第十章 程序设计复合类型-类

主讲教师：同济大学电子与信息工程学院 陈宇飞
同济大学电子与信息工程学院 龚晓亮

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

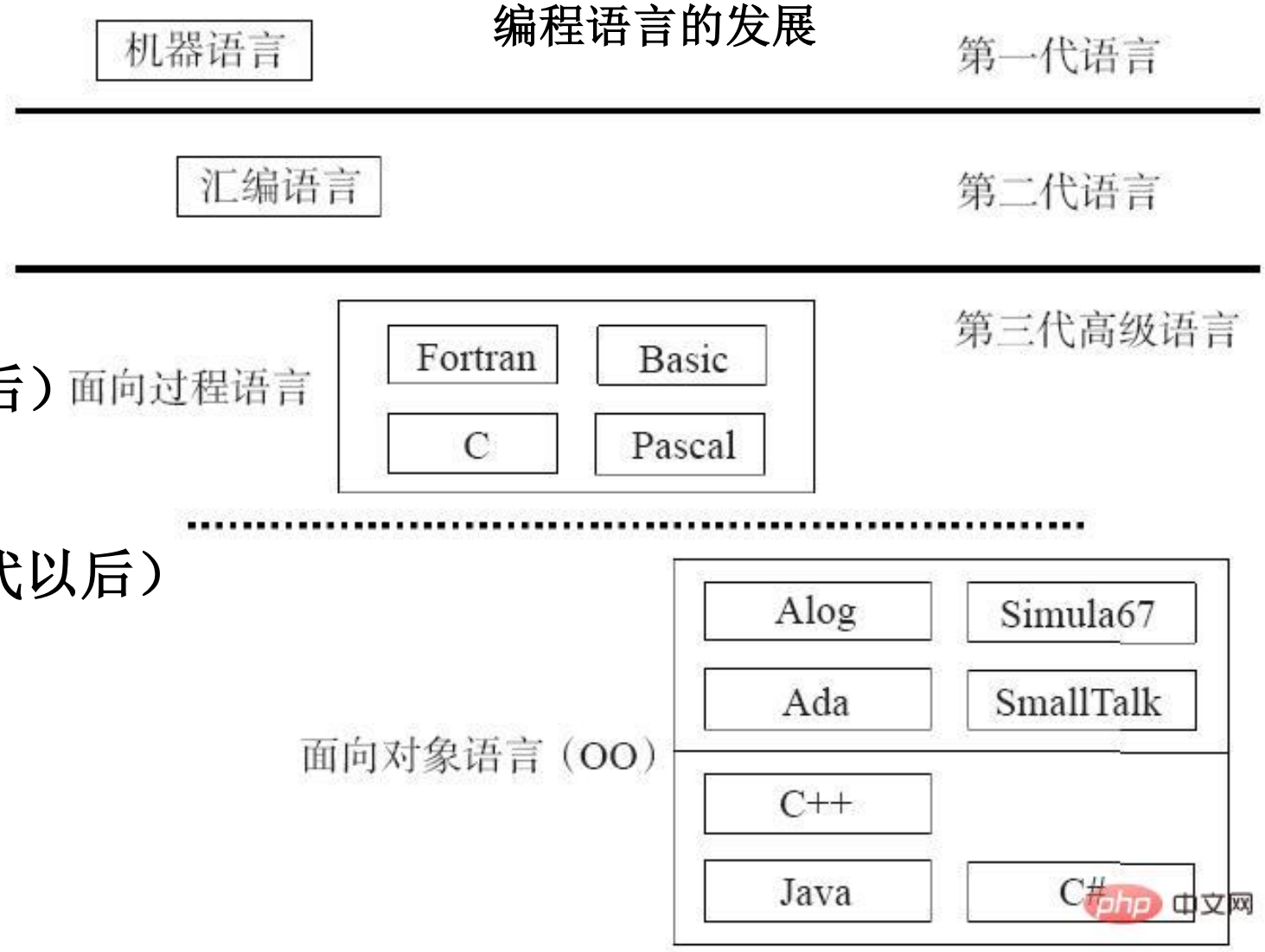
1.1 基本概念

- 概念的引出：

◆初期程序设计（60年代以前）

◆结构化程序设计（70年代以后）面向过程语言

◆面向对象的程序设计（80年代以后）



➢初期程序设计

goto语句与汇编语言里面的jmp指令相同，(无条件转移)

```
1+2+3........+8+9+10
```

NASM描述：

```
mov ax,1
mov bx,0
start:add bx,ax
inc ax
cmp ax,11;比较指令
jzend ;零转移
jmp start
end:
;
;start: end:是标号
```

```
A: //code section A
//code
goto B;
//code
goto C;
B: //code section B
//code
goto A;
//code
goto C;
C: //code section C
//code
//goto B;
//code
goto A;
```

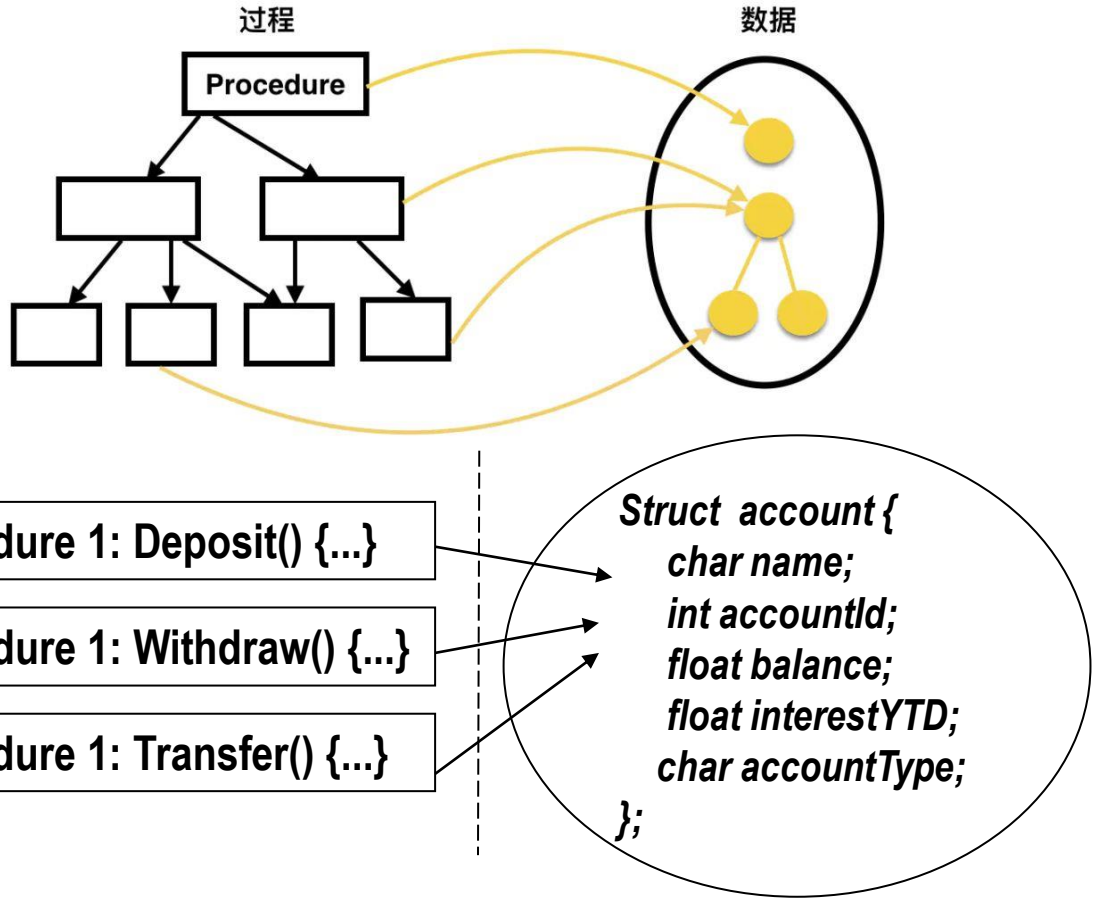
➢计算机特点：价格高、内存小、速度慢

➢程序追求：指令少、运行快，技巧型的方法

➢存在问题：质量、可读性、维护性、通用性差

（例如:大量使用GoTo语句）

➢ 结构化程序设计(Structured Programming, SP)



- 设计原则：从完成任务的过程的角度考虑，自顶而下、逐步求精、模块化
- 程序组成：功能相对独立的模块；每一模块内部均是由顺序、选择和循环三种基本结构组成
- 将数据与函数分开，围绕功能实现或操作流程来设计程序

➢ 结构化程序设计示例

例1：输入长方体的长宽高l,w,h，求长方体的体积？

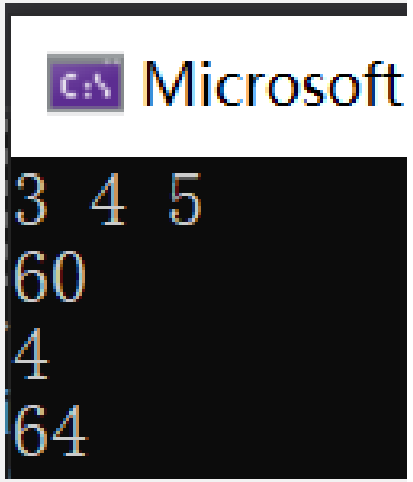
```
#include <iostream>
using namespace std;

double compute_volumn(double l, double w, double h)
{   double result;
    result = l * w * h;
    return result;
}
int main()
{   double l, w, h, result;
    cin >> l >> w >>h;
    result = compute_volumn(l, w, h);
    cout << result;
}
```

例2：求体积：第一组输入值为长方体的长宽高，第二组输入值为正方体的边长，仅含有一个值

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

double compute_volumn(double l, double w, double h)
{
    double result;
    result = l * w * h;
    return result;
}
double compute_volumn1(double edge)
{
    double result;
    result = edge * edge * edge;
    return result;
}
```




```
int main()
{
    double l, w, h, result,l1,result1;
    cin >> l >>w >>h;
    result = compute_volumn(l, w, h);
    cout << result;

    cin >> l1;
    result1 = compute_volumn1(l1);
    cout << result1;
}
```

当存在多组长方体的长宽高输入，且输入数据的格式不统一时，需要编写多个长方体体积的计算函数

例3:

```
#include <iostream>
using namespace std;
double compute_volumn(double l, double w, double h)
{   double result;
    result = l * w * h;
    return result;
}
int main()
{   double l, w, h, result;
    cin >> l >> w >> h; //假设输入5,6,7
    if (l = 10)
        cout << "The length is 10 !" << endl;
    result = compute_volumn(l, w, h); //期望求得体积210
    cout << result;
}
```

例3:

原因：对于主程序中输入的长方体的长宽高数值，这些变量可以被其他函数或语句随意访问或修改，从而使得程序得出错误的结果

```
    if (l = 10) //将 == 误写成 =，变量值被修改
        cout << "The length is 10 !" << endl;
    result = compute_volumn(l, w, h); //实际求得体积420
```

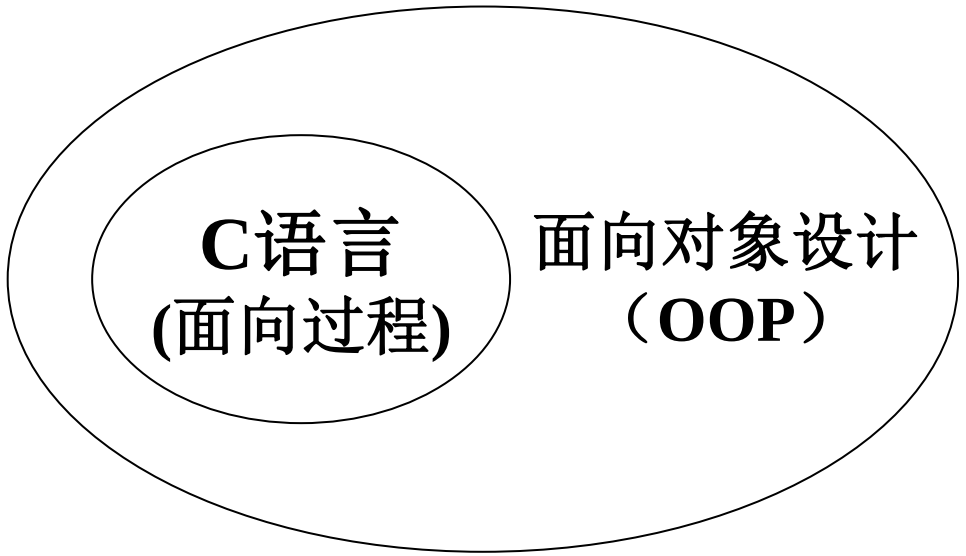
```
{   double l, w, h, result;
    cin >> l >> w >> h; //假设输入5,6,7
    if (l = 10)
        cout << "The length is 10 !" << endl;
    result = compute_volumn(l, w, h); //期望求得体积210
    cout << result;
}
```

1.1 基本概念

面向结构编程小结：

- 数据与函数分离，程序数据和操作数据的函数是分离的，对输入数据格式的改变需要编程者大量的重写操作数据的函数
- 数据存储在可以被随意访问修改的变量中（使用局部变量时，变量的作用域局限于函数内部，外界无法访问；使用全局变量时，虽然函数外界可以访问，但会带来非法访问的隐患），在大型的程序工程中，这样的不带有数据变量访问权限限制的程序设计可用性较差，安全性较低

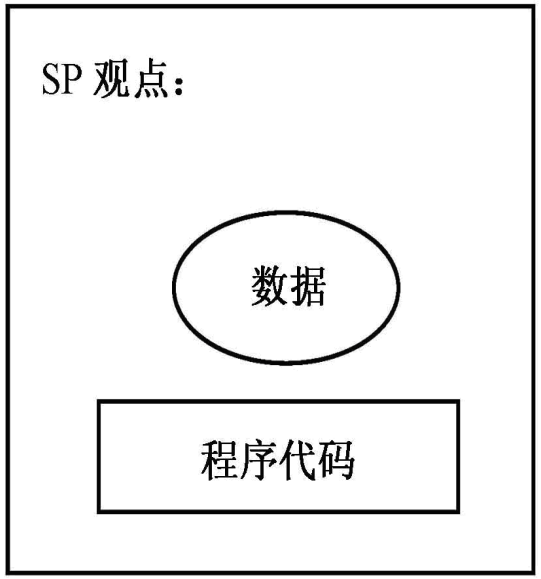
➢ 面向对象程序设计(Object Oriented Programming, OOP)

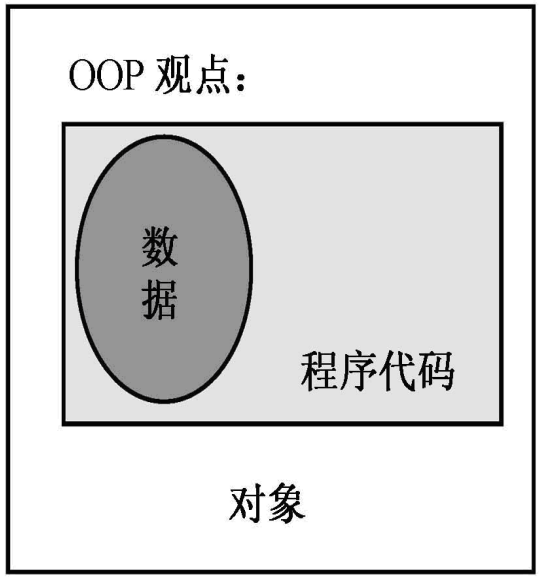


C++的组成

➢ 面向对象程序设计(Object Oriented Programming, OOP)

面向对象与面向过程程序设计





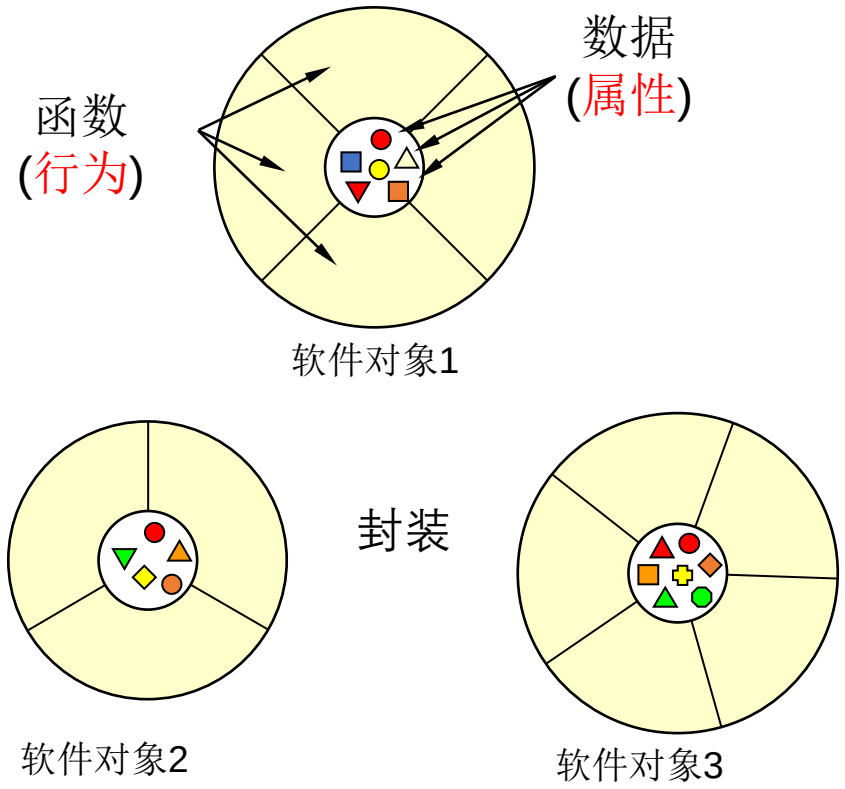
特点：

将数据与函数分开，围绕功能实现或操作流程来设计程序

特点：

将数据和函数结合在一起，放入类中

➢ 面向对象程序设计(Object Oriented Programming, OOP)



- 设计原则：从完成任务的对象的角度考虑，实现要完成的任务
- 程序组成：对象作为程序的基本单元
- 将数据和函数结合在一起，放入类中。通过封装和抽象的思想，使得程序的可复用性增强，程序语句更加简洁高效

1.1 基本概念

面向对象编程小结：

- 将数据和函数结合在一起，放入类中。通过封装和抽象的思想，使得程序的可复用性增强，程序语句更加简洁高效
- 在程序中加入数据变量访问权限限制，使得程序的可用性较强，安全性较高
- 更适合于大型软件工程项目开发

➢ 以五子棋为例：

面向过程：

1、开始游戏
2、黑子先走
3、绘制画面
4、判断输赢
5、轮到白子
6、绘制画面
7、判断输赢
8、返回步骤2
9、输出最后结果

面向对象：

1、黑白双方（玩家对象）：

—负责接受用户输入

2、棋盘系统（棋盘对象）：

—负责绘制棋子布局的变化

3、规则系统（规则对象）：

—负责判定棋局

？ 加入悔棋的功能

结构化的分解突出过程：

如何做(How to do)？它强调代码的功能是如何得以完成

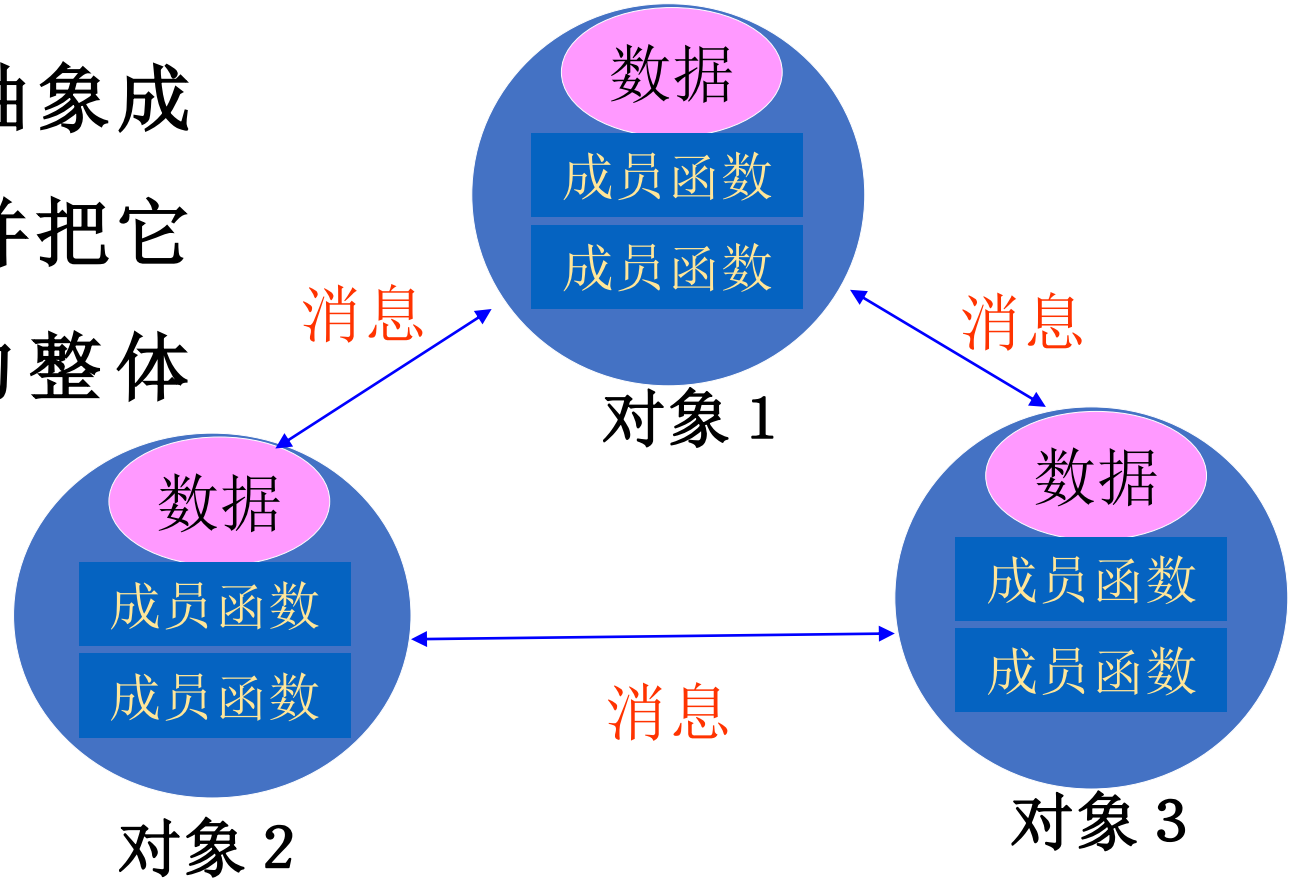
面向对象的分解突出真实世界和抽象的对象：

做什么(What to do)？它将大量的工作由相应的对象来完成，程序员在应用程序中只需说明要求对象完成的任务

1.1 基本概念

➢ 基本概念：对象

- 将客观事物的属性和行为抽象成数据和操作数据的函数，并把它们组合成一个不可分割的整体（即对象）
- 对象之间能够传递消息



1.1 基本概念

➢ 基本概念：类

- 具有相同属性和行为的一组对象的集合，它为属于该类的全部对象提供统一的抽象描述
  - 属性：即类中用于存储数据的变量
  - 行为：即类中用于操作数据的函数

类名	Clock
数据	-hour:int -minute:int -second:int
函数	+showTime():void +setTime(newH:int=0,newM:int=0,newS:int=0):void

- 类中的变量和函数统称为类的成员（分别是数据成员和成员函数）

1.1 基本概念

➢ 类的举例：

- 对于具有相同属性和行为的一组长方体对象的集合，定义该类对象的统一抽象描述长方体类

类名	Rectangle
数据	-L:double -W:double -H:double
函数	+computeVolumn():void

- 长方体类的属性：长方体的长，宽，高等变量
- 长方体类的行为：计算长方体的体积等函数

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

2.1 类的声明

➢ 类的声明格式：

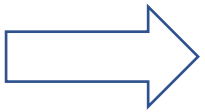
```
class classname
{
      private:
            //private members
      public:
            //public members
};
```

private members：类中的私有成员，通常为类的数据变量，不可以被类外的语句访问调用，例如长方体类的长，宽，高变量

public members：类中的公有成员，通常为类的成员函数，提供类与外界间的通信接口，可以被类外的语句访问调用，例如长方体类的体积计算函数

结构的声明：

```
struct student {
    int num;
    char name[20];
    char sex;
};
```



类的声明：

```
class student {

    int num;
    char name[20];
    char sex;

    void display() {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
        }
};
```

成员函数

结构的声明：

```
struct student {
    int num;
    char name[20];
    char sex;
};
```

类的声明：

```
class student {
  private:
    int num;
    char name[20];
    char sex;
  public:
    void display() {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
        }
};
```

访问限定符

- 成员访问限定符是限制"外部"的访问，类的"内部"不受限定符的限制

本例：
无论三个数据成员的限定符是什么，无论display函数的限定符是什么，都不影响display函数对三个数据成员的访问

类的声明：

```
class student {
 private:
    int num;
    char name[20];
    char sex;
 public:
    void display() {
        cout << "num:"  << num  << endl;
        cout << "name:" << name << endl;
        cout << "sex:"  << sex  << endl;
        }
};
```

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

3.1 对象的概念

✓类是某一类对象的统一抽象描述，而对象是某个类的一个具体的实体；对象又称为类的实例，它是由类定义的类变量

学生模型（类）

实例化

你（对象）

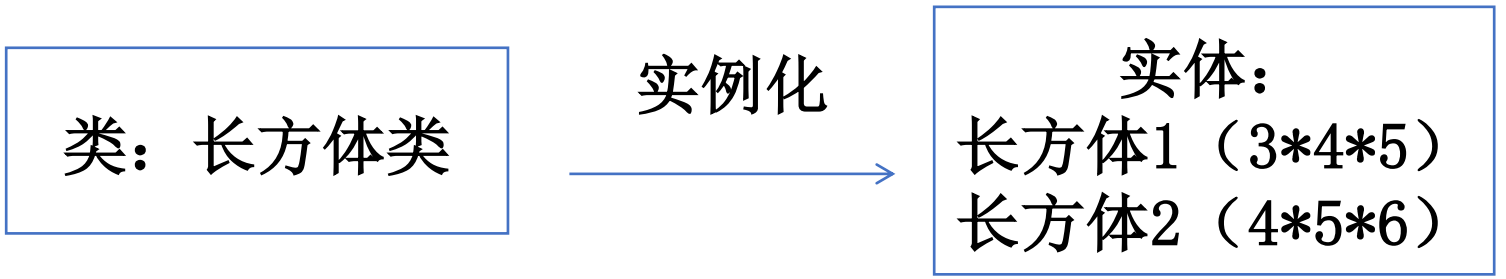
3.1 对象的概念

✓ 定义实例对象：

结构体类型 ---------> 变量

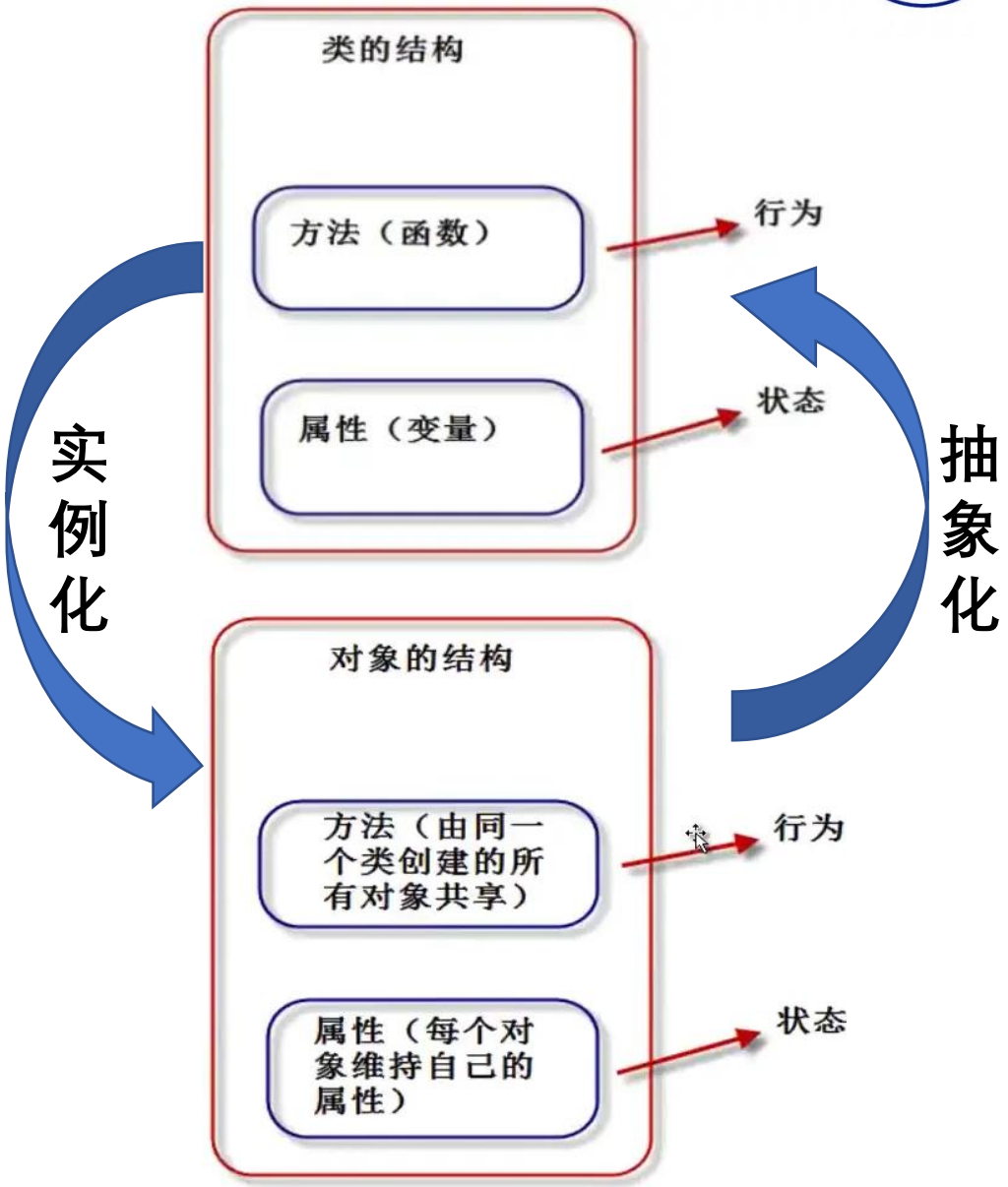
实例化

类 ---------> 对象



3.2 类和对象的关系

1. 对象是类的实例，它是由类定义的类变量
2. 同一个类定义的不同实例拥有相同的操作函数集合，相同的属性变量集合
3. 同一个类定义的不同对象根据不同的对象名互相区分，不同的对象将被分配不同的存储空间，不同的对象之间的存储空间互相独立不相关
4. 对象中只包含类的数据成员，不包含成员函数，同一类的所有对象，共用类的成员函数



3.3 对象的定义

✓ 对象的定义格式：

类似于基本数据类型变量的定义

```
classname objectname;
```

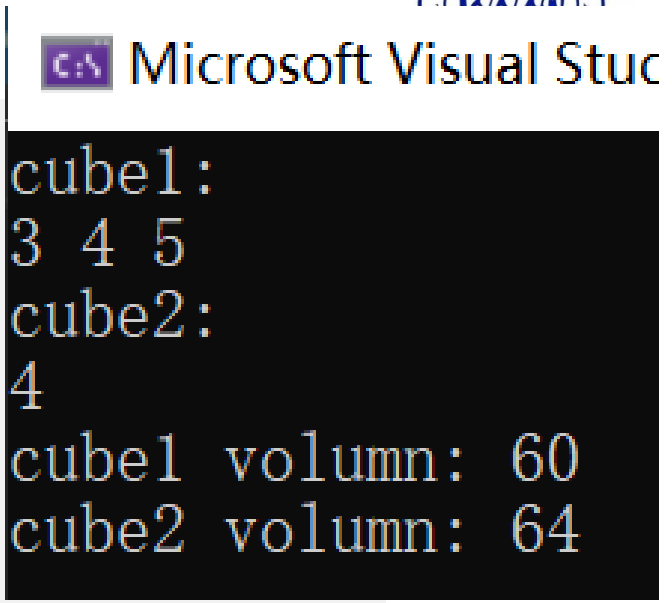
在对象定义时，通常需要为对象的成员中的变量设定初始值

对象定义的初始值设置可使用对象的构造函数实现（后续章节）

例4：实现前面的编程任务：输入长方体的长宽高l,w,h，求长方体的体积?

```
#include <iostream>
using namespace std;
class cuboid  //长方体类的定义
{  private:
      double length; double width; double height;
   public:
      double compute_volumn()
      {     return length * width * height;
      }
      void set_value(double l, double w, double h)
      {     //设置类的参数的函数，非构造函数
            height = h;
            width = w;
            length = l;
      }
};
```

```
//续
int main()
{
	cuboid cube1, cube2; //长方体1，长方体2对象的定义
	double length, width, height;
	cout << "cube1: " << endl;
	cin >> length >> width >> height;
	cube1.set_value(length, width, height);  //设置对象初值
	cout << "cube2: " << endl;
	cin >> length;
	cube2.set_value(length, length, length);   //设置对象初值
	//计算输出长方体体积
	cout << "cube1 volumn: " << cube1.compute_volumn() << endl;
	cout << "cube2 volumn: " << cube2.compute_volumn() << endl;
}
```



3.3 对象的定义

- 对象的定义方法一：先定义类，再定义对象

```
class student {
        ...
};
student s1;
student s2[10];
student *s3;
```

```
struct student {
    ...
};
struct student s1;
struct student s2[10];
struct student *s3;
```

◆对象占用实际的内存空间，根据不同类型在不同区域进行分配

3.3 对象的定义

- 对象的定义方法二：在定义类的同时定义对象

```
class student {
    ...
} s1, s2[10], *s3;
student s4;
```

```
struct student {
    ...
} s1, s2[10], *s3;
~~struct~~ student s4;
```

◆此时可以再次用上一页的方法一定义新的对象

3.3 对象的定义

- 对象的定义方法三：直接定义对象(类无名)

```
class {
    ...
} s1, s2[10], *s3;
```

```
struct {
    ...
} s1, s2[10], *s3;
```

◆因为类无名，此时无法再用的前两页的方法一进行新的对象定义

➢类与结构体的比较：

```
class student {
    private:
        int num;
        char name[20];
        char sex;
    public:
        void display() {
            cout << "num:"  << num  << endl;
            cout << "name:" << name << endl;
            cout << "sex:"  << sex  << endl;  }
};
```

替换为struct,功能完全相同

在C++中，结构体也可以加成员函数，能够实现和类完全一样的功能

➢类与结构体的比较：

```
struct student {
    int num;
    char name[20];
    char sex;
    void display() {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
        }
};
int main()
{
    student s1;
    s1.num = 1001;
    return 0;
}
```

编译结果？

```
class student {
    int num;
    char name[20];
    char sex;
    void display() {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
        }
};
int main()
{
    student s1;
    s1.num = 1001;
    return 0;
}
```

若不指定成员访问限定符，则struct缺省为public,class缺省为private

➢类与结构体的比较：

```
struct student {
    int num;
    char name[20];
    char sex;
    void display() {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
        }
};
int main()
{
    student s1;
    s1.num = 1001;//正确
    return 0;
};
```

全部public，外界(main)可访问,与C相比,多成员函数

```
class student {
    int num;
    char name[20];
    char sex;
    void display() {
        cout << num  << endl;
        cout << name << endl;
        cout << sex  << endl;
        }
};
int main()
{
    student s1;
    s1.num = 1001;//错误
    return 0;
};
```

全部private，外界(main)不可访问

➢ 类与结构体的比较小结：

① 在结构体只包含数据成员的基础上，引入成员函数的概念，使结构体同时拥有数据成员和成员函数（在C++中，结构体也可以加成员函数，能够实现和类完全一样的功能）

```
struct Student
{
    char _name[20];
    char _sex[5];
    int _age;
    int StudentInit(char *name, char *sex, int age)
    {
        strcpy(_name, name);
        strcpy(_sex, sex);
        _age = age;
    }
};
```

➢ 类与结构体的比较小结(续)：

② 类类型的使用与结构体的使用方法基本相同

③ 同时拥有数据成员和成员函数（用sizeof(类名)计算类的大小时，成员函数不占用空间）

④ 通过类的成员访问限定符，可以指定成员的属性是私有(private)或公有(public)，私有不能被外界访问，公有可被外界所访问，由实际应用决定(通常建议数据成员private,成员函数public)；缺省情况下，类的数据成员和成员函数都是私有的(不能被外界所访问，无意义)。在类的定义中，private/public出现的顺序，次数无限制

⑤ 结构体的成员默认都是公有的，都能被外界访问

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

4.1 成员函数

- 类的成员函数：
  - ❖类的成员函数即定义在类内的函数(private/public均可)
  - ❖使用类的实例即对象，可以调用类的成员函数，实现对类内数据变量的访问和操作
  - ❖类成员函数的定义、实现及调用时参数传递的语法规则与普通函数完全相同
  - ❖类的成员函数只可以通过类的对象调用

4.2 成员函数的定义

- 类的成员函数的定义格式一：

成员函数声明和成员函数定义都写在类定义内

```
class classname
{
      public:
            returnValueType  functionName (parameterlist)
            {
                ...
                return value;
            }
};
```

```
class cuboid
{
private:
	double length;
	double width;
	double height;
public:
	double compute_volumn() //成员函数的声明和定义
	{
		return length * width * height;
	}
	void set_value(double l, double w, double h)
	{	//成员函数的声明和定义
		height = h;
		width = w;
		length = l;
	}
};
```

4.2 成员函数的定义

➢类的成员函数的定义格式二：

成员函数声明写在类定义内，成员函数定义写在类定义外

```
class classname
{     public:
            returnValueType  functionName (parameterlist);
 };
returnValueType  classname::functionName (parameterlist)
{
      …
      return value;
}
```

函数实现时需要加类的作用域限定符

```
class cuboid
{
private:
      double length;
      double width;
      double height;
public:  //成员函数声明
      double compute_volumn();
      void set_value(double l, double w, double h);
};
```

```
double cuboid::compute_volumn()          成员函数的体外实现(class外)
{     return length * width * height;
}
void cuboid::set_value(double l, double w, double h)
{     height = h;
      width = w;
      length = l;
}
```

4.3 成员函数的参数

✓类的成员函数内部使用的变量可以使用普通函数的形式参数，此时函数的参数来自于类外部的对象调用类的成员函数的实际参数输入

✓与普通函数所不同的是，类的成员函数内部可以直接使用类的成员变量，这些变量通常是类的private私有变量。在成员函数中使用这些变量时，不需要在函数的输入参数中标明这些参数

成员函数的参数举例(1):

使用类外部的对象调用类的成员函数时的输入参数

```
class cuboid
{
private:
	double length;
	double width;
	double height;
public:
	double compute_volumn()
	{
		return length * width * height;
	}
	void set_value(double l, double w, double h)
	{	//成员函数使用类外部的输入参数，即使用形式参数
		height = h;
		width = w;
		length = l;  }
};
```

成员函数的参数举例(1):

使用类外部的对象调用类的成员函数时的输入参数

```
//续前页
int main()
{
      cuboid cube1, cube2; //长方体1，长方体2对象的定义
      double length, width, height;
      cout << "cube1: " << endl;
      cin >> length >> width >> height;
      cube1.set_value(length, width, height);
      //成员函数使用类外部的输入参数，调用时输入实际参数
      cout << "cube1 volumn: " << cube1.compute_volumn() << endl;
}
```

成员函数的参数举例(2)：

使用类内部的成员变量

```
#include <iostream>
using namespace std;
class cuboid
{
private:
      double length;
      double width;
      double height;
public:
      double compute_volumn()
      {    //成员函数使用类内部的成员变量
            return length * width * height;
      }
      void set_value(double l, double w, double h)
      {     height = h;
            width = w;
            length = l;  }
};
```

成员函数的参数举例(2)：
使用类内部的成员变量

```
//续前页
int main()
{
	cuboid cube1, cube2; //长方体1，长方体2对象的定义
	double length, width, height;
	cout << "cube1: " << endl;
	cin >> length >> width >> height;
	cube1.set_value(length, width, height);
	cout << "cube1 volumn: " << cube1.compute_volumn() << endl;
	//成员函数使用类内部的成员变量，在调用时不需要输入参数
}
```

4.4 成员函数的存储方式

✓每个类的实例对象仅包含数据成员(sizeof(类的对象)=所有数据成员之和)，根据不同的定义位置占用不同的数据空间(静态数据区或动态数据区)

✓类的成员函数占用函数(代码)区，每个类的每个成员函数(包括体内实现和体外实现)只占用一段空间，所有该类的对象共用成员函数的代码空间

✓当通过对象调用成员函数时，系统会缺省设置一个隐含的this指针，指向被调用的对象，并以此来区分成员函数对数据成员的访问

4.4 成员函数的存储方式

- 类的成员函数的存储方式：

```
#include <iostream>
using namespace std;
class student {
  private:
    int num;
  public:
    void set(int n) {
      num = n;
    }
    void display() {
      cout << num << endl;
    }
};
```

```
int main()
{  student s1, s2;
   s1.set(10);
   s2.set(15);
   s1.display();   10
   s2.display();   15
   return 0;
}
```

s1.display 和 s2.display 是不同的4个字节

- 类成员函数中隐含了一个this指针，调用时会隐含传入调用对象的地址

```
class student {
  private:
    int num;
  public:
    void set(student *this, int n) {
      this->num = n;
    }
    void display(student *this) {
      cout << this->num << endl;
    }
};
```

```
s1.set(10) ⇔ s1.set(&s1, 10);
s2.display()⇔s2.display(&s2);
```

注：this不能显式写在函数声明中，但可显式访问

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

5.1 访问限制

- 对类的成员的访问，为什么需要限制?
  - ❖在大型的软件工程中，能接触到程序的人包含程序员，用户，管理人员，软件测试人员等，而软件的用户又存在不同的用户等级，此时不同身份的人对于程序中的数据的访问权限应该是不同的，否则程序会存在很大的安全隐患
  - ❖例如，对于银行软件中的客户账户信息这个类，每位软件用户应该只能访问自己的账户信息，但不能访问他人的账户信息，而管理人员则应该可以访问多位用户的账户信息
- 类的访问限制由关键字public和private实现

5.2 访问限制：public

- ✓类的公有成员，可以被类内的所有成员访问，可以被友元函数访问，也可以被该类外的类的对象访问
- ✓类的公有成员通常是类的成员函数，这些函数提供类与外界间的通信接口，这些函数可以被类外的语句访问调用
- ✓类的成员函数通常用于查询和计算类的私有数据变量

```
#include <iostream>
using namespace std;
class cuboid
{
public:  //类中的变量为public公有变量
      double length;
      double width;
      double height;
      double compute_volumn()
      {
            return length * width * height;
      }
      void set_value(double l, double w, double h)
      {     height = h;
            width = w;
            length = l;
      }
};
```

```
//续
int main()
{     cuboid cube1, cube2; //长方体1，长方体2对象的定义
      double length, width, height;
      cout << "cube1: " << endl;
      cin >> length >> width >> height;
      cube1.set_value(length, width, height);
      //使用类的对象访问类的public公有函数
      cout << "cube2: " << endl;
      cin >> length >> width >> height;
      cube2.length = length;  //使用类的对象直接访问类的public公有变量
      cube2.width = width;    //使用类的对象直接访问类的public公有变量
      cube2.height = height;  //使用类的对象直接访问类的public公有变量
      cout << "cube1 volumn: " << cube1.compute_volumn() << endl;
      //使用类的对象访问类的public公有函数
      cout << "cube2 volumn: " << cube2.compute_volumn() << endl;
      //使用类的对象访问类的public公有函数
}
```

```
cube1:
3 4 5
cube2:
4 5 6
cube1 volumn: 60
cube2 volumn: 120
```

5.3 访问限制：private

✓类的私有成员，可以被类内的成员访问，也可以被友元函数访问，但不可以被该类外的类的对象访问，也不可以被类外的语句访问调用

✓类的私有成员通常是类的私有数据变量，这些变量是在类内保存，并不能被类外语句访问的敏感私有数据

✓类的私有数据变量通常在初始化后很少进行修改的操作，但可以进行查询操作

```
#include <iostream>
using namespace std;
class cuboid
{ private:  //类中的变量为private私有变量
      double length;
      double width;
      double height;
  public:
      double compute_volumn()
      {     return length * width * height;
      }
      void set_value(double l, double w, double h)
      {     height = h;
            width = w;
            length = l;
      }
};
```

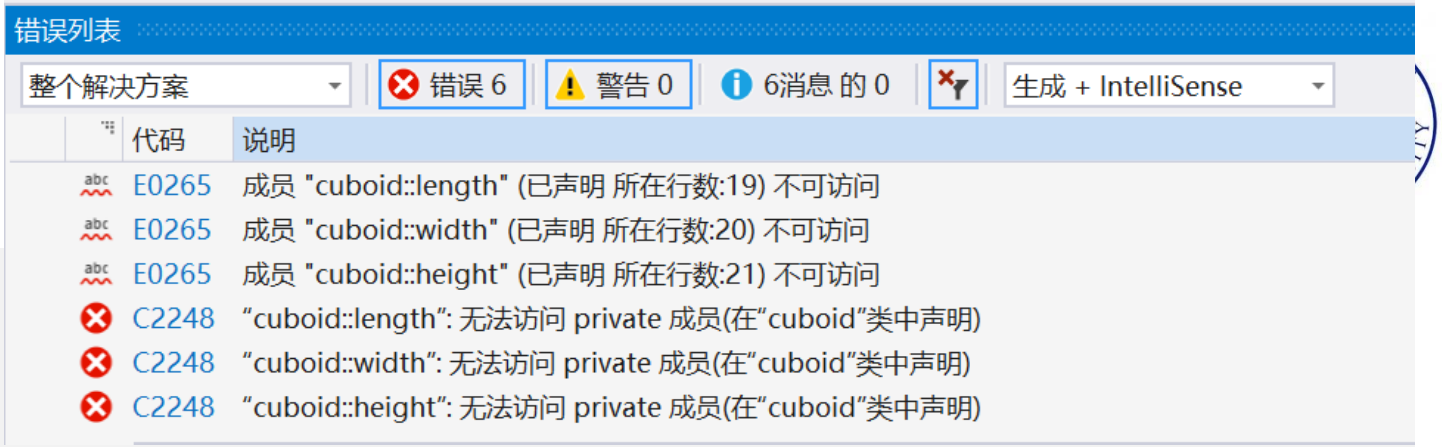
类的私有成员，可以被该类的所有成员函数访问，无论成员函数的属性是公有还是私有

```
//续
int main()
{
	cuboid cube1, cube2; //长方体1，长方体2对象的定义
	double length, width, height;
	cout << "cube1: " << endl;
	cin >> length >> width >> height;
	cube1.set_value(length, width, height);
	//使用类的对象访问类的public公有函数
	  通过类的成员函数访问类的private私有变量
	cout << "cube1 volumn: " << cube1.compute_volumn() << endl;
	//使用类的对象访问类的public公有函数
}
```

```
cube1:
3 4 5
cube1 volumn: 60
```

```
class cuboid
{ private: //类中的变量为private私有变量
      double length;
      double width;
      double height;
  public:
      double compute_volumn()
      {     return length * width * height;
      }
      void set_value(double l, double w, double h)
      {     height = h;
            width = w;
            length = l;
      }
};
```

类的私有成员，不可以被该类外的类的对象或语句访问




```
//续
int main()
{
    cuboid cube1, cube2; //长方体1，长方体2对象的定义
    double length, width, height;
    cout << "cube2: " << endl;
    cin >> length >> width >> height;
    cube2.length = length; //使用类的对象直接访问类的private私有变量
    cube2.width = width;   //使用类的对象直接访问类的private私有变量
    cube2.height = height; //使用类的对象直接访问类的private私有变量
    cout << "cube2 volumn: " << cube2.compute_volumn() << endl;
}
```

编译错！！！

5.4 访问限制（小结）

- public公有成员和private私有成员（小结）：
  - ❖对于类中的私有变量成员的访问操作通常是通过类中的公有成员函数间接进行
  - ❖对于类的操作都是通过类的实例进行的，类的实例即对象仅可以访问类的公有成员
  - ❖对于类的公有成员函数的编写过程，也应考虑对于类的私有成员的安全性保护

目录

- 基本概念
- 类的声明
- 对象的定义与访问
- 成员函数
- 访问限制
- 封装与隐藏

6.1 封装与隐藏

- 封装encapsulation：

❖封装的单位是对象(类)，将数据和操作数据的函数集合在一起，形成一个类(对象），从而实现对于数据和数据操作过程的隐藏

❖通过访问限制，从对象的外部看只能看到对象的外部特性，即能够受理哪些信息，具有哪些处理能力并会返回哪些信息；对象的内部，即处理能力的实现和内部状态，对外是不可见的

6.1 封装与隐藏

- 封装的特点：

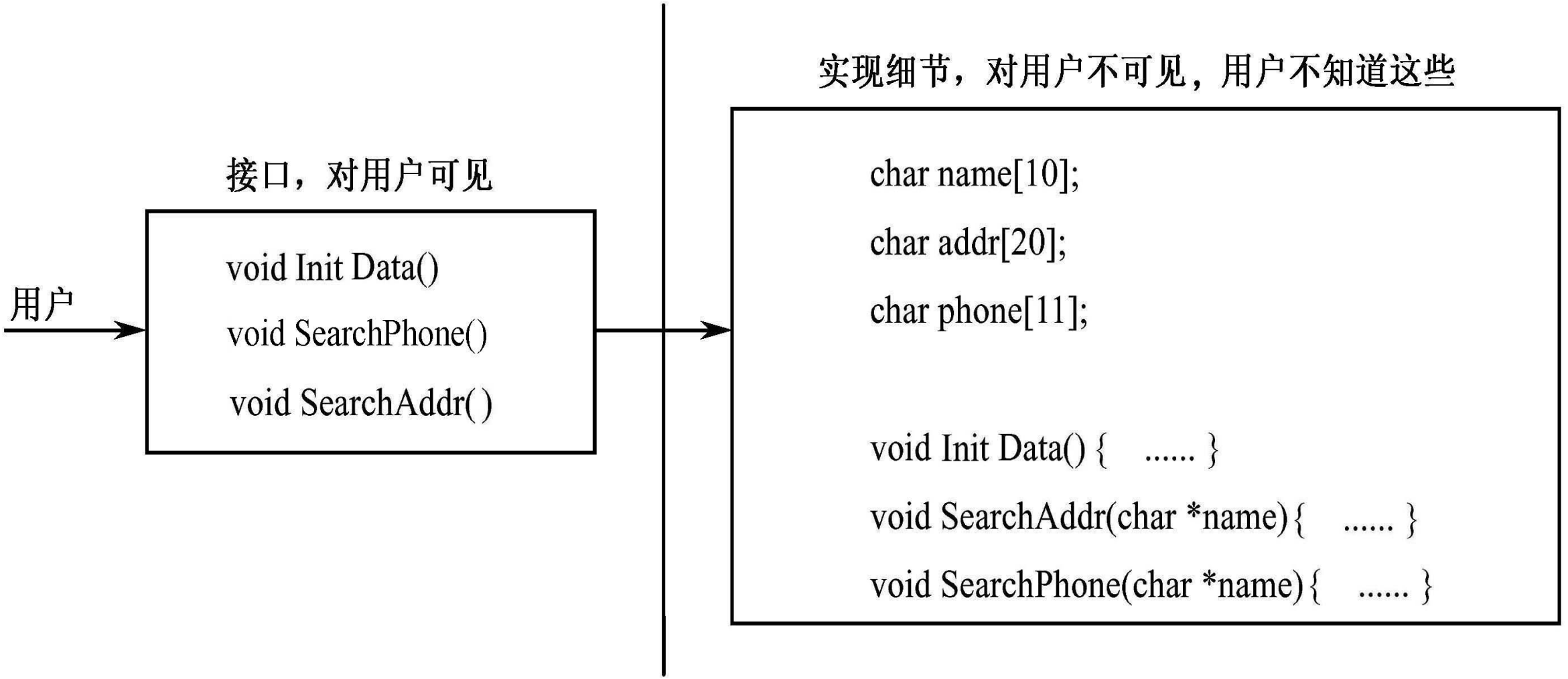
❖具有一个清楚的边界，对象的所有私有变量和函数细节都被固定在这个边界内

❖提供一些接口，供外部对象访问调用，这些接口描述了对象可以进行的操作和返回响应

❖对象内部的实现代码和数据受到封装的保护，从对象的外部不能直接修改其内部状态，即其它对象和代码不能直接修改本对象所拥有的数据和代码，对象的内部状态只能由其自身改变

6.1 封装与隐藏

- 封装的特点：



6.1 封装与隐藏

➢ 封装的优点：

❖提高部件的独立性

❖提高安全性(如银行的帐户)

❖易于维护(由于数据独立, 易于发现、修改问题)

❖封装将对象的使用者与设计者分开, 使用者只需要通过接口访问对象, 不必须了解对象的内部细节，提高了软件复用性

❖…

总结

基本概念	面向结构编程/面向对象编程
类的声明	类的含义/类的定义格式
对象的定义	对象的含义/对象的定义格式
成员函数	成员函数的声明/定义/参数/存储方式
访问限制	public公有成员/private私有成员
封装与隐藏	封装的概念/特点/优点