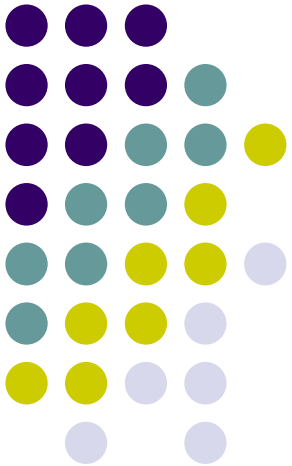
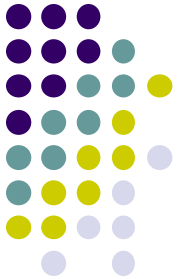
# 类

吴清锋

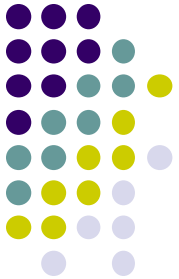


# 编程语言为何要有类型？



- 现实问题向虚拟世界的映射，需要对现实的数据进行有效地表征；
  - 类型的表征
  - 数据值的表征
- 除了基本类型外，编程语言提供给用户多大程度的“自主权”来自己创建类型就很有意义。

# 引言



- 之前,学习了:
  - 面向过程的编程
  - 使用标准库类型
- C++功能强大表现之一：在于提供了**类编程机制**，允许用户自己定义自己的类类型
- **学习时的关键:**
  - **掌握如何定义类(掌握语法)只是初级;**
  - **要回顾之前的一些内容（如：内联、重载等），这些内容在类中的应用**
  - **更重要的是,如何从实际问题出发,将问题域用类来表现.即通过定义合适类型来对应所要解决的问题中的各种概念.**

# 回顾与练习：

- 结构体，将不同类型的或是相关联的数据元素组合在一起成为一个整体。

- 完成，声明结构体类型
  1、学生结构类型

  2、日期类型

# 结构体类型的应用：学生结构体

返回

- 声明结构体

```
struct Student {
  int id;
  char name[10];  //string name;
  float score;
  char sex;
 };
```

- 定义结构体变量 **struct Student  s1,s2,s3;**
- 应用结构体变量 **s1.id=202001;**
  **s1.score=75;**
  **s2=s1;**

# 结构体类型的应用：日期结构体

返回

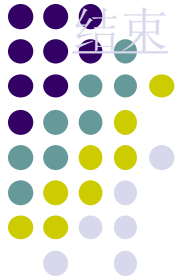
- 声明结构体

```
struct  DATE {
  int year;
  int month;
  int day;
};
```

- 定义结构体变量 **struct DATE  d1,d2,d3;**
- 应用结构体变量 **d1.year=2020;**

在使用上：视为一个变量，**main**函数中直接访问成员
存在风险：重要的属性值可以被外界任何人随意访问和修改。

- 判断是否为闰年：**bool  IsLeapyear(struct DATE x)**
  **{ ….. }**



# 提纲

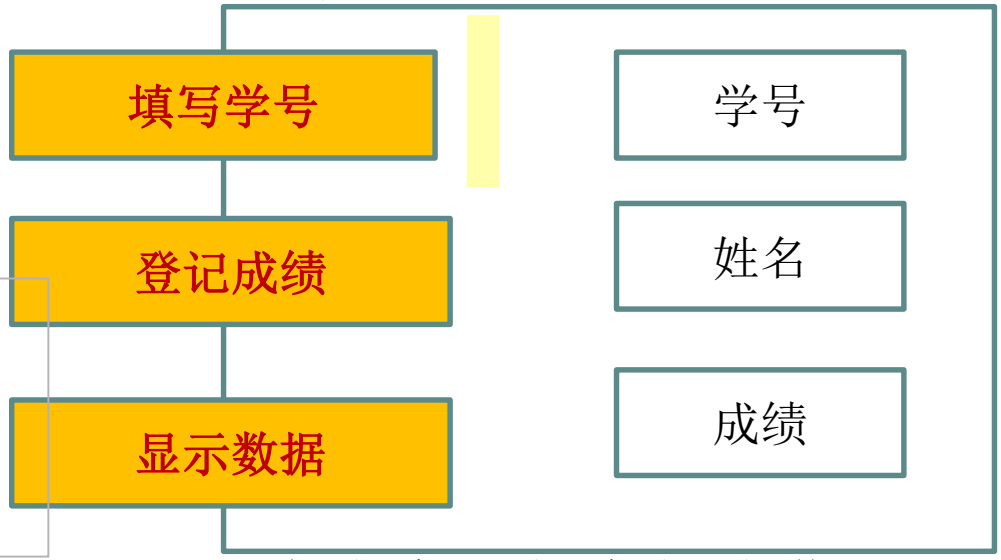
- 从结构到类
- 数据成员
- 成员函数
- 类对象定义
- 对象成员的操作
- this 指针
- 多文件编程

# 从结构体到类

返回

- 包含类和对象的简单程序
  - 例子

- 比较：声明结构体和类的差别例子
  - 相同点：都是定义了新类型
  - 不同点




学生类实现信息的隐藏

红色区域是学生类暴露给外界用以访问对象内部属性的接口
用户只能通过提供的接口访问访问对象允许外界了解的信息。

# 类的声明

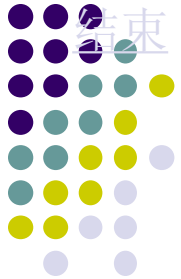
类由三部分组成：类名、数据成员和成员函数。

返回

- 类类型声明的一般形式

```
class 类名
{ private:
    私有的数据和成员函数;
  public:
    公用的数据和成员函数;
};
```

- 注释：
  - 类的声明与结构体声明类似，仅仅是定义了类型（不占空间）
  - private和public称为成员访问限定符；
    没有谁先谁后的硬性要求；**并且可以分别出现多次；**
    **基本原则：归类处理**
  - **不一定会有private和public，**缺省下默认为私有的；
- 一个例子



# 提纲

- 从结构到类
- **数据成员**
- 成员函数
- 类对象定义
- 对象成员的操作
- this 指针
- 多文件编程

# 类声明中的数据成员

- 类的数据成员说明对象的**内部本质特征**;
- 在类定义中，需要对数据成员的名字和类型进行说明

  例子
- 注意：在类中，描述数据成员一般不能给他们赋初值

```
class A  {
    int x=0;  //普通成员                    //ERROR
    const double y=0.0; //const成员          //ERROR
      …
  } ;
```

类似结构体的数据成员的描述一样【有类型无空间】

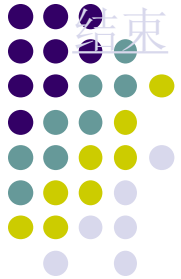
# 类声明中一个特殊情况：前向声明


- 类应该先声明，后使用.
- 如果需要在某个类的声明之前，引用该类，则应进行**前向引用声明**。即：
  - **先暂时声明一个类名但没有具体定义类中的各个成员；**
- 声明一个数据成员类型时，如果未见到相应的类型定义或相应的类型未定义完，则该数据成员的类型一般是类型的指针或引用。

```
class B;  //前向声明
class A
{  public:
    void f(B *b);
};
class B
{  public:
    void g(A a);
};//类的前向声明一般用来编写互相依赖的类
```

```
class A; //数据成员定义注意事项
class B
{  A a;  //Error
   B b;  //Error
   A *p;   //OK
   B *q;    //OK
   A &aa;   //OK
   B &bb;   //OK
};
```

- 即使有了前向声明，还是必须只有在正式声明后,才可以定义类对象



# 提纲

- 从结构到类
- 数据成员
- **成员函数**
- 类对象定义
- 对象成员的操作
- this 指针
- 多文件编程

# 成员函数：概述

- 类的数据成员说明对象的特征，而**成员函数决定对象的操作行为**。
- 成员函数是对封装的数据进行操作的**唯一途径**。
- 成员函数的性质
  - 本质：是函数的一种，用法和作用和一般函数基本上是一致的，也具有返回值和函数类型；
  - 与一般函数的区别是：
    - 它属于一个类的成员，出现在类体中，并且可以指定成员访问限定符(private,public等)；
    - 在使用时，要注意调用它的权限以及它的作用域；

# 成员函数定义位置

- 类的成员函数有两种定义位置：
  - 内联定义（在类定义体内）
    成员函数的定义直接写在类中
    适用于成员函数规模较小的情况，默认为inline
  - 在类外定义
    类定义体中只写成员函数的原型说明，成员函数的定义在类外

# 成员函数定义:在类内定义，即inline成员函数

返回

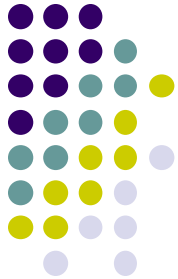
- 回顾：内联函数的特性？（见附录）
- inline成员函数
  - 前提：在类体中定义的成员函数，程序规模比较小且不包括循环等控制，C++会自动将它们作为内联函数来处理，即:对类内定义的成员函数,可以省略inline.
  - 例子
- 值得注意的是，如果成员函数不在类体内定义，而在类体外定义，系统并不把它默认为内联函数。此时，若想将这些成员函数指定为内联函数，应当用inline作显式声明；

返回

# 成员函数定义:在类外定义

- 在类外**定义**成员函数
  在类体中只写成员函数的声明，而在类的外面进行函数定义；
  - 与类体中直接定义函数差别：必须在函数名前加类名，即类名::；
  - 成员函数必须先在类体中作原型声明，然后在类外定义
  - 例子

# 为何要把成员函数在类外定义呢？

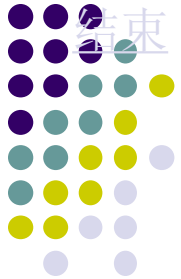


- 思考：为什么要把成员函数提取出来？

减少类体长度，使类体清晰，而且有助于把类的接口和类的实现细节分离

# 作业一

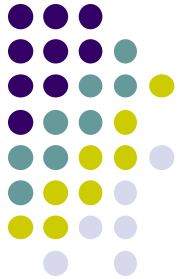
- 定义描述书的类，包括书名、价格、出版社、出版日期等。



# 提纲

- 从结构到类
- 数据成员
- 成员函数
- 类对象定义
- 对象成员的操作
- this 指针
- 多文件编程

# 什么是类对象



- 类对象是类类型的变量；
- 先有类再有对象；
- 一个对象就是一个实际问题域中的一个实体。它包含了数据结构和所提供的相关操作功能，即，对象是属性和服务的封装体。
  - 对象的属性用于描述对象的静态数据特征。对象的属性可用系统的或用户自定义的数据类型来表示。
  - 对象的服务用于描述对象的动态特征，它是定义在对象属性基础上的一组操作方法（method）的集合。
- 建议：用书和具体的一本书（C++编程教材）来对应

# 对象的定义1——静态对象

- 定义对象

①先声明类类型，然后再定义对象

②在声明类类型的同时定义对象

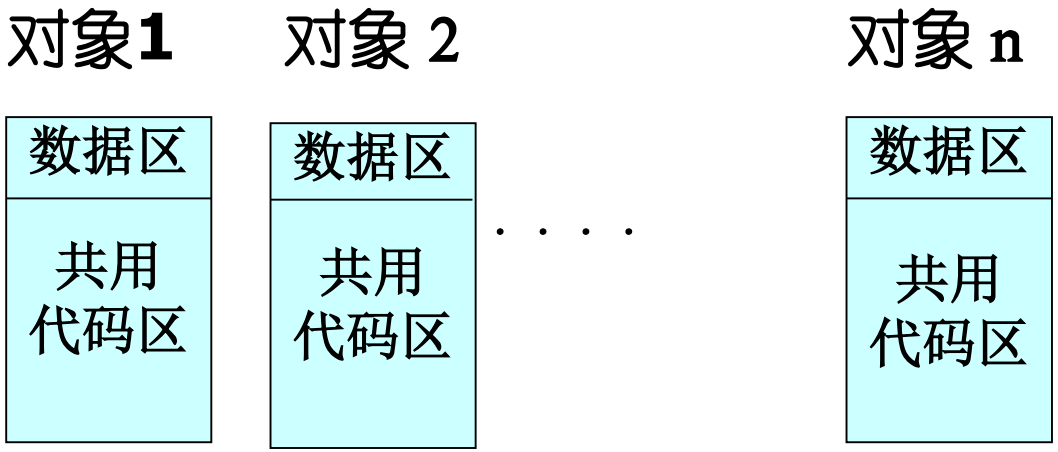
③不出现类名，直接定义对象

② class Student
{ public:
    void display() {….};
  private:
    ….}stud1,stud2;

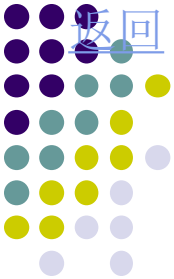
③ class
{ privae:
    …
  public:
    …
} stud1,stud2;

不可再扩充

# 对象的内存




- 系统为每一个对象分配了全套的内存。数据区存放数据成员，代码区存放成员函数。
- 区别同一个类的各个不同的对象的属性是由数据成员决定的，不同对象的数据成员的内容是不一样的；
- 行为（操作）是用函数来描述的，这些操作的代码对所有对象都是一样的。



# 对象的定义2——动态对象

- 动态对象的定义【类类型】
  - 在声明类的基础上，采用new和delete来创建和删除对象
  - **单个动态对象的创建与撤销**

    A *p;　//p为一个指针变量
    p=new A; //创建一个A类的动态对象，返回地址；p指向
    delete p;

  - **动态对象数组的创建与撤销**

    A *p;
    p=new A[100];
    delete [ ]p;

# 提纲

- 从结构到类
- 数据成员
- 成员函数
- 类对象定义
- **对象成员的操作**
- this 指针
- 多文件编程

返回

# 对象成员的操作方法：概述

- 在类外（类定义之外：即类的声明和实现之外），只能访问public成员，而不能访问private成员(对于，private成员只能通过该类的公有成员函数来访问它们)
  - 例子

- 对象成员的引用，形式：
  - 通过对象名和成员运算符访问对象中的成员；
  - 通过指向对象的指针访问对象中的成员；
  - 通过对象的引用访问对象中的成员

返回

# 对象成员的操作（1）

- **通过对象名和成员运算符访问对象中的成员**
  - 一般形式：对象名.成员名 例子
  - “.”是成员运算符，用来对成员进行限定，指明所访问的是哪一个对象中的成员（或数据成员或成员函数）

- **对比：**成员函数中访问数据成员时，不需要加点操作符（这是因为这些变量**默认属于**当前对象的数据成员，不需再额外指明）

# 对象成员的操作（2）

- 通过指向对象的指针访问对象中的成员
  - 例子

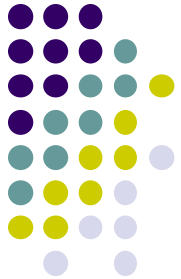
```
Tdate t,*p;   //定义对象t和指向Time类的指针变量p
p=&t;         //建立指向关系

p->set(11,12,2016);  //或是  (*p).set(11,12,2016);
```

# 对象成员的操作（3）

- 通过对象的引用来访问对象中的成员
  - 若：为一个对象绑定了一个引用。由于，他们共占同一段存储单元，实际为同一个对象，只是用不同的名字表示而已。则：可以通过引用来访问对象中的成员
  - 例
    已声明Time类，且有下列定义语句：

```
Time t1;                //定义对象t1
Time &t2=t1;            //声明Time类引用
t1.setTime(11,25,30);
t2.setTime(15,25,30);  //通过引用来调用成员函数
```

# 你是如何声明一个类的？



以声明一个矩形类为例

抽象性

- 先考虑：问题域中有什么？
  **【抽取本质特性，加以描述】**
- 然后分析：哪些是数据，有哪些操作？
  【以**属性**表示静态特征，以**行为**表示功能或服务】

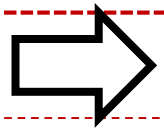
封装性

- 再思考：哪些是想隐藏的，哪些是公开的？
- 为何是想隐藏呢？
- 隐藏的部分，是将如何得到访问的？

# 练习下

静态特征：

- float width;
- float length;



封装成一个整体？

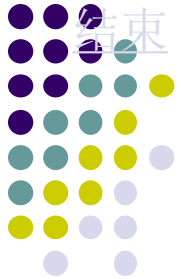
哪些要隐藏？

动态特征：

- SetData( );
- CalculateArea( );
- getWidth( );
- getLength( );

容易忽视掉

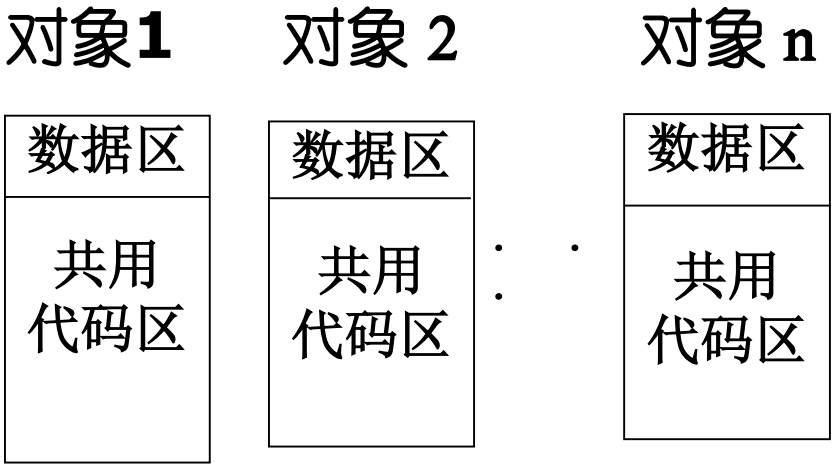
(返回值，易如何输出？)



# 提纲

- 从结构到类
- 数据成员
- 成员函数
- 类对象定义
- 对象成员的引用
- **this 指针**
- 多文件编程

# this指针（1）




返回

```
1.  class A {
2.    public:
3.      void f( );
4.      void g(int i) { x=i;  f( ); }
5.    private:
6.       int x,y,z;
7.  };
```

- A a,b,c;

  a.g(1);

  b.g(10);

  c.g(1000);

探讨背后机制：类的成员函数，如何知道要对哪一个对象进行操作？

# this指针（2）

- 在每一个类成员函数的形参表中都有一个**隐含的**指针变量this,该指针变量的类型就是成员函数所属类的类型;
- 当程序中调用成员函数时，this指针变量被自动初始化为发出函数调用的对象的地址;
- this指针变量是隐含的，但是在成员函数的函数体内可以使用this指针变量;
- 例：

```
class Example
 { private: int m;
   public:  void setvalue(int arg1) {m=arg1;}
} s;
 s.setvalue(100);
```

语句的作用：通过调用setvalue成员函数，将对象s的数据成员m的值赋成100，而不会应用于其他对象.其原因是成员函数的原型实际：

```
void setvalue(Example *this,int arg1)
  {this->m=arg1; }
```

```
setvalue(&s,100);
```

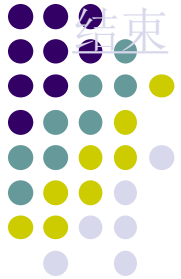
成员函数实际得到了对象s的地址，并将其传递给该函数的隐含指针变量this，函数中通过this指针给对象s的数据成员m赋值

# this指针（3）

- 注意：
  - 关于this指针变量的**参数传递**，都是由**编译系统自动**实现的，程序员无须人为在形参中增加this指针或是将对象的地址传递给this指针；
  - 在需要时，可以显式地使用this指针。即：
    使用*this表示当前的对象，表示被调用的成员函数所在的对象。有一成员函数，如:

```
bool same_isbn(const Sales_item &rhs) const
  {   return this->isbn==rhs.isbn;    }
```

- **总结：使用this指针，保证了每个对象可以拥有不同的数据成员，但处理这些数据成员的代码可以被所有的对象共享**

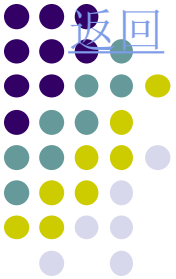


# 提纲

- 从结构到类
- 数据成员
- 成员函数
- 类对象定义
- 对象成员的引用
- this 指针
- 多文件编程

# 多文件编程（1）

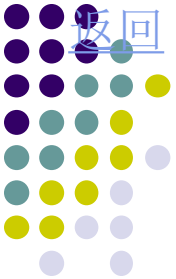
- 在类的编程中，常常：
  - 将所有关于**类的声明**都描述在一个头文件中（类的声明文件），如box.h，文件名和类名相同；
  - 将类中的**成员函数的实现**都描述在另一个编译文件中（类的实现文件）,如box.cpp，文件名和类名相同.
  - 将使用类**的代码（应用程序）**都描述在其他编译文件中,如main.cpp
- 这三个文件，相互独立又同属于一个程序体；
- 编写步骤：通过建立工程的方式来进行！



# 多文件编程——头文件的编写

- 头文件的编写
  - 在编写程序时，常常包含其他的头文件，有可能使一个头文件被多次包含进同一源文件中。因此在设计头文件时，必须保证多次包含同一头文件不会引起该头文件定义的类或对象被多次定义；
  - 头文件卫士：用于避免在已经见到头文件的情况下重新处理该头文件的内容

```
#ifndef BOX_H
#define BOX_H
…                    //类的声明
#endif
```

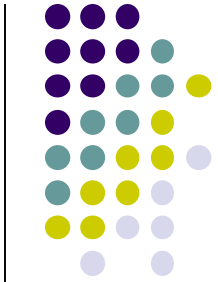


# 多文件编程——类实现与类应用

- 类的实现
  - 在编写时，必须包含自己定义的头文件，否则编译器就不知道该文件中的成员函数是哪个类的；
  - 形式：#include “box.h”
  - 上述书写形式与#include <iostream>有差别：
    - < >意味着为标准头文件
    - “ “意味着非系统头文件
  - 在该文件中，对于各个成员函数的描述，必须加载
    类名::成员函数名(…){ … }
- 类的应用
  - 在编写时，不需要访问类的实现文件，只需要包含自己定义的头文件即可；
  - 形式：#include “box.h”

# 作业二

- 用多文件实现日期类



# 区别：声明结构体与类

返回

```
struct Student
  { int num;
    char name[20];
    char sex;
  };
Student stud1,stud2;
```

---

```
class Student
  { private:
      int num;
      char name[20];
    public:
      void display()
        {cout<<"num:"
          <<num<<endl;
        };
Student stud1,stud2;
```

- C语言中的结构体一般只有数据成员，无成员函数。
- C++语言中的结构可有**数据成员和函数成员(只是有了类，就很少突出成员函数)**

- **结构体是一个没有封装的数据类型。在缺省情况下，结构体中的数据成员都是公有的，因此无法对外界隐藏自己的重要信息和私密信息。同时，外界可随意修改结构体变量中的数据，这样对数据的操作是很不安全的，不能通过结构体对数据进行保护和控制；**
- 若在类的声明中，既不指定private也不指定public，则系统默认为私有的；

# 例子：包含类的C++程序

返回

```
#include <iostream>
using namespace std;
class Student        //声明一个类
{ private:           //类中的私有部分
    int num;         //私有变量
    int score;
  public:            //类中公用部分
    void setdata()   //定义公用函数
     {cin>>num;      cin>>score;
     }
    void display()
     {cout<<"num="<<num<<endl;
     cout<<"score="<<score<<endl;
     };
};
Student stud1,stud2;    //定义对象
int main()
{ stud1.setdata();
  stud2.setdata();
  stud1.display();
  stud2.display();
  return 0;  stud2.score=6;  //ERROR
```

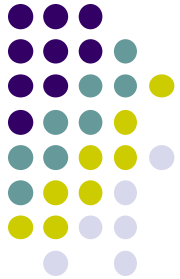
- 编程三步骤：
  ①声明一个类；
  ②定义对象；
  ③对对象进行相关操作。

- 定义类的关键字：class
- C++类中可以包含两种成员：
  - 数据成员（变量）
  - 函数成员：往往是调用数据成员
- 成员根据需要定义成私有或公有的
  - 私有：数据或函数只能被本类中成员函数所调用；
  - 公有：可以被本类中的成员函数调用，也可以被类外的语句所调用；

- 定义对象意味着：
  - 对象占实际存储空间；
  - 对象具有与类相一致的结构和特性；
- 对象成员函数的调用：使用“.”

# 类定义实例

返回

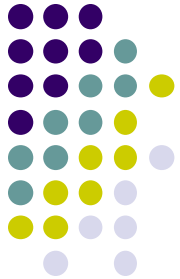


例：定义日期类

```
class Tdate                                  // 定义日期类
{
  public:                                    // 定义公有成员函数
      void Set(int m,int d,int y);  // 设置日期值
      int IsLeapYear();                      // 判是否闰年
      void Print();                          // 输出日期值
  private:                                   // 定义私有数据成员
      int month;
      int day;
      int year;
};      // 类定义体的结束
```

# 函数定义写在类外

返回



```
void Tdate::Set(int m,int d,int y)
                // 置日期值
{    month=m; day=d; year=y;
}
int Tdate::IsLeapYear()    // 判是否闰年
{   return
(year%4==0&&year%100!=0)||(year%400==0);
}
void Tdate::Print() // 输出日期值
{
  cout<<month<<"/"<<day <<"/"<<year<<endl;
}
```

# 在类定义体内定义内联函数

```
class Tdate
{
  public:
    void Set(int m,int d,int y)    // 置日期值
    {
      month=m; day=d; year=y;
    }
  private:
    int month;
    int day;
    int year;
};
```

返回

# 类外，通过对象访问类成员受到类成员访问控制的限制

返回

```
class A {
  private:
    int x;
    void g( )
      {  …//允许访问:x,f,g
         }
  pulbic:
    void f()
      {  ….//类成员函数允许访问:x,f,g
         }
    };
```

```
A a;
a.f();    //OK
a.x=1;    //ERROR
a.g();    //ERROR
```