C程序员的C++指南

SHARAM HEKMATPOUR 著
闾 龙 译
金茂忠 校

北京航空航天大学出版社

C程序员的C++指南

C程序员的C++指南

SHARAM HEKMATPOUR 著

阎 龙 译

金茂忠 校

北京航空航天大学出版社

(京) 新登字 166 号

内 容 简 介

《C 程序员的 C++指南》一书给出了关于 C++程序设计语言支持功能的简明扼要的描述．本书对于那些想要尽快掌握 C++的 C 程序员，以及涉及软件工程和面向对象程序设计课程的教师和学生是一本理想的教材和参考书．

本书包括以下主要内容:

- 四个完整详细的实例研究用来说明 C++与面向对象程序设计在有价值的问题中的应用．
- 关于 C++所有主要内容的简要介绍．包括有多继承、指针及动态存储．
- 每章后的小结强调了各章概括的主要问题以及复杂概念的图示说明．
- 章节后的练习，并在附录中提供答案．

本书既可以作为计算机专业本科生和研究生的教材，也可以作为计算机软件人员、广大科技工作者的参考资料．

C 程序员的 C++指南

C CHENGXUYUAN DE C++ ZHINAN

SHARAM HEKMATPOUR 著
阎 龙 译 金茂忠 校

责任编辑 韦秋虎

*

北京航空航天大学出版社出版
新华书店总店科技发行所发行 各地新华书店经销
北京航空航天大学软件工程研究所微机输入排版、激光打印
朝阳科普印刷厂印刷

*

787×1092 1/16 印张: 13.5 字数: 345 千字
1992 年 9 月第一版 1992 年 12 月第一次印刷
印数: 1－6000 册 定价: 10.00 元
ISBN 7-81012-375-0/TP・085

前言

近几年来，面向对象程序设计的日益普及导致了很多支持面向对象的程序设计语言的出现。C++就是一种这样的语言。C++是由 Bjarne Stroustrup 于 1986 年在 AT&T 的贝尔实验室开发的。C++可以支持真正的数据抽象与封装，使它成为对于大型和复杂软件系统开发的极具吸引力的语言。因为 C++是 C 程序设计语言的超集，所以 C 程序员可以很快地掌握 C++。

本书的目的在于向 C 程序员介绍在 C++中的 C 语言所不具有的功能。因此，要求对 C 语言具有适当的实际了解。本书适用于在科学与工程专业的已经掌握了一些 C 程序设计的研究生／本科生，以及在工业领域的 C 程序员。

本书共分为两个部分。第一部分包括第一至第九章，这一部分详细描述了 C++的功能。在每章中都介绍一个主要功能，并随后通过示例加以说明。我们鼓励读者去试验这些例子并尝试完成练习题。大部分练习答案的示例已在附录中提供。

第一章介绍了类的构成并描述了 C++程序的组成和编译方法。第二章描述了构造函数和析构函数——两个特殊类型的成员函数，用于创建和删除类的对象。第三章说明了友元的使用，它可以允许非成员函数访问类的私有部分。函数和运算符的重载在第四章中做了讨论，这一章还描述了用户定义的类型转换规则。第五章介绍了引用的使用以及引用与指针的比较。第六章描述了新的类如何从现有的类中派生而来。第七章区分单继承和多继承，并详细讨论了多继承。有关指针的使用以及动态分配内存块的问题在第八章中讨论。第九章描述了 C++的其它功能，这包括结构、类属类以及函数库支持的 I/O 功能。

本书的第二部分包括第十至第十三章。这部分描述了较大一些的实例研究。这些实例研究是自包含的，并且对于有些有价值的问题提供了解决方法。这些研究的目的是用来说明 C++作为面向对象程序设计语言的使用方法。建议读者对 C++语言有了充分了解后，再去尝试这些实例研究。

第十章描述了 B 树及其变体作为类的实现。本章中所开发的派生类说明了虚函数的典型使用。第十一章描述了一个内存管理算法的实现。并在后来把它用派生类扩展为包含可重定位的内存块。第十二章把 C++的功能应用于设计基于窗口的用户界面管理程序。该程序用四个类实现。第十三章通过给出在字处理应用中具有八个类的层次结构，说明了多继承的使用。

在本书中出现的 C++示例、练习答案以及代码段均已通过 GNU C++的编译和测试。并尽可能避免了关于具体实现特定功能的使用。

本书主要术语英汉对照

base class	基类
class	类
class herarchy	类层次
constructor	构造函数
derived class	派生类
destuctor	析构函数
friend	友元
generic class	类属类
inheritance	继承
inline	内置的
inline function	内置函数
member	成员
data member	数据成员
function member	函数成员
member object	成员对象
multiple inheritance	多继承
name resolution	名字解析
object	对象
object-oriented	面向对象
overload	重载
private member	私有成员
protected member	受保护的成员
public member	公有成员
reference	引用
single inheritance	单继承
static member	静态成员
stream	流
virtual base class	虚基类
virtual function	虚函数

目　录

第一章 类

本章首先介绍在 C++中有关定义新的数据类型的类的构造．并通过一些简单示例来解释类定义的主要构成．这些示例还显示了 C++的新的功能，如：enum、const 以及 inline，它们可以被用作替代 C 的宏定义来提高可读性．最后将讨论 C++程序的组成与编译．

1.1 引 言

用 C 语言和其它类似的传统语言所写的程序必然包含一组数据结构以及用来处理这些数据结构的例程．由于这些语言没有提供数据抽象的功能，所写的程序通常对哪些例程处理了数据结构不太清楚．此外，这些语言也无法防止没有授权访问某些数据结构的例程对这些数据结构进行操作．从而使大的程序变得不必要的复杂，并且很多偶然的设计错误被忽视了．

利用能够提供数据结构的封装（encapsulation）与抽象（abstraction）以及相关操作等功能的程序设计语言将对程序设计的方法产生重大影响．它提供程序员在问题定义时进行更仔细的研究，理清以往被忽略的问题的相关性．这就使得那些包括自包含单元（self-contained units）（称为数据类型（data type））的程序具有简单并且定义良好的接口．

类的构造是 C++对 C 语言的重要增强．它为程序员提供了定义新的数据类型的简单而强大的工具．数据类型包含以下两个部分：

- 这一类型的对象的具体描述
- 处理该对象的一组操作集合

此外是对它的一些限制．即：除非指定的操作，其它操作都不能处理这个对象．由于这个原因，我们经常说操作被类型化了，就是类型决定了什么可以与不可以对对象操作．由于同一原因，这样相应的数据类型通常被称为抽象数据类型（abstract data type），抽象是因为对象的内部表示方式对那些不属于该类型的操作是隐藏的．

所有可用的程序设计语言都包含很多内部定义数据类型．例如：浮点数具有该普通内部类型的四种算术运算：加、减、乘、除． C++与很多其它语言的区别在于它能定义新的数据类型使之具有与内部类型无法区分的使用方式．通过这一方式，一个程序（或至少其中一部分）可以成为该语言的自然扩展而不是与之分离的部分．

使用适当的数据类型抽象具有三个重要收益．首先，采用这一风格所写的程序由于程序整体复杂性被大大降低，程序容易理解和修改．其次，由于某一数据类型的对象只能由它的私有操作来处理，错误将被局部化，所以很多偶然的设计错误将会避免．最后，由于每个数据类型都是与其它无关的自包含实体，因此一个设计任务就可以被划分为很多可独立完成的子任务．这样对于那些包括很多设计人员和程序员的大型系统是很重要的．

1.2 类的构成

在 C++中，新的数据类型可以用 class 来构造．class 声明的语法与 C 语言中的 struct 声

明相似，只是前者还可包含函数声明．例如：

```
class Point {
    int   xVal, yVal;
public:
    void  SetPt(int, int);
    void  OffsetPt(int, int);
};
```

例子中声明了一个叫作 Point 的新类．它包含了两个数据成员（xVal 和 yVal）以及两个函数成员（SetPt 和 OffsetPt）．这四个成份统称为类的成员（member）．Point 由两个部分组成：私有（private）部分和公有（public）部分．这两部分被 public 关键字分隔．在私有部分出现的成员只能被函数成员访问，在公有部分出现的成员可以被类的使用者访问．换句话说，公有部分指定了类的接口（interface）．

函数成员的实现通常不是类的一部分，并且在别处出现（inline 除外）．例如，SetPt 和 OffsetPt 可以定义如下：

```
void Point::SetPt(int x, int y)
{
    xVal = x;
    yVal = y;
}

void Point::OffsetPt(int x, int y)
{
    xVal += x;
    yVal += y;
}
```

在这些定义中，在每个函数成员的名字前都加上 Point ::来指示它属于哪个类．这样就避免了不同类的函数成员具有相同名字时的二义性．函数成员可以自由访问类的私有部分：SetPt 和 OffsetPt 都引用了 Point 的私有部分 xVal 和 yVal．

一旦类按这种方式定义，它的名字就表示一个新的数据类型，并允许我们定义这种类型的变量①．

```
Point  pt;                  // pt is an object in class Point
pt.SetPt(10, 20);           // pt is set to (10, 20)
pt.OffsetPt(2, 2);          // pt becomes (12, 22)
```

函数成员的调用使用点表示法：pt.SetPt(10, 20)对 Pt 对象调用了 SetPt，即：Pt 是 SetPt 的隐含参数．

由于 xVal 和 yVal 被包含在类的私有部分，我们可以保证类的使用者不能直接处理它们．

---

①在C++中，除了C语言表示注释的方法外（即：/*…*/），任何出现在//后到该行末的正文都被作为注释．

```
pt.xVal = 10;                    // illegal
```

这将无法通过编译．因此，如果 Point 类型的一个对象在有些时候出现问题，我们就知道在那里寻找错误的原因；即一定是错在某个函数成员．C 语言则无法强制这种限制，如果出了问题，错误的原因就不明显．特别是当程序很大，并包含很多复杂的连接时．

在此，我们应该清楚地区分类和对象．类表示一个类型，属于该类型只有一个类．对象是一个特定类型（类）的一个元素，属于该类的对象可以有很多个．例如:

```
Point  pt1, pt2, pt3;
```

定义了三个对象（pt1，pt2 和 pt3）属于同一类（Point)．此外，类的操作作用于该类的对象而不是类本身．因此，类只是一个没有存在实体并由它的对象来反映的概念．

1.3 内置函数

类的每项操作都是通过函数成员实现的，使用某项操作则表明了一个函数调用．对于相对复杂的操作，函数调用的开销还是可以接受的．但对于小而常用的操作，这样的开销就是非常大的．这个问题可以通过定义函数为内置的（inline）来解决．这将使该函数的代码被插入在函数的每个调用处（即采用保留其语义的方式)．因此，可以避免函数调用机制所带来的开销．

在 Point 类中，两个函数成员都非常短（仅有两条语句)．把它们定义为内置函数可以大大地提高效率．一个函数可以通过把 inline 关键字插入函数定义的前面而使它被定义为内置函数．

```
inline void  Point::SetPt(int x, int y)
{
    xVal = x;
    yVal = y;
}
```

任何函数包括全程函数（即该函数不是任何类的成员）都可以成为内置函数．对于函数成员，把它定义为内置的简单方法就是把它的定义包含在类定义中．

```
class  Point {
    int  xVal, yVal;
public:
    void  SetPt(int x, int y)          { xVal = x; yVal = y; }
    void  OffsetPt(int x, int y)       { xVal = x; yVal = y; }
};
```

1.4 例子：集合的实现

集合是对象的一个无序组合．利用构造类，我们可以把集合定义为一个新的数据类型．为

了使简单化，我们限制集合为有限整数元素的集合．有了这个限制，集合可以作为一个静态整数数组实现．我们还需要记录集合中元素的个数（即：集合的基数（cardinality））①．

```
class  Set {
       int  elems [maxCard];              // set elements
       int  card;                         // set cardinality
public:
       void       EmptySet( )      { card = 0; }
       Bool       Member( int );
       ErrCode    AddElem( int );
       void       RmvElem( int );
       void       Copy( Set * );
       void       Equal( Set * );
       void       Print( );
       void       Intersect(Set *, Set *);
       ErrCode    Union(Set *, Set *);
};
```

maxCard 表明集合中元素个数的最大值，它被定义为一个常量．在 C++（以及 ANSI C）中，常量定义可以使用 const 来构成，这将比使用#define 更为方便．

```
const  maxCard = 16;          // similar to :    #define maxCard 16
```

Set 中的一些函数成员返回 Bool 或 ErrCode 结果，这些结果被定义为枚举类型．

```
enum  Bool { false, true };
enum  ErrCode { noErr, overflow };
```

这两行的作用与以下相同:

```
const  false      = 0;
const  true       = 1;
const  noErr      = 0;
const  overflow   = 1;
```

使用枚举的好处在于提高了可读性；在 enum 后定义的名字还可以在以后的程序中作为类型名来使用．其实，这个名字并非引入一个新的类型，而只是 int 类型的一个同义词．因此，Bool 和 ErrCode 都成为 int 的别名（有关枚举和常量定义的细节，参见 9.1 节）．

第一个函数成员 EmptySet 直接被定义为内置函数．它只是简单地设置集合的基数为 0．第二个函数成员 Member 检验给定的一个整数是否为集合元素．

```
Bool  Set:: Member (int elem)          // test if elem is a member of set
{
    for ( int i = 0;  i < card;  ++i )
```

①还有更好的方法来定义集合，我们将后续章节中作介绍．

```
        if ( elems [ i ] == elem )
            return  true;
    return false ;
}   / * Member * /
```

注意局部变量 i 的声明是在 for 循环的内部．在 C++中，变量的声明可以出现在语句能出现的任何位置．变量的定义域是从它的声明之处起到它所在的包含块为止①．Member 遍历集合中的所有元素与 elem 相比较，直到找到匹配的元素或所有的元素都不匹配为止．

AddElem 为集合增加一个元素．如果该元素已在集合中存在则不做任何操作．否则，如果数组有空间则插入该元素．

```
ErrCode  Set :: AddElem (int elem)          / / add elem to the set .
{
    for ( int i = 0;  i < card;  ++i )
        if ( elems [ i ] == elem )
            return  noErr ;
    if (card < maxCard) {
        elems [ card ++ ] = elem ;
        return  noErr ;
    } else
        return  overflow ;
}   / * AddElem * /
```

RmvElem 是 AddElem 的逆向操作．如果指定元素在集合中存在，它就删除该元素．

```
void  Set :: RmvElem (int elem)                 / / remove elem from the set
{
    for ( int i = 0;  i < card;  ++i )
        if ( elems [ i ] == elem ) {
            for ( ;  i < card--1;  ++i )        / / shift element left
                elems [ i ] = elems [ i+1 ];
            --card ;
        }
}   / * RmvElem * /
```

Copy 完成集合的复制．这个函数的参数是目标集合的指针．正如函数中表示的，我们可以通过递引用（dereference）这个指针来引用类的私有部分．

```
void  Set :: Copy (Set * set)                   / / copy a set into another
{
    for ( int i = 0;  i < card;  ++i )
        set->elems [ i ] = elems [ i ];
    set->card = card ;
}   / * Copy * /
```

Equal 比较两个集合的等价性．如果两个集合包含相同的元素则它们是相等的（记住：元

① i 的定义域除 i 的出现之外，还包括包含循环的块，即整个函数体．

素的顺序与等价性无关)．注意 Equal 如何调用 Member 来检查 elems[i]是否为 set 的成员．

```
Bool  Set::Equal (Set * set)                    //test if two sets are equal
{
    if (card != set->card)
        return false;
    for (int i = 0; i < card; ++i)
        if ( !set->Member (elems [i]) )
            return  false;
    return  true;
}  /* Equal */
```

Print 采用传统的数学表示方法打印集合的内容．例如：一个集合中包含 5, 2 和 10 三个数字，就被打印为{5, 2, 10}．

```
void  Set::Print ( )                            // print a set as { ··· }
{
    cout << "{";
    for ( int i = 0;  i < card-1;  ++i )
        cout << elems [i] << ",";
    if ( card > 0 )                             // no comma after the last element
        cout << elems [card-1 ];
    cout << "}\n";
}  /* Print */
```

在 Print 中，输出采用 cout << expr 形式的语句来完成．在此，cout 是 C++的缺省输出流(流的介绍在 9.6 节)．如果我们采用原来 C 语言输出风格来写 Print，它将是以下形式：

```
void  Set::Print ( )                            // in old C style
{
    printf ("{");
    for ( int i = 0;  i < card-1;  ++i )
        printf( "%1d,",  elems[i] );
    if ( card > 0 )
        printf( "%1d", elems[card-1] );
    printf ("}\n");
}  /* Print */
```

这在 C++中仍为合法代码．但使用 cout 形式要更为简洁一些（还有争论)．

练习 1.1

为 Set 引入一个名为 Card 的新成员，用于返回集合的基数．把 Card 编码成内置函数．

练习 1.2

编写函数成员 Intersect 和 Union 代码．这两个函数都是进行集合 s1 和 s2 的元素比较，并产生第三个集合 s3．Intersect 返回 s3，其中包含所有既在 s1 中又在 s2 中的元素，Union 返回 s3，其中包含在 s1 或者在 s2 中的所有元素．例如：集合{2, 5, 3}和{7, 5, 2}的交集为{2,

5}，并集为{2, 5, 3, 7}。

练习 1.3

为 Set 引入 Subset 和 PSubset 两个新成员。Subset 在集合 s1 的所有元素都是另一集合 s2 的元素时返回真，否则返回假。PSubset 与 Subset 相同，只是在 s1 与 s2 相等时返回假。

类 Set 可用一个简单的驱动程序进行测试。如同在 C 中，C++的执行也是从 main 函数开始。下面的 main 函数创建了三个集合并调用了 Set 的各种函数成员。

```
main ( )                                    // a simple test for the Set class
{
    Set  s1,s2,s3;
    s1.EmptySet ( );   s2.EmptySet( );   s3.EmptySet( );
    s1.AddElem(10);   s1.AddElem(20);   s1.AddElem(30);   s1.AddElem(40);
    s2.AddElem(30);   s2.AddElem(50);   s2.AddElem(10);   s2.AddElem(60);
    cout << "s1 = ";                        s1.Print ( );
    cout << "s2 = ";                        s2.Print ( );
    s2.RmvElem(50);
    cout << "s2-{50} = ";                   s2.Print ( );
    if ( s.Member (20) )
        cout << "20 is in s1\n";
    s1.Intersect (&s2, &s3);
    cout <<"s1 intsec s2 = ";               s3.Print ( );
    s1.Union (&s2, &s3);
    cout << "s1 union s2 = ";               s3.Print ( );
    if ( !s1.Equal (&s2) )
        cout << "s1 != s2\n";
}   /* main */
```

如果我们把 Set 类和 main 函数包含在一个文件中，这一文件的组织应如下：

```
#include  <stream.h>
const  maxCard = 16;
enum  Bool { false, true };
enum  ErrCode { noErr, overflow };

class  Set {
    ...
};
... function members ...

main ()
{
    ...
}
```

只要把 const 和 enum 的声明放在类之前，其它的排列也是可行的。在文件中的第一行是一条 include 语句，从而使 stream.h 头文件的内容被包含在相应位置。你应在使用 C++ I/O

功能的每个文件中包含 strcam.h 文件，这种用法与 C 语言中 stdio.h 头文件的使用相同。

1.5 编　译

要进行 C++程序编译，应遵从 C++编译所提供的指令。确切的命令随安装系统的不同而异。以下的示例只是用作一个说明。

基于 UNIX 的 AT&T 的 C++编译程序命令是 CC。假设以上章节中编写的程序文件名为 set.cc，以下的两个命令将完成程序的编译和运行①。

```
$ CC  set.cc
$ a.out
s1 = {10, 20, 30, 40}
s2 = {30, 50, 10, 60}
s2 - {50} = {30, 10, 60}
20 is in s1
s1 intsec s2 = {10, 30}
s1 union s2 = {30, 10, 60, 20, 40}
s1 /= s2
$
```

当 UNIX 的提示符 $ 出现后，用户输入上文中的黑体部分。第一行使 set.cc 被编译。在 UNIX 中，编译程序产生的执行程序被放在名为 a.out 的文件中。以这个文件名作为命令名输入就运行了这个程序，并产生上面的输出结果。

在 SUN 工作站上运行 GNU C++编译的命令为 g++。其余部分与前面相同。

```
$ g++  set.cc
$ a.out
…  output as above …
$
```

与 C 编译一样，基于 UNIX 的 C++编译也允许使用编译选项。例如：-o 选项使执行代码存在用户指定的文件中；-c 选项允许多个文件的分别编译。

练习 1.4

输入 set.cc 中的程序并进行编译。如果程序运行，按下面的建议把代码分成三个文件存放。

```
set.h:    #include <stream.h>
          const和enum声明
          类Set的声明
set.cc:   #include "set.h"
          函数成员的定义
```

①在AT&T和GNU C++中，源文件应以.c、.C或.cc为后缀，我们在本书中采用.cc的习惯用法。

main.cc: #include "set.h"

main的定义

分别编译这些文件，并把执行代码存在文件 set 中，然后运行这个程序。

```
$ g++ -c set.cc main.cc
$ g++ set.o main.o -o set
$ set
...
```

1.6 小 结

一个（抽象）数据类型由两个部分组成：该类型对象的具体描述及表示这一类型特征的一组私有操作。

新的数据类型定义通过使用 class 来构成。类的声明包括两个部分：私有部分（仅类型操作可以访问）和公有部分（构成类对外部世界的接口）。类型的操作被定义为函数成员。

函数成员（实际上任何函数）都可以被声明为内置的。这种作法被典型用作小而常用的函数，以避免函数调用的开销。

常量和枚举可以使用 const 和 enum 分别定义。这种用法比宏定义更为可取。

当一个函数成员被调用时，它接收被调用函数的对象作为缺省参数。

进行 I/O 操作的程序须包含 stream.h 头文件。类似地，用户定义的头文件可被用作把函数定义与常量、枚举、类型、宏、类声明分开。