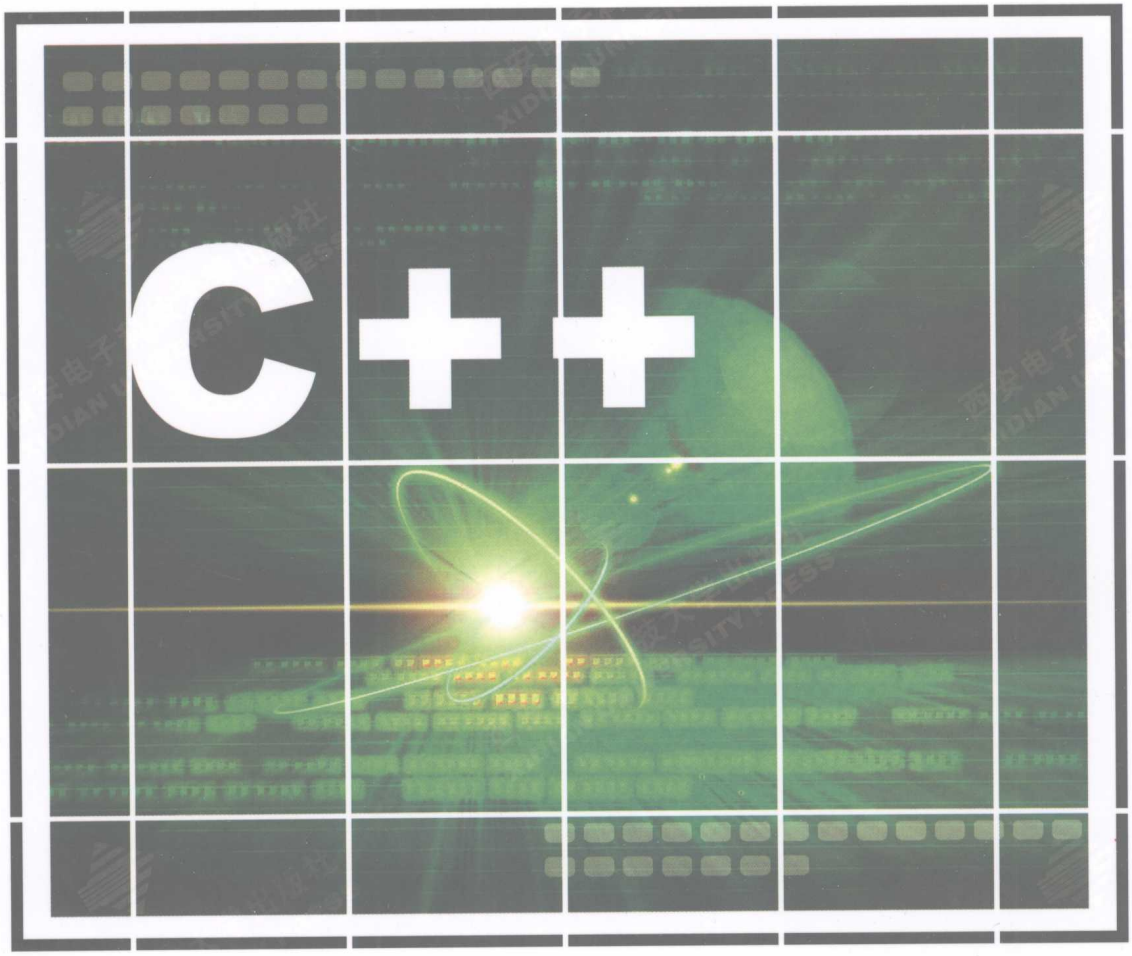
新世纪计算机类本科规划教材
COMPUTER

C++程序设计

主编　郑　炜



西安电子科技大学出版社
http://www.xduph.com

新世纪计算机类本科规划教材

C++程序设计

主　编　郑　炜
副主编　刘　斌　蔡康英　汪　芳

西安电子科技大学出版社
2009

内 容 简 介

本书采用生动轻松的语言，深入浅出地阐述了 C++ 语言和面向对象程序设计方法，包括类、对象、继承、重载、多态、虚函数和流等，将面向对象的思想逐步展开，然后再进一步扩展，讲述了 STL 的内容和 MFC 编程基础。

本书共 13 章，内容详实，体系合理，理论及应用兼顾，具有非常强的实用性。书中精选的例题和习题有助于读者加强对相关内容的理解。通过本书的学习，读者不但可以掌握 C++ 和面向对象的各种核心理论及技术，更能学以致用，领悟使用 C++ 进行程序设计的精髓。本书文字严谨流畅而又浅显易懂，是学习 C++的理想教材。对于没有 C 语言基础的读者，通过本书也能逐步学懂。

本书可作为高等学校相关专业程序设计课程的教材，也可作为各级软件开发技术人员的参考书，同时也是不可多得的自学用书。

★本书配有电子教案，需要者可登录出版社网站，免费下载。

图书在版编目(CIP)数据

C++程序设计 / 郑炜主编. —西安：西安电子科技大学出版社，2009.2

新世纪计算机类本科规划教材

ISBN 978-7-5606-2191-3

Ⅰ.C… Ⅱ.郑… 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(2008)第 209485 号

策　　划　臧延新

责任编辑　雷鸿俊　臧延新

出版发行　西安电子科技大学出版社(西安市太白南路 2 号)

电　　话　(029)88242885　88201467　　邮　　编　710071

网　　址　www.xduph.com　　电子邮箱　xdupfxb001@163.com

经　　销　新华书店

印刷单位　西安文化彩印厂

版　　次　2009 年 2 月第 1 版　2009 年 2 月第 1 次印刷

开　　本　787 毫米×1092 毫米　1/16　印 张 19.5

字　　数　462 千字

印　　数　1～4000 册

定　　价　28.00 元

ISBN 978 - 7 - 5606 - 2191 - 3/TP • 1120

XDUP 2483001-1

前　言

C++是以C语言为基础发展起来的面向对象程序设计语言，通常把C看做是C++的基础语言，但二者实质上是相互独立的。C++中基本包含了C语言的所有主要特性，但摒弃了C语言中某些格式复杂的部分，使得C++程序设计中过程化的程序设计部分更容易掌握。不过，由于面向对象思想的引入，C++的实质内容远比C语言更灵活和复杂。本书从程序设计的基础开始，先用通俗易懂的示例演示C++的简单程序和一般结构，随着内容的深入，逐步渗透面向对象的编程思想，以使读者逐渐建立起面向对象程序设计的概念。书中对C++语言的各种关键性问题、设计目标、语言思想，例如类、继承、抽象类、重载、存储管理、模板、异常处理、名字空间等都进行了详细的讨论。

本书旨在帮助初学者或者C++程序员更好地认识C++语言和面向对象的思想以及C++语言的基本概念和核心技术。

本书共13章，主要内容如下:

第1章介绍了C++数据类型、运算符和表达式，阐述了三种基本程序控制结构和函数。

第2章讲述了面向对象编程的一些基本概念和特性，包括类与对象的定义、构造和使用对象以及访问控制和友元。

第3章讲述了构造函数与析构函数、new和delete的运用。构造函数可以保证变量得到正确的初始化。关键字new和delete可以动态创建对象，还可以控制内存分配和释放的方式。

第4章讲述了函数重载。C++使用函数重载可以解决多个函数重名的问题，只要函数的参数列表不同，那么两个函数就可以使用同一个名字。

第5章讨论了流类体系与文件操作。

第6章讨论了异常和错误。

第7章介绍了const和inline。主要阐述了关键字const的意义和使用、内联函数的内部实现机制以及它的用法。

第8章讨论了C++结构中的相关内容，包括作用域和名字空间、static和预处理命令。

第 9 章讲述了继承。主要阐述了继承和组合的语法、如何重定义函数以及在继承和组合时构造函数与析构函数的重要性。

第 10 章讲述了多态和虚函数。

第 11 章讲述了模板和高级编程。模板通过使类型参数化来达到代码复用的目的。

第 12 章讲述了 STL。

第 13 章讲述了 MFC 编程基础。

本书内容详实，示例丰富，结构合理，讲解循序渐进、深入浅出，而且理论及应用兼顾，具有很强的逻辑性和实用性。

本书由西北工业大学郑炜主编并统稿，刘斌、蔡康英、汪芳担任副主编，参与编写的还有葛春平、李雨江、陶亚楠、段芳芳、苗育德、廖建、尤涛、吕强、崔留洋、曹俊等。

鉴于作者水平有限，书中不足之处在所难免，敬请广大读者批评指正。

编　者

2008年10月

目　录

第 1 章　程序设计基础

C++是一种使用非常广泛的计算机编程语言，它是以 C 语言为基础开发出来的，可以把 C 看做是 C++的子集，但 C++又与 C 不完全相同。作为基础语言，C++的设计保证了在它的类型、运算、语句与计算机直接处理的对象之间的紧密对应关系。

C++是一种通用型的程序设计语言，特别是面向系统程序设计；它是一个更好的 C，支持数据抽象，支持面向对象的程序设计技术，支持通用型程序设计。

本章主要介绍使用 C++ 语言进行程序设计的基础。

1.1　C++ 中的数据类型

绝大部分的程序设计语言中，数据都以变量或常量的形式来描述，每个变量或常量都有自己的数据类型。C++提供了大量的基本数据类型供我们使用。定义不同类型的数据主要是告诉编译器要分配不同大小的内存空间供我们存储程序中所要使用的数据，同时也告诉编译器对分配的内存空间怎么组织数据。

C++的数据类型可以分为基本数据类型和构造数据类型。基本数据类型也叫原子数据类型。

1.1.1　基本数据类型和非基本数据类型

1．基本数据类型

C++中的基本数据类型有整型(int)、字符型(char)、浮点型(float)、双精度型(double)和逻辑型(bool)，其中整型、字符型、浮点型和双精度型与 C 语言中的相同，逻辑型是 C++在 C 语言的基础上新增加的一种数据类型。对于这些基本数据类型又有 long(长型)、short(短型)、signed(有符号型)和 unsigned(无符号型)等修饰符。其中，long 可以用来修饰 int 型和 double 型；short 可以用来修饰 int 型；signed 和 unsigned 可以用来修饰 int 型和 char 型。signed (unsigned)还可以和 long(short)同时修饰一个基本类型。表 1.1 列出了所有的基本数据类型，包括它们在 win32 中所占用的空间大小以及它们所能表示的数的大小。

有了表 1.1 中的基本数据类型，我们就可以用它来定义相应类型的变量，系统也就为我们分配相应字节数的内存空间来存放程序中要用到的数据。

表 1.1　C++中的数据类型

基本数据类型	说　明	长度/B	表 示 范 围
int	整型	4	$-2^{31}\sim2^{31}-1$
unsigned int	无符号整型	4	$0\sim2^{32}-1$
signed int	有符号整型	4	$-2^{31}\sim2^{31}-1$
short int	短整型	2	$-2^{15}\sim2^{15}-1$
unsigned short int	无符号短整型	2	$0\sim2^{16}-1$
signed short int	有符号短整型	2	$-2^{15}\sim2^{15}-1$
long int	长整型	4	$-2^{31}\sim2^{31}-1$
unsigned long int	无符号长整型	4	$0\sim2^{32}-1$
signed long int	有符号长整型	4	$-2^{31}\sim2^{31}-1$
char	字符型	1	$-2^{7}\sim2^{7}-1$
unsigned char	无符号字符型	1	$0\sim2^{8}-1$
signed char	有符号字符型	1	$-2^{7}\sim2^{7}-1$
float	浮点型	4	$-3.4\times10^{38}\sim3.4\times10^{38}$
double	双精度型	8	$-1.7\times10^{308}\sim1.7\times10^{308}$
long double	长双精度型	8	$-1.7\times10^{308}\sim1.7\times10^{308}$
bool	逻辑型	1	true(1)、false(0)

以下列举一个 C++程序。

【程序 1.1】

```
#include <iostream>
using namespace std;
int main()
{
    int i=2;
    float f=2.5;
    double d=2.55;
    bool b=false;
    char c='a';
    cout<<"a="<<i<<"   int 占用字节数"<<sizeof(int)<<endl;
    cout<<"f="<<f<<"   float 占用字节数"<<sizeof(float)<<endl;
    cout<<"d="<<d<<"   double 占用字节数"<<sizeof(double)<<endl;
    cout<<"b="<<b<<"   bool 占用字节数"<<sizeof(bool)<<endl;
    cout<<"c="<<c<<"   char 占用字节数"<<sizeof(char)<<endl;

    return 0;
}
```

运行结果如下：

```
a=2  int 占用字节数4
f=2.5  float 占用字节数4
d=2.55  double 占用字节数8
b=0  bool 占用字节数1
c=a  char 占用字节数1
Press any key to continue
```

从程序 1.1 中可以看出，C++面向过程程序和 C 程序结构基本相同，都是从 main 函数开始执行，但也和 C 程序存在以下几点不同：

(1) 使用头文件 iostream。C++中使用输入/输出流(详见第 5 章)进行输入/输出。所有关于标准输入/输出流的定义都在头文件 iostream 中。当然 C++中也可以使用 C 语言的标准输入/输出头文件 stdio.h。

(2) 使用 cout 输出。cout 是利用 C++的输入/输出流类定义的对象(详见第 5 章)。在 C++中我们一般将要输出的内容放在 cout<<后就可以直接输出，其中<<叫做流插入运算符。而且<<可以连续书写进行输出。在程序中还用到了 endl，endl 是输入/输出流中的换行标志，功能类似于 C 中的 '\n'。

(3) bool 型变量。bool 是 C++中新加入的关键字，用来定义逻辑型变量。同时 C++中也提供了两个逻辑型常量 true 和 false，分别代表逻辑真和逻辑假。而在 C 语言中我们一般都用 int 型表示逻辑变量，用 0 代表逻辑真，非 0 代表逻辑假。

2．构造数据类型

C++中的构造数据类型有指针、数组、结构体、共用体(联合体)、枚举类型、类等。除了类类型以外其他的都和 C 语言中相应类型的含义一致，此处就不再讲解了。而类是 C++对 C 的一个重要扩充，C++的面向对象程序设计就是借助于类和一些其他的机制来实现的。

1.1.2　C_string 和 C++string

1．C_string 及其操作

C 语言使用 char 型数组或 char 型指针来表示字符串。同时在 C 的库文件 string.h 中还定义了若干字符串操作函数以供使用，这些函数有 strcat(字符串连接)、strcmp(字符串比较)、strcpy(字符串复制)、strlen(求字符串长度)、strstr(查找子串功能)等。

程序 1.2 中定义了一个 C_string(C 类型的字符串)并对它进行操作。

【程序 1.2】

```
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
    char c[20];
    strcpy(c,"abcdefg");
    cout<<c<<endl;
```

```
    char *pc="abcdegh";
    cout<<pc<<endl;
    if(strcmp(c,pc)>0)
        cout<<"abcdefg>abcdegh"<<endl;
    else if(strcmp(c,pc)==0)
        cout<<"abcdefg=abcdegh"<<endl;
    else
        cout<<"abcdefg<bcdegh"<<endl;
    return 0;
}
```

运行结果如下：

```
abcdefg
abcdegh
abcdefg<bcdegh
Press any key to continue
```

在程序 1.2 中定义了两个 C 类型字符串，一个用 char 型数组表示，一个用 char 型指针表示，并分别调用字符串处理函数对它们进行了操作。

2．C++string 及其操作

在 C++中除了可以用 C_string 来表示字符串外，还提供了一种更为方便的方法——用字符串类型(string)来定义字符串变量。例如：

```
string  str1;                    //定义了一个字符串变量
string str2="abcdefg";           //定义字符串变量的同时进行初始化
```

用 string 定义的字符串可以方便地进行一些类似于 C_string 中的 strcpy、strcmp 和 strcat 等计算，如下所示：

(1) 用赋值运算符 = 实现字符串复制。例如：

```
string  str1="abcd";
string  str2=str1;               //str 的值也为"abcd"
```

(2) 用加法运算符 + 实现字符串的连接。例如：

```
string  str1="abcd";
str1+="efgh";                    //str1 的值为"abcdefgh"
```

(3) 用逻辑运算符 = =、>、< 实现字符串的比较。例如：

```
string  str1="abcd";
string  str2="abce";
if(str1>str2)
    cout<<"str1>str2"<<endl;
else if(str1==str2)
    cout<<" str1==str2"<<endl;
else
    cout<<" str1<str2"<<endl;    //程序输出 str1<str2
```

程序 1.3 中将程序 1.2 改写成 C++string 形式。

【程序 1.3】

```
#include <iostream>
#include <string>
using namespace std;
int main()
{
    string str1="abcdefg";
    cout<<str1<<endl;
    string str2="abcdegh";
    cout<<str2<<endl;
    if(str1>str2)
        cout<<"abcdefg>abcdegh"<<endl;
    else if(str1==str2)
        cout<<"abcdefg=abcdegh"<<endl;
    else
        cout<<"abcdefg<bcdegh"<<endl;
    return 0;
}
```

在程序 1.3 中需要注意以下几个问题：

(1) 头文件使用了#include<iostream>而不是#include<iostream.h>。这是在 C++的新标准中推荐使用的头文件包含格式。它引用的头文件是 iostream 而不是 iostream.h(注意这是两个不同的头文件，在 VC6.0 的系统目录中它们都存在)。

(2) 语句“using namespace std;”是引用了标准命名空间 std(命名空间的概念见第 9 章)。在 C++新标准中，不带 .h 的头文件中定义的变量、函数等都在标准命名空间 std 中，所以当使用新标准的头文件时必须引用标准命名空间 std。using namespace std 就是引用标准命名空间的一种方法。

(3) 这里的头文件 string 并不是 C 语言中的头文件 string.h。 C 语言的 string.h 头文件(C++中也存在这个头文件)定义了一些字符串操作函数，而 C++的头文件 string 中使用了面向对象机制定义了字符串类型，这也就是我们使用 string 来定义一个字符串变量时要引入这个头文件的原因。C++中引用 C 头文件时去掉 .h，然后在头文件名称前面加上 c 前缀。

1.1.3　数组与向量 vector

1. 数组

C++中数组的概念和 C 语言中的完全一样。数组将同类型的多个变量归为一组，并且提供了使用下标偏移的方式来访问这些变量，从而使用户可以很方便地来操作它们。

一维数组的定义方式为：

类型　数组名[常量表达式]

常量表达式的值为数组元素的个数。以下语句就定义了一个 15 个元素的整型数组。

```
int array[15];
```

数组的初始化可以在定义时将数组各元素的值罗列在大括号中，如下所示：

```
int array[5]={0,1,2,3,4};
```

数组的存取通过下标来访问。上面的语句定义了一个数组 array，array 有 15 个元素，可以通过下标来访问这些数组元素。其实也可以认为 int array[15]定义了 15 个整型的下标变量，它们的名称为 array[0]，array[1]，…，array[14]。语句 array[0]=0 就为数组的第一个元素赋值为 0。正是由于数组中的元素采用了同一名称表示，同时提供了下标偏移的方式存取，因此在程序中应用起来比单独定义与数组元素个数相同数目的不同名称的变量要灵活得多。这也是数组的操作通常放在循环(循环结构见 3.3 节)中通过下标对数组元素进行统一操作的原因。如程序 1.4 所示的就是对数组元素进行存取的操作。

【程序 1.4】

```
#include <iostream>
using namespace std;
int main()
{
    int iArray[10];
    for(int i=0;i<10;i++)
        iArray[i]=i;
    for(int j=0;j<10;j++)
        cout<<iArray[j]<<"   ";
    return 0;
}
```

使用数组时应注意以下几个方面的问题：

(1) 定义数组时，方括号([])中的表达式必须为常量表达式而不能是变量表达式。例如，下面的定义方法是错误的：

```
int arraySize=20;
int iarray[arraySize] ;
```

这是因为编译系统要根据数组的定义来分配内存空间，所以在编译时就必须知道数组的大小。常量表达式在编译时其值已确定，而变量编译时其值还不确定，只有到运行时才能确定。

(2) 数组名代表了分配给数组的首元素地址，它是一个地址常量，所以不能对数组名赋值。例如，以下试图将数组 arraya 的值赋给数组 arrayb 的操作是非法的：

```
int   arraya[5]={0,1,2,3,4};
int   arrayb[5];
arrayb=arraya;      //非法操作
```

另外，数组也可以是多维的，即数组可以带有多个下标。例如，以下语句定义并初始化了一个二维数组：

```
int a[3][4]={{1,2,3,4},{5,6,7,8},{9,10,11,12}};
```

2．vector

从上面的讨论中可见，数组是一种静态数据结构，在定义时必须知道其大小，而在实际应用时我们经常需要用到动态数组，即按用户要求的大小定义数组的大小。C++ 中的 stl(标准模板库，见第 13 章)中的 vector 就给我们提供了一种创建动态数组的方法。关于 vector 的使用将在第 13 章中详细讨论，下面介绍一个简单的样例程序。

【程序 1.5】

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{
    int arraysize;
    cout<<"请输入要定义的数组的大小"<<endl;
    cin>>arraysize;
    vector<int> iarray(arraysize);
    for(int i=0;i<arraysize;i++)
    {
        iarray[i]=i;
    }
    for(int j=0;j<arraysize;j++)
    {
        cout<<iarray[j]<<"  ";
    }
    return 0;
}
```

运行结果如下：

```
请输入要定义的数组的大小
6
0  1  2  3  4  5  Press any key to continue
```

由此可见，我们可以使用 vector 创建动态数组，即数组的大小可以在运行时由用户确定。

在程序 1.5 中应该注意以下几点：

(1) vector 的定义在头文件 vector 中，所以使用 vector 时应包含头文件 vector。

(2) cin 是在 C ++中定义的用于标准输入的对象，cin>>arraysize 的作用就像 C 语言中的 scanf("%d", &arraysize)一样。

1.1.4　指针与引用

1．指针

指针是 C、C++中一个强有力的工具，正是有了指针，C、C ++ 才会如此高效、强大和

灵活。下面简要讨论指针的几点重要用法。

1) 指针变量的定义

指针变量是用来存储地址的一类变量。它的定义形式为：

类型名 * 指针变量名

如语句 int *p 定义了一个整型指针变量 p，该变量可以用来存放整型变量的地址。在C++中和C语言中一样，取一个变量的地址使用运算符&。以下语句将整型变量i的地址赋给了整型指针 p(也称为指针变量 p 指向了变量 i)。

```
int i=2;
int *p=&i;
```

2) 指针变量的间接引用

指针的间接引用是指通过指针变量来访问其所指向的变量。指针变量的间接引用方式如下所示：

*指针变量

考虑如下程序段的作用:

```
int i=2;
int *p=&i;              //整型指针指向了变量 i
*p=3;                   //指针的间接引用，*p 就代表了指针 p 所指向的变量 i
cout<<*p<<endl;         //输出结果为 3
cout<<i<<endl;          //输出结果为 3
```

上面的程序定义了指针变量 p 并将变量 i 的地址赋给了 p(也称为使 p 指向了 i)，以后通过 p 的间接引用就可以对变量 i 进行存取操作，所以执行 *p=3 的操作就相当于执行 i=3 的操作。

3) 指针变量作为函数参数

指针变量最重要的作用是作为函数的参数传递。详细讨论见 1.4 节。

2. 引用

C++在 C 的基础上扩展了一个引用的概念。使用引用可以达到指针的效果，但在某些情况下使用引用比使用指针要容易一些，程序的语法也要简洁一些。引用是为变量或对象取一个别名，当声明引用时用另一个变量或对象的名字对其进行初始化。这时引用和初始化它的变量(也称为它所引用的变量)占据同一内存空间，所以申明引用并不占用内存空间。在有的情况下用指针和引用能达到相同的效果，但使用引用比使用指针更节省内存空间。

1) 引用的申明

引用的声明使用下面的格式：

类型名 & 引用名 = 被引用的变量名

注意：声明引用使用运算符&，这里的&并不是取地址的意思。而且声明一个引用的时候就要使用别的变量或对象对其进行初始化。一旦声明了引用，引用就是其所引用的变量的别名，改变了引用的值也就改变了它所引用的变量的值，改变了变量的值也就改变了引用的值，因为它们占据的是同一内存空间。如下面的程序片断所示：

```
int a=5;
int b=7;
```

```
int &ra=a;
ra=b;
cout << ra << '\t'<< a << endl;
```

上面程序先定义了两个整型变量，然后定义了一个整型变量的引用并用变量 a 对其进行初始化。这时 ra 就是 a 的别名，它们占据同一内存空间。接着执行 ra=b，其作用并不是将 ra 重新引用到变量 b 上(因为引用的初始化只能在定义引用时进行)，而是将变量 b 的值赋给 ra，也就是赋给 a，所以最后输出 ra 的值和 a 的值都应该是 7。

2) 引用作为函数参数

C++中引用最主要的用处是作为函数参数。有关这方面的讨论详见 1.4 节。

1.2　运算符和表达式

运算符是 C 语言提供的用来表示一个、两个或多个变量或常量(也叫做操作数)之间进行算术、关系、逻辑和赋值等运算的符号。按照作用，运算符可以分为算术运算符、关系运算符、逻辑运算符和赋值运算符等。按照参与运算的操作数的个数，运算符分为单目运算符、双目运算符和多目运算符，这里的单、双和多就是指操作数的个数。

对于 C++初学者，一定要分清的三个概念是运算符、表达式和程序语句。运算符在前面已经介绍过了。表达式就是由运算符和操作数组成的一个序列，它们可以嵌套。表达式一般都有一个返回值，而程序语句是指包含“;”的一条完整的符号序列。程序语句有定义语句、表达式语句等。例如：+、−、* 和 / 是运算符，a+b*c+2 是一个表达式，而 2+3*4+2; 是一个语句；int a; 是一个定义语句；a=2+3*4+2; 是一个表达式语句。

1.2.1　算术运算符和赋值运算符

C++提供的算术运算符有 +、−、*、/ 和 %，它们都是双目运算符。其中除了%的操作数只能是整型数外，其他运算符的操作数可以是 int 型、char 型、float 型和 double 型。+、−、*、/ 代表通常意义上的加、减、乘和除运算。%的意义是取左操作数除以右操作数的余数。例如 7%3 的结果就是 1。

C++提供的赋值运算符有 =、+=、−=、*=、/=、++、−− 等。其中 = 是最常用的一种赋值运算符，它的作用就是将一个变量、常量或表达式的值赋给赋值运算符左侧的变量。

例如：

```
int a;
a=2+3;
```

将表达式 2+3 的值赋给了变量 a。赋值运算符左侧的标识符称为“左值”，赋值运算符右侧的表达式称为“右值”。变量可以作为左值，但常量不能作为左值，因为常量不能被赋值。很多表达式也不能作为左值，比如 a+5 就不能作为左值。

另外要注意的一点是赋值表达式作为左值时应加括号。例如：

```
int a;
(a=2)=3+a;
```