nonpublic (and public) members of the class.**

- **Friend Function(友元函数) & Friend Class(友元类)**
- 意义: 提高性能
- 典型应用: operator overloading(运算符重载)

10.4 friend Functions and friend Classes

- To declare a function as a friend of a class, precede the function prototype in the class definition with keyword friend.

```
class Count
{
  friend void setX( Count &, int );
public:
  Count() : x(0) // initialize x to 0
  {
     // empty body
  }

  void print() const { cout << x << endl; }
private:
  int x;
};
void setX( Count &c, int val ) { c.x = val; }
```

Count授权setX为其友元

私有

```
int main()
{
   Count counter;
   counter.print();

   setX( counter, 8 );
   counter.print();

   return 0;
}
```

```
0
8
```

10.4 friend Functions and friend Classes

- To declare ① all member functions of class ClassTwo as friends of class ClassOne
- ② place a declaration of the form

  friend class ClassTwo;
- in the definition of class ClassOne.

10.4 friend Functions and friend Classes

```
class ClassOne
{
   friend class ClassTwo;
   int x, y;
};
```

ClassOne授权ClassTwo为其友元

```
class ClassTwo{
public:
   void setX(ClassOne &one, int x){ one.x = x; }
   void setY(ClassOne &one, int y){ one.y = y; }
   void printClassOne(ClassOne &one){
      cout << "ClassOne.x = " << one.x
           << ", ClassOne.y = " << one.y << endl;
   }
};
```

```
int main()
{
   ClassOne one;
   ClassTwo two;
   two.setX( one, 10 );
   two.setY( one, 20 );
   two.printClassOne( one );

   return 0;
}
```

Topics

- 10.1 Introduction
- 10.2 const (Constant) Objects and const Member Functions
- 10.3 Composition: Objects as Members of Classes
- 10.4 friend Functions and friend Classes
- **10.5 Using the this Pointer**
- 10.6 Dynamic Memory Management with Operators new and delete
- 10.7 static Class Members

10.5 Using the this Pointer

Test类成员函数

void Test::printX( )

void Test::printY( ) const

test1.printX()

printX函数如何知道应访问哪个内存空间的x?

test1

int x, y

test2

int x, y

10.5 Using the this Pointer

- Implicitly隐性this指针调用
  - x
- Explicitly显式调用
  - this->x
  - (*this).x

程序解读（P332）

10.5 Using the this Pointer

- Cascaded Function Calls(级联函数调用)

```
30.  Time& Time::setHour( int h ) // note Time & return
31.  {
32.      hour = ( h >= 0 && h < 24 ) ? h : 0; // validate hour
33.      return *this; // enables cascading
34.  } // end function setHour
```

```
1.  t.setHour(18).setMinute(30).setSecond(22);
2.  t.setMinute(30).setSecond(22);
3.  t.setSecond(22);
```

程序解读（P334）

Topics

- 10.1 Introduction
- 10.2 const (Constant) Objects and const Member Functions
- 10.3 Composition: Objects as Members of Classes
- 10.4 friend Functions and friend Classes
- 10.5 Using the this Pointer
- **10.6 Dynamic Memory Management with Operators new and delete**
- 10.7 static Class Members

10.6 Dynamic Memory Management with Operators new and delete

- **Motivation**
  - char sentence[1000];
  - Fixed-size array.
  - 1000?
- **Dynamic memory management**
  - 根据需求分配(allocate)/释放(deallocate)内存
  - new / delete operator

10.6 Dynamic Memory Management with Operators new and delete

- use the new operator to dynamically allocate the exact amount of memory required at execution time in the free store (sometimes called the heap堆)
- return the memory to the free store by using the delete operator to deallocate (i.e., release) the memory, which can then be reused by future new operations
- memory leak (内存泄露)

10.6 Dynamic Memory Management with Operators new and delete

- (1) 基本数据类型

```
1.  double *ptr = new double(3.14159);
2.  cout << ptr << endl;
3.  cout << *ptr << endl;
4.  delete ptr;
5.  cout << ptr << endl;
6.  ptr = 0;
```

10.6 Dynamic Memory Management with Operators new and delete

- (2) 类对象

```
1.  class Time{
2.  public:
3.      Time(){ cout << "Time constructor called.\n"; }
4.      ~Time(){ cout << "Time destructor called.\n"; }
5.  };
6.  int main()
7.  {
8.      Time *timePtr = new Time;
9.      delete timePtr;
10.     return 0;
11. }
```

```
Time constructor called.
Time destructor called.
```

10.6 Dynamic Memory Management with Operators new and delete

- new 对象:
  - allocates storage of the proper size for an object
  - calls the default constructor to initialize the object
  - returns a pointer of the type specified to the right of the new operator
- delete 对象:
  - calls the destructor for the object to which pointer points
  - deallocates the memory associated with the object

10.6 Dynamic Memory Management with Operators new and delete

- (2) 类对象

```
1.  class Time2{
2.  public:
3.      Time2( int, int, int);
4.      ~Time2();
5.  };
6.  int main()
7.  {
8.      Time2 *timePtr = new Time2( 12, 45, 0 );
9.      delete timePtr;
10.     return 0;
11. }
```

构造函数参数列表

10.6 Dynamic Memory Management with Operators new and delete

- (3) 数组– 基本数据类型

```
1. int size = 10;
2. int *gradesArray = new int[ size ];
3. delete [ ] gradesArray;
```

- 注意与Fixed size数组的区别:
- *Constant integral expression* vs *Any integral expression*

10.6 Dynamic Memory Management with Operators new and delete

- (3) 数组– 类对象(有缺省构造函数)

```
1. Time *timePtr = new Time[5];
2. delete [ ] timePtr;
```

```
Time constructor called.
Time constructor called.
Time constructor called.
Time constructor called.
Time constructor called.
Time destructor called.
Time destructor called.
Time destructor called.
Time destructor called.
Time destructor called.
```

10.6 Dynamic Memory Management with Operators new and delete

- (3) 数组– 类对象(有缺省构造函数)

```
1. Time *timePtr = new Time[5];
2. delete timePtr;
```

```
Time constructor called.
Time constructor called.
Time constructor called.
Time constructor called.
Time constructor called.
Time destructor called.
```

10.6 Dynamic Memory Management with Operators new and delete

- (3) 数组– 类对象(无缺省构造函数)
- when allocating an array of objects dynamically, the programmer cannot pass arguments to each object's constructor.
- 无法调用带参数的构造函数!

Topics

- 10.1 Introduction
- 10.2 const (Constant) Objects and const Member Functions
- 10.3 Composition: Objects as Members of Classes
- 10.4 friend Functions and friend Classes
- 10.5 Using the this Pointer
- 10.6 Dynamic Memory Management with Operators new and delete
- **10.7 static Class Members**

10.7 static Class Members

- 需求: 火星人类（MARS类），每个火星人都希望知道现在有多少火星人。如何设计数据成员count?

- 特点: 所有火星人共享同一个数据成员
  - 静态数据成员count

- static数据成员
- static成员函数

10.7 static Class Members

- 在类定义中, 用static关键词修饰数据成员和成员函数的声明:
- • Static data member, 静态数据成员

  又称类变量, 由所有对象共享

- • Static member function, 静态成员函数

  可通过类名直接调用, 无this指针

10.7 static Class Members

- 在类定义中声明, 在类定义外定义和初始化(特例)

```
class Employee{
    static int count;
};
int Employee::count = 0;
```

- 至多初始化一次, 若没有显式初始化, 则
  - 基本数据类型: 缺省初始化为0
  - 抽象数据类型(类对象): 默认调用缺省构造函数

10.7 static Class Members

7.7 Case Study: Class GradeBook Using an Array to Store Grades

数据成员: 普通, const, static, static const

① 普通数据成员

- 在类定义中只能声明, 不能初始化
- 可在构造函数中赋值

② const数据成员

- 在对象整个生存期中都不能改变, 在类定义中只能声明, 不能初始化
- Conflict: const数据必须初始化
- 必须在构造函数初始化列表中初始化

10.7 static Class Members

7.7 Case Study: Class GradeBook Using an Array to Store Grades

数据成员：普通, const, static, static const

③ static数据成员

- 意义：在类的所有对象之间共享，称为class variable(类变量)
- public类变量可以直接通过类名+::访问

- 在类定义中只能声明，不能初始化
- 在类外部给出定义和初始化

10.7 static Class Members

7.7 Case Study: Class GradeBook Using an Array to Store Grades

数据成员: 普通, const, static, static const

④ static const数据成员

- 特殊的static数据成员, 一般在类定义中只能声明、不能初始化, 须在类外部给出定义和初始化
- 但如果是整数数据成员(integral), 则可在类定义中声明并初始化(ISO C++)

程序解读（Textbook P258 7.16-18）

10.7 static Class Members

```
#include <iostream>
using namespace std;
class Obj
{
public:
    Obj( int n ){ num = n; }
    static void print() {
        cout << num << endl;
    }
private:
    int num;
};
int main( )
{
    Obj::print();
    return 0;
}
```

- 在类定义中、函数返回类型前，加**static**
- 可通过对象访问，也可直接通过类名+::访问
- 没有**this**指针，不能访问非静态数据成员，也不能调用非静态成员函数

```
error C2597: illegal reference
to non-static member 'Obj::num'
```

程序解读（P338）

10.7 static Class Members

	static数据成员	non-static数据成员
static成员函数	/	X
non-static成员函数	/	/

Summary

- 常量对象和常成员函数
- 对象的组合
- 友元函数和友元类
- this指针
- 动态内存分配
- 静态类成员

Homework

- 实验必选题目(交实验报告):

  10.6 – 9

- 实验任选题目(不交实验报告):

- 作业题目(Homework):