CS 1502H 程序设计思想与方法 (C++)
Chapter 10. 创建新的类型 (2)

陈东尧
上海交通大学
2025年秋季

回顾：将函数放入结构体的意义

- C中，结构体将一组相关的数据捆绑起来，程序逻辑更加清晰。
- 函数加入结构体，它既描述属性，也描述对属性的操作。和内置类型一样的新的数据类型。
- **C++用类（class）这个概念表示**

从C到C++的根本改变

回顾：公有和私有成员

- 私有成员(private)：只能由类的成员函数调用
- 公有成员(public)：外部函数可以调用，类对外的接口
- 私有成员被封装在一个类中，类的用户是看不见的

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- **对象的使用**
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

对象的定义

类与对象的关系：类型与变量的关系

对象的定义

类与对象的关系：类型与变量的关系

对象定义

直接在程序中定义某个类的对象

存储类别 类名 对象列表；

如：DoubleArray arr1，arr2， *ap;

static DoubleArray array[10];

申请动态对象

Rational *rp;

rp = new Rational;

rp = new Rational[20];

delete rp；或 delete [] rp；

对象的使用

对象的使用是使用它的成员

公有成员可以**被任何函数使用**

私有成员**只能被自己的成员函数使用**

成员的引用

对象名.数据成员名

对象指针->数据成员名

对象名.成员函数名（实际参数表）

对象指针->成员函数名（实际参数表）

如 arr1.storage

rp->num

arr1.insert(5， 3.7)

rp->add(r1, r2)

有理数类的使用

计算两个有理数的和与积

```
#include <iostream>
using namespace std;
#include "Rational.h" //使用有理数类
int main()
{       int n, d;
        Rational r1, r2, r3; //定义三个有理数类的对象

        cout << "请输入第一个有理数（分子和分母）：";
        cin >> n >> d;
        r1.create(n,d);
```

有理数类的使用

```
cout << "请输入第二个有理数（分子和分母）：";
cin >> n >> d;
r2.create(n,d);

r3.add(r1, r2);                    //执行r3=r1+r2
r1.display();
cout << " + ";
r2.display();
cout << " = ";
r3.display();     cout<<endl;

r3.multi(r1, r2);                  //执行r3=r1*r2
r1.display();
cout << " * "; r2.display();
cout << " = "; r3.display(); cout << endl;
return 0;  }
```

有理数类的使用

```
 cout << "请输入第二个有理数（分子和分母）：";
 cin >> n >> d;
r2.create(n,d);

 r3.add(r1, r2);                  //执行r3=r1+r2
r1.display();
cout << " + ";
r2.display();
 cout << " = ";
r3.display();       cout<<endl;

r3.multi(r1, r2);                //执行r3=r1*r2
r1.display(); ();
cout << " * "; r2.display();
 cout << " = "; r3.display(); cout << endl;
 return 0;  }
```

运行过程

请输入第一个有理数（分子和分母）：1 6

请输入第二个有理数（分子和分母）：1 6

1 / 6 + 1 / 6 = 1 / 3

1 / 6 * 1 / 6 = 1 / 36

对象赋值语句

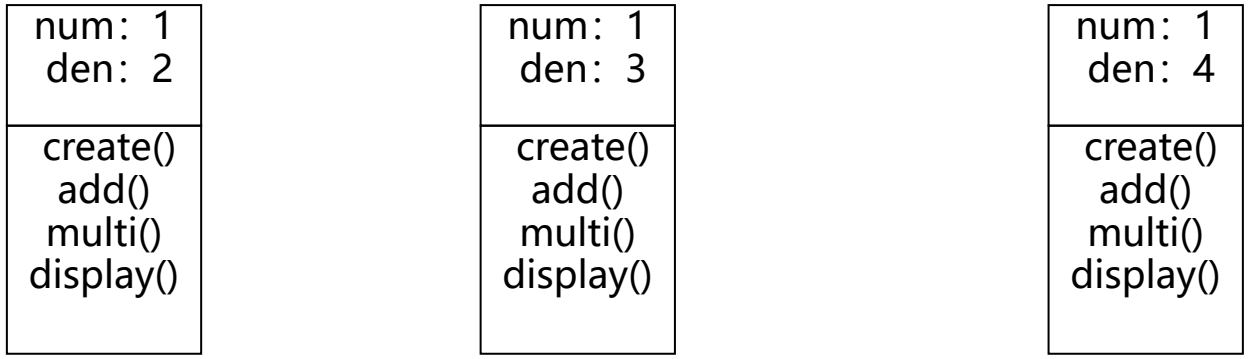
- 同类型的对象之间可以互相赋值
  - 如两个有理数对象r1和r2，可以执行: r1 = r2;
- 当一个对象赋值给另一个对象时，所有的数据成员都会逐位拷贝。上述赋值相当于执行

  r1.num = r2.num

  r1.den = r2.den

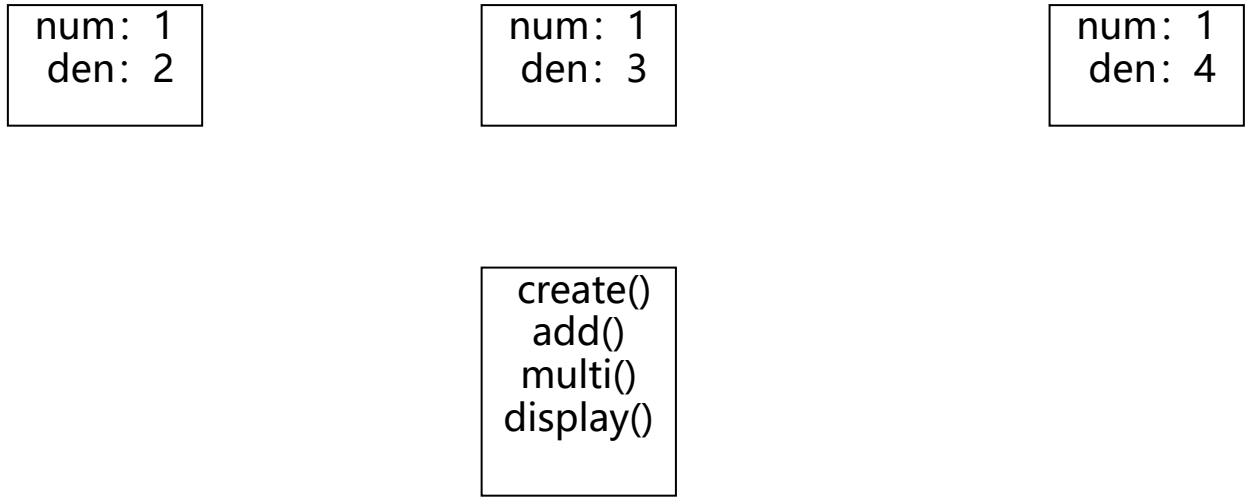
思考一个问题

•逻辑上，对象包括数据成员和成员函数。例如，定义了3个有理数类的对象r1、r2和r3，它们的值分别1/2、1/3和1/4，内存中应有3块空间



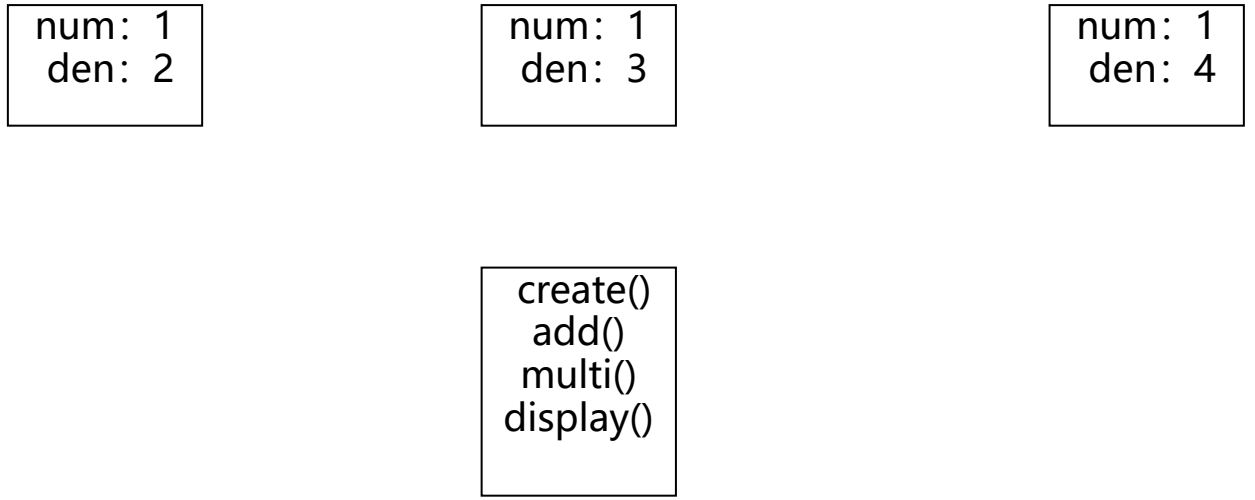
this指针

- 为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



this指针

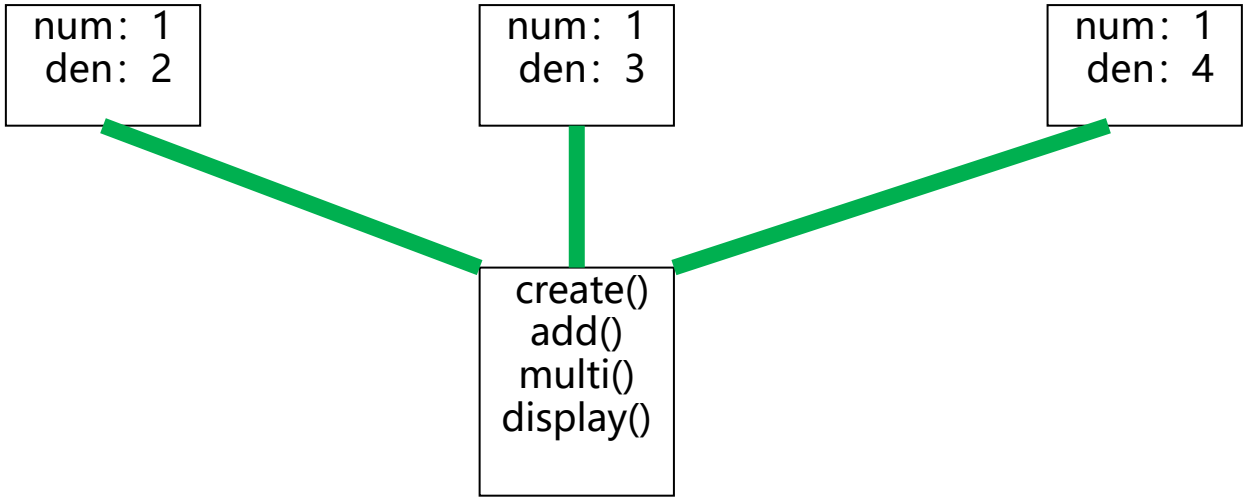
•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



问题：成员函数如何知道他操作的数据成员是哪个对象的数据成员？

this指针

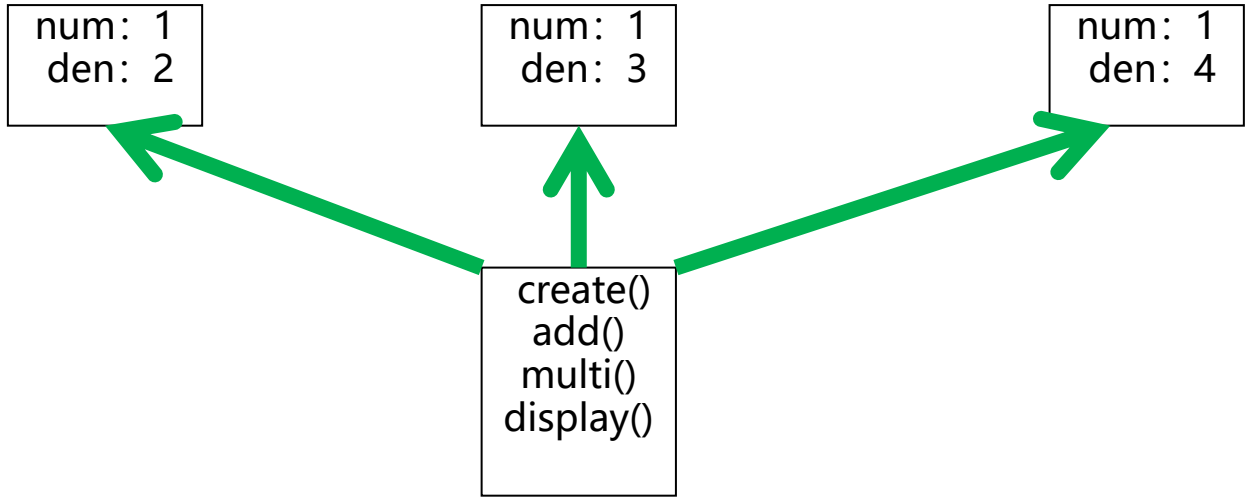
•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



如何实现这种链接？

this指针

•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



this指针

- 每个成员函数都有一个**隐藏的指向本类型的指针形参this**，指向当前调用成员函数的对象
- 成员函数中对象的访问是通过this指针实现的
- 如对函数

```
void create(int n, int d) { num = n; den = d;}
```

经过编译后，实际函数为

```
void create(Rational *this, int n, int d)    { this->num = n; this->den = d;}
```

this指针

•每个成员函数都有一个隐藏的指向本类型的指针形参this，指向当前调用成员函数的对象

•成员函数中对象的访问是通过this指针实现的

•如对函数

```
void create(int n, int d) { num = n; den = d;}
```

经过编译后，实际函数为

```
void create(Rational *this, int n, int d)    { this->num = n; this->den = d;}
```

当通过对象调用成员函数时，编译器会把相应对象的地址传给形参this

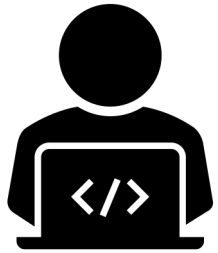
this指针实例

```cpp
#include <iostream>
using namespace std;

class Student {
    int age;
    public:
    Student(int n) {
    // "this->age" refers to the data
    member
    this->age = n;
}

// Function to display age
void display() const {
    cout << "Age: " << this->age << endl;
    }
};

int main() {
    Student s1(20);
    s1.display();
    Student s2(23);
    s2.display();
    return 0;
}
```



代码演示

this_pointer.cpp

注：此处的const代表常量成员函数，即该函数不会改变此类中的成员变量

https://www.geeksforgeeks.org/cpp/const-member-functions-c/

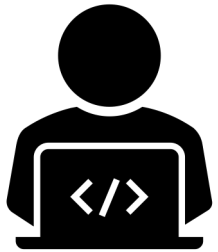
```cpp
class Student {
    private:
        string name;
        int age;
    public:
        // Constructor with parameters having the same name
        Student(string name, int age) {
            // 'this' is used to distinguish between class
            members and constructor parameters
            this->name = name;
            this->age = age;
        }

        // Member function using 'this'
        void introduce() {
            cout << "Hi, I am " << this->name
            << " and I am " << this->age << " years old." <<
            endl;
        }

        // Returning current object using 'this'
        Student& olderBy(int years) {
            this->age += years;
            return *this;
    }
};

int main() {

    Student s("Alice", 20);
    // Display initial introduction
    s.introduce();
    // Increase age and reintroduce
    s.olderBy(2).introduce();
    return 0;
}
```



代码演示

this_pointer-advanced.cpp

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- **对象的构造与析构**
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

对象中的专用函数：构造与析构（constructor and destructor）

- 变量可以在定义时赋初值，对象能赋初值吗？如何赋值？

将赋初值的过程写成一个函数，让计算机在定义对象时执行该函数

这个函数称为**构造函数**

同理，有时在对象消亡（回收）前也有些工作要做，可将该过程写成一个函数，让计算机在对象消亡时执行该函数

这个函数称为**析构函数**

构造函数和析构函数是特殊的成员函数

对象中的专用函数：构造与析构（constructor and destructor）

- 类的对象，需对它初始化后才能使用。
- 对象在消亡前，常需善后扫尾处理（例如delete动态变量）
- 初始化和扫尾给用户带来额的负担

希望定义后，能直接使用，用完时系统可以自动回收

对象中的专用函数：构造与析构（constructor and destructor）

- 构造函数和析构函数是特殊的成员函数
- 构造函数：对数据成员进行初始化，

在对象诞生(initialize)时自动运行。

- 析构函数：清理工作，如释放分配给对象的动态空间。

在对象释放(release)时自动运行。

【扩展阅读】Microsoft关于析构函数的文档
https://docs.microsoft.com/zh-cn/cpp/cpp/destructors-cpp?view=msvc-170

构造函数的特点

1. 定义对象时，系统自动调用构造函数
2. 构造函数的**名字与类名相同，这是它的标志**
3. 构造函数可以有任意类型的参数，也可以不带参数
4. **但不能具有返回类型**
5. 如果没定义构造函数，系统会自动生成一个缺省（default）的构造函数，它没有参数
6. 构造函数可以重载

构造函数实例

- 如DoubleArray类需要有一个构造函数

```
DoubleArray(int lh, int rh)
{    low = lh; high = rh;
     storage = new double [high – low + 1];
}
```

用它替代initialize函数更好，**定义时自动调用**

- 定义对象时，**须指定构造函数的实参**

```
DoubleArray  array(20, 30);
```

构造函数实例

- Rational类构造函数可定义为

  Rational(int n1, int n2)

  { num = n1; den = n2; ReductFraction();}

- 定义对象时，指定实参。例 Rational r(2, 7);

构造函数的使用

- 有了构造函数后，对象定义：

  类名 对象名（实参表）；

- 如果类有一个构造函数是**没有参数**的，可用：

  类名 对象名;

带缺省值的Rational类的构造函数

- Rational(int n1 = 0, int n2 = 1)

```
{  num = n1; den = n2;
   ReductFraction();
}
```

表示缺省情况下，构造的是一个值为0的有理数。此时，定义

```
Rational  r1(3,5), r2; //r1声明正确，  r2声明也是正确的
```

带缺省值的Rational类的构造函数

- Rational(int n1, int n2)

```
{  num = n1; den = n2;
   ReductFraction();
}
```

```
Rational  r1(3,5), r2; //r1声明正确,  r2声明错误
```

原因在于：一旦有自定义构造函数，系统缺省构造函数消失

动态对象的初始化

- 动态变量初始化在类型后面用圆括号指出实参表，如

  p = new DoubleArray(20, 30);

构造函数另一种调用方式：初始化列表 (initializer list)

- 构造函数和普通函数不同之处：可含构造函数**初始化列表**
- 什么是初始化列表：居函数头和函数体之间，**冒号开头，逗号分隔数据成员**

如DoubleArray的构造函数可写为

```
DoubleArray(int lh, int rh) : low(lh), high(rh)
 { storage = new double [high - low + 1]; }
```

用初始化列表的原因

- **可以在函数体内对数据成员赋初值**
- **提高构造对象的效率**
  - 执行每一个数据成员的构造函数。如果成员没有出现在初始化列表中，执行默认的构造函数，否则按初始化列表中列出的实际参数执行对应的构造函数
- **特殊数据成员的初始化**
  - 如果类包含了一个常量的数据成员，常量只能在定义时对它初始化，而不能对它赋值。因此也必须放在初始化列表中

Reference: https://www.geeksforgeeks.org/cpp/when-do-we-use-initializer-list-in-c/

用初始化列表的原因

- **特殊数据成员的初始化**
  - 如果类包含了一个常量的数据成员，常量只能在定义时对它初始化，而不能对它赋值。因此必须放在初始化列表中

```
class Test {
    const int t;
    public:
    //Initializer list must be used
    Test(int n):t(n) {}
    int getT() { return t; }
};

int main() {
    Test t1(10);
    cout<<t1.getT();
    return 0;
}
```

Reference: https://www.geeksforgeeks.org/cpp/when-do-we-use-initializer-list-in-c/

构造函数可以有多个吗?

- 构造函数可以重载（overload），因此对象可以有多种方式构造
- 试设计一个时间格式转换，用户可输入秒、分秒或时分秒输出相应的秒数

时间格式转换的实现

```
#include <iostream>
using namespace std;
class timer{
        int second;
  public:
        timer(int t){ second = t; }
        timer(int min, int sec)  { second = 60 * min + sec;}
        timer(int h, int min, int sec)  {second=sec+60*min+3600*h;}
        int getTime()    { return second; }
};
void main()
{       timer a(20), b(1, 20), c(1, 1, 10);
        cout << a.getTime() << endl;
        cout << b.getTime() << endl;
        cout << c.getTime() << endl;     }
```

```
int main(){
    People p1(18);
    cout<<"p1's age"<<p1.getAge()<<endl;
    People p2(p1);
    cout<<"p2's age"<<p2.getAge()<<endl;


    return 0;
}
```

拷贝构造（复制构造，copy constructor）

拷贝构造函数

- 创建对象时，可用一个同类的对象对其初始化。需用一个特殊的构造函数---拷贝（复制）构造函数。
- 拷贝构造函数以同类对象引用作为参数：

  类名(const <类名> &);

- 用户可据自己的需要定义拷贝构造函数

缺省的拷贝构造函数

- 如果用户未定义拷贝构造函数，系统会定义一个缺省的拷贝构造函数。该函数将已存在的对象复制给新成员

何时需要自定义拷贝构造函数?

- 一般情况下，默认的拷贝构造函数足以满足要求，但需要结合具体使用场景

何时需要自定义拷贝构造函数?

- 一般情况下，默认的拷贝构造函数足以满足要求，但需要结合具体使用场景

  例如：希望对DoubleArray类能定义一个和另一个数组完全一样的数组

  DoubleArray arr2(10,30)，arr1(arr2);

- 默认拷贝构造函数执行：

  arr1.low = arr2.low;　arr1.high = arr2.high;　arr1.storage = arr2.storage;

何时需要自定义拷贝构造函数?

- 一般情况下，默认的拷贝构造函数足以满足要求，但需要结合具体使用场景。

  例如：希望对DoubleArray类能定义一个和另一个数组完全一样的数组。

  DoubleArray arr2(10,30)，arr1(arr2);

- 默认拷贝构造函数执行：

  arr1.low = arr2.low;　arr1.high = arr2.high;　arr1.storage = arr2.storage;

操作中，storage是一个指针，意味着使arr1的storage指针和arr2的storage指针指向同一块区间！

使用同一块空间存在什么问题?

- 一个对象的修改将会影响另一个对象
- 当一个对象析构时，另一个对象也将丧失它的空间!
  - 例如特别当两个对象的作用域不同时

DoubleArray类的拷贝构造函数

```
DoubleArray(const DoubleArray &arr)
{   low = arr.low;
    high = arr.high;
    storage = new double [high – low + 1];
    for (int i = 0; i < high –low + 1; ++i)
        storage[i] = arr.storage[i];
}
```

拷贝构造的应用场合（重要！）

1. 对象**定义**时
2. 函数**调用**时，把对象作为**值传递**参数传给值传递的形式参数
3. 把对象作为**返回值**时

拷贝构造的应用场合（1）：对象定义时

```
#include <iostream>
using namespace std;

class point{
int x, y;
public:
point(int a, int b) { x = a; y = b; }
point(const point &p) { x = 2 * p.x; y = 2 * p.y; }
void print() { cout << x << " " << y << endl; }
};

int main()
{
point p1(10, 20), p2(p1), p3 = p1, p4(1, 2);
p1.print(); p2.print(); p3.print(); p4.print();
p4 = p1; p4.print();
return 0;
}
```

拷贝构造的应用场合（1）：对象定义时

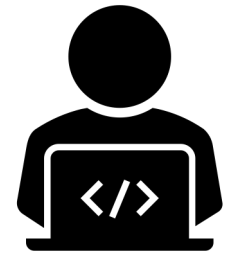
```
#include <iostream>
using namespace std;

class point{
int x, y;
public:
point(int a, int b) { x = a; y = b; }
point(const point &p) { x = 2 * p.x; y = 2 * p.y; }
void print() { cout << x << " " << y << endl; }
};

int main()
{
point p1(10, 20), p2(p1), p3 = p1, p4(1, 2);
p1.print(); p2.print(); p3.print(); p4.print();
p4 = p1; p4.print();
return 0;
}
```

```
10  20
20  40
20  40
1   2
10  20
```

拷贝构造的应用场合（1）：对象定义时



代码演示

copy_constructor_example-1.cpp

```
#include <iostream>
using namespace std;

class point{
int x, y;
public:
point(int a, int b) { x = a; y = b; }
point(const point &p) { x = 2 * p.x; y = 2 * p.y; }
void print() { cout << x << " " << y << endl; }
};

int main()
{
point p1(10, 20), p2(p1), p3 = p1, p4(1, 2);
p1.print(); p2.print(); p3.print(); p4.print();
p4 = p1; p4.print();
return 0;
}
```

通过赋值的定义默认调用拷贝构造函数

10 20

20 40

20 40

1 2

10 20

此处仅是赋值，没有拷贝过程。
因为p4已经初始化完毕了。

拷贝构造的应用场合（2）：对象作为值传递参数

- 如有函数：void f (DoubleArray array);
- 函数调用

  f(arr);

将创建一个形式参数对象array，并调用拷贝构造函数用对象arr初始化array

注意：如果是引用传递就没有这个构造过程了

拷贝构造的应用场合（3）：对象作为返回值时

- 如有函数

```
DoubleArray f()
{
    DoubleArray a;
     …
    return a;
 }
```

析构函数

- 析构函数在撤销对象时，完成善后，编译系统自动调用
- 析构函数和类名同名，**但前加波浪号（~）**
- 析构函数没有参数，没有返回值，故不能重载
- 类没有定义析构函数时，系统自动生成一个缺省的空析构函数

析构函数

- 在构造函数中有动态申请内存的，必须有析构函数，由它释放内存
  - 因此并非每个类都必须要有析构函数，如Rational类就不需要
  - 如DoubleArray类，析构函数释放storage指向的空间

构造、析构函数举例

```
class DoubleArray{
    int low;
    int high;
    double *storage;
public:
    DoubleArray(int lh, int rh):low(lh), high(rh)
        { storage = new double [high - low + 1]; }
    bool insert(int index, double value);
    bool fetch(int index, double &value);
    ~DoubleArray() {delete [] storage; }
};
```

DoubleArray类的使用

```
#include <iostream>
using namespace std;
#include "DoubleArray.h"
int main()
{
    DoubleArray array(20,30);  //自动调用了构造函数
    int i;
    double value;
    for (i=20; i<=30; ++i)
    {
        cout<< "请输入第" << i << "个元素:";  cin >> value;
        array.insert(i, value);
    }
```

DoubleArray类的使用

```
 while (true)
{
    cout << "请输入要查找的元素序号（0表示结束）:";
    cin >> i;
    if (i == 0) break;
    if (array.fetch(i, value)) cout << value << endl;
    else cout << "下标越界\n";
}

return 0;
//自动调用了析构函数
}
```

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

const 对象

const对象

- 定义后值不能被修改的对象

const对象的定义

- const MyClass obj（参数表）；

如何保证数据成员不被修改?

- 对const对象只能调用const成员函数

const成员函数

原型：返回类型　函数名（形式参数表）　const

一旦被定义成const成员函数，编译器会检查是否修改数据成员

- 如：

```
class A {
    int x;
public:
    A(int i) { x = i; }
    int getx() const  {  return x;   }
};
```

const是函数原型的一部分，
在类外定义时也必须加上

```
int A::getx() const
{
        return x;
 }
```

第10章 创建新的类型

- 新的数据类型
- 类（类型）的定义
- 对象的使用
- 对象的构造与析构
- 常量对象与const成员函数
- 常量数据成员
- 静态数据成员与静态成员函数
- 友元
- 面向对象程序设计

从问题（对象）出发，思考需求

- 关于固有变量的思考

从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改?

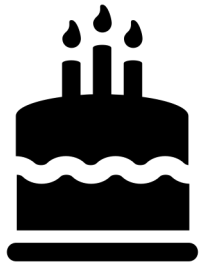
从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改?
  2. 哪些变量对于所属对象是固有的，但是不同对象间又不同

从问题（对象）出发，思考需求

- 关于固有变量的思考
  1. 对象的哪些变量是固有的不能更改?
  2. 哪些变量对于所属对象是固有的，但是不同对象间又不同

一个例子：生日



常量成员

const数据成员

在定义时赋初值，然后在整个对象生存期中值不能被修改

常量成员的声明

在该成员声明前加上const。如

```
class  People {
        const  int birthday ;
         …
};
```

常量成员

const数据成员

在定义时赋初值，然后在整个对象生存期中值不能被修改

常量成员的声明

在该成员声明前加上const。如

```
class  People {
        const  int birthday ;
         …
};
```

同一类的不同对象的const数据成员的值可以不同

（即：一个人的生日不会变，每个人的生日很可能不同）

常量数据成员

- 不能在类声明中初始化const数据成员。应在对象产生时初始化

```
class A
{
  //错误，企图在类声明中初始化const数据成员
  const int SIZE = 200;
};
```

const数据成员的初始化

- 只能在类构造函数的初始化表中进行
- 例：

```
class A
{       A(int size);                //构造函数
        const int SIZE;    }
A::A(int size) : SIZE(size)  //构造函数的初始化表
{…}
A a(100);   //对象a的SIZE的值为100
A b(200);   //对象b的SIZE的值为200
```