C++面向对象程序设计
实验指导书

张德慧　梁琛　丁春玲　编

西安邮电学院计算机科学与技术系

软件教研室

2006 年 2 月 20 日

前言

学习计算机程序设计的最好方法是上机实践，只有通过足够的上机编程训练才能获得程序设计的技能。为了提高同学们上机实践练习程序设计的效果，我们编写了这本《C++面向对象程序设计实验指导书》。本实验指导书描述了在 Visual C++6.0 开发环境中用 C++语言开发控制台应用程序(Win32 Console Application)的步骤，并介绍了 Visual C++6.0 中的调试工具和使用方法。

本实验指导书安排了 12 套精心设计的实验，每个实验题目都给出了实验目的、实验基本要求和实验内容。帮助读者掌握面向对象的程序设计方法和技巧，进一步加深对《C++面向对象程序设计》课程主要内容的理解。

由于编者水平有限，错误和不当之处在所难免，在此恳请广大读者批评指正。作者的电子邮件地址是：xazhangdehui@163.com。

编者

2006 年 2 月

目录

Visual C++6.0 开发环境使用入门

一、 点击 File 菜单项，选择 New 命令新建一个项目(new project)， 如图 1 所示。

图 1　文件 File 菜单中的 New 新建命令

出现图 2 的新建对话框，在确定了项目名称，项目类型，项目存放位置三个选项后点击 OK 按钮。

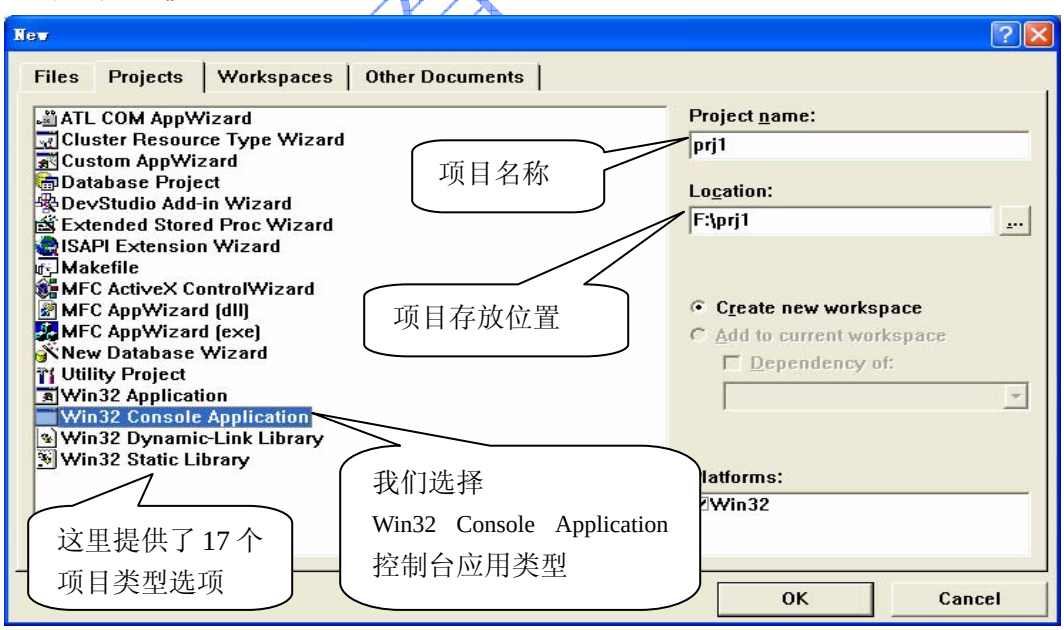

图 2　Visual C++6.0 的新建对话框的项目标签

然后出现如图 3 的对话框，选择 An empty project（空项目），点击 Finish 按钮，

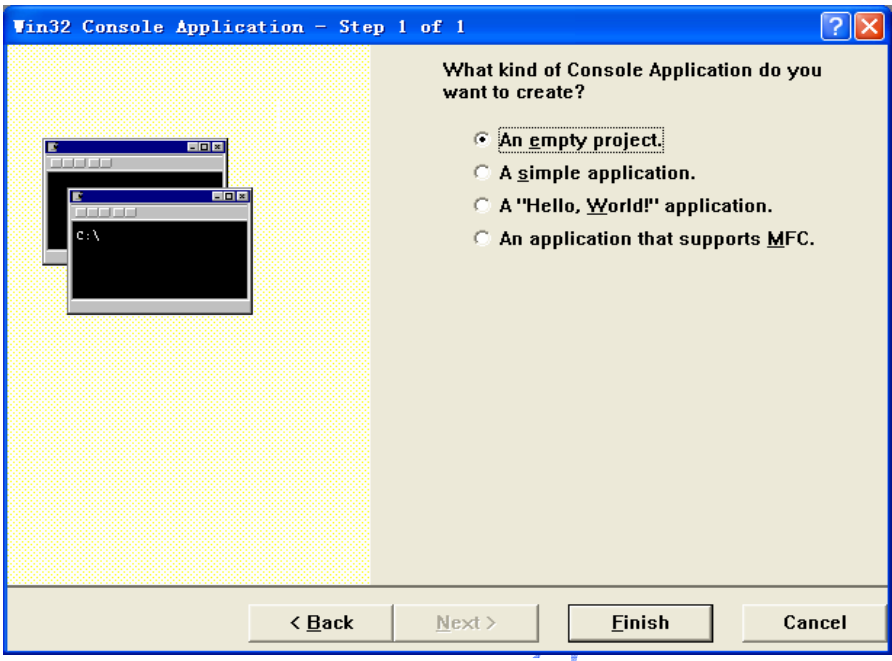

图 3　控制台应用对话框

然后出现如图 4 的新建项目信息对话框，点击 OK 按钮完成项目的创建。

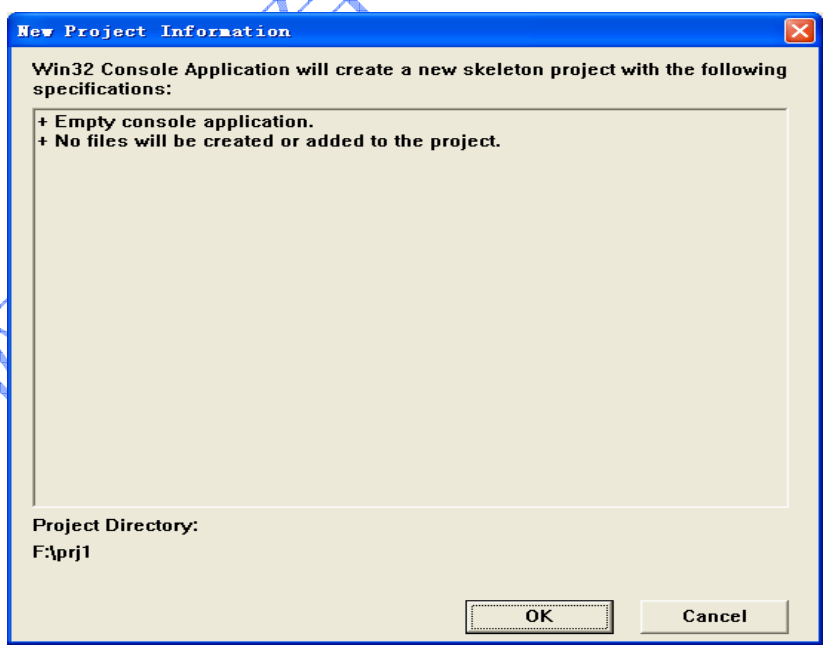

图 4　新建项目信息对话框

二、 点击 File 菜单项，选择 New 命令，新建一个 C++源程序文件，（或者编辑一个已有的 C++源程序文件），出现如下对话框：

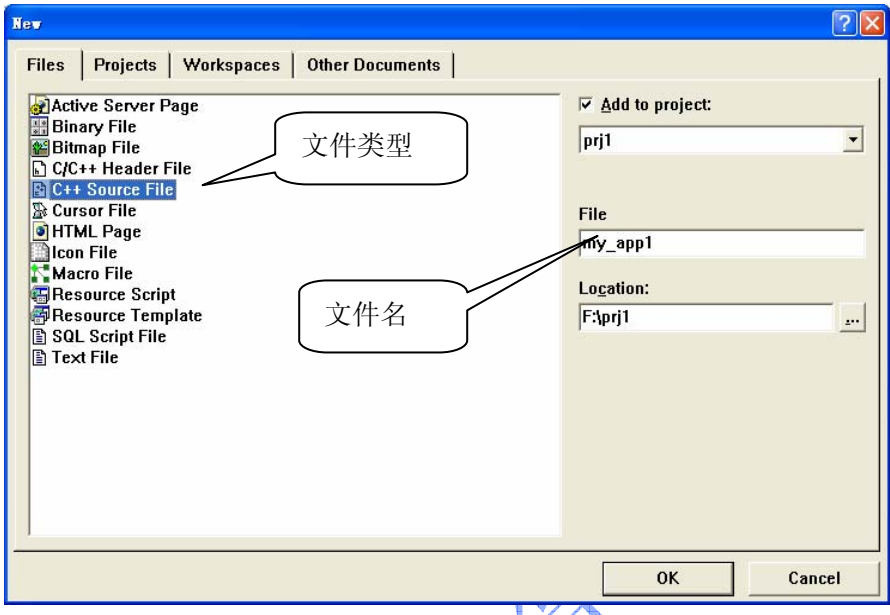

图 5　新建对话框的文件标签

确定文件类型和文件名后，点击 OK 按钮出现文本编辑窗口，在文本编辑窗口输入源程序，如图 6 所示。

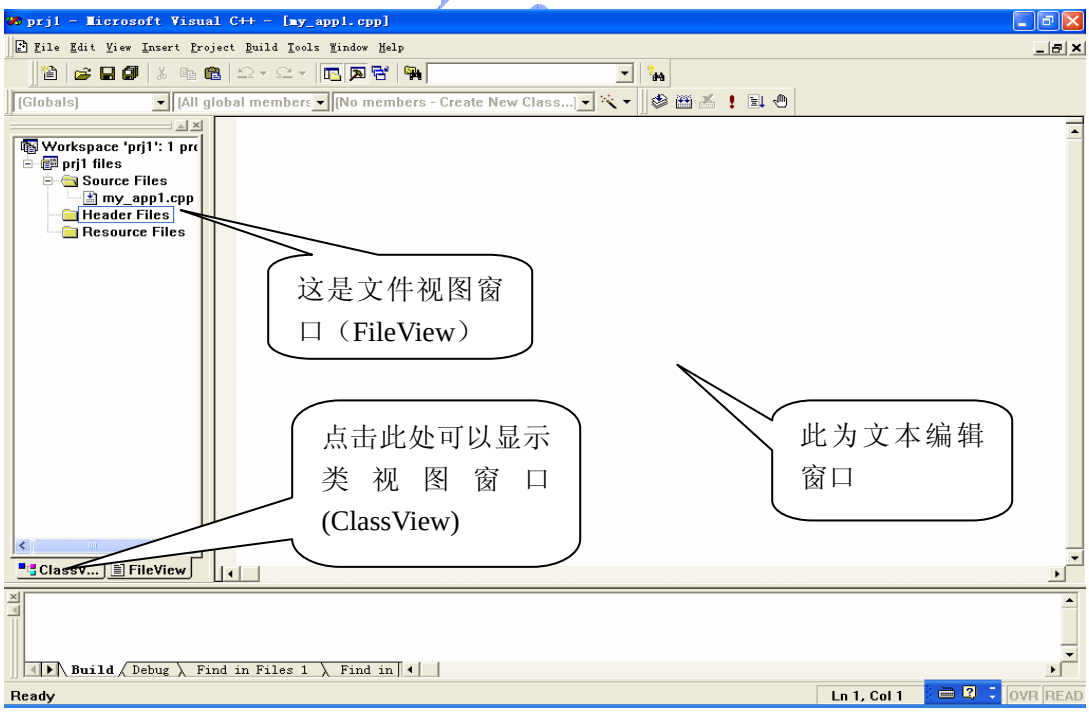

图 6　文本编辑窗口

在文本编辑窗口输入 C++源程序代码，如图 7 所示：

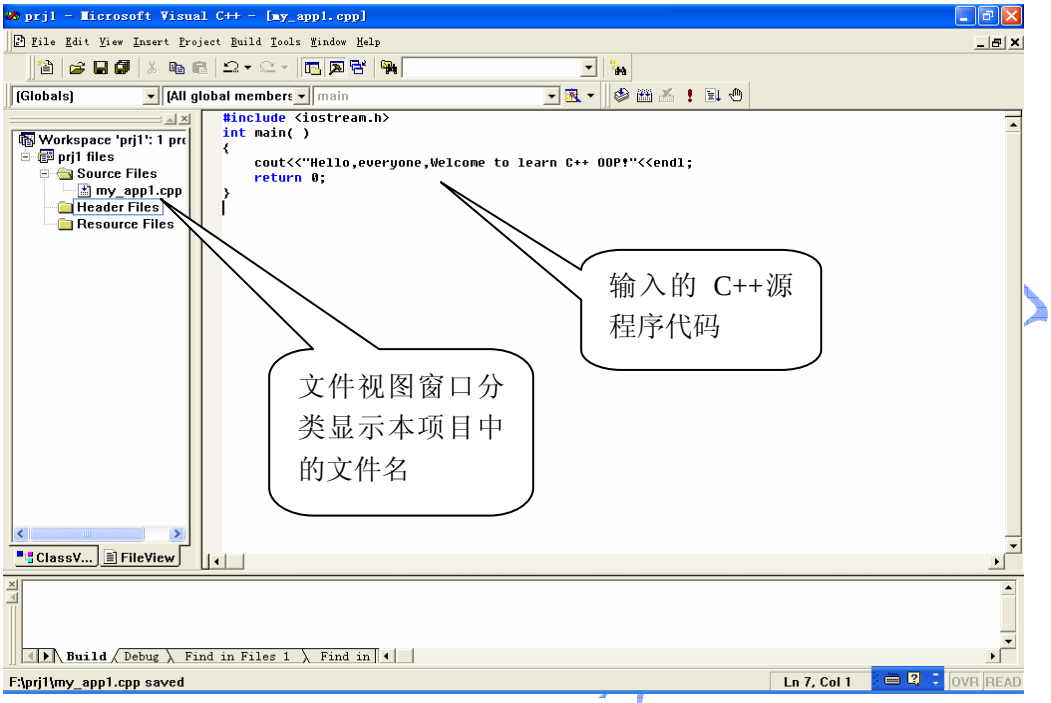

图 7　C++源程序文件的输入和编辑

三、 点击 Build 命令按钮，编译、链接程序

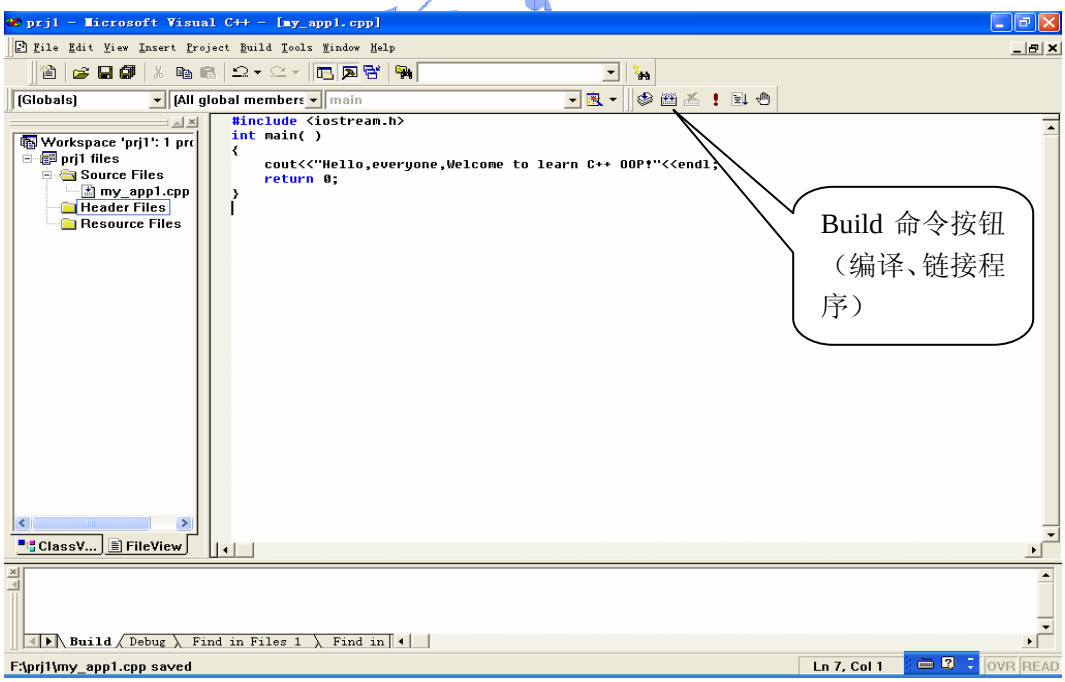

图 8　C++源程序文件的编译、链接

编译、链接完成后，观察编译信息显示窗口出现的提示信息，如果没有出现错误，下一步就试运行程序。

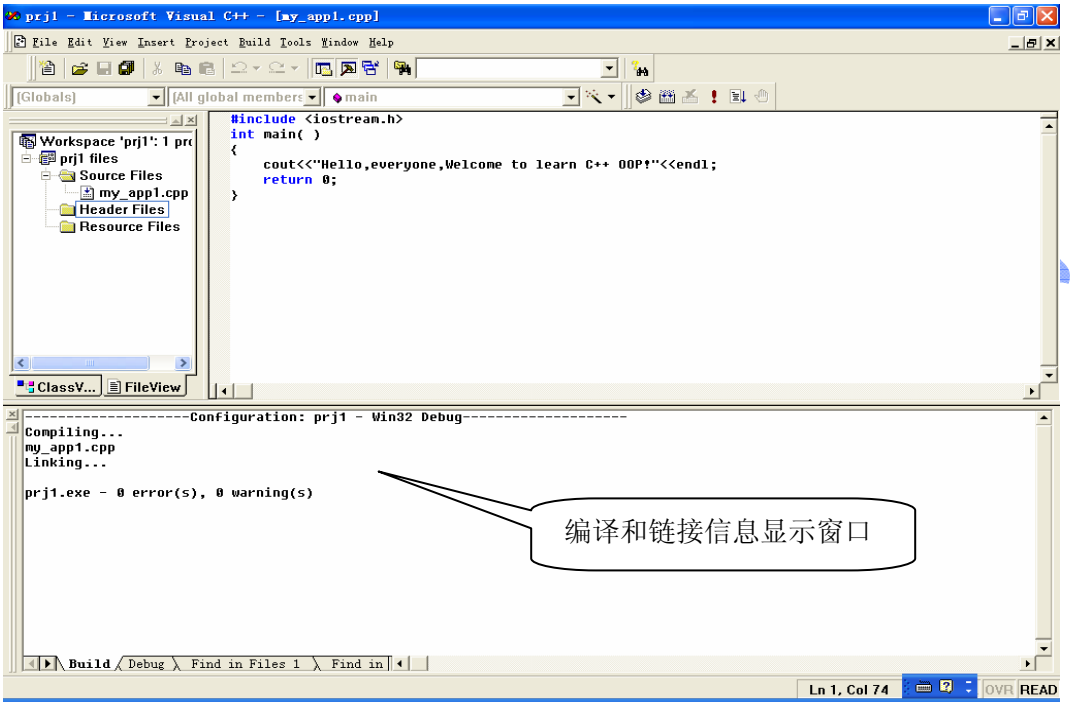

图 9　编译和链接信息显示窗口

四、 如图 10，点击 Execute Program 按钮或者按键盘（Ctrl+F5）， 运行程序

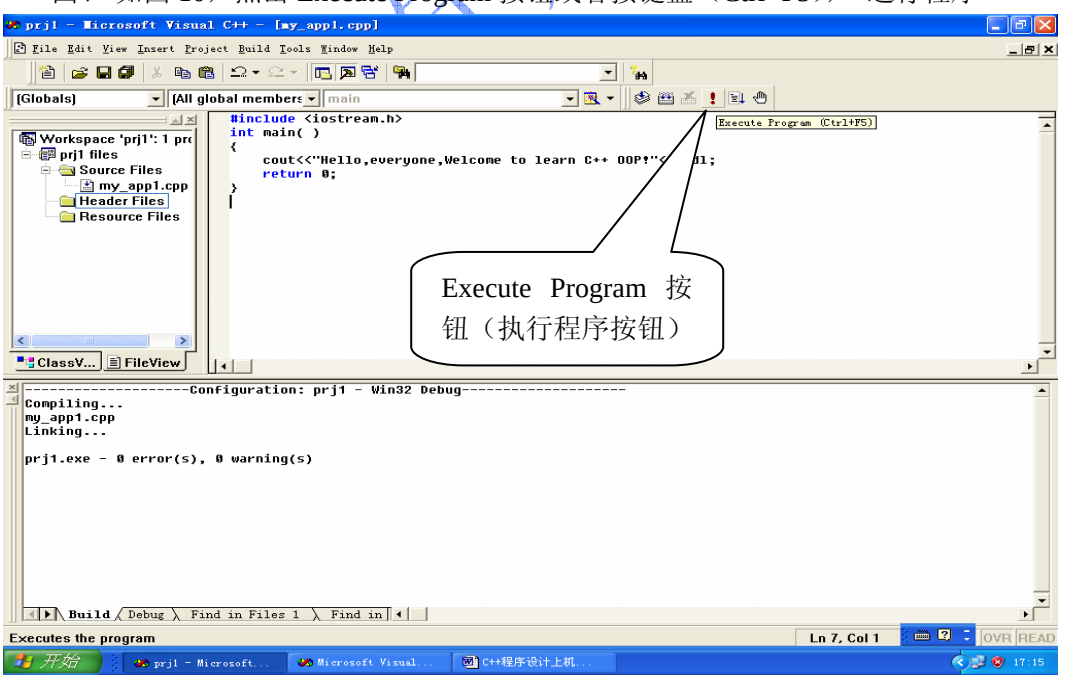

图 10　执行程序

五、 观察程序的运行结果是否正确。如图 11 所示，是上例程序的运行结果：

图 11　程序运行结果输出窗口

实验 1　面向对象软件开发环境使用入门

1. 实验目的

熟悉 Visual C++6.0 开发环境，学会利用 Visual C++6.0 开发环境创建控制台应用程序。

2. 实验要求

(1) 学习编写简单的 C++程序，并掌握 C++程序的基本格式与规范。
(2) 理解 C++程序结构的特点。
(3) 学习 C++程序基本的输入输出操作。

3. 实验内容

(1) 在 Visual C++6.0 开发环境编写一个完整的 C++控制台应用(win32 console application) 程序。

代码如下：

```
#include <iostream>
#include <string>
using namespace std;
int main( )
{
  string user_name;
  cout << "Please enter your first name: ";
  cin >> user_name;
  cout << '\n'
  << "Hello, "
  << user_name
  << " ... and goodbye!\n";
```

```
  return 0;
}
```

(2) 在 Visual C++6.0 开发环境中编译、运行该程序。

(3) 在 Visual C++6.0 开发环境中编辑、编译、运行下面的程序，理解 C++程序结构的特点。

```
#include "iostream.h"
int max(int x, int y); //函数声明
int main( )
{ int a, b, c;
  cin>>a>>b;
  c=max(a ,b);
  cout<<"max= "<<c<<'\n';
}
int max(int x, int y)  //函数定义
{  int z;
   if(x>y)  z=x;
   else z=y;
   return (z);
}
```

(4) 在 Visual C++6.0 开发环境中编辑、编译、运行下面的程序，体会 C++程序中的输入输出流类对象的输入和输出操作。

```
// I/O stream  单行注释(single line comment)
#include <iostream>
using namespace std;
int main( )
{  int i;  float f;
   char s[80];
   cout <<"Enter an integer,float,and string:";
   cin >>i>>f>>s;
   cout <<"Here's your data:"<<i<<' '<<f<<endl<<s<<'\n';
   return 0;
}
```

(5) 在 Visual C++6.0 开发环境中编辑、编译、运行下面的程序，体会 C++程序中 main( )函数的返回值的意义。

```
#include <iostream>
using namespace std;
int main( ) //前面的 int 指定了 main()函数返回值的数据类型
{ int a, b, c;
```

```
    cin>>a>>b;
    if(b==0) return -1; // main()函数返回非0值表示程序遇到错误而结束
    c= a/b;
    cout<<"c= "<<c<<'\n';
    return 0; //程序正常结束，函数main()返回值为0。
}
```

实验 2　简单的 C ++语言程序设计

1. 实验目的

学习并实践 C++语言在非面向对象方面的一些新特征：函数名重载，引用，const 修饰符，new 和 delete 运算符等。

2. 实验要求

(1) 学习 const 修饰符的使用。
(2) 理解函数原型的作用。
(3) 学习并练习函数名重载的方法。
(4) 学习 C++语言中内存动态分配与释放的新方法。
(5) 理解引用的概念，掌握引用的基本使用方法。

3. 实验内容

(1) 编辑并编译下面的 C++程序段，并尝试修改该程序，使其能够通过编译检查。

```
const int model = 90;   // model is a const
const int v[ ]={1,2,3,4}; // v[i] is a const
const int x;              // error: no initializer
void f( )
{
    model =200; // error
    v[2]++;   // error
}
```

(2)在 Visual C++6.0 开发环境中编译并运行下面的程序，理解函数原型的作用。

```
#include <iostream>
using namespace std;
void sqr_it(int *i);  // function prototype 函数原型
int main( )
```

```
{
    int x;
    x=10;
    sqr_it(x);  // compiling time error: type mismatch 类型不匹配
    cout<<"The square of x is "<<x<<'\n';
    return 0;
}
void sqr_it(int *i)
{
    *i=(*i)*(*i);
}
```

由于使用了函数原型，C++语言的编译器能够进行函数参数的类型匹配检查，从而使我们尽早发现错误，显著减少大型程序的排错和调试时间。

(3) 模仿教材中【例 1.10】的程序，将函数 max 重载三次，使它能够分别求两个整数、两个长整型数、和两个双精度浮点数的最大值，并在主函数 main( )中演示这三个函数。

(4) 模仿教材中【例 1.11】的程序，编写一个 C++程序，使用 new 开辟动态存储单元保存你的名字，并显示出你的名字，输出完毕后使用 delete 释放动态存储单元。

(5) 阅读下面的程序，将其中的指针函数参数修改为引用作为参数。

```
#include <iostream.h>
void f(int a[ ],int n, int *max, int *min)
{
    *max=*min=a[0];
    for(int i=1;i<n;i++)
    {
        if(*max<a[i]) *max=a[i];
        if (*min>a[i]) *min=a[i];
    }
}
void main( )
{
    int a[10]={2,5,3,9,0,8,1,7,6,4};
    int max,min;
    f(a,10,&max,&min);
    cout<<"Max: "<<max<<endl;
    cout<<"Min: "<<min<<endl;
}
```

实验 3　封装性：类与对象（1）

1. 实验目的

练习简单类的定义，熟悉类的构造函数、析构函数及其他成员函数、成员变量的定义和构造方法。掌握对象的建立与使用方法。理解常成员函数的概念及应用方法。

2. 实验要求

(1) 学习简单类的定义。
(2) 理解类的构造函数、析构函数及其他成员函数的作用。
(3) 学习对象的建立与使用方法。
(4) 理解常成员函数的概念及应用方法。

3. 实验内容

(1) 下面是一个计数器类的定义，请完成该类的实现，并在 main( )函数中演示该类的应用。

```
class counter{
    int value;
public:
    counter(int number);
    void increment();        //给原值加 1
    void decrement();        //给原值减 1
    int getvalue();          //取得计数器的值
    void print();            //显示计数器的值
};
```

(2) 编译、运行并调试教材中【例 2.7】的程序，理解构造函数和析构函数的作用。
(3) 设计一个程序，定义一个矩形类 Rectangle，它有长 length 和宽 width 两个属性，

有成员函数计算矩形的面积。并在 main()函数中建立矩形类 Rectangle 的对象，显示该矩形对象的面积。

(4) 编译下面的 C++程序段，并尝试修改该程序，使其能够通过编译检查。

```
class Date {
    int d, m, y;
public:
    int day( ) const { return d; }
    int month( ) const { return m; }
    int year( ) const;
    // ...
};
// const 是函数原型的一个组成部分，因此在函数定义部分也要带 const 关键字。
inline int Date::year( ) const
{
    return y++;
}
```

(5) 完成下面的成绩类 Score；再把教材【例 2.7】中的 Student 类加以修改，使其含有类 Score 的对象成员并在 main( )函数中测试 Student 类。

```
class Score{    //Score 类定义
   float computer;
   float english;
   float mathematics;
public:
   Score(float x1, float y1, float z1);
   Score( );
   void print( );
   void modify(float x2, float y2, float z2);
};
```

实验 4　封装性：类与对象（2）

1. 实验目的

学会运用对象数组、对象指针、this 指针、对象作为函数参数、友元（friend）等特征进行程序设计。

2. 实验要求

(1) 学习对象数组、对象指针的使用。
(2) 学习用引用向函数传递参数（对象）。
(3) 学习静态数据成员和静态成员函数的使用。
(4) 学习友元函数的使用方法，理解友元函数与友元的作用。

3. 实验内容

(1) 编写一个 C++程序，在其中创建一个含有 5 个元素的 Student 类的对象数组，并给对象数组成员赋值，然后输出对象数组。输出对象数组时分别使用点（.）运算符和箭头（->）运算符。

(2) 编译、运行并调试下列程序，观察调用函数 sqr_it(a)的前后对象 a 的值是否发生了变化。请问向函数 sqr_it( )传递对象是采用的是哪种传递方式？

```
#include <iostream>
using namespace std;
class samp {
 int i;
public:
  samp(int n) { i = n; }
  void set_i(int n) { i = n; }
  int get_i() { return i; }
};
void sqr_it(samp &o)
{
```

```
  o.set_i(o.get_i() * o.get_i());
  cout << "Copy of a has i value of " << o.get_i();
  cout << "\n";
}
int main()
{ samp a(10);
  sqr_it(a);
  cout << "Copy of a has i value of " << a.get_i();
  return 0;
}
```

(3) 下列Circle类的定义中包含了静态数据成员和静态成员函数。请在主函数mani( )中创建Circle类的3个对象，并调用静态成员函数输出对象的个数。

```
class Circle
{ public:
    Circle(float r)
    {   radius=r;
        ++count;
    }
    ~Circle() { --count; }
    static int num() { return count; } //静态成员函数
  private:
    float radius;
    static int count; //静态数据成员
};
```

(4) 下列程序中使用友元函数计算平面上两点之间的距离，删除程序中的关键字friend看看编译时会产生什么问题？

```
#include <iostream.h>
#include <math.h>
class Point
{  public:
     Point(double xi, double yi) {X=xi; Y=yi; }
     double GetX() {return X;}
     double GetY() {return Y;}
     friend double Distance( Point& a, Point& b);
     private:
        double X, Y;
};
double Distance( Point& a, Point& b)
```

```
{
      double dx=a.X-b.X;
      double dy=a.Y-b.Y;
       return sqrt(dx*dx+dy*dy);
}
int main()
{  Point p1(3.0, 5.0), p2(4.0, 6.0);
    double d=Distance(p1, p2);
    cout<<"The distance is "<<d<<endl;
    return 0;
}
```

下面的程序中使用成员函数 Distance(Point& b)计算平面上两点之间的距离，请你将它与上面的程序进行比较，找出主要区别。

```
#include <iostream.h>
#include <math.h>
class Point
{  public:
      Point(double xi, double yi) {X=xi; Y=yi; }
      double GetX() {return X;}
      double GetY() {return Y;}
      double Distance(Point& b);
      private:
         double X, Y;
};
double Point::Distance(Point& b)
{
      double dx=X-b.X;
      double dy=Y-b.Y;
      return sqrt(dx*dx+dy*dy);
}
int main()
{  Point p1(3.0, 5.0), p2(4.0, 6.0);
    double d=p1.Distance(p2);
    cout<<"The distance is "<<d<<endl;
    return 0;
}
```

实验 5　继承性：派生类

1. 实验目的

练习 C++语言中派生新类的方法，理解 C++语言实现继承的机制，体会继承带来的好处。

2. 实验要求

(1) 学习类的继承，能够定义和使用派生类。
(2) 理解基类与父类的关系。
(3) 熟悉公有派生和私有派生的访问特性。
(4) 了解虚基类在解决二义性问题中的作用。

3. 实验内容

(1) 已知下面的 C++程序框架，请按照注释中的提示补充细节，并编译、运行该程序。

```
#include <iostream.h>
class planet {
protected:
    double distance;    // miles from the sun
    int revolve;    // in days
public:
    planet(double d, int r)
    { distance=d;
      revolve=r;
    }
};
```

```
class earth: public planet{
    double circumference;  // circumference of orbit
public:
    /* Create earth(double d,int r). Have it pass the distance and days
of revolution back to planet. Have it compute the circumference of the
orbit. (Hint: circumference=2*r*3.1416)  */
    // Create a function called show( ) that displays the information.
};

int main( )
{
    earth ob(93000000,365);
    ob.show( );
    return 0;
}
```

(2) 从下面的基类 Person 派生出一个 Student 类，并在 main()函数中测试这个类。编译并运行你的 C++程序，使用单步运行程序的调试方法观察基类 Person 和派生类 Student 类的构造函数和析构函数的执行顺序。并请在你的练习本上记录这种执行顺序。

```
#include <iostream>
using namespace std;
class Person
{
public:
  Person(const char* s)
  { name = new char[strlen(s)+1]; strcpy(name, s); }
  ~Person() { delete [] name; }
protected:
  char* name;
};
```

(3) 公司内部有 4 种雇员（employee）：经理（manager），秘书（Secretary），技术员（technician），推销员（salesman）。对公司内部雇员进行概念建模，可得如图 12 类层次关系。

请使用 C++语言实现该类层次体系。

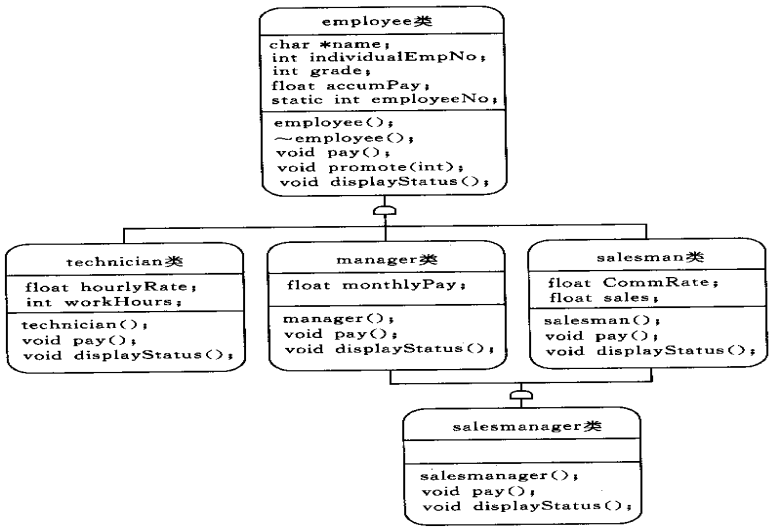

图 12 公司内部雇员的类层次关系

实验 6　多态性：运算符重载

1. 实验目的

练习 C++语言中重载运算符的方法，体会运算符重载带来的方便。

2. 实验要求

(1) 掌握运算符重载的基本方法。
(2) 学习重载运算符为成员函数的方法。
(3) 学习重载运算符为友元函数的方法。

3. 实验内容

(1) 将运算符+和-重载为 complex 类的成员函数。

```
#include<iostream.h>
class complex                    //复数类声明
{
public:                          //外部接口
    complex(double r=0.0,double i=0.0) //构造函数
    {real=r;imag=i;}
    complex operator + (complex c2); //+重载为成员函数
    complex operator - (complex c2); //-重载为成员函数
    void display( );             //输出复数
private:                         //私有数据成员
    double real;                 //复数实部
    double imag;                 //复数虚部
};
```

完成该类的实现并编译运行该程序。

(2) 将运算符+和-重载为 complex 类的友元函数。

```
#include<iostream.h>
class complex
{
public:
    complex(double r=0.0,double i=0.0)
    { real=r; imag=i; } //构造函数
    friend complex operator + (complex c1,complex c2);
    //重载运算符+为友元函数
    friend complex operator - (complex c1,complex c2);
    //重载运算符-为友元函数
    void display(); //显示复数的值
private:    //私有成员
    double real;
    double imag;
};
```

完成该类的实现并编译运行该程序。

(3) 完成下列的 String 类，并在主函数 main( )中测试它。

```
class String
{
public:
    String(const char *str = NULL);    //  constructor
    String(const String &other);       //  copy constructor
    ~ String(void);                    //  destructor
    String & operate =(char *str);
    String & operate =(const String &other);//  重载=运算符
    int operator==(String &other);     //  重载==运算符
    int operator==(char *str);
private:
    char    *m_data;                   //  used for storing the string
    int     length;
};
```

实验 7　多态性：虚函数

1. 实验目的

练习使用C++语言中的虚函数实现运行时多态性的方法。利用多态性设计简单可靠、灵活、易维护的程序。

2. 实验要求

(1) 学习由继承和虚函数获得多态性的方法。
(2) 理解静态联编和动态联编。
(3) 进一步熟悉类的设计。

3. 实验内容

(1) 设计一个描述正方形的类 Square，并具有计算面积的成员函数 GetArea( )。从 Square 类派生一个矩形类 Rectangle，Rectangle 也使用 GetArea( )函数计算面积。

头文件 SquareRectangle.h 源代码如下：

```
//SquareRectangle.h
#include<iostream.h>
#include<math.h>
class Square
{
  protected:
      int edge;
  public:
  virtual  double GetArea();
  Square(int edg);
  ~Square( )
  { }
};
Square::Square(int edg)
```

```
{ edge=edg;  }
//求正方形的面积
double Square::GetArea()
{
    double result;
    result=edge*edge;
    return result;
}
//下面的全局函数是专门为了调用虚函数实现运行时的多态性而设计的。
double  objGetArea(Square* base);
class Rectangle:public Square
{
 protected:
    int  length;
    //它的宽由正方形类 Square 的成员提供
public:
    double GetArea();
    Rectangle(int edg,int leng);
    ~Rectangle(){  }
};
double Rectangle::GetArea()//求矩形的面积
{
    double result;
    result=length*edge;
    return result;
}
Rectangle::Rectangle(int edg,int leng):Square(edg)
{
    //调用基类的构造函数 Square(edg)使 edge=edg;
    length=leng;
}
double objGetArea(Square* obj)
{
    return obj->GetArea();
}
```

在 SquareRectangle.cpp 中给出主函数 main()的源代码：

```
#include<iostream.h>
#include<math.h>
```

```
#include<SquareRectangle.h>
int main()
{
    Square  Squa1(100); //调用 Square::Square(int)初始化
    Rectangle Rect(200,300);//调用 Rectangle::Rectangle(int,int)初始
化
    //比较下面两个语句
    cout<<"正方形 Squa1(100)的面积为："<<Squa1.GetArea()<<endl;
    //静态联编，输出为：10000
    cout<<"正方形 Squa1(100)的面积为："<<objGetArea(&Squa1)<<endl;
    //比较下面两个语句
    cout<<"矩形 Rect(200,300) 的面积为：:"<<Rect.GetArea()<<endl;
    //静态联编，输出为 60000
    cout<<"矩形 Rect(200,300) 的面积为：:"<<objGetArea(&Rect)<<endl;
    //动态联编，输出为 60000
    return 0;
}
```

注意：如果在调用虚函数时使用点运算符“.”，编译时仍将采用静态联编，不能实现运行时多态性。所以我们在调用虚函数时，应该采用指针或引用。就像是上例中的全局函数 objGetArea(Square* obj)一样，编译系统会根据运行时的具体情况确定是哪一个类的对象在调用相应版本的虚函数。

另外，参数传递时要注意一一对应，在调用 objGetArea(Square*)时，一定要在对象 Squa1 和 Rect 前面加&，表示是对象 Squa1 和 Rect 的地址，这样才与指针相匹配。

思考题：除了用全局函数来调用虚函数外，大家还能用别的方法吗？试编写一例。

实验 8　函数模板和类模板

1. 实验目的

练习 C++中函数模板、类模板的创建和使用方法。

2. 实验要求

(1) 理解模板的作用。
(2) 学习函数模板及其声明方法，掌握模板函数及其生成方法。
(3) 学习函数模板的两种不同的实例化方法。
(4) 学习类模板的声明与使用方法。

3. 实验内容

(1) 编写一个函数模板，实现两个变量交换。

(2) 编写一个函数模板 sort 实现选择法排序的功能，并在主函数 main( )中分别实现对 int 整型数组和 double 型数组的排序。

(3) 编译并运行下面的程序。尝试将其中显式实例化修改成隐式实例化后再进行编译，观察会出现什么现象？为什么会出现这种现象？

```
#include <iostream>
using namespace std;
template <class T>  T min(T a, T b)
{
    return (a<b)?a:b;
}
int main( )
{
    double dobj1=1.1, dobj2=2.2;
    char cobj1='c', cobj2='W';
```

```
    int i=12,j= 68;
    cout<<min<int>(i, cobj1 )<<endl;   //显式实例化
    cout<<min(dobj1, dobj2 )<<endl;    //隐式实例化
    cout<<min<char>(cobj2, j )<<endl;  //显式实例化
    return 0;
}
```

(4) 以下是一个整数栈类的定义：

```
const int SIZE = 90;
class Stack
{
public :
    Stack();
    ~Stack();
    void push(int n);
    int pop();
private:
    int stack[SIZE];
    int tos;
};
```

编程实现一个栈类模板（包括其成员函数定义）。在应用程序中创建整数栈、字符栈和浮点数栈，并提供一些数据供进栈、出栈和显示操作。

实验 9　C++标准库（1）

1. 实验目的

练习 C++标准库中标准容器：链表，队列，向量等的简单应用，学会类库的分析、使用方法。

2. 实验要求

(1) 学习类库的分析、使用方法。
(2) 了解标准模板库 STL 中基本组件的作用。
(3) 学习标准模板库 STL 中向量模板的简单应用。
(4) 学习标准模板库 STL 中链表的简单应用。
(5) 学习标准模板库 STL 中队列模板的简单应用。

3. 实验内容

(1) 请模仿教材中【例 10.9】的程序，使用向量类模板 vector 实例化一个保存整数的向量。

(2) 请模仿教材中【例 10.10】的程序，使用线性表类 list 模板实例化一个保存整数的链表。

(3) 请模仿教材中【例 10.11】的程序，通过实例化标准库中的映射类模板 map 建立了一些人的姓名与其电话号码的对应关系，利用这种对应关系可以迅速查找到一个人的电话号码。

(4) 请模仿教材中【例 10.12】的程序，使用队列 queue 类模板实例化一个保存浮点数的队列。

(5) C++标准库中的 string 类实现的字符串比 C 语言风格的字符串的操作更简便。实际上 C++标准库中的 string 类定义了很多成员函数，功能很强。教材【例 10.14】中的 C++程序演示了 C++中的 string 类的字符串插入和替换操作等比较高级的使用方法。请编译、运行该程序，观察运行结果。回想 C 语言风格的字符串的进行类似操作时应该如何办？体会 C++中的 string 类的字符串好处。

实验 10　C++标准库（2）

1. 实验目的

练习比较复杂的输入输出流和文件的存取，熟悉 C++的 I/O 流类库。

2. 实验要求

(1) 熟悉输入输出流及流类库的作用。
(2) 掌握输入输出流类库中常用的类及其成员函数的使用方法。
(3) 学习输入输出基本方法及其格式控制。
(4) 学习 C++语言中重载输入和输出运算符的方法。
(5) 学习文本文件和二进制文件的输入输出方法。

3. 实验内容

(1) 教材上【例 11.5】中的程序在学生类 Student 中重载了输入运算符“>>”和重载输出运算符“<<”，使得输入、输出学生 Student 类的对象时与输入输出标准内置数据类型的方式一样。请你编译、运行【例 11.5】中的程序。

(2) 请你修改教材上【例 11.6】中的程序，使用操纵符来控制输出数据的格式。

(3) 请模仿教材上【例 11.12】和【例 11.13】中的程序编写一个 C++程序，在程序中使用文件流类对象创建一个文本文件 test1.txt 并向文件中输出一行字符串 "Learning C++ programming is fun!" 。然后从该文件中输入其内容显示在屏幕上。

(4) 请模仿【例 11.15】中的程序，将【例 2.7】程序中的 Student 类对象整体写到一个文件 STUD.DAT 中，然后从从 STUD.DAT 文件中读出这些数据，并显示在屏幕上。

实验 11　异常处理

1. 实验目的

练习 C++语言中处理异常错误的方法，体会这种方法的优越性。

2. 实验要求

(1) 了解 C++语言中异常处理的执行过程。
(2) 学习 C++语言中异常处理语句的功能，掌握其使用方法。
(3) 学会自定义运行终止函数。

3. 实验内容

(1) 下面的程序是教材上【例 12. 3】的程序，该程序主要处理除数为零的异常错误。请你编译并运行该程序。选取输入 m=6,n=3 和 m=6,n=0 单步调试该程序，观察程序的执行流程。

```
#include <iostream>
using namespace std;
int main( )
{  int m,n;
   cout<<"Please input two integers:"; cin>>m>>n;
   try
   {   if (n==0)   throw 0;
       cout<< (m/n)<<endl;
   }
   catch(int)
       {
       cout<<"Divided by 0!"<<endl;
       return -1;
   }
  return 0;
```

在 try 代码块中包含需要监控的程序部分

catch 语句捕获一个整型异常

```
}
```

(2) 下面的程序是教材上【例 12.4】的程序，该程序处理了从函数内部抛出的异常信息。请你编译并运行该程序。选取输入 m=6,n=3 和 m=6,n=0 单步调试该程序，观察程序的执行流程。

```
#include <iostream>
using namespace std;
int division(int x, int y);
int main( )
{  int m,n;
    cout<<"Please input two integers:";
    cin>>m>>n;
    try
    {
        cout<<division(m,n)<<endl;
    }
    catch(int)
    {
        cout<<"Divided by 0!"<<endl;
        return -1;
    }
   return 0;
}
int division(int x,int y)
{   if (y==0)
        throw 0;     //异常信息从函数内部抛出
    return x/y;
}
```

(3) 下面的程序是教材上【例 12.7】的程序，请你编译并单步调试该程序，观察程序的执行流程。请问用户自定义运行终止函数的主要作用是什么？

```
#include <iostream>
using namespace std;
void myterm() //自定义的运行终止函数
{
    cout<<"This is my terminater."<<endl;
    //...释放程序中申请的系统资源
    exit(1);
}
```

```
int main()
{
    //...
    try{
        set_terminate(myterm);
        //...
        throw "Exception ... ";
    }
    catch(int i){  }
    return 0;
}
```

实验 12　面向对象程序设计综合练习

1. 实验目的

通过编写一个较大的程序，亲身体会面向对象程序设计方法的优越性。

2. 实验要求

本实验为选作题，请同学们充分发挥主观能动性和创造性，自己选题，按照面向对象程序设计的方法进行设计、编码实现一个较大规模的程序。可以由三至五个同学组成一个开发小组，合作完成一个项目。但是必须分工明确，每一位同学都要承担一份工作。

3. 实验内容

利用 Visual C++的 MFC 类库或者 C++标准库，采用面向对象的思想设计完成一个较大规模的程序，并在 Visual C++环境中编程实现。

Visual C++6.0 中的调试工具使用入门

一、程序开发过程中出现的错误类型

我们在进行程序设计时，不可避免地会犯错误。程序中的错误可以分为三类：编译错误、运行时错误和逻辑错误。

编译错误（Compile errors）又称为编译时错误（Compiling-time errors）：是由于错误的编码产生的。例如关键字拼写错误、将中文标点符号当成英文符号使用、遗漏了某些必要的标点符号或者使用了一个没有定义的标识符。

编译错误一般都是语法错误，当编译器对程序进行语法检查时，都能发现这些错误，并能够指出产生错误的位置（标出行号）。我们可以根据编译出错信息指出的行号找到对应的源代码行改正错误，重新编译源程序。只有当所有的编译错误被改正后，才能通过编译检查，产生目标代码文件。

改正编译错误的关键是要能正确理解编译器给出的编译错误信息。VC++环境中的编译、链接错误信息是用英文表示的。对于英文基础薄弱的读者，可以参考本实验指导书后面的“VC++编译、链接常见错误和警告信息中英文对照”。

通常情况下，一个语法错误可能产生多条编译错误信息，这是由于株连错误造成的，建议读者在处理编译错误时，找到第一个出现错误的位置改正后重新编译。这样能够避免被株连错误迷惑。值得指出的是，现在大部分编译器对错误的定位不精确，如果在编译器指出的行没有发现错误，应该向前查找错误。例如，当提示第 10 行发生错误时，如果在第 10 行没有发现错误，请从第 10 行开始往前查找错误并修改之。

运行时错误（Run-time errors）是在程序的运行阶段出现的，当运行环境检测到程序的某些操作无法执行，例如除数为零时，就会出现运行时错误。当运行环境检测到程序的某些操作是被禁止的，也会产生运行时错误。例如，访问数组时超越数组的边界，空指针引用（NULL pointer assignment，空指针赋值，即有指针未赋具体地址就使用了）等等。

逻辑错误（Logic errors）：当程序没有按照程序员的意图执行时，就表明程序中存在逻辑错误。一个应用程序可能既没有语法错误，运行时也没有执行任何无效的操作，但是有可能产生错误的结果，这种错误结果一般都是程序内部的逻辑错误造成的。只有通过测试应用程序并分析它产生的结果，我们才能核实应用程序是否正确地执行了。

当然，如果输入了错误的或者无效的数据，执行程序后肯定也得不到正确的结果。软件行业中有一句名言“输入的是垃圾，输出也是垃圾。（Garbage in, garbage out.）”因此，一般的实用程序还需要对输入数据的正确性和有效性进行检验。

很显然，必须找出并改正程序中的错误，才能得到正确的执行结果。对于语法错误，我们可以根据编译错误信息指出的位置和错误原因来改正错误。请记住：编译器不能发现程序中的逻辑错误和运行时错误。我们可以通过仔细阅读源程序来发现逻辑错误，还可以借助开发环境中提供的调试工具来查找程序中的逻辑错误和运行时错误。

所谓调试（debug）就是定位程序中的错误并改正错误的过程。为了帮助程序员找出并改正程序中的错误，微软公司在 Visual C++6.0 环境中集成了调试器（debugger），调试器就是调试工具（debugging tools）。下面我们将介绍 Visual C++6.0 中的调试工具及其使用方法。

二、Visual C++6.0 中的调试工具简介

Visual C++6.0 环境中的调试工具功能十分强大，既可以支持 C++语言源代码级的调试，也可以支持汇编语言级和机器语言级的调试。我们在此仅介绍 C++语言源代码级的调试。Visual C++6.0 中的调试器允许我们在程序中设置断点（breakpoint），单步运行程序（step），运行到光标处（run to cursor），监视（watch）变量值等等。

断点是我们通过调试器在源程序中设置的一个位置。当运行到断点时，程序中断执行，这样我们可以观察断点处各个相关变量的值，通过变量的值分析、判断到断点处时程序的执行是否符合我们的期望。断点是最常用、最基本的技巧。在 Visual C++6.0 中设置断点的最简单方法就是通过点击 Build 工具条上的插入断点按钮（手掌形状按钮）或者按键盘上的功能键 F9。如图 13 所示，

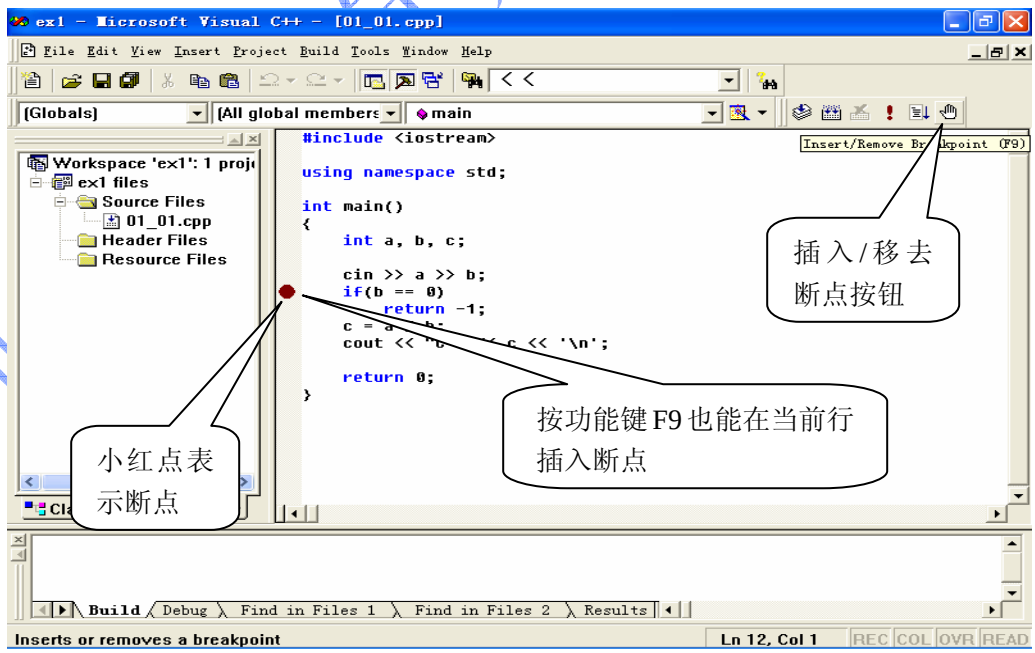

图 13 在源程序中设置断点

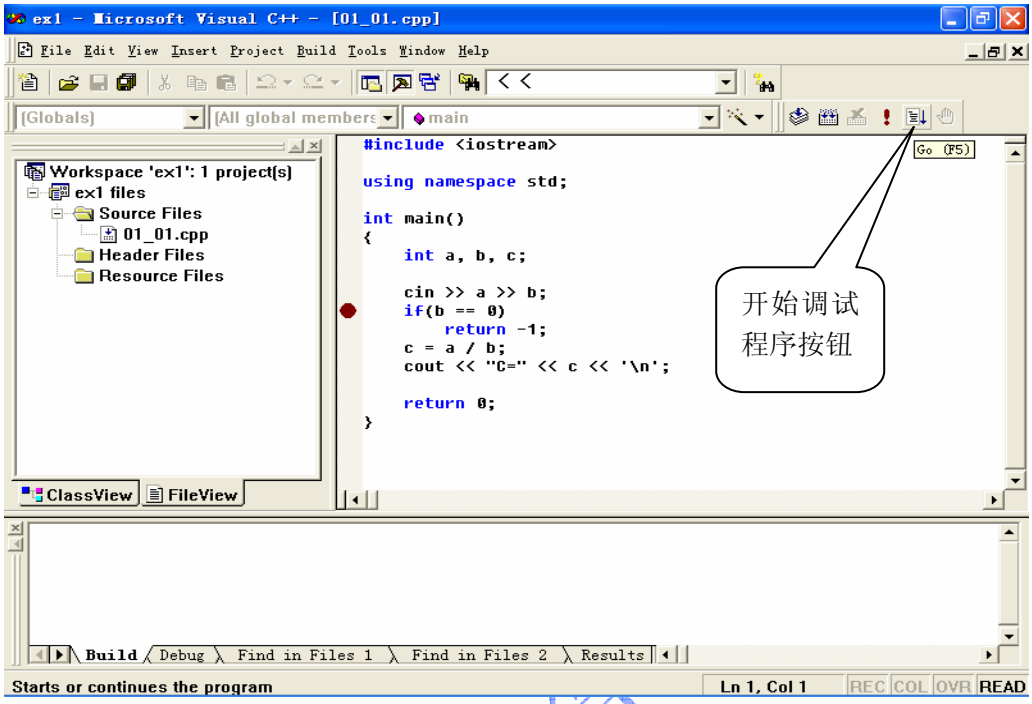

图 14　开始调试程序

一次可以设置多个断点，设置了断点之后，就可以通过点击 Build 工具条上的开始调试程序按钮或者按键盘上的功能键 F5 开始调试程序。如图 14 所示，

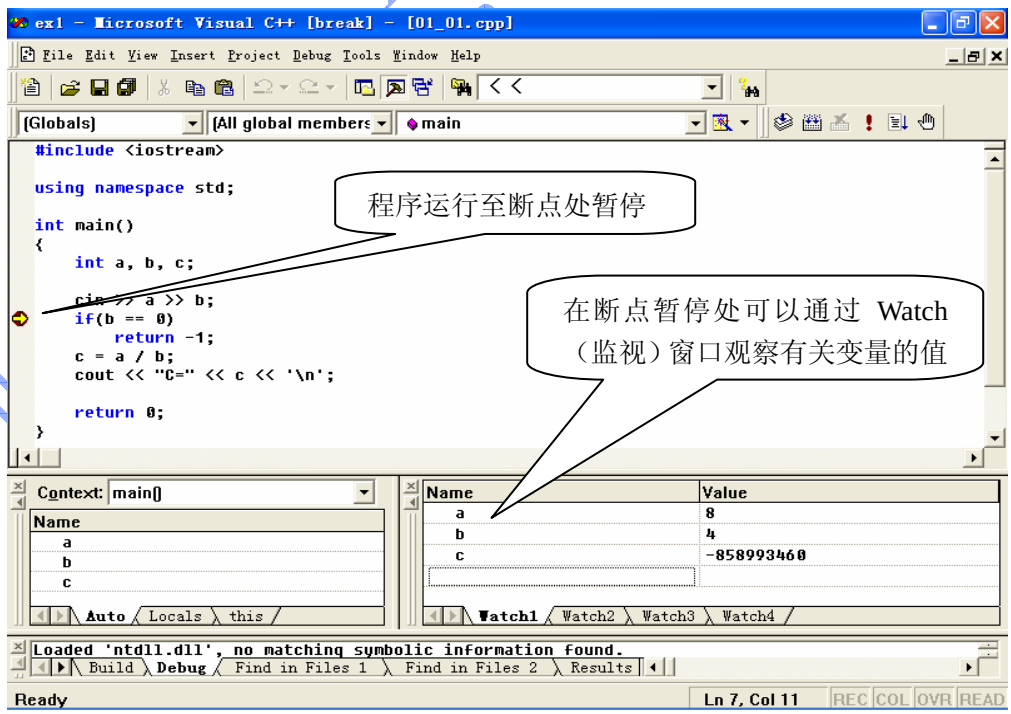

图 15　程序运行至断点处暂停

当程序运行至断点处会暂停，我们可以在断点暂停处通过 Watch 窗口观察有关变量的值，如图 15 所示。如果 Watch 窗口没有显示我们所关心的变量，可以自行添加。

另外还有一个更简单的观看变量的值方法：当程序运行到断点时，把鼠标光标移动到源程序中的某个变量上，停留一秒钟就可以看到这个变量的值。

移去断点：把键盘光标（插入符）移动到断点所在的行，再次按功能键 F9 就可以移去断点。也可以通过点击 Build 工具条上的插入/移去断点按钮（手掌形状按钮）取消断点。

单步运行程序就是每一次只执行一条语句就暂停。Visual C++6.0 中的调试工具提供三种单步运行方式：Step Into, Step Over, Step Out，如图 16 所示。Step Into 命令的快捷键是功能键 F11，Step Over 命令的快捷键是功能键 F10，Step Out 命令的快捷键是功能键 Shift+F9。Run to Cursor（运行到光标处）命令的快捷键是功能键 Ctrl+F10。

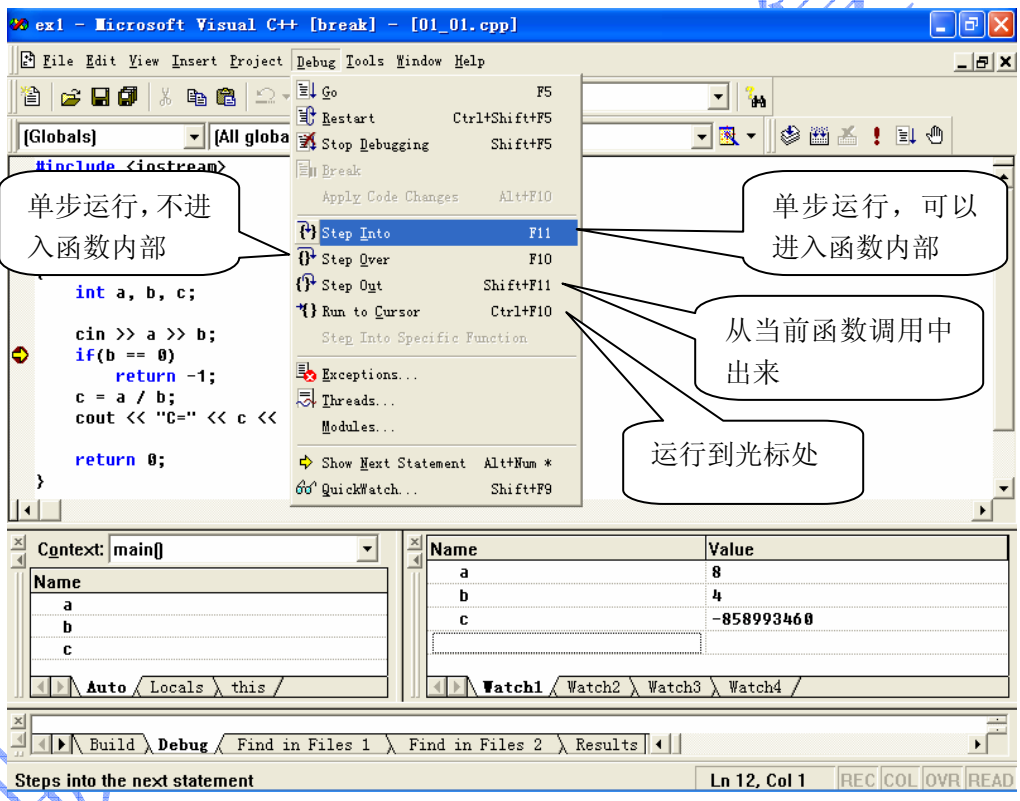

图 16　单步运行

每次单步运行一条语句后，都可以通过 Watch（监视）窗口观察有关变量的值，或者将鼠标光标移动到源程序中的某个需要观察的变量上，停留一秒钟就可以看到这个变量的值，如图 17 所示。

如果我们确认当前函数中没有错误时，可以使用 Step Out 命令快速地执行完毕当前的函数，暂停在该函数调用的下一条语句上。

如果我们确认某一段源代码中没有错误时，可以将键盘光标（插入符）放置在这段代码的末尾处，使用 Run to Cursor（运行到光标处）命令快速地执行完毕这一段代码，

暂停在光标所在语句上。例如我们可以将键盘光标移动到 main 函数的结尾处，然后按功能键 Ctrl+F10 执行 Run to Cursor 命令。这样程序将运行到结束处停止。

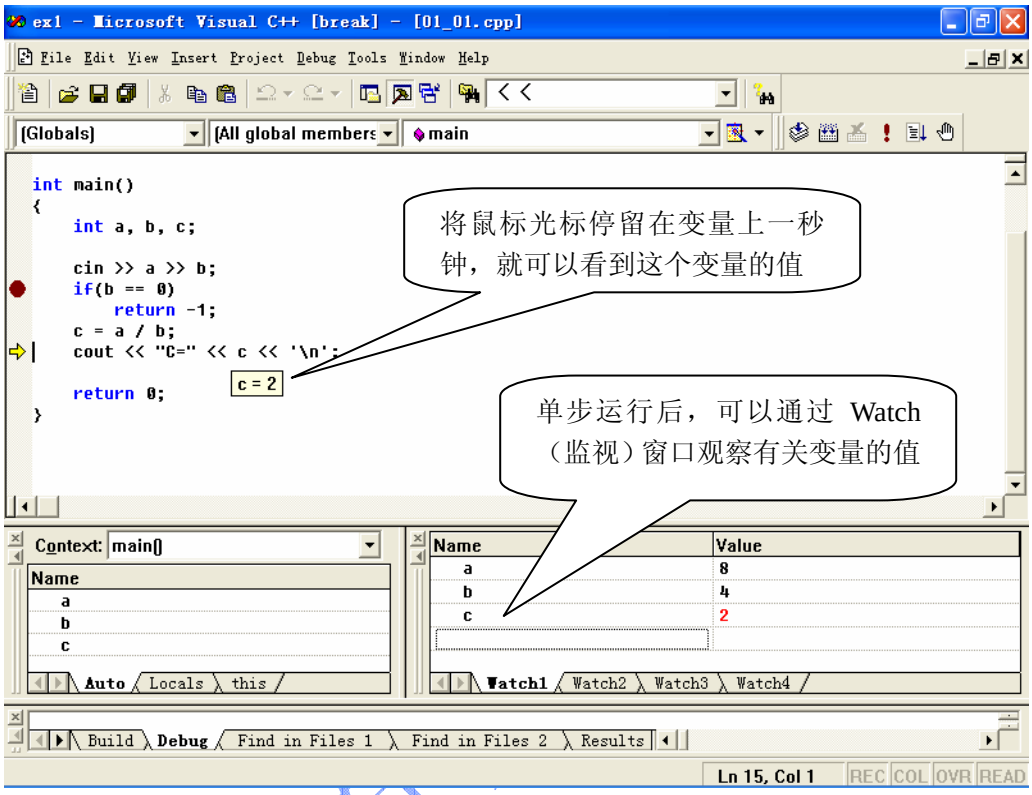

图 17　单步运行后观察相关变量的值

需要停止调试程序时，可以执行 Stop Debugging（停止调试）命令，如图 18 所示。Stop Debugging 命令的快捷键是功能键 Shift+F5，Restart（重新开始调试程序）命令的快捷键是功能键 Ctrl+Shift+F5。Go（开始调试程序）命令的快捷键是功能键 F5。

Go 命令还可以使程序从当前语句运行到下一个断点处，如果后面没有断点，就运行到到程序的结尾。

除了使用快捷键和 Debug（调试）菜单执行调试命令，还可以使用 Debug 工具条，如图 19 所示。使用工具条上的 QuickWatch 可以添加要观察的变量或者表达式。工具条上还有一些其他工具可以帮助我们查看 CPU 内部的寄存器（Registers）、内存单元（Memory）和堆栈（Stack）的情况。

Visual C++6.0 中的调试工具功能强大，在这里我们只介绍最基本的操作。当读者积累了一定的调试经验后可以继续学习调试工具的高级使用方法。

最后要强调一点，当程序员对程序进行调试时，一定要理解程序的算法，熟悉程序的结构和流程，准确估计变量和表达式的期望值。这样才能做出正确的分析和判断。调试的过程实质上是一个思考的过程。

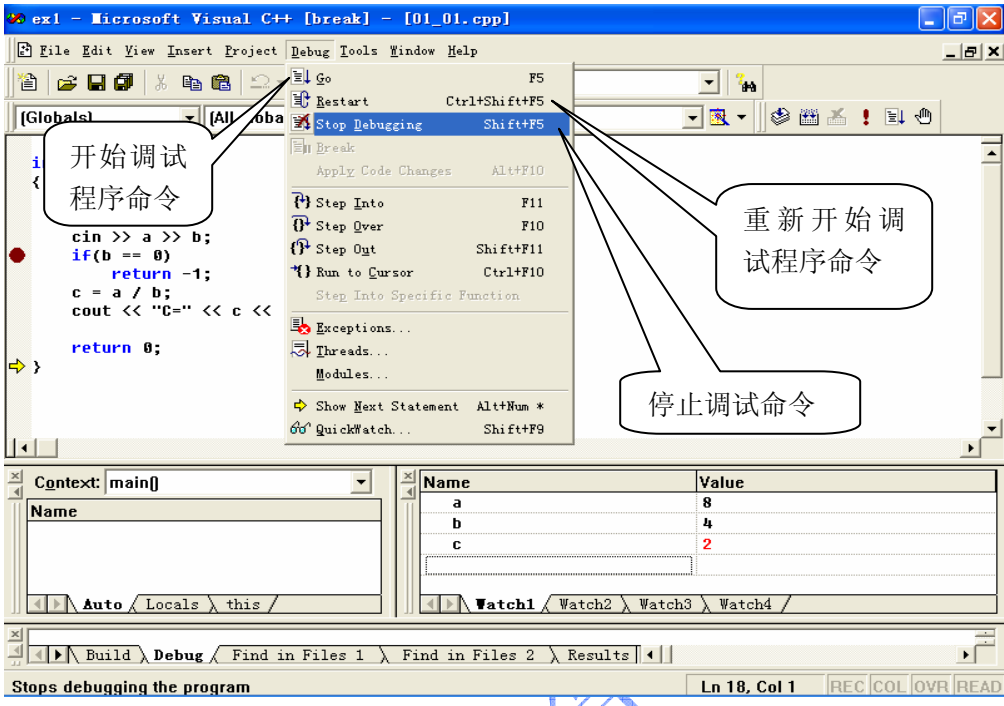

图 18 停止调试命令等

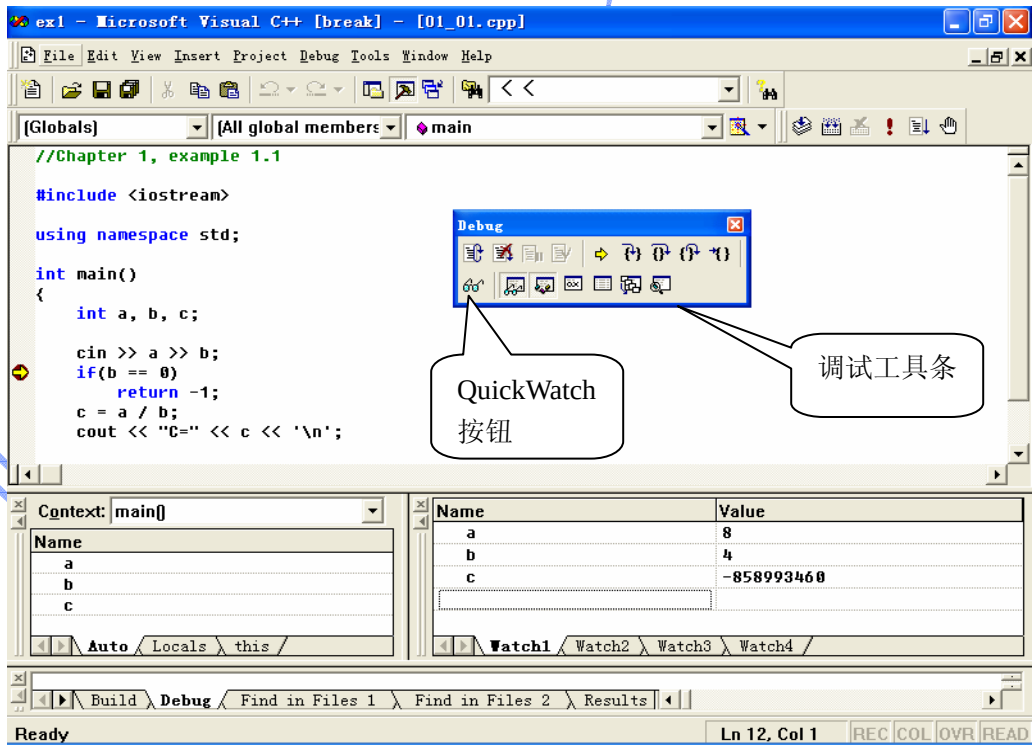

图 19 调试工具条

VC++编译、链接常见错误和警告信息中英文对照

1、fatal error C1010: unexpected end of file while looking for precompiled header directive。

寻找预编译头文件路径时遇到了不该遇到的文件尾。（一般是没有#include "stdafx.h"）

2、fatal error C1083: Cannot open include file: 'R…….h': No such file or directory

不能打开包含文件“R…….h”：没有这样的文件或目录。

3、error C2011: 'C……': 'class' type redefinition

类“C……”重复定义。

4、error C2018: unknown character '0xa3'

不认识的字符'0xa3'。（一般是汉字或中文标点符号）

5、error C2057: expected constant expression

希望是常量表达式。（一般出现在 switch 语句的 case 分支中）

6、error C2065: 'IDD_MYDIALOG' : undeclared identifier

“IDD_MYDIALOG”：未声明过的标识符。

7、error C2082: redefinition of formal parameter 'bReset'

函数参数“bReset”在函数体中重定义。

8、error C2143: syntax error: missing ';' before '{'

句法错误：“{”前缺少“;”。

9、error C2146: syntax error : missing ';' before identifier 'dc'

句法错误：在“dc”前丢了“;”。

10、error C2196: case value '69' already used

值 69 已经用过。（一般出现在 switch 语句的 case 分支中）

11、error C2509: 'OnTimer' : member function not declared in 'CHelloView'

成员函数“OnTimer”没有在“CHelloView”中声明。

12、error C2511: 'reset': overloaded member function 'void (int)' not found in 'B'

重载的函数“void reset(int)”在类“B”中找不到。

13、error C2555: 'B::f1': overriding virtual function differs from 'A::f1' only by return type or calling convention 类 B 对类 A 中同名函数 f1 的重载仅根据返回值或调用约定上的区别。

14、error C2660: 'SetTimer' : function does not take 2 parameters“SetTimer”函数不传递 2 个参数。

15、warning C4035: 'f……': no return value f……”函数 f……没有返回值。

16、warning C4553: '= =' : operator has no effect; did you intend '='?

没有效果的运算符“= =”；是否改为“=”？

17、warning C4700: local variable 'bReset' used without having been initialized

局部变量“bReset”没有初始化就使用。

18、error C4716: 'CMyApp::InitInstance' : must return a value

“CMyApp::InitInstance”函数必须返回一个值。

19、LINK : fatal error LNK1168: cannot open Debug/P1.exe for writing

连接错误：不能打开 P1.exe 文件，以改写内容。（一般是 P1.Exe 还在运行，未关闭）

20、error LNK2001: unresolved external symbol "……"

连接时发现没有实现的外部符号（没有定义的变量、没有定义的函数等）。

参考文献：

1. 张德慧，周元哲. 2005 年，C++面向对象程序设计，科学出版社，北京
2. 陈维兴，陈昕. 2003 年，C++面向对象程序设计习题解析与上机指导，清华大学出版社，北京