计算机软件
应用系列

Borland C++
实用编程技术与范例

字符串处理技术
各类数组实现方法
窗口实现技术
图形绘制技术
数值积分求解和多项式计算方法

张 力 魏 民 张 海 编著

科 学 出 版 社

计算机软件应用系列

Borland C++

实用编程技术与范例

张力　魏民　张海　编著

科学出版社

1995

（京）新登字 092 号

内 容 简 介

本书将 Borland C++与实际编程任务结合在一起，具体介绍了使用 Borland C++进行实际编程工作所要用到的一些技术。全书共分九章，主要内容如下：字符串的处理技术、各类数组的实现方法及数组在统计方面的具体应用、数值积分及多项式的处理技术、窗口的实现技术以及各种实际图形的实现技术等。本书所涉及到的都是编程方面的基本问题，而且包含了大量的编程实例，对用 Borland C++进行编程的广大读者而言无疑会具有重要的参考价值。

本书适用于各类编程技术人员。

计算机软件应用丛书

Borland C++实用编程技术与范例

张力 魏民 张海 编著

责任编辑 刘晓融 [illegible]

科学出版社出版

北京东黄城根北街 16 号

邮政编码：100717

国防科工委印刷厂印刷

新华书店北京发行所发行 各地新华书店经售

*

1995 年 1 月第 一 版 开本：787×1092 1/16

1995 年 1 月第一次印刷 印张：15 1/2

印数：1—3 100 字数：356 000

ISBN 7-03-004251-4/TP·383

定价：18.80 元

前　言

C++是对C的面向对象扩展，它是目前广为流传的一种编程语言。C++不仅完全保留了C的低级、高效和灵活等特点，而且还全面支持了面向对象程序设计的特性，从而更适于复杂程序的开发工作。以软件著称的Borland International公司在Turbo C++编译器的基础上，陆续推出了功能强大的Borland C++ 2.0和Borland C++ 3.0集成开发环境，这两种C++版本是目前最为成功的C++版本。

本书结合大量实例，具体介绍了使用Borland C++进行实际编程工作时所要用到的一些技术，其中包括基本数据类型的定义、字符串处理技术、数组实现技术、窗口实现技术和图形绘制技术等，同时也介绍了Borland C++在统计和科学计算方面的具体应用。相信本书会对读者掌握Borland C++编程技术有所帮助。

本书是集体智慧的结晶，参加编写工作的有张力、魏民、张海、郝梦实、李旭东、常江和孙彩虹等。由于作者水平有限，加上时间仓促，书中一定有许多不足之处，欢迎广大读者批评指正。

1994年6月

目　录

第一章　Borland C++概述

1.1　C++的起源

由于C语言在绝大多数系统程序设计任务中能够替代汇编语言，并且具有高效、灵活、可移植性好等特点，所以目前已经成为使用最为广泛的一种编程语言。但在使用C语言的过程中也存在着问题，即当程序达到一定规模时（如几万行的程序），它就会变得相当复杂，很难组合成为一个整体，这样不仅不利于程序的开发，而且维护起来也更加费力。为解决这一问题，贝尔实验室的Bjarne Stroustrup于1980年对C语言进行了扩展，将Simula67中类的概念引入到了C语言中。起初，他将这一新的语言称为“带类的C语言”，而后又在Rick Maseitti的提议下于1989年将该语言的名字改为“C++”。在使用C++语言的过程中，Stroustrup与他的同事们又对该语言进行了两次修订，先后引入了运算符重载、引用和虚拟函数等特性，使它更加精炼，并于1989年底推出了AT&T C++ 2.0版。在这个版本中，C++不仅完全保留了C的低级、高效和灵活等特点，而且还全面支持了面向对象程序设计的特性，这样就为广大C程序员学习和使用C++带来了极大的方便。

以软件著称的Borland International公司也紧接着推出了自己的C++版本，它首先推出了基于AT&T C++2.0的Turbo C++编译器，而后又推出了功能更为强大的Borland C++2.0和Borland C++3.0集成开发环境。后面两种C++版本不仅可以用在DOS环境下，而且也可以用在Windows环境下，它们是到目前为止最为成功的C++版本。本书的全部内容正是基于Borland C++2.0版和3.0版展开的。

1.2　C++与C的关系

C++是从C语言扩展而来的，C语言则是目前使用最为广泛的一种编程语言。C语言得到广泛使用的原因在于以下几点：

(1) C语言是一种高级语言，很适合于表达各类算法。

(2) 使用C语言有利于编写出精巧、易读的程序。

(3) 可以很容易地将C程序移植到其他计算机系统中去。

(4) C语言拥有优秀的编译器。

(5) C语言在许多领域中都拥有优秀的软件工具。

(6) C编译器可以生成短小、高效的可执行代码。

(7) C语言允许用户存取和控制硬件特性。

(8) 使用C语言编程可以考虑到低级的细节内容。

但从某种角度来看，原始的C语言并不是一种真正的高级语言，因为用C语言编程

常常要考虑到低级的地址方面的内容，这样就会导致编写出来的C程序与算法本身在形式上有着较大的差异。而且，尽管使用C语言有利于编写出精巧、易读的程序，但这不是必然的，因为使用C语言同样也可以编写出繁琐、难懂的程序。

正是基于上面的原因，导致了C++的诞生。而将C扩展成C++主要就是为了支持高级语言特性，这可以从下列C++特性中得到验证：

(1) 类的概念：可以用来将数据结构的定义与操作的定义封装起来。

(2) 将一个特定类 K 看作一个通用类 G 的实例：这样可以编写例程来操作 G 中的所有对象并将这些例程作用于 K 中的特定实例。

(3) 重载函数名和运算符：这样可以使用类似的记号来表达不同类对象上的类似操作。

(4) 引用参数：可用来通过引用而不是值来传递函数参数，这样就可能避免显式地引用函数地址。

(5) 参数初始化：可用来为某些函数参数赋缺省值，并可以在调用序列中将这些参数省略掉。

(6) 流输入/输出：这样会导致C++中的输入/输出操作较之C中的标准输入/输出函数更为精巧。

从上述内容可以看出，C++不仅保留了C语言的原始精华，而且还具有了面向对象程序设计的基本特征。

1.3 C++的基本特征

在使用各种编程语言的过程中，我们会发现各种面向功能的编程语言之间并没有什么本质的差别。用户从BASIC语言转到C，FORTRAN及PASCAL语言并没有多大困难，条件语句就是条件语句，函数就是函数，这与所用语言无关。

而面向对语言的编程却不同于面向功能的语言的编程，这是因为面向对象的程序在结构上不同于面向功能的程序。面向功能的程序是围绕着要执行的动作来组织的，而面向对象的程序则是按照被操作的对象来安排的。

作为一种面向对象的语言，C++不仅为程序员提供了C语言的灵活控制，同时也具有了面向对象的功能。从C过渡到C++不只是加进了一些面向对象的特征，更主要的是引入了面向对象程序设计的方法，它要求程序员从围绕数据类型及其交互作用的编程模式中解脱出来，按照面向对象程序设计方法的要求来进行编程。

C++具有面向对象语言的全部基本特征，这些基本特征包括封装性、继承性和多态性，而这三个特征又都同类和对象有关。类是指用户定义的抽象数据类型，它与传统程序语言中的数据类型有着较大的区别：一个类在定义数据的同时也定义了对这些数据操作的方式，这些操作也称作方法(method)，类体现了数据抽象与功能抽象的统一。对象是指声明为某一类的变量，也可以把对象看作是某个类的实例。类中定义的基本数据描述了对象的状态，类中定义的数据操作方法描述了对象的行为和功能。同一类的不同实例的状态可能会有所不同，但功能却相同。因此，对一个类的每个实例，都要为其分配存储空间以保存对象的状态，而所有的实例又都可以共享实现操作的代码。

封装性是指属于同一类的对象都拥有这个类的特性和操作。封装后的类就类似于“软件芯片”，它使得根据已建立的基础类来开发软件的程序员无需了解这些基础类的实现细节，只需了解方法（类的外部接口）的参数和作用，就能很好地使用这些基础类，并在各种适用的场合下重复使用软件芯片，通过装配各种软件芯片来开发出软件产品。

继承性是指所有子类都可以继承父类的特性和操作。借助于继承，我们可以在已有类的基础上建立新类，不同对象中的共同性质只需定义一次，增添一项新的功能，不再需要从头开始编程序，而只需说明新功能与已有类的差别，也就是说只增加类中没有的新行为和新状态。这样就可以大大降低软件开发的复杂性和费用，并使得软件系统易于扩充。

多态性是指同样的操作对不同的对象有不同的意义，它与编译连接和运行连接两个过程直接相关。编译连接意味着在编译时就将对象和方法的代码连接起来，其优点是执行速度快，所需内存少，缺点是灵活性差；运行连接意味着在运行时才决定将方法和对象连接起来，其优点是具有相当大的灵活性，有利于建立类，便于重用和扩充，并且对运行速度的影响很小。C++既支持编译期间的多态性，也支持运行期间的多态性。编译期间的多态性是通过重载函数或运算符来实现的。将若干个功能相近的函数定义成同一个名字被称作函数重载，其意义在于降低程序的复杂性。使某个运算符对某一类具有特定含义被称作运算符重载，重载后的运算符仍保持其原来的含义。运行期间的多态性是通过派生类和虚拟函数实现的。虚拟函数是一种在基类中被定义的函数。由于指向基类的指针可以指向其任一派生类对象，因此可以在运行时根据指针所指向的对象来选择执行虚拟函数的不同版本，从而获得运行时刻的多态性。

1.4 Borland C++简介

继 Turbo C++之后，Borland International 公司又陆续推出了 Borland C++2.0 和 Borland C++3.0 两种C++版本，这是目前最为常用的两种C++版本。下面就分别对这两种C++版本做一简单介绍。

（1）Borland C++2.0

该C++版本保留了 Turbo C++编译器的所有性能，并且保留了独立的 Turbo Debugger，Turbo Assembler 和 Turbo Profiler。虽然 Borland C++2.0 不提供 Turbo Vision 和 ObjectWindows，但用它足以编写出各种 Windows 应用程序。

Whitewater Resource Toolkit (WRT) 使 Borland C++2.0 能够创建 Windows 资源（比如图标、光标、位图、菜单和字符串等），这种资源管理工具可以使 Borland 产品与 Windows 兼容型编译器一起工作。

（2）Borland C++2.0 & Application Frameworks

该软件简称为 Borland C++2.0 & AF。虽然它是运行在 DOS 下的集成开发环境，但它为专业开发人员提供了创建 DOS 和 Windows 应用程序的工具和环境。

Borland C++ 2.0 & AF 包含了 Borland C++2.0编译器、DOS 应用程序的类 Turbo Vision 以及 Windows 应用程序的类库 ObjectWindows，它是 Borland C++2.0平台下的最完整的开发系统。

(3) Borland C++3.0

Borland C++3.0为专业开发人员提供了一种集成开发环境，可以用来生成C++，DOS 和 Windows 代码。在 Borland C++3.0中，C++方面的水平相当于 AT&T C++2.1版。该版本既包含了 Borland C++3.0编译器，又包含了 Turbo C++ for Windows 编译器，并且支持基于 Windows 的用户界面，同时也提供了所有的 Windows 和 DOS 开发工具，Turbo Assembler，Turbo Profiler 和 Turbo Debugger，但没有提供 Turbo Vision 和 ObjectWindows。Borland C++和独立的调试器仍使用 DOS 的文本界面，只有 Turbo C++ for Windows 才是一个真正的 Windows 图形用户界面的应用程序。

(4) Borland C++3.0 & Application Frameworks

该软件简称为 Borland C++3.0 & AF，它对用户来说是最好的选择。它可以运行在 Windows 下，并且为专业开发人员提供了创建 DOS 和 Windows 应用程序的工具和环境(即 Borland C++3.0的内容)，同时它也包含了最新的 Turbo Vision 和 ObjectWindows 以及类库和 C 库函数的源代码。

到目前为止，Borland C++已经成为最为成功的 C++版本，其取得成功的主要原因在于以下几点：

1) Borland C++拥有非常友好的用户界面。

2) 使用 Borland C++可以按自己的需要来设置大多数特性。

3) 可以将语法错误在编辑器中标识出来。

4) Borland C++的编辑器可与当前最流行的字处理程序相媲美。

5) 可以从编辑器中控制源代码调试器。

6) Borland C++的库具有易理解、文档完备和可靠性高的特点。

7) Borland C++中包含一些特别适合于开发大型软件的标准特性。

8) Borland C++可以使用扩充内存来编辑、编译和连接大型文件。

9) Borland C++可在 Microsoft Windows 下运行并且含有用于开发 Windows 应用程序的工具包。

第二章　基本工具库

在具体介绍 Borland C++编程技术之前，首先介绍一些公用的基本工具。将这些基本工具单独提取出来，可以省去重新创建的重复工作量。

2.1　基本数据类型的定义

在 Borland C++中，布尔（Boolean）类型是一种最基本的数据类型，它反映了某件事是“真”还是“假”。而在实际工作中，开关（Switch）类型也是一种经常会遇到的基本数据类型，它反映了某件事是“开”还是“关”。尽管在技术上说来开/关值同真/假值是相同的，但二者的物理含义却不同。人们不会说一个灯是真还是假，也不会说一个逻辑变元是开还是关。

开关是怎样实现的呢？许多 C++程序员会写出类似下面内容的代码：

```
#define Switch int
#define    OFF   0
#define    ON    1
```

但上述设计由于缺乏保护机制，会导致将非法值赋给 Switch。而且，#define 常量是绝对的，在 C++中使用很不方便，因此得给出更好的定义方法。

尽管可以使用 C++中的类来定义 Switch，但最简单的方法是使用枚举类型来更好地实现 Switch：

```
enum Switch
{
    OFF = 0,
    ON = 1
};
```

枚举类型是容易被人忽略的 C++特性，实际上它是完成这一任务的最有效方法。利用枚举类型可以简洁地实现 Switch，无需提供构造函数和成员函数，它就能够提供保护措施来防止非法赋值，而且易于使用，同时不需要源文件来定义 ON 和 OFF 常量。检查赋值合法性的工作是在编译期间而不是在运行期间进行的。下面使用类似的方法定义 Boolean（TRUE/FALSE）数据类型：

```
enum Boolean
{
BOOL_FALSE,
BOOL_TRUE
};
```

既然 Borland C++也支持布尔值定义（非零值为“真”，零值为“假”），为什么还要使用枚举类型呢？这只能说是编程风格的需要，因为采用上述定义可以将 Boolean 型值

和 Switch 型值分别限制在一对说明性的标识符范围内。

Switch 和 Boolean 类型的定义是在 switch.h 和 boolean.h 文件中，具体内容如下：

```
//=========================================
// UTILITY LIBRARY
// boolean.h
//
//=========================================
#ifndef BOOLEAN_H
#define BOOLEAN_H

enum Boolean
    {
    BOOL_FALSE = 0,
    BOOL_TRUE = 1
    };

#endif
//=========================================
// UTILITY LIBRARY
// switch.h
//
//=========================================
#ifndef SWITCH_H
#define SWITCH_H

enum Switch
    {
    OFF = 0,
    ON = 1
    };

#endif
```

2.2 错误处理工具

在程序运行过程中，常常会出现各种错误（比如除数为零），这时就要对错误进行处理。由于错误处理是大多数程序所必需的，因此有必要定义一个类来进行错误报告。下面给出出错报告 ErrReporter 类的定义，该类可以用于错误处理：

```
class ErrReporter
 {
 public:
    ErrReporter (const String * lead);
    ErrReporter ();
    virtual void Warning (const String & msg);
```

```
    virtual void Fatal (const String & msg);
protected:
    virtual void MsgOut (const String & msg);
    String * Leader;
};
```

ErrReporter 由六个成员组成：构造函数，析构函数，三个虚函数和一个指向字符串的指针。

在决定怎样表达该类的时候，我们遇到了一个棘手的问题：ErrReporter 类使用了 String 对象，而在 String 类中又要用到 ErrReporter 对象！这种互相依赖关系在 C++中是很常见的，这使程序很难办：应当先说明哪一个类？这里我们首先给出的是 ErrReporter 类，因为它比较简单。

有关 String 类的定义，将在下一章中给出。这里只要求读者了解下述事情：

(1) String 对象可以由 const char *S 构造。

(2) String 可以用在 const char * 的位置上。

(3) String 类定义了复制构造函数。

(4) String 类定义了流 I/O 函数。

2.2.1 ErrReporter 的成员

ErrReporter 对象只有一个实例变量 Leader，Leader 是一个指向 String 对象的指针。当告知 ErrReporter 显示错误信息的时候，首先显示 Leader 所指向的 String。该 String 通过给出一个公共的标题正文来标识相关的错误信息。

ErrReporter 的构造函数创建了由 lead 参数所指向的 String 拷贝，所拷贝的串由 new 进行分配，并将其指针赋给 Leader。如果 lead 为 NULL，Leader 也被赋成 NULL。函数如下：

```
ErrReporter:: ErrReporter (const String * lead)
    {
    if ( lead == NULL )
      Leader = NULL;
    else
      Leader = new String ( * lead);
    }
```

为什么不是简单地将 lead 赋给 Leader 呢？因为如果指向对象的指针超出了类的范围却还要存储它，这是一种不好的编程习惯。如果 lead 指向的 String 被删除了，那么ErrReporter 对象中的指针就不合法了。

如果 Leader 不是 NULL，则由析构函数来删除 Leader 指向的 String：

```
ErrReporter:: ~ErrReporter ()
    {
    // delete leader if it was allocated
    if ( Leader != NULL )
        delete Leader;
```

```
}
```

ErrReporter 将其余三个函数定义成虚的，这三个函数都被实现成空的嵌入 (inline) shell：

```
void ErrReporter:: Warning (const String & msg)
    {
    // does nothing!
    }
    void ErrReporter:: Fatal (const String & msg)
        {
        // does nothing!
        }
    void ErrReporter:: MsgOut (const String & msg)
        {
        // does nothing!
        }
```

在显示错误信息的时候，程序或者调用 Fatal 或者调用 Warning。Fatal 用来终止程序，而 Warning 只是用来显示一条信息，尚需再创建第三个函数 MsgOut 来实现真正的信息显示。

使用虚函数的目的是为了在特定的环境下构造 ErrReporter 类。因为环境不同，显示信息的方式也可能有所不同。例如，DOS 的命令行程序常常将错误信息显示成可以滚动的正文行；而窗口环境中的程序要在一个窗口或显示区中显示错误信息。由于大多数对象并不知道它所工作的环境，因此用与当前环境相兼容的方式显示一条错误信息就只能由 ErrReporter 对象来完成了。

如果需要对象在多种不同的环境中以不同的方式进行工作，就要利用多态性。这里将 ErrReporter 类的错误显示函数定义成虚的，然后由 ErrReporter 派生出适用于各种特定环境的派生类，并在程序中定义指向 ErrReporter 对象的指针，最后将不同环境下 ErrReporter 的派生类地址赋给它们。对于由 ErrReporter 派生出来的所有类对象的引用，可以使用指向 ErrReporter 的指针来实现公共接口。

2.2.2 一个应用实例

为说明清楚前面的内容，下面用 C++流来定义一个名字为 DosErrReporter 的类：

```
class DosErrReporter : public ErrReporter
{
public:
    DosErrReporter (const String * lead = NULL,
                    ostream * strm = NULL);
    virtual void Warning (const String & msg);
    virtual void Fatal (const String & msg);
protected:
        virtual void MsgOut (const String & msg);
private:
    ostream * Destination;
```

```
};
```

DosErrReporter 由 ErrReporter 派生而来。它定义了四个函数来反映 ErrReporter 的特性，并且加入了一个新的数据元素 Destination，它是指向 ostream 的指针。Destination 没有自己的析构函数。

DosErrReporter 的构造函数有两个参数：String 指针和 ostream 指针。String 指针被传递给 ErrReporter 的构造函数，如果 strm 为 NULL，Destination 就被置成预先定义的 cerr 流，否则，Destination 就被置成 strm：

```
DosErrReporter:: DosErrReporter (const String * lead,
                                 ostream * strm)
    : ErrReporter (lead)
    {
    if (strm == NULL )
       Destination = &cerr;
    else
       Destination = strm;
    }
```

三个虚函数都可以用来将错误信息 String 发送给 Destination。Warning 和 Fatal 之间的区别是：Fatal 调用一出口并终止程序执行，而 Warring 允许程序继续执行。MsgOut 则用来显示信息。具体内容如下：

```
void DosErrReporter:: Warning (const String & msg)
    {
    * Destination<<" \nWARNING";
    MsgOut (msg);
    }
void DosErrReporter:: Fatal (const String & msg)
    {
    * Destination << " \nFATAL ERROR ";
    MsgOut (msg);
    exit (EXIT_FAILURE);
    }
void DosErrReporter:: MsgOut (const String & msg)
    {
    if ( Leader == NULL )
        * Destination << msg << endl; // no leader
    else
        * Destination << * Leader << ":" << msg << endl;
    }
```

如果 Leader 为 NULL，MsgOut 只显示 msg 串。如果 Leader 不为 NULL，MsgOut 显示 Leader 和一个冒号，然后是 msg。

2.2.3 ErrReporter 的使用

下面给出一个使用 ErrReporter 对象的例子，其中类 Foobar 用来报告 Foobar 对象的所有错误，它是通过一个名字为 ErrOut 的静态 ErrReporter 成员来实现的：

```
ciass Foobar
    {
    public:
        // various public members
        .
        .
        .
        static void SetErrOut ( const ErrReporter &er)
    private:
      static ErrReporter * ErrOut;
      static void ReportError ();
    };
```

ErrOut 的初始化工作只需在程序中间执行一次下述语句即可完成：

```
ErrReporter * Foobar :: ErrOut = NULL;
```

在 Foobar 对象产生错误时，就会调用 ReportError 函数：

```
void Foobar:: ReportError ()
    {
    if ( ErrOut != NULL )
      ErrOut. Fatal (" Error in Foobar object");
    }
```

只有在 ErrOut 指向一个 ErrReporter 对象时，ReportError 才会调用 ErrOut 来显示一条信息。ErrOut 的值是通过 SetErrOut 函数指定的。

```
void Foobar:: SetErrOut (const ErrReporter &er)
    {
    if ( ErrOut != NULL )
      delete ErrOut
    ErrOut = new ErrReporter (er);
    }
```

如果 ErrOut 已经指向了一个 ErrReporter 对象，该对象就要被删除。然后创建一个新的 ErrReporter 对象并将它分配给 ErrOut。

下面的程序片段就表明了对 SetErrOut 的调用：

```
int main ()
{
// create a DosErrReporter object
DosErrReporter * der =new DosErrReporter (" Foobar error");
// if the object was creared, call SetErrOut
if (der != NULL)
  {
  // assign object
  Foobar:: SetErrOut (*der);

  //delete unneeded object
  delete der;
```

```
    // more program
    .
    .
    .
}
```

ErrReporter 类和 DosErrReporter 类的定义和实现分别放在 err_rptr.h 和 err_rptr.cxx 文件中，具体内容如下（其中与本节内容重复者已被略去，只保留了注释）：

```
//=============================================
//  UTILITY LIBRARY
//     err_rptr.h
//
//     A class defining an object for reporting warnings and
//     errors
//
//=============================================
#ifndef ERR_RPTR_H
#define ERR_RPTR_H
class ErrReporter;
#include " stddef.h"
#include " iostream.h"
#include " str.h"
//---------------------------------------------
//  ErrReporter
//     Base class for error reporting
//---------------------------------------------
//---------------------------------------------
//  DosErrReporter
//     Class for reporting objects in DOS text-based program
//---------------------------------------------
#endif
//=============================================
//  UTILITY LIBRARY
//     err_rptr.cxx
//
//     A class defining an object for reporting warnings and
//     errors
//
//=============================================
#include " err_rptr.h"
#include " stdlib.h"
#include " string.h"
ErrReporter:: ErrReporter (const String * lead)
ErrReporter:: ~ErrReporter ()
void ErrReporter:: Warning (const String & msg)
void ErrReporter:: Fatal (const String & msg)
void ErrReporter:: MsgOut (const String & msg)
```