章理 李凤云
路平 编著

C++程序设计语言

机械工业出版社

C++程序设计语言

章理　李凤云　路平　编著

机械工业出版社

(京)新登字 054 号

C++程序设计语言是C语言的增强版。本书全面地介绍了C++语言的全部内容，除详解了C++语言的一般特性外，更注重C++语言的专用特性。全书共分 4 部分：1) C++语言介绍部分；2) C++语言基础部分；3) C++语言特性部分；4) C++语法。本书内容丰富，层次分明，结构严谨。

本书适用于广大计算机用户及软件开发人员，同时可作为大专院校师生和自学者的参考用书。

C++程序设计语言

章理　李凤云　路平　编著

*

责任编辑：王中玉　责任校对：丁丽丽

封面设计：姚　毅

*

机械工业出版社出版（北京阜成门外百万庄南街一号）

邮政编码：100037

（北京市书刊出版业营业许可证出字第 117 号）

北京市通县达明印刷厂印刷

新华书店北京发行所发行・新华书店经售

*

开本 787×1092 1/16・印张 19 3/4・字数 499 千字

1993 年 6 月北京第 1 版・1993 年 6 月北京第 1 次印刷

印数 0 001—6 300・定价：11.00 元

*

ISBN 7-111-03742-1/TP・184

前　言

近些年来，面向对象编程(OOP)技术被计算机界的专家学者们一致认为是程序设计方面的一场实质性革命，面向对象编程技术已在科研人员中广泛推广，并在广大程序员中广泛使用。

C++作为一种语言充分地运用了面向对象编程技术的主要内容，使程序员可以使用某些强功能的面向对象编程技术。可以说，自C语言问世以来，C++是编程艺术和科学中的最重要进展，它将改变人们编写程序的方式。

在程序设计方面，C++向前迈出了重要一步。基于C语言，C++添加了扩展支持面向对象编程，这些扩展戏剧性地增强了C语言的性能。不过，它不仅仅适用于面向对象编程，而且还因其先进特性使得编写非面向对象程序也更为方便。

在编写程序时，C语言是专业程序员所选择的语言。由于C++是C语言的增强版本，因而可以预料，在今后几年中，C++的普及率在我国将会突飞猛进地增长。事实上，许多专家学者预测。90年代将是C++的年代。坦率地说，C++的功能和通用性，加之已普及的C语言定会使其大获成功。

本书全面而细致地详解了C++的一般特性和C++的专用特性。然而，更注重于C++的专用特性，因此，无论是对初级程序员，还是对有经验的程序员都定会有所裨益。

编　者

目　录

第1章 C++介绍

在我们开始介绍C++这一程序设计语言之前，请读者切记这样一句话：掌握一种新的程序设计语言的唯一方法就是用该语言编写程序。

本章将简要地介绍一下C++程序设计语言的主要特性，以便使读者有个初步印象。

我们先提供一个C++程序，说明C++程序如何编译和运行，并说明这样的程序如何产生输出并读取输入。之后，提供C++的更常规特性：基本类型，说明，表达式，语句，函数和程序结构。其余部分说明C++如何定义新类型，数据隐藏，用户定义算符和用户定义类型的结构。

1.1 介绍

C++是一种面向对象的程序设计语言，且其面向对象特性是高度相互关联的。可以说，C++是C语言的一个超集，因此，有关C的一切特性都适用于C++。

在我们介绍C++程序之前，先介绍一下面向对象的程序设计。

1.1.1 面向对象的程序设计

面向对象的程序设计(Object - oriented Programming，缩写为OOP)就是要产生一种与现实具有自然关系的软件系统，而现实就是一种模式。面向对象的程序设计中的关键就是模式。程序员的责任就是构造现实的软件模式。面向对象的程序设计是完成编程作业的一种新型方法。

自计算机问世以来，编程方法戏剧性地发生了变化。这一变化主要是为了适应并解决程序越来越复杂的状况。由二进制机器指令到汇编语言，由汇编语言到高级语言，进而发展到结构化语言，C和Pascal语言便是结构化语言。

结构化语言有时非常适用于编写一定难度的复杂程序，不过，一旦编程项目达到某一程度，使用结构化的编程方法就变得无法控制。其复杂程度达到了程序员无法管理的地步。

在编程开发的过程中，人们所建立的方法允许程序员处理日益复杂的问题，其中，每前进一步，新方法都沿用了以前方法的最佳成份。如今，许多程序设计停留在不再适用的结构化编程方法上。为解决这一问题，人们发明了面向对象的程序设计语言，而C++便是其中之一。

所有面向对象的程序设计语言都有三个公共部分：对象、多态性和继承。

1.1.1.1 对象

面向对象语言的最重要特性是对象。简而言之，对象(object)是一种逻辑实体，它含有数据以及处理该数据的代码。在对象中，有些代码和/或数据可能私用于该对象，且不得由该对象外的任意成份存取。这样，对象提供一个有意义的保护级，以防止程序中不相关部分偶尔修改或不正当使用该对象的私用部分。代码和数据的这一链接常称为封装(encapsulation)。

事实上，对象是用户定义类型的变量。初看起来它有些怪(把链接代码和数据的对象看作变量)。然而，在面向对象的程序设计中，这是一种极为精确的表现形式。当定义对象时，便隐

式地建立了一种新的数据类型。

1.1.1.2 多态性

面向对象语言支持多态性(polymorphism)。我们可以这样解释多态性:一个名字可适用于几个相关但稍有不同的目的。实际上,多态性允许一个接口与一般操作类一起用。所选定的操作由数据类型确定。接口的一般概念是:向堆栈推入数据,并从堆栈推出数据。

1.1.1.3 继承

继承(inheritance)是一种进程。通过这一进程,对象可获得另一对象的特性。

1.1.2 输出

我们先编写一个仅含有输出行的程序:

```
#include <stream.h>
main()
{   cout << "Hello,world \n";
}
```

#include <stream.h>的作用是告知编译程序:在文件 stream.h 中所发现的标准流输入和输出的说明。没有这些说明,表达式 cout << "Hello, world \n "就没有意义。二目算符"<<"将其第二个变元写入其第一个变元中(在该例中,串 " Hello, world \n "写入标准输出流 cont 中)。串是由双引号括起来的字符序列。串中的反斜线字符\后面跟另一个字符,相当于一个字符表示,一种操作功能。在该例中,\n 是换行符,因此,所写入的字符是 Hello, world 和换行。

该程序的其余部分

main(){…}

定义 main 函数。每个程序必须有 main 函数。且程序通过执行该程序得以启动。

1.1.3 编译

输出流 cout 和实现输出算符"<<"的代码来自何处?C++程序必须编译,以产生可执行代码;编译过程基本上与 C 程序一样,且共享多数程序。读取并分析程序文本,如果未发现错误,便生成代码。之后,检验程序,寻找已使用过但未定义的名字和算符(在该例中为 cont 和 <<)。

1.1.4 输入

如下转换程序提示用户输入数值。输入数值后,打印相应结果:

```
#include <stream.h>

main()
{
        int inch=0;
        cout<<"inches=";
        cin>>inch;
        cout<<inch;
```

```
    cout<<"in=";
    cout<<inch*2.54;
    cout<<"cm\n";
}
```

main()函数的第一行说明一个整型变量 inch 。其值是用>>算符读取的。cin 和>>的说明在<stream.h>中。

1.2 注释

有时，为了便于阅读程序，人们将注释插入到程序中，而编译程序忽略这一注释、在C++中可以使用两种方法进行注释。

第一种方法：字符/*作为注释的开头，而字符*/作为注释的结尾。整个序列等效于一个空白间隔字符。这种方法适用于多行注释，但切记，/* */注释不嵌套。

第二种方法：字符//作为注释的开头，直到行尾结束。整个序列等效于空白间隔字符。这种方法适用于短小注释。

1.3 类型和说明

每个名字和表达式都有一种类型。类型用于确定执行何种操作。例如：

```
int value;
```

指定 value 是 int 类型；即 value 是一个整型变量。

说明就是一条语句，将名字介绍给程序。说明指定该名的类型。类型定义名字和表达式的适当用法。像+，-，*和/这样的操作定义为整型操作。

对象的类型不仅确定执行何种操作，而且确定这些操作的含义。例如，语句：

```
cout << inch << "in=" << inch * 2.54 << "cm\n";
        第1种    第2种      第3种      第4种
```

以不同的方式正确地处理了4种输出值。

C++拥有多种基本类型，且有多种方法创建新类型。

1.3.1 基本类型

基本类型有：

char　　int　　float　　double　　void

前2种类型用于表示整数，第3和4种类型用于表示浮点数。void 类型是由 ANSI 标准添加的。它有两种用途：一是显式说明函数不返回值，二是用于建立类属指针。

除 void 类型外，各种基本类型的前面都可添加不同的修饰符。修饰符用于改变基本类型的含义，使之适合更为确切的情况。下面列出几个修饰符：

```
signed
unsigned
long
short
```

这 4 种修饰符适用于整型和字符型。而 long 也适用于 double 类型。

下面这个表列出了所有数据类型和修饰符的合法组合，以及位宽和最小范围。

请注意，signed int 虽是多余的，但这样组合却是合法的，因为在缺省情况下，整型说明假定为带符号数。

类型	位宽	最小范围
char	8	−127～127
unsigned char	8	0～255
signed char	8	−127～127
int	16	−32767～32767
unsigned int	16	0～65535
signed int	16	与 int 一样
short int	16	与 int 一样
unsigned short int	16	0～65535
signed short int	16	与 shortint 一样
long int	32	−2147483647～2147483647
signed long int	32	0～4294967295
float	32	3.4E−38～3.4E+38
double	64	1.7E−308～1.7E+308
long double	128	3.4E−4932～3.4E+4932

用单引号括起来的字符是字符常量，这样定义的常量不需要占用存储器；其值仅可直接用在必要的地方。常量必须在说明时初始化。对变量来说，初始化是可选的。

算术算符适用于这些类型的任意组合：

+	一目加和二目加
−	一目减和二目减
*	乘
/	除

如下算符也是合法的：

==	等于
!=	不等于
<	小于
>	大于
<=	小于等于
>=	大于等于

注释：整数除法产生整型结果：7/2 为 3。%算符用于产生余数：7%2 为 1。

在算术操作及赋值操作中，C++执行基本类型间所有有意义的转换，这样它们可随意组合：

```
double    d=1;
int       i=1;
```

```
d=d+i;
i=d - i;
```

1.3.2 派生类型

依据基本类型可建立许多派生类。在C/C++中有如下派生类：

- 指针
- 引用
- 数组
- 枚举类型
- 类类型(包括结构和联合)
- 常量

1.4 表达式和语句

C++拥有丰富的算符集合，它们用在表达式中以产生数值并更改变量值。语句用于指定程序中的控制流，而说明用于将变量名、常量等引入程序。注意，说明是语句，因此，可与其它语句混用。

1.4.1 表达式

表达式由算符、常量和变量构成。C/C++语言中表达式可以是这些元素的任意合法组合。一般来说，表达式都是遵循数学上的书写形式。只有少部分是C/C++语言所特有的。

C/C++拥有大量算符，我们将在适当的时候加以解释。不过，应注意如下算符：

~	变反
&	与
^	异或
|	或
<<	逻辑左移
>>	逻辑右移

这些算符适用于整数，而算符的含义依赖于数值操作数；一目 & 是地址算符，而二目 & 是逻辑与算符。算符的含义也依赖于其操作数的类型；表达式 a+b 中的"+"算符可表示浮点加，也可表示整数加。如果操作数是 float 类型，则是浮点加；如果操作数是整型，则是整数加。

C/C++拥有一个赋值算符=。该算符在某些语言中为赋值语句。例如：a=b=c 表示将 c 赋值给 b，之后赋值给 a。赋值算符的另一概念是：可以与二目算符一起合并到赋值算符中。例如：x [i+3] * =4 表示 x [i+3]=x [i+3] * 4。

在多数C/C++程序中，指针被广泛地使用。一目算符 * 引用指针；即：* P 是 P 所指的对象。这种操作也称为间接操作。例如，根据 char * p，* p 是 p 所指的字符。递增算符++和递减算符--常适用于指针。假如 p 指向数组 v 的一个元素，那么 p++使 p 指向下一个元素。

ANSI C 标准约定，表达式的子表达式求值的顺序未指定。这意味着 C/C++编译程序可

随意重新编排表达式，以产生最佳代码。这还意味着，代码从不依赖于表达式的求值顺序，例如，如下表达式

$$x=f_1()+f_2();$$

并不能确保 $f_1()$ 在 $f_2()$ 之前被调用。

此外，混合于同一表达式中的不同类型常量及变量，均应变换为同一类型的量。编译程序将所有操作数变换为最大类型操作数的类型。转换的具体规则如下：

如果两个操作数中任意一个是 long double 类，则另一个操作数也要转换成 long double 型。

(1)否则，如果其中一个是 double 型，则另一个也转换成 double 型。

(2)否则，如果其中一个是 float 型，则另一个也转换成 float 型。

(3)否则，如果其中一个是 unsigned long 型，则另一个也转换成 nusigned long 型。

(4)否则，如果其中一个是 long 型，则另一个也转换成 long 型。

(5)否则，如果其中一个是 unsigned 型，则另一个也转换成 unsigned 型。

此外，如果一个操作数是 long 型，而另一个是 unsigned 型，且 unsigned 的值不能由 long 型表示，那么两个操作数都可转换为 unsigned long 型。

一旦使用这些规则，每对操作数的类型应一样，且每次操作的结果应与操作数的类型相同。

1.4.2 表达式语句

表达式用于形成称为语句的其它结构，语句则是用分号结尾的表达式。语句分多种形式：赋值、说明，if 语句，switch 语句，迭代语句，转移语句和复合语句等。最常见的语句形式是表达式语句。如：

```
a=c+2*d;
cout<<"goto";
lseek(fd,0,2);
```

1.4.3 null(空)语句

最简单的语句就是 null 语句，null 语句什么也不执行。但有时需要程序空转时，null 语句是很有用处的。

1.4.4 块

块语句只是一组被视为一个单元的相关语句。构成块的语句逻辑上合为一体。块以{开头，并以}结束。例如：

```
{a=b+2;b++;}
```

1.4.5 if 语句

if 语句的一般形式是：

```
if(表达式)语句;
else 语句;
```

根据 C 语法，语句可由如下一种语句构成：单语句，语句块，或空语句。else 是可选的。

如果 if 表达式为真，则执行构成 if 目标的语句或块。否则执行构成 else 目标的语句或块。切记，仅执行与 if 有关的代码或与 else 有关的代码。

根据 ANSI 标准，控制 if 的条件表达式必须产生一个标量结果。标量可是整型、字符型、浮点型或指针型。不过，很少用浮点值控制条件语句，因为执行时间是相当慢的。它使用 CPU 执行浮点操作。

if 语句可以嵌套，甚至可以多重嵌套。ANSI 标准规定，必须至少支持 15 级嵌套。在实际编程中，绝大多数编译程序允许更多嵌套。

比较常见的程序设计结构是阶梯型的 if－else－if，其一般形式如下：

```
if(表达式)语句;
else
if(表达式)语句;
else
if(表达式)语句;
.
.
.
else 语句;
```

在这个结构中，自上而下地对条件进行测试。当某个条件为真时，便执行与此条件有关的语句，并且越过阶梯型的其余部分；如果无一条件为真，便执行最后一个 else。最后一个 else 常作为缺省条件处理。换句话说，如果其它条件测试都失败的话，便执行最后一个 else 语句。如果最后一个 else 不存在，且所有条件为假，那么，不执行任何操作。

如下程序说明了如何使用 if 语句。该程序执行英寸到厘米，以及厘米到英寸的转换。通过添加 i(英寸)或 c(厘米)指出输入内容的单位(注意：if 语句中的条件必须用括号括起来)：

```
#include <stream.h>

main()
{
    const float fac=2.54;
    float x,in,cm;
    char ch=0;

    cout <<"enter length:";
    cin >> x >> ch;

    if (ch == 'i') { //inch
        in = x;
        cm = x * fac;
    }
    else if (ch == 'c') { // cm
        in = x/fac;
        cm = x;
    }
```

```
else
    in = cm = 0;

cout << in << " in =" << cm << " cm\n";
```

1.4.6 switch 语句

C或C++语言有一个多路选择语句，称为switch语句。它根据某些整数或字符常量对某个表达式进行连续测试。当某一常量与其值相匹配时，执行与之相应的语句或语句块。switch语句的一般形式为：

```
switch(表达式){
    case    常量1:
            语句序列
            break;
    case    常量2:
            语句序列
            break;
    case    常量3:
            语句序列
            break;
     .
     .
     .
    default:
            语句序列
}
```

表达式的值依次与case语句中指定的常量值进行测试比较。当出现匹配时，执行与该case相关的语句序列，直到遇见break或switch语句尾为止。如果未出现匹配，则执行default语句。default是可选的。如果default不存在，那么当所有匹配失败时，不执行任何操作。

ANSI标准规定，switch至少可有257条case语句。在实际编程中，应限制case语句的数量。尽管ANSI标准把case划分为标号语句，但在switch外，它不能独立存在。

break语句是C/C++的一种转移语句。在循环语句或switch语句中可使用break语句。当在switch中遇到break时，程序执行跳至switch语句后面的代码中。

有关switch语句，有三点应加以注意：

(1)switch语句与if语句的不同之处在于只能对等式进行测试，而if可计算关系表达式或逻辑表达式。

(2)在switch语句中，任意两个case的常量均可拥有相同的值。不过，互相嵌套的switch语句可有相同的常量。

(3)如果在switch语句中使用字符常量，它们将自动转换为整型值。

switch语句中的break语句是可选的。它通常用于终止与每个常量有关的语句序列。如果

省略 break 语句，便执行下一条 case 语句，一直遇到 break 或 switch 的末尾为止。

此外，switch 语句可以嵌套。

如下举例说明了如何使用 switch 语句：

```
switch (ch)
case' i' :
    in=x;
    cm=x*fac;
    break;
case' c' :
    in=x/fac;
    cm=x;
    break;
default:
    in=cm=0;
    break;
}
```

1.4.7 while 语句

while 语句的一般形式为：

while (条件) 语句 ;

其中语句可是空语句、单语句或语句块。条件可是任意表达式，且真值可是任意非零值。当条件为真时，循环重复执行；当条件为假时，程序从循环代码后的第一行开始执行。

例如，拷贝字符串，通过转换，该书的结尾字符为整型值 0。

```
while(*p!=0){
    *q=*p;          //copy character
    q=q+1;
    p=p+1;
}
*q=0;               //terminating 0 not copied
```

while 后的条件必须括起来，求条件的值。如果其值为非零，执行紧跟其后的语句。重复这一操作，直到所求的值为零时结束。

该例有些冗长。算符++可用于增值操作，而测试可以简化：

```
while (*p) *q++=*p++;
*q=0;
```

其中，结构 *p++ 表示“取 p 所指的字符，之后 p 增值。”

该例还可进一步压缩，因为每当循环时，指针 p 被两次间接引用。字符拷贝可在测试条件时进行：

```
while(*q++=*p++);
```

其中，取 p 所指的字符，p 增值，将该字符拷贝到 q 所指的位置处，q 增值。如果字符为非零，重复循环。由于所有操作是有条件地操作，所以不需要语句。在此，使用了空语句。

1.4.8 do－while 循环

do－while 语句对循环条件的检查是在循环尾部执行的。这就是说，do－while 循环至少执行一次操作。do－while 循环的一般形式为：

```
do {
    语句
}while(条件);
```

虽然当仅有一条语句时一对花括号是不必要的，但花括号增强了程序的可读性，可以避免混淆(这里所谓的混淆是针对读者而言的，与编译程序无关)。如下程序使用 do－while 语句。用户从键盘输入数据，直到发现有一个不大于 100 的数字时才结束。

```
do {
    scanf("%d",&num);
}while(num>100);
```

1.4.9 for 语句

for 循环的一般格式常出现于所有过程化程序设计语言中。for 循环拥有很大的灵活性和非常强的功能。for 循环的一般形式如下：

for(初始化；条件；增值)语句

初始化用于设置循环控制变量的赋值语句；条件用于确定何时终止循环操作；增值用于控制变量的变化情况。这三部分必须用分号隔开。只要条件为真，for 循环便一直执行下去，一旦条件为假，程序就从紧跟在 for 循环语句后面的语句重新开始执行。

例如，我们将 10 个元素从一个数组拷贝到另一个数组中：

```
for(int i=0;i<10;i++) q[i]=p[i];
```

它等效于

```
int i=0;
while (i<10){
    q[i]=p[i];
    i++;
}
```

for 语句的第一部分不必说明。例如，如果 i 已说明，如下举例也是等效的：

```
for (i=0; i<10; i++) q[i]=p[i];
```

对于 for 循环，有一点非常重要：条件测试始终在循环开始时进行。这就是说，如果在循环开始时条件为假，那么根本不执行循环中的代码。

1.4.10 return 语句

return 语句用于从函数返回。ANSI 标准指定它为转换语句，因为它使执行返回到调用函数的地方。如果 return 有一个与它有关的值，那么该值是函数的返回值。在 C 语言中，未说明为返回 void 的函数从技术上说不必返回值。如果未指定返回值，则返回无用值。不过，在C++语言中，未说明返回 void 的函数必须返回值。

return 的一般形式如下：

return 表达式;

1.4.11 goto 语句

一般来说,我们不提倡使用 goto 语句,但使用 goto 语句有时便于某些编程环境。

goto 语句需要一个操作标号。标号是一个带冒号的有效标识符。ANSI 标准称这种结构为标号语句。进一步说,标号必须与 goto 用在同一函数中,不能在函数之间转移。goto 的一般形式是:

```
goto 标号;
.
.
.
标号:
```

其中,标号是任意有效标号。它可在 goto 之前。也可在 goto 之后。如下程序使用了 goto 和标号:

```
x=1;
loop:
x++;
if(x<100) goto loop;
```

1.4.12 break 语句

break 语句有两种用途:一是终止 switch 语句中的 case 语句,如本章前面所述;二是立即中断循环操作,避开正常的循环条件测试。

当在循环中遇到 break 语句时,循环立即终止,程序从循环语句后的第一条语句执行,如下例:

```
#include"stdio.h"
main()
{
    int t;
    for(t=0;t<100;t++){
    printf("%d",t);
    if(t==10)break;
}
return 0;
}
```

该程序在屏幕上打印数字 0～10,之后终止,因为 break 导致程序从循环立即退出。显然 t<100 仍成立,但避开了 for 语句的条件测试。

在 switch,语句中使用的 break 仅影响该 switch 语句,而不会影响它所在的任何循环。

1.4.13 continue 语句

在许多方面,continue 语句是 break 语句的补充,但 continue 语句不是强迫终止,它强迫循环操作的下一次递归执行,跳至其中的任意代码。对 for 循环来说,continue 语句导致条件测试,之后执行循环的增值部分。对 while 和 do－while 来说,程序控制回到条件测试。