实验6-2

- 编写一个函数f，将传入此函数的直角坐标值转换为极坐标值，并返回主调函数中。求极坐标的公式是：c=sqrt（x*x+y*y）；
  q=arctan(y/x)
  若要将两值返回主调函数，有多种方式可以完成，请试之：
  （1）两值均以指针形参带回；
  （2）由指针形参带回一个值，函数值返回另一个值。
  （3）两值均以引用形参返回。

- ```
  void fun(float x,flaot y,float &c,float &c)
  {
  c=sqrt(x*x+y*y);
  q=arctan(y/x);
  }


  void fun (float x,flaot y,float *c,float *c)
  {
  *c=sqrt(x*x+y*y);
  *q=arctan(y/x);
  }

  float fun (float x,flaot y,float &c)
  {
  float q;
  c=sqrt(x*x+y*y);
  q=arctan(y/x);
  return q;
  }
  ```
  ```

例4.2 调用函数时的数据传递

```
#include <iostream>
using namespace std;
int main( )
{
   int a,b,c;
   int max(int x,int y) ;
   cout<<"please enter two integer numbers： ";
   cin>>a>>b;
   c=max(a,b);          //调用max函数，给定实参为a,b。函数值赋给c
   cout<<"max="<<c<<endl;
   return 0;
}
int max(int x,int y)          //定义有参函数max，x，y为形式参数
{int z;
 z=x>y?x：y;
 return(z);
}
```

有关形参与实参的说明

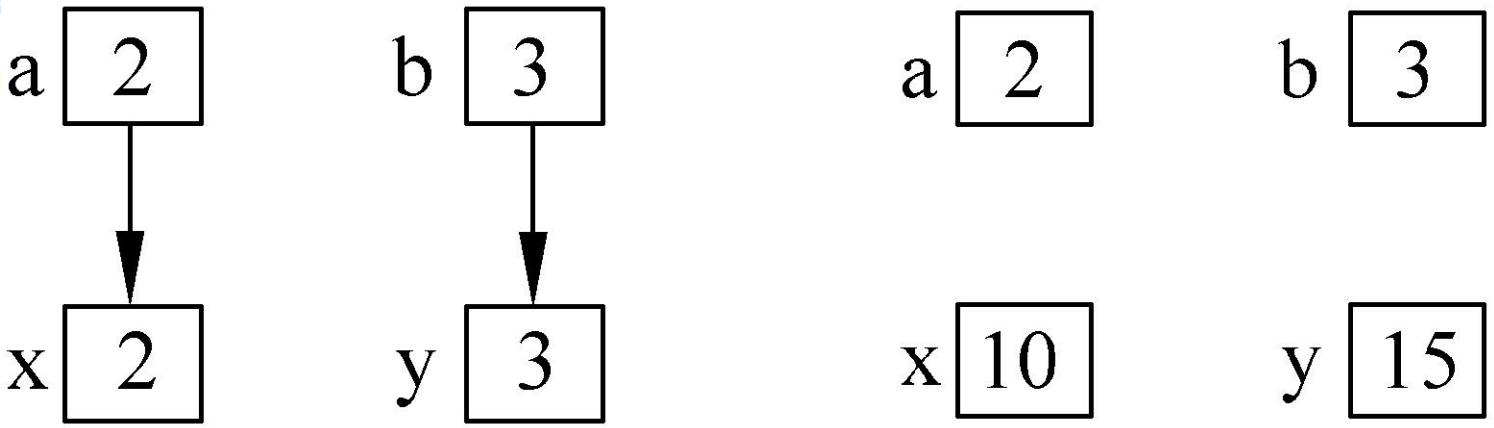
(1) 在定义函数时指定的形参，在未出现函数调用时，它们并不占内存中的存储单元，因此称它们是形式参数或虚拟参数。

只有在发生函数调用时，函数max中的形参才被分配内存单元，以便接收从实参传来的数据。

在调用结束后，形参所占的内存单元被释放。

(2) 定义函数时，必须在函数首部指定形参类型。

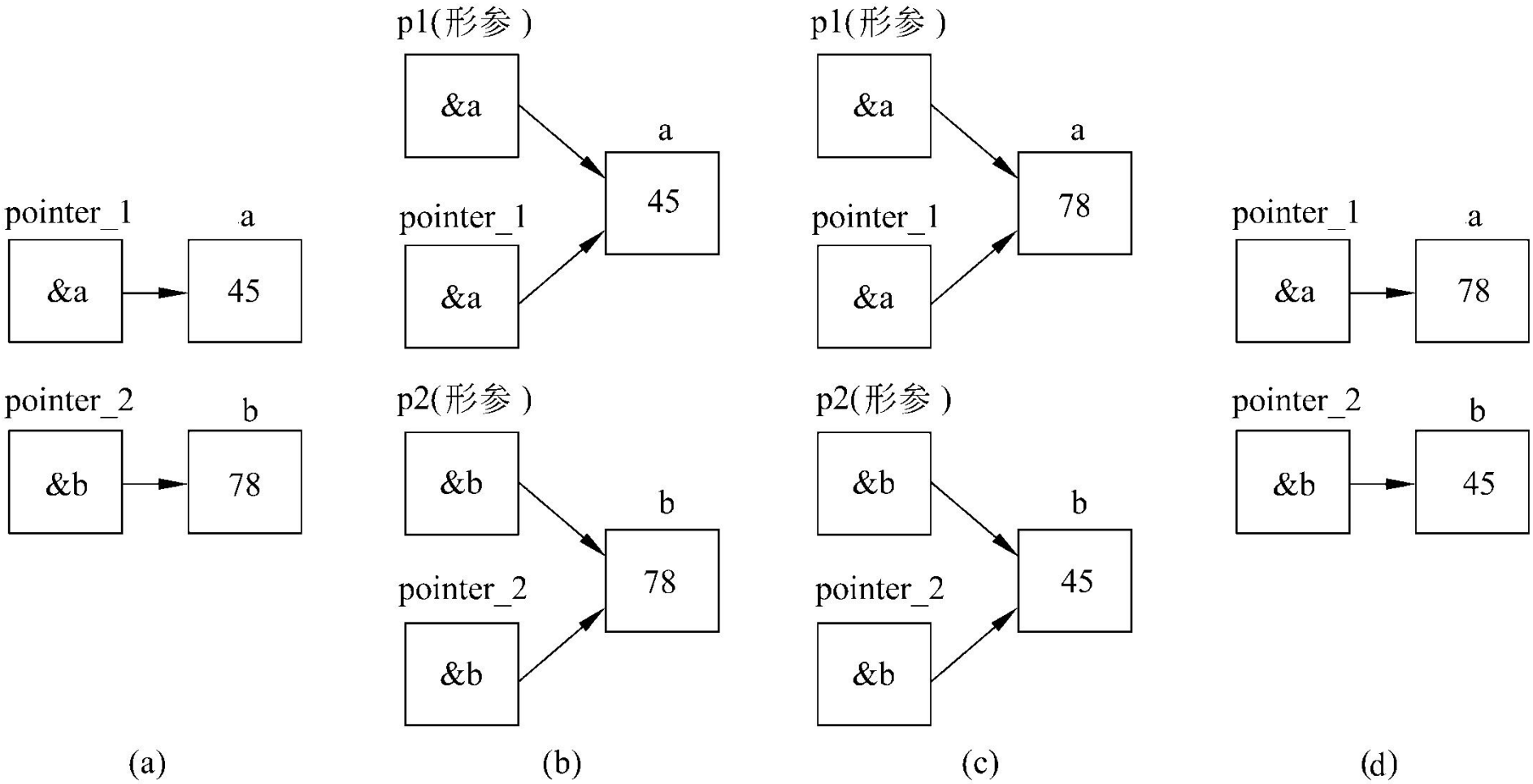
(3) 实参变量对形参变量的数据传递是“值传递”，即单向传递，只由实参传给形参。形参的值如果改变，不会改变主调函数中实参的值。



例6.3 输入a和b两个整数，按先大后小的顺序输出a和b，要求用函数处理，函数参数为指针。

```
#include  <iostream>
using namespace std;
int main( )
{
   void swap(int *p1,int *p2);    //函数形参为指针
   int *pointer_1,*pointer_2,a,b;
   cin>>a>>b;
   pointer_1=&a;                  //使pointer_1指向a
   pointer_2=&b;                  //使pointer_2指向b
   if(a<b) swap(pointer_1,pointer_2);
   cout<<"max="<<a<<" min="<<b<<endl;
   return 0;
}
```

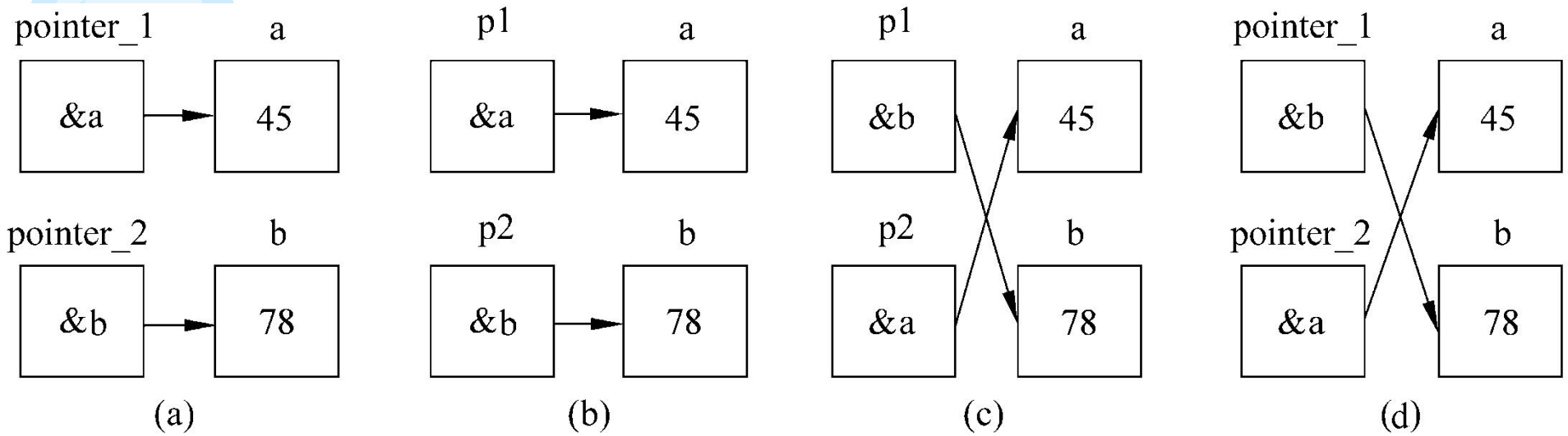
void swap(int *p1,int *p2)



```
void swap(int *p1,int *p2)
{
    int *temp;
    temp=p1;
    p1=p2;
    p2=temp;
}
```

//不能通过改变形参指针变量的值而使实参指针变量的值改变。

实参变量和形参变量之间的数据传递是单向的“值传递”方式。指针变量作函数参数也要遵循这一规则。调用函数时不会改变实参指针变量的值，但可以改变实参指针变量所指向变量的值。



```
#include<iostream>
#include<cmath>
using namespace std;
double* f(int a,int b);
double*f1(int x,int y,double*m);
int main(){
double*z, m;
int x,y;
cout<<"请输入两个数"<<endl;
cin>>x>>y;
z=f(x,y);
cout<<"sqrt值"<<*z<<endl;
cout<<"tan值"<<*(z+1)<<endl;
z=f1(x,y,&m);
cout<<"sqrt值"<<*z<<endl;
cout<<"tan值"<<m<<endl;return 0;
}
```

```
double* f(int x,int y){
double c,q,*p;
c=sqrt(x*x+y*y);
q=atan(y/x);
p=new double(sizeof(double)*2);
*p=c;
p++;
*p=q;
p--;
return p;
}
double*f1(int x,int y,double*m){
double c,q,*p;
c=sqrt(x*x+y*y);
p=new double(sizeof(double));
q=atan(y/x);
*p=c;
*m=q;
return p;
}
```

第3篇

基于对象的程序设计

第8章 类和对象

第9章 关于类和对象的进一步讨论

第10章 运算符重载

第8章 类和对象

Review：Structured programming

- **Emphases:**
  - **How to separate the data from the function**
- **Form：**
  - **Main module + several sub-module (main()+ sub-functions)。**
- **Characteristic：**
  - **Break up the task into several modules。**
- **Defects：**
  - **Low efficiency， reuse rate of the program is low。**

OOP，object oriented programming

- **Purpose：**
  - Realize the industrialization of software design
- **Viewpoint：**
  - The nature is composed of entities (objects)
- **Programmer method：**
  - Use the viewpoint of object-oriented to describe and handle the actual problems
- **Request：**
  - Highly-generalization、classification and abstract.

8.1 object oriented programming, OOP

面向过程的程序设计对于大规模开发程序，显得力不从心。

面向对象程序设计（**object oriented programming, OOP**）的思路：

委托式指挥

搬运工

1. 对象（Object）

客观世界中任何一个事物都可以看成一个对象(object)。

对象是构成系统的基本单位。

对象的两个要素，即属性(attribute)和行为(behavior)，它能根据外界给的信息进行相应的操作。

一个对象往往是由一组属性和一组行为构成的。

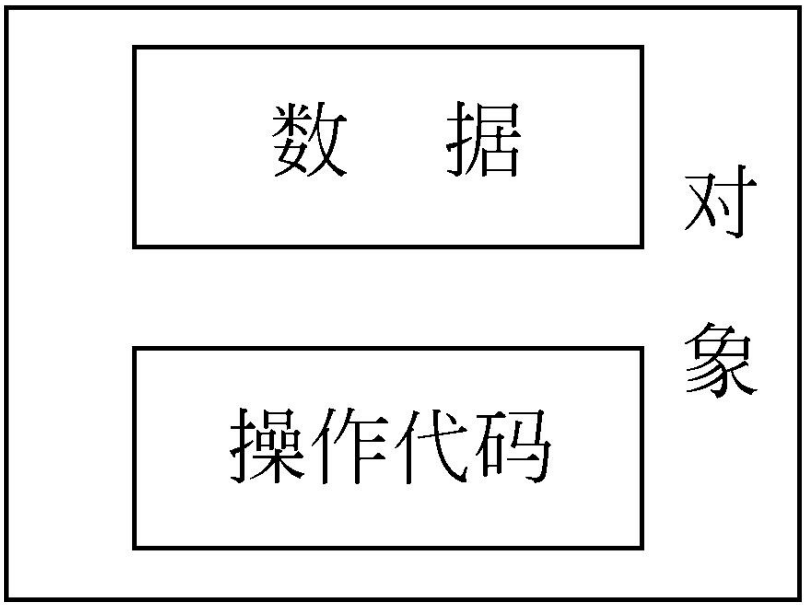
对象之间通过发送和接收消息互相联系。

对象由数据和函数组成。

数据体现“属性”。

函数体现“行为”。也称为方法(method)。

调用对象的函数就是向该对象传送一个消息(message)，要求该对象实现某一行为(功能)。



2. 抽象

抽象(abstraction）的过程是将有关事物的共性归纳、集中的过程。

抽象的作用是表示同一类事物的本质。

对象是具体存在，它的类型称为“类(class)”。

类是对象的抽象，代表了某一批对象的共性和特征。而对象则是类的实例，是类的具体表现形式。

2. Abstract

Abstract is course of generalizing the actual object, getting the common characters of this object and describing the characters

- Generalize the actual object;
- Data abstract：attribute or state to the object （Physical value to tell apart the different object）。
- Code abstract：To describe the mutual behavior character or function of a particular object
- Realization of abstract：using declaration of class

Abstraction Example——Clock

- **Data abstraction：**

  int Hour，int Minute，int Second

- **Code abstraction：**

  SetTime（ ），ShowTime（ ）

Abstraction Realization——clock class

```
class  Clock
{
public:
        void SetTime(int NewH,  int NewM, int NewS);
        void ShowTime（ ）;
private:
        int Hour,Minute,Second;
};
```

Abstraction Example—human beings

- **Data abstraction：**

  char *name, char *sex, int age, int id

- **Code abstraction：**

  Angle from biology attribute ：

  GetCloth（ ）， Eat（ ）， Step（ ）,…

  Angle from social attribute：

  Work（ ）, Promote（ ） ,…

3. Encapsulation

- **Definition:**
  - Combining the data member and code member abstracted from entity together as a whole
- **Goal:**
  - Reinforce security and predigest programming.
  - the user needn't know the realization detail.
  - use the members of the class with special accessing purview by using the outer interface of the class.
- **Realization of encapsulation:**
  - The "{}" in the class

3. 封装与信息隐蔽

封装 (encapsulation)：

一是将有关的数据和操作代码封装在一个对象中，形成一个基本单位，各个对象之间相对独立，互不干扰。

二是隐蔽对象内部细节，只留下少量接口（如函数名）与外界联系，接收外界的消息，称为信息隐蔽(imformation hiding)。信息隐蔽有利于数据安全。

Encapsulation Example

- class Clock

```
{
public:
    void SetTime(int NewH,int NewM, int NewS);
    void ShowTime（ ）;
private: int Hour,Minute,Second;
};
```

Outer interface

Special accessing purview

4. 继承与重用

C++提供了继承机制，利用继承可以很方便地利用一个已有的类建立一个新的类。

即“软件重用”(software reusability)。

“马”是父类，或称为基类；

“白马”是从“马”派生出来的，称为子类或派生类。

4. Inheritance and derivation

- **Definition:**

It is a mechanism which supports arrangement classification in C++ programming. And it allows the programmer to explain the class in detail based on keeping the characters of original class.

- **Realization:**

Assertion of derivation class

5. 多态性

在C++中，所谓多态性(polymorphism)是指：

由继承而产生的相关的不同的类，其对象对同一消息会作出不同的响应。

多态性是面向对象程序设计的一个重要特征，能增加程序的灵活性。

5. Polymorphism

- **Polymorphism：**

  The same name, but the different function-realization way

- **Goal：**

  Achieve the unification of action， and lessen the number of identifier in the program.

- **Realization：**

  overloading functions and virtual functions

8.1.2 面向对象程序设计的特点

面向对象程序设计面对的是一个个对象。

程序设计者的任务包括两个方面：

一是设计所需的各种类和对象，即决定把哪些数据和操作封装在一起；

二是考虑怎样向有关对象发送消息，以完成所需的任务。

8.1.3 类和对象的作用

类是面向对象程序设计的基础，是所有面向对象的语言的共同特征。

基于对象的程序：以类和对象为基础的，程序的操作围绕对象进行。

面向对象的程序：利用继承机制和多态性。

面向过程的结构化程序设计：

程序=算法＋数据结构

算法和数据结构两者是互相独立、分开设计的，面向过程的程序设计是以算法为主体的。多对多的关系。

面向对象程序设计就是把一个算法和一组数据结构封装在一个对象中。一一对应关系。

对象 = 算法 + 数据结构

程序 = (对象+对象+对象+...) + 消息

消息的作用就是对对象的控制。

程序设计的关键是设计好每个对象，确定向这些对象发出的命令，使各对象完成相应操作。

8.1.4 面向对象的软件开发

面向对象的软件工程包括以下几个部分：

1. 面向对象分析(object oriented analysis,OOA)

软件工程中的系统分析阶段，系统分析员要和用户一起，对用户的需求作出精确的分析和明确的描述，从宏观的角度概括出系统应该做什么。

面向对象的分析，要归纳出有关的对象(包括对象的属性和行为)以及对象之间的联系，并将具有相同属性和行为的对象用一个类(class)来表示。

建立一个能反映真实工作情况的需求模型。

2. 面向对象设计(object oriented design,OOD)

根据面向对象分析阶段形成的需求模型进行具体的设计，首先是进行类的设计。然后以这些类为基础提出程序设计的思路和方法，包括对算法的设计。

在设计阶段，并不牵涉某一种具体的计算机语言，而是用一种更通用的描述工具(UML)来描述。

3. 面向对象编程(object oriented programming, OOP)

根据面向对象设计的结果，选用一种面向对象的计算机语言把它写成程序。

4. 面向对象测试(object oriented test,OOT)

在写好程序后交给用户使用前，必须对程序进行严格的测试。测试的目的是发现程序中的错误并改正它。面向对象测试是以类作为测试的基本单元。

5. 面向对象维护(object oriented soft maintenance, OOSM)

因为对象的封装性，修改一个对象对其他对象影响很小。利用面向对象的方法维护程序，大大提高了软件维护的效率。

8.2 类的声明和对象的定义

8.2.1 类和对象的关系

类(class)是对象的抽象（不占用内存），而对象是类的具体实例(instance)是具体的（占用存储空间）。

在C++中通常先声明一个类类型，然后用它去定义若干个同类型的对象。对象就是类类型的一个变量。

Class in C++

- **Definition**

  The class is a collection of objects with similar attributes and actions, and it provides uniform abstraction description to the whole objects belonging to this class. The interior of the class includes the two main parts: attributes and actions.

- **Function of class**

  Making use of class, you can realize the data's encapsulation、concealment 、inheritance and derivation.

- **Characteristic**

  More higher modularization than functions in C/C++ languange.

The definition of class

```
// 声明部分
class Student                    //以class开头,大写字母开头
{
    private:                     //私有成员
    int num;
    char name[20];
    public:                      //公有成员（外部接口）
    void display( )
    protected:
    char sex;                    // 保护型成员
};
```

- private和public称为成员访问限定符(member access specifier)。

The definition of class

- //实现部分－各个成员函数的实现
- void Student::display( )
- {
- cout<<"num:"<<num<<endl;
- cout<<"name:"<<name<<endl;
- cout<<"sex:"<<sex<<endl;
- }

- “::”是作用域限定符(scope/field qualifier)

member access specifier

- **Public members:**
- **Declared behind the keyword "public", it is an interface through which class can interact with outer functions. Any outer functions can access the public members: public data and functions.**
- **Private members:**
- **Declared behind the keyword "private", it only can be accessed by the functions in the class and can't be accessed by those out of the class.**
- **Protected members:**
- **Similar to private members, the difference is the different effect to derivation class when realizing the inheritance and derivation.**

8.2.3 定义对象的方法

1. 先声明类类型，然后再定义对象

[class] 类名 对象名

如 class Student stud1,stud2;

2. 在声明类类型的同时定义对象

```
class Student                    //声明类类型
{
…
}stud1，stud2;                   //定义了两个Student类的对象
```

3. 不出现类名，直接定义对象

8.2.4 类和结构体类型的异同

C++增加了class类型后，仍保留了结构体类型(struct)，而且把它的功能也扩展了（结构也可包含成员函数）。

用struct声明的结构体类型实际上也就是类。

用struct声明的类，如果对其成员不作private或public的声明，系统将其默认为public。

用 class 声明的类，如果对其成员不作private或public的声明，系统将其默认为private。

8.3 类的成员函数

8.3.1 成员函数的性质

类的成员函数与一般函数的区别只是： 它是属于一个类的成员，出现在类体中。它可以被指定为private(私有的)、public(公用的)或protected(受保护的)。

在使用类函数时，要注意调用它的权限(它能否被调用)以及它的作用域(函数能使用什么范围中的数据和函数)。

例如私有的成员函数只能被本类中的其他成员函数所调用，而不能被类外调用。

成员函数可以访问本类中任何成员(包括私有的和公用的)，可以引用在本作用域中有效的数据。

一般的做法是将需要被外界调用的成员函数指定为public，它们是类的对外接口。

工具函数(utility function) 应该指定为private，其作用是支持其他函数的操作。

The realization of member functions

- **Located within the class**
  - **When located within the class, the member functions can be defined as mentioned previously, and it is defaulted as inline functions.**

  Note:

  - **Within the class, the member functions have a small scale, and you can't use the "switch" sentences.**
- **Located out of the class**

  建议：在类的内部对成员函数作声明，而在类体外定义成员函数。

8.3.2 在类外定义成员函数

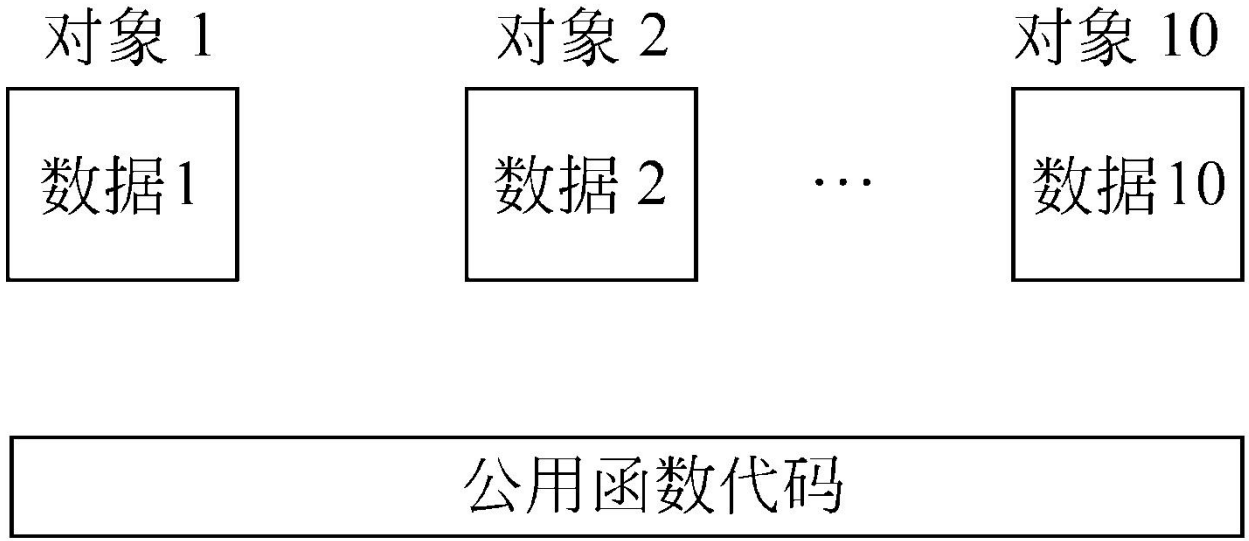
```
class Student
{
public:
        void display( );          //公用成员函数原型声明
private:
        int num;                  //私有数据成员
        string name;
        char sex;
};
void Student::display( )                  //在类外定义display类函数
{
        cout<<"num:"<<num<<endl;
        cout<<"name:"<<name<<endl;
        cout<<"sex:"<<sex<<endl;
}
Student stud1,stud2;                      //定义两个类对象
```

8.3.3 inline 成员函数

```
class Student
{
public:
        [inline] void display( );    //成员函数被隐含地指定为内置函数
private:
        int num;
        string name;
        char sex;
};
inline void Student::display( )  // 在类外定义为内置函数
{
        cout<<"num:"<<num<<endl;
        cout<<"name:"<<name<<endl;
        cout<<"sex:"<<sex<<endl;
}
```

8.3.4 成员函数的存储方式

用类定义对象时，系统会为每一个对象分配存储空间。每个对象所占用的存储空间只是该对象的数据部分所占用的存储空间，而不包括函数代码所占用的存储空间。



8.4 对象成员的引用

- 通过对象名和成员运算符访问对象成员：
- 对象名.成员名
- 通过指向对象的指针访问对象成员：
- Time t,*p; p=&t;
- (*p).hour ， p->hour，t.hour三者等价。
- 通过对象的引用变量访问对象成员：
- Time &t2=t;
- t2.hour就是t.hour。

8.5 类的封装性和信息隐蔽

8.5.1 公用接口与私有实现的分离

C++通过类来实现封装性，把数据和与这些数据有关的操作封装在一个类中。

用户主要是通过调用公用的成员函数来实现类提供的功能。

公用成员函数是类的对外公用接口(**public interface**) 。

通过成员函数对数据成员进行操作称为类的实现（implementation）。

用户看不到公用成员函数的源代码，只能接触到其目标代码（object code）。

类中数据是私有的，实现的细节对用户是隐蔽的，称为私有实现(private implementation)。

“类的公用接口与私有实现的分离”形成了信息隐蔽。

软件工程的一个最基本的原则就是将接口与实现分离，信息隐蔽是软件工程中一个非常重要的概念。它的好处在于：

(1) 如果想修改或扩充类的功能，只需修改本类中有关的数据成员和与它有关的成员函数，程序中类外的部分可以不必修改。

(2) 如果在编译时发现类中的数据读写有错，不必检查整个程序，只需检查本类中访问这些数据的少数成员函数。

8.5.2 类声明和成员函数定义的分离

在面向对象的程序开发中，一般做法是将类的声明(其中包含成员函数的声明)放在指定的头文件中。

在程序中可以用该类来定义对象，可以调用这些对象的公用成员函数。

为了实现信息隐蔽，对类成员函数的定义一般不放在头文件中，而另外放在一个文件中。

例：类声明和成员函数定义的分离

```
//student.h                    //头文件，进行类的声明
class Student                  //类声明
{
public:
   void display( );                //公用成员函数原型声明
private:
   int num;
   char name[20];
   char sex;
};
```

```
//student.cpp                    //在此进行函数的定义
#include <iostream>
using namespace std;
#include "student.h"           //不要漏写此行
void Student::display( )    //在类外定义display函数
{
   cout<<"num:"<<num<<endl;
   cout<<"name:"<<name<<endl;
   cout<<"sex:"<<sex<<endl;
}
```

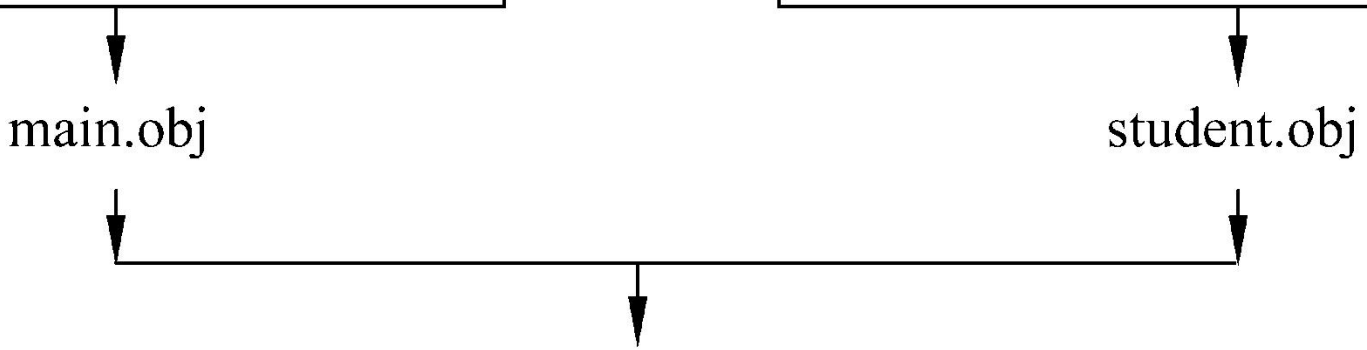
```
//main.cpp                    //主函数模块
#include <iostream>
#include "student.h"          //将类声明头文件包含进来
int main( )
{
   Student stud;              //定义对象
   stud.display( );           //调用stud对象的display函数
   return 0;
}
```

主模块 main.cpp

```
#include <iostream>
#include  "student.h"
void main( )
{
   …
}
```

成员函数定义文件 student.cpp

```
#include <iostream>
#include  "student.h"
void Student: :display(  )
{
   …
}
```



如果一个类声明多次被不同的程序所选用，每次都要对包含成员函数定义的源文件进行编译，这是否可以改进呢？

可以把第一次编译后所形成的目标文件保存起来，以后在需要时把它调出来直接与程序的目标文件相连接即可。

这和使用函数库中的函数是类似的。

这也是把成员函数的定义不放在头文件中的一个好处。

类库包括两个组成部分：

(1)类声明头文件；

(2)已经过编译的成员函数的定义，它是目标文件。

用户只需把类库装入到C++编译系统所指定的目录下，并在程序中用#include命令行将有关的类声明的头文件包含到程序中，就可以使用这些类和其中的成员函数。

例8.2 在主函数引用对象成员

```
//（1）使用成员运算符引用对象成员
#include <iostream>
using namespace std;
class Time
{
public:
    int hour;
    int minute;
    int sec;
};
```

```
int main( )
{
    Time t1;
    cin>>t1.hour;
    cin>>t1.minute;
    cin>>t1.sec;
    cout<<t1.hour<<":"<<t1.minute<<":"<<t1.sec<<endl;
    Time t2;
    cin>>t2.hour;
    cin>>t2.minute;
    cin>>t2.sec;
    cout<<t2.hour<<":"<<t2.minute<<":"<<t2.sec<<endl;
    return 0;
}
```

```
//（2）使用函数引用对象成员
int main( )
{
    void set_time(Time &);        //函数声明
    void show_time(Time &);       //函数声明
    Time t1;
    set_time(t1);
    show_time(t1);
    Time t2;
    set_time(t2);
    show_time(t2);
    return 0;
}
```

```
void set_time(Time & t)//函数形参t是引用变量
{
   cin>>t.hour;              //输入设定的时间
   cin>>t.minute;
   cin>>t.sec;
}

void show_time(Time & t)      //函数形参t是引用变量
{
   cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl;
}
```

```
//（3）使用带默认参数的函数引用对象成员
int main( )
{
    void set_time(Time&,int hour=0,int minute=0,int sec=0);
    void show_time(Time&);
    Time t1;
    set_time(t1,12,23,34);   //通过实参传递时、分、秒的值
    show_time(t1);
    Time t2;
    set_time(t2);                //使用默认的时、分、秒的值
    show_time(t2);
    return 0;
}
```

```
void set_time(Time& t,int hour,int minute,int sec)
{
   t.hour=hour;
   t.minute=minute;
   t.sec=sec;
}

void show_time(Time& t)
{
   cout<<t.hour<<":"<<t.minute<<":"<<t.sec<<endl;
}
```

例8.3 含成员函数的类

```
#include <iostream>
using namespace std;
class Time
{
public:
    void set_time( );                    //公用成员函数
    void show_time( );                   //公用成员函数
private:                                 //私有数据成员
    int hour;
    int minute;
    int sec;
};
```

```
int main( )
{
   Time t1;          //定义对象t1
   t1.set_time( );
   t1.show_time( );
   Time t2;          //定义对象t2
   t2.set_time( );
   t2.show_time( );
   return 0;
}
```

```
void Time::set_time( )       //类外定义set_time函数
{
   cin>>hour;
   cin>>minute;
   cin>>sec;
}

void Time::how_time( )   //类外定义show_time函数
{
   cout<<hour<<":"<<minute<<":"<<sec<<endl;
}
```

注意

(1) 在主函数中调用两个成员函数时，应指明对象名(t1,t2)。

(2) 在类外定义函数时，应指明函数的作用域(如void Time::set_time( ))。

在成员函数引用本对象的数据成员时，只需直接写数据成员名。

(3) 应注意区分什么场合用域运算符“::”，什么场合用成员运算符“.”，不要搞混。

例8.4 找出一个整型数组中元素的最大值

```
#include <iostream>
using namespace std;
class Array_max                    //关键的工作是类的设计
{
public:                            //成员函数原型声明
   void set_value( );              //对数组元素设置值
   void max_value( );              //找出数组中的最大元素
   void show_value( );             //输出最大值
private:
   int array[10];                  //整型数组
   int max;                        //max用来存放最大值
};
```

```
void Array_max::set_value( )
{
    int i;
    for (i=0;i<10;i++)
       cin>>array[i];
}

void Array_max::max_value( )
{
    int i;
    max=array[0];
    for (i=1;i<10;i++)
       if(array[i]>max) max=array[i];
}
```

```
void Array_max::show_value( )
{cout<<"max="<<max;}

int main( )
{
    Array_max  arrmax;        //定义对象arrmax
    arrmax.set_value( );
    arrmax.max_value( );
    arrmax.show_value( );
    return 0;
}                                    //主函数很简单
```

小 结

- 面向对象程序设计的特征
  - 抽象、封装、继承、多态
- 类的声明
- 对象的定义
- 对象的引用
- 类的封装和信息隐蔽

思考题

- OOP中信息隐蔽是如何实现的？
- class和struct的联系和区别是什么？
- 对象成员的访问方法与struct变量、数组元素等有何异同？

作　　业

- 课本习题
- P262：5，6;