高等学校计算机专业教材

C++语言例题、习题及实验指导

陈志泊　主编
陈志泊　王春玲　编著

人 民 邮 电 出 版 社

图书在版编目（**CIP**）数据

C++语言例题、习题及实验指导/陈志泊主编；陈志泊，王春玲编著.
—北京：人民邮电出版社，2002.4
高等学校计算机专业教材
ISBN 7-115-09869-7

I.C…　II.①陈…②陈…③王…　III.C 语言—程序设计—高等学校—教材　IV.TP312

中国版本图书馆 CIP 数据核字（2003）第 016612 号

内　容　提　要

C++语言是计算机及其相关专业重要的程序设计语言。本书简明扼要地讲解了 C++语言的基本内容和主要知识点。全书共分为 10 章，每章都由内容要点、例题解析、练习题和上机实验四个部分组成，主要讲解了函数及其重载、内联函数、Const 与指针、类与对象的定义和使用、构造函数与析构函数、继承与派生、虚函数与多态性、类与静态成员、友元函数与友元类、运算符重载、函数模板与类模板、流与文件操作等。

本书可作为大学计算机及相关专业的本、专科生学习 C++程序设计语言课程的参考书，也是报考计算机专业硕士研究生的学习参考书。本书内容实用，也比较适合广大计算机爱好者自学和参考。

高等学校计算机专业教材

C++语言例题、习题及实验指导

◆ 主　编　陈志泊

◆ 编　著　陈志泊　王春玲

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号

邮编　100061　　电子函件　315@ptpress.com.cn

网址　http://www.ptpress.com.cn

读者热线：010-67180876

北京汉魂图文设计有限公司制作

印刷厂印刷

新华书店总店北京发行所经销

◆ 开本：787×1092　1/16

印张：

字数：　　2002 年 3 月第 1 版

印数：1 – 0 000 册　　2002 年 3 月北京第 1 次印刷

ISBN 7-115-　　-1/TP

定价：　　元

编 者 的 话

随着计算机技术飞速发展及其应用领域的扩大，快速而扎实地掌握、精通一门程序设计语言，是计算机专业人员不可回避的课题。多年来的实践和经验告诉我们，C 语言以其代码效率高、灵活、运算符丰富、结构化程序设计方法、与操作系统的结合紧密等特点而受到了广大计算机专业人士及计算机爱好者的欢迎。

C++语言是在 C 语言的基础上发展起来的，不仅继承了 C 语言的原有特点，而且引入了面向对象的程序设计方法，从而成为计算机及相关专业程序设计语言课程的首选语言。另外，Microsoft 公司推出的 Visual C++6.0 集成开发环境，提供了建立控制台应用程序的方法，为我们学习和掌握 C++语言提供了良好的开发环境；同时它又很好地利用了 C++语言，并提供了大量的 MFC（Microsoft Foundation Class）基础类库、应用程序向导和“可视化”的资源编辑器，这为程序员快速而高效地开发出 Windows 应用程序提供了方便。因此，C++语言是当今计算机专业及其相关专业的大学生所必须掌握的重要的程序设计语言之一。

为帮助广大同学和计算机爱好者更好地学习和掌握 C++语言，我们特别编写了本书，全书共分成了 10 章，每章都由内容要点、例题解析、练习题和上机实验四个部分组成。内容要点部分以比较简明和通俗易懂的方式对 C++语言的知识点进行了概括和讲解；例题解析部分针对每一章的知识点列举了大量的例题，并对例题进行了详细地分析，帮助读者进一步加深和巩固知识点的内容；练习题部分针对知识点的要求精选了大量的习题并给出了参考答案，可以帮助读者自学和练习；上机实验部分给出了上机的题目和要求，目的是对每一章的内容进行综合练习和提高程序设计的能力。

本书第 1 章是 C++语言基础，重点介绍了函数与重载、Const 与指针、引用的定义、特点及使用；第 2 章是类与对象，重点介绍了类与对象的定义、类的成员的访问方法；第 3 章介绍了构造函数和析构函数，主要包括构造函数、析构函数和拷贝构造函数的定义和特点、对象成员的定义和使用、常对象和常对象成员等；第 4 章介绍了继承和派生；第 5 章介绍了虚函数与多态性；第 6 章介绍了静态成员；第 7 章介绍了友元函数与友元类；第 8 章介绍了运算符重载；第 9 章介绍了函数模板与类模板；第 10 章介绍了 C++流与文件。

本书强调对 C++语言知识点的概括和综合，强调学习过程的习题练习和实验练习，每章中所附的习题都有相应的答案，便于读者自学时检查。另外，全书中所附的程序都已在 Visual C++ 6.0 环境下调试通过。

本书由陈志泊主编，全书由陈志泊、王春玲编写。

由于时间仓促，加之作者水平有限，不当之处在所难免，敬请读者批评指正。

编　者

2003 年 1 月

目　　录

第 1 章　C++语言基础

1.1　内 容 要 点

C++语言是在 C 语言的基础上发展起来的，不仅继承了 C 语言的原有特点，而且引入了面向对象的程序设计方法。本章主要介绍 C++语言本身独有的基础知识，而不再介绍与 C 语言共同的部分，如：常量、变量、运算符、表达式、数组、判断、循环、函数、结构体等，目的是重点突出 C++语言，使读者更有重点地掌握 C++语言。

1. 内联函数

（1）内联函数的定义

内联函数在定义时与普通函数基本一致，只是在函数值的类型前加“inline”关键字，其定义方法和格式如下：

inline 函数值的类型 函数名（形参及其类型列表）

{　函数体　}

（2）内联函数与普通函数的区别和联系

① 在定义内联函数时，函数值的类型左面有“inline”关键字，而普通函数在定义时没有此关键字。

② 程序中调用内联函数与调用普通函数的方式和方法相同。

③ 当在程序中调用一个内联函数时，是将该函数的代码直接插入到调用点，然后执行该段代码，所以，在调用过程中不存在程序流程的跳转和返回问题；而普通函数的调用，程序是从主调函数的调用点转去执行被调函数，待被调函数执行完毕后，再返回到主调函数的调用点的下一语句继续执行。

④ 从调用机理看，内联函数可以加快程序代码的执行速度和效率并减少调用开销，但这是以增加程序代码为代价来求得速度的。

（3）对内联函数的限制

不是任何一个函数都可定义成内联函数的。

① 内联函数的函数体内不能含有复杂的结构控制语句，如：switch、循环和 goto 语句，如果内联函数的函数体内有这些语句，则编译程序将该函数视同普通函数那样产生函数调用代码。

② 递归函数不能被用作内联函数。

③ 内联函数中不能说明数组。

④ 内联函数一般适合于只有 1～5 行语句的小函数，对于一个含有很多语句的大函数，函数调用和返回的开销相对来说是微不足道的，所以也没有必要用内联函数来实现。

2. 函数重载

（1）函数重载的概念

在 C++语言中，允许定义多个相同名称的函数，但这些函数的形式参数表不同。

函数重载是指一个函数可以和同一作用域中的其他函数具有相同的名字，但这些同名函数的参数类型、参数个数、返回值以及函数功能可以完全不同。

在原有的 C 语言中，每个函数必须有其惟一的名称，这样的缺点是所有具有相同功能、而只是函数参数不一样的函数，就必须用一个不同的名称，而 C++语言中采用了函数重载后，对于具有同一功能的函数，如果只是由于函数参数类型不一样，则可以定义相同名称的函数。

由于重载函数具有相同的函数名，在进行函数调用时，系统依据什么来确定所调用的函数是谁呢？系统一般按照调用函数时的参数个数、类型和顺序来确定被调用的函数。

（2）定义重载函数时的注意事项

① 重载函数间不能只是函数的返回值不同，应至少还在形参的个数、类型或顺序上有所不同。

② 应使所有的重载函数的功能相同。如果让重载函数完成不同的功能，是不好的编程风格，因为这样会破坏程序的可读性。

3. 默认参数的函数

在 C 语言中，被调函数的形参的值是由主调函数的调用点上的实参的值传递过来的，因此，调用函数时的实参的个数、类型、顺序都必须与被调函数的形参的个数、类型、顺序保持一致。

而 C++语言则允许在定义函数时给其中的某个或某些形式参数指定默认值，指定默认值的方法就是在相应的形参后面写上“=值”，这样，当发生函数调用时，如果省略了对应位置上的实参的值，则在执行被调函数时以该形参的默认值进行运算。

注意：

① 默认参数一般在函数说明中提供。如果程序中既有函数的说明又有函数的定义，则定义函数时不允许再定义参数的默认值；如果程序中只有函数的定义而没有函数的说明，则默认参数才可以出现在函数定义中。

② 默认参数的顺序。如果一个函数中有多个默认参数，则形参分布中默认参数应从右至左逐渐定义。如：

```
void myfunc(int a=1,float b,long c=20);        //错误
void myfunc(int a,float b=2.1,long c=30);      //正确
```

4. const 与指针

（1）指向常量的指针变量的定义与使用

指向常量的指针变量的定义方法：

const 类型标识符 *指针变量名；

如：

```
const int *p;
```

借助这种方法定义的指针变量只可读取它所指向的变量或常量的值，不可借助该指针变量对其所指向的对象的值进行修改（即重新赋值）。但是，可允许这种指针变量指向另外一个同类型的其他变量。

（2）指针常量

指针常量的定义格式为：

类型标识符 * const 指针变量名=初始指针值;

如：

```
char * const p="abcde";
```

用该方法定义的指针变量，其值（是一个指针值）不可进行修改（即不允许该指针变量再指向另外一个别的变量），但可以借助该指针变量对其所指向的对象的值进行读取或修改。另外，这种指针在定义时必须初始化。

（3）指向常量的指针常量

指向常量的指针常量的定义方法为：

const 类型标识符 * const 指针变量名=初始指针值；

如：

```
int b;
const int * const p=&b;
```

用这种方法定义的变量，既不允许修改指针变量的值也不允许借助该指针变量对其所指向的对象的值进行修改。另外，该变量在定义时必须初始化。如：

```
void main()
{   int a=100; int c=200;
    const int b=10;
    const int * const p=&a;      //定义 p 是指向常量的指针常量，所以必须初始化
    const int * const q=&b;      //定义 q 是指向常量的指针常量，所以必须初始化
    p=&c;      //该语句错误，p 的值不能更改
    *p=50;     //该语句错误，不能借助指针 p 对其所指向的对象的值进行更改
}
```

5. 引用及声明方法

引用就是某一变量（目标）的一个别名，这样，对引用的操作就是对目标的操作。

引用的声明方法：

类型标识符　　&引用名=目标变量名；

如：

```
int a;
int &ra=a;     //定义引用 ra,它是变量 a 的引用，即别名
```

说明：

① &在此不是求地址运算符而是起标识作用，标识在此声明的是一个引用名称；

② 类型标识符是指目标变量的类型；

③ 声明引用时，必须同时对其进行初始化；

④ 引用声明完毕后，相当于目标变量名有两个名称，即该目标原名称和引用名；

⑤ 声明一个引用，不是新定义了一个变量，它只表示该引用名是目标变量名的一个别名，所以系统并不给引用分配存储单元。

6. 引用的使用

① 一旦一个引用被声明，则该引用名就只能作为目标变量名的一个别名来使用，所以，不能再把该引用名作为其他变量名的别名，任何对该引用的赋值就是对该引用对应的目标变量名的赋值。

② 对引用求地址就是对目标变量求地址。

③ 由于指针变量也是变量，所以也可以声明一个指针变量的引用。方法是：

类型标识符 *&引用名=指针变量名；

④ 不能建立数组的引用，因为数组是一个由若干个元素所组成的集合，所以就无法建立一个数组的引用。

⑤ 引用是对某一变量或目标对象的引用，它本身不是一种数据类型，因此，引用本身不占存储单元，这样就不能声明引用的引用，也不能定义引用的指针。

如：下例中的操作是执行不了的。

```
int a;
int &ra=a;
int &*p=&ra;     //错误
```

⑥ 不能建立空指针的引用，如：不能建立 int &rp=NULL;

⑦ 也不能建立空类型 void 的引用，如：不能建立 void &ra=3;，因为尽管在 C++语言中有 void 数据类型，但没有任何一个变量或常量属于 void 类型，所以无法建立其引用，而且引用是对某一目标变量、常量或对象的引用，而不是对某一类型的引用。

7. 引用作为函数的参数

在定义一个函数时，函数的形参可以是基本数据类型的变量、数组名和指针变量等，除此之外，一个函数的参数也可定义成引用的形式，如：我们定义交换两个数的函数 swap，将函数的参数定义成引用的形式：

```
void swap(int &p1, int &p2)     //此处函数的形参 p1、p2 都是引用
{     int p;
      p=p1;
```

```
    p1=p2;
    p2=p;
}
```

为了在程序中调用该函数，在相应的主调函数的调用点处直接以变量作为实参进行调用即可，而不需要对实参变量有任何的特殊要求。如：对应上面定义的 swap 函数，相应的主调函数可写为：

```
main()
{   int a,b;
    cin>>a>>b;          //输入 a,b 两变量的值
    swap(a,b);          //直接以变量 a 和 b 作为实参调用 swap 函数即可
    cout<<a<<' '<<b;   //输出结果
 }
```

由此可看出，当发生函数调用时，主调函数的调用点处的实参变量 a 和 b 分别传递给被调函数的形参 p1 和 p2，由于形参 p1 和 p2 定义成引用的形式，所以这时 p1 就是实参 a 的引用，p2 就是实参 b 的引用。因此，在被调函数 swap 中，任何对形参 p1 和 p2 的操作实质就是对实参 a 和 b 的操作。

可以看出：

① 传递引用给函数与传递指针的效果是一样的，这时被调函数的形参就作为原来主调函数中的实参变量或对象的一个别名来使用，所以在被调函数中对形参变量的操作就是对其相应的目标对象（在主调函数中）的操作。

② 使用引用传递函数的参数在内存中并没有产生实参的副本（因为引用只是目标变量的别名而不是一个新的变量），且它是直接对实参操作；而使用一般变量传递函数的参数，当发生函数调用时需要给形参分配存储单元，这样形参与实参就占用不同的存储单元，所以形参变量的值是实参变量的副本。因此，当参数传递的数据量较大时，用引用比用一般变量传递参数的效率和空间占有率都好，这是利用引用的主要目的。

③ 使用指针作为函数的参数虽然也能达到与使用引用一样的效果，但是一方面，在被调函数中需要重复使用“*指针变量名”的形式进行运算，这很容易产生错误且程序的阅读性较差；另一方面，在主调函数的调用点处必须用变量的地址作为实参，这也较容易造成在被调函数中要对该变量的地址进行操作的错觉。

8. 返回引用的函数

要以引用返回函数值，则函数定义时的格式如下：

类型标识符&函数名（形参列表及类型说明）

{ 函数体 }

说明：

① 要以引用返回函数值，则定义函数时需要在函数名前加上&。

② 用引用返回一个函数值的最大好处是：在内存中不产生被返回值的副本；而用普通

的返回值的方法返回一个函数的函数值时，需要在内存中先创建一个临时变量，在被调函数返回时，将函数值复制到该临时变量中，主调函数再以该临时变量的值进行运算，可以看出，这种方法产生了函数值的副本。

③ 在引用的使用中，单纯给某个变量取个别名是毫无意义的，引用的目的主要是在函数参数传递中解决大对象的传递效率和空间不如意的问题。

④ 用引用传递函数的参数能保证参数传递中不产生副本和提高传递的效率，且通过 const 的使用，又保证了引用传递的安全性。

⑤ 引用与指针的区别在于：指针通过某个指针变量指向一个对象后，对它所指向的变量间接操作，程序中使用指针使程序的可读性差；而引用本身就是目标变量的别名，对引用的操作就是对目标变量的操作。

1.2 例 题 解 析

【例 1-1】定义一个求两个整数中较小值的函数 min()，要求定义成内联函数，然后在 main 函数中进行调用。

```
#include <iostream.h>
inline int min(int x,int y)        //将 min 函数定义成内联函数
{
    return (x<y?x:y);
}
void main()
{
    int m=10,n=20,t;
    t=min(m,n);
    cout<<"最小值："<<t;
}
```

说明：

从本例中可以看出，从形式上看，内联函数只是在定义方法上与普通函数稍有不同，而在调用方法上与普通函数一致。但从函数内部调用机理上来看，在程序编译过程中，程序中遇到调用内联函数的地方，编译程序就把该函数的代码插入到调用处，然后再编译，因此调用内联函数的效率高了。

【例 1-2】定义并测试函数的重载。

```
#include <iostream.h>
int absolute(int x)
{
    return (x<0?-x:x);
}
```

```
double absolute(double x)
{
    return (x<0?-x:x);
}
int min(int x,int y)
{
    return (x<y?x:y);
}
int min(int x, int y,int z)
{
    int t;
    if(x<y)
        t=x;
    else
        t=y;
    if(t>z)
        t=z;
    return (t);
}
void main()
{
    int m=10,n=20,k=-10;
    double d1=-43.2
    cout<<absolute(k)<<endl;
    cout<<absolute(d1)<<endl;
    cout<<min(m,n)<<endl;
    cout<<min(m,n,k)<<endl;
}
```

程序的运行结果是：

10

43.2

10

-10

说明：

程序中定义了两个 absolute 函数和两个 min 函数，这些函数虽然名称相同（即函数重载），但这些函数要么在参数的类型上有区别，要么在参数的个数上有区别，所以当发生函数调用时，程序会根据实参的类型和个数来调用相匹配的函数，因此程序会出现以上的运行结果。

【例 1-3】定义具有参数默认值的函数，并进行测试。

```
#include <iostream.h>
int sum(int i, int j=10);     //对 sum 函数的声明
int sum(int i, int j)
{
    return (i+j);
}
void main()
{
    int m=10, n=20;
    cout<<sum(m,n)<<endl;
    cout<<sum(m)<<endl;
}
```

程序的结果为：

30

20

说明：

在本程序中，定义了 sum 函数，其中形参 j 具有默认值 10。由于本例中，对函数 sum 进行了说明，所以参数 j 的默认值在函数说明中提供，而在定义函数时不允许再定义参数 j 的默认值。从程序的运行结果看，当未给第二个参数提供默认值时，该参数按默认值进行运算。

【例 1-4】指向常量的指针变量的定义、特点与使用。

```
#include <iostream.h>
void main()
{ const int i=20;              //定义常量 i
  int k=40;
  const int *p;                //定义指向常量的指针变量 p
  p=&i;                        //指针变量 p 指向常量 i
  cout<<*p<<i;
  *p=100;                      //该句错误，不可借助 p 对它所指向的变量进行重新赋值
  p=&k;                        //可以使 p 指向另外一个同类型的变量
  cout<<*p<<k;
  *p=200;                      //该句错误
  k=200;
}
```

【例 1-5】指针常量的定义、特点与使用。

```
#include <iostream.h>
void main()
{ char s[]="askdfsljfl";
```

```
    char * const p=s;        //指针常量定义时必须初始化
    p="xyz";                 //该句错误，不可再使指针变量指向另外一个地址（指针）
    cout<<*p;
    *p='s';
    cout<<*p;
    p++;
    *p='q';
    cout<<*p;
}
```

【例 1-6】指向常量的指针常量的定义、特点与使用。

```
#include <iostream.h>
void main()
{   int a=10; int c=30;
    const int b=20;
    const int * const p=&a;
    const int * const q=&b;
    p=&c;              //错误
    *p=50;             //错误
}
```

【例 1-7】 引用的定义及使用方法。

```
#include <iostream.h>
void main()
{   int a,b=10;
    int &ra=a;                  //定义引用 ra，初始化成变量 a，所以 ra 是变量 a 的引用（别名）
    a=20;
    cout<<a<<endl;
    cout<<ra<<endl;             //等价于 cout<<a<<endl;
    cout<<&a<<endl;             //输出变量 a 所占存储单元的地址（指针）
    cout<<&ra<<endl;            //等价于 cout<<&a<<endl;
    ra=b;                       //等价于 a=b;
    cout<<a<<endl;
    cout<<ra<<endl;             //等价于 cout<<a<<endl;
    cout<<b<<endl;
    cout<<&a<<endl;
    cout<<&ra<<endl;            //等价于 cout<<&a<<endl;
    cout<<&b<<endl;
}
```

【例 1-8】定义指针变量的引用及使用方法。

```
#include <iostream.h>
void main()
{
int *a;                  //定义指针变量 a
int *&p=a;               //定义引用 p,初始化为指针变量 a,所以 p 是 a 的引用（别名）
int b=10;
p=&b;                    //等价于 a=&b，即将变量 b 的地址赋给指针变量 a
cout<<*a<<endl;          //输出变量 b 的值
cout<<*p<<endl;          //等价于 cout<<*a;
}
```

【例 1-9】以下程序中定义了一个普通的函数 fn1（它用返回值的方法返回函数值），而另外一个函数 fn2，它以引用的方法返回函数值。

```
#include <iostream.h>
float temp;                  //定义全局变量 temp
float fn1(float r);          //声明函数 fn1
float &fn2(float r);         //声明函数 fn2
float fn1(float r)           //定义函数 fn1,它以返回值的方法返回函数值
{ temp=(float)(r*r*3.14);
   return temp;
}
float &fn2(float r)          //定义函数 fn2,它以引用方式返回函数值
{temp=(float)(r*r*3.14);
 return temp;
}
void main()                  //主函数
{
float a=fn1(10.0);           //第 1 种情况,系统生成要返回值的副本（即临时变量）
float &b=fn1(10.0);          //第 2 种情况,可能会出错（不同 C++系统有不同规定）
                             //不能从被调函数中返回一个临时变量或局部变量的引用
float c=fn2(10.0);           //第 3 种情况,系统不生成返回值的副本
                             //可以从被调函数中返回一个全局变量的引用
float &d=fn2(10.0);          //第 4 种情况,系统不生成返回值的副本
                             //可以从被调函数中返回一个全局变量的引用
cout<<a<<","<<c<<","<<d;
}
```

说明：

1．在 main 函数中，语句 float a=fn1(10.0);表示定义了一个变量 a，同时以 fn1(10.0)的函

数值初始化变量 a，这种情况是一般的函数返回值方式。当 fn1 函数被调用结束时，将计算结果存放到全局变量 temp 中，并返回该全局变量的值，这时 C++系统在内存中创建一个临时变量并将 temp 变量的值 314 复制给该临时变量。返回到 main 函数后，赋值语句 a=fn1(10.0); 把临时变量的值 314 赋给变量 a。

2．在 main 函数的语句 float &b=fn1(10.0);中，函数 fn1 也是以值方式返回函数值的，当返回时，复制全局变量 temp 的值给临时变量。当返回到主函数后，引用 b 以该临时变量来初始化，使得 b 成为该临时变量的别名。但由于临时变量的生存期短暂，根据 C++语言的规定，临时变量或对象的生存期在一个完整的语句表达式结束后便宣告结束，也就是说，在 float &b=fn1(10.0)；执行完毕后，临时变量便不再存在，所以引用 b 的值是个无法确定的值，因此出错。

3．在 main 函数中的第 3 条语句 float c=fn2(10.0)；表示在调用函数 fn2 后直接将全局变量 temp 的值返回给主函数，所以变量 c 直接从变量 temp 中得到该值，这样，就避免了产生返回值的一个副本后，再将副本赋给变量 c 的情况。因此，在这种情况下避免了临时变量的产生，同时也带来了程序执行的效率和空间利用，这也是用引用返回值的最大好处。

4．在 main 函数的第 4 条语句 float &d=fn2(10.0);中，函数 fn2 也是以引用返回函数值的，因此也不产生返回值的副本，并以 fn2 的函数值初始化引用 d，即使得 d 成为 temp 的别名。由于 temp 是全局变量，所以在 d 的有效期内 temp 始终保持有效，这种做法也是正确的。

一般情况下，赋值表达式的左边只能是变量名，即被赋值的对象必须是变量，因为只有变量才能被赋值，而常量或表达式不能被赋值，但如果一个函数的返回值是引用，则赋值号的左边可以是该函数的调用。

【例 1-10】测试用返回引用的函数值作为赋值表达式的左值。

```
#include <iostream.h>
int &put(int n);
int vals[10];
int error=-1;
void main()
{   put(0)=10;        //以 put(0)函数值作为左值，等价于 vals[0]=10;
    put(9)=20;        //以 put(9)函数值作为左值，等价于 vals[9]=20;
    cout<<vals[0]<<endl;
    cout<<vals[9]<<endl;
}
int &put(int n)
{   if (n>=0 && n<=9 )
        return vals[n];
    else
     { cout<<"subscript error";
        return error;
     }
```

```
}
```

【例 1-11】用 const 限定引用。

```
#include "iostream.h"
double &fn(const double &pd)
{ static double ad=32;
   ad+=pd;
   cout<<pd<<endl;
   return ad;
}
void main()
{ double a=100.0;
   double &pa=fn(a);
   cout<<pa<<endl;
   a=200.0;
   pa=fn(a);
   cout<<pa<<endl;
}
```

程序运行的结果为：

100
132
200
332

说明：

用 const 限定引用的声明方式为：

const 类型标识符&引用名=目标变量名；

用这种方式声明的引用不能通过引用对目标变量的值进行修改，从而使引用的目标成为 const，保证了引用的安全性。

【例 1-12】使用重载函数的方法定义两个重载函数，分别对整数和字符串用选择法从小到大排序。

```
#include <iostream.h>
#include <string.h>
void sort(int num[],int count);
void sort(char *ch[],int count);                    //声明 2 个重载函数 sont(),它们的形参类型不同
void main()
{
    char *city[]={"Shanghai","Beijing","Tianjin","Qingdao","Chongqing"};
    int number[]={100,-20,25,-44,20};
```