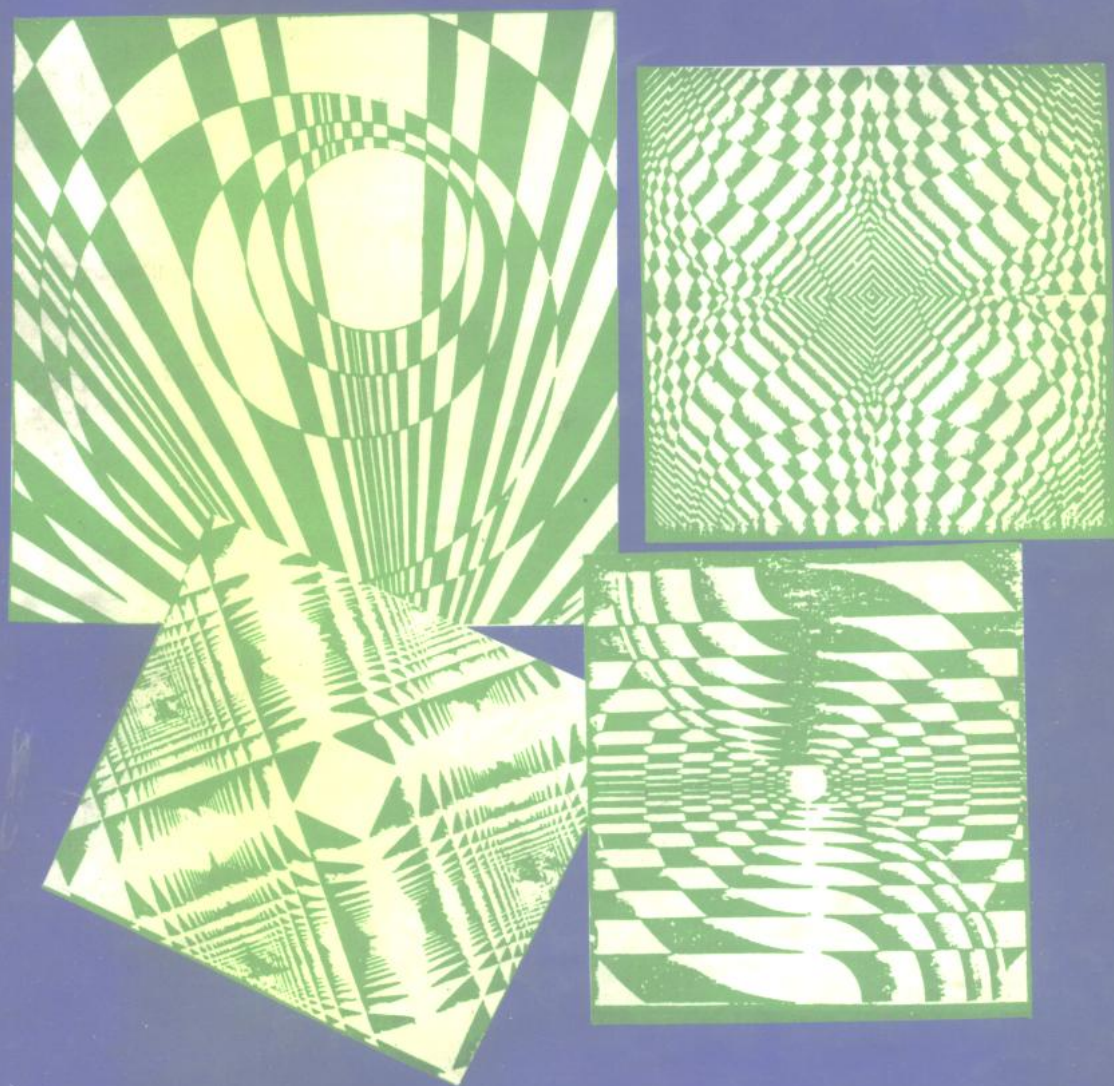


面向对象的 Windows 编程技术

王培杰 张应中 周金刚 编著



大连理工大学出版社

面向对象的 Windows 编程技术

王培杰 张应中 周金刚 编著

大连理工大学出版社

9510167

(辽)新登字 16 号

内容提要

本书系统地介绍了 Windows 提供的各种基本功能,并侧重介绍了面向对象的开发方法,且侧重在实际应用,以实际例子说明了如何把 Windows 提供的功能和函数应用在面向对象的程序设计中,本书在例子中开发的各种类都可作为以后实际开发应用程序的类库。

本书在介绍 Windows 功能的同时,也介绍了如何使用 ObjectWindows 类库,对比较重要的类进行了剖析,力求使读者对此类库有一个比较深刻的理解。

本书的例子是一个实际图形编辑程序,从最基本的 Windows 程序开始,每一章增加一些内容,最后完成一个可实际应用的图形编辑程序。此例子涉及到了开发一个实际应用程序需要的各种内容,同时,以实际程序作为例子也使读者在学习 Windows 功能和面向对象开发方法的同时,也学会了如何组织开发一个实际应用程序。

本书适用于有 C 语言或 C++ 基础,想学习面向对象的 Windows 开发的读者,本书例子的开发方法对 Windows 程序员也很有参考价值。

面向对象的 Windows 编程技术
Mianxiang Duixiang De Windows Biancheng Jishu

王培杰 张应中 周金刚 编著

大连理工大学出版社出版发行

(邮政编码:116024)

大连斯达电脑公司激光照排

大连理工大学印刷厂印刷

开本:787×1092 1/16 印张:39 $\frac{3}{4}$ 字数:915 千字

1994 年 12 月第 1 版

1994 年 12 月第 1 次印刷

印数:0001—5000 册

责任编辑:李鹤

责任校对:曲江

封面设计:孙宝福

ISBN 7-5611-0963-6

TP·64

定价:28.00 元

7810128

前 言

计算机的发展速度快的实在是惊人,今天买了一个最先进的计算机,一年之后可能是被淘汰的产品。计算机有如此之快的发展速度,主要归功于大规模集成电路的发展。

软件的发展虽然也很快,但和硬件相比还是慢了一些。这主要是不能象硬件那样可以利用以前的成果,每一个新的软件都要重头开始,就象硬件每个新的机型都要重新设计硬件电路一样。这给软件的发展带来了很大的困难。计算机界的专业人员一直想突破此障碍,现在已经得到了一定的解决,这就是面向对象的编程。

面向对象的编程方法被看做是今后计算机领域发展的热点之一,可称为软件开发领域的一次革命,现在为编程者越来越青睐。它把事物分解为一个个的模块,就象是一个个集成电路块,每个模块完成一定的功能,只要各个模块现成,我们就可以用它组合成一个新的应用软件。

Windows 系统大家都比较熟悉了,从应用的角度来说,它为微机提供了一个先进的操作系统,界面漂亮,使用方便,功能强大;从编程的角度来说,它是面向对象编程的一个典范,它为编程者提供了一个方便的环境,程序设计人员很容易用它设计出一个和 Windows 本身同样漂亮的应用程序,这实际上是利用了 Windows 提供的现成的模块,就象我们用现成的集成电路块组合成一个新的机器一样。

随着 Windows 系统应用的不断广泛,程序设计人员纷纷转向 Windows 系统应用软件的开发,也急需 Window 编程方面的参考书,虽然现在各出版社出版了不少这方面的书,但采用 C++ 编程的参考书并不多见,且大都从国外直接翻译过来,不太适合国内读者的口味。鉴于这种情况,根据作者这几年来在 Windows 应用软件开发方面的经验,编写了此书。

本书介绍如何用面向对象的方法来编写 Windows 应用程序。由于 Windows 本身就是用面向对象的方法写成的,因此,Windows 的应用程序很适合采用面向对象的编程方法来编写。本书力求通俗易懂,从入门开始,由浅入深,系统地介绍 Windows 提供的各种基本功能和 Borland C++ 提供的 ObjectWindows 类库,并侧重介绍了面向对象的开发方法。为说明问题,使读者易于理解,本书提供了大量的例子。这些例子涉及到了开发一个应用程序遇到的基本问题,这些例子也是由浅入深,最后形成一个完整的可以实际应用的应用程序。在本书例子中开发的各种类都可作为以后实际开发应用程序的类库。

本书适用于有 C 语言或 C++ 基础,想学习 Windows 应用软件开发的新学者,对于有用 C 语言开发 Windows 应用程序经验,想转到用面向对象的方法开发 Windows 应用程序的读者也很有帮助。

由于作者水平有限,书中难免存在错误和疏漏之处,恳请读者提出宝贵意见。

作者

1994 年 12 月

目 录

第一章 C++基础	1
1.1 C++要素	1
1.1.1 封装	1
1.1.2 继承	4
1.1.3 多态性	4
1.1.4 重载	5
1.2 用类描述现实世界	5
1.2.1 用类来描述点	5
1.2.2 成员的存取控制	6
1.2.3 构造函数和析构函数	8
1.2.4 成员函数的实现	9
1.2.5 对象的创建和使用	11
1.2.6 复合对象	12
1.3 应用继承	14
1.3.1 继承和访问控制	15
1.3.2 派生类的构造函数	16
1.3.3 派生类的成员函数	17
1.3.4 动态对象	20
1.4 C++的虚函数	23
1.4.1 虚函数的说明	24
1.4.2 虚函数的应用	25
1.5 C++中更多的灵活性	32
1.5.1 引用	32
1.5.2 类的静态成员	33
1.5.3 类的友元	34
1.5.4 操作符重载	35
1.6 使用C++的流	36
1.6.1 C++流概述	37
1.6.2 C++流类库结构	37
1.6.3 标准输入输出	38
1.6.4 格式化的输入输出	40
1.6.5 文件的输出	41
1.6.6 文件的输入	42

1.6.7 用户定义类型的输入输出	44
第二章 Windows 系统概述	48
2.1 Windows 的功能与特点	48
2.1.1 标准用户界面	48
2.1.2 多任务	49
2.1.3 设备独立性	50
2.1.4 动态连接	51
2.2 Windows 应用程序的用户界面	51
2.2.1 窗口	52
2.2.2 对话框和控制	53
2.3 Windows 的资源	54
2.3.1 图标	54
2.3.2 光标	54
2.3.3 插入符	54
2.3.4 位图	54
2.3.5 字体	55
2.3.6 画笔	55
2.3.7 刷子	55
2.4 Windows 编程的特点	55
2.4.1 事件驱动	55
2.4.2 应用程序和操作系统联系密切	56
2.4.3 图形界面	57
2.4.4 用户界面	58
2.5 面向对象的编程方法	58
2.5.1 面向对象的编程方法	58
2.5.2 Windows 面向对象的特点	59
2.5.3 面向对象的 Windows 程序设计	60
第三章 建立 Windows 应用程序	61
3.1 Windows 应用程序的组成	61
3.2 Borland C++ 3.1 集成环境	63
3.2.1 起动 IDE	63
3.2.2 文件操作	63
3.2.3 编辑操作	64
3.2.4 建立工程文件	65
3.2.5 设置编译和连接参数	65
3.2.6 编译连接应用程序	67
3.2.7 运行调试应用程序	68
3.3 Borland C++ 资源编辑器 Resource Workshop	68

3.3.1 Resource Workshop 如何组织资源文件	69
3.3.2 使用工程	69
3.3.3 Resource Workshop 的工程文件编辑窗口	69
3.3.4 创建资源文件实例	71
3.4 建立一个 Windows 程序	72
3.4.1 定义自己的应用程序类和窗口类	72
3.4.2 编写成员函数代码	73
3.4.3 建立模块定义文件	74
3.4.4 建立资源描述文件	74
3.4.5 建立工程文件	74
3.4.6 编译连接、运行 Sdraw	74
3.5 Windows 的编程风格	75
3.5.1 Windows 程序的命名规则	75
3.5.2 OWL 的命名规则	76
3.5.3 预处理命令的使用	77
第四章 Windows 编程要点	78
4.1 Windows 的管理机制	78
4.1.1 消息	79
4.1.2 消息的产生	79
4.1.3 消息的传递	79
4.1.4 消息的处理	80
4.1.5 Windows 的管理机制	80
4.2 Windows 应用程序的主程序	80
4.2.1 主程序的格式	80
4.2.2 Windows 中常用的数据类型	82
4.2.3 窗口类的注册	82
4.2.4 创建窗口	86
4.2.5 窗口的显示	88
4.2.6 建立消息循环	89
4.2.7 终止应用程序	90
4.2.8 完整的主程序	90
4.3 Windows 应用程序的窗口函数	92
4.3.1 窗口函数	92
4.3.2 窗口函数的声明	92
4.3.3 窗口函数对消息的响应	93
4.3.4 Windows 的模块定义文件	94
4.4 OWL 的应用程序对象类	95
4.4.1 ObjectWindows 类库(OWL)简介	96

4.4.2	ObjectWindows 的 TModule 类	98
4.4.3	ObjectWindows 的 TApplication 类	100
4.5	OWL 的窗口对象类	102
4.5.1	TWindowsObject 类	103
4.5.2	TWindow 类	106
4.5.3	OWL 的消息响应函数	107
4.6	OWL 的工作过程	109
4.7	建立自己的应用程序	111
4.7.1	SDraw 的应用程序对象类	111
4.7.2	SDraw 的窗口对象类	112
第五章	菜单和键盘加速键	114
5.1	菜单组成	114
5.2	定义一个标准菜单	116
5.2.1	用资源语句定义一个菜单	116
5.2.2	在 Resource Workshop 的 Menu 编辑器上定义菜单	118
5.3	在应用程序中加入菜单	120
5.3.1	为窗口类指定菜单	120
5.3.2	为特定窗口指定菜单	121
5.4	响应菜单消息	122
5.5	增强的菜单功能	128
5.5.1	菜单创建	129
5.5.2	菜单修改	131
5.5.3	位图作为菜单项	143
5.6	在菜单中加入键盘加速键	149
5.6.1	在菜单项里加入加速键正文	150
5.6.2	在资源描述文件中加入加速键表	150
5.6.3	装载加速键表	151
第六章	图形输出	160
6.1	图形设备接口(GDI)概述	160
6.1.1	GDI 的功能特点	160
6.1.2	GDI 设备描述表(DD)	161
6.1.3	逻辑绘图对象	165
6.2	在窗口里画图	166
6.2.1	画线函数	166
6.2.2	画填充图函数	167
6.2.3	在程序中应用绘图函数	169
6.3	使用绘图属性	177
6.3.1	绘图坐标系	177

6.3.2 绘图模式	178
6.3.3 颜色设置	180
6.3.4 笔	180
6.3.5 刷子	182
6.3.6 背景模式和背景颜色	184
6.3.7 多边形填充方式	185
6.3.8 和画线有关的设备描述表属性	185
6.3.9 和填充图有关的设备描述表属性	186
6.3.10 WM_PAINT 消息	186
6.3.11 应用示例	187
第七章 图标和光标	204
7.1 图标	204
7.1.1 自定义图标	205
7.1.2 在应用程序中装载图标	206
7.1.3 指定一个类图标	207
7.1.4 绘制一个图标	207
7.1.5 显示动态图标	207
7.1.6 在对话框内显示图标	208
7.1.7 图标显示程序	209
7.2 光标	213
7.2.1 控制光标的形状	213
7.2.2 显示光标	216
7.2.3 显示一个砂漏状光标的例子	217
7.2.4 鼠标器输入	217
7.2.5 应用程序实例	219
第八章 面向对象编程	235
8.1 对绘图程序的重新思考	235
8.2 建立图形对象类	236
8.2.1 图形类的基类	236
8.2.2 随意线类 TPen	238
8.2.3 直线类 TLine	239
8.2.4 圆类	240
8.2.5 TBox 类	240
8.2.6 矩形类和椭圆类	240
8.3 建立绘图工具类	241
8.3.1 绘图工具的基类	241
8.3.2 画笔工具 TPenTool	242
8.3.3 画线工具 TLineTool	242

8.3.4	TBoxTool	243
8.3.5	矩形工具类 TRectTool	243
8.3.6	椭圆工具类 TEllipseTool 和圆工具类	244
8.4	建立画布类	244
8.5	工具箱类	245
8.6	调色板类	247
8.7	修改后的绘图程序	249
第九章	对话框	279
9.1	对话框概述	279
9.2	创建对话框	281
9.2.1	对话框模板	281
9.2.2	用 WorkShop 建立对话框模板	284
9.2.3	显示模式对话框	285
9.2.4	显示无模式对话框	289
9.3	OWL 的对话框类(TDialog)	294
9.4	OWL 的控制类	297
9.4.1	按钮控制	297
9.4.2	静态控制(static control)	300
9.4.3	编辑框(edit control)	300
9.4.4	按钮和编辑控制应用示例	302
9.4.5	列表框(List Box)	310
9.4.6	组合框(ComboBox)	313
9.5	传递控制数据	315
9.5.1	控制的数据类型	316
9.5.2	定义数据传递缓冲区	319
9.5.3	构造控制	320
9.5.4	利用 ObjectWindows 的数据传递机制进行数据传递的例子	320
第十章	位图	332
10.1	位图简介	332
10.2	创建位图	333
10.2.1	创建和装载位图文件	333
10.2.2	在内存中创建位图	335
10.3	创建设备无关位图	337
10.3.1	设备无关位图的结构	337
10.3.2	创建设备无关位图	339
10.4	显示位图	342
10.4.1	使用 BitBlt 函数显示一个内存位图	343
10.4.2	放大、缩小位图	344

10.4.3 在模式刷子中使用位图.....	345
10.4.4 显示一个与设备无关的位图.....	347
10.4.5 应用实例.....	347
第十一章 文本输出	386
11.1 文本输出函数.....	386
11.1.1 TextOut	386
11.1.2 ExtTextOut	387
11.1.3 DrawText	388
11.1.4 TabbedTextOut	390
11.1.5 文本输出函数的应用.....	391
11.2 文本属性的控制.....	414
11.2.1 控制文本的颜色.....	414
11.2.2 控制文本的背景色.....	415
11.2.3 设置字符间距.....	416
11.2.4 设置文本的排列方式.....	417
11.2.5 文本属性控制实例.....	148
11.3 字体.....	419
11.3.1 物理字体.....	419
11.3.2 逻辑字体.....	420
11.3.3 使用备用字体.....	423
11.3.4 使用逻辑字体.....	423
11.3.5 旋转字体.....	425
11.4 获得文本信息.....	425
11.4.1 GetTextMetrics	426
11.4.2 GetTextExtent	426
11.5 使用字体及文本信息函数实例.....	427
11.6 键盘输入.....	460
11.6.1 Windows 字符的输入过程	460
11.6.2 虚拟键.....	461
11.6.3 翻译消息.....	463
11.6.4 WM_CHAR 消息	463
11.7 插入符.....	464
11.7.1 创建插入符(Caret)	465
11.7.2 插入符(Caret)的显示和隐藏	466
11.7.3 插入符的位置控制.....	466
11.8 字符输入应用实例.....	467
第十二章 滚动杠	505
12.1 Windows 的滚动杠	505

12.2	OWL 的滚动杠类(TScroller)	506
12.3	在窗口中加入滚动杠	508
12.4	自动滚动和跟踪	509
12.4.1	自动滚动	509
12.4.2	跟踪	509
12.5	修改滚动单位和范围	509
12.5.1	修改滚动范围	509
12.5.2	修改滚动单位	510
12.6	修改滚动的位置	510
12.7	设置页大小	510
12.8	应用实例	511
第十三章	流式类	543
13.1	流式类的构造	543
13.1.1	流式类的构造函数	544
13.1.2	流式类的建造器	544
13.1.3	流式类的写入器	545
13.1.4	流式类的读入器	545
13.1.5	流式类名字	546
13.1.6	重载输入输出操作符>>和<<	546
13.1.7	流式类的注册	548
13.1.8	链入流管理器代码	548
13.2	ObjectWidnows 的流	549
13.2.1	ObjectWidnows 流的结构	549
13.2.2	opstream 类	549
13.2.3	ipstream 类	550
13.2.4	文件输出	551
13.2.5	文件输入	553
13.3	流式类的应用及实例	554
13.4	流管理器的管理机制	617
13.4.1	流式类的基类 TStreamable	617
13.4.2	流式类引用输出操作符	617
13.4.3	流式类引用输入操作符	619
13.4.4	流式类指针输出操作符	620
13.4.5	流式类指针输入操作符	621

第一章 C++ 基础

C++ 是流行的 C 语言的扩充,其中增加了对面向对象程序设计(Object Oriented Program)的支持。面向对象的设计方法是最近非常流行的程序设计方法,它试图用客观世界中描述事物的方法来描述一个程序要描述的事物。自然语言中对现实世界经常见到的事物都有其相应的概念,例如房子,要描述一间房子我们要知道它的大小、形状等,同时,一提房子我们还知道它有哪些功能,这些功能是我们日常经验的总结并为大家所公认。其它的事物也是一样,如植物、动物等,而现实世界就是由这些事物组成的,它们各自按自己的规律运行并相互作用,相互影响。C++ 采用了这种对事物分类和抽象的方法,定义了一种新的数据结构——类,用类来描述一个对象,就象我们描述房子时有尺寸和功能一样,在类中即有描述此对象的数据,又有描述功能的函数,可以很完整地描述对象。因此,用 C++ 设计程序的思路和 C 语言是不同的,它更接近现实世界思维方式。

由于 C++ 中引入了类的概念,使程序的模块很独立,以后再用到此对象或类似的对象时不必改动任何源程序即可使用,这使软件界一直很关心的软件重用问题得到一定的解决,这是软件界很推崇 C++ 的一个很重要的原因。

C++ 的另一个特点是程序的可扩充性,这也是非常重要的一个方面。一般情况下软件开发的前期对整个并不很清楚,开发人员可根据自己的理解和掌握的工具进行程序结构设计,但是到软件开发的后期,开发人员往往发现开始的理解并不正确或不全面,但这时要进行大的修改就比较困难了,因为这要涉及到程序的结构,而要改变结构,其工作量就太大了。但 C++ 利用其继承特性和高度模块化结构可使程序提高可扩充性。

1.1 C++ 要素

这一节将介绍 C++ 的主要特点:继承、封装、多态性和重载,这也是和 C 语言完全不同的部分。本节介绍将帮助你尽快了解 C++ 并进入 C++ 的境界。

1.1.1 封装

在 C 语言程序设计中,我们一般注重的是程序代码,而把数据放在次要位置上。实际上,只有数据和这些数据对应的函数结合起来,才能完整地描述一个对象。比如我们可以把窗口看做一个对象,要描述它需要窗口的位置、大小、背景颜色、窗口风格等数据,同时窗口还有显示、隐藏、移动、改变大小等功能,只有这两方面结合起来才能完整地描述一个

窗口。在 C 语言中通常的做法是:定义一个结构来描述窗口的数据,再定义一些函数完成一些特定的功能,如:

```
struct Window
{
    int x;
    int y;
    int Length;
    int Width;
    int Style;
    int Color;
};

typedef struct Window TWindow;

TWindow AWindow;
void Show(TWindow * AWindow);
void Hide(TWindow * AWindow);
void Move(TWindow * AWindow,int NewPositionX,int NewPositionY);
void ResizeLength(TWindow * AWindow,int NewLength);
void ResizeWidth(TWindow * AWindow,int NewWidth);
```

一般情况下把定义的数据和代码放在一起编译,这当然是朝正确的方向前进了一步。但这也存在问题,因为这时我们定义的数据是全局的,其它的编程者可以随意使用、修改这些数据,如果另一个程序员直接对数据结构 Window 进行操作,修改了它的数据值,这时其它的程序员并不知道,显示器上也没有反映。

C++ 扩充了 C 语言 struct 和 union 的功能,并引入了新的数据类型 class,可以把数据和函数集成在一起,并可对数据和函数提供不同的存取权限,形成一个完整的模块,这就是面向对象语言中经常提到的封装。上面的问题可以用 C++ 的类来描述如下:

```
class Window
{
protected:
    int x;           //the x position of the window
    int y;           // the y position of the window
    int Length;      //the length of the window
    int Width;       //the width of the window
    int Style;       //the style of the window
    int Color;       //the back color of the window

public:
    Window(int x,int y);
    void Show();     //Show the window
    void Hide();     //Hide the window
    void Move();     //Move the window
```

```

void SetLength(int length); //Set the length of the window
void ResizeLength();        //Resize the length of the window
void ResizeWidth();         //Resize the width of the window

private:
    int IsShow; //如果窗口已经显示,IsShow=TRUE, 否则 IsShow=FALSE
};

Window AWindow(10,10);

```

从上面的例子可以看出,C++从外形上和C语言有如下不同:

1. 在C++的类(class)或结构(struct)、联合(union)中可以定义函数,在这里定义的函数叫此类的成员函数,而类中的数据叫此类的数据成员。

2. 在C++的类中引入了三个新的关键字:public、protected 和 private。此三个关键字用于说明数据成员或成员函数的存取权限。public 表示此后的数据成员或成员函数是公有的,也就是说其它的程序可以象C语言的结构那样对其存取,如:

```
AWindow.Show();
```

protected 表示此后的数据成员是受保护的,不允许其它的程序对它进行直接存取,只能通过它所在类的成员函数进行存取。如下面的操作是非法的:

```
AWindow.x=10; //非法
```

```
AWindow.Length=200; //非法
```

但可以通过函数操作,如:

```
AWindow.SetLength(200); //合法
```

private 表示后面的数据是私有的,和 protected 成员一样,不允许其它的程序对它进行直接存取,只能通过它所在类的成员函数进行存取。它和 protected 成员的区别是:protected 的成员是可以继承的,而 private 成员是不可继承的。

3. 在 Borland C++ 中,可以用斜杠//来表示C或C++中的一个单行注释,当然,C语言的/* */注释符仍然有效,实际上有很多行注释时用它还是很方便的。虽然这点改动不大,但给程序员带来很大方便,特别是//注释符可以嵌套在/* */注释符中。

4. 在C语言中,要定义一个数据类型要用 typedef,如前面的例子:

```
typedef struct Window TWindow;
```

在C++中,可以把结构名当做数据类型来使用,不必在用 typedef 定义。如:

```
Window AWindow(10,10);
```

在C++中,结构(struct)和联合(union)也可以有自己的成员函数,因此也可以用来描述一个对象,在这方面它和 class 很接近,但也有区别。

用 struct 定义的数据成员或成员函数缺省时是公有的,而用 class 定义的数据成员或成员函数缺省时是私有的,struct 和 class 只有这点区别,其它方面完全相同,比如在 struct 中也可以用 public、protected 或 private 来改变其成员的存取类型。

用 union 定义的类其所有成员都是公有的,这种安排是不能改变的,它不能使用存取限定符 public、protected 或 private。

通过上面的介绍可以看出,C++通过定义类把数据和函数集成在一起,外部程序只能

通过类规定的接口和类进行通讯,这很像硬件的集成电路块,每个集成电路块完成某些特定的功能,我们把这些集成电路块组合起来完成更复杂的功能。我们的目的也是尽量象硬件那样,能够利用已经调试好的模块组合起来完成更复杂的功能。 C^{++} 的封装功能使我们在这一方面前进了一大步。

1.1.2 继承

前面建立了一个窗口的类,它描述了窗口的最基本特征。在后续的程序设计中也许需要一个稍微有些不同的窗口,比如需要一个新的窗口 `NewWindow`,它带有标题,还要有一个边界,其它的定义和前面的窗口相同,如窗口大小、位置、显示、隐藏等。如果要描述这一新的窗口,当然可以建立一个新类,但这样和前面相同的操作也需要重新编写代码,这似乎不太合理。实际上也没有这个必要, C^{++} 为我们提供了解决此问题的机制,这就是继承(inheritance),它允许从已经存在的类中继承特征。只要告诉编译器你的新类是由另一个类继承而来的,它会把另一个类中除私有成员之外的所有成员,包括数据成员和成员函数,都赋给你的新类,就象你重新定义了一样,可以随意使用数据成员和成员函数。如果 `B` 类是由 `A` 类继承而来的,我们把 `B` 类叫做派生类(derived class),把 `A` 类叫做基类(base class),派生类也可以作为其它类的基类,这样一层一层的继承,可形成一个树状的类结构。

继承的概念很象植物学中对植物的分类。所有的植物有它的共性,我们用名词“植物”来描述它,但这不能完全描述具体植物,因为具体植物有它自己的特点,为了更清楚地描述它可以继续分类,比如再分成草本植物和木本植物,草本植物和木本植物还可以继续下分,这样就形成树状结构的植物分类。当谈到一颗树时,我们知道它是木本植物,因此具有木本植物的所有特征,同时我们知道木本植物属于植物,因此它也具有植物的所有特征。

在具体编程时,实际上就是建立这样的—个类结构,关键是怎样对描述的问题进行分类。有些类不必从头建起,因为已经有大量的为 Borland C^{++} 提供的类库,应该尽量使用现成的类库,这不仅节省编码时间,也节省调试时间,同时程序也可靠,因为这些类库是经过严格调试的。

1.1.3 多态性

上一节提到要用到一个新的窗口 `NewWindow`,它有一个标题还有一个边界,同时也提到新的窗口可以从老的窗口 `Window` 中继承而来。但这会出现一个问题,即在调用新窗口的 `Show()` 函数时,它显示的是老窗口 `Window` 的外形,我们需要的标题和边界并没有显示出来,这是因为 `NewWindow` 中的 `Show()` 函数是由 `Window` 类继承来的,这当然不是我们所要求的。 C^{++} 为我们提供了解决此问题的一个办法,这就是多态性。

多态性(polymorphism)是指多个函数具有相同的名字但具有不同的作用。比如我们可以在 `NewWindow` 类中重新定义一个 `Show()` 函数(这时要用 `virtual` 关键字),这样新的窗口类 `NewWindow` 中就有了两个 `Show()` 函数,具体用哪一个不是在编译时确定的,而是在程序运行时再决定,这叫迟后联编。

C^{++} 的这种多态性给我们带来了很大的灵活性,使我们重用以前的类又能满足新类

的需要,如果继承的函数和我们的要求有别,可以在不改变源代码的情况下满足我们的要求。

1.1.4 重载

在C语言中,一个给定的函数名只能对应一个函数,其它函数不能和它重名。例如,如果定义了一个计算面积的函数:

```
int area(int length,int width);
```

可以用它计算整数的面积,但要求一个长和宽为浮点数表示的面积,就不能用此函数了,需要重新定义一个函数:

```
float farear(float length,float width);
```

并且新函数不能用以前的函数名 area。这需要记住哪个函数需要什么类型的参数,和自然语言的概念不同。但在C++中,你可以使用相同的函数名,只要参数类型不同,C++就能根据其类型确定你所用的是哪个函数,所有求面积的操作只用一个函数名就可以了,如:

```
int arear(int length,int width);
```

```
float area(float length,float width);
```

这就是对 area 进行了重载。

在C++中,不但自己定义的函数可以重载,C++本身的操作符也可以重载,并且这是用得最多的一种重载。如在C中,+操作只能用于两个数字,但在C++中你可以重新定义它的含义,比如定义对复数的操作,当C++发现进行+操作的操作数是复数时,就自动调用我们自己定义的操作符。这大大增加了程序的易读性。

1.2 用类描述现实世界

从前面的介绍读者也许已经感觉到,C++语言很像通常我们描述现实世界的概念,事实上,开始时设计该语言的目的就是为了模拟一个大型电话交换系统。用C++很容易建立起现实世界的模型。下面以图形系统为例,介绍怎样使用C++提供的功能。

一个复杂的平面图形可以看成由简单图形组合而成,最基本图形为点、直线、圆、圆弧,由这些基本图形可以组成大部分图形。一条直线可以由两个端点来决定,因此可以由两个点来描述,而圆可以由圆心和半径来描述,因此平面上的一个点是图形系统的基础。

1.2.1 用类来描述点

类(class)的概念很象是C语言中结构(struct)的概念,是结构概念的扩充。它既允许把变量作为自己的成员(member)也允许把函数作为自己的成员,这就为我们完整地描述现实世界的对象提供了条件。

平面上的一个点可由一对坐标(X,Y)来决定,因此应把坐标的X值和Y值作为一个点类的数据成员,从外观上还需要一个点的颜色,另外还需要一个变量来表示此点是否是可见的,我们定义一个枚举类的逻辑变量类表示:

```
enum BOOL
```