CS 1502H 程序设计思想与方法 (C++)
Chapter 10. 创建新的类型 (1)

陈东尧
上海交通大学
2025年秋季

面向对象的程序设计

- 从过程化到面向对象

抽象过程

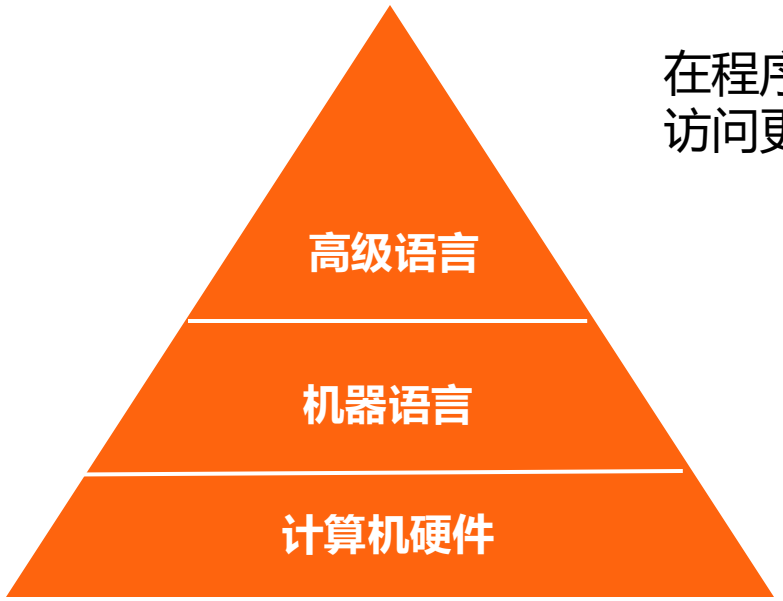
- 计算机的工作是建立在抽象（abstraction）的基础之上

抽象过程

- 计算机的工作是建立在抽象的基础之上

抽象过程

- 计算机的工作是建立在抽象的基础之上



在程序设计中，抽象提供了访问权限，同时隐藏了可能使访问更具挑战性的细节

抽象过程

- 计算机的工作是建立在抽象的基础之上

现有的问题

当程序员要解决一个问题时，必须要在机器模型和**实际要解决的问题**模型之间建立联系。

而计算机的结构本质上还是**为了支持计算**，当要解决一些非计算问题时，这个联系的建立是很困难的

面向对象的程序设计

Object-oriented Programming

Object-oriented Programming

物体/对象　　导向的　　程序设计

Object-oriented Programming

以对象为导向的程序设计

Object-oriented Programming

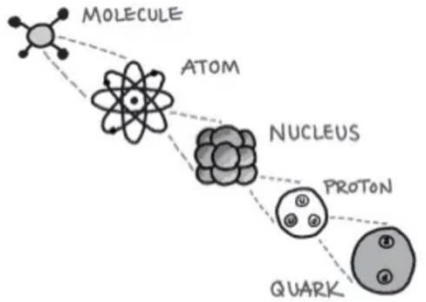
以对象为导向的程序设计

如何解决复杂的大问题?

- 若想要解决这样的问题，一定要

第一性原理思维（First Principle Thinking）

- 将问题分解为不能再分解的基本构件，即基本要素，然后开始自下而上一步一步解决问题



从过程化到面向对象

过程化程序设计

- 将解决问题的过程分解成一个个程序设计语言能提供的功能，把**每个步骤用一个语句代替**
- 设计时必须考虑到每一个细节
- 让人去适应程序设计语言
- 适合解决小问题

VS.

面向对象程序设计

- 面向对象编程的主要目标是将数据及其相关的函数绑定在一起。
- 让工具适应问题
- 大程序的解决方案

对象与变量

- 对象与变量都是程序中处理某一类问题的工具
  - 变量是程序设计语言做好的工具
  - 对象是程序员自己创建的工具

面向对象的程序设计

- **为程序员提供了创建工具的功能**
- **解决问题过程**
  - 首先考虑的是需要哪些工具
  - 创建或收集这些工具
  - 设计用这些工具解决问题的过程

过程化vs面向对象

以计算圆的面积和周长的问题为例

过程化的设计方法：从功能和过程着手

- 输入圆的半径或直径
- 利用$S = \pi r^2$和$C = 2\pi r$计算面积和周长
- 输出计算结果

面向对象的程序设计方法：

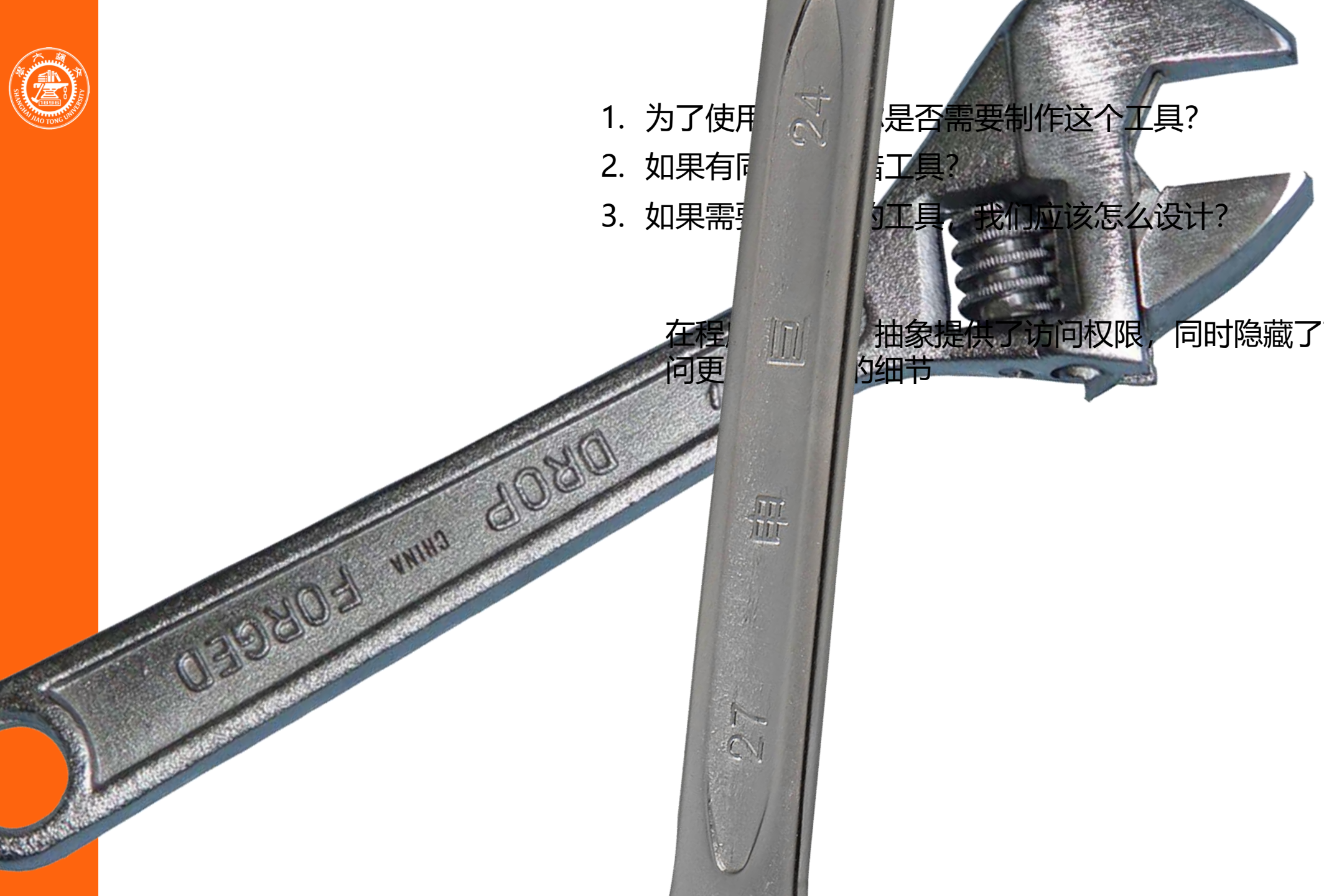
- 需要什么工具。如果计算机能提供给我们一个称为圆的工具，它可以以某种方式保存一个圆，告诉我们有关这个圆的一些特性，如它的半径、直径、面积和周长
- 定义一个圆类型的变量，以他提供的方式将一个圆保存在该变量中，然后让这个变量告诉我们这个圆的面积和周长是多少

面向对象的程序设计的三个特点

1.为了使用工具，你是否需要制作这个工具？ **隐藏**（abstraction：在程序设计中，抽象提供了访问权限，同时隐藏了可能使访问更具挑战性的细节）

2.如果有其他人找你借工具？ **重用**

3.如果需要有更棒的工具，我们应该怎么设计？ **继承**



1. 为了使用[illegible]是否需要制作这个工具?
2. 如果有[illegible]工具?
3. 如果需要[illegible]的工具，我们应该怎么设计?

在程[illegible]抽象提供了访问权限，同时隐藏了可能使访问更[illegible]的细节

面向对象的程序设计的三个特点

- **代码重用**
- **实现隐藏**
- **继承**

面向对象的程序设计的特点：代码重用

- **代码重用**
- 圆类型也可以被那些也需要处理圆的其他程序员使用

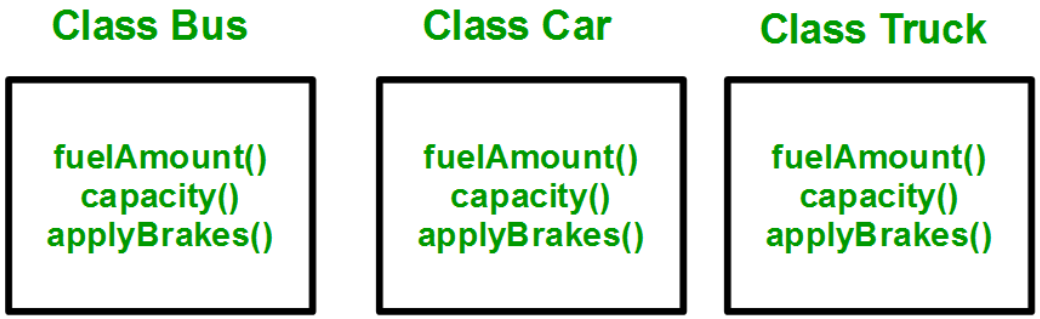
面向对象的程序设计的特点：实现隐藏

- **实现隐藏**
- 类的创建者创造新的工具
- 类的使用者则收集已有的工具快速解决所需解决的问题：这些工具是如何实现的，类的使用者不需要知道

面向对象的程序设计的特点

- **继承**

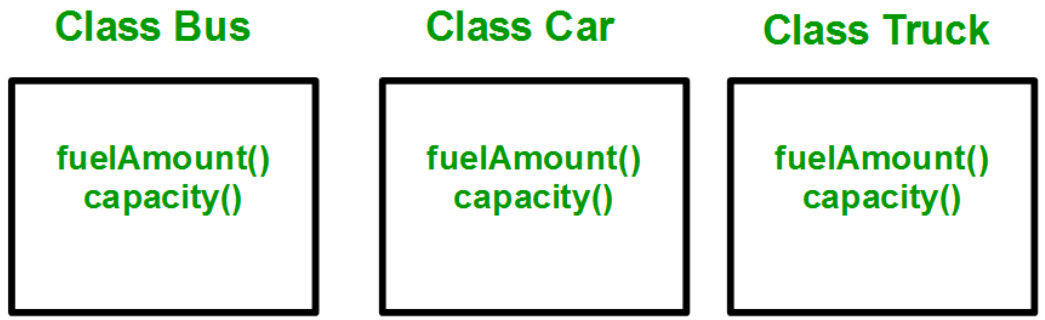
为车辆设计一个类



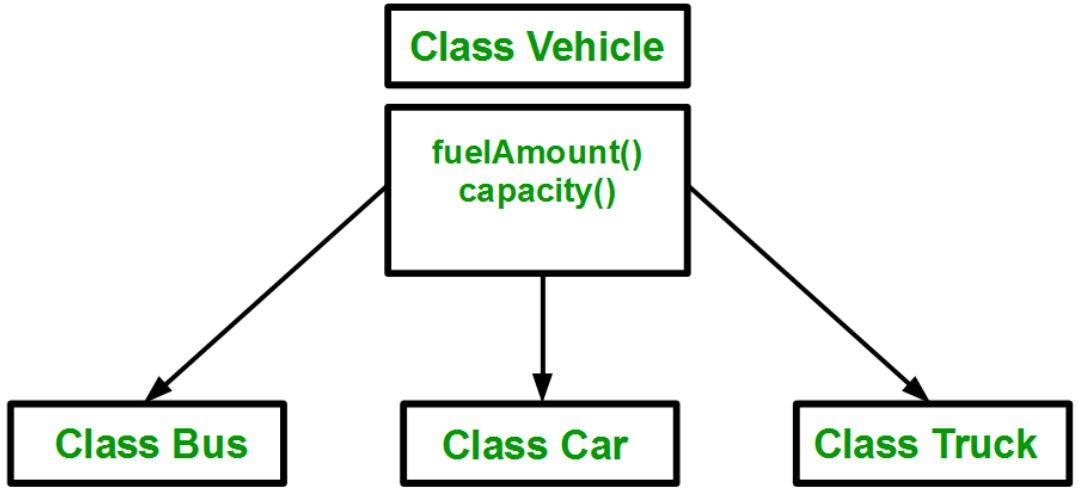
面向对象的程序设计的特点

- **继承**

为车辆设计一个类



一个明显的更优方案



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

如何定义及处理新型数据？

通过结构体扩展新的数据类型

a[0], a[1], …, a[n]

如何定义及处理新型数据？

通过结构体扩展新的数据类型

a[0], a[1], …, a[n]

能否让我来确定下标的范围？而且在使用时确定？它能不能自动帮我检查下标越界？

如何定义及处理新型数据？

通过结构体扩展新的数据类型

- 设计一个动态实型数组类型，该数组满足三个要求：
  1. 可以任意指定下标范围；
  2. 下标范围可在运行时确定；
  3. 使用下标变量时会检查下标的越界。

数组的处理要求

- **数组的保存**
  - 数组需要空间。执行时动态分配。
  - 数组的下标由用户指定，需要保存下标的上下界。
  - 用结构体把它们封装在一起。

- **数组操作**
  - 给数组分配空间
  - 给数组元素赋值
  - 读取某数组元素的值
  - 释放动态空间

3个关键组成部分：

一、类型定义及函数声明 （array.h）

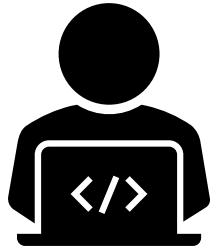
二、函数定义（实现）(array.cpp)

三、类型的使用 (main.cpp)

类型定义放头文件Array中---作为库

```
#ifndef _array_h
#define _array_h

//可指定下标范围的数组的存储
struct DoubleArray
{       int low;
        int high;
        double *storage;
};
```



代码演示

array.h

array.cpp

main.cpp

以普通（外部）函数实现数据的处理

函数声明

```
 //根据low和high为数组分配空间
bool initialize(DoubleArray &arr, int low, int high);
//设置数组元素的值
bool insert(const DoubleArray &arr, int index, double value);
//取数组元素的值
bool fetch(const DoubleArray &arr, int index, double &value);
void cleanup(const DoubleArray &arr); //回收数组空间
#endif
```

Array库的实现文件---函数定义

```
#include "array.h"
#include <iostream>
using namespace std;

bool initialize(DoubleArray &arr, int low, int high)
{  arr.low = low;
   arr.high = high;
   arr.storage = new double [high - low + 1];
   if (arr.storage == nullptr) return false;
       else return true;
}
```

Array库的实现文件---函数定义

```
#include "array.h"
#include <iostream>
using namespace std;

bool initialize(DoubleArray &arr, int low, int high)
{  arr.low = low;
   arr.high = high;
   arr.storage = new double [high - low + 1];
   if (arr.storage == nullptr) return false; // 内存申请失败
       else return true;
}
```

```
bool insert(const DoubleArray &arr, int index, double value)
{  if (index < arr.low || index > arr.high) return false;
   arr.storage[index - arr.low] = value;
   return true;
}
bool fetch(const DoubleArray &arr, int index, double &value)
{  if (index < arr.low || index > arr.high) return false;
   value = arr.storage[index - arr.low] ;
   return true;
}
void cleanup(const DoubleArray &arr)
{    delete [] arr.storage;  }
```

新型数据类型的使用

```
#include <iostream>
using namespace std;
#include "array.h"
int main()
{   DoubleArray array;        double value;     int low, high, i;

    cout << "请输入数组的下标范围：";
    cin >> low >> high;

    if (!initialize(array, low, high))
        {  cout << "空间分配失败" ; return 1; }
```

```
    for (i = low; i <= high; ++i) {
        cout << "请输入第" << i << "个元素：";    cin >> value;
        insert(array, i, value);
    }
    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 (fetch(array, i, value)) cout << value << endl;
        else cout << "下标越界\n";
    }
    cleanup(array);
    return 0;
}
```

Array库的问题

- 调用函数时，要传一个结构体，开销较大
- 许多库都有初始化和清除工作。initialize和cleanup**函数名易冲突**
- 使用结束后，用户需要**显式释放空间**
- 数组元素不能用**下标变量形式表示**

Array库的问题

- 调用函数时，要传一个结构体，开销较大
- 许多库都有初始化和清除工作。initialize和cleanup函数名易冲突 //成员函数、组合与继承
- 使用结束后，用户需要显式释放空间 //构造与析构
- 数组元素不能用下标变量形式表示 //运算符重载

发现了问题，我们就提出解决方案

Array库的改进

函数放入结构体

- 好处：
  - 函数原型中的第一个参数不再需要。编译器来安排
  - 函数从属某一结构体，函数名冲突问题解决。

改进后的Array库的头文件

```
#ifndef _array_h
#define _array_h
struct DoubleArray{
    int low;
    int high;
    double *storage;

    bool initialize(int lh, int rh);
    bool insert(int index, double value);
    bool fetch(int index, double &value);
    void cleanup();
};
#endif
```

改进后的Array库的实现文件

- 与原来的实现有一个变化：函数名前要加限定

```
bool DoubleArray::initialize(int lh, int rh)
{           low = lh;
            high = rh;
            storage = new double [high - low + 1];
            if (storage == NULL) return false;
        else return true;
}
```

VS.

```
bool initialize(DoubleArray &arr, int low, int high)
{  arr.low = low;
   arr.high = high;
   arr.storage = new double [high - low + 1];
   if (arr.storage == NULL) return false;
       else return true;
}
```

改进后的Array库的应用 --- 变化

- 函数的调用方法不同。就如引用结构体的成员一样，要用点运算符引用这些函数

```
#include <iostream>
#include "array.h“

using namespace std;

int main()
{   DoubleArray array;
    double value;
    int low, high, i;

    cout << "请输入数组的下标范围：";
    cin >> low >> high;

    if (!array.initialize(low, high))
          { cout << "空间分配失败" ; return 1;}
```

```
for (i = low; i <= high; ++i) {
        cout << "请输入第" << i << "个元素:";    cin >> value;
        array.insert(i, value);
    }
 while (true) {         //数组元素的查找
        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";
 }
    array.cleanup();  return 0;
}
```

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**

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++的根本改变之一

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

类型（类）定义

- struct 类型名
  {　属性（数据成员、数据）
  　　成员函数
  };
- 成员函数是对属性的各种基本处理，属性通过函数和外界打交道。

类型（类）定义的一般格式

- class 类名
  { private:
    私有数据成员和成员函数
  public:
    公有数据成员和成员函数
  };

类型（类）定义的一般格式

- class 类名

  { private:

  私有数据成员和成员函数

  public:

  公有数据成员和成员函数

  };

private 关键词可以缺省，则默认是私有

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

用struct和class的区别

- 如不加访问权限
  - class，默认私有
  - struct，默认公有

类定义说明

- private 和public出现次序可任意， 出现次数可任意。
- 数据成员一般为private， 函数一般为public的。

在模块化设计中，接口和实现分开

- 与库设计一样，类定义在头文件中，成员函数实现在实现文件中。
- 简单的成员函数的定义可写在类定义中。

```
class A
{ private:
        int x;
   public:
        int get(){return x;};
};
```

```
void  f()
{ A  a;
   a.x = 10;//?
}
```

```
class A
{ private:
        int x;
   public:
        int get(){return x;};
};
```

```
void  f()
{ A  a;
   a.x = 10;//
}
```

那么，如何在外部访问私有成员？

```
class A
{ private:
        int x;
   public:
        int get(){return x;}
        void set(int y){x=y;}
};
```

```
void  f()
{ A  a;
   a.x = 10;//
   a.set(10); //
}
```

解释：操作私有成员需要通过开放的成员函数

```
class A
{ private:
        int x;
   public:
        int get(){return x;}
        void set(int y){x=y;}
};
```

```
void  f()
{ A  a;
   a.x = 10;//
   a.set(10); //
   cout<<a.x<<endl; //?


}
```

```
class A
{ private:
        int x;
   public:
        int get(){return x;}
        void set(int y){x=y;}
};
```

```
void  f()
{ A  a;
   a.x = 10;//
   a.set(10); //
   cout<<a.x<<endl;//
   cout<<a.get()<<endl;//
}
```

通过这些学习，如何提升我们目前的程序设计？

DoubleArray类的定义改进 1：加访问权限控制

```
class DoubleArray {
private:
    int low;
    int high;
    double *storage;
public:
    bool initialize(int lh, int rh);
    bool insert(int index, double value);
    bool fetch(int index, double &value);
    void cleanup();
};
```

类定义又一个实例

试定义一个有理数类，该类能提供有理数的加和乘运算。

要求保存的有理数是最简形式。如2/6应记录为1/3。

设计考虑

- 保存有理数：
- 有理数类的操作：
- 访问权限设计：

设计考虑

- 保存有理数：
  - 保存一个有理数就是保存两个整数。
- 有理数类的操作：
- 访问权限设计：

设计考虑

- 保存有理数：
  - 保存一个有理数就是保存两个整数。
- 有理数类的操作：
  - 运算函数：加函数，乘函数
  - 创建有理数的函数：用以设置有理数的分子和分母
  - 输出有理数函数
  - 化简函数
- 访问权限设计（**哪些应该是私有的？**）：

设计考虑

- 保存有理数：
  - 保存一个有理数就是保存两个整数。
- 有理数类的操作：
  - 运算函数：加函数，乘函数
  - 创建有理数的函数：用以设置有理数的分子和分母
  - 输出有理数函数
  - 化简函数
- 访问权限设计：
  - 数据成员是私有的
  - 化简函数是内部调用的函数，与用户无关，是私有的
  - 其他函数都是公有的

有理数类的定义： Rational.h

```
#ifndef  _rational_h
#define  _rational_h
#include <iostream>
using namespace std;
class Rational {
private:
        int num;
        int den;
        void ReductFraction(); //将有理数化简成最简形式
public:
        void create(int n, int d) { num = n; den = d;}
        void add(const Rational &r1, const Rational &r2);
        void multi(const Rational &r1, const Rational &r2);
        void display() { cout << num << '/' << den;}
};
#endif
```

有理数类的实现---Rational.cpp

```
#include "Rational.h“

//add函数将r1和r2相加，结果存于当前对象
void Rational::add(const Rational &r1, const Rational &r2)
{//注意函数头部顺序结构
 num = r1.num * r2.den + r2.num * r1.den;
 den = r1.den * r2.den;
 ReductFraction();
}
```

有理数类的实现---Rational.cpp

```
void Rational::multi(const Rational &r1, const Rational &r2)
{
        num = r1.num * r2.num;
        den = r1.den * r2.den;
        ReductFraction();
}

// ReductFraction实现有理数的化简
void Rational::ReductFraction()
{       int tmp = (num > den) ? den : num;
        for  (; tmp > 1; --tmp) {
                if (num % tmp == 0 && den % tmp ==0)
                        {num /= tmp; den /= tmp; break;}
        }
}
```

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

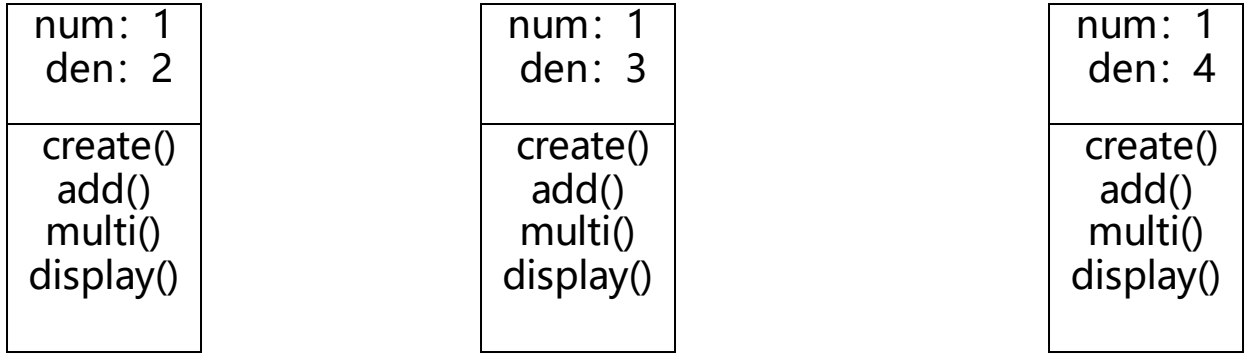
- 同类型的对象之间可以互相赋值
  - 如两个有理数对象r1和r2，可以执行: r1 = r2;
- 当一个对象赋值给另一个对象时，所有的数据成员都会逐位拷贝。上述赋值相当于执行

  r1.num = r2.num

  r1.den = r2.den

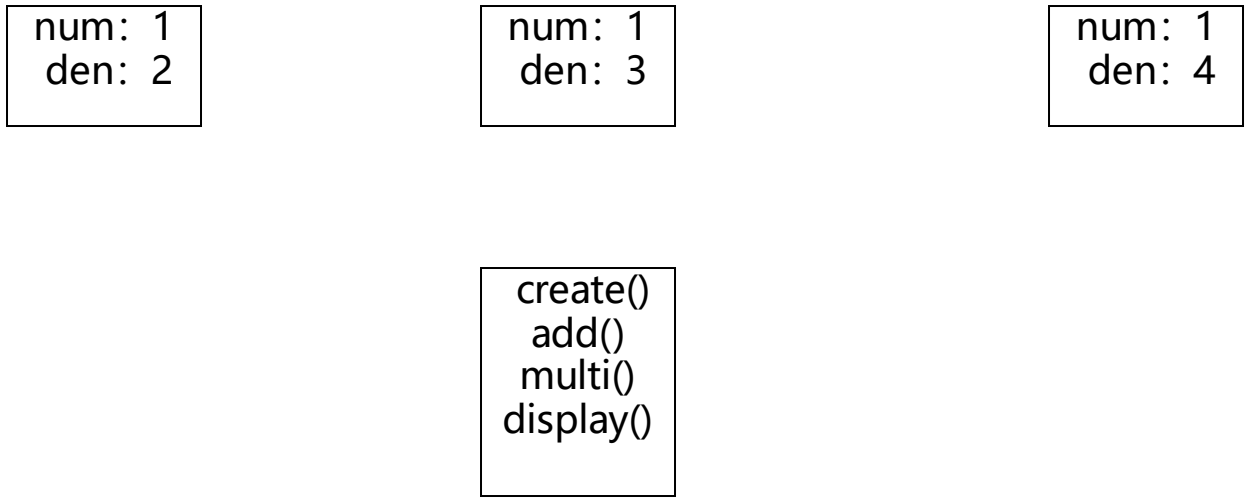
思考一个问题

•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



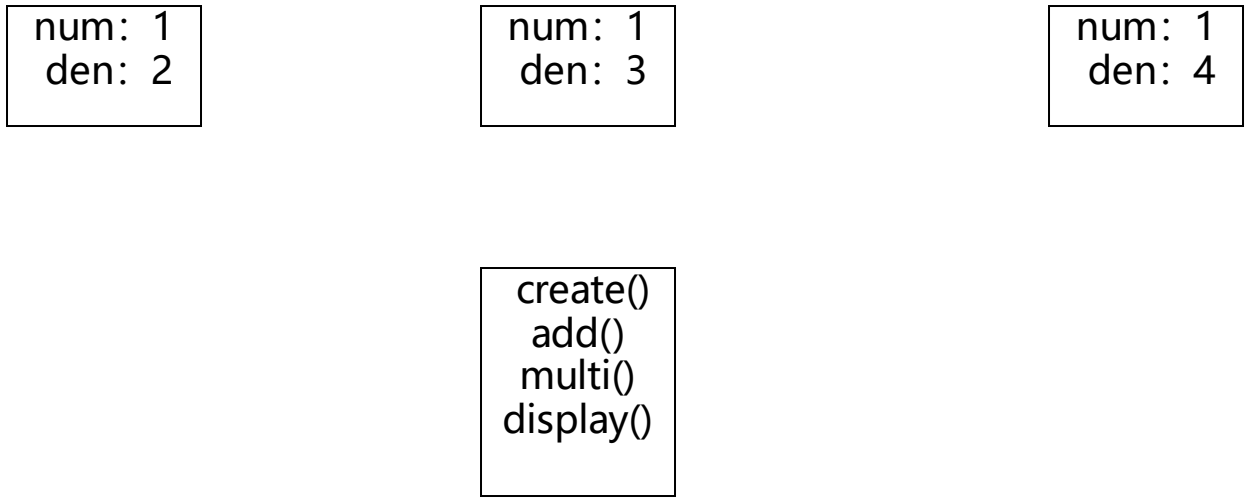
this指针

•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

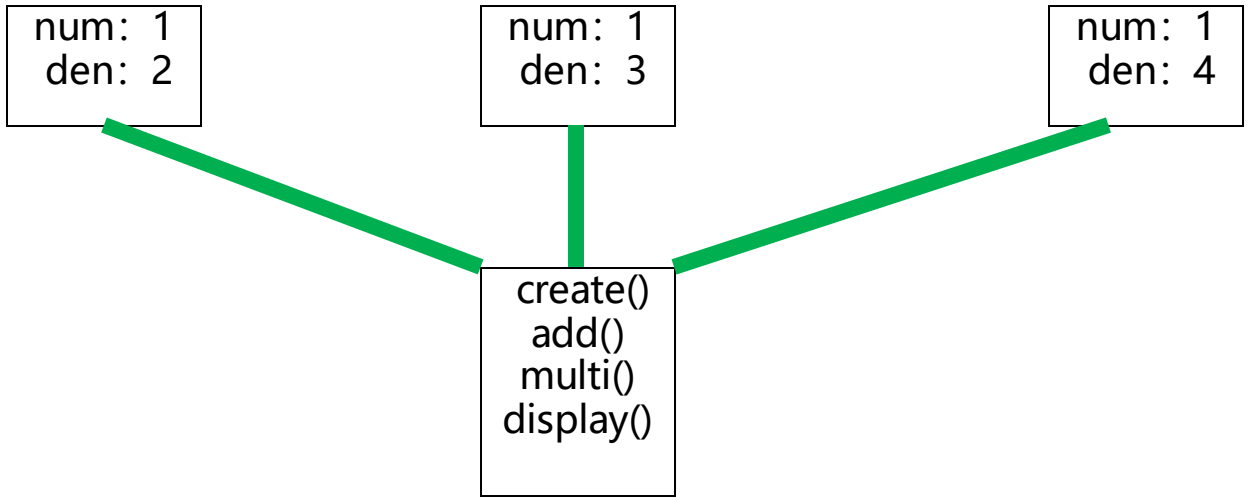
•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



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

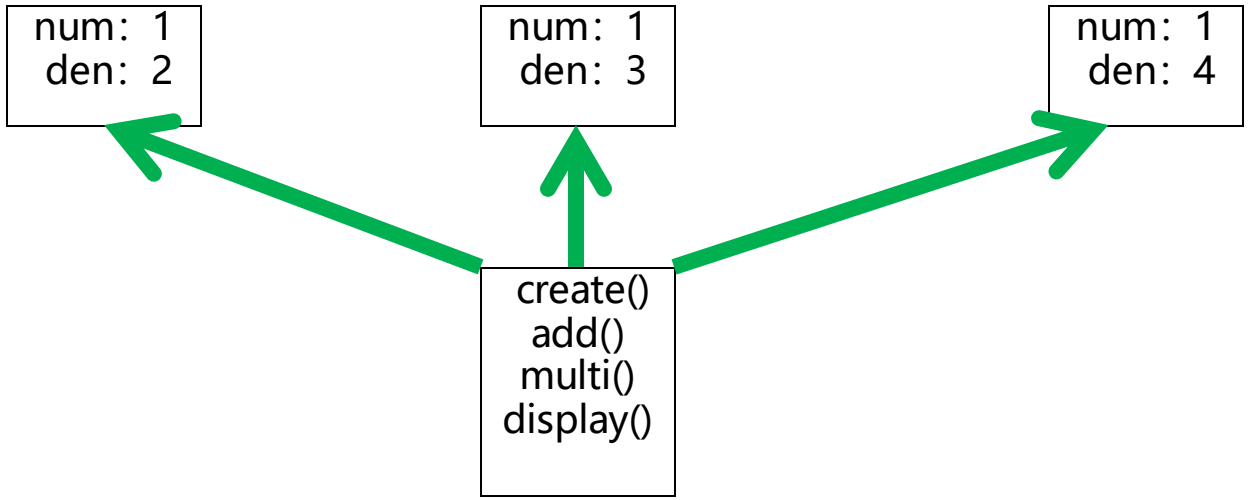
•为了优化程序，C++中，不论创建了多少个对象，成员函数在内存中只有一个副本，所有对象共享



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;}
```