

高校计算机教学系列教材

Delphi 面向对象 程序设计教程

唱江华 邓文新 等编著

北京航空航天大学出版社

内容简介

Delphi 作为一种当今最为流行的面向对象的编程工具,以其短小精悍、功能强大而著称。针对目前市面上的 Delphi 书籍多为使用手册形式,学习者读起来内容杂乱,不易得到要领的现象,本书编者结合自身的教学与程序开发经验,以教程的形式,循序渐进、由浅入深,结合相应的实例,介绍了 Delphi 的基础知识及其程序设计方法,并侧重介绍应用程序的开发过程。

本书共分两篇:上篇为 Delphi 程序设计,共 7 章,包括 Delphi 概述;基本的程序设计方法;图形、图像与多媒体设计;数据库程序设计;Internet 程序设计;定制组件及软件的集成技术。下篇为面向对象的程序设计基础,共 5 章,介绍了 Pascal 语言语法以及面向对象的程序设计概念。

本书既可以作为高校本、专科学生的教材,又可以作为相关领域的培训教材,对 Delphi 程序开发人员也有一定的借鉴作用。

图书在版编目(CIP)数据

Delphi 面向对象程序设计教程/唱江华等编著. 北京:北京航空航天大学出版社,2003.6

ISBN 7-81077-315-1

I. D... II. 唱... III. 软件工具—程序设计—高等学校—教材 IV. TP311.56

中国版本图书馆 CIP 数据核字(2003)第 024662 号

Delphi 面向对象程序设计教程

唱江华 邓文新 等编著

责任编辑 陶金福

责任校对 戚 爽

*

北京航空航天大学出版社出版发行

北京市海淀区学院路 37 号(100083) 发行部电话:(010)82317024 传真:(010)82328026

<http://www.buaapress.com.cn>

E-mail:bhpress@263.net

河北省涿州市新华印刷厂印装 各地书店经销

*

开本:787×1 092 1/16 印张:17.5 字数:448 千字

2003 年 7 月第 1 版 2003 年 7 月第 1 次印刷 印数:5 000 册

ISBN 7-81077-315-1 定价:24.00 元



高校计算机教学系列教材编委会成员

主 任: 赵沁平

副主任(常务): 陈炳和

顾 问: 麦中凡

委 员 (以姓氏笔划为序):

吕景瑜(北工大教授)

乔少杰(社长,副教授)

麦中凡(北航教授,教育部工科计算机基础教学指导委员会副主任、中专计算机教学指导委员会顾问)

苏开娜(北工大教授)

陈炳和(北工大教授)

张鸿宾(北工大博导)

郑玉明(北工大副教授)

金茂忠(北航博导)

赵沁平(北航博导,国务院学位办主任)



总 前 言

科教兴国,教育先行,在全国上下已形成共识。在教育改革过程中,出现了多渠道、多形式、多层次办学的局面。同时,政府逐年加大教育的投入力度。教育发展了,才能有效地提高全民族的文化、科学素质,使我们中华民族屹立于世界民族之林。

计算机科学与技术的发展日新月异,其应用领域迅速扩展,几乎无处不在。社会发展的需求,促使计算机生气蓬勃。从普通高校的系统性教学,到远距离的电视、网上教学;从全面讲述,到不同应用领域的、星罗棋布的培训班;从公办的到民办的;从纸介教材到电子教材等等,可以说计算机教学异彩纷呈。要进行教学,就必须有教材。

面对我们这么大的国家和教学形势,在保证国家教学基本要求的前提下,应当提倡教材多样化,才能满足各教学单位的需求,使他们形成各自的办学风格和特色。为此,我们组织北京工业大学、北京航空航天大学、北京理工大学、南开大学、天津工业大学等高校的有丰富教学经验的教师编写了计算机教学的系列教材,将陆续与师生见面。

系列教材包括以下各项。

(一) 基础理论:离散数学。

(二) 技术基础:电路基础与模拟电子技术;数字逻辑基础;计算机组成与体系结构;计算机语言(拼盘、选择使用),包括 C++ 程序设计基础、Visual Basic 程序设计基础、Matlab 程序设计基础、Java 程序设计基础、Delphi 语言基础、汇编语言基础等;数据结构;计算机操作系统基础;计算方法基础;微机与接口技术;数据库技术基础等。

(三) 应用基础:计算机控制技术;网络技术;软件工程;多媒体技术等。

(四) 技术基础扩展:编译原理与编译构造;知识工程——网络计算机环境下的知识处理。

(五) 应用基础扩展:计算机辅助设计;单片机实用基础;图像处理基础;传感器与测试技术;计算机外设与接口技术。

本系列教材主要是针对计算机本科和专科教学编写的,供普通高校、社会民办大学、高等职业学校、业余大学等计算机本科或专科选用。其中一部分教材也适合非计算机专业本科教学使用。在这些教材的内容简介或前言中对使用范围均作了说明。

本系列教材在编写时,注重以下几点:(1) 面对计算机科学与技术发展的现实,在内容上应具有前瞻性;(2) 面对学以致用,既要具有系统的基础知识,又要具有应用价值的实用性;(3) 具有科学性、严谨性。另外,力求排版紧凑,使有限的版面具有最大的信息量,以使读者得到实惠。

能否实现这些愿望,只有师生在教学实践中评价。我们期望得到师生的批评和指正。

高校计算机教学系列教材编委会



前 言

随着计算机技术的飞速发展,出现了一大批面向对象的开发工具,Inprise 公司推出的 Delphi 即是面向对象开发工具的一个优秀代表。它以其基于窗口和面向对象的程序设计方法,采用 Pascal 的语法结构、高速的编译能力,以及强大的数据库功能和网络功能,备受用户的喜爱,特别是 Delphi 采用面向对象的方法,使得用户所见即所得;它具有程序代码框架的自动生成功能,使用户可以把精力投入到实际的程序代码编写上。

本教材主要针对目前市面上的 Delphi 书籍多为使用手册形式,学习者读起来内容杂乱,不易得到要领的现象,编者结合自身的教学与程序开发经验,以教程的形式循序渐进、由浅入深,结合相应的实例,介绍了 Delphi 的基础知识及其程序设计方法,并侧重介绍了应用程序的开发过程。本书不仅适合于 Delphi 的初学者,对于专业的 Delphi 编程人员也具有一定的参考价值。

本书是以 Delphi 6.0 版为基础编写的。全书共分上、下两篇:Delphi 程序设计篇和面向对象的程序设计基础篇。考虑到一些阅读对象可能没有 Pascal 基础或是不了解面向对象的概念,本书特设面向对象的程序设计基础篇;如果有这方面基础的读者可以直接阅读上篇。上篇共分七章,包括 Delphi 概述、Delphi 基本的程序设计方法、图形图像与多媒体设计、数据库程序设计、Internet 程序设计、定制组件及软件集成技术等;下篇共分五章,介绍了 Pascal 语言的语法基础、控制语句、过程和函数、构造类型以及面向对象的程序设计概念。

本教材由齐齐哈尔大学的几位教师合作完成,其中,唱江华、邓文新、张益铎负责编写面向对象的程序设计基础篇,张银霞负责编写 Delphi 程序设计篇。

由于作者的水平有限,时间仓促,书中难免有疏漏或不当之处,敬请读者提出宝贵的意见。

编 者

2002 年 11 月

目 录

上篇 Delphi 程序设计

第 1 章 Delphi 概述

1.1 Delphi 的发展概况	2
1.2 Delphi 的集成开发环境 IDE 简介	3
1.2.1 主窗口	3
1.2.2 对象观察器(object inspector)	4
1.2.3 窗体窗口	5
1.2.4 代码编辑器和代码浏览器	5
1.2.5 对象浏览器(object tree view)	6
1.3 Delphi 程序设计的基本步骤	6
1.3.1 设计用户界面	7
1.3.2 编写事件处理过程	8
1.3.3 保存文件	9
1.3.4 编译运行应用程序	9
1.3.5 Delphi 程序设计的步骤	9
1.4 Delphi 程序的基本组成	10
1.4.1 项目文件.....	10
1.4.2 单元文件.....	10
1.4.3 窗体文件.....	12
1.5 Delphi 的类结构	14
1.6 Delphi 中的基本概念	15
1.6.1 组 件.....	15
1.6.2 特 性.....	15
1.6.3 方 法.....	16
1.6.4 事 件.....	16
习题与思考题	16

第 2 章 Delphi 的基本程序设计

2.1 窗体(form)设计	18
2.1.1 常用特性.....	18
2.1.2 窗体的事件.....	20
2.1.3 窗体的方法.....	20
2.1.4 应用举例.....	20
2.2 基本输入/输出组件	22



2.2.1	标签组件(label)	22
2.2.2	编辑框组件(edit)	22
2.2.3	几个显示对话框的过程和函数	23
2.2.4	程序举例	25
2.3	按钮组件	27
2.3.1	Button(按钮)组件	27
2.3.2	BitBtn(位图按钮)组件	27
2.3.3	RadioButton(单选按钮)组件	28
2.3.4	Checkbox(复选按钮)组件	28
2.3.5	GroupBox(组框)组件	29
2.3.6	RadioGroup(单选按钮组)组件	29
2.3.7	应用举例	29
2.4	其他常用组件	31
2.4.1	ListBox(列表框)组件	31
2.4.2	ComboBox(组合框)组件	32
2.4.3	StatusBar(状态条)组件	33
2.4.4	应用举例	33
2.5	常用对话框组件	35
2.5.1	OpenDialog(打开对话框)组件	36
2.5.2	SaveDialog(保存对话框)组件	37
2.5.3	FindDialog(查找对话框)组件	37
2.5.4	ReplaceDialog(替换对话框)组件	38
2.5.5	FontDialog(字体对话框)组件	39
2.5.6	ColorDialog(颜色对话框)组件	40
2.5.7	PrintDialog(打印对话框)组件	40
2.5.8	应用举例	41
2.6	菜单程序设计	47
2.6.1	常用菜单组件	47
2.6.2	菜单设计器	47
2.7	多文档界面 MDI 设计	49
2.7.1	创建父窗体和子窗体	49
2.7.2	MDI 窗体的特性与方法	51
2.7.3	多文档界面 MDI 中菜单的设计	51
	习题与思考题	53

第3章 图形、图像与多媒体设计

3.1	绘制图形	55
3.1.1	Canvas 对象	55
3.1.2	绘制各种图形	56
3.1.3	绘图组件	58
3.1.4	应用举例	59



3.2 图像处理.....	63
3.2.1 常用图像类.....	63
3.2.2 Image(图像)组件	64
3.2.3 应用举例.....	65
3.3 多媒体开发.....	67
3.3.1 声音处理.....	67
3.3.2 Animate(动画)组件	68
3.3.3 MediaPlayer(多媒体播放)组件	69
3.3.4 应用举例.....	70
习题与思考题	71
第4章 数据库程序设计	
4.1 Delphi 数据系统概述	72
4.2 创建数据库.....	73
4.2.1 定义数据库别名.....	73
4.2.2 利用 DataBase Desktop (数据库桌面)建立数据表结构	74
4.2.3 编辑数据表.....	76
4.3 利用数据库组件实现数据库编程.....	76
4.3.1 数据库组件介绍.....	76
4.3.2 数据库维护程序.....	80
4.3.3 数据库查询程序.....	82
4.3.4 Query 组件的 SQL 查询方法	85
4.4 图书馆管理系统的实现.....	87
4.4.1 系统主要功能.....	88
4.4.2 数据库结构.....	88
4.4.3 子功能的实现.....	88
4.4.4 窗体之间的联系方法.....	94
4.5 数据库报表设计.....	94
4.5.1 数据库报表的常用组件.....	95
4.5.2 报表应用实例.....	98
习题与思考题.....	101
第5章 Internet 程序设计	
5.1 实例一:Web 浏览器	103
5.1.1 WebBrowser 组件	103
5.1.2 Web 浏览器的实现	104
5.2 实例二:网上聊天器.....	107
5.2.1 ClientSocket 组件	107
5.2.2 ClientWinSocket 组件.....	108
5.2.3 ServerSocket 组件	108
5.2.4 ServerWinSocket 组件	109
5.2.5 网上聊天器的实现	110



习题与思考题.....	114
第 6 章 定制组件	
6.1 创建组件	115
6.1.1 创建组件单元文件	115
6.1.2 添加特性	116
6.1.3 添加事件和方法	117
6.1.4 定义构造函数和析构函数	118
6.1.5 ClockLabel 单元的完整代码	118
6.2 安装组件	120
习题与思考题.....	122
第 7 章 Delphi 软件集成技术	
7.1 帮助文件的制作	123
7.1.1 创建 RTF 文件	123
7.1.2 生成 HLP 文件	124
7.1.3 嵌入应用程序	127
7.2 软件安装盘的制作	127
7.3 学习 Delphi 的一些技巧	131
7.3.1 使用联机帮助	131
7.3.2 查看 Delphi 提供的例程	131
7.3.3 获取 Delphi 的源代码	131
习题与思考题.....	132

下篇 面向对象程序设计基础

第 8 章 PASCAL 语言概述

8.1 PASCAL 源程序结构	134
8.1.1 程序首部	134
8.1.2 程序说明部分	135
8.1.3 语句部分	135
8.2 字符集和符号	135
8.2.1 基本字符	135
8.2.2 符 号	136
8.2.3 分隔符	137
8.3 数据类型的概念	137
8.4 常量与变量	138
8.4.1 常 量	138
8.4.2 常量定义	138
8.4.3 变量说明	138
8.5 简单数据类型	139
8.5.1 整数类型	139



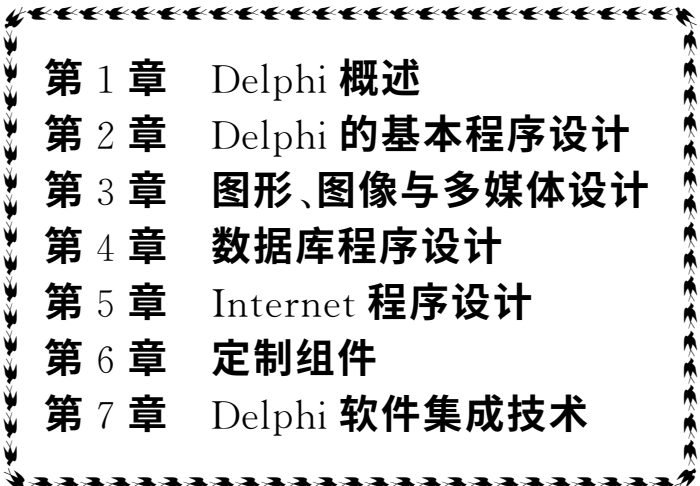
8.5.2	实数类型	140
8.5.3	字符类型	141
8.5.4	布尔类型	142
8.5.5	枚举类型的定义、运算规则和输入、输出方法	142
8.5.6	子域类型的定义和运算规则	145
8.5.7	类型间的关系	147
8.6	表达式与赋值语句	148
8.6.1	表达式	148
8.6.2	赋值语句	150
8.7	输入和输出过程	150
8.7.1	读语句	151
8.7.2	写语句	153
8.7.3	字符类型变量的输出	154
8.7.4	整数类型变量的输出	155
8.7.5	实数类型变量的输出	156
8.7.6	布尔类型变量的输出	158
	习题与思考题	159
第9章 控制语句		
9.1	复合语句	161
9.2	条件语句	162
9.2.1	单分支条件语句	162
9.2.2	双分支条件语句	163
9.2.3	条件语句嵌套	164
9.3	情况语句	165
9.4	当语句	167
9.5	直到语句	168
9.6	循环语句	169
9.7	循环嵌套	171
9.8	goto 语句	172
	习题与思考题	173
第10章 过程与函数		
10.1	过程	174
10.1.1	无参过程	174
10.1.2	带参过程	175
10.1.3	值参数和变量参数	177
10.2	函数	178
10.3	过程嵌套与标识符作用域	180
10.3.1	过程和函数嵌套的概念	180
10.3.2	标识符的作用域	181
10.4	非局部量与副作用	181



习题与思考题	183
第 11 章 构造类型	
11.1 数组类型	185
11.1.1 一维数组	185
11.1.2 多维数组	192
11.1.3 字符数组和字符串类型	195
11.2 集合类型	202
11.3 记录类型	209
11.3.1 记录类型的定义	209
11.3.2 记录的嵌套	211
11.3.3 记录类型变量及其访问	211
11.3.4 开域语句	214
11.4 文件类型	218
11.4.1 磁盘文件的逻辑组织	219
11.4.2 随机文件	221
11.4.3 text 文件	226
11.4.4 无类型文件	229
11.4.5 标准文件	231
11.4.6 I/O 检查	233
11.5 指针类型与动态变量	234
11.5.1 指针类型	234
11.5.2 new 标准过程与指针变量	236
11.5.3 动态变量的访问	236
11.5.4 dispose 标准过程	238
习题与思考题	238
第 12 章 面向对象程序设计基础	
12.1 对象类型的定义	239
12.2 对象类型的封装性	242
12.3 对象类型的继承性	244
12.4 静态方法	247
12.5 虚方法和多态性	250
12.6 对象类型的可扩展性	254
12.7 动态对象类型	258
12.7.1 方法调用与一般过程和函数调用的不同	258
12.7.2 构造方法和析构方法同一般方法调用的不同	259
12.7.3 Pascal 的动态内存管理	259
习题与思考题	264
参考文献	

上 篇

Delphi 程序设计

- 
- | | |
|-------|----------------|
| 第 1 章 | Delphi 概述 |
| 第 2 章 | Delphi 的基本程序设计 |
| 第 3 章 | 图形、图像与多媒体设计 |
| 第 4 章 | 数据库程序设计 |
| 第 5 章 | Internet 程序设计 |
| 第 6 章 | 定制组件 |
| 第 7 章 | Delphi 软件集成技术 |



第 1 章

Delphi 概述

Delphi 是一种可视化的面向对象的编程工具。它提供了大量的可视化组件(类),用户可以方便地借助这些组件来组织程序结构。因为 Delphi 具有程序结构框架代码自动生成功能,用户只须适当地在框架中添加部分程序代码,即可设计出界面美观、实用可靠的 Windows 应用程序。

本章首先介绍 Delphi 的发展概况,然后以一个简单的应用程序为例,介绍 Delphi 的集成开发环境 IDE、Delphi 应用程序的开发步骤以及 Delphi 程序的基本组成。

DP 1.1 Delphi 的发展概况

Delphi 一词原为古希腊的一个都市名称,因拥有 Apollo(太阳神)的神殿而著称。1994 年,Borland 公司(现更名为 Inprise 公司)研制出了一个基于 Windows 的面向对象的可视化软件开发工具,为了使其成为可视化开发工具的先驱与核心,将其命名为 Delphi。

Delphi 实际上是 Pascal 语言的一个版本,但它与传统的 Pascal 语言有天壤之别。其一,Delphi 是可视化的开发工具,借助 Delphi 开发者可以很容易开发出图形用户界面的应用程序;其二,Delphi 是真正的面向对象的开发工具,提供了应用程序的一些基本框架,开发者只须在框架中添入自己的程序代码(Object Pascal 代码)即可,不必自己建立对象。因此说,Delphi 是一个将可视化编程与面向对象的开发框架有机地结合为一体的软件开发工具。

Delphi 从 1994 年问世以来,已经有六个版本:

最初的 Delphi 1.0 运行于 Windows 3. X 环境下,因其提供了全新的可视化环境而引起了人们的关注。

随着 Windows 95 及 Windows NT 的出现,Delphi 2.0 和 Delphi 3.0 版本相继推出。Delphi 2.0 版本增加了数据模块等数据重用功能,改善了用户界面及程序调试功能;Delphi 3.0 版本增加了多层 C/S 数据库开发、Internet/Intranet 程序设计等功能。

之后,又推出了 Delphi 4.0 和 Delphi 5.0 版本。Delphi 4.0 扩展了 Pascal 语言,并增加了 Code Explorer、项目管理、COM 和 CORBA 组件的开发等功能;Delphi 5.0 版本增强了数据库功能(特别是对 ADO 的支持)、程序调试功能、项目管理功能,增加了网络浏览器组件,丰富了帮助系统。

Delphi 发展到现在的 Delphi 6.0 版本,已经是一个比较完善、先进的软件开发工具。主要体现在,Delphi 具有目前世界上最快的 32 位源代码编译器,产生的可执行文件可以直接运行;Delphi 可以通过 ADO 或 BDE 快速方便地访问各类数据库,具有强大的数据库处理功能;



Delphi 提供了图形、图像以及多媒体处理功能;具有强大而快捷的 Internet/Intranet 应用程序开发功能;Delphi 支持开发 XML 组件组,简化了数据分布,优化了数据交换,支持用户定制自己的组件;此外,Delphi 还支持虚拟内存、多线程等技术,可以充分发挥操作系统的强大功能。

在本书中,以 Delphi 6.0 版本为基础,介绍利用 Delphi 开发各种应用程序的方法。

DP 1.2 Delphi 的集成开发环境 IDE 简介

从 Windows 98/2000 的“开始”菜单中选择“程序 | Borland Delphi 6 | Delphi 6”,启动 Delphi 6.0 应用程序后,会出现如图 1.1 所示的集成开发环境。Delphi 6.0 开发环境由 6 个窗口组成:主窗口、对象浏览器、对象观察器、代码浏览器、代码编辑器、窗体窗口。



图 1.1 Delphi 集成开发环境 IDE

1.2.1 主窗口

Delphi 的主窗口位于屏幕的顶部,包括标题栏、菜单栏、加速栏和组件板四部分。标题栏显示的是当前项目的名称(缺省为 Project1)。

1. 菜单栏

包含 10 个菜单,用于管理应用程序的设计,管理 Delphi 窗口界面,配置 Delphi 环境,获得在线帮助等。



2. 加速栏

加速栏由 16 个加速按钮组成,提供最常用的 Delphi 操作的快捷方式(这些功能在菜单栏中均可找到)。各按钮功能如图 1.2 所示。



图 1.2 加速栏

3. 组件板

组件板为用户提供进行可视化界面及程序设计的各类组件。其上共有 19 个标签,每个标签中包括功能相近的一组组件,默认 Standard 标签位于最前面,如图 1.3 所示。要选取其他标签,只须用鼠标单击即可,其中未显示的标签页可以通过单击组件板右上角的黑三角按钮来显示。Standard 标签中放置的是在设计项目时最常用到的组件,例如:Button(按钮)、Edit(编辑框)、Label(标签)、MainMenu(主菜单)、PopupMenu(弹出菜单)等。



图 1.3 组件板

1.2.2 对象观察器(object inspector)

该窗口位于屏幕的左下方,包含对象选择组合框、特性(properties)标签和事件(events)标签三部分,如图 1.4 所示。

1. 对象选择组合框

对象选择组合框是一个下拉列表框,显示当前选择的对象。可以通过单击下拉按钮列出当前窗体中的所有对象。

2. 特性(properties)标签

显示当前对象的特性信息,并允许改变对象的特性。左侧是特性名称,右侧是特性值。常见的特性有 Caption(标题)、Name(名称)、Width(宽度)、Height(高度)等。有的特性带“+”



图 1.4 对象观察器

号,单击“+”后,会显示与该特性相关的一系列特性,这时“+”变为“-”;单击“-”,相关的特性又隐藏起来。有的特性值中带有“...”,单击省略号会打开一个对话框。

3. 事件(events)标签

列出当前对象可以响应的所有事件,在列表框的左侧是事件名称,右侧是事件值。当双击事件值时,将在代码编辑器中产生相应的事件处理程序。

1.2.3 窗体窗口

该窗口位于屏幕的中部偏右,窗口中充满了小点。用户可以将各种组件按设计目标放入窗体构造程序界面,窗体使程序开发时的界面即为程序运行时的界面。

1.2.4 代码编辑器和代码浏览器

位于窗体的下面,当被其他窗口覆盖时,可以按快捷键 F12 切换,如图 1.5 所示。

1. 代码编辑器

代码编辑器是一个功能强大的文本编辑器。它能够自动生成大多数 Windows 应用程序必需的缺省定义和程序结构框架,用户只须在此窗口中添加部分程序代码即可完成应用程序的设计。窗口底部的状态栏显示的是当前文件中光标的位置、修改/保存状态、插入模式等信息。代码编辑器为每个打开的文件设有一个标签,文件名显示在标签上。当开始一个新项目时,代码编辑器只有一个标签(unit1),在项目中增加一个单元(unit)或窗体(form)时,代码编辑器就会增加一个标签。当打开一个文本文件,即使该文件不属于当前项目,或扩展名不是 .Pas,代码编辑器也会增加一个标签。当一个标签被选中,对应的文件内容显示在代码编辑器

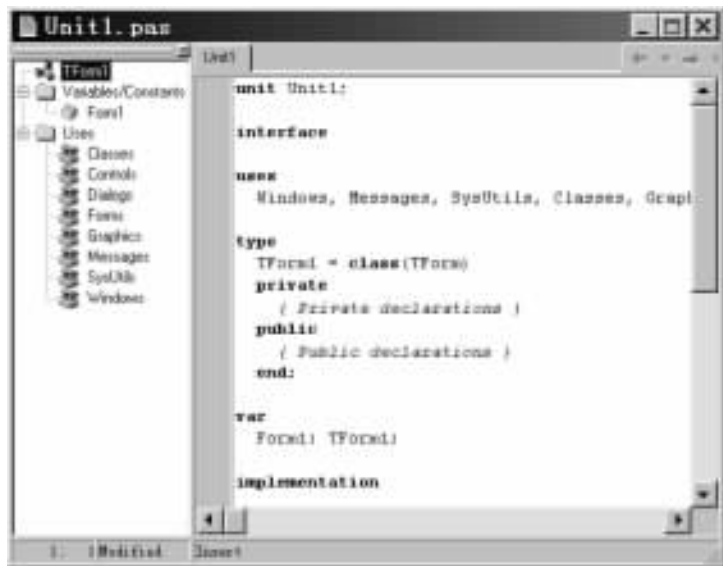


图 1.5 代码编辑器和代码浏览器

中,对应的文件名显示在窗口标题条上。

2. 代码浏览器

该窗口中显示的是一个树形结构图。用户可以通过代码浏览器快速地在程序代码单元中定位,也可以向单元代码中添加变量、类的数据成员、特性和方法等。

1.2.5 对象浏览器(object tree view)

该窗口位于加速栏和对象观察器中间,用来显示当前应用程序中使用的所有可视化组件对象。各个对象以树形结构显示,清楚地表示了各个组件对象之间的关系。在此窗口中,用户若单击某一组件对象名,可以在对象观察器中看到该组件的特性及事件设置;用户若双击某一组件对象名,可以在代码编辑器中看到该组件的事件处理代码。



1.3 Delphi 程序设计的基本步骤

在 Delphi 应用程序的设计中,主要完成以下两方面工作:

- (1) 设计用户界面,即在空白窗体中放置各种组件,并设置各组件的特性;
- (2) 编写事件处理代码,即在代码编辑器中已自动生成的程序框架中,添加部分代码。

使用 Delphi 设计应用程序一般包括以下几个步骤:设计用户界面、编写事件处理过程、编译运行应用程序。

下面以图 1.6 所示的应用程序为例,说明程序设计的基本步骤。

该程序的功能如下:当该程序运行时,在窗口中设有三个按钮,即当用户单击“显示”按钮



图 1.6 “Delphi 应用程序举例”的运行界面

时,在窗口中央显示“齐齐哈尔大学计算中心欢迎您!”;单击“清除”按钮时,窗口中央的内容消失;单击“关闭”按钮时,窗口关闭,退出程序。

1.3.1 设计用户界面

1. 开始一个新项目

创建一个应用程序必定对应一个项目文件,因此首先要建立一个新项目。启动 Delphi 后,会自动建立一个新项目,其默认的项目名称为 Project1;还可以使用 File 菜单的 New Application 菜单项重新开始一个新项目。

2. 创建窗体

窗体是设计应用程序的基本结构单元,在 Delphi 启动后会自动生成一个空白窗体;用户也可以用 File 菜单的 New Form 菜单项添加一个新窗体。

下面设置窗体的特性,在对象观察器窗口的 Properties 标签中设置窗体的各种特性。

本程序的窗体特性设置如下:设置窗体的标题,在 Properties 标签中找到 Caption(标题)特性,将其内容改为“程序举例”;设置窗体的大小,将 Width(宽度)特性设为 450,将 Height(高度)特性设为 250。

3. 放置组件

单击组件板中相应标签中的组件,然后在窗体适当位置单击,即可将组件放置在窗体中。

例如:首先单击组件板上 Standard 标签的 Button(按钮)组件,然后在窗体的适当位置单击,即可将 Button1 按钮放入窗体。若觉得位置不满意,可以在按钮上单击,选中该按钮(周围将出现小黑点),然后将其拖到适当的位置。用户可以通过拖动按钮四周的小黑点调整其大小。

用同样的方法,再将两个 Button 组件和一个 Label(标签)组件放在窗体中,如图 1.7 所示。



4. 设置对象特性

单击窗体中的组件,再到对象观察器窗口中的 Properties 标签中设置组件的各种特性。

例如:单击组件 Button1,在对象观察器窗口中的 Properties 标签中找到 Caption 特性,将其改为“显示”,找到 Font 特性将字体设为 18 磅隶书。

用同样方法,将另外两个按钮的 Caption 特性分别设置为“清除”和“关闭”,字体均为 18 磅隶书;将 Label1 组件的 Caption 特性设置为空,字体为 16 磅宋体。结果如图 1.8 所示。



图 1.7 添加组件后的窗体界面



图 1.8 设置组件特性后的窗体界面

1.3.2 编写事件处理过程

窗体上放置的各组件,必须经过添加事件处理过程,才能接受用户的各种操作。Delphi 大部分组件都有缺省的事件处理过程(Delphi 自动生成的程序框架);但是在事件处理过程的 begin...end 之间是空的,等待用户来添加具体的程序代码。具体方法如下:

1. 定位事件处理过程

单击窗体中的组件,再到对象观察器窗口 Events 标签中双击对应的事件,即可激活代码编辑器,光标定位在该事件处理过程的 begin...end 之间。如果某事件是组件的缺省处理过程,可以直接在窗体中双击该组件,定位该事件处理过程。

例如,单击“显示”按钮,在对象观察器窗口中的 Events 标签中双击 OnClick 事件的取值栏,这时光标定位在对应过程的 begin...end 之间。(按钮的 OnClick 事件,是指当用户单击该按钮时,OnClick 事件被触发,Delphi 调用该事件的处理过程。)

2. 添加程序代码

在光标出现的位置输入事件的实现代码。

例如,“显示”按钮的 Onclick 事件处理代码为

```
Label1.Caption:='齐齐哈尔大学计算中心欢迎您!';
```

用同样的方法,添加“清除”按钮的 Onclick 事件处理代码为

```
Label1.Caption:='';
```



添加“关闭”按钮的 Onclick 事件处理代码为

```
Close;
```

1.3.3 保存文件

在 File 菜单中选择 Save All 命令, Delphi 将要求用户输入与窗体有关的单元文件名及项目文件名。首先保存的是单元文件, 输入“ExampleUnit”, 单击“保存”; 接下来保存项目文件, 输入“ExamplePro”, 单击“保存”。注意, 窗体名与单元文件名及项目文件名不能相同。为防止程序在调试过程中发生意外, 在编程过程中应该注意随时存盘。

1.3.4 编译运行应用程序

用户可以使用 Project 菜单中的 Compile 菜单项(或 Ctrl+F9)编译应用程序。编译的作用是检查程序的语法错误, 如无语法错误, 则生成可执行的 EXE 文件。

用户可以使用 Run 菜单中的 Run 菜单项(或 F9)运行应用程序。若应用程序已编译好, 则执行该命令直接运行程序; 否则, 先编译再运行。例如, 直接按 F9 运行该程序, 程序运行结果如图 1.6 所示。

1.3.5 Delphi 程序设计的步骤

Delphi 程序设计的基本步骤可以用图 1.9 所示的流程图来表示。

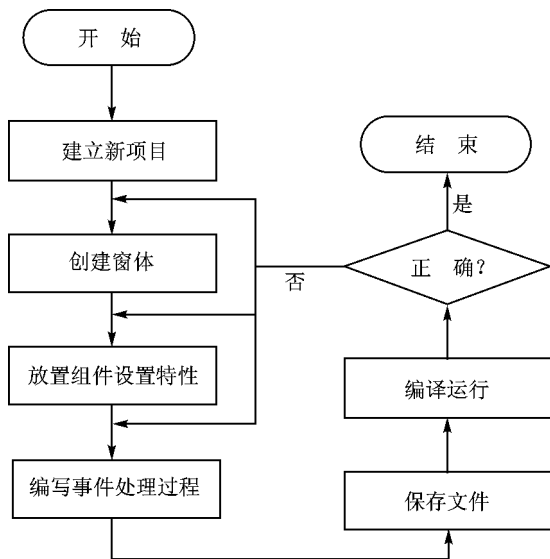


图 1.9 Delphi 程序设计基本步骤流程图



DP 1.4 Delphi 程序的基本组成

在运行完上一节的“程序举例”程序后,如果查看保存 Delphi 文件的文件夹,会发现有如下几个文件:ExamplePro. EXE(可执行文件)、ExamplePro. DPR(项目文件)、ExamplePro. DOF(项目选择文件)、ExamplePro. RES(资源文件)、ExampleUnit. PAS(单元文件)、ExampleUnit. DFM(窗体文件)和 ExampleUnit. DCU(编译单元文件)。其中与 Delphi 程序设计紧密相关的文件主要有三类:项目文件(. DPR)、单元文件(. PAS)和窗体文件(. DFM)。

1.4.1 项目文件

项目文件是 Delphi 程序的主程序,扩展名为“. DPR”,描述了整个程序的结构及启动代码。项目文件一般由 Delphi 自动生成,用户不必改动。

用户可以单击 View 菜单的 Units 菜单项(或 Ctrl+F12),打开对话框,选择项目文件(例如 ExamplePro),单击 OK,该项目文件即显示在代码编辑器中。下面是“程序举例”的项目文件。

```
program ExamplePro;
uses
  Forms,
  ExampleUnit in 'ExampleUnit.pas' {Form1};
{$R *.RES}
begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

(1) program 为程序标志,后面是程序名。

(2) uses 列出本程序使用的单元;Forms 单元是 Delphi 提供的单元,其中定义了 Application 变量;ExampleUnit 单元是用户自定义的,声名了“Delphi 应用程序举例”窗体 Form1 对应的类 TForm 。

(3) {\$R *.RES} 是一个编译开关,表示要连接的资源文件。

(4) begin...end 之间是初始化代码,通过调用 Application 变量的三个方法实现。其中 Initialize 方法完成程序的初始化工作,CreateForm 方法负责建立 Form1 窗体,Run 方法使程序进入运行阶段。

从以上结构可以看出,应用程序的具体功能是靠引用单元来实现的。

1.4.2 单元文件

单元文件存储 Delphi 中的单元代码,扩展名为. PAS。通常创建一个窗体也就创建了一



个单元;用户也可以通过单击 File|New 菜单项,打开 New Item 对话框,再双击 Unit 创建一个独立的单元。用户可以在 View Units 对话框选择单元文件名(例如 ExampleUnit),单击 OK,查看单元文件;也可以在设计窗体出现时,按 F12 键切换。以下为 ExampleUnit 单元的内容:

```
unit ExampleUnit;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    Label1: TLabel;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
var
  Form1: TForm1;
implementation
{ $R *.DFM }
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption:= '齐齐哈尔大学计算中心欢迎您!';
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  Label1.Caption:= '';
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  Close;
end;
```



end.

(1) unit 为单元的标志,后面是单元名。

(2) interface 和 implementation 之间是接口部分,其间定义的类型、变量、过程等,可以被其他单元或程序的其他部分访问。uses 后列出的是该单元用到的其他单元。type 部分用来定义数据类型。数据类型可以是对象类型,例如,ExampleUnit.pas 中定义了 TForm1 类。TForm1 继承了 TForm 类的成员(TForm 是 Delphi 为创建窗体提供的一个类),同时又说明了 7 个成员:3 个 TButton 对象、1 个 TLabel 对象、3 个类的方法(过程)。接下来是类的附加说明部分。private(私有说明)中的成员只能被同类成员使用,public(公有说明)中的成员可以被其他单元中的语句使用。该程序中没有私有和公有说明,其中{ }包围的是注释部分,在运行时并不起作用;var 部分用来说明变量,例如 ExampleUnit.pas 中定义了一个 TForm1 类的变量(或者说类的实例)Form1。

(3) implementation 后面的部分是单元的实现部分,定义所有在 Interface 部分说明的过程和函数。{\$R *.DFM} 不是注释部分(花括号内以 \$ 开头),它的作用是把单元连到窗体的.dfm 中。

1.4.3 窗体文件

窗体文件存储的是窗体的特性,扩展名为.dfm。用户在窗体上单击鼠标右键弹出快捷菜单,选择其中的 View as Text 菜单项,窗体文件即显示在代码编辑器中。具体内容如下:

```
object Form1: TForm1
  Left = 192
  Top = 122
  Width = 450
  Height = 300
  Caption = 'Delphi 程序举例'
  Color = clBtnFace
  Font.Charset = DEFAULT_CHARSET
  Font.Color = clWindowText
  Font.Height = -11
  Font.Name = 'MS Sans Serif'
  Font.Style = []
  OldCreateOrder = False
  PixelsPerInch = 96
  TextHeight = 13
  object Label1: TLabel
    Left = 96
    Top = 80
    Width = 8
    Height = 16
    Caption = ''
    Font.Charset = GB2312_CHARSET
    Font.Color = clWindowText
```



```
Font.Height = -16
Font.Name = '宋体'
Font.Style = []
ParentFont = False
end
object Button1: TButton
  Left = 64
  Top = 168
  Width = 65
  Height = 25
  Caption = '显示'
  Font.Charset = GB2312_CHARSET
  Font.Color = clWindowText
  Font.Height = -21
  Font.Name = '隶书'
  Font.Style = []
  ParentFont = False
  TabOrder = 0
  OnClick = Button1Click
end
object Button2: TButton
  Left = 188
  Top = 168
  Width = 65
  Height = 25
  Caption = '清除'
  Font.Charset = GB2312_CHARSET
  Font.Color = clWindowText
  Font.Height = -21
  Font.Name = '隶书'
  Font.Style = []
  ParentFont = False
  TabOrder = 1
  OnClick = Button2Click
end
object Button3: TButton
  Left = 312
  Top = 168
  Width = 65
  Height = 25
  Caption = '关闭'
  Font.Charset = GB2312_CHARSET
  Font.Color = clWindowText
```



```
Font.Height = -21  
Font.Name = '隶书'  
Font.Style = []  
ParentFont = False  
TabOrder = 2  
OnClick = Button3Click  
end  
end
```

从窗体文件中,可以看出各个组件的层次关系,以及它们的一系列特性。

DP 1.5 Delphi 的类结构

在“程序举例”中,用到了组件板中的一些组件。实际上,Delphi 中的各个组件都是使用对象类型定义的对象。在 Delphi 中,所有的类都是从 TObject 类继承下来的。TObject 类的声明在 System 单元中,它定义了操作类的最基本的方法。图 1.10 是 Delphi 中类的继承关系示意图。

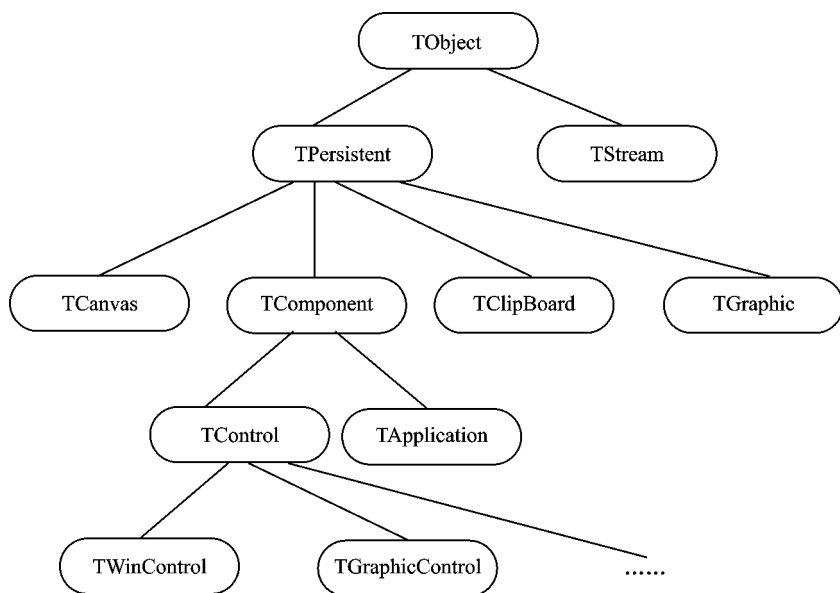


图 1.10 Delphi 中的类结构

从图 1.10 中可以看出,TObject 类位于继承关系的最顶层,是所有类的祖先类。接下来一层是 TPersistent 类和 TStream 类。TStream 类是能够实现与各种存储介质进行读/写操作的流对象的基类,TPersistent 类主要为它的继承者提供对流的读/写能力。TPersistent 类的继承者有 TCanvas 类、TClipboard 类、TGraphic 类和 TComponent 类等。其中,TCanvas 类一般用于实现画图功能,TClipboard 类用于对 Windows 的剪切板进行处理,TGraphic 类用于处理图形文件,TComponent 类是组件板中所有组件的基类。但直接继承下来的却只有几



个非可视组件(如 TTime、TDataSource 等),绝大多数组件是从 TComponent 类的下级 TControl 类继承下来的。TControl 类定义了组件板中所有可视化组件(如 TButton、TLabel 等)的基本特性、方法和事件等。TControl 类的子类又有许多,例如, TWinControl 类和 TGraphicControl 类等。TWinControl 类用于窗口控制(如按钮、对话框、组合框的控制)。

DP 1.6 Delphi 中的基本概念

1.6.1 组 件

组件是构成程序及用户界面的基本组成元素,如上一章例程中用到的按钮、标签以及窗体,甚至程序本身。在 Delphi 中,所有的组件都被封装成类(组件类)。用户在程序中使用的组件是其对应的组件类的实例。例如,“程序举例”中的按钮组件 Button1、Button2、Button3 均是 TButton 类的实例,Label1 组件是 TLabel 类的实例。

1.6.2 特 性

特性是描述组件特征的数据。任何一个组件都通过称为特性的一组有一定含义的数据加以限定。大部分组件中常用的公共特性有:

- Caption(标题)——在组件的标题栏中或图标表面显示的文本。用户可以在程序中动态地修改该特性,有些组件无此特性。
- Name(名称)——组件在程序内部的名称,与组件是一一对应的。用户可以在程序设计之初自定义,但不可在程序中动态地修改该特性。
- Font(字体)——用于设置组件上文本的字体、大小、颜色等。
- Visible——决定组件是否可见。其值为 True,则组件可见;为 False,则不可见。
- Enable——决定组件是否响应用户触发的事件。其值为 True,则能响应事件;否则不能。
- Width(宽度)、Height(高度)——决定组件的宽度和高度。
- Left(左端)、Top(顶端)——决定组件与其容器左端及顶端之间的距离。

设置组件特性值的方法有两种:

- (1) 在对象观察器的 Properties 标签页中直接设置;
- (2) 通过程序代码在程序中动态的设置。

通过程序代码设置组件特性的格式如下:

组件名.特性:=特性值;

例如,设置按钮组件 Button1 表面的显示文本内容为“确定”,高度为 50,宽度为 100:

```
Button1.Caption:='确定';  
Button1.Height:=50;  
Button1.Width:=100;
```



1.6.3 方 法

方法是可以在组件上操作的过程。组件通过调用方法完成规定的操作。常用的方法有：

- Show——显示组件。
- Hide——隐藏组件。
- Close——关闭组件。

方法的调用是在程序中实现的,调用的一般形式为

组件名.方法名;

例如,要隐藏窗体 Form1,可以用

```
Form1.Hide;
```

来实现。

1.6.4 事 件

事件是能被组件识别的固定动作。事件处理是指事件发生时,程序对该事件作出的反应。例如:按钮组件的单击 Click 事件。当单击该按钮组件时则触发该事件(至于按钮组件如何捕捉到此事件的发生,用户不必关心),用户只须要编写一些程序(事件处理程序)来实现该事件发生后将完成的具体任务。

在 Delphi 程序开发中,程序设计的重点是编写事件处理程序。事件处理是由过程或函数来实现的,例如,若希望当按下按钮组件 Button3 时,将关闭当前窗体,则可在 Button1 的单击事件中设置如下代码:

```
procedure Form1.Button3Click(Sender:TObject);  
begin  
    Close;  
end;
```

其中该过程的程序框架由 Delphi 自动生成,用户只要在其中填上 Close 语句即可。这样用户可以把精力集中在实现应用程序的实质性问题上了。

DP 习题与思考题

1.1 填 空:

(1) 在 Delphi 的 IDE 窗口中,特性(properties)标签的某个特性名前面带有“+”表示(),某个特性名前面带有“-”表示()。

(2) 快速切换到代码编辑器和代码浏览器的功能键是()。

1.2 问 答:

(1) 试述 Delphi 集成环境的组成以及各部分的功能。

(2) 什么是组件的特性、方法和事件?

1.3 调试 1.3 节的例程,熟悉利用 Delphi 进行程序设计的基本步骤,并回答下列问题:



- (1) 改变窗体上各组件的高度、宽度、字体等特性,观察程序运行效果。
- (2) 如何在窗体中添加组件?
- (3) 如何为一个组件添加代码?
- (4) 如何保存 Delphi 的项目?项目文件的扩展名是什么?
- (5) 查看项目文件、单元文件以及窗体文件的内容。



第 2 章

Delphi 的基本程序设计

从上一章的程序举例可以看出,借助 Delphi 提供的一些组件可以方便地设计应用程序,这也正是 Delphi 的功能强大之所在。在这一章中,结合具体的实例介绍一下 Delphi 中常用组件的基本概念及其使用方法。当然,Delphi 提供的组件比较多,我们不可能一一列举;但是各组件的使用方法具有共性,希望读者仔细体会本章内容,以达到触类旁通的目的。

DP 2.1 窗体(form)设计

窗体,也就是应用程序运行时的窗口,是可视化程序设计的基础。程序设计者可以把各种组件放置其上,然后通过配置组件的特性,编写事件处理代码,实现程序设计目的。

窗体对象属于 TForm 类,是 Delphi 应用程序设计中最常用的组件。但它并不在组件板中,可以在创建一个新项目时自动生成;也可以通过 File|New Form 菜单生成。一个项目中可以有多个窗体,但主窗体只有一个。

下面简要介绍窗体的常用特性和事件。

2.1.1 常用特性

1. 窗体的布局特性

- Left 和 Top 特性

确定窗体左上角相对于屏幕的位置。Left 为横坐标,Top 为纵坐标。

- Height(高度)和 Width(宽度)特性

确定窗体的大小。

- ClientHeight 和 ClientWidth 特性

确定窗体客户区(不包括边框、标题和菜单栏)大小。

- Align 特性

确定窗体与屏幕的对齐方式,取值有:alTop(上对齐)、alBottom(下对齐)、alLeft(左对齐)、alRight(右对齐)、alClient(充满客户区),默认为 alNone(保持原始位置)。

- AutoSize 特性

确定窗体的大小是否根据窗体上的组件的位置变化,默认值为 False。

用户可以拖动窗体改变其大小和位置,也可以在对象观察器(object inspector)中选择相应的各个特性值,还可以用程序代码实现。例如,设置 Form1 的各个特性如下:

```
with Form1 do
```



```
begin
    Left:=200;
    Top:=80;
    Height:=100;
    Width:=200;
end;
```

则,Form1 的左上角在(200,80)位置处,窗体高 100,宽 200。

2. 窗体的外观特性

- Color 特性

设置窗体客户区(窗体中除标题栏之外的部分)的颜色。

- Font 特性

设置窗体中组件上的字体。

- Icon 特性

设置标题左边的小图标,也是窗体最小化时的图标。

- Visible 特性

确定窗体是否可见。其值为 False 不可见,为 True 可见。

- BorderIcons 特性

确定窗体标题栏上的图标。其子特性有:biSystemMenu(系统菜单)、biMinimize(最小化按钮)、biMaximize(最大化按钮)、biHelp(帮助按钮)。各子特性值为 True 表示具有相应图标,否则为 False。

- BorderStyle 特性

控制窗体的边框类型。取值有:bsDialog(标准对话框边框,不能改变大小,无最大化、最小化按钮和系统菜单)、bsSingle(单线边框,不能改变大小,无最大化和最小化按钮)、bsNone(无边框,不能改变大小,无最大化、最小化按钮,无系统菜单和标题)、bsSizeable(标准可变大小时边框)、bsToolWindow(工具窗口边框,不可改变大小)、bsSizeToolWin(工具窗口边框,可改变大小)。

- HorizScrollBar(水平滚动条)特性和 VertScrollBar(垂直滚动条)特性

其子特性有:Visible(滚动条是否可见)、Position(滚动条上滑块的起始位置)、Range(滚动条的最大滑动距离)。若想使滚动条可见,对于水平滚动条应保证 Range 值大于 ClientWidth 值;对于垂直滚动条应保证 Range 值大于 ClientHeight 值。

3. 窗体的状态和控制特性

- Enabled 特性

控制窗体是否有效,只有取值为 True 时,才能对窗体进行操作。

- FormStyle 特性

设置窗体的类型,取值有:fsNormal(一般类型)、fsMDIForm(MDI 父窗体)、fsMDIChild(MDI 子窗体)、fsStayOnTop(窗体始终保持在顶层)。

- ActiveControl 特性



控制窗体上哪个组件获得聚焦。

例如,要使 Form1 窗体中的 Button 组件获得聚焦,其程序代码如下:

```
Form1.ActiveControl:=Button1;
```

- WindowState 特性

控制窗体所处的状态,取值有:wsNormal(正常状态)、wsMinimized(最小化状态)、wsMaximized(最大化状态)。

2.1.2 窗体的事件

- OnCreate 和 OnDestroy

OnCreate 事件在创建窗体时触发,用于执行初始化代码;OnDestroy 事件在释放窗体时被调用,放置释放内存资源的代码。

- OnShow 和 OnHide

OnShow 事件在窗体被显示时触发;OnHide 事件在窗体被隐藏时触发。

- OnActivate 和 OnDeActivate

OnActivate 事件在焦点转移到本窗体上时触发;OnDeActivate 事件在焦点转移到其他窗体时触发。

- OnDbClick 和 OnClick

OnDbClick 事件在用户双击窗体空白区时触发;OnClick 事件在用户单击窗体空白区时触发。

- OnClose 和 OnCloseQuery

当一个窗体即将关闭时,首先触发 OnCloseQuery 事件。该事件处理代码中有一个布尔型变量参数 CanClose。若其值为 False,则窗体不被关闭,否则将可能被关闭;若 CanClose 值为 True,则会触发 OnClose 事件。窗体最终能否被关闭取决于该事件的参数 Action。Action 取值的意义如下:caNone(窗体不关闭,无任何操作)、caHide(隐藏窗体)、caFree(释放窗体所占内存)、caMinimize(最小化窗体)。

2.1.3 窗体的方法

- Show:显示窗体。

- Hide:隐藏窗体。

- Close:关闭窗体。

2.1.4 应用举例

【例 2.1】设计一个无标题栏的窗体,上设置三个按钮,功能分别是:最大化、还原、关闭。当单击窗体的空白处时,窗体中央显示“您单击了鼠标左键。”当双击窗体的空白处时,窗体中央显示“您双击了鼠标左键。”

在该程序中需要用到如下组件:一个 Form,用于放置其他组件;三个 Button,分别对应最大化、还原、关闭按钮;一个 Label,用于显示单击或双击的结果提示信息。具体实现步骤如下:



1. 窗体设计

因为要设计无标题栏窗体,所以,窗体的 `BorderStyle` 特性值为 `bsNone`, `ClientHeight` 为 200, `ClientWidth` 为 300。单击 `Font` 特性取值栏右侧的“...”号,在弹出的字体对话框中设置 10 磅规则楷体,用于设定窗体中各组件的字体。

2. 放置组件设置特性

在窗体中央放置一个 `Label` 组件,其 `Caption` 特性为空。然后在窗体下部连续放置三个 `Button` 组件。由于重复放置三个相同的组件,用户可以先按下 `Shift` 键,再选取组件。这样,在窗体上连续单击即可放置多个相同的组件;想取消时,只要单击组件板上的复位图标(斜向上的光标)即可。要想统一调整三个按钮的大小及位置,可以按住 `Shift` 键选定三个 `Button` 组件(组件四周呈灰色小点)。然后,在对象观察器的特性栏中设置其 `Height` 为 25, `Width` 为 50,选定三个 `Button` 组件;单击 `Edit` 菜单或快捷菜单中的 `Align(对齐)` 菜单项,在 `Horizontal(水平)` 框中选择 `Space equally(等间距)`, `Vertical(垂直)` 框中选择 `Centers(居中对齐)`。下面依次将各按钮组件的 `Caption` 设置为“最大化”、“还原”、“关闭”。

3. 编写事件处理代码

“最大化”按钮的 `OnClick` 事件的处理代码为

```
WindowState := wsMaximized;
```

“还原”按钮的 `OnClick` 事件的处理代码为

```
WindowState := wsNormal;
```

“关闭”按钮的 `OnClick` 事件的处理代码为

```
Close;
```

窗体 `Form1` 的 `OnClick` 事件的处理代码为

```
Label1.Caption := '您单击了鼠标左键。';
```

窗体 `Form1` 的 `OnDblClick` 事件的处理代码为

```
Label1.Caption := '您双击了鼠标左键。';
```

程序的运行界面如图 2.1 所示。



图 2.1 窗体实例运行界面



DP 2.2 基本输入/输出组件

2.2.1 标签组件(label)

Label 组件用于在窗体中显示一些说明性的文字及程序的运行结果等。该组件位于 Standard 标签页中,属于 TLabel 类。具体应用见例【2.1】中的 Label1。其主要特性如下:

- AutoSize 特性

决定 Label 组件的大小是否随着显示文本的大小和长短自动调整。标签组件的大小一般可以通过鼠标拖动实现,但 AutoSize 特性有可能改变这种设置。当其值为 True 时,能够自动调整;为 False 时,则不能。默认为 True。

- Alignment 特性

决定文本的水平对齐方式,取值有:taLeftJustify(左对齐,默认值)、taCenter(居中对齐)、taRightJustify(右对齐)。

- Layout 特性

决定文本的垂直对齐方式,取值有:tlTop(顶端对齐,默认值)、tlCenter(中间对齐)、tlBottom(底端对齐)。

- Transparent 特性

控制标签对背景的透明性。当其值为 True 时,由 Color 特性设置的值不起作用,而使用背景的图案颜色;当其值为 False(默认值)时,则用 Color 特性的值作为背景色。

- FocusControl 特性

其值是当按下加速键时,获得聚焦的组件名。当用户按下加速键时,Label 组件将控制权转交给该特性指定的组件。

2.2.2 编辑框组件(edit)

Edit 组件用于接收用户输入的单行文本,也可以用于显示文本信息。该组件位于组件板 Standard 标签页中,属于 TEdit 类。

1. 主要特性

- Text 特性

其值是编辑框中的内容。用户可以通过访问该特性值读取或改变编辑框中的内容。通常,在程序开始时,应将该特性值置空。

- ReadOnly 特性

决定是否可以修改编辑框中的内容。当其值为 True 时,用户不能修改编辑框中的内容;当其值为 False(默认值)时,用户可以改变编辑框中的内容。

- CharCase 特性

用于控制编辑框中文本的大小写情况,取值有:ecLowerCase(转换为小写形式)、ecUpperCase(转换为大写形式)、ecNormal(不改变大小写,默认值)。



- PasswordChar 特性

控制编辑框中字符是否隐藏显示。当其值为非 #0 时,编辑框中显示的字符均由该特性的字符代替,起到保密作用,而 Text 特性值不受影响;当其值为 #0(默认值)时,编辑框中的字符正常显示。

- BorderStyle 特性

决定编辑框的边框类型,其值有:bsNone(无边框)、bsSingle(单线边框,默认值)。

- Enable 特性

其值为 False 时,编辑框将变为灰色,不接收用户的输入。默认值为 True。

- AutoSize 特性

其值为 True(默认值)时,编辑框的高度会随着其中文本高度的改变而自动调整。

- AutoSelect 特性

其值为 True(默认值)时,若编辑框获得聚焦,则编辑框内的文本会被自动选中。

2. 主要事件和方法

- OnChange 事件:当 Text 特性值发生变化时触发该事件。

- Clear 方法:可以清除编辑框中 Text 特性值。

3. 几个相关函数

Edit 组件的 Text 特性值为字符串类型,有时要利用 Edit 组件接收或显示整数或实数信息。同样,Label 组件的 Caption 特性值为字符串类型,有时也要用它来显示数值型数据。遇到这样的情况,可以借助以下函数完成其间的转换:

IntToStr()——将整型数值转换成字符串;

StrToInt()——将字符串转换成整数值;

FloatToStr()——将实型数值转换成字符串;

StrToFloat()——将字符串转换成实数值。

2.2.3 几个显示对话框的过程和函数

Delphi 提供了一些过程函数,利用它们,可以在屏幕上显示一些提示信息或接收输入的对话框。下面介绍较常用的几个。

1. ShowMessage 过程

ShowMessage 过程用于显示一个带有 OK 按钮的消息框,适用于显示各类提示信息。它的过程说明如下:

```
procedure ShowMessage(const Msg: string);
```

其中,Msg 参数指定出现在消息框中的字符串信息。

2. MessageDlg 函数

MessageDlg 函数在屏幕的中间显示一则消息对话框。该对话框中除了可以显示信息外,还可以接受用户的选择。它的函数定义如下:



```
function MessageDlg (const Msg: string; AType: TMsgDlgType; AButtons: TMsgDlgButtons;
HelpCtx: Longint): Word;
```

其中,Msg 参数指定要显示的字符串。

- AType 参数指明该对话框的作用(用不同的符号表示)。它是一个 TMsgDlgType 类型的参数,取值有:

mtWarning(带黄色三角形的警告对话框);

mtError(带红叉的出错对话框);

mtInformation(带蓝色‘i’字样的信息对话框);

mtConfirmation(带绿色问号的确认对话框);

mtCustom(不包含任何图标,对话框的标题是应用程序的名字)。

- AButtons 参数指明对话框中出现的按钮。它是一个 TMsgDlgButtons 类型的参数,取值为以下值: mbYes、mbNo、mbOK、mbCancel、mbAbort、mbRetry、mbIgnore、mbAll、mbHelp 的组合。另外,在 Delphi 中预定义了几个集合常量,用户可以直接使用,名字和含义如下:

mbYesNoCancel(表示对话框中有 Yes、No、Cancel 按钮);

mbOKCancel(表示对话框中有 OK 和 Cancel 按钮);

mbAbortRetryIgnore(表示对话框中有 Abort、Retry、Ignore 按钮)。

- HelpCtx 参数指定当用户单击了帮助按钮时将要显示的帮助主题的 ID,通常为 0。

- MessageDlg 函数返回值表示用户所选择的按钮,返回值如下:

mrNone、mrAbort、mrYes、mrOk、mrRetry、mrNo、mrCancel、mrIgnore、mrAll。

以下是一个简单例子:

```
if MessageDlg('您是否需要继续?', mtConfirmation, [mbYes,mbNo], 0) = mrNo then
  ShowMessage('谢谢您的访问,再见!');
```

首先出现如图 2.2 所示的对话框。若用户选择“No”按钮,则出现如图 2.3 所示的消息框。



图 2.2 消息对话框



图 2.3 消息框

3. MessageDlgPos 函数

MessageDlgPos 函数与 MessageDlg 函数非常类似,只是多了在屏幕上指定对话框出现位置的功能。它的函数定义如下:

```
function MessageDlgPos(const Msg: string; AType: TMsgDlgType; AButtons:
TMsgDlgButtons; HelpCtx: Longint; X, Y: Integer): Word;
```



前面四个参数的含义与 MessageDlg 函数完全相同,后两个参数 X、Y 则用来指定对话框(左上角)在屏幕上的位置。

4. InputBox 函数

InputBox 函数显示一个输入对话框,并能够接收用户在编辑框中输入的一个字符串。它的函数定义如下:

```
function InputBox(const ACaption, APrompt, ADefault: string): string;
```

- ACaption 参数设置该对话框的标题。
- APrompt 参数设置提示用户在编辑框中进行输入的提示文字。
- ADefault 参数设置出现在编辑框中的默认值。

函数的返回值:如果用户选择了 Cancel 按钮,则返回默认的字符串;如果用户选择了 OK 按钮,则返回编辑框中的字符串。

5. MessageBox 函数

MessageBox 函数与 MessageDlg 函数功能基本相同,只是多了全中文显示功能。它是 Application 类的一个方法。其函数定义如下:

```
function MessageBox(const Text, Caption: PChar; Flags: Longint): Integer;
```

- text 参数指定要显示的字符串。
- Caption 参数设置该对话框的标题。
- Flags 参数指定该对话框中出现的按钮类别,取值有:

mb_YesNo, 包括“是”和“否”按钮;

mb_YesNoCancel, 包括“是”、“否”、“取消”按钮;

mb_RetryCancel, 包括“重试”和“取消”按钮;

mb_OKCancel, 包括“确定”和“取消”按钮。

函数返回值表示用户选择的按钮:IDOK(确定)、IDYES(是)、IDNO(否)、IDCANCEL(取消)、IDRETRY(重试)。

例如下面语句:

```
if Application.MessageBox('您是否需要继续?', '确认窗口',  
mb_YesNo) then
```

```
ShowMessage('感谢您的访问,再见!');
```

则图 2.2 会被图 2.4 所示窗口取代。

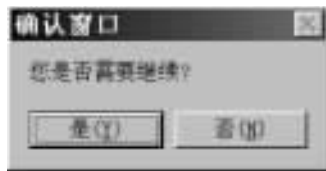


图 2.4 全中文消息对话框

2.2.4 程序举例

25

【例 2.2】实现如图 2.5 所示的用键盘输入的两个数之和的程序。当用户在输入框中输入两个加数后,单击求和按钮,窗口中即可显示和值。

在该程序中用到如下组件:两个 Edit 组件,用来接收键盘输入的数据;三个 Label 组件分别用于显示编辑框的作用和运算结果;一个 Button 组件,用于激发求和操作。主要实现步骤



图 2.5 两数之和程序运行界面

如下:

(1) 窗体设计。将窗体的 Caption 特性设为“两数之和”; Height 和 Width 特性分别为 280 和 400(确定窗体的大小); BorderStyle 特性为 bsDialog(窗体只有关闭按钮,不能改变大小)。

(2) 添加各个组件。按图所示在窗体中放置各个组件,适当调整其大小和位置,并将 Label1 的 Caption 特性设为“加数 1”,Label2 的 Caption 特性设为“加数 2”,Button1 的 Caption

特性设为“求和(&A)”,& 的作用是使其后的英文字母称为该按钮的快捷键,按下 Alt+字母与单击该按钮的作用是相同的。

(3) 编写事件处理代码。双击按钮 Button1,在其 OnClick 事件过程中添加如下代码:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    N1,N2:Integer;
begin
    N1:=StrToInt(Edit1.Text);           //将 Edit1 中接收的字符串转换成整型数赋给 N1
    N2:=StrToInt(Edit2.Text);           //将 Edit2 中接收的字符串转换成整型数赋给 N2
    Label3.Caption:=‘和值是:’+IntToStr(N1+N2); //将 N1 与 N2 的和转换成字符串显示
end;
```

运行程序过程中,编辑框内输入的数据有时会出现非数值字符。这样在转换时会出现错误。可以采用 try...except 结构来处理这种异常,则上述程序代码可修改如下:

```
try
    N1:=StrToInt(Edit1.Text);
    N2:=StrToInt(Edit2.Text);
    Label3.Caption:=‘和值是:’+IntToStr(N1+N2);
except
    ShowMessage(‘输入了非数值字符,无法计算’);
    exit;
end;
```

当出现了非数值字符时,程序直接执行 except 之后的语句,显示出错对话框。若输入正确时,会跳过 except 之后的语句。

为了使程序运行前输入框和结果位置处均无显示,可以在窗体 Form1 的 OnCreate 事件过程中添加如下代码:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
    Edit1.Text:=‘’;
```



```
Edit2.Text:='';  
Label3.Caption:='';  
end;
```

DP 2.3 按钮组件

按钮组件是 Windows 编程中设计用户界面时用得最多的组件。常用的基本按钮组件有 Button、RadioButton、Checkbox, 按钮组有 GroupBox 和 RadioGroup, 它们都位于组件板的 Standard 标签中; 两个位图按钮 BitBtn 和 SpeedButton, 它们位于组件板的 Additional 标签中。Panel 组件和 SpeedButton 组件, 将在下一节介绍。

2.3.1 Button(按钮)组件

1. 常用特性

- Cancel 特性

决定该按钮是否为取消按钮。当 Cancel 值为 True 时, 单击该按钮或按下 ESC 键, 都能触发该按钮的 Onclick 事件。该特性的缺省设置为 False。

- Default 特性

决定该按钮是否为默认按钮。当 Default 值为 True 时, 单击该按钮或按下 Enter 键, 都能触发该按钮的 Onclick 事件。该特性的缺省设置为 False。

2. 常用事件

- Onclick 事件: 当用鼠标单击 Button 按钮或在该按钮获得聚焦时按下 Enter 键均可触发该事件。

2.3.2 BitBtn(位图按钮)组件

该组件的表面上有一个位图, 方便用户识别。它具有 Button 组件的一切特性。其主要特性如下:

- Glyph 特性

指定出现在位图按钮上的位图(位图文件是 .BMP 文件)。

- Kind 特性

指定位图按钮的类型。当 Kind 为 bkCustom 时, 表示其类型由用户自定义, 位图由 Glyph 特性指定。否则, 位图由 Kind 的其他取值决定。

- Layout 特性

指定位图按钮上位图与文本(Caption 中的文字)出现的位置关系: blGlyphLeft(位图在左)、blGlyphRight(位图在右)、blGlyphTop(位图在上)、blGlyphBottom(位图在下)。



2.3.3 RadioButton(单选按钮)组件

该组件的外观是一个小圆圈,旁边有一行说明文字。通常成组使用。当用户选中某一个时,该组件的小圆圈内出现一个小黑点。同一组内只能有一个被选中。其主要特性如下:

- Alignment 特性
指定与该单选按钮相关的文字在小圆圈的左侧(值为 `StaLeftJustify`)还是右侧(值为 `taRightJustify`)。
- Checked 特性
其值为 `True` 时,该按钮处于选中状态。

2.3.4 Checkbox(复选按钮)组件

该组件是一个旁边带有说明文字的小方框。用户可以用鼠标单击小方框改变它的状态(选中/未选中);另外,当该组件变成灰色时,表示处于不确定状态,用户无法对其操作。在一组 `Checkbox` 组件中用户可以一次选择多个复选按钮。其主要特性如下:

- Alignment 特性
同 `RadioButton` 组件。
- AllowGrayed 特性
其值为 `True` 时,该组件可以处于不确定状态。默认为 `False`。
- Checked 特性
其值为 `True` 时,该组件处于选中状态;当该特性为 `False` 时,该组件处于未选中或不确定状态。默认为 `False`。
- State 特性
表示复选按钮的状态(当 `AllowGrayed` 特性为 `True` 时,该特性才有意义)。该特性有三种状态:`cbChecked`(选中)、`cbUnchecked`(未选中)、`cbGrayed`(不确定)。

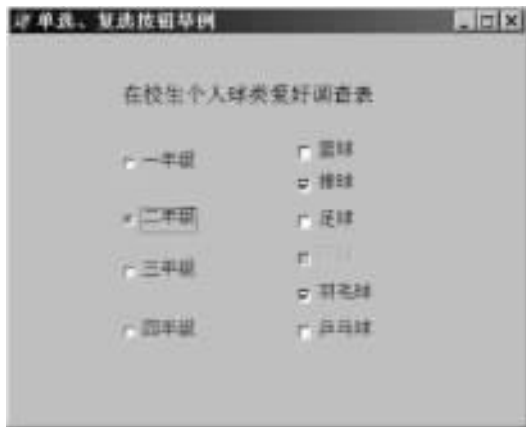


图 2.6 单选、复选按钮举例

【例 2.3】设计一个统计在校生各年级同学球类爱好的用户界面。如图 2.6 所示,每个学生必须有唯一确定的年级,所以年级用单选按钮实现;每个学生可以有多种爱好,因此,爱好用复选按钮实现。具体实现步骤如下:

(1) 在窗体上放置一个 `Label` 组件,将其 `Caption` 特性设为“在校生个人球类爱好调查表”,适当设置其 `Font`(字体)特性。

(2) 在窗体上放置 4 个 `RadioButton` 组件。为方便操作,用户可以在组件板

中选取该组件的同时按下 `Shift` 键。这样在窗体上单击一次鼠标即可放置一个组件。操作结束后,单击组件板上的该组件。用同样方法在窗体上放置 6 个 `CheckBox` 组件。



(3) 按图设置各组件的 Caption 特性。其中“网球”复选按钮的 Enable 特性值为 False,使其在程序运行时不可选。在程序运行过程中,当用户按下某个单选按钮时,其 Checked 特性值为 True,而其他单选按钮的 Checked 特性值均变为 False;当用户选中某个复选按钮时,其 Checked 特性值为 True。用户可以一次选择多个复选按钮。用户可以根据各组件的 Checked 特性编程统计相应的数据项。

2.3.5 GroupBox(组框)组件

该组件将出现在窗体中的相关组件集合在一起,使它们更好的协同工作。典型的应用是集中显示一组单选按钮或复选按钮。

2.3.6 RadioGroup(单选按钮组)组件

该组件用来容纳一组相关的单选按钮,使它们协同工作。主要特性有:

- Columns 特性 指定该组件中单选按钮以几栏进行排列。
- Items 特性 用来显示和编辑在该组件中的各单选按钮旁边的选项名。
- ItemIndex 特性 指示该组件中被选中的单选按钮的序号。默认值是 -1,表示该组件中各单选按钮均未被选中。

【例 2.4】在上例中,如果再增加一组单选按钮表示性别,因为班级和性别是两组互不相干的单选按钮,所以可用 RadioBox 组件实现;爱好是一组相关的复选按钮,可以放在一个 GroupBox 组件中,如图 2.7 所示。实现步骤如下:

(1) 开始一个新的应用程序,在窗体上放置一个 Label 组件,设置方法同上例。

(2) 在窗体上放置一个 RadioGroup 组件,将其 Caption 特性设为“年级”。适当设置 Font(字体)特性,然后设置它的 Item 特性。单击 Item 特性值右端的省略号,在弹出的对话框中依次输入各单选按钮的标题,以回车结束。单击 OK 按钮,即完成了“年级”单选按钮组的设置。用同样方法在窗体上放置“性别”按钮组。



图 2.7 按钮组举例

(3) 在窗体上放置一个 GroupBox 组件,其 Caption 特性为“爱好”。在该组件上依次插入图 2.7 中所示的各复选按钮。

(4) 存盘并运行该程序,用户会发现两组单选按钮之间是互不影响的。

2.3.7 应用举例

【例 2.5】设计模拟猜硬币的正反面游戏的程序,程序运行画面如图 2.8 所示。程序功能



和操作规则如下:当用户选择正面或反面,并按下“确定”按钮后,程序将此值与计算机产生的正、反面随机值进行比较,正确与否在结果框中显示,多次累积的正确率在正确率框中显示;若按下“重新开始”按钮,将重新开始新一轮猜测游戏;按下“关闭”按钮则退出程序。



图 2.8 猜硬币程序运行界面

用户在实现上述程序功能时,涉及到以下几类组件:标签组件(实现结果、正确率字样的显示),编辑框组件(显示结果和正确率的具体值),单选按钮组组件(实现正、反面选择),按钮组件(实现确定、重新开始、退出操作)。

下面给出猜硬币游戏程序的具体实现方法。

1. 界面的设计

调整窗体大小,在窗体中按图 2.8 放置组件。各个组件的功能和主要特性如下:

Label1——标签组件,标记猜测结果,其 Caption 特性为“结果”。

Label2——标签组件,标记猜测的累积正确率,其 Caption 特性为“正确率”。

Edit1——编辑框组件,显示猜测结果。

Edit2——编辑框组件,显示猜测的累积正确率,其 Text 特性为空。

Button1——按钮组件,执行一次猜测的“确定”操作,其 Caption 特性为“确定”。

Button2——按钮组件,执行“重新开始”操作,其 Caption 特性为“重新开始”。

Button3——按钮组件,执行“退出”操作,其 Caption 特性为“退出”。

RadioGroup1——单选按钮组组件,用于接收正、反面的选择,其 Caption 特性为空。

Items 特性设置为“正面”、“反面”。ItemIndex 特性值为 -1,使两个单选按钮均处于未选中状态。

2. 程序的实现

(1) “确定”按钮的 OnClick 事件

单击“确定”按钮后,能够实现将用户在单选按钮组中的选择与计算机产生的随机值进行比较,将结果在 Edit1 中显示,并将累积正确率在 Edit2 中显示。程序代码如下:

```
procedure Button1Click(Sender: TObject);
var
    A,B:Boolean;
begin
    S:=S+1;
    if RadioGroup1.ItemIndex=0 then //表示用户选择的是正面
        A:=True
    else
        A:=False;
    if Random < 0.5 then
```



```

//计算机产生 0~1 之间的随机值,小于 0.5 代表正面,否则为反面
    B:=True
else
    B:=False;
if A=B then //用户的猜测与计算机产生的值相同
begin
    Edit1.Text:= '正确';
    S0:=S0+1;
end
else
    Edit1.Text:= '错误';
    Edit2.Text:= IntToStr(S0)+'/'+IntToStr(S)+'='+'FloatToStr(Round(S0/S*1000)/10)+'%';
    //在编辑框 Edit2 中显示:猜正确次数/总次数=正确率
end;
```

其中, S0 为累积猜正确的次数; S 为总的次数, 因此应在整个单元中起作用, 所以在单元的量说明部分说明。

(2) “重新开始”按钮的 OnClick 事件

单击“重新开始”按钮后, 重新开始新一轮猜测, 因此要将编辑框 Edit1 和 Edit2 置空, S0、S 变量清 0。程序代码如下:

```

procedure TForm1.Button2Click(Sender: TObject);
begin
    Edit1.Text:= '';
    Edit2.Text:= '';
    S:=0; S0:=0;
end;
```

(3) “退出”按钮的 OnClick 事件

当单击“退出”按钮后, 关闭窗口体, 程序结束。代码如下:

```

procedure TForm1.Button3Click(Sender: TObject);
begin
    Close;
end;
```

DP 2.4 其他常用组件

2.4.1 ListBox(列表框)组件

列表框是一个含有若干个选项的显示框, 用户可以在其中进行选择、增加或删除一些选项, ListBox 组件属于 TListBox 类, 位于组件板的 Standard 标签页。

● ItemsIndex 特性



它表示列表框中所选选项的序号,序号从 0 开始排列。

- **Items 特性**

它表示列表框中的选项,可以用 `Items[I]` 表示列表框中的第 $I+1$ 项的内容。可以通过该特性的 `Append`、`Delete` 和 `Insert` 方法在列表框中追加、删除或插入选项。例如: `ListBox1` 为姓名列表框。代码如下:

```
n:=ListBox1.ItemsIndex;           //取当前选项的序号赋给变量 n
Label1.Caption:= ListBox1.Items[n]; //将以 n 值为序号的选项内容显示在标签上
ListBox1.Items.Append('张静');     //将文本行“张静”追加在列表框中
ListBox1.Items.Delete(3);          //从列表框中删除序号为 3 的选项
ListBox1.Items.Insert(1,'李翔');   //将文本行“李翔”插入第 1 号选项位置处
```

- **MultiSelect 特性**

用它决定用户是否可以选定多个选项,其值为 `True` 表示可以选定多个选项。

- **Style 特性**

它确定列表框的显示高度,其取值有:

`LbStandard`——各选项的高度固定且相同;

`LbOwnerDrawFixed`——各选项的高度固定,由 `ItemHeight` 确定;

`LbOwnerDrawVariable`——各选项的高度可变。

- **OnClick 事件**

当单击列表框时触发该事件。

- **Clear 方法**

用它清空列表框中的选项。

2.4.2 ComboBox(组合框)组件

组合框是编辑框与列表框的组合体,既可以向组合框中输入信息,又可以在下拉框中选择选项。`ComboBox` 组件属于 `TcomboBox` 类,位于组件板的 `Standard` 标签页中。它的大部分特性与列表框相同,其特别的特性有:

- **Text 特性**

用它表示在组合框中当前显示的文本信息。

- **DropDownCount 特性**

用它表示组合框中下拉列表的长度,默认为 8。

- **Style 特性**

用它确定组合框的外观,其取值如下:

`csSimple`——组合框由编辑框下面加一个固定的列表框构成。

`csDropDown`——组合框由编辑框和下拉列表框组成。下拉列表框中各选项的高度固定。

`csDropDownList`——组合框由下拉列表框构成。各选项的高度固定,用户不能进行输入操作。

`csOwnerDrawFixed`——组合框由下拉列表框构成。各选项的高度固定,由 `ItemHeight` 确定,用户不能进行输入操作。



csOwnerDrawVariable——组合框由下拉列表框构成。各选项的高度可以不同,用户不能进行输入操作。

2.4.3 StatusBar(状态条)组件

状态条用于显示应用程序的执行状态及相关的提示信息,一般放置在窗体的底部,位于组件板的 Win32 标签页中。状态条可以显示一条信息,也可以分为多个区域显示多条信息。这些应用都可以通过对 StatusBar 组件的各个特性进行设置来实现。

● Panels 特性

用它确定状态条的结构和显示的内容。在对象观察器单击 Panels 特性值右侧的“...”按钮,会打开如图 2.9 所示对话框。其中工具栏左侧的按钮是向状态条中添加新的显示区,按下该按钮后,可以在对象观察器的 Text 特性中设置新显示区的名称;工具栏中第二个按钮的作用是取消当前的显示区;向上和向下的箭头按钮是选择当前显示区。状态条中的显示区可以用 Panels[1] 表示,其中 1 (从 0 开始计数)为显示区的序号。显示区的特性除了 Text 之外,还有

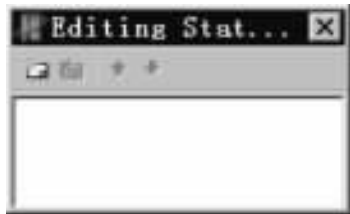


图 2.9 状态条的结构和内容编辑对话框

Alignment 对齐方式特性(取值有 taLeftJustify 左对齐、taRightJustify 右对齐、taCenter 居中)、Bevel 边框风格特性(取值有 pbLowered 向内凹陷、pbRaised 凸出、pbNone 与窗体平齐)、Width 宽度特性、Style 显示类型(取值有 psText 文本、psOwnerDraw 图形)等。

● SimplePanel 特性

其值为 True 时,状态条中只有一个显示区。

● SimpleText 特性

确定具有单一显示区的状态条上显示的文本内容。

● Align 特性

确定状态条在窗体上的位置,取值有 alTop(顶端)、alBottom(底端)、alLeft(左边)、alRight(右边)等。

2.4.4 应用举例

【例 2.6】编写班级选举班长程序,运行界面如图 2.10 所示。该程序要完成在指定候选人中选举,当投票结束,统计出选举结果。

1. 界面设计

调整窗体大小,在窗体中按图 2.10 放置组件,各个组件的功能和主要特性如下:

在窗体上放置 2 个 Panel 组件(位于 Standard 标签页),其中 Panel1 是放置选举结果的矩形框,Panel2 是放置选举过程的矩形框,两个组件的 Caption 特性为空,BorderStyle 特性为 bsSingle(表示外边框为单线);窗体上还要放置 13 个 Label 组件用于显示“选举新任班长”、

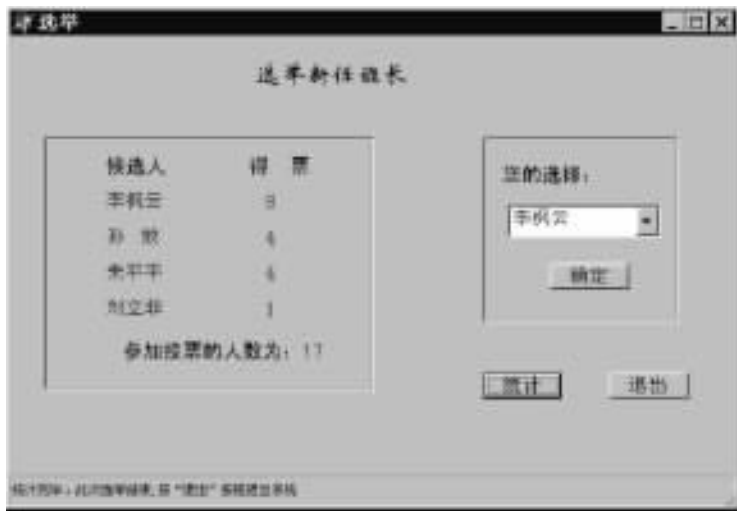


图 2.10 “选举新任班长”应用程序运行界面

“候选人”、“得票”等信息;3 个 Button 组件,分别对应“确定”、“统计”和“退出”按钮;1 个 ComboBox 组件,用于接收选举者的选择,由于是等额选举,不允许选举者录入候选人之外的人选,因此 ComboBox 组件的 Style 特性设置为 csDropDownList,在其 Items 特性中输入各个候选人的名单;1 个 StatusBar 组件,用于显示程序执行的状态,因为只是作为状态提示,所以只需要一个区,故 SimplePanel 特性设置为 True。按图设置各组件的其他特性。

2. 程序实现

在程序中需要统计各个候选人的得票情况及参加选举的总人数,因此设置两个全局量:

```
a:array[0..3] of integer; //累计各个候选人的得票
s:integer;                //累计参加选举的总人数
```

(1) “确定”按钮的事件处理代码

```
procedure TForm1.Button1Click(Sender: TObject);
var
  i:integer;
begin
  i:=combobox1.itemindex; //取组合框中当前选择项的序号
  a[i]:=a[i]+1;           //第 1 号候选人的票数加 1
  s:=s+1;                 //参加选举的人数增加 1
end;
```

(2) “统计”按钮的事件处理代码

```
procedure TForm1.Button2Click(Sender: TObject);
var
  i:integer;
begin
```



```

for i:=0 to 3 do    //显示4个候选人的得票情况
case i of
0:  Label9.Caption:= inttostr(a[i]);
1:  Label10.Caption:= inttostr(a[i]);
2:  Label11.Caption:= inttostr(a[i]);
3:  Label12.Caption:= inttostr(a[i]);
end;
Label13.Caption:= '参加投票的人数为:' + inttostr(s);    //显示总的参加投票人数
for i:=0 to 3 do    //清空选票,准备重新选举
a[i]:=0;
s:=0;
end;

```

(3) “退出”按钮的处理代码

```
Close;
```

(4) 其他程序代码

为了在状态栏中显示程序执行的各种状态,需要设计如下几段程序:

● 窗体创建时,窗体的 OnCreate 事件的处理代码:

```

procedure TForm1.FormCreate(Sender: TObject);
begin
    StatusBar1.SimpleText:= '选择您满意的候选人,按“确定”按钮确认;按“统计”  
按钮查看选举结果;按“退出”按钮退出选举界面;
end;

```

● 选举者在组合框中选择候选人时,组合框内的信息发生改变,组合框的 OnChange 事件的处理代码:

```

procedure TForm1.ComboBox1Change(Sender: TObject);
begin
    StatusBar1.SimpleText:= '您正在选举,按“确定”按钮确认;
end;

```

● 当按下“确定”按钮后,应提示“选举结束”等信息,因此“确定”按钮的处理代码后应加一句:

```

StatusBar1.SimpleText:= '您选举完毕,请下一个同学选举;所有同学选举结束,  
按“统计”按钮显示统计结果;

```

● 当按下“统计”按钮后,应提示“统计完毕”等信息:

```
StatusBar1.SimpleText:= '统计完毕,此次选举结束,按“退出”按钮退出系统'
```



Windows 支持许多标准的对话框,比如,打开对话框、保存对话框、字体设置对话框等等。



Delphi 将这些对话框分别以组件的形式封装起来,放在组件板的 Dialogs 标签页中。可以利用这些组件完成相关的操作。下面介绍几种较常用的对话框组件。

2.5.1 OpenFileDialog(打开对话框)组件

打开对话框给用户提供了一种在多个目录之间切换选择文件的方法。当对 OpenFileDialog 组件执行 Execute 方法时,会弹出如图 2.11 所示的对话框。



图 2.11 打开文件对话框

1. 常用特性

- Filename 特性

它为 String 类型,用于记录最后一个被选中的文件的全路径名。默认为空。

- DefaultExt 特性

当用户未输入文件扩展名时它用于指明默认的扩展名。

- Filter(过滤器)特性

它用于设置打开对话框中显示的文件类型,使在打开的对话框中仅显示与该特性所列的扩展名相匹配的文件。在程序设计阶段用户可以在对象观察器中,单击 Filter 特性值框中的“...”按钮,将弹出 Filter Editor 对话框,可以在此设置预显示文件的扩展名,即设置过滤器。用户还可以在程序代码中动态地设置该特性值。例如:

```
Filter:=' Delphi Unit Files| *.pas; *.dpr';
```

- FilterIndex 特性

该特性为 Integer 类型,在设置了多个过滤器时,用于确定在打开对话框中文件类型下拉列表框中初始显示的过滤器的序号,其默认值为 1。

- InitialDir 特性

用它指定对话框的默认目录。

- Title 特性

用它指明 OpenFileDialog 组件标题栏中的字符串,默认值为“打开”。

2. 常用方法

- Execute 方法

其作用是打开对话框(所有的对话框都具有该方法,用法均相同,以后不再赘述)。用



户单击“打开”或 OK 按钮,该方法返回值为 True;用户单击“取消”或 Cancel 按钮,该方法返回值是 False。通常用程序代码打开文件的结构如下:

```
if OpenFileDialog1.Execute then
begin
    //打开文件操作
end;
```

2.5.2 SaveDialog(保存对话框) 组件

保存对话框用于设置即将保存的文件名字及路径。当对 SaveDialog 组件执行 Execute 方法后,会弹出如图 2.12 所示的保存对话框。SaveDialog 组件除了 Title 特性的默认值为“另存为”外,其他特性与 OpenFileDialog 组件基本相同。



图 2.12 保存文件对话框

2.5.3 FindDialog(查找对话框) 组件

查找对话框可以实现在文本中查找某个字符串。FindDialog 组件属于 TFindDialog 类。对其执行 Execute 方法后,会弹出如图 2.13 所示的查找对话框。



图 2.13 查找对话框

1. 主要特性

- FingText 特性
用它记录要查找的字符串。
- Position 特性
用它控制对话框在屏幕上的位置。
- Options 特性



用它决定查找对话框的外观和行为特征,其子特性(若值为 True 时)如下:

frMatchCase——选中区分大小写复选框。

frDisableMatchCase——区分大小写复选框不可用。

frHideMatchCase——隐藏区分大小写复选框。

frDown——向下查找,反之为向上查找。

frDisableUpDown——方向按钮组不可用。

frhideUpDown——隐藏方向按钮组。

frWholeWord——选中全字匹配复选框。

frDisableWholeWord——全字匹配复选框不可用。

frHideWholeWord——隐藏全字匹配复选框。

frShowHelp——显示帮助按钮。

2. 主要事件

● OnFind 事件

当单击“查找下一个”时,触发该事件。此事件用于实现用户要求的查找工作,实现时按 Options 中所确定的查找方式对 FindText 中的字符串进行查找。

2.5.4 ReplaceDialog(替换对话框)组件

替换对话框如图 2.14 所示,可以接受用户的查找和替换请求。ReplaceDialog 组件属于 TReplaceDialog 类,是 TFindDialog 类的衍生类。也就是说,该组件除了具有 FindDialog 组件的特点外,还具有自己的一些新特性和事件。



图 2.14 替换对话框

● ReplaceText 特性

用户在“替换为”编辑框中输入的字符串。

● Options 特性

比 FindDialog 组件多出两个子特性,即

frReplace: 为 True,表示用替换字符串替换当前找到的字符串;

frReplaceAll: 为 True,表示用替换字符串替换所有找到的字符串。

● OnReplace 事件

当单击“替换”或“全部替换”按钮时,触发该事件。



2.5.5 FontDialog(字体对话框) 组件

该组件用于显示“字体设置对话框”,并把选中的字体信息记录在 Font 特性中。对该组件执行 Execute 方法时,会弹出如图 2.15 所示的对话框窗口。



图 2.15 字体设置对话框

1. 常用特性

● Device 特性

用它确定字体对话框中的字体类型,取值有:fdScreen(显示器字体)、fdPrinter(打印机字体)、fdBoth(显示器及打印机字体)。

● Font 特性

指明选中的字体特性,包括字体(name)、颜色(color)、大小(size)以及字形(style)等信息。

● MaxFontSize 特性和 MinFontSize 特性

用它们指定可供选择的最大、最小字体尺寸。

● Options 特性

它决定了字体对话框的外观和行为。其主要子特性有:

FdAnsiOnly——显示 Windows 的 ANSI 字符集字体。

FdApplyButton——显示“应用”按钮。

FdEffects——显示“效果”复选框和“颜色”列表框。

FdFixPitchOnly——只有固定宽度的字体出现在字体组合框中。

FdNoFaceSel——在对话框出现时,字体组合框中无默认字体。

FdNoOEMFonts——在字体组合框中只出现非矢量字体。

FdNoShowHelp——显示“帮助”按钮。

FdNoSizeSel——在对话框出现时,大小组合框中无默认字体尺寸。

FdNoStyleSel——在对话框出现时,样式组合框中无默认样式。



FdWysiwyg——列出打印机和显示器均可使用的字体。

2. 主要事件

● OnApply 事件

当用户在字体对话框中单击“应用”按钮时,触发该事件。

2.5.6 ColorDialog(颜色对话框)组件

颜色对话框如图 2.16 所示,用于设置字体或背景的颜色。ColorDialog 组件的主要特性如下:

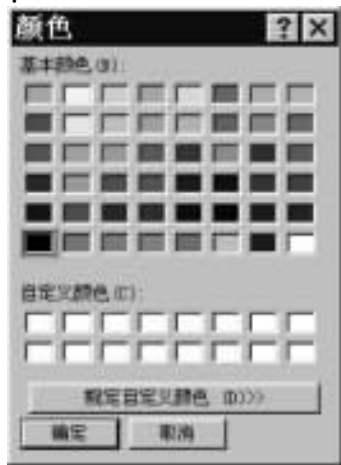


图 2.16 颜色设置对话框

● Color 特性

用它设定颜色对话框中显示的初始颜色,或记录用户所选定的颜色信息。

● CustomColor 特性

用它记录用户自定义的颜色信息。

● Options 特性

用它控制颜色对话框的外观。其子特性有: cdFulOpen(显示自定义颜色框)、cdShowHelp(显示帮助按钮)、cdPreventFullOpen(自定义颜色框不可用)、cdSolidColor(只能使用纯色)、cdAnyColor(可以使用各种颜色),默认 Options 值为空。

2.5.7 PrintDialog(打印对话框)组件

打印对话框如图 2.17 所示,用于设置打印时的打印范围、打印份数、多份打印时的打印方式等。PrintDialog 组件的主要特性如下:



图 2.17 打印对话框



- Copies 特性
用它记录打印文件的份数。
- FromPage 特性
用它记录打印时的起始页码。
- ToPage 特性
用它记录打印时的终止页码。
- PrintRange 特性
用它表示打印范围,取值有:prAllPages(打印所有页)、prSelection(打印选择部分)、prPageNums(打印指定页码的页)。
- Options 特性
用它调整打印对话框的外观和操作。取值有:poHelp(有帮助按钮)、poPageNums(可以自定打印范围)、poPrintToFile(可以输出到文件中)、poSelection(可以打印选定部分)、poWarning(无打印机时出现警告对话框)。

2.5.8 应用举例

【例 2.7】我们都使用过诸如 Word、WPS 或者写字板、记事本等文字编辑软件。下面将利用 Delphi 设计一个自己的编辑软件,称它为文本编辑器,如图 2.18 所示。用户可以使用这个软件,完成文本文件的建立、打开、编辑、保存、打印以及字体设置等操作。在设计“文本编辑器”程序时,除了用到对话框组件外,还要用到加速面板以及文本组件等方面知识。下面在介绍 Delphi 相关知识的同时,逐步实现上述程序。

1. 加速面板的实现

加速面板在 Windows 窗口界面设计中比较常见,它为用户提供了快捷的操作方式:用户通过鼠标点击其上的加速按钮,即可完成相应菜单中的功能。在设计过程中,要注意加速按钮的数量也不宜过多,一般放置较常用的部分按钮。

加速面板设计时,用到两类组件:Panel 组件和 SpeedButton 组件。其中,Panel 组件属于 TPanel 类,位于组件板的 Standard 标签页;SpeedButton 组件属于 TSpeedButton 类,位于组件板的 Additional 标签页。下面分别介绍这两类组件。

(1) SpeedButton(加速按钮)组件

该组件是一个可成组工作的组件,表面也可以显示位图,通常被成组的放置在面板(panel)上,构成加速面板,其主要特性如下:

- AllowAllUp 特性
其值为 True 时,允许同一组的加速按钮都处于弹起状态;否则,同一组的加速按钮必须有且仅有一个处于按下状态。默认值为 False。
- Down 特性



图 2.18 文本编辑器程序运行界面



其值为 True 时,表示加速按钮处于按下状态;否则,处于弹起状态。默认值为 False。

- Flat 特性

其值为 True 时,加速按钮在未被选中时像普通标签一样,仅在鼠标指向它时,才凸现出来。当该特性为 False 时,加速按钮始终凸显示。默认值为 False。

- Glyph 特性

它用于设置出现在按钮上的位图类型。

- GroupIndex 特性

当用户按下加速按钮时指定了该按钮的反应。当 GroupIndex 为 0 时,表示该按钮不与其他按钮协同工作;当 GroupIndex 值大于 0 时,表示该按钮可以与其他 GroupIndex 值相同的按钮协同工作(同一时刻,只有一个按钮处于按下状态)。默认值为 0。

(2) Panel(面板) 组件

该组件是一个可以以多种三维效果显示的矩形区域,用户可以将相关的组件集中在该矩形区域中。Panel 组件为窗体中相关组件提供了一种可视化分组的途径。其主要特性有:

- Align 特性

它指定 Panel 组件在窗体上的位置。

- Alignment 特性

它指定 Panel 的 Caption(标题)在面板上的位置。

- BevelInner(内层)特性和 BevelOuter(外层)特性

它们分别控制该组件的内层和外层表现形式: bvLowered(凹下)、bvNone(无效果)、bvRaised(凸出)。

- BevelWidth 特性

它表示该组件内层和外层的宽度(单位是像素点数)。

- BorderStyle 特性

它指定该组件周围是否有一个单线边框。

2. 多行文本编辑组件

前面介绍的 Edit 组件只能处理单行文本的输入/输出操作,若想实现多行文本的编辑操作,则需要用到 Memo 组件或 RichEdit 组件。

(1) Memo 组件

该组件属于 TMemo 类,用于显示或接收用户输入的多行文本。

- Lines 特性

该特性是 Memo 组件中用得最多的特性,用户对 Memo 中的文本除了用 Text 特性存取外,还可以用 Lines 特性分行处理。Lines 的类型为 Tstrings。例如:

```
Memo1.Lines[4] := 'This is 5th line';
```

```
//向 Memo1 组件的第 5 行赋值(第 1 行为 Lines[0])
```

另外, Tstrings 类提供了许多方法,可以对 Lines 中的数据进行处理。例如:

```
Memo1.Lines.Add('Add a new line'); //在最后增加 1 行
```

```
Memo1.Lines.Delete(4); //删除第 5 行
```

```
Memo1.Lines.Insert(4, 'Insert a new line'); //在第五行前插入 1 行
```



```

Memo1.Lines.Move(4,5);           //将第5行内容移至第6行
Memo1.Lines.Clear;               //清除所有的内容
Memo1.Lines.ReadFromFile('C:\My Documents\AAA.TXT');
                                //将文本文件内容读到该组件中
Memo1.Lines.SaveToFile('A:\Example.TXT'); //将该组件的内容存入一个文件

```

(2) RichEdit 组件

该组件属于 TRichEdit 类,是 TMemo 的衍生类,所以该组件除了能像 Memo 组件一样处理多行文本外,还可以包括大量的编辑格式信息,比如文本的字体、颜色和对齐方式等。主要特性有:

- DefAttributes 特性和 SelAttributes 特性

它们属于 TTextAttributes 类型,用于设置文本的字体、大小、颜色等特性。二者不同之处,DefAttributes 特性用于控制所有使用默认特性的文本;SelAttributes 特性用于控制选定区域中的文本,也会影响到即将输入的文本。其子特性有:

- ◆ Color 特性——确定文本的颜色;
- ◆ Height 特性——控制文本的高度;
- ◆ Name 特性——表示所使用的字体名称;
- ◆ Pitch 特性——控制文本字符的宽度是否可变,取值有 fpFixed(不变)、fpVariable(可变)、fpDefault(由字体决定是否可变);
- ◆ Size 特性——表示字体的大小;
- ◆ Style 特性——表示文本的类型,取值有 fsBold(粗体)、fsItalic(斜体)、fsUnderline(带下划线)、fsStrikeout(带删除线);
- ◆ ConsistentAttributes 特性——表示选定区中的文本是否具有共同的文本特性。
- Paragraph 特性
用它设置光标所在段落或被选定段落的段特性。
- SelLength 特性
用它返回选中文本的长度。

3. 例程的实现

(1) 设置加速面板

首先利用 Panel 组件和 SpeedButton 组件设计加速面板,如图 2.19 所示。

具体步骤如下:

① 在窗体上放置一个 Panel 组件,将 Align 特性设置为 alTop(固定在窗体的顶端),Caption 特性设置为空。

② 在 Panel 组件上依次放置 8 个 SpeedButton 组件,Flat 特性均为 true,Caption 特性均为空,Name 特性依次为 SpeedButtonNew、SpeedButtonOpen、SpeedButtonSave、SpeedButtonPrint、SpeedButton-



图 2.19 加速面板的外观



Cut、SpeedButtonCopy、SpeedButtonPaste 和 SpeedButtonFont。下面设置各个 SpeedButton 组件表面的图案:单击 Glyph 特性右侧的小按钮,将打开 Picture Editor 对话框;再单击对话框中的 Load 按钮,打开 Open Picture 对话框,从中选择一个位图文件;单击“打开”按钮,将返回 Picture Editor 对话框,在此编辑位图文件;如果满意,单击 OK 按钮,就为当前的 SpeedButton 组件指定了一个位图图形。用同样方法,用户可以实现图 2.19 所示的各个按钮设置。

(2) 放置文本编辑组件

在“文本编辑器”程序的设计中,若想实现文本的各项操作,离不开文本组件,因为希望该编辑器有字体设置功能,所以我们选用 RichEdit 组件。具体实现步骤如下:

在窗体上放置一个 RichEdit 组件,将 Align 特性设置为 alClient,使 RichEdit 组件始终占据窗体的客户区域。

(3) 放置各种对话框组件

为了能够实现在打开文件、存盘以及设置字体时,均使用对话框方式操作,因此在窗体上放置 4 个对话框组件:OpenDialog 组件、SaveDialog 组件、PrintDialog 组件和 FontDialog 组件。

(4) 添加程序代码

① 设计首次启动编辑器的界面

功能:在首次启动编辑器时,默认同时打开了一个新文件,其默认文件名为“无标题”。

实现:在对象观察器中选择窗体 Form1,双击 Events 标签页的 OnCreate 事件值位置,在 OnCreate 事件处理过程中输入以下代码:

```
RichEdit1.Clear;  
FileName:= '无标题'  
Caption:= '文本编辑器——'+ FileName;
```

变量 FileName 是用来保存文件名的。用户可以在单元代码的 Implementation 保留字下面加上变量说明:

```
var  
    FileName:String;
```

该变量在整个单元中均起作用,是单元全局变量。

② 新建文件功能的实现

功能:在新建文件之前,当前窗口中的文件若被修改过,则应提示是否保存文件;待用户处理后,编辑窗口中应该清空,准备编辑新文件,同时默认文件名为“无标题”。

实现:双击 SpeedButtonNew 按钮,输入它的 OnClick 事件处理代码如下:

```
if RichEdit1.Modified and (messageDlg('是否保存已更改的文件?',tConfirmation,  
    mbYesNoCancel,0)=mrYes) then SpeedButtonSave.Click;  
RichEdit1.Clear;  
Filename:= '无标题'  
Caption:= '文本编辑器——'+ FileName;
```

③ 打开文件功能的实现

功能:当按下“打开”加速按钮时,同样要确认当前窗口中的文件是否需要存盘。然后,将



用户在打开对话框中选择文件,调入编辑窗口,同时窗口标题栏显示被打开文件的文件名。

实现:双击 SpeedButtonOpen 按钮,输入它的 OnClick 事件处理代码如下:

```
if RichEdit1.Modified and (messageDlg('是否保存已更改的文件?',mtConfirmation,
mbYesNoCancel,0)=mrYes) then SpeedButtonSave.Click;
if OpenFileDialog1.Execute then
begin
    FileName:=OpenDialog1.FileName;
    RichEdit1.Lines.LoadFromFile(FileName);
    Caption:='文本编辑器——'+FileName;
end;
```

其中调用 OpenFileDialog 组件的 Execute 方法,将打开一个对话框。当用户选择了一个文件并按下“打开”按钮时,返回 True 值,并将文件名保存在 FileName 中。

④ 保存文件的实现

功能:当按下“保存”加速按钮时,若当前文件曾经被保存过(已被命名),则直接保存;否则,打开保存对话框,接收用户输入的文件名,同时将文件名在编辑器标题栏中显示。

实现:双击 SpeedButtonSave 按钮,输入它的 OnClick 事件处理代码如下:

```
if FileName<>'无标题' then
begin
    RichEdit1.Lines.SaveToFile(FileName);
    RichEdit1.Modified:=False; //标记当前编辑窗口中的文件已存盘
end
else if SaveDialog1.Execute then
begin
    FileName:=SaveDialog1.FileName;
    RichEdit1.Lines.SaveToFile(FileName);
    Caption:='文本编辑器——'+FileName;
    RichEdit1.Modified:=False;
end;
```

其中调用 SaveDialog 组件的 Execute 方法,将打开一个对话框。当用户选择了一个文件并按下“保存”按钮时,返回 True 值,并将文件名保存在 FileName 中。

⑤ 打印功能的实现

功能:当单击“打印”加速按钮时,将当前文档在打印机上输出。

实现:双击 SpeedButtonPrint 按钮,输入它的 OnClick 事件处理代码如下:

```
PrintDialog1.Execute;
```

⑥ 剪切功能的实现

功能:当单击“剪切”加速按钮时,将当前选定区中的内容转移到剪贴板中。可以用 CutToClipboard 方法实现。

实现:双击 SpeedButtonCut 按钮,输入它的 OnClick 事件处理代码如下:

```
RichEdit1.CutToClipboard;
```



⑦ 复制功能的实现

功能:当单击“复制”加速按钮时,将当前选定区中的内容复制到剪贴板中。可以用 CopyToClipboard 方法实现。

实现:双击 SpeedButtonCopy 按钮,输入它的 OnClick 事件处理代码如下:

```
RichEdit1.CopyToClipboard;
```

⑧ 粘贴功能的实现

功能:当单击“粘贴”加速按钮时,将剪贴板中的内容粘贴到当前光标位置处。可以用 CutToClipboard 方法实现。

实现:双击 SpeedButtonPaste 按钮,输入它的 OnClick 事件处理代码如下:

```
RichEdit1.PasteFromClipboard;
```

⑨ 字体设置的实现

功能:当按下“字体”加速按钮时,将打开字体对话框,接收用户的各项设置;当用户按下“确定”按钮时,有关字体设置作用于编辑窗口的文本。

实现:双击 SpeedButtonFont 按钮,输入它的 OnClick 事件处理代码如下:

```
if FontDialog1.Execute then  
begin  
    if RichEdit1.SelLength>0 then  
        RichEdit1.SelAttributes.Assign(FontDialog1.Font)  
        //Assign 方法使字体对话框中的设定作用于 RichEdit1 组件  
    else  
        RichEdit1.DefAttributes.Assign(FontDialog1.Font);  
end;  
RichEdit1.SetFocus;           //使 RichEdit1 获得聚焦
```

⑩ 最后编译并运行程序,运行结果如图 2.20 所示。



图 2.20 带加速面板的文本编辑器运行界面



DP 2.6 菜单程序设计

菜单为用户执行程序提供了一种简单易行的交互界面。作为一个实用的应用程序一般都有菜单,在菜单中将相关功能的操作分类组织,便于用户使用。下面介绍菜单的设计方法,并逐步在前面的“文本编辑器”中添加菜单功能。

2.6.1 常用菜单组件

常用的菜单组件有以下两类:MainMenu(主菜单),是一个附加在窗体标题条的菜单;PopupMenu(弹出菜单),又称快捷菜单,当用户单击鼠标右键时该菜单出现在屏幕上。图 2.21 所示是菜单各元素的名称。

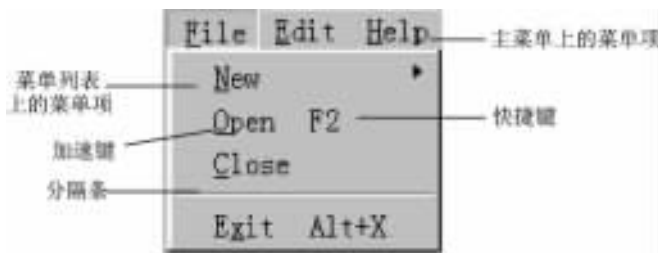


图 2.21 菜单的构成元素

2.6.2 菜单设计器

菜单设计器(menu designer)用于应用程序的菜单设计。

下面以在文本编辑器程序中添加如图 2.22 所示的菜单为例,说明菜单的设计步骤。



图 2.22 添加了菜单的文本编辑器

1. 放置菜单组件

将组件板中 Standard 页的 MainMenu 组件放置于窗体。



2. 打开菜单设计器

选定菜单组件,用以下两种方法均可以打开菜单设计器:一是双击该组件;二是在对象观察器中的 Items 特性,双击取值列的[Menu]或单击(…)按钮。

3. 设计菜单命令

(1) 添加菜单项

打开菜单设计器后在窗口的左上角有一个蓝底的白色矩形框,称为占位符,按回车键定位到对象观察器中相应的 Caption 特性,在其中输入标题,例如输入“文件”;再按回车键,即创建了“文件”菜单,并且菜单占位符下移,用户可继续输入各菜单项名称;若想在“文件”菜单的右侧输入一个菜单,只要按向右的方向键,菜单占位符即可右移,如图 2.23 所示。



图 2.23 菜单设计器窗口

(2) 编辑菜单命令

在菜单设计器窗口中,可以方便地修改、插入和删除菜单项。

修改:选中要改动的菜单项,按回车键对其修改即可。

插入:选择要插入菜单项的位置,按 Insert 键,即可插入一个空白菜单项。

删除:选择要删除的菜单项,按 Del 键即可。

(3) 添加分隔线

有些菜单中,为了区分各类菜单项,常在同类菜单项之间添加一条线来间隔。要添加这条分隔线,只要将该位置菜单项的 Caption 特性设置为连字符(-)即可。

(4) 添加加速键和快捷键

加速键是指在某些菜单和菜单项的名字中带下划线的字母。在菜单未被激活时,可以按 Alt+加速键激活该菜单;在激活菜单时,可以按加速键激活菜单项。用户可以在菜单项的 Caption 特性中设置加速键。例如:将打开菜单项的 Caption 特性设置为“打开(&O)”,则菜单项显示“打开(O)”,字母 O 成为该菜单项的加速键。

快捷键是指出现在菜单项右边的键。在任何情况下,按下快捷键都会直接选择该菜单项。用户可以在菜单项的 ShortCut 特性中,设置快捷键。例如:将“打开”菜单项的 ShortCut 特性设为 F2,则 F2 成为该菜单项的快捷键。

(5) 创建子菜单

在使用菜单时,常见到某些菜单中还附有子菜单。子菜单可以为主菜单提供附加的相关命令,合理的使用子菜单,可以减少单层菜单的命令数,节约垂直方向上的屏幕空间。创建方法如下:用户选中要创建子菜单的菜单项,单击鼠标右键,在弹出菜单中选择 Creat SubMenu 菜单项或使用快捷键 Ctrl+→,出现子菜单占位符,键入子菜单的菜单项。按回车键或↓,继续创建其他子菜单项。当创建结束,按 ESC 键返回上一级菜单。

用户可以按以上步骤创建如图 2.22 所示的菜单结构。其中编辑菜单包括剪切、复制、粘



贴等菜单项。格式菜单中包括字体菜单项。

4. 编写菜单事件处理代码

一般情况下,每个菜单项有一个 OnClick 事件。在菜单设计器中双击菜单项,即可进入该菜单项的 OnClick 事件处理过程进行代码编写。由于各个菜单中的菜单项与加速栏上的各个按钮依次对应,功能相同,因此,只要将各个菜单项的 OnClick 事件与各个加速按钮的 OnClick 事件相关联即可。方法如下:打开菜单设计器,选中“新建”菜单项,在对象观察器的 Event 标签页中,选中 OnClick 事件,从它的下拉列表框中选取 SpeedButtonNewClick,这样即建立了关联。当用户选取“新建”菜单项时,执行的是 SpeedButtonNew 的 OnClick 事件。用同样的方法建立另外其他关联。在 Exit 菜单项的 OnClick 事件处理过程中添上程序代码 Close。到此为止,一个简单的文本编辑器初具规模。用户可以仿照例程添加一些菜单项和按钮,继续完善。

DP 2.7 多文档界面 MDI 设计

Windows 应用程序可以分为单文档 SDI 应用程序和多文档 MDI 应用程序两类。单文档应用程序中一般只有一个窗体,或者在多个独立的窗体中,各个窗体的活动不受主窗体的限制。在多文档应用程序中主窗体(父窗体)包含其他窗体(子窗体),子窗体隶属于主窗体,子窗体的位置随主窗体的改变而改变,子窗体随主窗体的关闭而关闭。例如 Word 就属于多文档应用程序,在 Word 主窗口中可以同时打开多个文档,当主窗口关闭时,所有文档也随之关闭。下面说明多文档界面的设计方法。

2.7.1 创建父窗体和子窗体

1. 创建父窗体

新建一个项目,将默认窗体的 Caption 特性设置为“父窗体”,Name 特性设置为 ParentForm,FormStyle 特性设置为 fsMDIForm。

2. 添加子窗体

选择 File|New Form 菜单项,添加一个新窗体。将该窗体的 Caption 特性设置为“子窗体”,Name 特性设置为 ChildForm,FormStyle 特性设置为 fsMDIChild。

3. 存盘并运行该程序

如图 2.24 所示,子窗体会自动出现在父窗体中。

4. 动态生成子窗体

若想使程序运行时只看到父窗体,而子窗体打开和关闭是由用户控制的。也就是说,子窗体是在程序中动态的生成的。可以采用以下办法:



图 2.24 父窗体和子窗体运行界面

选择 Project|Options 菜单项,弹出如图 2.25 所示的对话框。其中 Auto—create forms 列表框中列出的是自动生成的窗体名称,Available forms 中列出的是动态生成的窗体名称。因为 ParentForm 窗体在程序运行时希望直接看到,所以保留在 Auto—create forms 框中。要使 ChildForm 子窗体动态生成,只要选中 ChildForm 名称,单击“>”按钮,该子窗体就进入 Available forms 列表框中。

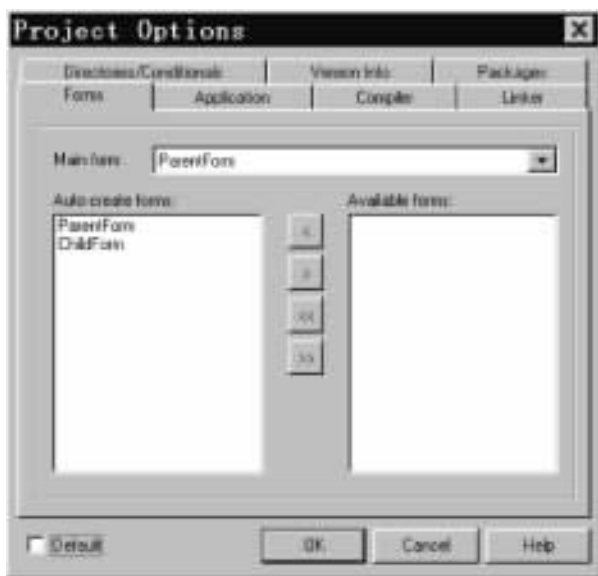


图 2.25 窗体选项设置

若想生成子窗体 ChildForm,需要如下代码:

```
var  
ChildForm:TChildForm;  
ChildForm:= TChildForm. Create(self);  
ChildForm. Show;
```

若要关闭子窗体可以调用 Close 方法,但此时关闭窗口操作只相当于最小化。为了能达到真正关闭窗口体的目的,需要再调用 Release 方法释放窗体所占的内存;或者在窗体的



OnClose事件中加入如下代码:

```
Action := caFree;
```

2.7.2 MDI 窗体的特性与方法

- ActiveMDIChild 特性

该特性为 TForm 类,存储父窗体中活动的子窗体;如果无子窗体,其值为 nil。

- MDIChildCount 特性

它记录父窗体中子窗体的个数。

- MDIChildren 特性

该特性值是记录父窗体中所有子窗体的数组,可以用 MDIChildren[I-1]访问第 I 个子窗体。

- WindowMenu 特性

若将其值设为父窗体中的某个菜单名,则所有打开的子窗体的名称将会显示在该菜单的下面。

- Tile 方法

它将各个子窗体平铺排列。

- Cascade 方法

它将各个子窗体层叠排列。

- ArrangeIcons 方法

它排列最小化的子窗体图标。

- Next 方法

它使下一个子窗体成为活动窗体。

- Previous 方法

它使前一个子窗体成为活动窗体。

2.7.3 多文档界面 MDI 中菜单的设计

在多文档界面的设计中,父窗体和子窗体都可以有自己的菜单。当程序运行时,父窗体上将包含子窗体上的菜单;当子窗体关闭时,父窗体的菜单会恢复原样。

下面以实例说明在 MDI 程序中的菜单设计方法。

【例 2.8】设计一个多文档应用程序,其中父窗体和子窗体的菜单分别如图 2.26 和 2.27 所示。父窗体的菜单主要完成子窗体的创建和对多个子窗体进行平铺和层叠方式的排列;子窗体的菜单主要完成对子窗体的最大化、最小化、恢复及关闭等功能。



图 2.26 父窗体的菜单



图 2.27 子窗体的菜单



具体实现步骤如下。

(1) 在父窗体上放置 MainMenu 组件,按图 2.26 所示设置各个菜单项。

(2) 设置父窗体中的各个菜单项的事件处理代码。

“创建”菜单项的处理程序如下:

```
procedure TParentForm.CreateClick(Sender: TObject);
var
    ChildForm: TChildForm;
begin
    ChildForm := TChildForm.Create(self);      //创建新的子窗体
    ChildForm.Caption := '子窗体' + inttostr(MDICHildCount);
                                                //新窗体名为“子窗体 i”,i 为该子窗体的序号
    ChildForm.Show;                            //显示创建的子窗体
end;
```

“关闭”菜单命令的处理程序如下:

```
procedure TParentForm.ExitClick(Sender: TObject);
begin
    close;
end;
```

“水平平铺”菜单命令的处理程序如下:

```
procedure TParentForm.HorizontalClick(Sender: TObject);
begin
    Tilemode := tbHorizontal;                //执行 Tile 方法之前,要设置平铺模式
    Tile;
end;
```

“垂直平铺”菜单命令的处理程序如下:

```
procedure TParentForm.VerticalClick(Sender: TObject);
begin
    Tilemode := tbVertical;
    Tile;
end;
```

“层叠”菜单命令的处理程序如下:

```
procedure TParentForm.CascadeClick(Sender: TObject);
begin
    Cascade;
end;
```

(3) 在子窗体上放置 MainMenu 组件,按图 2.27 所示设置各个菜单项。

(4) 设置子窗体中的各个菜单项的事件处理代码。

“最大化”、“最小化”及“恢复”菜单项的处理代码分别如下:



```
WindowState:= wsMaximized;
WindowState:= wsMinimized;
WindowState:= wsNormal;
```

“关闭”菜单项的事件处理程序如下：

```
procedure TChildForm. CloseClick(Sender: TObject);
begin
    Close;
end;
```

为了保证使子窗体从内存释放掉,还要在子窗体的 OnClose 事件的处理程序中作如下设置：

```
procedure TChildForm. FormClose(Sender: TObject; var Action: TCloseAction);
begin
    Action:= caFree;
end;
```

(5) 若要将所有子窗体的名称列表显示在“窗体”菜单的下面,可以将父窗体的 WindowsMenu 特性设置为 FormMenu(窗体菜单项的名称)。

程序的运行结果如图 2.28 所示,图中为连续创建三个子窗体,并且按垂直平铺排列方式显示,各个子窗体的名字显示在“窗体”菜单的底部,子窗体 3 前面带√,表示为当前活动窗体。

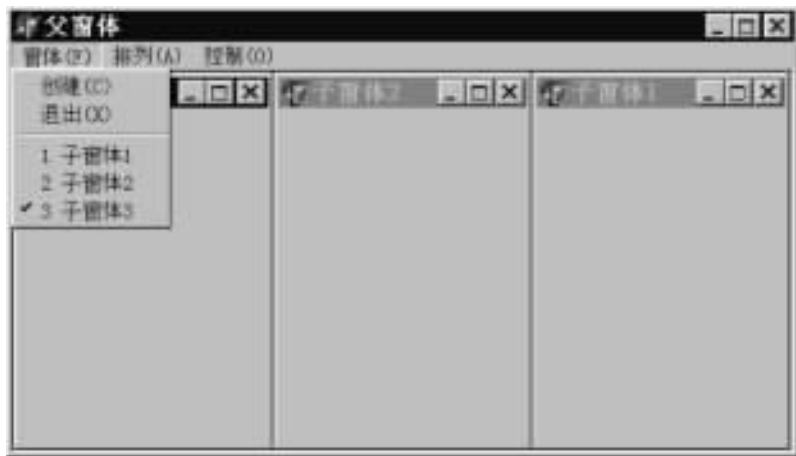


图 2.28 多文档应用例程运行结果



2.1 设计一个不能改变大小的只含有最小化按钮的窗体。

2.2 利用标签、编辑框及按钮组件设计鸡、兔同笼问题:已知笼子中鸡和兔的总头数与总脚数,计算鸡和兔子各有多少只(运行界面如图 2.29 所示)。

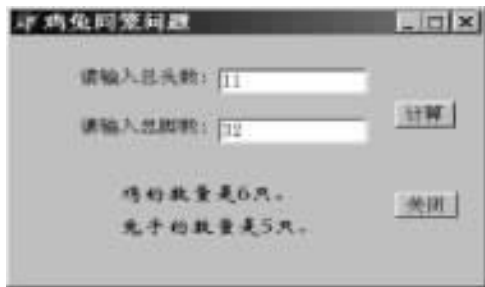


图 2.29 鸡、兔同笼问题运行界面

2.3 在 2.2 题基础上修改程序,使程序在运行中若接收的鸡、兔的总头数和总脚数为负数或小数,或是在鸡、兔的数量不是整数时,在屏幕上出现如图 2.30 所示的出错提示框。

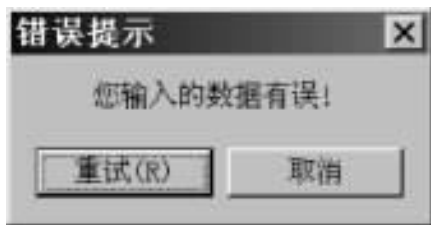


图 2.30 鸡、兔同笼问题出错提示框

- 2.4 将【例 2.4】补充完整,使其能够统计出各个年级中男生、女生的每种爱好的比例。
- 2.5 对键盘上输入的文本统计指定的元音字母出现的次数(利用 RichEdit 组件接收文本、ComboBox 组件选择元音字母)。
- 2.6 为 2.5 题的程序增加存盘功能(将原始文本及统计结果分别以文件形式存入磁盘)。
- 2.7 为 2.6 节的“文本编辑器”程序增加弹出菜单功能(弹出菜单中包括新建、打开、剪切、复制、粘贴、字体等菜单项)。
- 2.8 什么叫多文档应用程序?



第3章

图形、图像与多媒体设计

Delphi 不仅为用户提供了简洁有效的图形、图像处理能力,而且提供了功能强大、使用灵活的多媒体组件,使开发图形界面的多媒体应用程序变得轻松自如。

DP 3.1 绘制图形

这里的图形是指由平面上的点线构成的几何图案。

3.1.1 Canvas 对象

Canvas 对象属于 Tcanvas 类。Delphi 将常用的绘图功能封装在 Tcanvas 类中,通过设置 Canvas 对象的特性,调用其方法即可实现常用的画图功能。用户在组件板上看不到 Canvas 对象,因为它是各种绘图组件的一种特性。常用的绘图组件有 Form(默认的绘图组件)、PaintBox、Shape 等。

Canvas 对象的主要特性有:

- Brush(刷)特性

该特性为矩形、圆或多边形等封闭图形提供填充色和填充图案,为文本提供背景色。其子特性有 Color(确定填充颜色)、Style(确定填充图案)。Color 的取值有: clBackground(桌面背景色)、clBlack(黑色)、clBlue(蓝色)、clBtnFace(按钮表面颜色)、clGray(灰色)、clGreen(绿色)、clLime(草绿色)、clPurple(紫色)、clRed(红色)、clTeal(宝石蓝色)、clWhite(白色)。Style 的取值有: bsBDDiagonal(斜线图案)、bsClear(无填充图案)、bsCross(横向交叉图案)、bsDiagCross(斜向交叉图案)、bsFDDiagonal(反向斜线图案)、bsHorizontal(横线图案)、bsSolid(实心图案)、bsVertical(竖线图案)。

- Font(字体)特性

该特性用于设置 Canvas 对象中显示文字的 size(大小)、Color(颜色)、Style(样式)等。

- Pen(笔)特性

该特性设置图形中线条的颜色、粗细和类型等。其子特性有 Color(确定线条的颜色,与 Brush 的 Color 特性相同)、Style(用于控制线条的类型)、Width(确定线条的宽度)、Mode(绘图模式)。Style 特性的取值有 psClear(清除线)、psDash(长划线)、psDot(点线)、psDashDot(点划线)、psdashDotDot(双点划线)、psSolid(实线); Mode 特性的取值有 pmBlack(总是黑色)、pmWhite(总是白色)、pmNot(取背景的反转颜色)、pmCopy(取 Color 特性设置的颜色)、pmXor(将画笔颜色与背景的颜色求异或)、pmNotXor(将 pmNot 的颜色取反)等。



● Pixels(像素)特性

像素是屏幕上图形的最小单位,对应屏幕上的点。每个点都由 x 和 y 坐标确定其位置,每个点的取值为颜色值,表示方法为 `Pixels[x,y]`。因此,可以通过给每个像素着色来绘制图形。当然效率较低,一般用在图形的局部处理上。

例如:可以用以下程序段在屏幕上横坐标从 200 到 600,纵坐标为 100 位置处画一条蓝色的直线。其代码如下:

```
for i:= 200 to 600 do  
    Canvas.Pixels[i,100]:=clBlue;
```

3.1.2 绘制各种图形

在绘制图形前,应设置好 Canvas 对象的 Brush 和 Pen 特性,例如,作如下设置:

```
Canvas.Brush.Color:=clRed;  
Canvas.Brush.Style:=bsSolid;  
Canvas.Pen.Color:=clBlue;  
Canvas.Pen.Width:=3;
```

下面,学习绘制各种图形。

1. 直 线

可以使用 `MoveTo` 和 `LineTo` 方法绘制直线,其语句如下:

```
procedure MoveTo(x,y:Integer);  
procedure LineTo(x,y:Integer);
```

- `MoveTo` 方法:将画笔位置移到 (x,y) 所指定的坐标上。
- `LineTo` 方法:实现从画笔当前位置处到 (x,y) 坐标位置处画一条直线[不包括 (x,y) 点],同时画笔的位置移到 (x,y) 位置处,直线具有 Pen 中定义的各种特性。例如:

```
Canvas.MoveTo(100,200);  
Canvas.LineTo(400,200);
```

以上代码可以实现从 $(100,200)$ 到 $(400,200)$ 位置处画一条 3 磅宽的蓝色直线。

2. 矩 形

- `Rectangle` 方法用于绘制矩形,其语句如下:

```
procedure Rectangle(x1,y1,x2,y2:integer);
```

其中参数 $x1,y1$ 表示矩形左上角坐标, $x2,y2$ 表示矩形右下角坐标。

- `RoundRect` 方法用于绘制圆角矩形,其语句如下:

```
procedure RoundRect(x1,y1,x2,y2,x3,y3:integer);
```

其中参数 $x1,y1,x2,y2$ 同上,参数 $x3,y3$ 表示圆角的宽和高。例如:

```
Canvas.Rectangle(100,100,400,200);
```

将在屏幕上画出一个矩形,左上角坐标为 $(100,100)$,右下角坐标为 $(400,200)$,边框为蓝色,内部填充红色。而

```
Canvas.RoundRect(100,300,400,400,40,80);
```



将在屏幕上画出一个圆角矩形,圆角宽 40,高 80。

- **FillRect 方法**用于空心矩形,其语句如下:

```
procedure FillRect(const Rect:TRect);
```

- **FrameRect 方法**用于绘制实心无边框的矩形,其语句如下:

```
procedure FrameRect(const Rect:TRect);
```

其中参数 Rect 表示矩形的位置和大小。例如:

```
Canvas.FillRect(Rect(100,100,300,200));
```

在屏幕上显示一个红色实心无边框的矩形,其语句如下:

```
Canvas.FrameRect(Rect(400,100,600,200));
```

在屏幕上显示一个红色边框的空心矩形。

3. 折 线

- **PolyLine 方法**用于绘制一条折线,其语句如下:

```
procedure PolyLine(Points:array of Tpoint);
```

其中参数 Points 是不确定点的坐标构成的数组,绘制的折线的各个顶点的坐标可以用逗号间隔,放在方括号中作为实际参数传入。例如:

```
Canvas.PolyLine([Point(160,40),Point(80,240),Point(280,120),  
Point(40,120),Point(240,240),Point(160,40)]);
```

以上代码在屏幕上显示一个由折线连成的五角星,因为首尾顶点坐标相同,所以呈多边形,但内部没有填充颜色。

4. 多边形

- **Polygon 方法**用于绘制多边形,其语句如下:

```
procedure Polygon (Points:array of Tpoint);
```

其中参数 Points 表示各点的坐标,格式同 PolyLine 方法。例如:

```
Canvas.Polygon([Point(160,40),Point(80,240),Point(280,120),  
Point(40,120),Point(240,240)]);
```

在屏幕上显示一个五角星,由边线构成的封闭区域的填充色为红色。

可以看出,在使用 Polygon 方法绘制多边形时,首尾顶点不必一致。

5. 圆和椭圆

- **Ellipse 方法**用于绘制圆和椭圆,其语句如下:

```
procedure Ellipse(x1,y1,x2,y2;integer);
```

该方法可以绘制与矩形(x1,y1,x2,y2)相切的椭圆,当矩形呈正方形时,所绘制的图形就是一个圆。

- **Chord 方法**用于绘制椭圆中的一个弓形,其语句如下:

```
procedure Chord(x1,y1,x2,y2,x3,y3,x4,y4;integer);
```

其中参数 x1、y1、x2、y2 确定一个椭圆,参数 x2、y2、x3、y3、x4、y4 用于确定直线与椭圆的两个交点。



6. 字符串的图形显示

- **TextOut** 方法用于将字符串以图形方式显示,其语句如下:

```
procedure TextOut(x,y:integer;const Text:string);
```

参数 *x*、*y* 表示字符串要显示的位置,参数 *Text* 表示要显示的字符串。

字符串的风格由 *Canvas* 对象的 *Font* 特性确定。例如:

```
Canvas.Font.Color:=Clgreen;  
Canvas.Font.Size:=40;  
Canvas.Font.Style:= [fsItalic];  
Canvas.TextOut(100,200,'ABCD');
```

以上程序段的作用是在(100,200)位置处显示绿色、40 磅斜体字符串“ABCD”。

3.1.3 绘图组件

Delphi 除了提供在窗体上直接绘制图形外,还提供了 *PaintBox* 和 *Shape* 等绘图组件。

1. PaintBox(绘图板)组件

PaintBox 组件位于 *System* 标签页中。该组件提供了一个矩形的绘图区,用户可以在该区域内利用 *Canvas* 对象绘制各种图形。该区域内的图形采用相对左上角为原点的相对坐标。例如,在 *PaintBox1* 组件的区域内显示位图文件 *f:\init.bmp*,其程序段如下:

```
var  
    bmp:Tbitmap;  
begin  
    bmp:=Tbitmap.Create;  
    bmp.LoadFromFile('f:\init.bmp');  
    PaintBox1.Canvas.Draw(0,0,bmp);  
end;
```

其中 *Canvas* 对象的 *Draw* 方法的前两个参数表示位图相对 *PaintBox1* 组件左上角的坐标值,后一个参数表示要显示的位图对象。

细心的读者会发现,当对显示位图的窗体大小调整后(比如最小化),位图不能正常显示。为了避免这种情况,可以在 *Canvas* 对象的 *OnPaint* 事件的处理程序中添加重画代码,例如:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);  
begin  
    PaintBox1.Canvas.Draw(0,0,bmp);  
end;
```

2. Shape(图形)组件

Shape 组件位于 *Additional* 标签页中,可以实现在窗体中绘制一些简单的几何图形。其主要特性有 *Pen*、*Brush* 和 *Shape* 等。



- Pen 特性用于确定图形的边框线。
- Brush 特性用于确定图形的填充图案。
- Shape 特性用于确定图形的形状,其取值有 stCircle(圆形)、stEllipse(椭圆)、stRectangle(矩形)、stRoundRect(圆角矩形)、stRoundSquare(圆角正方形)、stSquare(正方形)。Left、Top、Width 和 Height 特性确定了图形的位置和大小。

3.1.4 应用举例

利用 Delphi 的绘图功能设计一个简单的图形编辑器,如图 3.1 所示。该编辑器可以提供各种图形及颜色的选择。利用该编辑器可以绘制各种颜色的矩形、圆形和直线。



图 3.1 图形编辑器的设计界面

1. 界面的设计

(1) 新建一个项目,空白窗体的 AutoSize 特性设置为 True;Caption 特性设置为“图形编辑器”,并在其上放置一个 MainMenu 组件,使其包含三个菜单。其中“文件”菜单包含“新建”、“打开”、“保存”和“退出”菜单项;“图形”菜单包含“矩形”、“圆形”、“直线”菜单项;“颜色”菜单包含“画笔”和“画刷”两个菜单项。

(2) 在窗体上放置 OpenFileDialog 组件、SaveDialog 组件、ColorDialog 组件各一个,分别用于显示打开对话框、保存对话框和颜色对话框。其中 OpenFileDialog 组件和 SaveDialog 组件的 Filter 特性均设为 bmp|*.bmp。

(3) 在窗体上放置一个 Image 组件,用作绘制图形的画板,其 Align 特性设置为 alClient。

2. 程序的实现

(1) 定义公用变量

```
drawing:boolean;           //标志是否开始绘图
Origin,MovePoint:TPoint;  //绘制图形的起始点、终止点
C:char;                   //表示用户所选择的图形类别
```

(2) 定义绘制图形的公共过程

该过程能根据用户的选择绘制不同的图形,代码如下:



```
procedure drawShape(Image1:TImage;TopLeft,BottomRight:TPoint;
pMode:TPenMode);
begin
  with Image1.Canvas do
  begin
    Pen.Mode:=pMode;
    case c of
      'r': //绘制矩形
      begin
        Rectangle(TopLeft.x,TopLeft.Y,BottomRight.X,BottomRight.Y);
      end;
      'c': //绘制圆形
      begin
        Ellipse(TopLeft.x,TopLeft.Y,BottomRight.X,BottomRight.Y);
      end;
      'l': //绘制直线
      begin
        MoveTo(TopLeft.x,TopLeft.y);
        Lineto(BottomRight.x,BottomRight.Y);
      end;
    end;
  end;
end;
```

(3) “矩形”菜单项的 OnClick 事件处理代码

```
procedure TForm1.Rect1Click(Sender: TObject);
begin
  c:='r';
end;
```

(4) “圆形”菜单项的 OnClick 事件处理代码

```
procedure TForm1.Circle1Click(Sender: TObject);
begin
  c:='c';
end;
```

(5) “直线”菜单项的 OnClick 事件处理代码

```
procedure TForm1.Line1Click(Sender: TObject);
begin
  c:='l';
end;
```

(6) “画刷”颜色菜单项的 OnClick 事件处理代码



```
procedure TForm1.Brush1Click(Sender: TObject);
begin
    ColorDialog1.Color := Image1.Canvas.Brush.Color;
    if ColorDialog1.Execute then
        Image1.Canvas.Brush.Color := ColorDialog1.Color;
end;
```

(7) “画笔”颜色菜单项的 OnClick 事件处理代码

```
procedure TForm1.Pen1Click(Sender: TObject);
begin
    ColorDialog1.Color := Image1.Canvas.Pen.Color;
    if ColorDialog1.Execute then
        Image1.Canvas.Pen.Color := ColorDialog1.Color;
end;
```

(8) “新建”菜单项的 OnClick 事件处理代码

```
procedure TForm1.New1Click(Sender: TObject);
var
    Area: TRect;
    OldColor: TColor;
begin
    //初始化绘图区及画笔、画刷的设置
    Area := Rect(0, 0, Image1.Picture.Width, Image1.Picture.Height);
    OldColor := Image1.Canvas.Brush.Color;
    Image1.Canvas.Brush.Color := clWhite;
    Image1.Canvas.FillRect(Area);
    Image1.Canvas.Brush.Color := OldColor;
end;
```

(9) “打开”菜单项的 OnClick 事件处理代码

```
procedure TForm1.open1Click(Sender: TObject);
var
    PenCol, BrushCol: TColor;
    PenSize: Integer;
begin
    if OpenFileDialog1.Execute then
begin
        //打开文件后初始化画笔、画刷设置
        PenCol := Image1.Canvas.Pen.Color;
        BrushCol := Image1.Canvas.Brush.Color;
        PenSize := Image1.Canvas.Pen.Width;
        Image1.Picture.LoadFromFile(OpenDialog1.Filename);
        Image1.Canvas.Pen.Color := PenCol;
```



```
Image1.Canvas.Brush.Color := BrushCol;  
Image1.Canvas.Pen.Width := PenSize;  
end;  
end;
```

(10) “保存”菜单项的 OnClick 事件处理代码

```
procedure TForm1.save1Click(Sender: TObject);  
begin  
if SaveDialog1.Execute then  
Image1.Picture.SaveToFile (SaveDialog1.FileName);  
end;
```

(11) “退出”菜单项的 OnClick 事件处理代码

```
procedure TForm1.exit1Click(Sender: TObject);  
begin  
Close;  
end;
```

(12) Image1 组件的 OnMouseDown(鼠标被按下)事件处理代码

```
procedure TForm1.Image1MouseDown(Sender: TObject; Button: TMouseButton;  
Shift: TShiftState; X, Y: Integer);  
begin  
if Button = mbLeft then  
// 若鼠标的左键被按下,则绘图开始,并记下起始点  
begin  
Drawing := True;  
Image1.Canvas.MoveTo(X,Y);  
Origin := Point(X,Y);  
MovePoint := Origin;  
end;  
end;
```

(13) Image1 组件的 OnMouseMove(鼠标移动)事件处理代码

```
procedure TForm1.Image1MouseMove(Sender: TObject; Shift: TShiftState;  
X, Y: Integer);  
begin  
if Drawing then  
begin  
// 鼠标移动则绘制图形  
DrawShape(Image1,Origin,MovePoint,pmNotXor);  
MovePoint := Point(X,Y);  
// 若鼠标在移动时回到原位置,表示没有绘制图形  
DrawShape(Image1,Origin,MovePoint,pmNotXor);
```



```
end;  
end;
```

(14) Image1 组件的 OnMouseUp(鼠标被释放)事件处理代码

```
procedure TForm1.Image1MouseUp(Sender: TObject; Button: TMouseButton;  
    Shift: TShiftState; X, Y: Integer);  
begin  
    if Drawing then  
        // 鼠标释放时按设定的颜色绘制图形,绘图完毕  
        begin  
            DrawShape(Image1, Origin, Point(X, Y), pmCopy);  
            Drawing := False;  
        end;  
    end;  
end;
```

程序的运行结果如图 3.2 所示。

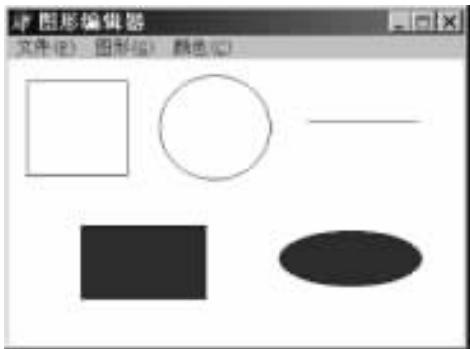


图 3.2 图形编辑器的运行界面

DP 3.2 图像处理

通过 Delphi 程序可以调用图像文件中的图像并对其进行各种处理。

3.2.1 常用图像类

Delphi 可以处理的图像文件有位图文件、图标文件、图元文件三种。这三种文件对应的类均是 TGraphic 图像类的派生类。Delphi 中的常用的图像类有 TBitmap、TGraphic 和 TPicture。

1. TBitmap 类

该类用于接受位图文件,在图像处理中用得比较多。其主要方法有:

- Create 方法



该方法用于创建一个 TBitmap 类的实例。例如：

```
var  
    Bmp:Tbitmap;           //定义一个 TBitmap 类的变量  
    Bmp:=Tbitmap.Create;   //创建一个 TBitmap 类的实例存入变量 Bmp
```

- LoadFromFile 方法

该方法可以从磁盘中读取一个位图文件到 TBitmap 类的实例。例如：

```
Bmp.LoadFromFile('C:\windows\Clouds.bmp');
```

- SaveToFile 方法

该方法将 TBitmap 类的实例存入磁盘中。例如：

```
Bmp.SaveToFile('C:\windows\clouds1.bmp');
```

2. TGraphic 类

该类一般用作参数或组件的特性类型,可以把位图、图标、图元等各种图像文件对象赋给 TGraphic 类型的参数、变量或特性。

Canvas 对象的 Draw 方法和 StretchDraw 方法的语句如下：

```
procedure Draw(X, Y: Integer; Graphic: TGraphic);
```

将在左上角坐标为(X,Y)处,显示 Graphic 指定的图像。

```
procedure StretchDraw(const Rect: TRect; Graphic: TGraphic );
```

将 Graphic 指定的图像按比例缩放显示在 Rect 指定的矩形区中。例如：

```
Canvas.StretchDraw(Rect(100,100,400,400),Bmp);
```

将 Bmp 中的图像显示在矩形区域(100,100,400,400)中。

3. TPicture 类

该类的对象可以接受各种图像文件,而且它的文件处理方法是靠文件的扩展名来确定图像的类型的。

3.2.2 Image(图像)组件

Image 组件位于 Additional 标签页中,可以在窗体上显示一幅图像;与 Canvas 对象不同,它能够自动处理 Onpaint 事件,不必由用户设计重画程序代码。其主要特性有：

- Picture 特性

该特性属于 TPicture 类,用于保存 Image 组件中显示的图像信息(可以是各种类型的图像文件)。

- Canvas 特性

该特性用于在 Image 中绘制位图。

- Center 特性

该特性控制图像居中显示。

- Stretch 特性



该特性的值为 True 时,图像调整自身比例充满整个显示区;否则,按原始尺寸显示。

3.2.3 应用举例

设计一个如图 3.3 所示的能实现图像显示及复制的程序。

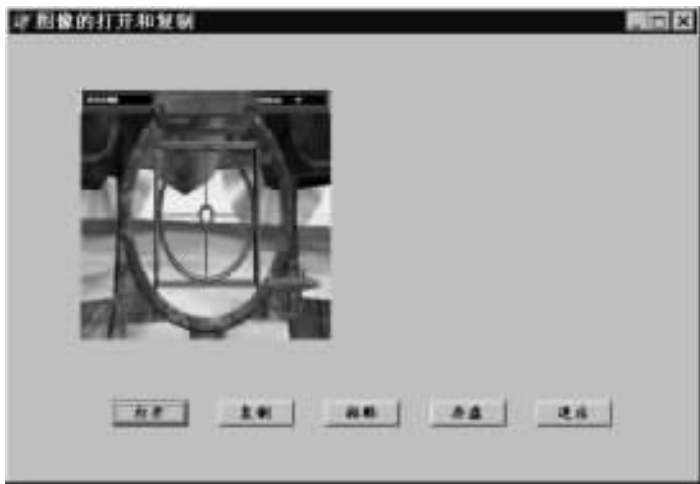


图 3.3 图像的打开与复制程序运行界面

1. 界面的设计

在窗体上放置如下组件:

- 2 个 Image 组件,其中 Image1 用于显示要打开的图像,Image2 用于显示复制的图像。这 2 个组件的 Stretch 特性均为 True。
- 1 个 OpenFileDialog 组件,用于打开已存在的图像文件。
- 1 个 SaveDialog 组件,用于保存复制出的图像文件。
- 5 个按钮组件,分别对应“打开”、“复制”、“粘贴”、“存盘”及“退出”功能。

2. 程序的实现

(1) “打开”按钮 OnClick 事件的处理代码

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  if opendialog1.execute then
  begin
    CurFileName:=Opendialog1.FileName;
    //取“打开”对话框中选定的文件名
    Image1.Picture.LoadFromFile(CurFileName);
    //将打开的图像文件显示在 Image1 框中
  end;
end;
```



(2) “复制”按钮 OnClick 事件的处理代码

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    Clipboard.Assign(Image1.Picture);
    //将 Image1 中的图像复制到剪贴板上
end;
```

(3) “粘贴”按钮 OnClick 事件的处理代码

```
procedure TForm1.Button3Click(Sender: TObject);
var
    BitMap: TBitmap;
begin
    if Clipboard.HasFormat(CF_BITMAP) then
    begin
        Bitmap := TBitmap.Create;
        //创建一个位图对象
    try
        BitMap.Assign(Clipboard);
        //从剪贴板上取得位图信息
        Image2.Canvas.StretchDraw(Rect(0,0,250,250), BitMap);
        //将位图显示在 Image2 中
    finally
        Bitmap.Free;
        //释放 BitMap 中的图像信息
    end;
    end;
end;
```

(4) “存盘”按钮 OnClick 事件的处理代码

```
procedure TForm1.Button4Click(Sender: TObject);
begin
    if SaveDialog1.Execute then
    begin
        CurFileName := SaveDialog1.FileName;
        //取“保存”对话框中当前的文件名
        Image2.Picture.SaveToFile(CurFileName);
        //将 Image2 中的图像存入设定的文件中
    end;
end;
```

(5) “退出”按钮 OnClick 事件的处理代码

```
procedure TForm1.Button5Click(Sender: TObject);
```



```
begin  
    Close;  
end;
```

DP 3.3 多媒体开发

多媒体是指集声音、图形、图像于一体的信息表示方式, Delphi 提供了对多媒体信息处理方法, 使开发出的程序更具有感染力。

3.3.1 声音处理

在应用程序中加入声音效果, 可以起到警告, 引起注意, 增强气氛的效果。根据播放声音的来源不同, 有以下两种方式。

1. 播放系统的缺省声音

在程序执行过程中可以调用 MessageBeep 函数或 Beep 过程来播放 Windows 系统缺省的声音。

MessageBeep 函数的首部如下:

```
BOOL MessageBeep( UINT uType );
```

uType 的取值如下:

0xFFFFFFFF(扬声器的标准鸣叫);

MB_ICONASTERISK(特殊声音);

MB_ICONEXCLAMATION(感叹声);

MB_ICONHAND(处理声);

MB_ICONQUESTION(问题声);

MB_OK(默认声)。

如果函数调用成功, 则返回 True; 否则返回 False。

Beep 过程的定义如下:

```
procedure Beep;  
begin  
    MessageBeep(0);  
end;
```

其作用是使计算机发出一声标准的鸣叫。

2. 播放其他声音文件

播放系统默认声音之外的声音文件需要使用 PlaySound 函数。PlaySound 函数首部如下:

```
BOOL PlaySound( LPCSTR pszSound, HMODULE hmod, DWORD fdwSound );
```



其中,参数 `pszSound` 用于指定要播放的声音文件,其值为 `nil`,表示声音播放停止;参数 `hmod` 用于指定可执行文件的处理方式,通常为 0 值;参数 `fdwSound` 是声音的播放方式。如果函数调用成功,则返回 `True`;否则返回 `False`。例如:

```
PlaySound('c:\windows\media\chord.wav',0,snd_Async);
```

表示以异步方式播放声音文件 `chord.wav`;

```
PlaySound(nil,0,0);
```

表示停止声音文件的播放。

3.3.2 Animate(动画)组件

所说的动画是将多帧图像连续显示产生的效果。利用 `Animate` 组件可以产生动态的显示效果。`Animate` 组件位于组件板的 `Win32` 标签页上。

(1) Animate 组件的主要特性

- **Active 特性**

其值为 `True` 时,可以启动动画的显示。

- **CommonAVI 特性**

它用于确定动画效果,其取值有:

`AviCopyFind`(复制单个文件动画);

`AviCopyFinds`(复制多个文件动画);

`AviDeleteFind`(删除文件动画);

`AviEmptyRecycle`(清空回收站动画);

`AviFindCompute`(查找计算机动画);

`AviFindFile`(查找文件动画);

`AviFindFolder`(查找文件夹动画)。

- **FileName 特性**

它用于指定预播放的 AVI 文件名,若设置了此特性,则 `CommonAVI` 特性值自动变为 `aviNone`。

- **FrameCount 特性**

它指出图像序列中图像的总帧数。

- **StartFrame 和 StopFrame 特性**

它们分别用于指定动画的开始帧和结束帧。

(2) Animate 组件的主要方法

- **Play 方法**

它可以指定播放某段动画若干次。例如:

```
play(1, 5, 10);
```

将第一帧到第五帧的图像连续播放 10 次。

- **Reset 方法**

它关闭动画的显示,重新调入图像序列,显示第一帧图像。

- **Seek 方法**

用它设置预显示的图像帧号。例如:



```
seek(3);
```

如有动画显示操作,将从第3帧图像开始显示。

- Stop 方法

停止图像的播放。

3.3.3 MediaPlayer(多媒体播放)组件

MediaPlayer 组件可以处理和播放各种格式的多媒体文件,例如 AVI 文件、MIDI 文件和 WAV 文件。该组件位于组件板 System 标签页中。

1. MediaPlayer 组件的外观

MediaPlayer 组件的外观如图 3.4 所示。每个按钮对应一个特定的值和方法,当单击该按钮时,触发 OnClick 事件,并执行相应的事件处理程序。



图 3.4 MediaPlayer 组件的外观

各个按钮的功能依次如下:

- Play 按钮——值为 btPlay,用于媒体的播放;
- Pause 按钮——值为 btPause,用于暂停或恢复播放;
- Stop 按钮——值为 btStop,用于停止播放或录制;
- Next 按钮——值为 btNext,用于跳到下一轨道;
- Prev 按钮——值为 btPrev,用于跳到上一轨道;
- Step 按钮——值为 btStep,用于向前跳过几帧画面;
- Back 按钮——值为 btBack,用于向后跳过几帧画面;
- Record 按钮——值为 btRecord,用于开始录制;
- Eject 按钮——值为 btEject,用于弹出媒体。

2. MediaPlayer 组件的主要特性

- DeviceType 特性

它用于指定设备类型,默认值是 dtAutoSelect,表示可以根据媒体文件的扩展名自动识别设备类型。MediaPlayer 组件支持的设备类型有 dtAVIVideo、dtCDAudio、dtDAT、dtDigitalVideo、dtMMMovie、dtOverlay、dtScanner、dtSequencer、dtVCR、dtWaveAudio 等。

- FileName 特性

它指定用于播放的文件名。

- AutoOpen 特性

其值设置为 True 时,会自动打开 DeviceType 特性指定的设备。

- Notify 特性

它用于决定当一个多媒体控制方法(如 Play、Pause、Stop 等)完成后,是否触发 OnNo-



tify 事件表示操作已结束。操作结束时可以通过 Notify Value 特性值来查看控制命令的执行状况；nvSuccessful(成功)、nvSuperseded(控制命令被其他命令替代)、nvAborted(命令被终止)、nvFailure(失败)。

3. 主要方法

- Play、Pause、Stop、Next、Previous、Step、Back、StartRecording、Eject 等方法
这组方法与 MediaPlayer 组件的按钮一一对应；当按下某一按钮时，将调用相应的方法，控制媒体的播放情况。
- Open 方法
它打开多媒体设备，与将 AutoOpen 特性的值设置为 True 作用相同。
- Close 方法
它关闭已打开的多媒体设备，当应用程序结束时，自动调用 Close 方法。
- Rewind 方法
它使多媒体设备回到开始位置。

3.3.4 应用举例

设计一个简单的多媒体播放器。该播放器能够播放 MIDI、WAV 和 AVI 等格式的文件。

1. 界面的设计

窗体 Form1 的 Caption 特性为“简单的媒体播放器”。按图 3.5 所示的位置关系在窗体上放置各个组件。组件的作用及其特性设置如下：



图 3.5 简单的多媒体播放器的设计界面

1 个 Panel 组件——显示播放的视频文件信息，其 Caption 特性设置为空。

1 个 MediaPlayer 组件——用于播放各种格式的文件，其 Display 特性为 Panel1。

1 个 OpenFileDialog 组件——用于打开磁盘上的文件，可以通过设置其 Filter 特性确定打开文件的类别。这里设置成“多媒体文件 | *.WAV; *.AVI; *.MID”。

2 个 Button 组件——用于“打开文件”和“退出”播放器。

2. 程序的实现

“打开文件”按钮的 OnClick 事件的处理程序如下：

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    OpenFileDialog1.Execute;           //打开 OpenFileDialog 对话框
    Caption:= OpenFileDialog1.FileName; //使窗体的标题栏显示当前打开的文件名
```



```

with MediaPlayer1 do
begin
  FileName:= OpenFileDialog1.FileName;    //设置将要播放的文件
  Open;                                   //打开将要播放的文件
  displayrect:= rect(0,0,panel1.Width,Panel1.Height);
                                           //使播放的图像显示在 Panel 面板上
  Notify:= True;                         //当多媒体播放控制(如播放、停止、暂停等)结束后
                                           //触发 OnNotify 事件表示操作结束
end;
end;

```

“退出”按钮的 OnClick 事件的处理程序如下:

```

procedure TForm1.Button2Click(Sender: TObject);
begin
  Close;
end;

```



习题与思考题

3.1 按图 3.6 所示,输入两点的平面坐标,在屏幕上绘制出对应的直线。

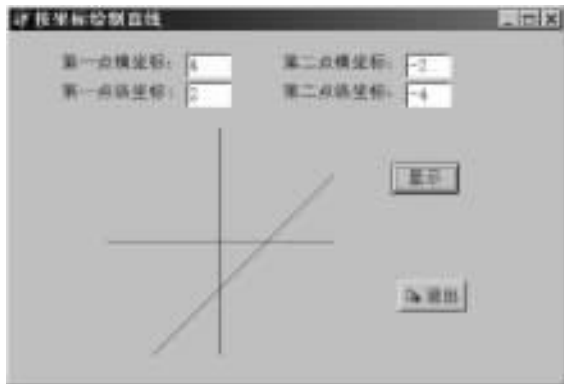


图 3.6 按坐标绘制直线运行界面

- 3.2 调试多媒体播放器程序,修改程序使其播放声音文件时,屏幕空白处显示进度情况。
- 3.3 设计一个动画播放程序,对一个存放多帧信息的 AVI 文件进行连续播放。



第 4 章

数据库程序设计

当今是信息时代,信息的处理离不开数据库。Delphi 具有强大的数据库处理功能,借助 Delphi 提供的数据库管理工具可以很容易建立基本的数据库。采用 Delphi 的数据库组件,使数据库编程同 Delphi 的其他应用程序设计一样直观方便。

在本章中主要介绍 Delphi 数据库系统的组成,利用 Delphi 内置的数据库引擎 BDE 和数据库桌面开发工具 Database Desktop 创建数据库的步骤,以及利用 Delphi 提供的数据库组件,实现数据库的查询、维护和报表的程序设计。

DP 4.1 Delphi 数据系统概述

数据库系统主要由三大部分组成:数据管理系统 DBMS,负责组织和管理数据信息;数据库应用程序,通过它可以访问数据库管理系统中的数据信息;数据库,是按照一定的结构组织在一起的相关数据的集合。

Delphi 数据库管理系统将 Object Pascal 语言与先进的关系型数据库 Paradox 完美的结合在一起。而且 Delphi 提供了各种数据库存取和控制组件,将数据管理的实现细节封装在其中,实现了数据库应用程序的开发真正面向对象。

Delphi 数据库系统的构成如图 4.1 所示。从图中可以看出 Delphi 的数据库应用程序要通过 BDE(数据库引擎)与本地或远程数据库交换数据。对于本地数据库,BDE 可以直接访

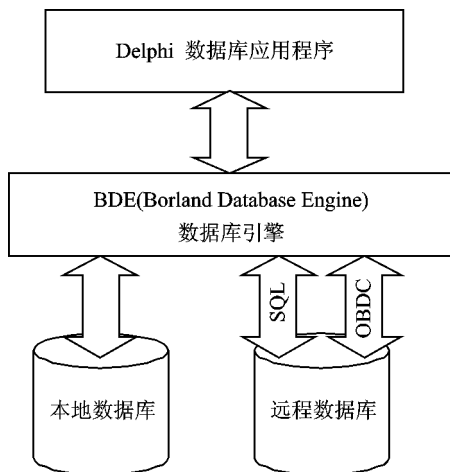


图 4.1 Delphi 数据库系统结构



问;对于远程数据库,则要通过 SQL(结构化查询语言)或 ODBC(开放数据库链接包)访问。本地数据库主要有 Paradox、dBASE、FoxPro、ASCII 和 Access 等,远程数据库是指存放在远程数据库服务器(例如 ORACLE、SyBase、Microsoft SQL Sever、Imformix、InterBase)上的数据库。此外,Delphi 可以识别其他各种符合 ODBC 标准的数据库。表 4.1 列出了 Delphi 可访问的数据源。

表 4.1 Delphi 可访问的数据源及其说明

数据源	说 明	扩展名
DBASE	每个表是一个独立的文件	DBF
Paradox	每个表是一个独立的文件	DB
ASCII	每个表是一个独立的文件	TXT
InterBase	Delphi 自带的数据库系统,一个数据库包括多个表,数据库通过 InterBase DBMS 建立	GDB
ORACLE、SyBase、Informix Microsoft SQL Server 等	网络数据库,一个数据库包括多个表,数据库通过各自的管理系统建立	

DP 4.2 创建数据库

Delphi 使用的数据库是关系数据库。在关系数据库中,信息被存放在二维表中,由行(记录)和列(字段)组成。每个记录描述一个对象的信息,字段描述对象的某一特性。例如,图书馆书库的结构如表 4.2 所列,第一行为字段名,其余各行为记录。每个记录有 6 个字段:书号、书名、作者、出版社、数量和摘要。下面以创建一个书库为例,说明如何创建数据库。

表 4.2 图书馆书库基本结构

书 号	书 名	作 者	出 版 社	数量	摘要
TP312-140	计算机公共基础	徐士良	清华大学出版社	10	
TP312-150	PASCAL 语言程序设计	唱江华	哈尔滨工程大学出版社	10	
TP312-166	C 语言程序设计	谭浩强	高等教育出版社	10	
⋮	⋮	⋮	⋮	⋮	

4.2.1 定义数据库别名

数据库通常以两种方式存储:将整个数据库以一个单独的文件存储;分别将每个数据表、索引及其他数据库元素存储在各自的文件中。Delphi 以数据库别名的方式引用这两种不同格式的数据库。用户可以通过数据库别名直接访问数据库,无须了解其存储方式。

1. 建立文件夹

在定义数据库别名前,一般要建立存放数据库的文件夹。例如,准备建立数据库别名为 Book,则可以先建立如下两个文件夹 C:\Book 和 C:\Book\Data。C:\Book 用来保存应用程序文件,C:\Book\Data 用来保存数据文件。



2. 利用 BDE Administrator(BDE 管理器)定义数据库别名

BDE Administrator 是一个设置和管理 BDE 的工具,利用它可以管理 BDE 中的别名和驱动程序。进入 BDE Administrator 之前,应关闭所有的 BDE 应用程序。用户可以通过“开始”菜单在 Delphi 图标组中打开 BDE Administrator,如图 4.2 所示。

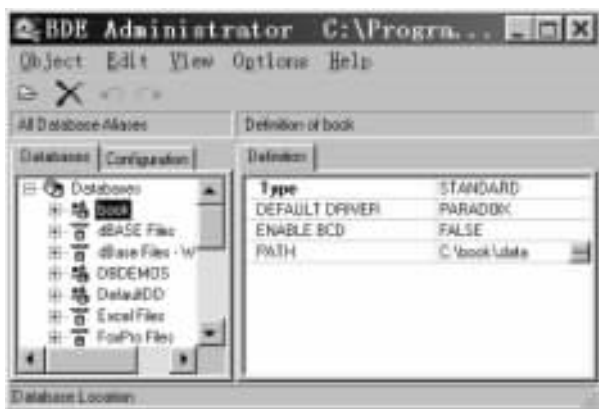


图 4.2 BDE Administrator 窗口

首先选中 Databases 标签页,单击右键,在弹出菜单中选择 New 菜单项,出现 New Database Alias 对话框;再在 Database Driver Name 组合框中选择数据库驱动程序,在此选择 STANDARD。单击 OK 按钮后,在 Databases 标签页中出现一个新增项 STANDARD1,可以将其直接修改为数据库别名,例如设置为 Book;然后在右侧的 Definition 标签页中设置各项参数,例如将路径设置为 C:\Book\Data。

4.2.2 利用 DataBase Desktop (数据库桌面)建立数据表结构

DataBase Desktop 是一个能够实现创建、修改以及查询数据库的工具。在 Delphi 的集成环境中,选择 Tools 菜单中的 DataBase Desktop 即可打开如图 4.3 所示的 DataBase Desktop 窗口。

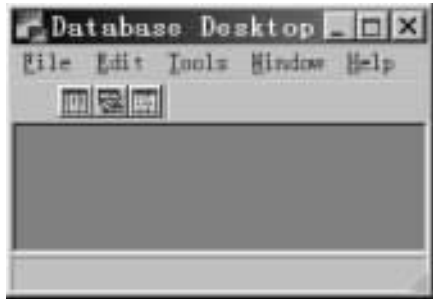


图 4.3 数据库桌面(Database Desktop)窗口

1. 建立基本结构

选择 File | New | Table... 命令,将打开 Create Table 对话框;再选择 Paradox7,单击 OK 按钮,会出现如图 4.4 所示的 Create Paradox 7 Table 窗口。在此窗口中可以定义数据库结构。用户可以依次输入各个字段的字段名(field name)、类型(type)、长度(size)、是否为关键字(key)。在输入字段类型值时,可以单击鼠标右键,在弹出菜单中选择。常用的类型有以下几种:



Alpha——字符串型,长度在 0~255 个字符;

Short——短整型,不必指定长度;

Date——日期型,不必指定长度;

Logical——逻辑型,不必指定长度;

Memo——备注型,长度单位在 1~240MB



图 4.4 创建数据库结构窗口

2. 建立索引的方法

在建立了数据表的基本结构后,还应根据需要确定索引关键字,目的是方便对数据表进行分类排序以及快速查询等操作。索引分为主索引和次索引两种。

(1) 建立主索引

主索引关键字所对应的数据必须是唯一的,每个数据表只能有一个主索引。若确定某一字段为主关键字,可以在 Key 列中按下空格键,出现‘*’,说明设置成功,主索引也就建立成功。例如,图 4.4 中“书号”字段为主关键字。

(2) 建立次索引

次索引关键字所对应的数据不必是唯一的,而且每个数据表可能有多个次索引。在图 4.4 所示的窗口中,选择 Table Properties 下拉列表框的 Secondary Indexes 选项,然后单击 Define 按钮进入次索引定义窗口。先在左边列表框中选择次关键字段名(可以选择多个),然后单击中间的 ➡ 按钮,将其移向右边的列表框;完成后,单击 OK 按钮,这时,将弹出 Save Index As 对话框,用户在此输入次索引文件名;然后单击 OK 按钮,即完成了次索引的建立。

(3) 设定有效性检查

借助 DataBase Desktop 可以为各个字段设定有效性检查。在图 4.4 的窗口中,选择 Table Properties 下拉列表框的 Validity Checks 选项,可以为当前字段设置如下的检查项目:

- Required Field



当其对应的复选框出现√,表示该字段在输入时不可缺少。

- Minimum Value
该字段在输入时的最小值。
- Maximum Value
该字段在输入时的最大值。
- Default Value
该字段在输入时的默认值。
- Picture
该字段的输入格式。

当所有字段信息设置完毕,单击 Save As... 按钮,在 Alias 列表框中选择数据库别名 Book,输入文件名 BookLib 后,单击“保存”,文件将自动保存在 C:\Book\Data 文件夹下。

4.2.3 编辑数据表

在 DataBase Desktop 窗口中,单击 File|Open|Table 菜单项,在 Alias 列表框中选择数据库别名 Book,在上面的文件列表中选择要编辑的数据表名 BookLib.db,单击 OK 后,出现如图 4.5 所示的窗口。



图 4.5 数据表编辑窗口

单击 Table|Edit Data 菜单项,进入编辑状态,这时就可以依次输入各个记录内容。录入完毕,关闭此窗口,会自动保存录入的信息。若想修改数据表的内容,也可按以上步骤进行。

DP 4.3 利用数据库组件实现数据库编程

4.3.1 数据库组件介绍

在 Delphi 的数据库管理中,引入了面向对象的概念,将数据管理的细节封装在对象(组件)中,通过对象(组件)来管理数据。也就是说,可以借助数据库组件访问上一节中建立的数据表。在 Delphi 中有关数据库编程的组件有两类:数据库存取组件(位于 Data Access 标签页)、数据控制组件(位于 Data Control 标签页)。Delphi 数据库组件之间的关系如图 4.6 所示。

Delphi 的应用程序要通过 BDE 访问数据库,数据库存取组件就是实现与 BDE 相联系的

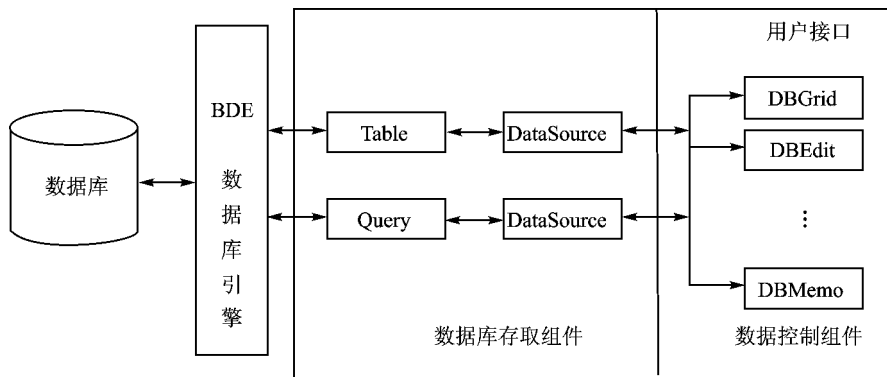


图 4.6 Delphi 数据库组件之间的关系

对象,在程序运行时,它们是看不到的,只是在幕后管理着数据的存取。常用的数据库存取组件有 Table、Query、DataSource 等。数据控制组件提供了数据库访问的用户界面,主要用于显示和编辑数据表中的数据。常用的数据控制组件有 DBGrid、DBEdit、DBText、DBMemo 等。从图中还可以看出,DataSource 是连接数据库存取组件与数据控制组件之间的媒介。

下面依次介绍各组件的主要特性和方法。

1. Table(数据表)组件

Table 属于 TTable 类,用于连接一个数据表,并对数据表的各种状态进行控制。

(1) 主要特性

- DataBaseName 特性——指定数据表所在的数据库名,通常为一个数据库别名。
- TableName 特性——设置数据表名,该数据表应确实在 DataBaseName 所指定的数据库中。
- Active 特性——其值为 True,数据表处于打开状态;为 False,数据表处于关闭状态。用户要借助 Table 组件对一个数据表操作,必须依次设置以上三个特性。
- ReadOnly 特性:其值为 True 时,数据表处于只读方式。
- Exclusive 特性:其值为 True 时,则数据表为独占方式,其他程序不能使用。
- CanModify 特性:用于记录是否可以改变数据表,当以只读方式打开数据表时,该特性返回值为 False。
- Modified 特性:用于记录数据表是否被修改过,其值为 True 时,表示被修改过。
- Bof 特性:其值为 True 时,表示当前指针指向首记录。
- Eof 特性:其值为 True 时,表示当前指针指向末记录。
- FieldCount 特性:返回数据表中的字段数(列数)。
- RecordCount 特性:返回数据表中的记录总数。

(2) 主要方法

① 数据表操作方法

- Open——打开数据库。
- Close——关闭数据库。



② 访问字段方法。

- FieldByName——取得指定字段的信息,返回值为一个字段对象。

③ 指针移动方法

- First——移至首记录。
- Last——移至末记录。
- Prior——向前移动一个记录。
- Next——向后移动一个记录。
- MoveBy——将当前指针移动一个相对位置,例如:

MoveBy(-2);

表示向前移动 2 个记录。

④ 记录操作方法

- AppendRecord——在数据表后追加一个记录,记录的各个字段值依次列在参数中用逗号隔开。
- Append——在数据表的最后添加一个空记录,并将当前指针移到该位置。
- Delete——删除当前指针位置处的记录,并将指针指向下一个记录。
- Edit——使数据表处于编辑状态。
- Insert——在当前指针位置处插入一个空记录,并将指针指向该记录。
- Post——将对数据表的修改写入数据库中。
- Cancel——取消对数据表的修改。

2. Query 组件

与 Table 组件很相似,但它可以通过 SQL 语句(有关 SQL 语句的用法,读者可以参考相关的书籍)访问数据表,而且可以同时访问多个数据表。Query 组件的大部分特性与 Table 组件相似,但 Query 组件没有 TableName 特性,而具有一个非常重要的 SQL 特性。

● SQL 特性

主要用于设置查询数据的条件,由若干行 SQL 语句组成,为 Tstring 类型。当单击该特性值右边的“...”时,会弹出如图 4.7 所示的对话框,可以在编辑框中输入要执行的 SQL 语句,以待程序运行时执行;当然也可以在应用程序中通过程序代码加入 SQL 语句。

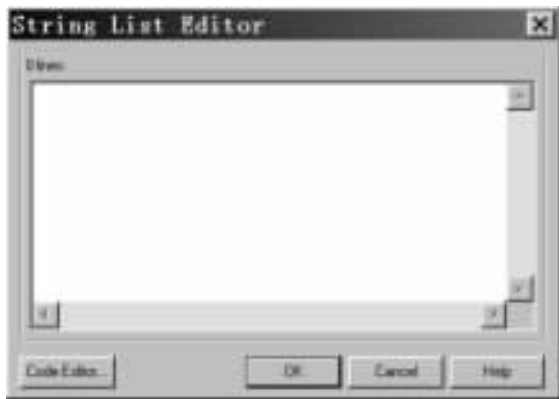


图 4.7 Query 组件的 SQL 特性编辑框



Query 组件还具有四个重要的方法。

- Open 方法

该方法用于打开与 Query 组件相联系的数据集,只能执行 Select 命令。

- ExecSQL 方法

该方法用于执行 Query 组件的 SQL 特性中设置的各种 SQL 语句。

- Prepare 方法

该方法实现在执行 SQL 查询前,先把 SQL 语句送往 BDE 数据库引擎,并对其进行语法分析和优化,从而加快查询速度。

- Close 方法

该方法用于关闭与 Query 组件相联系的数据集。通常在向 Query 组件中添加 SQL 语句前,要先调用此方法关闭数据集。

3. DataSource(数据源)组件

该组件是联系 Table(或 Query)组件与数据控制组件的纽带,用户应在 DataSet 特性中指明与该数据源相联系的 Table(或 Query)组件名。AutoEdit 特性用于控制数据源的状态,如果值为 True,表示与之联系的数据控制组件中的数据变化时,数据表会自动随其变化;值为 False 时,数据表不产生变化,可以起到保护数据库的作用。

4. DBGrid 组件

以表格的形式显示数据表中的记录信息。用户应在 DataSource 特性中指出与该组件相联系的数据源。利用 Fields Editor 可以对数据库表中字段的显示格式、显示顺序、是否显示等进行控制。

5. DBText 组件、DBEdit 组件和 DBMemo 组件

- DBText 组件

该组件用于显示数据表中当前记录的固定不变的文字,用户不能对其中的数据进行修改。

- DBEdit 组件

该组件用于显示和编辑数据表中当前记录指定的字符串特性的字段的值。

- DBMemo 组件

该组件用于显示和编辑数据表中的某一 Memo 类型的字段,可以用来显示数据库表中当前记录中的 BLOB 型字段。

用户在使用这三个组件时,除了要指明 DataSource 特性外,还要在 DataField 特性中指明与该组件相联系的字段名。

6. DBNavigator 组件

使用该部件可以向前、向后移动记录指针,可以使用该部件对单条记录进行编辑,还可以用它来插入、删除记录以及刷新显示和取消前一次的操作。



7. DBCheckBox 组件

它用于显示和编辑数据表中某一个布尔型字段的取值。

8. TDBListBox 组件

它是浏览数据库中数据的列表框,可以用它来显示数据库表中一个字段的值。

9. DBComboBox 组件

它是浏览数据库中数据的组合框,可以用来显示数据库表中一个字段的值。

10. DBRadioGroup 组件

它是浏览数据库表中数据的单选钮,用一组单选钮可以确定显示数据库表中哪一个字段。

11. DBImage 组件

它是浏览数据库中数据的图像框,可以用于显示、复制、粘贴数据库表中图像类型的字段。

12. DBLookupList 组件

它是浏览数据库表中数据的列表框,可以实现在一个数据库表中显示另一个数据库表中一个指定的字段值。

13. DBLookupCombo 组件

它是浏览数据库表中数据的组合框,可以实现在一个数据库表中显示另一个数据库表中一个指定的字段值。

4.3.2 数据库维护程序

数据库的维护是指对数据库中的记录进行增加、删除和修改等操作。图 4.8 是书库管理



图 4.8 书库管理窗体运行界面



程序的运行界面。这就是一个数据库维护程序。在窗口左边的列表框内集中显示书库中的记录信息,用户可以通过移动滚动箭头查看未显示的部分;窗口右边显示的是当前记录的各项信息。当单击“新书录入”按钮后,用户可以进行新书信息的录入,完毕按“确认”按钮确认。若想取消录入,可以按“取消”按钮。当单击“修改书目”按钮后,用户可以对所有记录信息进行修改,结束后同样按“确认”按钮确认修改,或按“取消”按钮取消修改。当单击“注销旧书”按钮,会弹出对话框,确认后,将当前的记录删除。单击“退出”,退出书库管理程序。

1. 界面设计

窗体上放置的各组件的功能及特性设置如下:

(1) Table1——数据表组件,其 DataBaseName 特性为 Book(数据库别名),TableName 特性为 BookLib.db(与该组件联系的数据表名),为了能对数据表操作,可以在此或程序代码中将其 Active 特性置为 True。

(2) DataSource1——数据源组件,其 DataSet 特性为 Table1(使其与 Table1 相联系),AutoEdit 特性设置为 False(防止与之相联系的数据库受破坏)。

(3) DBGrid1——数据列表显示组件,其 DataSource 特性为 DataSource1,这样该组件就可以显示与 DataSource1 相联系的数据库。

(4) DBEdit1、DBEdit2、DBEdit3、DBEdit4、DBEdit5——数据编辑组件,其 DataSource 特性均为 DataSource1,DataField 特性分别为“书号”、“书名”、“作者”、“出版社”和“数量”,用户可以借助这类组件显示或编辑当前记录中与之相联系的字段值。

(5) DBMemo——备注编辑组件,其 DataSource 特性为 DataSource1,DataField 特性为“摘要”,用户可以借助该组件显示或编辑当前记录中的摘要字段的值。

(6) Label1、Label2、Label3、Label4、Label5、Label6——标签组件,用于标记各个字段名,其 Caption 特性依次为“书号”、“书名”、“作者”、“出版社”、“数量”和“摘要”。

(7) Button1——按钮组件,Caption 特性为“新书录入”。

Button2——按钮组件,Caption 特性为“修改书目”。

Button3——按钮组件,Caption 特性为“注销旧书”。

Button4——按钮组件,Caption 特性为“确认”。

Button5——按钮组件,Caption 特性为“取消”。

Button6——按钮组件,Caption 特性为“退出”。

2. 程序实现

(1) 新书录入功能

```
Table1. Last;
```

```
Table1. Append;
```

(2) 修改书目功能

```
Table1. Edit;
```

(3) 注销旧书功能

```
N:= Application. MessageBox('确实要删除吗?', '确认框', mb_OKCANCEL);
```

```
if N=idOK then
```



```
Table1.Delete;
```

其中 N 为整型变量。

(4) 确认功能

```
Table1.Post;
```

(5) 取消功能

```
Table1.Cancel;
```

(6) 退出功能

```
Close;
```

4.3.3 数据库查询程序

数据库查询是指在数据库中查找符合某种条件的记录。图 4.9 所示即是一个数据库查询程序。用户可以在各个输入框中输入查找条件,并选择查询方式,单击“开始查询”按钮即可进行相应的查询。其中,精确查询是指查找与用户输入的条件完全匹配的书目,若找到则将指针拨到该记录处,否则提示“没有您查找的书目”;模糊查询是指查找与用户输入的条件相近的书目,若找到,则将指针拨到该记录处,并提示“是这本书吗?”如果用户“确定”,则停止查询,否则继续查找,直到没有符合条件的记录为止。下面先介绍数据库查询的一般方法,再解释程序的实现方法。数据库查询方法通常有两种:顺序查找和快速查找。



图 4.9 书目查询程序运行界面

1. 顺序查找

顺序查找是指从数据表中逐条取出记录,依次判断是否符合查找条件的方法。该方法比较简单,但如果数据表中记录数过多会影响速度,所以该方法适合查找记录数不是很多的数据表。

在判断查找条件时,经常要用到以下几个有关字符串比较、匹配的函数:

- function CompareStr(const S1,S2:string):Integer;
比较字符串 S1、S2 的大小,区分大小写。
- function CompareText(const S1,S2:string):Integer;



比较字符串 S1、S2 的大小,不区分大小写。

以上两个函数的返回值情况:S1=S2 时,返回值为 0;

S1>S2 时,返回值大于 0;

S1<S2 时,返回值小于 0。

- function Pos (Substr: string; S: string): Integer;

若字符串 Substr 是 S 的子串,则返回该子串在 S 中的位置;否则返回 0。

2. 快速查找

快速查找能够实现在数据表中快速定位到某一记录位置处。使用该方法的前提是数据表中已经为查询关键字建立相应的索引。

若使用次关键字查询时,要用如下语句先指明次关键字段名或次索引文件名:

数据表名.IndexFieldNames:= 次关键字段名;

或

数据表名.IndexName:= 次索引文件名;

实现快速查询的方法:

- function FindKey (const KeyValues: array of const): Boolean;

函数 FindKey 用于精确查询。当在数据表查找到关键字段值与参数 KeyValues 完全相同的记录时,将指针指向该记录,并返回函数值 True。

- procedure FindNearest (const KeyValues: array of const);

过程 FindNearest 用于模糊查询。当在数据表查找到第一个关键字段值与参数 KeyValues 相同或相近的记录时,将指针指向该记录。

3. 用户界面

窗体中放置的各个组件的功能和特性如下:

(1) Table1——数据表组件,用于表示书库,其 DataBaseName 特性为 book,TableName 特性为 BookLib.db,Active 特性设置为 True。

(2) DataSource1——数据源组件,其 DataSet 特性为 Table1,AutoEdit 特性设置为 False。

(3) DBGrid1——数据列表显示组件,其 DataSource 特性为 DataSource1。

(4) RadioGroup1——单选按钮组组件,用于选择查询方式,其 Caption 特性为“查询方式”,Items 特性为“精确”、“模糊”。

(5) Label1、Label2、Label3、Label4——标签组件,用于标识查询字段名,其 Caption 特性依次为“书号”、“书名”、“作者”、“出版社”。

(6) Edit1、Edit2、Edit3、Edit4——编辑框组件,用于接收用户输入的查询条件,其 Caption 特性均为空。

(7) Button1——按钮组件,Caption 特性为“开始查询”。

(8) Button2——按钮组件,Caption 特性为“退出”。



4. 程序实现

单击“开始查询”按钮后,应完成所有查询操作。我们设立 4 个布尔变量,标识各个查询字段是否匹配。如果用户未输入查询数据时,则默认匹配。如果用户输入了查询数据,在“精确”查询方式下,只有判断各字段完全相同才认为匹配;在“模糊”查询方式下,当输入数据是各个字段值的子串时就认为匹配。程序代码如下:

```
procedure TForm3.Button1Click(Sender: TObject);
var
    No1,No2,Name1,Name2,Author1,Author2,Publisher1,Publisher2:String;
    B1,B2,B3,B4:Boolean;
begin
    No1:=Edit1.Text;
    Name1:=Edit2.Text;
    Author1:=Edit3.Text;
    Publisher1:=Edit4.Text;
    Table1.First;
    while not Table1.Eof do
    begin
        No2:=Table1.FieldByName('书号').asString;
        Name2:=Table1.FieldByName('书名').asString;
        Author2:=Table1.FieldByName('作者').asString;
        Publisher2:=Table1.FieldByName('出版社').asString;
        if No1='' then
            B1:=True
        else if RadioGroup1.ItemIndex=0 then
            B1:=CompareText(No1,No2)=0
        else
            B1:=Pos(No1,No2)>0;
        if Name1='' then
            B2:=True
        else if RadioGroup1.ItemIndex=0 then
            B2:=CompareText(Name1,Name2)=0
        else
            B2:=Pos(Name1,Name2)>0;
        if Author1='' then
            B3:=True
        else if RadioGroup1.ItemIndex=0 then
            B3:=CompareText(Author1,Author2)=0
        else
            B3:=Pos(Author1,Author2)>0;
        if Publisher1='' then
            B4:=True
```



```

else if RadioGroup1.ItemIndex=0 then
    B4:=CompareText(Publisher1,Publisher2)=0
else
    B4:=Pos(Publisher1,Publisher2)>0;
if B1 and B2 and B3 and B4 then
begin
    if RadioGroup1.ItemIndex=0 then
        Break
    else
        if Application.MessageBox('是这本书吗?',确认框,mb_okcancel)=idok then
            Break;
        end;
    Table1.Next;
end;
if Table1.Eof then
    ShowMessage('没有您所查找的书目。');
end;

```

4.3.4 Query 组件的 SQL 查询方法

Query 组件提供了在 Delphi 中使用 SQL 语句的支持,而 SQL 语句具有通用易学、查询方便快捷的特点。

利用 Query 组件实现 SQL 查询的具体步骤如下:

- (1) 设置 DatabaseName 特性值。可以是 BDE 别名,或者是数据库的目录名。
- (2) 设置 SQL 特性。可以通过两种方法:一是在对象观察器的 SQL 特性编辑器中输入 SQL 语句;二是在应用程序通过 Query 组件 SQL 特性的 ADD 方法添加 SQL 语句。
- (3) 调用 Open 或 ExecSQL 方法执行 SQL 语句。

下面以实现对前面所建的书库 BookLib.db 的查询为例,说明用 Query 组件实现查询的方法。查询界面如图 4.10 所示。当用户输入查询条件(书名、书号、作者、出版社的全部或部



图 4.10 用 SQL 语句实现书库查询的运行界面



分信息),并且在“要显示的字段”框中,输入希望显示的字段信息(中间用“,”间隔的字段名)后,单击“查询”按钮,将在窗口下部显示查询结果。

1. 界面设计

(1) 新建一个项目,在窗体上放置 1 个 Query 组件、1 个 DataSource 组件、1 个 DBGrid 组件。其中 DataSource 组件的 DataSet 特性设置为 Query1,DBGrid 组件的 DataSource 特性设置为 DataSource1。

(2) 在窗体上按图所示放置 5 个 Label 组件、5 个 Edit 组件、2 个 Button 组件,分别用于显示提示信息、接受录入信息、实现“查询”及“退出”操作。

2. 程序的实现

“查询”按钮的 OnClick 事件处理程序主要完成按用户的输入生成 SQL 语句,进行查询,具体代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
var
    Str,S,T:string;
    A:integer;
begin
    S:= '';           // S 变量用于生成查询的条件,即用 and 连接的各个查询条件
    if Edit1.Text <> '' then
        S:= ' 书号=' + Edit1.Text + ' ';
        //若“书号”编辑框中有内容,则将“书号=编辑框中的内容”连接到 S 中
    if (S<> '') and (Edit2.Text <> '') then
        S:= S + ' and 书名=' + Edit2.Text + ' ';
        //若 S 变量和“书名”编辑框均不为空,则将“and 书名=编辑框中的内容”连接到 S 中
    else if Edit2.Text <> '' then
        S:= ' 书名=' + Edit2.Text + ' ';
        //若 S 变量为空,“书名”编辑框不为空,则将“书名=编辑框中的内容”连接到 S 中
    if (S<> '') and (Edit3.Text <> '') then
        S:= S + ' and 作者=' + Edit3.Text + ' ';
        //若 S 变量和“作者”编辑框均不为空,则将“and 作者=编辑框中的内容”连接到 S 中
    else if Edit3.Text <> '' then
        S:= ' 作者=' + Edit3.Text + ' ';
        //若 S 变量为空,“作者”编辑框不为空,则将“作者=编辑框中的内容”连接到 S 中
    if (S<> '') and (Edit4.Text <> '') then
        S:= S + ' and 出版社=' + Edit4.Text + ' ';
        //若 S 变量和“出版社”编辑框均不为空,则将“and 出版社=编辑框中的内容”连接到 S 中
    else if Edit4.Text <> '' then
        S:= ' 出版社=' + Edit4.Text + ' ';
        //若 S 变量为空,“出版社”编辑框均不为空,则将“出版社=编辑框中的内容”连接到 S 中
    T:= Edit5.Text;
```



```

//T 变量用于存放要显示的字段列表,各个字段名中间用“,”间隔
A:=Pos(',',t);
while A<>0 do
begin
Delete(T,A,2);
Insert(',',',T,A);
A:=Pos(',',A); //',指的是全角的逗号
end;
//以上程序段用于将全角的逗号转换成半角的逗号
Query1.Close;
//首先关闭与组件 Query1 相联系的数据集
Query1.SQL.Clear;
//清空 Query1 组件的 SQL 语句
Str:='select '+T+' from BookLib.db where '+S;
//生成 SQL 查询语句,从 BookLib.db 数据表中按 S 变量中的条件显示变量 T 所指定的字段值
Query1.SQL.Add(Str);
//将生成 SQL 查询语句添加到 Query1 组件的 SQL 特性中
Query1.Open;
//打开指定的数据表 BookLib.db,执行 SQL 查询语句
end;

```

DP 4.4 图书馆管理系统的实现

图书馆管理系统主要包括以下几个子功能:借书管理,还书管理,书库管理,查询书目,借阅证管理。从以上功能可以分析出,该系统主要涉及到两个数据库:书库(包括书号、书名、作者、出版社、数量、摘要),学生库(包括借阅证号、姓名、密码、班级、借书信息)。下面,借助 Delphi 的数据库处理功能逐步实现图书馆管理系统的各项功能。其主界面如图 4.11 所示。

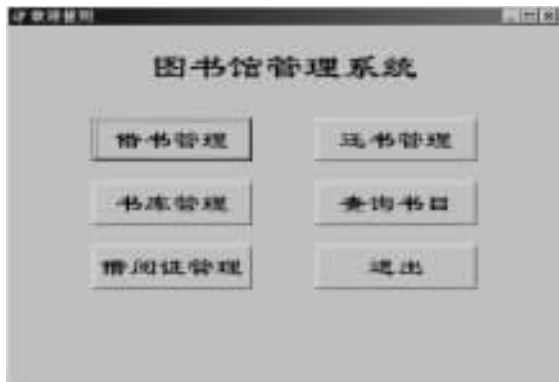


图 4.11 图书馆管理系统主窗口



4.4.1 系统主要功能

图书馆管理系统主要完成以下几项功能：

- (1) 书库管理。实现新书入库,注销旧书以及修改书目。
- (2) 查询书目。按用户的要求(书号、书名、作者以及出版社等)在书库中查找书目。
- (3) 借阅证管理。实现发证及注销证的操作。
- (4) 借书管理。当校验学生借阅证号、密码符合后,在学生有空卡的情况下,将书目信息存在学生卡上,学生卡数减 1,并将书库中该书目数量减 1。
- (5) 还书管理。当校验学生借阅证号、密码符合后,将学生卡上的书目删除,学生卡数加 1,并将书库中该书目数量加 1。

4.4.2 数据库结构

在该系统中除了用到前面介绍的书库外,还用到学生库(存放学生信息)、借阅库(存放学生借书信息)、空证库(存放注销的借阅证号)。各个数据库结构如下:

(1) 书库(Book. db)

书号	Alpha	10	主关键字
书名	Alpha	50	
作者	Alpha	20	
出版社	Alpha	30	
数量	Short		
摘要	Memo		

(2) 学生库(StudentLib. db)

证号	Alpha	5	主关键字
密码	Alpha	5	
姓名	Alpha	20	
班级	Alpha	10	
卡数	Short		

(3) 借阅库(BookID. db)

卡号	Alpha	16	主关键字
----	-------	----	------

(卡号在借书过程中形成,由证号和书号中间加 * 连接而成,故此取 16 位)

书号	Alpha	10
书名	Alpha	50
借期	Date	

(4) 空证库(BlankID. db)

证号	Alpha	5	主关键字
----	-------	---	------

4.4.3 子功能的实现

前面已经介绍了书库管理程序和查询书目程序的实现方法。下面实现其余功能。



1. 借书管理

为了安全起见,在借书过程中,应加密码校验程序。密码校验界面如图 4.12 所示。



图 4.12 密码校验的运行界面

密码校验程序用于检查证号和密码是否相符。若不符,用户按下“确定”按钮后,应提示“密码错误”,请求用户重新输入;若相符,则检查是否有空卡(每个学生有 5 张卡,允许借 5 本书)。若有空卡,应打开借书窗口,并初始化该窗口;否则,提示用户:“无空卡,不能借书”。如果想验证证号与密码是否相符,应该将用户输入的值与学生库中的值比较,因此要打开学生库,所以,应该在“密码校验”窗体上放置一个 Table 组件,与 StudentLib.db 建立联系。密码校验窗体中的“确定”按钮的 OnClick 事件处理代码如下:

```
Table1.Active:=True;
Table1.FindKey([Edit1.Text]); {在学生库中找到用户输入的证号所在的记录}
if (Table1.FieldByName('证号').asString=Edit1.Text)
    and (Table1.FieldByName('密码').asString=Edit2.Text) then
//判断密码校验相符
begin
    N:=Table1.FieldByName('卡数').asInteger;
    if N>0 then
        //还有空余卡,可以借书
        begin
            Form6:=TForm6.Create(Self); //创建借书窗体
            Form6.Show;
            //初始化借书窗体,显示学生的相关信息(从学生库得到)
            Form6.Edit1.Text:=Edit1.Text;
            Form6.Edit2.Text:= Table1.FieldByName('姓名').asString;
            Form6.Edit3.Text:= Table1.FieldByName('班级').asString;
            Form6.Edit4.Text:= Table1.FieldByName('卡数').asString;
            //打开借阅库,准备装入学生的借书信息:卡号、书号、书名、借期
            Form6.Table1.Active:=True;
            Str:=Edit1.Text+'*';
            Form6.Table1.FindNearest([Str]);
            Form6.Table1.Insert;
```




```

Str:=Edit1.Text+'*'+Table1.FieldByName('书号').asString;
Table1.FieldByName('卡号').asString:=Str;
Table1.Post;
Table2.Active:=True;           //打开借阅库,将所借书目的数量减1
Table2.FindKey([dbEdit2.Text]);
Table2.Edit;
N:=Table2.FieldByName('数量').asInteger;
Table2.FieldByName('数量').asString:=IntToStr(N-1);
Table2.Post;
Table3.Active:=True;           //打开学生库,将学生卡数减1
Table3.FindKey([Edit1.Text]);
N:=Table3.FieldByName('卡数').asInteger;
Table3.Edit;
Table3.FieldByName('卡数').asString:=IntToStr(N-1);
Table3.Post;
Edit4.Text:=IntToStr(N-1);

```

“下一本”按钮的 OnClick 事件的处理代码如下：

```

N:=StrToInt(Edit4.Text);
if N>0 then
begin
    Table1.Active:=True;
    Str:=Edit1.Text+'*';
    Table1.FindNearest([str]);
    Table1.Insert;
    Table1.FieldByName('借期').asString:=DateToStr(Date);
end
else
begin
    ShowMessage('对不起！您没有空卡了');
    Close;
end;

```

2. 还书管理

还书管理同样要有密码校验过程,其界面同图 4.12,不同之处是在单击“确定”按钮后的事件处理中,当密码校验正确后的处理过程。其代码如下：

```

N:=Table1.FieldByName('卡数').asInteger;
if n<5 then
//卡数<5,说明有未还的书
begin
//打开还书窗口,并进行初始化,找到该同学的卡号位置,前5位是证号加*
Form8:=TForm8.Create(Self);

```



```
Form8.Show;  
Form8.Edit1.Text:=Edit1.Text;  
Form8.Edit2.Text:= Table1.FieldByName(姓名).asString;  
Form8.Edit3.Text:= Table1.FieldByName(班级).asString;  
Form8.Edit4.Text:= IntToStr(5-N);  
Form8.Table1.Active:= True;  
Form8.Table1.FindNearest([Edit1.Text+'*']);  
end  
else  
//无未还的书  
begin  
    ShowMessage(您的书均已还清。);  
    Form8.Close;  
end;  
Close;
```

还书管理程序的界面如图 4.14 所示,还书过程要完成与借书过程相反的操作,因此在还



图 4.14 还书管理程序的运行界面

书窗体上也应放置 3 个 Table 组件,分别对应借阅库、学生库和书库,1 个 DataSource 组件和 1 个 DBGrid 组件,对应借阅库。

“确定”按钮的 OnClick 事件处理代码如下:

```
Table2.Active:= True;           //打开书库,将所还书的数量加 1  
Table2.FindKey([dbEdit2.Text]);  
Table2.Edit;  
N:= Table2.FieldByName(数量).asInteger;  
Table2.FieldByName(数量).asString:= IntToStr(N+1);  
Table2.Post;  
Table3.Active:= True;           //打开学生库,将学生的卡数增加 1  
Str:= dbEdit1.Text;
```



```

Delete(Str, Pos(' * ', Str), 11);
Table3. FindKey([Str]);
N:= Table3. FieldByName('卡数'). asInteger;
Table3. Edit;
Table3. FieldByName('卡数'). asString:= IntToStr(N+1);
Table3. Post;
Edit4. Text:= IntToStr(4-N); //在还书窗体中显示学生已借书的数量
Table1. Delete;
if 4-N=0 then
    ShowMessage('您的书均以还清。');

```

“下一本”按钮的 OnClick 事件处理代码如下：

```
Table1. FindNearest([Edit1. Text+'*']); //将指针拨到学生的卡号位置处
```

3. 借阅证管理

借阅证管理与书库管理功能相近,程序实现方法基本相同。其中“发证”是向学生库中增加记录,“修改”指修改学生库中的记录,“注销”是从学生库中删除记录,其运行界面如图 4.15 所示。为了保证证号的有效管理,在发证时,证号从空证库中取得,注销证时将证号存入空证库。因此在借阅证管理窗体中还要放置一个数据表组件对应空证库。



图 4.15 借阅证管理程序的运行界面

“发证”按钮的 OnClick 事件的处理代码如下：

```

Table1. Last; //学生库的指针拨到最后,追加一条空记录
Table1. Append;
Table2. First; //从空证库取出一个证号填到学生库的当前记录中
Table1. FieldByName('证号'). asString:= Table2. FieldByName('证号'). asString;
Table2. Delete;
DbEdit5. Text:= '5';

```

“注销”按钮的 OnClick 事件处理代码如下：

```
n:= StrToInt(DBEdit5. Text);
```



```

if N=5 then
//空卡数为 5,表示书已还清,可以注销证
begin
    Table2. Last;
    Table2. Append;
    //将证号存入空证库
    Table2. FieldByName('证号'). asString:=Table1. FieldByName('证号'). asString;
    if Application. MessageBox('确实要注销吗?', '确认框', mb_OKCANCEL)=idOK
    then
        Table1. Delete;
    end
else
    ShowMessage('您还有'+IntToStr(5-N)+'本书未还,不能注销!');

```

4.4.4 窗体之间的联系方法

在该系统设计中用到很多窗体,各个窗体之间是靠按钮组件的 OnClick 事件来联系的。比如,当用户在按下“书库管理”按钮后,应该打开“书库管理”窗口,同时打开书库 BookLib. db,所以该按钮的 OnClick 事件处理代码如下:

```

Form2:=TForm2. Create(Self);
Form2. Show;
Form2. Table1. Active:=True;    //打开书库

```

再如,“书目查询”按钮的 OnClick 事件处理代码如下:

```

Form3:=TForm3. Create(Self);    //打开“书目查询”窗体
Form3. Show;
with Form3 do                    //初始化“书目查询”窗体
begin
    Edit1. Text:='';
    Edit2. Text:='';
    Edit3. Text:='';
    Edit4. Text:='';
    RadioGroup1. ItemIndex:=0;
    Table1. Active:=True;
end;

```

当然,这个系统还很不完善。比如,在还书时,应该有检查是否超期的功能,在查询书目时,作为已借出的书目应有所提示等等,这些都有待于读者去进一步完善。

DP 4.5 数据库报表设计

在设计数据库应用程序时,经常要用到报表。报表是一种比较直观简洁的数据表现形式,



一般用于显示数据库的数据以及程序的结果数据。利用 Delphi 提供的报表组件,可以方便地将数据库中的数据以报表的形式显示出来。

4.5.1 数据库报表的常用组件

1. QuickRep 组件

该组件位于组件板 Qreport 标签页中,如图 4.16 所示。该组件是制作报表的基本组件,构成了报表的主要框架,其上可以包含多个带区:页眉、页脚、标题、表头、明细栏等,在带区上可以放置文本或图形的显示组件。这些带区可以通过 QuickRep 组件的 Bands 特性设置。QuickRep 组件的主要特性和方法如下:

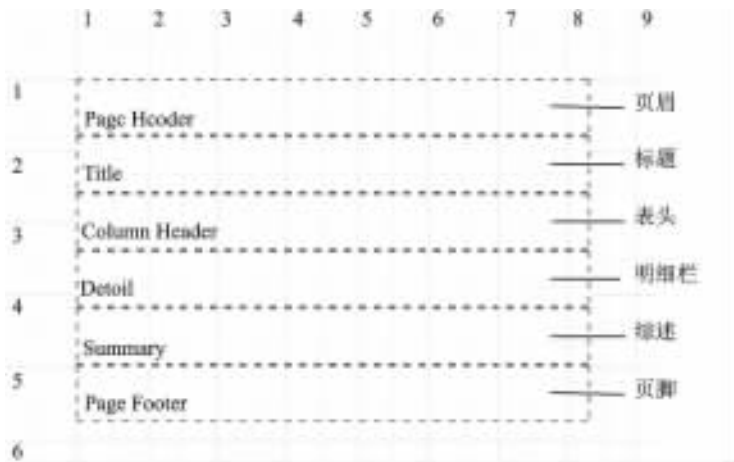


图 4.16 QuickRep 组件的带区结构

(1) 主要特性

● Bands 特性

用于设置报表上出现的带区,包含六个布尔类型的子特性。若各子特性值均设置为 True 时,其含义如下:HasColumnHeader(包含表头带区,用于设置列标题)、HasDetail(包含明细栏区,为每个记录打印明细)、HasPageFooter(包含页脚区,在每页打印页脚)、HasPageHeader(包含页眉区,在每页打印页眉)、HasSummary(包含综述带区,为报表打印一次总计)、HasTitle(包含标题带区,在报表开始处打印标题)。当把 Bands 特性的某个子特性设置为 True 时,相当于在 QuickRep 组件上放置了一个 QRBand 组件。

● DataSet 特性

用于设置与报表相关联的数据集名,可以是 Table 组件、Query 组件或 RemoteDataset 组件。

● RecordCount 特性

返回主表中的记录数。

● RecordNumber 特性

返回当前正在被处理的记录号。



- **Page 特性**

用于设置报表的布局,包括 PaperSize(页面大小)、Columns(分栏数)、BottomMargin(下边距)、TopMargin(上边距)等子特性。

- **Options 特性**

用于设置报表的特性,其子特性有 FirstPageHeader(第一页是否打印页头)、LastPageFooter(最后一页是否打印页脚)、Compression(是否以压缩格式存储报表)。

- **PrinterSetting 特性**

用于设置打印的副本数、打印的起始页和终止页等等打印参数。

- **ShowMessage 特性**

用于设定在准备和打印报表期间显示一个过程对话框。

(2) 主要方法

- **Preview**——显示报表的预览窗口。

- **Print**——打印报表。

- **PrintSetup**——显示打印设置对话框。

(3) 主要事件

- **BeforePrint**——若调用了 Preview 或 Print 方法,则触发该事件。

- **OnPreview**——预览报表时触发该事件。

- **OnStartPage**——当 QuickReport 生成一个新页时触发该事件。

2. QRBand 组件

该组件位于组件板 Qreport 标签页中,用于放置可打印的对象,如 QRLabel 组件等。实际上,在通过 QuickRep 组件的 Bands 特性向一个报表中添加一个带区时,即创建了一个 QRBand 组件。因此有两种向报表中添加带区的方法:从组件板 Qreport 标签页中选择 QRBand 组件置于报表中;由报表组件的 Bands 特性设置生成。

其主要特性如下:

- **BandType**——指定一个 QRBand 组件的带区类型。

- **Frame**——指定带区的边框类型。

- **HasChild**——用于设置该带区是否有子带区。

- **ForceNewColmun**——使该带区显示在新的一列中。

- **ForceNewPage**——使该带区显示在新的一页中。

- **LinkBand**——保证报表打印时,使该带区与 LinkBand 特性指定的带区显示在同一页中。

3. QRChildBand 组件

除了通过设置某个带区的 HasChild 特性添加子带区外,还可以向报表中添加 QRChildBand 组件增加子带区。QRChildBand 组件的两个主要的特性是:

- **ParentBand**——设置该子带区的父带区,这样可以使该子带区连接到其父带区上。

- **LinkBand**——使该子带区与其他带区相连。



4. QRLabel 组件

该组件用于在报表中显示静态文本,如标题、提示等。可以通过 Caption 特性设置。

5. QRDBText 组件

该组件用于在报表中输出数据库中的字段值,可以是字符字段、数值字段、日期字段或备注字段。其主要特性有:

- DataSet——指定与该组件相关连的数据集;
- DataField——指定与该组件相关连的字段名;
- AutoStretch——确定该组件是否随显示内容的多少自动扩展。

6. QRExpr 组件

该组件用于在报表生成时计算并显示一个表达式的值。

- Expression——用于设定所要计算的表达式。
- ResetAfterPrint——指定表达式的值在打印之后复 0。

7. QRSysData 组件

该组件用于在报表中显示系统信息,如系统时间、当前页号及报表中的记录数等。其主要特性有:

- Data——用于指定 QRSysData 组件显示信息的种类,其取值可以是:qrsDate(当前日期)、qrsTime(当前时间)、qrsDateTime(当前日期时间)、qrsDetailCount(报表内的记录数)、qrsDetailNo(当前的记录号)、qrsPageNumber(当前的页号)、qrsReportTitle(报表的 ReportTitle 特性值)。
- Text——指定在系统信息之前的提示信息。

8. QRMemo 组件

用于在报表中显示备注信息(即多行文本信息),并不是数据库中的备注信息。其上显示的文本信息可以通过 Lines 特性设置。

9. QRRichText 组件

用于在报表中显示 RTF 类型的文件,其上显示的内容可以用 Lines 特性设置;也可通过设置 ParentRichEdit 特性为其他 RichEdit 组件名,从而显示其他 RichEdit 组件的内容。

10. QRDBRichText 组件

用于在报表中显示数据库中记录的备注信息。其主要特性有:

- DataSet——指定与该组件相关联的数据集;
- DataField——指定与该组件相关联的字段名;
- AutoStretch——确定该组件是否随显示内容的多少自动扩展。



11. QRShape 组件

用于在报表中绘制一些简单的图形,可以用作边框以及修饰等。图形的形状由 Shape 特性决定,其取值有 qrsCircle(圆)、qrsHorLine(水平直线)、qrsRectangle(矩形)、qrsRightAndLeft(左、右各画一条直线)、qrsTopAndBottom(顶部、底部各画一条直线)、qrsVerLine(垂直直线)。

12. QRImage 组件

用于在报表中显示图像。可以通过该组件的 Picture 特性向报表中添加位图图像。

13. QRDBImage 组件

用于在报表中显示数据库中的图像信息。可以通过 DataSet 和 DataField 特性设置与该组件相连的数据集和字段名。

14. QRPreview 组件

利用该组件用户可以自己定义预览窗口。这时,须要在报表组件的 OnPreview 事件的处理代码中加上:

```
QRPreview1.QRPrinter := TQRPrinter(Sender);
```

即将报表的 Printer 对象赋给 QRPreview 组件的 Printer 对象。

若用户未定义自己的预览窗口,Delphi 将使用默认的预览窗口。

4.5.2 报表应用实例

本实例将为书库数据表 BookLib.db 创建报表,从而详细说明利用 QuickReport 创建报表的具体步骤。

(1) 新建一个应用程序,将缺省窗体的 Caption 特性设置为“书库报表”。

(2) 添加 Table 组件。其 Name 特性默认为 Table1,将其 DatabaseName 特性设置为“book”(图书馆数据库的别名),TableName 特性设置为 BookLib.db。

(3) 添加 QuickRep 组件,其特性设置如下:

- Name 特性默认为 QuickRep1;
- DataSet 特性设置为“Table1”,表示报表的数据源为 Table1 所对应的数据表;
- Bands 特性用于设置报表上的带区,其子特性 HasColumnHeader、HasDetail、HasPageFooter、HasTitle、HasSummary、HasPageheader 取值均设为 True;
- Page 特性用于设置页面,这里将其子特性 PaperSize 设置为“A4”。

(4) 设置报表标题。在 Title 带区上放置 1 个 QRLabel 组件,将其 Caption 特性设置为“图书馆藏书报表”,可以通过 Font 特性调整其字体和字号。

(5) 设置报表的表头(列标题)。在 ColumnHeader 带区上放置 4 个 QRLabel 组件。其 Caption 特性依次为“书名”、“作者”、“数量”和“摘要”。

(6) 设置报表的明细(主要内容)。在 Detail 带区上放置 4 个 QRDBText 组件。其 DataSet 特性均为 Table1,DataField 特性依次为“书名”、“作者”、“数量”和“摘要”。



(7) 设置报表的综述。在 Summary 带区上放置 2 个 QRLabel 组件,用于显示“种类”和“总册数”提示信息,其 Caption 特性分别为“种类”和“总册数”;2 个 QRExpr 组件,分别用于计算和显示书库中的书的种类及库存书的总册数。以设置计算总册数的表达式为例,单击对应的 QRExpr 组件的 Expression 特性值右侧的“...”,弹出如图 4.17(a)所示的 Expression Wizard 对话框;首先单击 Function 按钮,弹出如图 4.17(b)所示的对话框,从中选择计算函数 Sum;单击 Continue 按钮,弹出如图 4.17(c)所示的对话框;单击“...”按钮,弹出如图 4.17(d)所示的对话框;单击 DataBase Field 按钮,从弹出的对话框 4.17(e)中选择 Table1 和“数量”;单击 OK 按钮,这时会在图 4.17(d)中显示“Table1. 数量”;连续单击 OK 按钮,会在图 4.17(a)中出现“SUM[Table1. 数量]”;再单击 OK 按钮,计算表达式即设置完毕。

(8) 设置报表的页眉。在 PageHeader 带区上放置 1 个 QRSysData 组件,将该组件的 Data 特性设置为 qrsDateTime,用于显示系统日期和时间。

(9) 设置报表的页脚。在 PageFooter 带区上放置 1 个 QRSysData 组件,将该组件的 Data 特性设置为 qrsPageNumber,用于显示当前页号。

(10) 设置报表中的边框。边框的设置有两种方法:一是通过对 QuickRep 组件及其各带区的 Frame 特性的设置完成,二是通过添加和设置 QRShape 组件完成。在本报表中外边框和内边框的横线由方法一实现,内边框的竖线由方法二实现。具体实现:通过 QuickRep1 组件 Frame 特性设置报表的外边框,其子特性 DrawBottom、DrawTop、DrawLeft、DrawRight 取值均设为 True;通过各带区的 Frame 特性设置报表内边框的横线,将 Title、ColumnHeader 和 Detail 带区的 Frame 特性的 DrawBottom(底端画线)子特性设置为 True,Summary 带区的 DrawBottom 和 DrawTop(顶端画线)子特性设置为 True;通过 QRShape 组件绘制报表内边框的竖线,在需要绘制竖线的带区上放置 QRShape 组件,将其 Shape 特性设置为 qrsVertLine(垂直线),Height 特性设置为带区高度值,Width 特性设置为 1。

(11) 设置报表的预览和打印功能。在窗体上放置 3 个 Button 组件,其 Caption 特性依次为“预览”、“打印”和“退出”。设置 OnClick 事件的处理代码如下:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
    QuickRep1.Preview;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    QuickRep1.Print;
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
    Close;
end;
```

Delphi 为了方便用户进行报表设计,在程序设计阶段,用户只要将报表的数据源激活(即 Active 特性为 True)后,随时通过双击报表组件的空白位置,便可看到报表的设置和预览界面。



(a)



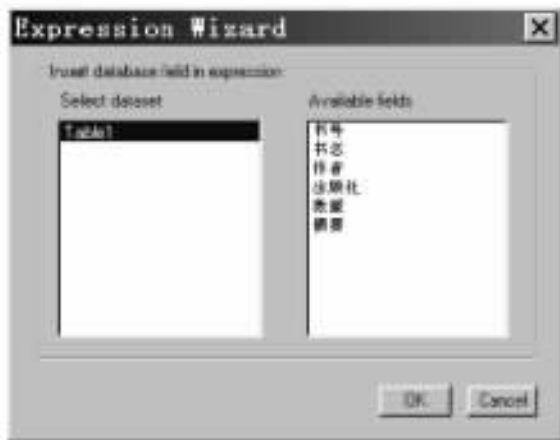
(b)



(c)



(d)



(e)

图 4.17 Expression Wizard 对话框

(12) 程序的运行界面。将程序存盘运行后,出现如图 4.18 所示的运行界面。当单击“预览”按钮后显示如图 4.19 所示的报表结果。



图 4.18 书库报表程序的运行界面



图 4.19 单击“预览”按钮后的报表预览效果

DP **习题与思考题**

4.1 选 择:

- (1) 以下叙述错误的是()。
- a. 每个数据表只能有一个主索引;
 - b. 每个数据表可以有多个次索引;
 - c. 每个数据表可以有多个主关键字;
 - d. 只有各个记录中取值均不相同的字段才有可能成为主关键字段。



- (2) 以下关于 Findkey 方法的叙述中正确的是()。
- a. Findkey 方法是从头到尾对数据表进行顺序查找；
 - b. 使用 Findkey 方法查询前,一定要建立索引；
 - c. Findkey 方法不仅可以实现精确查询,而且可以实现模糊查询；
 - d. 如果只建立了次索引,则不能使用 Findkey 方法进行查询。
- (3) 以下关于数据控制组件中的叙述正确的是()。
- a. DBText 组件用于显示不希望用户修改的字段；
 - b. 当 DBEdit 组件的 Visible 特性设为 False 时,其作用与 DBText 组件相同；
 - c. DBEdit 组件可以用于显示和编辑包括 BLOB 型字段在内的各种字段；
 - d. DataSource 组件是功能最强、用途最广的数据控制组件。

4.2 程序设计。

- (1) 以 student 为别名,利用 Database Desktop 来建立一个同学通讯录表(包括姓名、性别、生日、地址、电话、移动电话等)。
- (2) 设计一个应用程序从别名为 student 的数据表中查询某个记录。
- (3) 在上题的应用程序中添加插入、修改、删除和浏览功能。
- (4) 在上题的应用程序中添加同学通讯录的报表预览和打印功能。



第 5 章

Internet 程序设计

随着 Internet 在全球的推广,网络技术正在各个领域得到越来越广泛的应用,目前正方兴未艾。Delphi 以其强大的程序设计覆盖功能为用户提供了丰富的网络组件。在本章主要通过两个具体的实例,即 Web 浏览器和网上聊天器来说明利用 Delphi 中的网络组件进行 Internet 程序设计的方法。



5.1 实例一 :Web 浏览器

Web 浏览器是从 WWW 上获得信息的最主要方式。在此利用 Delphi 的 WebBrowser 组件设计一个简单的 Web 浏览器。读者可以在此基础上添加其他组件,使其功能更加完善。

5.1.1 WebBrowser 组件

该组件位于组件板 Internet 标签页中,可以实现 Web 网页的显示、下载以及数据导航等功能。

1. 主要特性

- Busy 特性

其值为 True 时,表示 WebBrowser 组件正处在浏览或下载状态。此时,可以通过 Stop 方法终止当前操作。

- FullName 特性

它用于指定被浏览网页的全名。

- LocationName 特性

它表示正在浏览的资源的名字。若该资源是 HTML 页面,则该特性值为此页面的标题;若该资源为本地网络或磁盘上的一个文件或文件夹,则该特性值为此文件或文件夹的路径名。

- Offline 特性

其值为 True 时,表示从本地缓冲区中获得 HTML 页面。

- ReadState 特性

它表示 WebBrowser 组件所处的状态。通过判断其值,可以防止在 WebBrowser 组件未准备好之前对其进行操作。其具体取值及含义如表 5.1 所列。



表 5.1 WebBrowser 组件 ReadState 特性的取值及其含义

常量名称	取 值	含 义
READYSTATE_UNINITIALIZED	0	默认的初始状态,表示没有加载任何特性
READYSTATE_LOADING	1	正在加载特性
READYSTATE_LOADED	2	已经初始化特性,但还未激活
READYSTATE_INTERACTIVE	3	已经激活,数据不是全部可用(正在下载)
READYSTATE_COMPLETE	4	已经激活,数据全部可用(全部下载完毕)

● Silent 特性

其值为 True 时,表示 WebBrowser 组件不显示对话框。

2. 主要方法

- GoBack——转到历史列表中的前一个文档。
- GoForward——转到历史列表中的下一个文档。
- GoHome——转到历史列表中的第一个文档。
- GoSearch——转到当前的搜索页。
- Navigate——转到指定的路径名或 URL,定位并下载指定的资源。
- Navigate2——转到指定的资源,定位并下载指定的资源。
- Refresh——重新下载当前文档。
- Refresh2——在必要的情况下,重新下载当前文档。
- Stop——可以终止当前的操作,如终止定位或下载操作、终止页面上的动态元素等。

3. 主要事件

- OnBeforeNavigate2——在浏览器转向新的资源时发生。
- OnDownloadBegin——在浏览器开始下载文档时发生。
- OnDownloadComplete——在浏览器下载文档完成、失败或终止时发生。
- OnNavigateComplete2——在浏览器成功地转向新的位置时发生。
- OnProgressChange——在下载操作的进度改变后发生。
- OnTitleChange——在浏览器中的页面标题改变时发生。
- OnVisible——在浏览器窗口显示或隐藏时发生。

5.1.2 Web 浏览器的实现

1. 界面的设计

按图 5.1 所示设置 Web 浏览器的界面。

(1) 新建一个项目,将其上空白窗体的 Caption 特性设置为“Web 浏览器”,Name 特性设置为 FormWeb。

(2) 在窗体上放置一个 CoolBar 组件,其 Align 特性设置为 alTop,AutoSize 特性设置为 True,Name 特性为 CoolBar1。



图 5.1 Web 浏览器的界面设计

(3) 在 CoolBar1 组件上依次放置 1 个 Panel 组件、1 个 ComboBox 组件和 1 个 Button 组件。其中 Panel 组件的 Caption 特性为空, Name 特性为 Panel1; ComboBox 组件的 Name 特性为 ComboBox1, Text 特性为空; Button 组件的 Caption 特性为“转到”, Name 特性为 ButtonTurn”。

(4) 选中 CoolBar1 组件, 单击鼠标右键, 在弹出的菜单中选择 Bands Editor, 出现 CoolBar 的 Bands 编辑窗口, 如图 5.2 所示。由第(3)步将 CoolBar1 组件分割为 3 个 Band(段)。当然也可以在此窗口中单击 Ins 按钮插入新段, 单击 Del 删除选中的段, 还可以通过上下箭头调整各段的相对位置。当选图中图 5.2 所示窗体中的某个段名时, 可以通过 Object Inspector 设置该段的特性。在此将 1-TCoolBand 的 Text 特性设置为“地址”, 使该段前面显示“地址”; 将 2-TCoolBand 的 Break 特性设置为 False, 使该段和前一段位于同一行。



图 5.2 CoolBar1 组件的 Bands 特性的编辑窗口

(5) 在 Panel1 组件上放置 5 个 SpeedButton 组件, 其 Name 特性依次为 SpeedButtonBack、SpeedButtonForward、SpeedButtonRefresh、SpeedButtonStop 和 SpeedButtonExit; 按图 5.1 设置各 SpeedButton 组件的 Glyph 特性; 各组件的 Hint 特性依次为‘后退|后退’、‘前进|前进’、‘刷新|刷新’、‘停止|停止’、‘退出|退出’; 各组件的 ShowHint 特性均为 True。

(6) 在窗体上放置 1 个 StatusBar 组件, 其 Align 特性为 alBottom, AutoHint 特性为 True, SimplePanel 特性为 True, Name 特性为 StatusBar1。

(7) 在窗体上放置 1 个 WebBrowser 组件, 其 Align 特性为 alClient(充满窗体的客户区), Name 特性为 WebBrowser1。



2. 程序的实现

各组件的事件处理代码如下：

(1) SpeedButtonBack(后退按钮)的 OnClick 事件处理代码：

```
procedure TFormWeb.SpeedButtonbackClick(Sender: TObject);
begin
    try
        WebBrowser1.GoBack;
    except
        StatusBar1.SimpleText:='Can not Go Back!';
        Halt;
    end;
end;
```

(2) SpeedButtonForward(前进按钮)的 OnClick 事件处理代码：

```
procedure TFormWeb.SpeedButtonForwardClick(Sender: TObject);
begin
    try
        WebBrowser1.GoForward;
    except
        StatusBar1.SimpleText:='Can not Go Forward!';
        Halt;
    end;
end;
```

(3) SpeedButtonRefresh(刷新按钮)的 OnClick 事件处理代码：

```
procedure TFormWeb.SpeedButtonRefreshClick(Sender: TObject);
begin
    WebBrowser1.Refresh;
end;
```

(4) SpeedButtonStop(停止按钮)的 OnClick 事件处理代码：

```
procedure TFormWeb.SpeedButtonStopClick(Sender: TObject);
begin
    WebBrowser1.Stop;
end;
```

(5) SpeedButtonExit(退出按钮)的 OnClick 事件处理代码：

```
procedure TFormWeb.SpeedButtonExitClick(Sender: TObject);
begin
    Close;
end;
```

(6) ButtonTurn(转到按钮)的 OnClick 事件处理代码：



```
procedure TFormWeb.ButtonTurnClick(Sender: TObject);  
begin  
  WebBrowser1.Navigate(ComboBox1.Items.Text);  
  StatusBar1.SimpleText := 'Opening '+ComboBox1.Items.Text;  
end;
```

DP 5.2 实例二 :网上聊天器

网上聊天器可以为异地的用户提供互相交谈的工具,交谈双方均启动网上聊天器程序,一方输入对方的 IP 地址,双方即可自由交谈。利用 Delphi 提供的 ClientSocket 组件和 ServerSocket 组件将很容易实现该程序。

5.2.1 ClientSocket 组件

该组件位于组件板 Internet 标签页上,用于管理和控制 TCP/IP 客户端的网络连接。也就是说,借助 ClientSocket 组件可以使程序转化成 TCP/IP 客户端的应用程序。具体可以指定要连接的 TCP/IP 服务器,管理已打开的连接或在接到连接终止的指令关闭连接。

1. 主要特性

- ClientType 特性
它指定客户端读/写信息是否与网络连接同步。其取值有 ctNonBlocking(非阻塞型)和 ctBlocking(阻塞型)两种。当取值为 ctNonBlocking 时,表示客户端可以同步地响应读/写事件。
- Active 特性
它指定连接是否打开以及能否与其他计算机通信。
- Address 特性
它存放服务器的 IP 地址。
- Host 特性
它指定服务器的 IP 地址的别名,其值为包含域名和服务的字符串。
- Port 特性
它指定标识服务器端连接的 ID 号。
- Service 特性
它指定连接所使用的服务名称,例如 ftp、http、finger 以及 time 等。
- Socket 特性
它描述 Windows 客户端连接的结点信息的 TClientWinSocket 对象。

2. 主要方法

- Create 方法——创建 TClientSocket 对象。
- Destroy 方法——撤消 TClientSocket 对象。



- Close 方法——关闭与服务器端的连接。
- Open 方法——打开与服务器端的连接。

3. 主要事件

- OnConnect 事件——当客户端与服务器端建立连接时发生。
- OnConnecting 事件——当服务器已找到,但尚未建立连接时发生。
- OnDisconnect 事件——在客户端关闭与服务器端的连接之前发生。
- OnError 事件——在客户端与服务器端建立、使用或关闭连接产生错误时发生。
- OnLookup 事件——在客户端寻找要连接的服务器时发生。
- OnRead 事件——当客户端通过连接读取信息时发生。
- OnWrite 事件——当客户端向连接写入信息时发生。

5.2.2 ClientWinSocket 组件

该组件是一个不可视的组件,其父类为 TCustomWinSocket,用于描述 Windows 客户端网络连接的结点信息。

1. 主要特性

- Connected 特性——指定是否与其他计算机建立连接并且可用。
- LocalHost 特性——指定本地系统名称。
- LocalAddress 特性——指定本地系统的 IP 地址。
- RemoteAddress 特性——指定要建立连接的另一端 IP 地址。
- RemoteHost 特性——指定要建立连接的远程另一端主机名。

2. 主要方法

- Connect——建立连接。
- Disconnect——断开连接。
- Open——打开连接。
- Close——关闭连接。
- RecieveText——从网络连接读取一个字符串。
- SendText——向网络连接发送一个字符串。

5.2.3 ServerSocket 组件

该组件位于组件板 Internet 标签页上,用于管理和控制 TCP/IP 服务器端的网络连接。也就是说,借助 ServerSocket 组件可以使程序转化成 TCP/IP 服务器端的应用程序。具体可以监听其他计算机发来的 TCP/IP 连接请求,并在接到请求后建立连接。

1. 主要特性

- ServerType 特性

它指定服务器端的读/写操作类型,其取值有 stNonBlocking(非阻塞型)和 stThread-



Blocking(线程阻塞型)。

- Active 特性

它指定服务器端的连接是否打开,是否可以与其他的计算机建立连接。

- Port 特性

它指定标识服务器端网络连接的 ID 号。

- Service 特性

它指定网络连接中正在使用的服务名称。

- Socket 特性

它指定监听连接点的 TServerWinSocket 对象。

2. 主要方法

- Create 方法——创建 TServerSocket 对象。

- Destroy 方法——撤消 TServerSocket 对象。

- Close 方法——关闭服务器端的连接。

- Open 方法——打开服务器端的连接。

3. 主要事件

- OnClientConnect 事件

当某个客户端与服务器端建立连接时发生。

- OnClientDisconnect 事件

在某个客户端与服务器端的连接关闭时发生。

- OnClientError 事件

某个客户端在与服务器端建立、使用或关闭连接产生错误时发生。

- OnClientRead 事件

当服务器端从某个客户端读取信息时发生。

- OnClientWrite 事件

当服务器端向某个客户端写入信息时发生。

- OnAccept 事件

在服务器端接受某个客户端连接后发生。

- OnListen 事件

在服务器端打开监听前发生。

5.2.4 ServerWinSocket 组件

该组件是一个不可视的组件,其父类为 TCustomWinSocket,用于描述监听服务器端网络连接的结点信息。

1. 主要特性

- Connections 特性

为数组类型值,列出当前正与服务器连接的各个客户端的信息。



- ActiveConnections 特性

表示当前正与服务器连接的客户端数量。

2. 主要方法

- Accept——接受某个客户端的连接。
- Disconnect——关闭 Windows 网络连接。
- Listen——为服务器端打开监听连接。

5.2.5 网上聊天器的实现

1. 界面的设计

按图 5.3 设置网上聊天器的界面。



图 5.3 网上聊天器的界面设计

(1) 新建一个项目,调整空白窗体的大小,并将窗体的 Caption 特性设置为“网上聊天器”,Name 特性设置为 FormChat,AutoSize 特性设置为 True。

(2) 在窗体上放置一个 Panel 组件,其 Caption 特性为空,BevelInner 特性为 bvLowered(内部凹陷),BevelOuter 特性为 bvRaised(外边框凸起),Name 特性为 Panel1。

(3) 在 Panel1 组件上放置 4 个 BitBtn 组件,其 Caption 特性依次为“监听”、“连接”、“断开”、“退出”,Name 特性依次为 BitBtnListen、BitBtnConnect、BitBtnDisconnect、BitBtnExit,并按图所示设置各个按钮表面的位图。

(4) 在窗体上放置 2 个 Memo 组件,其 Name 分别为 Memo1 和 Memo2,按图所示调整其位置和大小,并将 2 个组件的 Lines 特性设置为空,ScrollBar 特性设置为 ssVertical(具有垂直滚动条)。

(5) 在窗体上放置 1 个 ClientSocket 组件,其 ClientType 特性为 ctNonBlocking,Port 特性为“50”(交谈双方约定的端口号),Name 特性为 ClientSocket1。

(6) 在窗体上放置 1 个 ServerSocket 组件,其 ServerType 特性为 ctNonBlocking,Port 特性为“50”,Name 特性为 ServerSocket1。

(7) 在窗体上放置 1 个 StatusBar 组件,显示状态信息,其 SimplePanel 特性设置为 True,



Name 特性设置为 StatusBar1。

2. 程序的实现

在该程序中用到以下两个变量：

```
Server:string;           //接收用户输入的 IP 地址值
isServer:Boolean;        //服务器端的标志,为 True,表示是服务器端
```

(1) “监听”按钮的 OnClick 事件处理代码

```
procedure TFormchat.BitBtnlistenClick(Sender: TObject);
begin
if isServer then
//若在服务器端,则使“监听”按钮为可用,“连接”和“断开”按钮不可用
begin
    BitBtnlisten.Enabled:=True;
    BitBtnConnect.Enabled:=False;
    BitBtnDisconnect.Enabled:=False;
end
else
//若在客户端,则使“监听”按钮为不可用,“连接”和“断开”按钮可用
begin
    BitBtnlisten.Enabled:=False;
    BitBtnConnect.Enabled:=True;
    BitBtnDisconnect.Enabled:=True;
end;
if BitBtnlisten.Enabled then
//若“监听”按钮可用,则使服务器端激活处于监听状态
begin
    ClientSocket1.Active:=false;
    ServerSocket1.Active:=true;
    StatusBar1.SimpleText:=‘服务器正在监听中……’;
end
else
//若“监听”按钮不可用,则退出监听状态
begin
    if ServerSocket1.Active then
        ServerSocket1.Active:=false;
        StatusBar1.SimpleText:=‘’;
    end;
end;
end;
```

(2) “连接”按钮的 OnClick 事件处理代码

```
procedure TFormchat.BitBtnconnectClick(Sender: TObject);
```



```
begin
  if ClientSocket1.Active then
    //若客户端已经建立连接,则断开连接
    ClientSocket1.Active:=False;
  if inputquery('将要连接到','服务器地址',Server) then
    //提示输入要连接的新的服务器地址,建立新的连接
    if length(Server)>0 then
      with ClientSocket1 do
        begin
          Host:=Server;
          try
            Active:=True;
            StatusBar1.SimpleText:=连接服务器成功;
          except
            StatusBar1.SimpleText:=连接服务器失败;
          end;
        end;
      end;
    end;
  end;
```

(3) “断开”按钮的 OnClick 事件处理代码

```
procedure TFormchat.BitBtnDisconnectClick(Sender: TObject);
begin
  //断开客户端的连接
  try
    ClientSocket1.Active:=False;
    StatusBar1.SimpleText:=断开与服务器的连接;
  except
    StatusBar1.SimpleText:=断开连接失败;
  end;
end;
```

(4) “退出”按钮的 OnClick 事件处理代码

```
procedure TFormchat.BitBtnExitClick(Sender: TObject);
begin
  //关闭服务器端、客户端的连接,关闭窗口
  ServerSocket1.Close;
  ClientSocket1.Close;
  Close;
end;
```

(5) ClientSocket1 组件的 OnConnect 事件的处理代码

```
procedure TFormchat.ClientSocket1Connect(Sender: TObject;
  Socket: TCustomWinSocket);
```



```
begin
    //显示要连接的主机名,清除客户端 Memo1 和 Memo2 中的内容
    StatusBar1.SimpleText:= '连接到 :'+Socket.RemoteHost;
    Memo1.Lines.Clear;
    Memo2.Lines.Clear;
end;
```

(6) . ServerSocket1 组件的 OnConnect 事件的处理代码

```
procedure TFormchat.ServerSocket1ClientConnect(Sender: TObject;
    Socket: TCustomWinSocket);
begin
    //清除服务器端 Memo1 和 Memo2 中的内容
    Memo1.Lines.Clear;
    Memo2.Lines.Clear;
end;
```

(7) ServerSocket1 的 OnAccept 事件的处理代码

```
procedure TFormchat.ServerSocket1Accept(Sender: TObject; Socket: TCustomWinSocket);
begin
    //服务器端若接受了连接,则将 isServer 置为 True,显示接受连接的客户端地址
    isServer:=true;
    StatusBar1.SimpleText:= '连接到 :'+Socket.RemoteAddress;
end;
```

(8) Memo1 组件的 OnKeyDown 事件的处理代码

```
procedure TFormchat.Memo1KeyDown(Sender: TObject; var Key: Word;
    Shift: TShiftState);
begin
    if Key=VK_Return then
        //交谈者可以在 Memo1 中输入文本,若键入回车,则将信息发送出去,并使 Memo1
        //中的文本换行
        if isServer then
            ServerSocket1.Socket.Connections[0].SendText(Memo1.Lines[Memo1.Lines.Count-1])
        else
            ClientSocket1.Socket.SendText(Memo1.Lines[Memo1.Lines.Count-1]);
end;
```

(9) ClientSocket1 组件的 OnRead 事件处理代码

```
procedure TFormchat.Read(Sender: TObject; Socket: TCustomWinSocket);
begin
    //客户端从网络连接中读到信息后,将其显示在 Memo2 中
    Memo2.Lines.Add(Socket.ReceiveText);
end;
```



(10) ServerSocket1 组件的 OnClientRead 事件的处理代码

```
procedure TFormchat.ServerSocket1ClientRead(Sender: TObject;  
    Socket: TCustomWinSocket);  
begin  
    //服务器端从网络连接中读到信息后,将其显示在 Memo2 中  
    Memo2.Lines.Add(Socket.ReceiveText);  
end;
```

(11) ClientSocket1 组件的 OnError 事件处理代码

```
procedure TFormchat.ClientSocket1Error(Sender: TObject;  
    Socket: TCustomWinSocket; ErrorEvent: TErrorEvent; var ErrorCode: Integer);  
begin  
    //客户端连接出错时,给出出错提示  
    Memo2.Lines.Add('与服务器+Server+连接错误');  
    ErrorCode:=0;  
end;
```

DP 习题与思考题

- 5.1 为 Web 浏览器程序添加菜单功能。
- 5.2 网上聊天器程序中主要用到哪两个网络组件?它们的功能是什么?



第 6 章

定制组件

利用 Delphi 所提供的组件,可以很容易的实现各种复杂程序的设计。当然,用户在开发程序过程中,可能会遇到 Delphi 中的组件不是很合适,希望能自己设计一些组件,Delphi 提供了用户自己定制组件的功能。在这一章中,主要介绍如何创建自己的组件以及如何将其安装在组件板中。

DP 6.1 创建组件

在 Delphi 中,各个组件是属于对应的组件类的,所有的组件类都是从 TComponent 类继承下来的。用户要创建新的组件,首先应选择一个已有的组件类(一般使用 TCustom 开头的类)作为基类,新组件将继承基类的所有特性和事件,用户只要在基类的基础上修改或添加新的特性和事件即可。

下面以自定义一个时钟标签组件为例,介绍创建组件的具体步骤。该时钟标签组件能够在屏幕上显示当前的系统时钟信息,具有 Color、Font、Active 等特性,具有 OnClick(单击)事件。该时钟标签组件类可以从 TCustomLabel 继承;若想在其表面显示系统时间,要借助 Delphi 提供的 Timer 组件,因为该组件可以产生连续的系统时间。

6.1.1 创建组件单元文件

选择 Component | New Component 菜单项,弹出如图 6.1 所示的对话框。

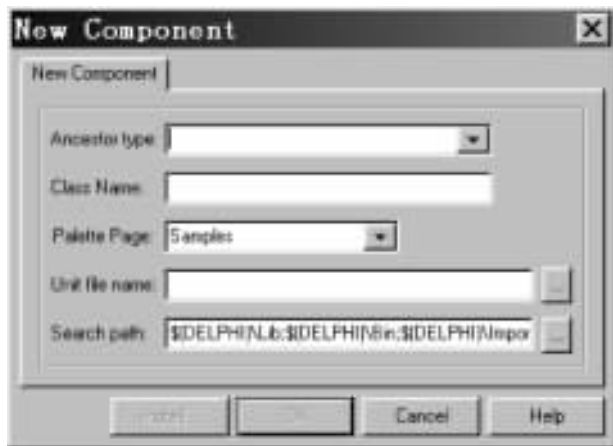


图 6.1 新组件设置对话框



下面在新组件对话框中进行设置:在 Ancesor Type 下拉列表框中选择组件的基类 TCustomLabel[StdCtrls],表示该组件类是从 TCustomLabel 继承的;在 Class Name 文本框中输入创建的新组件类名 TClockLabel;Pallette Page 下拉列表框用于指定新组件所在的标签页,这里选择 Samples;Unit File Name 指出新组件对应的单元文件名(可根据类名自动产生)及路径;Search Path 指出搜索单元文件的路径。

当设置好各选项后,单击 OK 按钮,会自动生成如下代码:

```
unit ClockLabel;
interface
uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls;
type
  TClockLabel = class(TCustomLabel)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;
procedure Register;
implementation
procedure Register;
begin
  RegisterComponents('Samples', [TClockLabel]);
end;
end
```

这是新组件 ClockLabel 的代码框架。可以看出,TClockLabel 类是以 TCustomLabel 为父类;TClockLabel 类的内部还是空的,有待于添入特性和事件。另外,Delphi 还自动生成了一个 Register 过程,它的作用是将新的组件类 TClockLabel 添加到组件板指定的标签页 Samples 上。

6.1.2 添加特性

组件 ClockLabel 的基类 TCustomLabel 提供了很多的保护(protected)特性。若要使用户可以访问这些特性,需要在 TClockLabel 类的变量声明部分将其定义为 published 特性(或 public 特性),各个特性前要加上关键字 property,例如 Align、Alignment 及 Visible 等特性。定义在 published 下的特性,设计时可以在对象观察器中看到;而定义在 public 下的特性在对象观察器中看不到。

若要为该组件添加新的特性,可以采用 read、write、default 或 stored 等关键字。其中



read 表示读取特性值, write 表示设置特性值, default 表示设置特性的默认值, stored 表示该特性被重新设置后是否会永久保存。例如要为 ClockLabel 组件添加一个是否显示系统时间的特性 Active, 该特性值应为 Boolean 型, 可以在 published 下录入: property Active: Boolean, 然后按下 Shift + Ctrl + C 键, 系统将自动生成代码:

```
read FActive write SetActive default False;
//表示通过 FActive 变量取得 Active 的特性值,通过 SetActive 过程
//设置 Active 的特性值,而且 Active 的初值为 False
```

并在 Private 下自动生成:

```
FActive: Boolean;
procedure SetActive(const Value: Boolean);
```

在 implementation 下生成:

```
procedure TClockLabel.SetActive(const Value: Boolean);
begin
    FActive := Value;
end;
```

6.1.3 添加事件和方法

此处的事件可以看作是一种特殊的特性。该类特性的值应为事件类型,而且必须在 published 下面定义。

例如:要为 ClockLabel 组件定义一个 OnClick 事件,可以在 published 下面添加定义:

```
property OnClick: TnotifyEvent read FnotifyEvent write FnotifyEvent;
```

在 Private 下添加定义:

```
FnotifyEvent: TnotifyEvent;
```

其中, TnotifyEvent 类是一个表示不需要参数的事件类。

为组件添加事件后,还要添加处理事件的方法。

例如,在 public 下为 ClockLabel 组件添加单击鼠标事件的方法代码:

```
procedure ClickCreate(var Message: TMessage); Message WM_LButtonDown;
```

其中,为 Windows 的消息,表示按下鼠标左键。

在 implementation 后添加单击鼠标事件的方法实现代码:

```
procedure TClockLabel.ClickCreate(var Message: TMessage);
//用于指定消息 WM_LButtonDown(按下鼠标左键)与 OnClick 事件相连
begin
    if assigned(OnClick) then
        OnClick(Self);
end;
```

下面为 ClockLabel 组件添加方法 OpenClock 和 CloseClock,分别用于显示时钟和关闭时



钟,其代码如下:

```
procedure TClockLabel.OpenClock (Sender: TObject);
begin
    Caption := TimeToStr(Time);           //取系统时间显示在 ClockLabel 组件上
end;
procedure TClockLabel.CloseClock (Sender: TObject);
begin
    Caption := '';                       //ClockLabel 组件的 Caption 特性为空,不显示系统时间
end;
```

6.1.4 定义构造函数和析构函数

构造函数用于建立和初始化对象,析构函数用于删除对象,释放分配给对象的内存等资源。作为新的组件对象也需要定义构造和析构函数,而且在设计期间将组件放置在窗体上;或运行期间放置该组件的窗体被创建时调用其构造函数,在运行期间放置组件的窗体被释放时则调用组件的析构函数。析构函数和构造函数必须在 public 部分声明,在 implementation 部分给出具体的定义,例如:

```
constructor TClockLabel.Create (AOwner: TComponent); //ClockLabel 组件的构造函数
begin
    inherited Create (AOwner);           //调用基类的构造函数
    FTimer := TTimer.Create (self);      //创建 TTimer 对象,Ftimer 为 Ttimer 类型的变量
    FTimer.Enabled := True;              //使对象处于活动状态
    FTimer.Interval := 1000;             //定时间隔为 1000ms
end;
destructor TClockLabel.Destroy;          //ClockLabel 组件的析构函数
begin
    Inherited Destroy;                  //调用基类的析构函数
end;
```

6.1.5 ClockLabel 单元的完整代码

其代码如下:

```
unit ClockLabel;
interface
uses
    Windows, Messages, SysUtils, Classes, Controls, StdCtrls, ExtCtrls;
// ExtCtrls 单元包含 Ttimer 类的定义
type
    Tclocklabel = class(TCustomLabel)
    private
        { Private declarations }
        FActive: Boolean;
```



```

    FTimer: TTimer;
    FnotifyEvent: TnotifyEvent;
    procedure SetActive(const Value: Boolean);
protected
    { Protected declarations }
    procedure OpenClock (Sender: TObject);
    procedure CloseClock (Sender: TObject);
public
    { Public declarations }
    constructor Create (AOwner: TComponent); override;
    destructor Destroy ;override;
    procedure ClickCreate(var Message: TMessage); Message WM_LButtonDown;
published
    { Published declarations }
    property Align;
    property Alignment;
    property Color;
    property Font;
    property Transparent;
    property Visible;
    property Active: Boolean read FActive write SetActive default False;
    property OnClick; TNotifyEvent read FNotifyEvent write FNotifyEvent;
end;
procedure Register;
implementation
procedure Register;
begin
    RegisterComponents('Samples', [Tclocklabel]);
end;
constructor TClockLabel. Create (AOwner: TComponent);
begin
    inherited Create (AOwner);
    FTimer := TTimer. Create (Self);
    FTimer. Enabled := True;
    FTimer. Interval := 1000;
end;
destructor TClockLabel. Destroy;
begin
    inherited destroy;
end;
procedure TClockLabel. SetActive(const Value: Boolean);
begin
    FActive := Value;

```



```

    if FActive then
    FTimer. OnTimer := OpenClock
    //当 Active 特性为 True 时,用 OpenClock 作为 OnTimer 事件的处理过程
    else
    FTimer. OnTimer := CloseClock;
    //当 Active 特性为 False 时,用 CloseClock 作为 OnTimer 事件的处理过程
end;
procedure TClockLabel. ClickCreate(var Message; TMessage);
begin
    if assigned(OnClick) then
        OnClick(Self);
end;
procedure TClockLabel. OpenClock (Sender: TObject);
begin
    Caption := TimeToStr(Time);
end;
procedure TClockLabel. CloseClock (Sender: TObject);
begin
    Caption := '';
end;
end.

```

DP 6.2 安装组件

安装组件是将编译后的组件放置在组件板上,实质上是把组件安装到程序包中。其具体步骤如下:

- (1) 选择 Component | InstallComponent 菜单项,弹出如图 6.2 所示的对话框。

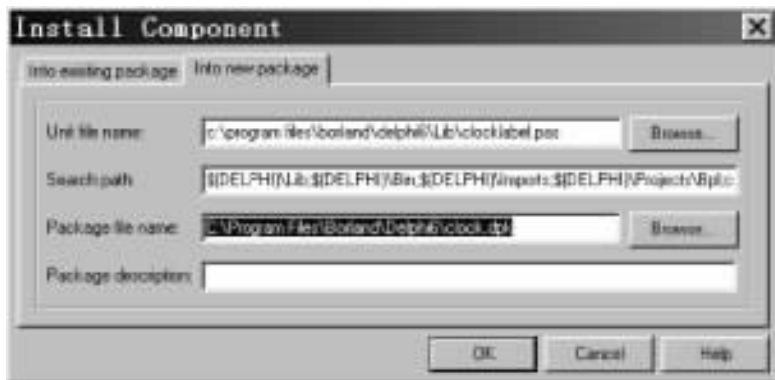


图 6.2 安装组件设置对话框

该对话框中有 2 个标签页:Into existing package(安装到已存在的包中)、Into new pack-



age(安装到新的包中)。这里选择 Into new package 标签页;Unit file name 指的是欲安装组件的单元文件名,在此输入 clockLabel.pas;Search path 指的是欲安装组件的单元文件的搜索路径;Package file name 指的是包含该组件类的包文件名,这里录入 clock.dpk;Package description 指的是包文件的说明。当设置完毕,单击 OK 按钮。

(2) 在弹出的如图 6.3 所示的对话框中,单击 Yes 按钮后,Delphi 会执行 Register 过程,编译并注册组件。若成功,则显示如图 6.4 所示的提示信息,生成负责管理定制的组件文件 clock.dpk 和 clock.bpl。

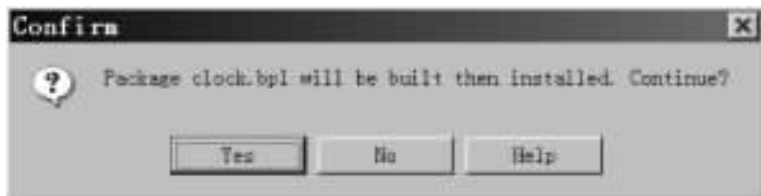


图 6.3 编译确认对话框

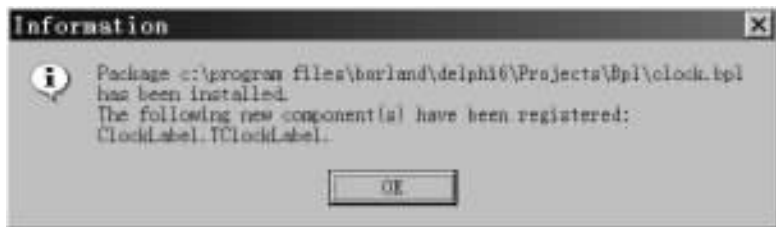


图 6.4 编译注册成功信息显示框

(3) 在操作过程中,用户会发现在屏幕上会出现如图 6.5 所示的窗口。该窗口显示的是当前目录下所有的包文件。其中 Compile 按钮是对添加到包中的单元文件进行编译,Add 按钮用于添加单元文件,Remove 按钮用于删除单元文件,Install 按钮是将当前包中的组件安装到组件板中,Option 按钮用于设置 Delphi 中的包文件。

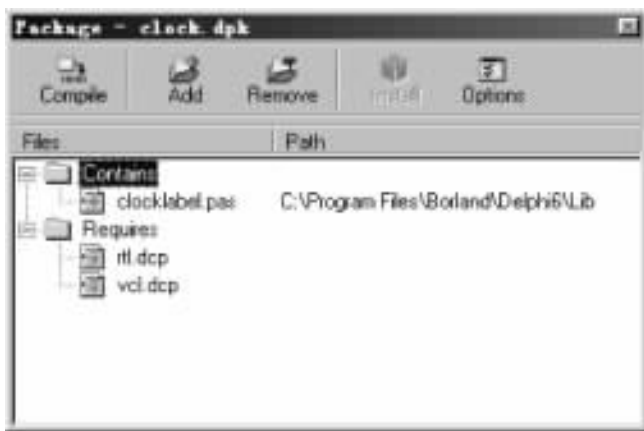


图 6.5 组件包管理窗口



通过以上步骤,可以在组件板 Sample 标签页上发现一个新的组件 ClockLabel——如图 6.6 所示的 Sample 标签页的最左边的组件。用户可以像使用 Delphi 其他组件一样,向应用程序中添加该组件,并通过设置该组件的特性和事件处理程序,进行程序设计。该组件除了具有 Color、Font 等特性外,其中 Active 特性比较特殊,当其值为 True 时,其表面将显示系统时间。当 Active 特性为 False 时,表面空白。该组件还具有 OnClock(单击)事件。

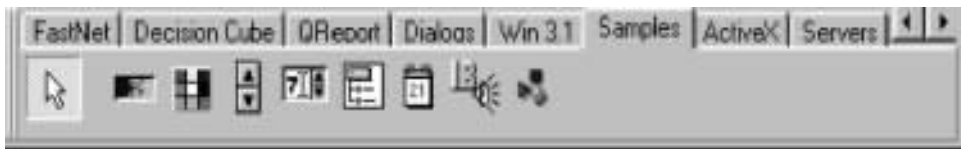


图 6.6 添加了 ClockLabel 组件的 Sample 标签页

DP 习题与思考题

- 6.1 创建组件的基本步骤是什么?
- 6.2 如何安装组件?
- 6.3 以 TCustomStatusBar 组件为基类创建一个新组件,使其上能显示当前的系统时间,并具有 Active、Color、Font、Height、Weight 等特性。



第 7 章

Delphi 软件集成技术

DP 7.1 帮助文件的制作

作为一个有实用价值的应用软件,为了便于用户操作,通常带有联机帮助,而帮助信息是由帮助文件提供的。下面结合具体的实例,介绍一下帮助文件的制作过程。

在这一节中,将要建立一段关于 Delphi 主窗口的帮助文件。如图 7.1 所示,在目录或索引标签页中,选择“主窗口”后,单击“显示”按钮,会弹出与“主窗口”相关的帮助内容,如图 7.2 所示。其中有些文字带有下划线,例如“标题栏”。当单击“标题栏”时,又会弹出关于“标题栏”的帮助信息。



图 7.1 Delphi 主窗口帮助的主界面

7.1.1 创建 RTF 文件

RTF 文件是帮助文件的主体,其中包含向用户显示的大部分帮助内容。制作帮助文件首先要创建 RTF 格式的文件,选用支持 RTF 文件格式的编辑软件建立 RTF 文件。这里选择 Word2000。

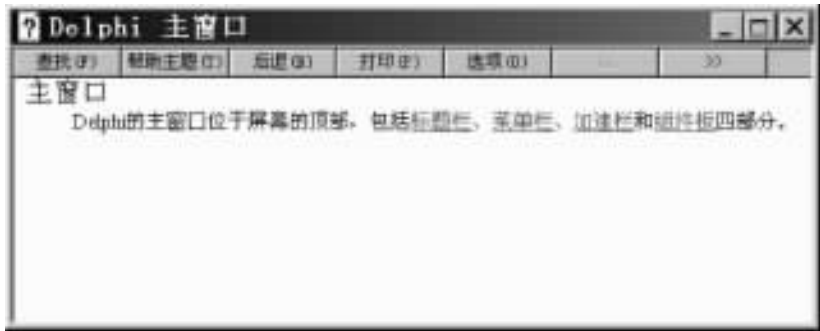


图 7.2 主窗口的帮助信息

在创建 RTF 文件时,每一个帮助主题占一页,也就是要用分页符将各个主题分开。在每个主题的前面插入适当的脚注或尾注,用来控制主题在帮助窗口出现的位置及顺序等。

常用的脚注符号有:

- #(主题标识符)——用于唯一地标识每一个主题,各个主题不能重名;
- \$(标题)——用此符号标识的主题可以显示在书签定义对话框、历史记录窗口中;
- K(关键字)——用此符号标识的主题可以在索引对话框中出现;
- +(浏览序号)——确定用“<<”或“>>”浏览主题的顺序;
- >(窗口类型)——确定显示主题的窗口的类型。

设定热点文字。热点文字通常以不同的颜色显示在文本信息中,当鼠标移到其上时,鼠标箭头呈手状。若单击鼠标,则转向与该热点文字相关的其他主题窗口。

首先选定欲设为热点的文字,通过“格式”|“字体”菜单项,设置下画线类型为双画线。其后录入与该热点文字相关的主题名,并通过“格式”|“字体”菜单项将其效果设为“隐藏文字”。

例如:热点文字为“标题栏”,其相关的主题名为“标题栏简介”,则可以这样设置:

标题栏标题栏简介

主题名“标题栏简介”应紧接在热点文字之后,并且为“隐藏文字”格式。

图 7.3 显示了与 Delphi 主窗口帮助文件对应的 RTF 文件的具体内容,其中包括了各个主题中尾注的设置和热点文字的设置。

各个主题的尾注信息如表 7.1 所列。

表 7.1 “Delphi 主窗口帮助”中各主题及其尾注对照表

主题名	主窗口	标题栏简介	菜单栏简介	加速栏简介	组件板简介
尾 注	# 主窗口	# 标题栏简介	# 菜单栏简介	# 加速栏简介	# 组件板简介
	\$ 主窗口	\$ 标题栏简介	\$ 菜单栏简介	\$ 加速栏简介	\$ 组件板简介
	K 主窗口	K 标题栏简介	K 菜单栏简介	K 加速栏简介	K 组件板简介
	+ 1	+ 2	+ 3	+ 4	+ 5
	> 小窗口				

7.1.2 生成 HLP 文件

建立了 RTF 文件后,可以用 Delphi 自带的 HCW. EXE 程序生成 HLP 文件,HCW. EXE



图 7.3 Delphi 主窗口帮助文件对应的 RTF 文件内容

文件位于 Borland\Delphi 6.0\Help\Tools 文件夹下。该程序启动后,界面如图 7.4 所示。



图 7.4 HCW.EXE 程序的运行界面

要生成 HLP 文件,首先要建立 CNT(目录)文件和 HPJ(工程)文件。

1. 建立目录文件

在图 7.4 中选择 File|New|Help Contents 项,进入目录文件建立界面,如图 7.5 所示。

在 Default filename 位置处输入欲建立的帮助文件名,在 Default title 处输入帮助窗口的标题,可以使用 Add Above 和 Add Below 按钮在被选项上面或下面添加目录项(这些目录项将出现在帮助窗口的目录标签页中),Move Right 和 Move Left 按钮可用来调整目录项的左右位置,ReMove 按钮用于删除目录项。当目录项设置完毕,可以单击存盘按钮保存目录文件。为方便起见,通常将目录文件、工程文件和帮助文件取相同的文件名,而通过扩展名区分。

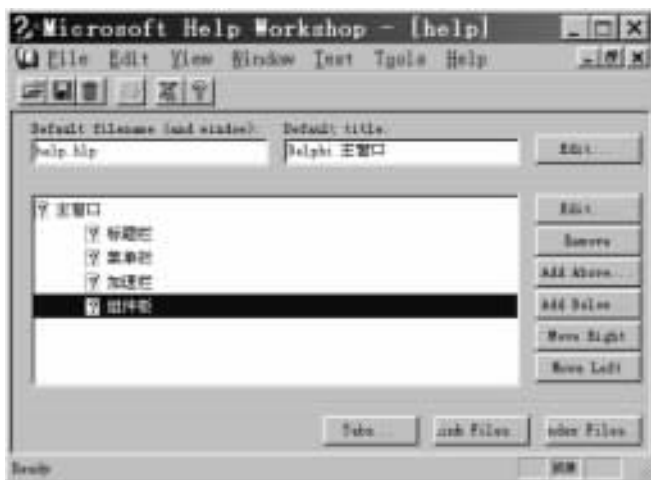


图 7.5 建立目录文件窗口

2. 建立工程文件

在 HCW 的主菜单中选择 File|New|Help Project 项,在弹出的对话框中输入工程文件名,这时,弹出如图 7.6 所示的窗口。单击 Options 按钮,在弹出的窗口中可以设置各种编译选项。其中 General 标签页用于设定默认的主题名,编译时决定是否显示进度及提示信息以及该帮助的版本信息等。单击 Files 按钮,在弹出的对话框中再单击 ADD 按钮,添加已生成的 RTF 文件。单击 Windows 按钮,可以在弹出的对话框中自定义窗口类型(包括窗口的大小、位置、颜色以及窗口中出现的按钮的设置等)。这些窗口类型可应用于各个主题窗口。单击 Map 按钮,在弹出的对话框中为每个主题设定一个主题索引值(应为整数)。



图 7.6 建立工程文件窗口

3. 生成帮助文件

当工程文件设置完毕,单击 Save and Compile 按钮,将工程文件存盘并编译生成帮助文



件。若编译正常会出现如图 7.7 所示的窗口,其中 notes 数和 Warnings 数均为 0。至此,生成了一个名为 Help. HLP 的帮助文件。双击该文件,即可看到帮助信息,如图 7.1 所示。



图 7.7 编译生成帮助文件

7.1.3 嵌入应用程序

要在应用程序中添加帮助功能,可以借助 Delphi 的 Application 方法实现。Application 方法中与帮助有关的特性和方法:

- HelpFile 特性——设置帮助文件的名字;
- HelpJump 方法——指定帮助文件中相关的主题名;
- HelpContext 方法——指定帮助文件中相关的主题索引号。

例如:若希望在按下某个按钮时,出现 Help. HLP 的帮助信息,则可以在该按钮的 OnClick 事件的处理代码中加入:

```
Application. HelpFile:= 'c:\My Documents\Help. HLP';
Application. HelpJump(标题栏简介);
//主题“标题栏简介”的主题索引值为 2,因此也可以写成 Application. HelpContext(2);
```



7.2 软件安装盘的制作

一个应用程序在设计完成后,为保护设计者的劳动果实,也为了用户使用方便,通常需要建立安装程序。Delphi 6.0 所提供的 InstallShield Express 可以为一个应用程序创建安装程序,以便将与该应用程序有关的文件自动地拷入新的系统中,并对系统进行一些必要的设置。在这一节中,详细说明如何为一个应用程序创建安装程序。

运行 InstallShield Express 后,将出现如图 7.8 所示的窗口。窗口中央的几个选项的作用如下。

- Welcome——显示欢迎界面,给出相关的帮助信息。



图 7.8 InstallShield Express 的运行界面

- Create a new project——用于创建一个新的安装程序。
- Open a project——用于打开一个已存在的安装程序。
- InstallShield on the Web——用于在互连网上获取 InstallShield 的最新信息。

- Open a project——用于打开一个已存在的安装程序。

- InstallShield on the Web——用于在互连网上获取 InstallShield 的最新信息。

要创建一个新的安装程序,首先单击 **Create a new project** 选项,在 **Project Name and Location** 中录入要建立的安装程序名称,然后单击 **Create** 按钮,弹出如图 7.9 所示的窗口。在该窗口中按左窗格的项目依次设置,最后即可生成安装程序。



图 7.9 InstallShield Express 的设置窗口

下面逐一说明各个选项的作用。



1. Organize Your Setup

主要用于设置软件的作者、版本、安装位置、安装类型等信息。主要有以下三个选项：

- General Information

用于设置应用程序的 Author(作者)、Product Name(软件名称)、Product Version(软件版本信息)、INSTALLDIR(安装目录)、Publisher(出版公司)、DATABASEDIR(所用数据库的目录)、Product Code(产品序列号)等信息。

- Features

用于设置安装程序的功能部件。当应用程序功能比较多时,可以将其分成多个功能模块,这样用户在安装时,可以根据自己的需要有选择的安装其中的模块,使应用程序的使用更加灵活。具体实现方法:在 Features 目录树上右击 Features(或其他已建立的功能部件名),单击快捷菜单 New Feature Ins.,然后输入新建功能部件的名称,并设置其特性参数即可。

- Setup Types

用于设定安装的类型。常用的安装类型有:Typical(典型安装,安装所有的程序)、Minimal(最小化安装、仅安装必要的文件)、Custom(自定义安装,按用户的选择安装)。

2. Specify Application Data

用于设置与该应用程序相关的文件及数据。

- Files

用于设置应用程序文件。具体步骤:单击 Files 项,在 Feature 中选择功能部件名;在 Source Computer's Folder(源机文件夹)中选择应用程序所在的文件夹;在 Source Computer's Files 中选择应用程序名,单击右键,再在快捷菜单中选择“复制”,将应用程序复制到剪贴板上;在 Destination Computer's Folder(目标机文件夹)中,右击 Destination Computer 项,在快捷菜单中利用 Show Predefined Folder 选择已定义的文件夹,或利用 Add 建立新的文件夹;选中目标机上的某个文件夹(将要存放安装程序的文件夹),在 Destination Computer's Files 窗格中单击右键,选择“粘贴”,将应用程序复制到此位置。

- Files and Features

用于浏览或修改文件及功能部件的设置。

- Objects/Merge Modules

用于设定在制作安装程序过程中,除应用程序之外的其他辅助程序或模块。例如在制作与报表有关的安装程序时,需要加载 QuickReport 模块;在制作与 Internet 应用相关的安装程序时,需要加载 Internet 模块等。

3. Configure the Target System

用于配置将要运行安装程序的目标系统。其主要选项如下:

- Shortcuts/Folders



用于设置目标机上应用程序的快捷方式及快捷方式所在的文件夹。默认的位置有：Task Bar(任务栏)、Start Menu(开始菜单)、Programs Menu(程序菜单)、Startup(运行)、Send to(发送到)、Desktop(桌面)。若欲建立应用程序的快捷方式，可以在某个默认的位置名称上右击，然后选择 New Shortcut 项，并输入快捷方式名称即可。当然也可以在某个默认的位置名称上右击，然后选择 New Folder 项，建立新文件夹名称，再在新的文件夹下建立快捷方式。

- Registry

用于设置注册表信息。

- ODBC Resources

用于设置 ODBC 驱动程序及数据源。

- INI File Changes

用于设置系统的初始化文件(.ini)。

- File Extensions

用于设置与应用程序项关联的文件扩展名。这样当用户使用该类型文件时，将自动运行该应用程序。

4. Customize the Setup Appearance

用于设置安装程序的界面。选择 Dialogs 项后，在弹出的 Dialogs 目录树中包括一些复选项，可以通过设置复选项实现安装程序界面的定制。各复选项的作用如下：Splash Bitmap(确定安装程序封面的位图)、Install Welcome(安装欢迎界面)、Licence Agreement(许可协议界面)、Readme(自述文件)、Customer Information(用户信息设置界面)、Destination Folder(目标文件夹设置界面)、Database Folder(数据库文件夹设置界面)、Setup Type(安装类型设置界面)、Custom Setup(自定义安装界面)、Ready to Install(安装设置确认界面)、Setup Progress(安装进度界面)、Setup Complete Success(安装完成界面)。

5. Define Setup Requirements and Actions

用于设定安装程序的必要条件和行为。如果应用程序对计算机的软、硬件环境有要求，可以通过 Requirements 项来设置，这样在用户安装过程中能够提示这些信息。这里可供选择的有：Os(操作系统)、Processor(处理器)、RAM(内存)、Resolution(显示器的分辨率)和 Color(颜色)等。

6. Prepare for Release

用于编译、测试及运行新建的安装程序。

当选择 Build Your Release 项后，会弹出 Builds 目录树，其下包括可以存放安装程序的各种介质：CD-ROM、DVD、Custom(自定义介质)、SingleImage(单一映象)。这时，可以单击某一介质，在出现的列表中对介质的相关特性进行设置；还可以设置是否压缩安装文件、是否包含 Setup.EXE 文件、是否自动运行等功能。当介质设置完毕，可以在介质上单击右键，选择快捷菜单的 Build 项，或在 Build 菜单中单击 Build Custom 菜单项，对上述安装程序的设定进行编译，以生成安装程序。如果设置无误，则会显示 Express\Custom build completed with 0 er-



rors, 0 warnings;若发现错误,则需要重新设置对应的选项,然后再编译。

当编译通过后,可以选择 Test Your Release 项后,单击相应的介质名称,选择 Test Your Setup(或是单击 Build|Test Custom 菜单项)测试安装程序,这时屏幕上将依次显示该安装程序实际运行时的具体过程。如果有不满意的地方,还可以重复前面步骤重新设置。当然,若用户单击介质名称后,选择 Run Your Setup(或是单击 Build|Run Custom 菜单项)会实现在本机上运行此安装程序的过程。

当安装程序测试满意后,可以通过 Distribute Your Release 项将安装程序发布到对应的介质上。具体步骤:在 Distribute View 窗格中单击 Browse 按钮,选择安装文件要存放的位置;单击 Distribute to Location 按钮,安装程序制作成功(注意安装程序是一组文件的集合)。

DP 7.3 学习 Delphi 的一些技巧

由于篇幅有限,本书只是介绍了 Delphi 的一些基础知识,Delphi 的功能远不止于此,还有待于读者去进一步学习。用户在学习过程中,遇到困难,可以参阅各种参考手册,也可以借助以下几种方法。

7.3.1 使用联机帮助

启动 Delphi 的联机帮助功能。比如要了解 Button 组件的 Caption 特性,可以在帮助窗口的索引标签页的输入框内输入该组件的类名称 TButton(一般是组件名前面加 T)后,单击“显示”,这时会显示与该组件有关的各种信息(Hierarchy 继承关系、Properties 特性、Methods 方法、Events 事件)。例如单击 Properties 之后,弹出一个窗口显示了 Button 组件的所有特性;选择 Caption 后,会显示有关该特性的说明;若用户单击 Example,还能看到有关 Caption 特性的举例程序。当然,Delphi 提供的是英文帮助,阅读起来不是很方便。

7.3.2 查看 Delphi 提供的例程

学习编程的一个好方法是大量的阅读程序。Delphi 提供了不少现成的例程,用户可以通过阅读这些程序提高编程能力。这些例程保存在 Delphi 的 Demos 目录中,用户可以直接打开相应的程序,阅读程序代码,观察运行结果。

7.3.3 获取 Delphi 的源代码

Delphi 提供了许多组件及其特性、方法的源代码。当用户的编程能力达到一定程度,阅读这些源代码有助于了解 Delphi 程序的内部运行机制,有助于用户自己构造组件。这些源代码保存在 Delphi 的 Source 目录下,用户找到对应代码文件名后,直接运行即可看到。用户也可以在代码编辑器中,按下 Ctrl 键,将鼠标指向要查看源代码的组件特性、方法或类名标识符,这时标识符变色,鼠标变成手状,单击该标识符,Delphi 会自动指向该标识符的源代码位置。



DP 习题与思考题

- 7.1 制作帮助文件的基本步骤是什么？
- 7.2 为第 2 章中的“文本编辑器”程序添加帮助功能。
- 7.3 InstallShield Express 的主要功能是什么？
- 7.4 为第 2 章中的“文本编辑器”程序制作安装程序。

下 篇

面向对象程序设计基础

第 8 章 PASCAL 语言概述

第 9 章 控制语句

第 10 章 过程与函数

第 11 章 构造类型

第 12 章 面向对象程序设计基础



第 8 章

PASCAL 语言概述

DP 8.1 PASCAL 源程序结构

一个 PASCAL 程序类似一篇英文文章。英文文章是由字母、数字、标点符号和空格等按英语语法组成的;PASCAL 程序也是由字母、数字和特殊符号按照它的一套语法规则组成的。

为了使初学者从一开始对 PASCAL 程序的结构建立一个整体概念,先介绍一个简单的 PASCAL 源程序。

```
program Example(input,output);    {程序首部}
var
  Len,Wid,Area:integer;           {变量说明}
begin
  writeln('Calcuat Area');
  write('Input The Length:');
  readln(Len);                    {输入 Len 值}
  write('Input The Width:');
  readln(Wid);                   {输入 Wid 值}
  Area:=Len*Wid;
  writeln('Area=',Area);          {输出 Area}
end.
```

说明部分

程序体

语句部分

这是一个计算长方形面积的 PASCAL 语言源程序。先从键盘输入长方形的长和宽,经计算,由显示器上显示出“Area=”字符和长方形面积值。如长输入 5,宽输入 4,显示器上将显示:Area=20。

从上述简单程序中,可以看出 PASCAL 源程序是由程序首部、后面跟着的分号“;”、程序体和句号“.”组成。TURBO PASCAL 6.0 允许在程序首部和程序体之间加 USES 子句,说明被程序直接和间接使用的单元。

其中程序体又由程序说明部分和语句部分组成。

8.1.1 程序首部

程序首部以保留字 program 开头,接着给出程序名。在上面程序例子中,程序名为 Example。在圆括号内列出程序的参数,如 Input、Output、Lst 等。

程序首部的一般形式为



program<标识符>(参数表)

8.1.2 程序说明部分

程序说明部分,是对数据和过程的描述。说明所有在该程序体的语句部分中(以及包含在该程序体中的其他程序体)出现的标识符。程序说明部分又分为五个部分:

- (1)标号说明部分;
- (2)常量定义部分;
- (3)类型定义部分;
- (4)变量说明部分;
- (5)过程与函数说明部分。

标准 PASCAL 规定,上述每一部分只能出现一次,或者不出现,并且必须严格按上述顺序出现。但是,TURBO PASCAL 允许这些部分出现任意次,或者不出现;也允许它们以任何顺序出现。

在上面的简单例子中,仅有变量说明部分。

8.1.3 语句部分

语句部分是程序体的最后一部分。语句部分以保留字 begin 开始,以 end 结束,中间含有一系列语句。语句之间用分号“;”分隔,每个语句执行一定的功能,完成一定的任务。在语句部分的 end 后必须加句号“.”,表示整个程序结束。



8.2 字符集和符号

每种程序设计语言,都有一套自己的基本符号。TURBO PASCAL 基本符号由三部分组成。

8.2.1 基本字符

字母:A 到 Z,a 到 z 和 (下画线)。

数字:0、1、2、3、4、5、6、7、8、9。

特殊字符:+ - * / = ^ <> [] {} . : ; ' # \$ @。

赋值运算符: :=。

关系运算符: <> <= >= < >。

子域分隔符: ..。

括号:(和)可以用来代替[和]。

注释号:(* 和 *)可以用来代替{和}。



8.2.2 符 号

1. 保留字

保留字是由字母拼成的字。PASCAL 语言预先规定了它们的意义,不允许被再定义,因此不能用作下小节介绍的用户自定义的标识符。

下面列出保留字的一部分。

* absotue	* external	nil	* shl
and	file	not	* shr
array	forward	overlay	* string
begin	for	of	then
case	function	or	type
const	goto	packed	to
div	* inline	procedure	until
do	if	program	var
downto	in	record	with
end	mod	set	* xor

其中不带 * 号的是标准 PASCAL 的保留字。

2. 标识符

标识符是用来表示程序、常量、变量、过程、函数、文件、类型等名称的符号。标识符必须以字母开头,后面可以是字母和数字的任意组合。TURBO PASCAL 中允许下画线“_”按字母对待,在标识符中使用。标识符长度是任意的,前 63 个字符有效。其中不允许有空格。

当出现几个同名标识符时,为了区分它们,必须用单元标识作限定。例如,用单元标识符 `unitname` 限定标识符 `exams`,可写成 `unitname.exams`。这种标识符称为限定标识符。

标识符可以分标准标识符和用户定义的标识符。

标准标识符是预先给标准函数、标准过程、标准常量、标准类型、标准文件定义的标识符,即 PASCAL 语言已确定的标准标识符的名和性质,如标准输入文件 `input`、标准输出文件 `output` 等,在书写程序中可直接使用,不必事先定义。

下面列出标准标识符的一部分。

标准常量: `false`、`true`、`maxint`;

标准类型: `integer`、`real`、`boolean`、`char`、`text`;

标准文件: `input`、`output`;

标准函数: `abs`、`arctan`、`chr`、`cos`、`eof`、`eoln`、`exp`、`ln`、`odd`、`ord`、`pred`、`round`、`sin`、`sqr`、`sqrt`、`succ`、`trunc`;

标准过程: `read`、`readln`、`writeln`、`write`、`page`、`pack`、`unpack`、`reset`、`rewrite`、`new`、`dispose`、`get`、`gotoxy`、`put`。



用户自定义的标识符是用户按标识符定义的规则自己定义的,常选用说明某种意义的英文单词。用户标识符必须事先定义或说明,才能在程序中使用它。用户自定义的标识符不能用保留字,最好也不要选用标准标识符。

8.2.3 分隔符

在 PASCAL 语言中,把空格符、行结束符作为分隔符。规定程序中任何两个相邻的保留字、标识符、数或符号之间都必须插入一个或几个分隔符。但在一个保留字、标识符或数的内部不允许出现分隔符。另外逗号“,”、分号“;”、括号“()”、子域分隔符“.”和注释等也可起分隔符作用。

PASCAL 语句之间也要分隔开,一般用分号“;”分隔。

一个程序行最多有 127 个字符。



8.3 数据类型的概念

PASCAL 语言建立了数据类型的概念,对描述的数据进行分类。每一种数据类型定义了一个具有相同性质的数据集合。各种数据类型的数据具有不同的性质。程序中所用到的每一个数据,包括常量和变量,都有一个和它相联系的类型。由此,决定了数据所能具有的值,也决定了对该数据所能进行的操作。

PASCAL 语言中数据具有丰富的类型。按它们的特点可以分为简单类型、构造类型、指针类型三大类,如图 8.1 所示。

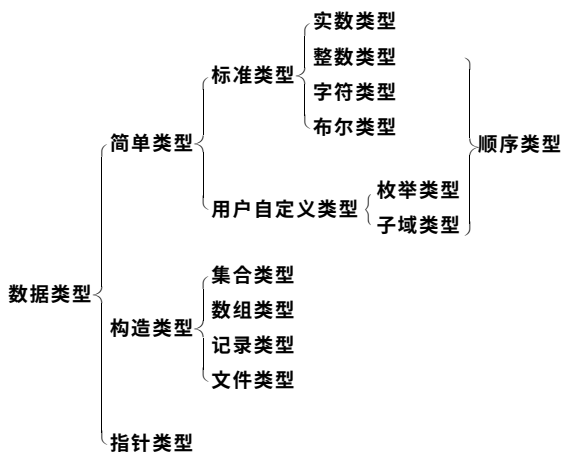


图 8.1 PASCAL 语言数据类型

图中标准类型用 PASCAL 语言系统预先定义的标准标识符表示;整数类型用 `integer` 表示;实数类型用 `real` 表示;布尔类型用 `boolean` 表示;字符类型用 `char` 表示。

本章将详细介绍标准类型,其他类型在以后各章中介绍。



DP 8.4 常量与变量

8.4.1 常 量

常量是指在程序中使用的一些具体的整数类型数、实数类型数和字符串。

为了提高程序的可读性和使程序便于修改,在程序中往往用一些标识符来代表具体的常量。为此,要按一定的语法形式定义它。

8.4.2 常量定义

在 PASCAL 语言中,可以给一些常量取个名字,用一个标识符代表它,这就是常量定义。

例如: `cost=60; blank=''`。

经常量定义的标识符,有时称为常量标识符。

在 PASCAL 语言中,常量定义要写在常量定义部分中。

常量定义部分的一般形式:

`const`

```
<常量标识符 1>=<常量 1>;  
<常量标识符 2>=<常量 2>;  
      :  
      :  
<常量标识符 n>=<常量 n>;
```

`const` 是保留字,表示开始一个常量定义部分,其后有若干个常量定义。这些常量定义之间要用“;”号分隔。

PASCAL 语言对常量定义有如下要求:

- (1) 常量定义要放在常量定义部分,即程序的首部之后,语句部分之前;
- (2) 必须遵循先定义后使用的原则,即只有定义了常量标识符,以后在程序中才能使用。

一般称在常量定义中等号左面的常量标识符为定义性出现,以后在程序中的引用为使用性出现。

8.4.3 变量说明

在程序执行过程中其值可以改变的数据,称为变量。每个变量都要有一个名称,这就是变量名。变量名由用户自己定义,但必须符合标识符的规定。

在一个程序中,一个变量只能属于一种确定的类型。因此,程序中出现的每个变量,都必须说明其数据类型。

其中类型,可以是标准数据类型标识符 `integer`、`real`、`char` 和 `boolean`,也可以是用户自定义的各种类型。



变量说明形式是：一个变量标识符或由逗号隔开的多个变量标识符，在它的冒号“:”后面说明其数据类型。

在 PASCAL 程序中，变量说明要写在变量说明部分中。

变量说明部分的一般格式：

```
var
    <变量说明 1>;
    <变量说明 2>;
    :
    <变量说明 n>;
```

其中 var 是保留字，表示开始一个变量说明部分。一个 var 可以含有多个不同的变量说明，每个变量说明之间用分号隔开。有时称被分隔开的变量说明为变量说明项。

DP 8.5 简单数据类型

8.5.1 整数类型

用标准标识符 integer 标明整数类型，简称整型。整型的数据可以是正整数、负整数和整数零。

PASCAL 中的整型常数必须按规定严格书写。一台计算机由于字长的限制只能表示整数的一个有限子集。整型的数据 N 取值范围为

$$-\text{maxint}-1 \leq N \leq \text{maxint}$$

前已说明 maxint 是标准常量标识符，表示机器所能表示的最大整数。不同字长的计算机，其 maxint 值也不同。由于整数在计算机内采用补码表示，maxint 可用公式表为

$$\text{maxint} = 2^{w-1} - 1$$

式中，w 为字长，TURBO PASCAL 中整型数用两个字节的补码表示，即用 2B 存放一个整数，因此 N 取值范围为

$$-32768 \leq N \leq 32767$$

在程序运行中，如果整型变量超过这个范围，即发生溢出，程序将终止。因此，当估计到程序中变量会超出 maxint 时，就不能用整数类型；而应采用实数类型。

在 TURBO PASCAL 语言中，还允许整型数采用十六进制，在数字前加 \$ 表示十六进制整数。如 \$10，表示：\$(10)_{16} = (16)_{10}\$；\$1A，表示：\$(1A)_{16} = (26)_{10}\$

PASCAL 语言规定可以对整型数据进行如下操作：

1. 允许两个整型数使用下列运算符进行运算

(1) 算术运算符

算术运算符 +、-、*、div、mod 它们分别表示加、减、乘、整除和取余。这五种运



算,要参加运算的两个数都是整型,运算结果也是整型。前三种运算与一般的算术运算加、减、乘相同。div 整除运算,是两个整型数相除,取整数部分(商的整数部分),得到整型结果。mod 取余运算,是两个整型数相除,取余数。余数的符号与被除数符号相同。

(2) 关系运算符

关系运算符 =、<>、<、>、<=、>= 分别表示等于、不等于、小于、大于、小于等于和大于等于。关系运算的结果只有两种可能:成立或不成立。这一结果在 PASCAL 中用两个特殊常量真(true)或假(false)表示。标识符 false 和 true 是布尔类型的值。关系运算结果是布尔类型的值。

2. 标准函数

PASCAL 语言提供了自变量为整型量的标准函数,有顺序函数、算术函数和转换函数等。

标准函数是 PASCAL 语言预先定义的。它们实际上是能完成特定功能的称为子程序的程序段。每个标准函数都用一个标识符来标识。每个标准函数都能完成一个特定的功能,在程序中可以直接调用它们。PASCAL 语言中某些标准函数与数学中函数有相似之处。整数类型是顺序类型,即所有的整型数都是按一定的顺序排列的。如 3 的后序数是 4,350 的后序数是 351。以后介绍的布尔类型、字符类型、枚举类型和子域类型等都是顺序类型。顺序函数可以对顺序类型数据进行操作。

前趋函数 pred(x),函数值为 $x-1$ 。例如:

pred(6)=5, pred(-7)=-8

pred(-maxint-1) 无意义

后继函数 succ(x),函数值为 $x+1$ 。例如:

succ(7)=8, succ(-10)=-9

succ(maxint) 无意义

绝对值函数 abs(x),函数值为 $|x|$ 。例如:

abs(-3)=3

平方函数 sqr(x),函数值为 x^2 。例如:

sqr(-4)=16, sqr(3)=9

以上四个函数的结果仍是整型数。

奇函数 odd(x),函数的结果是布尔型。当 x 为奇数时,函数值为 true;当 x 为偶数时,函数值为 false。例如:

odd(3)=true, odd(4)=false

字符函数 chr(x),函数值是序号为 x 的 ASCII 字符,是字符型。例如:

chr(66)='B'(B 在 ASCII 码表中的序号为 66)。

8.5.2 实数类型

用标准标识符 real 标明实数类型,简称实型。实型数据可以是正实数、负实数和零实型数据,一般用小数和指数形式(亦称科学表示法)表示。例如:



+986.33;3.5E+8($=3.5 \times 10^8$)
-0.5E-3($=-0.5 \times 10^{-3}$)
-20.0;-41E3($=-41 \times 10^3$)
0.0

都是合法实型数。

实型数的取值范围是绝对值在 $1\text{E}-38$ 到 $1\text{E}+38$ 之间,尾数可以有 11 位有效数字。

实型的一个数据在内存中占 6B。

PASCAL 语言规定可以对实型数据进行如下操作。

1. 允许实型数使用下列运算符进行运算

运算符 +、-、*、/ 分别表示加、减、乘和除。对这几个运算符,只要一个运算对象是实型,则结果类型就是实型。“/”叫实数除,即使两个整型数相除,其结果也总是实型。例如:

$$7/2=3.5, 6/3=2.0$$

另外 6 个关系运算符均适用于实型。它们的运算对象还可以一个是实型,另一个是整型,结果仍是布尔类型。

2. PASCAL 语言提供了自变量为实型的标准函数

(1) 算术函数

在下列算术函数中,x 可以是实型或整型数。对于函数 abs 和 sqr,其结果类型和变量 x 的类型相同,其他算术函数的结果类型都是实型。

绝对值函数 abs(x)——函数值为 x 绝对值;

平方函数 sqr(x)——函数值为 x 的平方;

正弦函数 sin(x)——函数值为 x 的正弦,其中 x 的单位为弧度;

余弦函数 cos(x)——函数值为 x 的余弦,其中 x 的单位为弧度;

指数函数 exp(x)——函数值为指数 e^x ;

对数函数 ln(x)——函数值为 x 的自然对数;

平方根函数 sqrt(x)——函数值为 x 的平方根;

反正切函数 arctan(x)——函数值为 x 的反正切,单位为弧度。

(2) 转换函数

在下面的转换函数中,x 是实型数,函数的结果类型是整型。

截尾函数 trunc(x)——舍去实数 x 的小数部分,并转换为整型。

舍入函数 round(x)——对实数 x 作四舍五入,并转换为整型。

8.5.3 字符类型

141

用标准标识符 char 标明字符类型。字符类型数据可以是字母、符号、数字(0~9)等 ASCII 所有字符。

在程序中,用单引号括起来的字符表示字符类型数据。例如,'B'、'D'、'3'、'7'、'-'、' ',分别表示字母 B、D,数字 3、7,字符 - 和空格。



但目前使用最广泛的是 ASCII 码字符集。下面主要针对 ASCII 码进行讨论。

由于采用 ASCII 码,字符按着 ASCII 码序号排列。这样字符与 ASCII 码序号有一一对应的映射关系。字符类型也是顺序类型。

PASCAL 语言规定可以对字符类型数据进行如下操作:

(1) PASCAL 语言允许字符类型数据使用 6 个关系运算符进行运算操作,运算结果是布尔类型。

(2) PASCAL 语言提供如下自变量为字符类型的标准函数,其中 ch 为字符类型。

后继函数 succ(ch),例如:

$\text{succ}('5') = '6', \text{succ}('B') = 'C'$

前趋函数 pred(ch),例如:

$\text{pred}('6') = '5', \text{pred}('C') = 'B'$

序数函数 ord(ch):给出字符 ch 在 ASCII 字符集中的序号,结果为整型。

8.5.4 布尔类型

用标准标识符 boolean 标明布尔类型。布尔类型数据的值只有: true(逻辑真)和 false(逻辑假)。布尔类型是顺序类型。规定 $\text{false} < \text{true}$, false 的序号为 0, true 序号为 1。

PASCAL 语言规定可以对布尔类型数据进行如下操作:

(1) 允许两个布尔类型数据使用下列逻辑运算符: not{逻辑非}、and{逻辑与}、or{逻辑或}。逻辑运算符运算对象都是布尔型,结果也是布尔型。TURBO PASCAL 还包含一个异或运算符: xor。

(2) 布尔型数据可以使用 6 个关系运算符进行运算,结果仍是布尔类型值。

(3) 对布尔类型值,可以使用如下标准函数,其中 B 为布尔类型数值:

ord(B)

pred(B)

succ(B)

除了上面 3 个标准函数外,还有 2 个标准函数值是布尔型: eof(f) 和 eoln(f)。其中 f 是文件变量,若输入文件是标准文件 input,可写成 eof 或 eoln。这两个函数的值在开始读文件时为 false。当读到一个行结束符时(回车符 chr(13)), eoln 为 true。当读到文件结束符时(chr(26)或 ctrl-z)时 eof 为 true。

8.5.5 枚举类型的定义、运算规则和输入、输出方法

1. 枚举类型的定义

枚举类型是简单类型的用户自定义类型。枚举类型的定义方法是通过枚举一系列有序的用户自定义标识符来实现的。每一个用户自定义标识符表示一个数据值,称为枚举类型元素。枚举类型的定义要放在以保留字 type 开始的类型定义部分。

枚举类型定义部分的一般形式为



type

⟨类型标识符⟩ = (⟨标识符 1⟩, ⟨标识符 2⟩, …, ⟨标识符 n⟩);

从枚举类型定义的一般形式可以看出,枚举类型的类型标识符和枚举类型元素标识符都要由用户自己定义。定义标识符时,必须符合 PASCAL 语言用户自定义标识符的语法规则。另外,要注意括号内的元素标识符之间用逗号隔开。

例如,定义枚举类型,表示一星期中的每一天:

type

Week = (SUN, MON, TUE, WED, THU, FRI, SAT)

上面定义 Week 为枚举类型。Week 有 7 个枚举类型元素,表示一星期中的每一天。

在上面的类型定义之后,类型标识符 Week 代表了一种新的类型——枚举类型,其使用方法与标准类型的使用方法类似。可以在变量说明部分说明某变量的类型为已经定义了的枚举类型,例如:

var

Day1, Day2: Week;

在上面的变量说明部分,变量 Day1 和 Day2 被说明为枚举类型 Week。

PASCAL 语言还允许将类型定义和变量说明合并写为

var

Day1, Day2: (SUN, MON, TUE, WED, THU, FRI, SAT);

枚举类型在定义时应注意以下几点:

(1) 类型标识符与枚举类型元素标识符要符合 PASCAL 语言用户自定义标识符语法规则,不得使用保留字和标准标识符,否则在编译时将出现错误或造成混乱。

(2) 作为枚举类型元素的标识符不允许双重定义。

(3) 枚举类型变量不允许直接通过 read 过程输入,也不能通过 write 过程输出。例如:

readln Day1;

是错误的。

2. 枚举类型的运算规则

枚举类型数据主要可以做以下运算:

(1) 赋值运算

在枚举类型的类型定义时,列出了该枚举类型的各个元素,因而枚举类型变量的取值范围就是定义该枚举类型时所列出的各个枚举类型元素。对枚举类型变量的赋值,就是将某一枚举类型元素赋给该类型的变量。例如,有以下说明:

type

Color1 = (Red, Yellow, Blue);

Color2 = (Black, White);

var



```
C1,C2,C3:Color1;  
C4:Color2;
```

下列赋值是合法的:

```
C1:=Red;  
C2:=Yellow;  
C3:=C1;
```

而如下赋值是非法:

```
C1:=Green;  
C4:=Red;
```

因为赋给枚举类型变量 C1 的值 Green 不是枚举类型 Color1 的元素,赋给枚举变量 C4 的值 Red 与 C4 不属于同一枚举类型。这两个赋值操作都超出了相应的枚举类型变量的取值范围,属于非法赋值。

(2) 关系运算

PASCAL 语言规定,枚举类型的各个元素从左到右各分配一个整数序号,第一个元素的序号为 0,第二个元素的序号为 1……依次类推,例如:

```
Week=(SUN,MON,TUE,WED,THU,FRI,SAT);
```

其中 SUN 的序号为 0,MON 的序号为 1…SAT 的序号为 6。

由于每个枚举类型元素都具有整数序号,故可以对它们进行大于、等于之类的比较运算(也称关系运算)。这些关系运算的运算符共有 6 种:

= (等于)	<> (不等于)
< (小于)	<= (小于等于)
> (大于)	>= (大于等于)

枚举类型数据(变量或元素)关系运算的结果为布尔类型。例如表达式 MON<TUE、SAT>FRI 和 SUN<MON 的值都是 true;表达式 MON>TUE、SUN>SAT 和 SAT<FRI 的值都是 false。简单类型的值是有序的。我们把具有整数序号的类型称为顺序类型。实数类型、整数类型、字符类型和布尔类型的值都是有序的,枚举类型的值也是有序的。整数类型、字符类型和布尔类型的值都具有整数序号,都是顺序类型;枚举类型的值也具有整数序号,也是顺序类型。而实数类型的值虽然是有序的,但没有整数序号,不属于顺序类型。

(3) 函数运算

枚举类型数据适合于前趋函数(pred)、后继函数(succ)和序数函数(ord)3 种函数。对前边定义的枚举类型 Week 而言,下列各式的值分别为

```
ord(SUN)的值为 0;  
ord(SAT)的值为 6;  
ord(FRI)=5 的值为 true;  
pred(FRI)的值为 THU;  
pred(FRI)=WED 的值为 false;
```



succ(FRI)的值为 SAT;

succ(FRI)=SAT 的值为 true。

第一个枚举类型元素的前趋函数值和最后一个枚举类型元素的后继函数值没有实际意义。

8.5.6 子域类型的定义和运算规则

1. 子域类型的定义

定义一个类型为任何其他已经定义的顺序类型的子域,这种类型称为子域类型。子域类型的一般形式为

$\langle \text{类型标识符} \rangle = \langle \text{常量 1} \rangle . . \langle \text{常量 2} \rangle ;$

常量 1 称为子域类型的下界;常量 2 称为子域类型的上界。子域类型的上、下界必须属于同一顺序类型。这一顺序类型称为该子域类型的宿主类型,并且下界不能大于上界。只有顺序类型才能有子域,实数类型不能有子域。子域类型本身也是顺序类型。

常用的宿主类型为整数类型、字符类型和枚举类型。

例如,统计某中学学生的年龄可以用宿主类型为整数类型的子域类型;处理英文单词可以用宿主类型为字符类型的子域类型;计算每星期的的工作的工作日可以采用宿主类型为枚举类型的子域类型。定义如下:

```
type
Day=(SUN,MON,TUE,WED,THU,FRI,SAT);
Age=10..25;
Word1='A'..'Z';
Word2='a'..'z';
WorkDay=MON..SAT;
```

在上面的类型定义中,Day 定义为枚举类型,共有 SUN、MON、TUE、WED、THU、FRI、SAT 7 个枚举类型元素。标识符 Age 定义为子域类型,表示某中学学生的年龄。标识符 Word1 和 Word2 分别表示大、小写英文字母,均是子域类型。标识符 WorkDay 被定义为宿主类型为枚举类型的子域类型,表示每星期的工作日 MON~SAT。

通过类型定义后,标识符 Age、Word1、Word2 和 WorkDay 均代表子域类型,其使用与标准类型的使用方法类似。变量说明部分如下:

```
var
StudentAge:Age;
Ch1,Ch2:Word1;
Ch3:Word2;
WD:WorkDay;
```

也可将类型定义和变量说明合并为



```
var
    StudentAge:10..25;
    Ch1,Ch2:'A'..'Z';
    Ch3:'a'..'z';
    Day:(SUN,MON,TUE,WED,THU,FRI,SAT);
    WD:MON..SAT;
```

2. 子域类型的运算

对于子域类型的宿主类型的任何运算,同样也适用于其子域类型。子域类型具有其子域类型的宿主类型的一切性质,只是其取值范围受到约束,只包括上、下界自身在内的一段数据范围。例如:

```
type
    ID=1..10;
    JD=1..100;
    Ch='A'..'Z';
    Day=(SUN,MON,TUE,WED,THU,FRI,SAT);
var
    I:ID;
    J:JD;
    Chs:Ch;
    Days:MON..FRI;
    M:integer;
    Da:3..3;
    Cb:'B'..'B';
```

子域类型变量 I 的取值范围是 1~10 这 10 个数, J 的取值范围是 1~100 这 100 个数, Da 的取值范围只有 3, 变量 Chs 的取值范围是包括字母 A 和 Z 在内的 26 个大写英文字母, Cb 的取值范围只能是字母 B。

从取值范围来看,下列语句是合法的,即

```
I:=5;
Chs:='M';
Days:=WED;
J:=I+10;
M:=J*I+ord(Chs)+ord(Days)+5;
```

而下面两个赋值语句,分别超出了子域类型变量 I 和 J 的取值范围,属于非法语句,即

```
J:=J*1000;
I:=I*J+20;
```

这种超界赋值,会使程序出现运行错误。但许多 PASCAL 系统一般不做超界检查。这种错误往往不能准确地被检查出来。在 TURBO PASCAL 中,在设置相应的编译器指示后才进行超界检查。



当子域类型数据作为函数的自变量时,取决于该子域类型的宿主类型。例如,当变量 Chs 的值为 M 时,ord(Chs) 的值为 77,pred(Chs) 的值为 'L',succ(Chs) 的值为 'N';变量 Days 的值为 WED 时,ord(Days) 的值为 3,pred(Days) 的值为 TUE,succ(Days) 的值为 THU;变量 I 中的值为 5 时,ord(I) 的值仍为 5,odd(I) 的值为 true,sqr(I) 的值为 25。

子域类型变量的输入/输出操作取决于所对应的宿主类型。宿主类型为整数类型或字符类型时,该子域类型变量可以使用 read(或 readln)和 write(或 writeln)过程直接输入/输出;宿主类型为枚举类型时,该子域类型变量不能用 read(或 readln)和 write(或 writeln)过程直接输入/输出,但可以采用与枚举类型变量相同的输入/输出方法。

8.5.7 类型间的关系

1. 类型一致

把变量说明为属于某类型时有两种方法:一是先进行类型定义,然后用该类型标识符来说明变量的类型;另一种方法是用类型直接说明变量。为讨论类型相容、赋值相容和过程的参数传递等问题的方便,先引入变量类型一致的概念。

标准 PASCAL 规定,满足下列条件之一就属于两个或多个变量类型一致,即

- (1) 在变量说明中被同一类型标识符说明的两个或多个变量;
- (2) 在同一变量说明项里被说明成某一确定类型的各个变量;
- (3) T1、T2 虽为不同的标识符,但已在类型定义部分用 $T1 = T2$ 说明相等,用 T1、T2 说明的变量。

假如有如下说明:

```
type
    t1 = (one, two, three);
    t2 = t1;
    t3 = 1..31;
var
    a: t1;
    b: t2;
    c, d: t3;
    e, f, g: 1..31;
    h, i: t3;
```

c、d、h、i 类型一致,因为它们被同一标识符 t3 说明(条件(1))。e、f、g 在同一变量说明语句中说明,其类型一致(条件(2))。由于有类型说明 $t2 = t1$,根据第(3)条,a 和 b 类型一致。但 c、d、h、i 和 e、f、g 之间不能满足上述三条中的任何一条,它们不是类型一致的变量。

2. 类型相容

两个类型若满足下列条件之一,就称为类型相容:

- (1) 它们是一致的类型;



- (2) 它们是同一类型的子域,或其中一个是另一个类型的子域;
- (3) 它们是有相容的基类型的集合类型;
- (4) 它们是具有同一长度的串类型。

如果有以下说明:

type

```
L=0..50; W=L;  
ch1='A'..F; ch2='A'..Z; I=0..10; J=5..100;
```

则类型 W 和类型 L、ch1 和 ch2、I 和 J 都分别是类型相容的。

3. 赋值相容

若满足下列条件之一,就说表达式 E 的类型对变量 T 的类型赋值相容:

- (1) E 与 T 的类型一致,且为非文件类型;
- (2) T 为实数类型,E 为整数类型或为整数类型的子域类型;
- (3) E 和 T 是相容的简单类型,且 E 的值是 T 所允许的值;
- (4) E 和 T 是相容的集合类型,且 E 的每一个元素都是 T 的基类型所允许的值;
- (5) E 和 T 是相容的串类型。

4. 类型一致、相容和赋值相容概念的应用

类型一致、相容和赋值相容是三个不同的概念,它们之间不存在蕴含关系,但有一定的交叉。在程序中,不同的地方对类型之间的关系有不同的要求,若违背了这种要求将导致程序失败。在程序应用中,类型一致、类型相容和类型赋值相容主要应注意以下几点:

- (1) 在进行算术运算、逻辑运算、关系运算及集合运算时,要求参加运算的量的类型相容。
- (2) 在赋值语句、读语句、FOR 语句中要求类型赋值相容。
- (3) 在过程与函数调用中,如果参数是变量参数,则要求实在参数与对应的形式参数之间类型一致;如果参数是值参数,则要求实在参数与对应的形式参数类型赋值相容。

一般情况下,PASCAL 系统在编译时发现一致性和相容性方面的错误,在运行时发现赋值相容性方面的错误。

DP 8.6 表达式与赋值语句

8.6.1 表达式

表达式是一种算法结构,它指出运算规则。程序中所有的运算都是在表达式中完成的。它与数学中的表达式有很多相似之处。PASCAL 语言中的表达式由运算符和运算对象组成。

运算对象包括常量、变量、函数、集合和圆括号内的表达式。例如, $a+b-a*(m+n)$ 就是一个表达式。



表达式用来计算值。根据值的类型,PASCAL 语言中表达式可分为:算术表达式(值是实数类型或整数类型)、布尔表达式(值是布尔型)、字符表达式(值是字符类型或字符串类型)和集合表达式(值是集合类型)。

1. 运算符

PASCAL 语言有 4 种运算符。

- (1) 一元运算符:逻辑非运算符 not,对布尔型运算对象取反。
- (2) 乘法运算符,如表 8.1 所列。

表 8.1 乘法运算符

运算符	运 算	运算对象类型	结果类型	例 子
*	乘	整型、实型	整型、实型	5.0 * 2 = 10.0
/	除	整型、实型	实型	125 / 12.5 = 10.0
div	整数除	整型	整型	5 div 3 = 1
mod	取余	整型	整型	5 mod 3 = 2
and	逻辑与	布尔型	布尔型	true and false = false

- (3) 加法运算符,如表 8.2 所列。

表 8.2 加法运算符

运算符	运 算	运算对象类型	结果类型	例 子
+	加	实型、整型	整型、实型	3.5 + 6 = 9.5
-	减	实型、整型	整型、实型	46 - 5.5 = 40.5
or	逻辑或	布尔型	布尔型	true or false = true
xor	逻辑异或	布尔型	布尔型	true xor false = true

(4) 关系运算符。关系运算符可以用于任何标准类型常量、变量的运算。整数类型和实数类型的运算对象可以混用。关系运算符运算结果总是布尔型,即 true 或 false。关系运算如下:

=	等于	<	小于
<>	不等于	>=	大于等于
>	大于	<=	小于等于
in	集合成员检查		

2. 优先级规则

优先级规则,指明了表达式运算的顺序。基本规则有三条:

- (1) 运算符优先级从高向低的次序是:一元运算符、乘法运算符、加法运算符、关系运算符。运算时先进行优先级高的运算。
- (2) 具有同一优先级的运算符是按照自左至右依次执行。
- (3) 括号中的表达式优先运算。有多层括号的,按自内向外的次序执行。

一般的,若表达式中出现的变量和函数都是算术型的(整型或实型),且出现的运算符也都是算术运算符,则这样的表达式是算术表达式;若表达式中出现布尔型变量、布尔运算符或关



系运算符,则是一个布尔表达式。

整型表达式可以对整型或实型的变量赋值。但实型表达式只能对实型变量赋值,布尔型表达式只能对布尔型变量赋值。

3. 表达式的语法单元

PASCAL 语言按照运算符的优先级关系,将表达式又细分为表达式、简单表达式、项、因子等更小的语法单元。运算按由小到大的顺序进行。

表达式是由一个简单表达式或几个简单表达式用一个关系运算符连接而成。如 $x+5=y$ 、 $a>n$ 均为表达式。

简单表达式是一个项或几个项用加法运算符连接而成,如 $a+b$ 、 $-x$ 、 $a*b+y$ 都是简单表达式。

项是一个因子或几个因子用乘法运算符连接而成,如 $x*y$ 、 $a \operatorname{div}(a+b)$ 、 $(4>3)$ 和 $(5<2)$ 都是项。

因子由下列成分之一构成:无符号常量、变量、 $\operatorname{not}<\text{因子}>$ (直接递归定义)、 $<\text{表达式}>$ (间接递归定义)。如 20 、 a 、 $(m+n-b)$ 、 $\operatorname{not} k$ 都是因子。

8.6.2 赋值语句

在程序中,语句是描述要实现的动作。一切语句中,最基本的是赋值语句。它可用来表明赋予某一变量某一确定值。赋值语句由一个变量标识符后面跟上赋值符“:=”和一个表达式构成。赋值语句的一般形式:

$\langle \text{变量标识符} \rangle := \langle \text{表达式} \rangle$

赋值语句可以用于对除文件类型外的任何类型变量赋值,但该变量的类型与赋值符后面的表达式的类型必须相同。严格的说应是赋值相容的。

另外需要指出的是:

- (1) 赋值符“:=”的左面只能是变量,而不允许是表达式,如 $-x:=a+b$ 是非法的;
- (2) 变量可以被多次赋值,赋值后的变量将在程序行中一直保持该值,直到被重新赋成其他的值;
- (3) 被赋值的变量可作为表达式的因子参与运算。

DP 8.7 输入和输出过程

使用计算机解题时,要把数据输入计算机;数据处理后,又要求计算机将计算结果输出。可见,输出/输入是不可缺少的。

TURBO PASCAL 预定义了两个标准文件 input、output。当程序中不做说明时,input 指键盘、output 指显示器。另外 input、output 还可以重定向到其他设备。具体输入和输出,则通过调用标准过程 read 和 write 实现。标准过程是 PASCAL 语言预先说明的。它们实际上



是能完成某种特定功能的称为子程序的程序段,在程序中可以直接调用。标准过程 read 可完成输入功能,标准过程 write 可完成输出功能。

8.7.1 读语句

read 过程和 readln 过程是标准输入过程。在程序中调用这两个标准过程,在程序运行时,完成从输入设备(这里指键盘)给变量输入数据的输入功能。

程序中调用 read 过程和 readln 过程,通常称为读语句。读语句有两种形式:

```
read(<变量名表>);
```

```
readln(<变量名表>);
```

其中变量名表是一串用逗号隔开的变量标识符。

1. read 语句

设 $a_1, a_2, \dots, a_n$ 的类型为 integer、real 或 char 的变量。读语句可以写成:

```
read(a1, a2, ..., an);
```

等价于

```
begin read(a1); read(a2); ... ; read(an) end
```

TURBO PASCAL 语言规定整型和实型变量可以共用一个读语句,但字符型变量不能与整型、实型变量共用一个读语句。

下面解释读语句 read(a):

(1) 当 a 是字符型变量, read(a) 执行时,从键盘读入一个字符,存入变量 a。

这里需特别指出,当程序执行到 read(a) 语句时,程序就停下来,等待从键盘输入数据,然后存入变量 a 中。如不从键盘输入数据,程序一直等下去。下面的整型和实型变量输入也同样。

(2) 当 a 是整型或实型变量, read(a) 执行时,按 PASCAL 语言关于数的语法,从键盘读一个形成一个数的序列,遇到不能形成数的字符(一般用空格符)即停止读入,完成将数存入变量 a 中。当 a 是整型变量时,被读入的数必须也是整型。当 a 是实型变量时,被读入的数可以是实型,也可以是整型。在读入时,数之前的空格与行分隔符被跳过。

从关于数的语法,我们知道,空格符不是组成数的符号。当它出现在数之后时,它标志数的结束,而它在数之前则被跳过。对于行的分隔符,也是这样。因而,当要输入若干整型或实型数时,可以用一个或几个空格符或另起一行把它们分隔开。

如 read(a, b); 当从键盘上敲入:

25 □□□□□□ 36 ↵; (□代表空格; ↵代表回车,表示本行输入完)就将数据 25、36 分别存入变量 a、b。这相当两个赋值语句:

```
a := 25; b := 36;
```

例如,设变量说明部分为



```
var
    A,C:integer;
    b1,b2,b3:real;
    ch1,ch2:char;
```

若要把值‘x’、‘y’、3、4.1、15、3、-2.5E5,按顺序赋值给变量 ch1、ch2、b1、b2、a、c、b3,则读语句可写成:

```
read(ch1,ch2);
read(b1,b2,a,c,b3);
```

从键盘输入:

```
xy      3      4.1      15      3      -2.5E5 ✓
```

也可分成两行:

```
xy      3      4.1      15 ✓
3      -2.5E5 ✓
```

语句执行后,就可完成上述赋值要求。

应该特别指出,这里‘x’之前不许有空格符,‘x’和‘y’之间也不允许有空格符。否则,若‘x’与‘y’之间有空格符,则 ch2 获得的值将是空格符,而不是‘y’。随后 b1 得到的头一个非空格字符将是‘y’,而不是数字 3,这样就全乱了。

2. readln 语句

```
readln(a1,a2,... an)
```

等价于

```
begin read(a1,a2,... an);readln end。
```

当程序执行到 readln 时就跳到下一行准备读下一行的数据,不管当前行还有没有数据没读完。以后再执行输入语句时,将从下一行的第一个数据开始读入。

例如,对上面的例子,从键盘输入

```
xy      3      4.1      15      5.6      ef
3      -2.5E5 ✓
```

为了使输入后得到与前面要求的一样的赋值,读语句可改写为

```
read(ch1,ch2);
readln(b1,b2,a);
read(c,b3);
```

当执行 read(ch1,ch2)、readln(b1,b2,a)时,从第一行数据 x、y、3、4.1 和 15 依次读入变量 ch1、ch2、b1、b2、a。第一行剩下的数据 5.6、e、f 被自动跳过。在执行 read(c,b3)时,从第二行的第一个数据 3 开始读入。

3. 标准过程 read(kbd,var)

上面介绍的读语句,在从键盘输入数据时,显示器回显这些输入数据。有的程序不希望显



示输入的数据, `read(kbd, var)` 可从键盘接受一个字符, 而不回显。该语句中 `kbd` 是又一标准文件标识符, 有它时, 输入数据就不回显。

综上所述, `read` 语句和 `readln` 语句基本功能相同:

- (1) 输入数据读入给程序中的变量;
- (2) 当一个数据行中的数据不足以供给读语句中的变量读数时, 则接着从下一个数据行读数。

`read` 语句和 `readln` 语句也有不同之处:

- (1) 一个 `read` 语句执行完后, 数据行中多余未读的数据可被下一个读语句继续读入; 而一个 `readln` 语句执行完后, 数据行中多余未读的数据不能被下一个语句读入, 由于该语句本身功能决定, 后续读语句将从下一个数据行读入。

- (2) `read` 语句必须有输入项, 而 `readln` 语句可以没有任何输入项。

8.7.2 写语句

`write` 过程和 `writeln` 过程是标准输出过程。在程序中调用这两个标准过程能完成将计算结果或其他数据传送到显示器(或打印机)的输出动作。

程序中调用的 `write` 过程和 `writeln` 过程, 通常称写语句。写语句有两种形式:

```
write(<项目名表>);
writeln(<项目名表>);
```

其中, 项目名表可以是算术表达式、布尔表达式或字符串表达式。输出时要先对表达式进行运算, 然后输出运算结果。显然, 如果表达式是数值常量, 则直接输出该常量; 如果是字符串常量, 即用单引号括起来的字符序列, 则输出字符串内容; 如果是变量, 则输出该变量存储的数据。

1. write 语句

写语句可写成

```
write(a1, a2, ... an)
```

它等价于

```
begin write(a1); write(a2); ...; write(an) end
```

执行 `write(a)` 语句时, 将项目 `a` 的数据内容输出到显示器上, 并不换行。再执行第二个写语句时, 数据仍在原输出行接着输出。写语句必须至少输出一项内容。例如:

```
n:=34; ch:='*'; m:=12;

write(n);
write(ch);
write(m);
```

执行上述写语句后, 在显示器上输出 `34 * 12`。

上面三个写语句可以写成一个语句:



```
write(n,ch,m);
```

其输出结果相同。

2. writeln 语句

写语句可写成

```
writeln(a1,a2,...,an)
```

等价于

```
begin write(a1);write(a2);...;write(an);writeln end
```

程序执行到不带项目表的写语句 `writeln` 时,不输出任何内容,只换行。接着执行第二个语句时,将从下行开始输出。

因此,执行 `writeln(a1,a2,...,an)` 语句时,按项目表一项一项输出。输出完最后一项后自动换行。

8.7.3 字符类型变量的输出

在介绍字符变量输出前,先简单介绍一下屏幕显示方式。通常将显示器屏幕上横向排列的字符序列称为行,纵向称为列。一般情况全屏幕可显示 25 行,80 列,即 25 行每行 80 个字符。每个显示位置可以用 (x,y) 坐标表示。左上角定为 $(1,1)$ 。

屏幕上光标(如在 IBM PC 微型计算机中)是指在显示屏幕某一位置上闪烁的一小横条,指示用户从这个位置输入数据或输出数据。

与屏幕显示有关的,有如下几个标准过程:

(1) clrscr 过程

`clrscr` 过程可以完成清屏功能。在程序调用它时,可以将整个屏幕变为背景色,完成清屏的作用,并将光标置于屏幕的左上角 $(1,1)$ 的位置。

(2) clreol 过程

`clreol` 过程能清除屏幕上光标所在行从光标位置起到该行结尾的所有字符,且光标仍在原来位置。

(3) gotoxy 过程

`gotoxy` 过程,可以完成光标定位功能。例如:

```
gotoxy(5,1);
```

将光标移到第 1 行,第 5 列的位置上。

注意:调用以上标准过程时,在程序首部的下一行必须调用 `uses` 子句列出 CRT 单元。

下面介绍字符类型变量的输出。PASCAL 语言字符类型变量有两种输出格式。

1. 标准输出格式

写语句形式为



```
write(ch);
```

字符类型值输出占显示宽度为 1 个字符宽(或称 1 列)。

执行这条语句有下面两个动作:

- (1) 字符变量 ch 的内容输出到显示器屏幕光标当前位置上;
- (2) 光标下移一个位置,等待输出下一个内容。

2. 域宽可控的输出格式

写语句形式为

```
write(ch;n);
```

其中, ch 为字符表达式; n 为整型表达式,表示 ch 所占据的输出域宽列数。输出按右对齐的方式,先印出 n-1 个空格,然后输出 ch 的字符值。

【例 8.1】 本例说明字符型变量的输入/输出,代码如下:

```
program EI(input,output);
var
    Ch1,Ch2:char;
begin
    writeln('please input Two Characters:');
    readln(Ch1,Ch2);
    write(Ch1,Ch2);
    writeln(Ch1;5);
    writeln(Ch2;4);
end.
```

程序运行果为

```
Please input Two Characters:
mn ↙ {下划线表示此字符由键盘敲入}
mn      m
      n
```

另外输出语句可以加上字符串常量一并输出。在上例中加一条语句:

```
writeln('First Character is',Ch1;3)
```

则输出为

```
First Character is m
```

在这里需要指出,PASCAL 语言的程序是按程序语句书写的次序,一条一条地执行的。从上面简单例子中,读者可以清楚地看到 PASCAL 语言顺序执行的特点。

8.7.4 整数类型变量的输出

155

PASCAL 语言对整型变量有两种输出格式。

1. 标准输出格式

写语句形式为



```
write(x)
```

整型值从光标当前位置开始输出。输出域宽和数值位数相同。

2. 域宽可控的输出格式

写语句形式为

```
write(x;n);
```

其中, x 是整型变量; n 是整型表达式, 表示 x 所占据的输出域宽列数。如果 x 项的数据太小, 则在它前面加空格; 如果 x 项数值位数大于 n , 系统自动取数值位数输出, 数值的正负号占一列位置。

例如, 有下面写语句:

```
x:=32;  
writeln('==',x,'==');  
writeln('==',x:6,'==');  
writeln('==x:1,==');
```

输出结果为

```
==32==  
== 32==  
==x:1==
```

【例 8.2】 下面代码是整型数据的输入/输出:

```
program Ax(input,output);  
var  
  A,B,C:integer;  
begin  
  writeln('Input Three Integer Nums:');  
  readln(A,B,C);  
  writeln(A,B:6,C:0);  
  end.
```

程序运行结果为

```
Input Three Integer Nums;  
22  15  66 ✓  
22      1566
```

8.7.5 实数类型变量的输出

实数类型变量的输出有以下几种格式。

1. 标准输出格式

写语句形式为

```
write(R)
```



在 17 个字符宽的输出域上输出实型变量 R 值,用十进制指数表示。

R $\geq$ 0.0 时格式为

□#.#####E*##

R $<$ 0.0 时格式为

-#.#####E*##

□表示空格, # 表示一位数字, * 表示加号或减号。例如:

```
a:=26.3;
```

```
write(a);
```

输出为

```
2.630000000E+01
```

2. 域宽可控的指数格式

写语句形式为

```
write(R;n)
```

R 为实型变量, n 为整型表达式。

输出格式为在 n 个字符的输出域上,按右对齐方法输出 R 值,用十进制指数表示。

当 R $\geq$ 0.0 时,格式为

blanks#.digitsE*##

当 R $<$ 0.0 时,格式为

blanks-#.digitsE*##

其中,blanks 表示零个或多个空格,digits 表示一到十个数字,# 表示一位数字。当域宽不够时,则自动按实际数值输出。例如:

```
a:=26.3;
```

```
write(a;10);
```

输出为

```
2.630E+01
```

而

```
write(a;1);
```

输出为

```
2.63E+01
```

3. 域宽可控的小数格式

写语句形式为

```
write(R;n;m)
```

其中,n、m 是整型表达式。

输出格式是在 n 个字符宽的输出域上,按右对齐方式输出 R 值,用十进制小数表示。其中小数点后有 m 位数字。数字之前加上适当个数的空格以隔成 n 个字符宽的输出域。当域宽 n 不够时,则自动按实际数值输出。例如:

```
b:=3.5;
```

```
writeln(b;10;2);
```

```
writeln(b;2;2);
```



□□□□□□ 3.50

3.50

【例 8.3】 下面代码是实型变量的输入/输出:

```
program Abc(input,output);
var
    R:real;
begin
    write('enter R=');
    readln(R);
    writeln(R);
    writeln(R:10);
    writeln(R:10:2);
    writeln(R:2:2);
    writeln(R:1);
    writeln(' * * * ',R,' * * * ');
end.
```

运算结果

```
enter R= 4.6 ✓
4.6000000000E+00      { 标准格式 }
4.600E+00             { 域宽控制为 10 }
    4.60              { 域宽为 10 列,小数部分占 2 列 }
4.60                  { 域宽不够,自动取位 }
4.6E+00               { 域宽不够,自动取位 }
* * * 4.6000000000E+00 * * * { 字符串常量与实数一起输出 }
```

8.7.6 布尔类型变量的输出

布尔类型变量的输出也有两种格式:一种是标准格式,每项布尔型数据从光标当前位置开始输出;另一种是域宽可控的输出格式。

写语句形式为

```
write(B;n)
```

其中,B 为布尔型值 false 和 true;n 是整型表达式,表示 B 所占据的域宽列数。

【例 8.4】 下面代码说明布尔数据的输出:

```
program Bool(input,output);
var
    B:boolean;
    A,C:integer;
begin
    write('input A,C:');
    readln(A,C);
    B:=A>C;
```



```
writeln(A:1,'>',C:1,'is',B);
writeln(A:1,'>',C:1,'is',B:8);
end.
```

运行结果：

```
input A,C:5 7
5>7 is FALSE
5>7 is |||| FALSE
```

布尔类型不能使用 read 语句进行输入。

DP **习题与思考题**

8.1 PASCAL 语言源程序由哪几部分组成？

8.2 指出下列用户自定义的标识符中哪些是非法的？为什么？

ab name 2xy begin AB-S a(3) a+b

8.3 下列数据中哪些是整数类型？哪些是实数类型？哪些是非法的？

5.6 44 1.2e+6 .88 6.6E+0.5 .5e3 0.06E-39 123000 21.0

8.4 下列哪些是合法的常量定义？

Const

```
Big=80;
Blank=' ';
Ch:'b'
-b=60;
m=20;
n=m+1;
t=true;
r=36.7;
```

8.5 指出下列变量说明中不正确的,并分析错误原因。

(1) var

```
m,n:integer;
a,b,c:real;
m:char;
```

(2) var

```
i,j,k:integer;
b:character;
```

(3) var

```
a1,a2,a3:integer;ch1:char;
```

8.6 求表达式的值。

(1) $8 > 5$; (2) $'A' < 'B'$; (3) $\text{succ}('M') = 'L'$; (4) $\text{chr}(66)$;



(5) $\text{ord}('B')$; (6) $\text{odd}(3)$; (7) $5 * (6 \bmod 4) - 3$; (8) $\text{sqr}(3) - 4 \text{ div } 5$;

(9) $\text{pred}(\text{true})$; (10) $(3 < > 4) \text{ and } (4 < 3) \text{ or not } (2 > 1)$ 。

8.7 将下列代数式用 PASCAL 表达式表示。

(1) $\frac{a+b+c}{2}$; (2) $\sqrt{s(s-a)(s-b)(s-c)}$;

(3) xy^3 (4) $\sin^2 x + \cos^2 x$ 。

8.8 若有如下定义:

const

a='*';

b=3;

var

x,y:integer;

m,n:char;

a1,a2:real;

b1,b2:boolean;

指出下列赋值语句中哪些是错误的?为什么?

(1) $y = a1 + 3 * b$;

(2) $a1 := x + y$;

(3) $2a2 := 3 + b$;

(4) $b1 := (3 > 1) \text{ and } (b < 1)$

(5) $m := a$;

(6) $a2 := a$;

(7) $b2 := a > \text{pred}(n)$

(8) $y = 6/3$;

8.9 编写计算三角形面积的程序 $\left[s = \frac{a+b+c}{2}, \text{面积} = \sqrt{s(s-a)(s-b)(s-c)} \right]$ 。

8.10 编写程序输出如下图形。

```

      *   *   *   *   *   *
    *
  *
*
*
      *   *   *   *   *   *

```



第 9 章

控制语句

在 PASCAL 语言中,按其程序结构可分为三种,即顺序结构、分支结构和循环结构。由于三种结构间存在着差异,构成三种结构的语句控制方式是不同的。如:在第 1 章中,由赋值语句、读过程语句和写过程语句所构成的程序,只能按其书写顺序从上到下,自左至右依次执行。故称此类程序结构为顺序结构。而本章中将引出的条件语句、重复语句所构成的程序结构则分别称之为分支与循环结构。前三节介绍复合语句与分支程序的概念,后五节介绍循环程序等有关概念。

DP 9.1 复合语句

复合语句一般包含一系列成分语句,每个语句可以是简单语句,如赋值语句、过程语句或结构型语句(分支结构或循环结构)。它按照书写顺序执行其成分语句。

语句格式:

```
begin
    <语句 1>;
    :
    <语句 2>;
    <语句 n>
end
```

begin 和 end 是 PASCAL 保留字,相当于一个语句括号把若干语句括起来作为一个语句处理。括号内的每个语句之间用分号隔开。

复合语句写成

```
begin
end;
```

形式也是合法的。

当复合语句写成如下形式:

```
begin
    R:=3;
    L:=2*3.14*R;
    write('L=',L);
end
```

逻辑上在 write('L=',L) 与 end 间存在着一空语句。采用这种形式调试时也带来了方便之



处,因为修改程序时若在最后一语句与 end 之间加一空语句,可避免忘记加分号的错误。复合语句在后续的控制语句中,当需要多个语句作为一个语句出现时经常被使用。

DP 9.2 条件语句

到现在为止,读者已经能够利用顺序结构进行一些简单的程序设计。但是随着讨论问题的深入,这种结构是远远满足不了用户程序设计要求的。在程序设计中经常遇到要求计算机进行逻辑判断,并根据判断结果进行不同处理的情况。

针对上述问题的提出,并根据条件决定某语句是否执行或决定两个语句中执行哪一个,这种做出抉择的语句称为条件(if)语句。本节将对两种条件语句分别进行介绍。

9.2.1 单分支条件语句

单分支条件语句的一般格式为

```
if〈布尔表达式〉then  
    〈语句〉
```

if, then 为 PASCAL 保留字。布尔表达式称为“条件”判断。当布尔表达式的值为 true 时,则执行 then 后面的语句;若布尔表达式的值为 false 时,程序将跳过 then 后的语句部分去执行下条语句。语句部分可以是一复合语句或空语句。

【例 9.1】 乘坐齐齐哈尔至北京的民航班机,每张客票当携带行李小于等于 20 kg 时,免费托运。大于 20 kg 时超过部分按 0.15 元/kg 收费。试编制计算收费的程序。

为给初学者打下一个良好的程序设计风格,程序设计一般按如下步骤进行。

(1) 数据说明

X: real——质量(单位:kg);

Y: real——收费(单位:元)。

(2) 算法设计

$$Y = \begin{cases} 0 & X \leq 20 \\ (X - 20) * 0.15 & X > 20 \end{cases}$$

(3) 由 write 语句提示用户录入数据,用 readln 语句接收用户输入的行李质量数据。

(4) 由单分支结构,构成程序段即对用户键入数据进行判断,并按计算标准核算收费额。程序如下:

```
program Example1(input,output);  
var  
    X,Y: real;  
begin  
    write('input the weight of your laggage please:');  
    readln(X);  
    Y:=0;  
    if X>20 then
```



```

Y:=(X-20)*0.15;
write('The weight of your laggage X=',X:8:3);
writeln('The payment is Y=',Y:8:3);
end.

```

运行结果:

```

input the weight of your laggage please: 12 ✓
The weight of your laggage X=12.00
The payment is Y=0.00

```

再次运行结果:

```

input the weight of your laggage please: 25 ✓
The weight of your leggage X=25.00
The payment is Y=0.750

```

9.2.2 双分支条件语句

PASCAL 语言中提供了另一种条件语句——双分支条件(if)语句。

语句格式:

```

if<布尔表达式> then
  <语句 1>
else
  <语句 2>

```

双分支语句是在原单分支语句的基础上,增加了保留字 else 和语句 2。其中 else 前不能加分号“;”,如加分号则将 if 与 else 割开来,认为是两个语句。双分支语句功能为首先计算布尔表达式的值,当布尔表达式的值为 true 时执行语句 1,为 false 时执行语句 2。两语句必执行其一,语句 1、2 可是单一语句,也可是一复合语句或是一空语句。

【例 9.2】某工厂为每个职工调整工资原则如下:(1)工龄在 10 年以下者,每人增加原工资的 10%;(2)工龄在 10 年以上者增加原工资的 15%。试编制计算调整工资额的程序。

(1) 数据说明

Age——工龄(以一整年为准,设其为整型);

Last——原工资额(有小数部分,设为实型);

New——调整后的工资额(实型)。

(2) 算法设计

$$\text{New} = \begin{cases} \text{Last} + \text{Last} * 0.15 & \text{Age} \geq 10 \\ \text{Last} + \text{Last} * 0.1 & \text{Age} < 10 \end{cases}$$

程序如下:

```

program Example4(input,output);
var
  Last,New:real;

```



```
Age:integer;
begin
  write('input,the salary please:');readln(last);
  write('input,year of work please:');readln(Age);
  if Age>=10 then
    New:=Last+Last * 0.15
  else
    New:=Last+Last * 0.1;
  write('The raised salary is:',New;6:3);
end.
```

运行结果：

```
input the salary please: 123 ✓
input years of work please: 9 ✓
The raised salary is :135.30
input the salary please: 123 ✓ {第二次运行}
input years of work please: 12 ✓
The raised salary is :141.45
```

9.2.3 条件语句嵌套

前面讲过的条件语句的语句部分,原则上可以是任何语句,当然也可以是条件语句。这种条件语句的语句部分仍是一个条件语句的情况称之为条件语句嵌套。形成嵌套的重要目的是为解决较为复杂、多种条件出现的情况下的一些问题。例如:

```
if <布尔表达式 1> then
  if <布尔表达式 2> then
    语句
```

其中,if<布尔表达式 2>then 语句本身是一个条件语句。它又是 if<布尔表达式 1>then 条件语句的语句部分,因而这是一个嵌套的条件语句。

提倡的嵌套的一般形式为

```
if <条件 1>then
  <语句 1>
else if<条件 2> then
  <语句 2>
  :
else if <条件 n> then
  <语句 n>
else
  <语句 n+1>
```

【例 9.3】 按税务部门核收个人所得税规定,假设当综合收入(工薪、承包、劳务、租赁四



项和)小于 400 元时,国家免收税款。若在 400 元至 600 元之间时,其超过 400 元部分税率为 30%,600 元以上的,除收 30%外,其超过 600 元部分再核收 20%税,外加手续费 80 元。试编制计算税收的程序。

(1) 数据说明

X——个人收入总额,单位:元;

Y——税收额,单位:元。

(2) 算法设计

$$Y = \begin{cases} 0 & X \leq 400 \\ (x-400) * 0.3 & 400 < x \leq 600 \\ (x-400) * 0.3 + (x-600) * 0.2 + 80 & x > 600 \end{cases}$$

程序如下:

```
program Example6(input,output);
var
    X,Y:real;
Begin
    Y:=0;
    write('input your income:');
    readln(X);
    if (X>400)and(X<=600) then
        Y:=(X-400) * 0.3
    else if X>600 then
        Y:=(X-400) * 0.3+(X-600) * 0.2+80
    else
        writeln('exempt from taxation');
        writeln('The amaunt of tax Y=',Y:8:3);
    end.
```

运行结果:

```
input your income: 400 ✓
The amount of tax Y=0.000
input your income: 623 ✓ {第二次运行}
The amount of tax Y=87.840
```

这里只是一个假设的例子,通过例子说明其算法与编程,不反映真实的所得税征收情况。



上一节介绍的具有多分支的选择结构是用嵌套的 if 语句实现的。当多分支选择的各个条件由同一个表达式的不同结果值决定时,还可以用 case 语句实现。它的选择过程,很像一



个多路开关,即由语句选择表达式的值,决定切换至哪一路去工作。因此,在程序设计中,它是一种强有力的手段。在实现多路分支控制时,有时用 `case` 对某些问题的处理和设计,比用 `if` 语句写程序具有更简洁、清晰之感。

语句格式:

```
case <选择表达式> of
    情况常量 1; 语句 1;
    情况常量 2; 语句 2;
    :
    情况常量 n; 语句 n
end
```

`case`、`of`、`else`、`end` 为保留字。其表达式部分,可以是整型、字符型、布尔型,但不能是实型。稍后,我们将知道情况表达式只能是顺序类型; `of` 后的常量称为情况常量,是根据将要选择的情况设置的常量。在程序设计中,语句前面的情况常量部分可由具体问题决定由一个或多个常量构成。当出现多个情况常量时,它们之间由逗号隔开。情况常量应与表达式具有相同的类型。

该语句的执行过程为:首先计算表达式的值,若其值与 `of` 后面的情况常量之一相等,则执行该情况常量后的语句部分;然后将其转到 `case` 语句结尾。如果选择表达式不与任何情况常量相等,则判断 `else` 是否存在。若存在则执行 `else` 后的语句部分,执行后再转到 `case` 语句结尾;否则,直接转到语句结尾。其中语句部分,可以是复合语句或空语句。这里 `case` 语句与上一节中引出的 `if` 语句不同,前者为二选一;而后者则为多选一,是通过表达式的切换来达到多路径的控制功能的。`case` 语句对多分支结构问题描述十分方便、清晰。

【例 9.4】 试编写根据用户键入的两个操作数和一个运算符,由计算机输出运算结果的程序。

程序如下:

```
program Example7(input,output);
var
    Result,X,Y:real;
    ch:char;
begin
    write('input X,Y:');readln(X,Y);
    write('input opertor(+、-、*、/):');readln(ch);
    Result:=0;
    case c of
        '+' : Result:=X+Y;
        '-' : Result:=X-Y;
        '*' : Result:=X * Y;
        '/' : if Y<>0.0 then
                Result:=X/Y
            else
    end
```



```

        write('divide overflow!');
    end;
write(X:6:2, ch:2, Y:6:2, '=');
writeln(Result:6:2);
end.

```

运行结果：

```

input X,Y: 12 3 ✓
input operator(+、-、*、/): ± ✓
12.00+3.00=15.00
input X,Y: 12 3 ✓
input operator(+、-、*、/): = ✓
12.00-3.00=9.00.
{第三、四次运行略}

```

从以上用 case 语句编写的程序可以看出,当多分支的分支条件可用一个表达式表示,且该表达式的值为顺序类型数据时,用 case 语句描述是方便和清晰的。但有一个问题请读者注意,此程序运行中输入的 ch 值一定要在 +、-、*、/ 4 个符号内,否则程序将无结果输出与误操作提示。为解决此程序存在的问题,即输入值超出常量范围时提示用户,TURBO PASCAL 在 case 语句中引用了 else 语句,可使此问题得以解决。因此上述程序段部分可写成如下形式:

```

        :
case ch of
    '+' : Result:=X+Y;
    '-' : Result:=X-Y;
    '*' : Result:=X*Y;
    '/' : if Y<>0.0 then Result:=X/Y
          else writeln('divide overflow');
else
    write('sorry you have made mistakes please do it again');
end;
        :

```

case 语句中的 else 部分是对标准 PASCAL 的扩充,按标准 PASCAL 情况表达式的值必须与情况常量之一相等,否则认为出错。有了 else 后,可解决当无表达式值与情况常量相等时的一些问题。但在使用时,仅能出现在 case 最后,不可插在中间。另外在 case 语句中,情况常量必须是相互间独立存在的,即相互间不等。



问题。例如求：

$$\sum_{n=1}^{100} n$$

即求 $1+2+3+4+\cdots+100$ 。

在 PASCAL 中提供了三种具有重复功能的语句,可较好地解决上述问题。本节引出第一种: while 语句。

语句格式:

```
while <布尔表达式> do  
    <语句>
```

while、do 为保留字。首先计算布尔表达式的值,当布尔表达式取值为 false 时,则 while 语句将跳过语句部分而结束,转向下一语句;反之,若布尔表达式取值为 true 时,则执行 do 后面的语句。执行后再重复上述过程,直到当布尔表达式的值为 false 时停止,转去执行下一条语句,故又称此语句为“当”语句。其中 do 后面的语句可以是一复合语句或是一空语句。

【例 9.5】 求累加和(即 $1+2+3+\cdots+100$)的问题,可按如下步骤设计。采用 while 语句的程序设计如下:

```
program Example8(output);  
var  
    S,I:integer;  
begin  
    S:=0;I:=1;  
    while I<=100 do  
        begin  
            S:=S+I;  
            I:=I+1;  
        end;  
    write('sigma S is:',S);  
end.
```

运行结果:

```
sigma S is:5050
```

程序中 $I=1$ 为循环变量赋初值部分,即以此值为循环的起点; $I\leq 100$ 是循环终止条件,此程序终值为 100。若小于循环终值,则重复上述过程,直到 $I>100$ 才终止循环过程。 $S:=S+I$ 是程序循环的累加器。 $I:=I+1$ 是修正循环条件部分。 S 与 I 两赋值语句,将在循环语句控制下被重复执行。这里称被执行的语句为循环体。

DP 9.5 直到语句

repeat 语句为重复语句的第二种形式,其语句格式为

```
repeat
```



<语句>

until<布尔表达式>

repeat、until 为保留字。计算机首先执行夹在 repeat 与 until 之间的语句部分,然后对布尔表达式进行计算。当布尔表达式取值为 true 时,结束 repeat 语句,即直到布尔表达式取值为 true 时终止循环,故称此重复语句为“直到”语句。反之,如布尔表达式取值为 false,则重复 repeat 与 until 之间的语句序列,即循环体。

由于 repeat 本身结构所决定,当循环体为两个以上语句构成时也不必采用复合语句。因 repeat 与 until 已相当于一个语句括号。由此可将【例 9.5】中求 $1+2+3+\cdots+100$ 的算式用 repeat 和 until 写成如下形式:

```
program Example8(output);
var
    S,I:integer;
begin
    S:=0; I:=1;
    repeat
        S:=S+I;
        I:=I+1;
    until I>100;
    write('sigma S is',S);
end.
```

DP 9.6 循环语句

重复语句的第三种形式为 for 语句,它的格式有两种:

for <控制变量>:=<初值表达式> to <终值表达式> do<语句>

for <控制变量>:=<初值表达式> downto <终值表达式> do<语句>

for、to、downto、do 为保留字,do 后的语句称为循环体。其中控制变量可以是整型、字符型或布尔型,但不能为实型。一般的,可以是顺序类型。初值和终值表达式的类型通常应与控制变量的类型相同。

for 语句的执行过程如下,对于

```
for V:=e1 to e2 do body
```

等价于执行下面的复合语句:

```
begin
    temp1:=e1;
    temp2:=e2;
    V:=temp1;
    while V<=temp2 do
```



```
begin
  body;
  V:=succ(V);
end;
end;
```

对于 downto 形式的 for 语句:

```
for V:=e1 downto e2 do body
```

等价于执行上面类似的复合语句,只不过把其中的

```
V(temp1)<=temp2 和 V:=succ(V)
```

分别修改为

```
V(temp1)>=temp2 和 V:=pred(V)
```

(1) 对于 to (或 downto) 形式的 for 语句,若初值大(或小)于终值,则循环体一次也不执行。

(2) 在进入 for 语句时,计算初值表达式 e1 和终值表达式 e2 的值,且以后就使用这两个值。因此当执行循环体时,即使改变了 e1 和 e2 中变量的值,初值与终值也不会变。

(3) 当 for 语句执行时,控制变量逐次取其后继(对 to 型)或前趋(对 downto 型)。因此,在执行过程中,改变不了步长。

(4) 函数 succ 和 pred 对于顺序类型(整型、字符与布尔类型是顺序类型)有意义,而对实数类型无意义,因此控制变量的类型不能为实型。

下面采用 for (1) 语句将【例 9.5】设计成如下程序:

```
program E5 (output);
var
  I,S: integer;
begin
  S:=0;
  for I:=1 to 100 do
    S:=S+I;
    write('segmas s:s:',s);
  end.
```

程序设计中有三点提请读者注意:

(1) for 语句中控制变量与其他变量一样,必须在说明部分进行定义。

(2) 当执行 for 循环完毕后,控制变量已无定义,不可再对其进行引用。

(3) 在对相应的控制变量定义并代入 for 语句结构内以后,不得在循环体内有给控制变量重新赋值的语句出现;但可引用控制变量参与相应的表达式运算。

综上所述,由于 for 语句本身步长为一定值与循环次数为已知等条件所限,类似下列问题 for 语句均无法解决。例如,对从键盘输入的任意数进行筛选直到键入值 $\leq$ 某值,显示满足条件值并终止循环的执行;再如,利用循环控制变量来计算 1、3、5、7、9...的平方根值等。



DP 9.7 循环嵌套

上几节讨论了三种重复语句的使用及相互间的区别,并通过一定的实例对此进行了说明。但以上介绍的循环结构都是单一的重复语句构成,即循环体部分不再是循环语句。我们称之为单重循环。当重复语句的语句部分本身又是一个循环语句时,我们称发生了循环的嵌套。对应的循环称二重循环;若再有一次嵌套称三重循环;等等。

【例 9.6】 试编写能够打印出如下所示图形的程序。

```

      *
    * * *
  * * * * *
* * * * * * *
* * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *

```

程序如下:

```

program ExampleH(output);
var
  I,J,K:integer;
begin
  for I:=1 to 8 do
    begin
      for J:=1 to 8-I do
        write(' ');
      for K:=1 to 2 * I-1 do
        write('* ');
      writeln;
    end;
  end.

```

此程序的设计是由二个并列环,外加一个嵌套循环构成的,即两个并列环嵌在最外环内。各环作用如下:

- (1) 外层环控制打印的行数,此三角形共 8 行,故外环的设置为 for I:=1 to 8 do。
- (2) 嵌套在外环内的第一个并列环为空格,输出环根据图形变化控制输出不同数量的空格,如第一行需打印 7 个空格,方能打印图形中第一个星号。故第一次进入循环时 for J:=1 to 8-I,此时 I=1,此环重复 7 次输出 7 个空格。往后依次类推,请读者自行分析。
- (3) 外环内第二个并列环为控制三角形中每行中星号的个数,如首次进入此环时 for k:=1 to 2 * I-1,此时 I=1,故此环仅执行一次,刚好达到控制目的。



DP 9.8 goto 语句

现代程序设计思想主张尽量不用或少用 goto 语句,甚至国外有些版本的 PASCAL 语言不讨论 goto 语句。因此本节仅简要的介绍一下 goto 语句的使用方法和它给程序控制带来的不良影响。在 TURBO PASCAL 程序中,任何语句前都可以加上一个标号,使得 goto 语句可以直接转向它。一个语句标号可以由一个标号名跟上一个冒号(:)构成。例如:

```
      :  
1: A:=A+5;  
      goto 4;  
2: A:=A * 5;  
      goto 4;  
      :  
4: write('A=A',A);  
      :
```

在 PASCAL 中,任何一个标号在其可以被使用之前,必须预先在标号说明部分加以说明。标号说明部分以保留字 label 开始,然后是一系列的语句标识符,各标号间用逗号“,”分开,最后以分号结束,其一般形式为

label<标号 1>,<标号 2>,...,<标号 n>;

例如,下列标号说明部分说明语句标号 10、999、error、quit。

label 10,999,error,quit;

这里要说明的是,尽管在标准 PASCAL 中规定标号必须是无符号数字,并且最多不超过四位,即 1~9999 之间。但在 TURBO PASCAL 中为实现较好可读性,规定不但允许数字做标号,而且还允许用标识符如 error、quit 作语句标号。在对语句标号加以说明之后,就可以在程序中使用标号,进而使得可以用 goto 语句根据程序处理的需要,转向语句标号标出的语句去执行。goto 语句可以用在程序任何需要转移的地方。它转向的去处由 goto 语句中标号规定。goto 语句的一般形式为

goto <语句标号>

它的作用是把程序的下一步执行,转向由保留字 goto 后面给出的语句标号所标识出的语句处。一个良好的程序结构应在程序的每部分里只有一个入口和一个出口。这样才便于验证程序的正确性,有效地避免逻辑上的错误。而 goto 语句由于本身的工作特点所决定,它可使程序出现多个入口和多个出口,使程序结构变坏。这里我们指出滥用 goto 语句的危害,是提醒读者应有限制地、慎重地使用它。使用 goto 语句应注意如下规则:

- (1) 标号在使用之前,必须预先加说明;
- (2) 不准用 goto 语句从构造语句(复合语句、条件语句、重复语句)外转入构造语句内。



DP 习题与思考题

9.1 编程判断用户输入值 x , 当 x 大于零时求 x 的平方根, 否则不做什么操作。

9.2 从键盘输入两个数, 求这两个数之差的绝对值。

9.3 编写程序计算下列分段函数:

$$y = \begin{cases} 0.7x + 0.53 & x > 2.1 \\ 0.5x + 0.95 & x \leq 2.1 \end{cases}$$

9.4 从键盘输入 20 个实数, 分别求出所有正数的平均值与所有负数的平均值。

9.5 某学院截止到 2000 年在校生为 4000 人, 计划年增长率为 5%, 试编程计算 2001 年到 2010 年每年的在校生人数。

9.6 编写程序输出如下图形:

```

* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *

```



第 10 章

过程与函数

过程与函数是 PASCAL 语言中的重要概念之一,也是实现结构化程序设计的主要手段之一。前几章中所接触的标准过程和标准函数,是由 PASCAL 语言系统预说明的,只要遵循一定的调用规则使用就可以了;除此之外,在 PASCAL 中还允许程序员根据需要自己说明一些其他过程与函数。

在一个程序中,相同的程序段可能多次出现,为了避免相同程序段的重复书写,减少程序的录入量及录入错误,在程序设计中,通常将这些重复出现的程序段抽出来,单独书写成为子程序,以供随时调用。这些子程序一般是以过程或函数的形式在 PASCAL 中呈现的。

DP 10.1 过 程

PASCAL 允许用户根据需要自己说明一些其他过程。这些过程,可以带有自己的参数,称为带参过程;也可以没有参数,称为无参过程。让我们先从最简单的过程——无参过程入手。

10.1.1 无参过程

无参过程的一般形式为

```
procedure <标识符>;  
    <过程体>;
```

其中,保留字 `procedure` 表示这是一个过程说明, `procedure` 后面跟的标识符称为过程标识符,是用户给过程起的名字。过程体就是一个程序段。一般而言,它可以有标号说明部分、常量定义部分、类型定义部分、变量说明部分、过程或函数说明部分和语句部分。

过程说明与下节将要介绍的函数说明均写在过程说明与函数说明部分中,过程说明与函数说明部分或为空,或为若干个过程说明与函数说明,它们的每一个都要跟一个分号。

其中过程说明又可分为无参过程说明与带参过程说明。先通过一个简单的例子来看一下无参过程的说明和引用是怎样实现的。

【例 10.1】 编写一加法程序,为了让输出格式更加清晰,将加法所得结果的上下各画一条直线(这里可以用连续输出许多个“—”号来实现)。假设直线由 10 个符号组成,其输出格式如下:



```
-----  
sum=66  
-----
```

用过程来解决这一问题,就会得到很好的效果,其程序如下:

```
program exmple(input,output);  
uses CRT;  
var  
n1,n2,t:real;  
procedure DrawLine;  
const  
L=10;  
Var  
i:integer;  
begin  
for i:=1 to L do  
write(' ');  
writeln;  
end;  
begin  
read(n1,n2);  
t:=n1+n2;  
gotoxy(10,10);  
DrawLine;  
gotoxy(10,11);  
writeln('sum=',t:8:2);  
gotoxy(10,12);  
DrawLine;  
end.
```

从【例 10.1】中可以看到,由于过程的引入,使程序结构更加简捷、清晰。DrawLine 过程的功能是画出一条直线。

可以看出,虽然在主程序前面过程说明中包含了若干语句序列,但是,在程序开始运行时,这些语句并不执行;而是在主程序执行到 DrawLine 过程调用语句时,相应过程中的那部分语句才被“激活”,得以一一执行。

主程序中,程序体的标识符 DrawLine 被看成是一条语句,称为过程调用语句,或简称为过程语句。对于无参过程,过程语句就是执行一遍相应过程说明的程序体,即过程调用。过程说明用来规定怎样实现某项任务的具体作业,而过程调用则规定了何时发挥该过程的作用。

10.1.2 带参过程

在过程的使用中,人们往往希望一个过程(子程序)可对多组数据进行处理。这样只要每次调用过程时,赋以一组具体参数,就会得到对应这组参数的运算或操作的结果,使过程更具



灵活性、通用性。

带参过程说明的一般形式为

```
procedure<标识符>(<形式参数表>);  
<过程体>;
```

与无参过程相比,带参过程的说明,在过程标识符后面多了一个形式参数表。

形式参数表由若干个形式参数段组成,其一般形式为

形式参数段;形式参数段;...;形式参数段

其中,形式参数段是用来对形式参数进行说明;形式参数又分为数值参数和变量参数两种。

数值参数说明的一般形式为

<标识符>,<标识符>,...,<标识符>:<类型标识符>

变量参数说明一般形式为

var<标识符>,<标识符>,...,<标识符>:<类型标识符>

在 PASCAL 语言中,不仅整数类型、实数类型、字符类型和布尔类型的变量可作为过程参数,而且数组类型、文件类型、集合类型、记录类型等构造类型的变量和指针类型的变量也都可作为参数进行传递。

带参过程的调用与无参过程的调用不同,它除要写出被调用过程的标识符外,还要有传递参数的实参表。带参过程调用语句的一般形式:

<过程名>(<实在参数表>);

任何过程的形式参数的个数必须与实在参数个数相等。实在参数与形式参数的对应关系是按它们出现在过程语句的实在参数表中的先后次序与出现在过程形式参数表中的先后次序一一对应的,类型相同,其数据类型应符合如下要求:

(1) 当形式参数是值参数时,其实在参数可以是一个表达式(变量是表达式的特例)。通常表达式的类型应与对应形参的类型相同;但也有例外,如当形参的类型是实数时,实参表达式也可以是整型的。一般而言,实参表达式的类型与对应形参的类型应是赋值相容的。

(2) 当形参是变量参数时,则实参只能是变量访问,不能是常量或一般表达式,实参的类型与对应形式变量参数应该是相同类型。

对于形参与实参的结合,还有下列限制。

若与变参对应的实参是下标变量,则其下标表达式的值在结合前要先计算出来;之后,与变量参数结合的总是下标为该值的下标变量。

在实际应用中,采用过程的概念,的确为程序设计提供了很大的方便,增加了程序功能的灵活性,减少了程序行数。如【例 10.1】中的画框过程,由于无参,只能画一种线条的边框;如果将此过程改写为带参过程,就可按用户要求,画出不同线条的边框。其实现代码如下:

```
program exmple(input,output);  
procedure drawline(L:integer;Ch:char);
```



```

var
    i: integer;
begin
    for i:=1 to L do
        write(Ch);
    writeln;
end;
begin
    write('1— ');
    drawline(10,'#');
    writeln;
    write('2— ');
    drawline(20,'*');
end.

```

运行输出结果形式如下：

```

1— #####
2— *****

```

10.1.3 值参数和变量参数

带参过程中的参数,从其对数据传递的特点上可分为两种:一种为数值参数,另一种为变量参数。这两种参数的特点,可从下例程序的执行结果来观察、分析。

【例 10.2】 设有一个二元函数, $Z=10X+Y^2$,我们的目的是设计能够求出不同 X 、 Y 下的 Z 值的程序,其中, X 、 Y 的值由键盘输入。用带参过程实现的代码如下:

```

program Main(input,output);
var
    A,B,C,D,E,F: real;
procedure Calculate(X,Y: real; var Z: real);
begin
    X:=X*10; Y:=sqr(Y);
    writeln('Z=',Z:7:2); Z:=X+Y;
end {of Calculate};
begin
    C:=0; F:=0;
    write('Please input A,B:');
    readln(A,B); Calculare(A,B,C);
    writeln('A=',A:7:2, 'B=',B:7:2, 'C=',C:7:2);
    write('Please input D,E:');
    readln(D,E); Calculare(D,E,F);
    writeln('D=',D:7:2, 'E=',E:7:2, 'F=',F:7:2);
end (of Main).

```

运行结果为



```
Please input A,B: 3    6 ✓  
Z=  0.00  
A=  3.00  B=  6.00  C=  66.00  
Please input D,E: 2.4  3 ✓  
Z=  0.00  
D=  2.40  E=  3.00  F=  33.00
```

在过程 `calculate` 的形参表中的参数 `X`、`Y` 和 `Z` 代表着两种不同形式的参数。其中,不带保留字 `var` 的 `X`、`Y`; `real` 是形式值参数说明, `X`、`Y` 称数值参数。带保留字 `var` 为前缀的 `var Z`; `real` 是形参表中的变量参数说明, `Z` 称变量参数。

主程序 `Main` 的前两条语句分别给变量 `C`、`F` 赋初值为零,紧跟着的写语句提示用户输入变量 `A`、`B` 的值;然后由 `readln(A,B)` 读入了两个实型数,随着执行对 `Calculate` 过程的调用。在程序 `Main` 中通过过程语句 `Calculate(A,B,C)` 与 `Calculate(D,E,F)`,两次调用了过程 `Calculate`,这里的两组实型参数 `A`、`B`、`C` 和 `D`、`E`、`F` 与形式参数 `X`、`Y`、`Z` 之间应该存在个数、位置和类型的一一对应关系。第一次调用 `Calculate` 过程时,分别将实参 `A`、`B`、`C` 放在对应形参 `X`、`Y`、`Z` 的位置上;后面的写语句是将经过过程加工后的 `A`、`B`、`C` 的值输出至屏幕。

可以看到,尽管在过程 `Calculate` 中有 `X:=X*10`、`Y:=SQR(Y)` 的语句执行, `A`、`B` 的值并没有改变;而 `C` 被赋以新值。这说明 `X`、`Y` 这两个形式值参数,只具有将实参的值带入过程体内的作用。无论 `X`、`Y` 在过程中参与何种运算,它们并不改变与之相配实参的值;而只在本过程内有意义。这正是值参数的重要特性。

对于形式变量参数 `Z` 从过程体中的写语句 `writeln('Z=',Z:7:2)` 执行的结果看, `C` 的值也由 `Z` 带入到过程体中,屏幕输出 `Z=0`;当过程 `Calculate` 执行后, `C=66` 的值是由形式变量参数 `Z` 传递回来的。这说明变量参数具有将实参的值“传入”过程和将过程内变量参数的值“传出”至实参的作用。

另外,变量参数不能作为过程体内 `for` 循环的控制变量,因在循环结束后循环控制变量已无意义,而变量参数对程序的运行应是有意义的变量。

以上了解了值参数与变量参数的特点及其作用。

由此可见,对于形式值参与形式变参在使用时应注意的要点是:要求把值的改变传出过程的参数必须用变量参数,而希望在过程运行时其值不受影响的实参只能用值参数与之对应。



10.2 函 数

过去接触到的 `sin(X)`、`ord(Ch)` 等,就是函数过程,只不过是由系统预先说明的标准函数过程;在应用中,用户可以像前述自定义过程那样自己说明一些其他函数。函数说明的结构与过程说明的结构类似,一般格式为

```
function<函数名>(<形式参数表>):<类型>;{函数首部}  
    <函数体>
```



其中,形式参数段与过程说明中的完全相同。

函数说明分为函数头部和函数体两部分。函数头部中的 function 是 PASCAL 语言的保留字。函数名应符合标识符的定义规则。形式参数表的作用是说明函数的自变量,即给出每个参数的标识符和类型。标识符和类型之间用冒号分隔,同一类型的不同参数之间用逗号分隔,不同类型的参数说明之间用分号分隔。形式参数表后面给出了函数结果的类型。函数结果的类型能用简单类型标识符或指针类型标识符表示,因为这两种类型才有单个的值。函数头部最后以分号结束。

这里的函数体包括说明部分和语句部分。说明部分对仅在函数中使用的量加以说明,可以出现常量定义、类型定义和变量说明,也可以再出现函数说明。也就是说,函数说明允许嵌套定义。函数体以分号结束。在函数体的语句部分中,必须有一个赋值语句,它将所求得的函数值赋给函数名。

【例 10.3】 求三角形面积的程序段写成函数说明。其代码如下:

```
function ts(x,y,z:real):real;      {函数首部}
var   s: real;
begin
    s:=(x+y+z)/2;
    ts:=sqrt(s*(s-x)*(s-y)*(s-z));
end;
```

其中,ts 是函数名,是实型量;x、y、z 分别表示三角形的三条边长,也都是实型;s 的作用只局限于函数体内。ts:=sqrt(s*(s-x)*(s-y)*(s-z))是对函数名赋值的语句。函数说明与过程说明的结构基本相同,主要有如下区别:

- (1) 函数说明的标志是 function,而过程说明的标志是 procedure。
- (2) 过程说明允许没有形式参数表。但一般情况下,函数说明应有自己的形参,这些形参相应于函数的自变量。
- (3) 在函数说明的首部必须指出函数的数据类型。
- (4) 在函数体的语句部分中,必须有对应函数标识符赋值的语句,并且这些语句在函数被引用时至少要有一句被执行,最后赋给函数标识符的值就是函数值。

在主程序中,通过函数调用语句来引用某个函数。函数调用的一般格式为

〈函数名〉(实在参数表)

实在参数表中的参数之间用逗号分隔。它们和函数说明中的形式参数按序一一对应、个数相等及类型赋值相容。

函数调用的结果返回的是一个值,因此函数名不能同过程名一样作为一个语句写在程序中;而只能像标准函数那样出现在表达式中。程序执行函数调用时,先将实在参数与形参一一对应结合;之后转去执行函数中的语句;然后从函数体返回调用点,继续执行调用程序。由此可见,当有了函数说明之后,自定义函数的调用与标准函数的调用是完全相同的。

【例 10.4】 编制程序,计算下式:

$$Y = X^2 + SH(X)$$



SH(X)是双曲正弦函数,在 PASCAL 的标准函数中没有该函数,必须由用户自己说明。

由数学知识:

$$\text{SH}(X) = (E^X - E^{-X})/2$$

这里只有两个变量 X、Y,都取实型,其程序如下:

```
program Exoffuct(input,output);
var
    X,Y:real;
function SH(X1:real):real;
begin
    Sh:=(exp(X1)-exp(-X1))/2;
end(of SH(x));
begin
    write('Please input X:');
    readln(X);
    Y:=sqr(x)+SH(X);
    writeln('Y=',Y);
end(of Exoffuct).
```

用户自定义函数 SH(X)的引用是在主程序中的表达式中出现的。

DP 10.3 过程嵌套与标识符作用域

10.3.1 过程和函数嵌套的概念

过程(包括函数——下同)的嵌套是实现结构程序设计的主要手段之一。递归则是 PASCAL 语言程序设计的重要的描述方法。本节主要介绍嵌套。

嵌套就是一层套一层的程序结构。在过程或函数中还可以说明另一些过程或函数,比如在过程 A 的过程与函数说明部分又说明过程 B,在过程 B 的过程与函数说明部分又说明过程 C……它们的相互关系是嵌套关系。

在上例嵌套结构中,外层过程可以调用其相邻内层的过程,但不能隔层调用内层的过程。也就是说,A 可以调用 B,B 可以调用 C,C 可以调用 D 等;而 A 调用 C 或 B 调用 D 是非法的。但是,内层过程可以调用其外层相邻及隔层的过程。也就是说,D 调用 A、B、C(包含调用 D 自身)是合法的。这种调用关系属递归调用。但这里要强调一点,A 不能是主程序,任何过程都不能调用主程序。

这种调用关系表现为两条规则:任何过程都可以调用直接隶属于它的下一层过程;在同一层过程中,后说明的过程可以调用先说明的过程。

如果违背了这些基本规则,则会在编译时指示程序设计的错误。若一个过程要调用在其后面具体说明的过程,可使用“向前引用”来实现。



10.3.2 标识符的作用域

在一个大程序中,可能包含许多过程说明或函数说明,它们可能分别由不同的程序员编写。因此,就有可能出现多处定义或说明同一标识符,发生重复定义。在 PASCAL 中允许一个标识符在程序的不同位置上用于不同的目的,只要它遵从作用域的规则。

一个标识符在程序中的出现,分为定义性出现与使用性出现两种。

关于标识符的使用与定义,有两条必须遵从的规则:

- (1) 任何标识符必须先定义后使用;
- (2) 在一个程序体中,除过程或函数说明的形参以及相应程序体中的标识符外,任一标识符不可以被定义一次以上。

在过程或函数的使用中使用 goto 语句应注意如下规则:

- (1) 标号的作用域是其被说明的程序体内部,因此,不允许跳入或跳出过程与函数。
- (2) 不准利用 goto 语句从构造语句(复合语句、条件语句、重复性语句)外转入构造语句内部。这条限制规则也适用于程序模块,即不允许从程序模块内部向程序模块外部进行转移,从程序模块的外部向程序模块内部转移也是非法的。

为了进一步理解标识符的作用域,我们引入以下几个概念。

- (1) 全程量。全程量是指在主程序中被说明,并且在其他子程序内部未被重新说明的标识符。因此,这样的标识符,在主程序及该主程序所包含的任何子程序中都有效。
- (2) 局部量。在过程或函数中被定义的,在该程序体以外无效的标识符,称为该程序体的局部变量。



10.4 非局部量与副作用

首先看一个程序的开始:

```
program Sid(input,output);
var
  A,Z:integer;
function Sneaky(X:integer):integer;
begin
  Z:=Z-X;
  Sneaky:=sqr(X);
end{of Sneaky};
begin
  Z:=10; A:=Sneaky(Z); writeln(A,'* ',Z);
  Z:=10; A:=Sneaky(10)*Sneaky(Z); writeln(A,'* ',Z);
  Z:=10; A:=Sneaky(Z)*Sneaky(10); writeln(A,'* ',Z);
end{of Sid}.
```



输出结果为

```

100      * *      0
    0      * *      0
1000     * *     -10

```

虽然变量 Z 都是 10, 但 $\text{Sneaky}(10) * \text{Sneaky}(Z)$ 与 $\text{Sneaky}(Z) * \text{Sneaky}(10)$ 结果不一样。乘法的交换率在此发生问题, 其原因是函数 `Sneaky` 中改变了非局部量 Z 。

在过程体或函数体内如果改变了非局部量的值, 则称这个过程或函数具有副作用。

有时这种副作用的出现改变了程序原来的意图, 使程序验证大大复杂化, 搞乱了程序的结构, 甚至使程序难于理解, 所以应当引起足够的重视。

在函数的使用中关于改变变量参数的值而引起副作用可见下例:

```

function F(var X:integer):integer;
begin
    X:=X-1;
    F:=X;
end.

```

对语句序列

```
A:=3; Y:=A+F(A);
```

和

```
A:=3; Y:=F(A)+A;
```

Y 的结果分别是 5 和 4。这是因为在 F 中改变变量参数 X 的值而引起的。

与此有关的问题是所谓“意外的副作用”。假如你在过程体中使用了变量 X , 却忘记了事先对它进行说明, 这就有两种情况可能发生。一种是外层程序中从未定义过标识符 X , 这时, 编译程序会指出“ X 未定义”的错误。这种错误在明处, 是初学编程的人易犯的毛病。另一种情况是外层程序体中曾经定义过标识符 X , 于是编译程序认定过程中所使用的标识符就是外层定义的那个标识符 X , 这样, X 就成了过程体的非局部量。由于程序在语法上并无错误, 它能通过编译被运行; 但是其运行结果却与实际情况不相符, 而你却因苦苦找不出问题所在而大伤脑筋。

现看一个在过程中对非局部量操作引起副作用的例子, 代码如下:

```

program Add(input,output);
var
    I,J:integer;
procedure Line;
var
    CH:char;
begin
    CH:='*'; I:=1;
    repeat
        write(CH); I:=I+1;
    until I=20;

```



```
writeln;
end;
begin
  write('Please input I,J:');
  readln(I,J);
  Line; writeln('I+J',I+J);
end.
```

运行结果:

```
Please input I,J: 20  5 ✓
*****
I+J=25
```

再运行:

```
Please input I,J: 8  16 ✓
*****
I+J=36
```

第一次运行输出结果看来是正确的,但是,这纯粹是一种巧合;第二次运行时就暴露出副作用对过程的影响。避免这一副作用的方法之一,就是在过程体的变量说明部分,将 I 说明为内部(局部)变量。

DP 习题与思考题

10.1 以下形式参数表中有无错误?为什么?

- (1) (var x,y:real,z:integer);
- (2) (var a:integer;var c:char);
- (3) (var send:real;count);
- (4) (dates:1..31;var week:char)。

10.2 编制一个程序计算输入数据 X 的平方根。(其中有计算平方根值的过程,在过程中有对输入数值合理性的校验,即 X 是否大于等于零。)

10.3 输入实数 U 和 V,应用函数按下列公式计算并输出 Z 值。

$$Z = \frac{F(U,V)}{F(U+V,U/V)+F(3U+2V,3U-2V)}$$

式中,函数 $F(x,y) = \frac{2\sin^2(2x+y) - 3\sin(x+y)\cos(x-y)}{e^x \ln|x| \cdot \text{ch}(x/2)}$ 。

10.4 编程计算组合数 C_m^n 。

10.5 指出下列程序中的全程变量、局部变量、变量参数、数值参数,并写出程序的运行结果。

```
Program Exercise(input,output);
Var
  A,B,C:integer;
```



```
Procedure change(var X:integer;Y:integer);  
Var  
    M,N:integer;  
Begin  
    M:=X * Y; X:=X+1; Y:=Y+10; N:=X+Y;  
    Writeln('X=',X:3,'Y=',Y:3,'M=',M:4,'N=',N:4);  
End;  
Begin  
    A:=3;B:=3;  
    Change(A,B);  
    Change(A,B);  
    Change(A,B);  
End.
```



第 11 章

构造类型

上述章节涉及的数据类型均为简单类型,是用户可直接引用的在 PASCAL 中预定义的标准数据类型——实数类型、整数类型、布尔类型和字符类型以及用户定义后才能使用的枚举类型和子域类型。这些数据类型的特点是该种数据类型的变量本身,都是书写简单的变量标识符(而不再含有其他分量),并以整体标识符的形式出现在表达式中,用单独一个标识符就可表示出来,与这里将要介绍的数组类型相比,它们是非构造的数据类型。

PASCAL 语言为方便地描述形形色色的不同数据对象引入了十分丰富的数据类型,即不仅规定有上述简单类型,还在这些类型基础上,扩充一些可由用户定义的由基本数据类型构造出来的构造类型,包括数组类型、集合类型、记录类型和文件类型。

构造类型的特点是它的每个数据元素是由多个简单类型或其他构造类型的成分按一定方式(法)或规则构造组合而成的。

由此可见,为定义构造类型变量,既应描述它们的成分类型,还要指明其构造方法。



11.1 数组类型

11.1.1 一维数组

在数组类型定义中,下标类型只有一个时,称为一维数组。常用的一维数组是非紧缩数组,即一维数组类型定义的一般格式为

<类型标识符> = array [下标类型] of <元素类型>

例如:

type

Coordinate = (x, y, z);

PointSet = array [Coordinate] of real;

CharSet = array [1..26] of char;

定义了两个一维数组类型。Coordinate 是枚举类型标识符,而 PointSet 和 CharSet 都是一维数组类型标识符。其中 Coordinate 为 PointSet 的下标类型,而 real 为其元素类型。类似的子域类型 1..26 为 CharSet 的下标类型,Char 是其元素类型。

数组类型一旦在类型定义部分定义后,就可以在变量说明部分引用来说明某些变量为数组类型变量。例如:



```
var
    p,q:PointSet;
    u,d:CharSet;
```

就表示 p 、 q 均是 PointSet 类型的数组变量,且均由三个元素构成。 p 的第一元素为 $p[x]$,第二个元素为 $p[y]$,第三个元素为 $p[z]$ 。它们都是 p 数组变量的一个下标变量。 q 和 p 类似。而 u 、 d 是 CharSet 类型的数组变量,且均由 26 个元素组成的字符数组。 u 的第一个元素为 $u[1]$,第二元素为 $u[2]$ ……第二十六个元素为 $u[26]$ 。每个元素都可能是一个 ASCII 字符,也都是 u 数组变量的一个下标变量。 d 和 u 类似。数组元素在内存中存储方式如图 11.1。

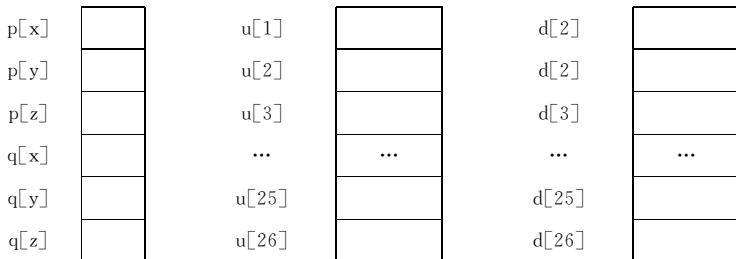


图 11.1 数组元素在内存中存储方式示意图

另外,对数组类型的说明,不仅可按上述方式进行,也可以采取直接在变量说明部分说明数组变量类型的方法。例如:

```
var
    p,q:array [(x,y,z)] of real;
    u,d:array [1..26] of char;
```

说明了 p 、 q 均是由 3 个实数元素, u 、 d 均是由 26 个字符元素构成的数组类型变量。数组的每个元素即下标变量可同一般变量一样被赋值或在表达式中参加运算。例如:

```
p[x]:=1.1;p[y][y]:=2.2;p[z]:=3.3;
q[x]:=p[x];q[y]:=p[y];q[z]:=p[z];
u[1]:=‘a’;u[2]:=succ(u[1]);u[3]:=u[1];
```

如果两数组变量类型一致,如上述的 p 和 q 可直接用数组赋值语句 $q:=p$ 来进行数组的整组拷贝。这相当于把 p 的 3 个成分分别赋给 q 的相应成分。它与用循环语句等效。例如:

```
for i:=1 to 3 do
    q[i]:=p[i];
```

应该注意,在 PASCAL 中无变界数组。在数组类型定义后,数组元素的个数已经确定,以后再用程序语句改变下标的上、下界值。例如:

```
program xyz(input,output);
type
    whole=array [1..n]of integer;
var
    s:whole;
    i,n:integer;
```



```
begin
    read(n);
    s[n]:=5;
    :
```

这段程序说明和程序是非法的。

因为在定义 `whole` 时, `n` 的值不定, 编译程序无法处理; 而在语句部分又本想通过 `read(n)` 来改变 `whole` 数组类型的下标上界, 但因在类型定义部分就未能通过编译, 所以也根本达不到改变数组下标上界的目的。只有在 `type` 保留字前加上 `const` 定义 `n` 是某个常量, 并在语句部分下标值一定不大于该常量时才是正确的 (防止下标越界产生错误)。

1. 数组的访问

欲对一个数组访问时, 一般不能直接对其整体 (数组变量) 进行输入/输出操作, 只能按其下标变量依次对其元素访问, 但是受元素类型操作限制。下面举例说明对不同下标类型和元素类型的数组的访问。

(1) 当数组下标是枚举类型, 元素是实数类型时, 代码如下:

```
program e1 (input,output);
type
    Cood=(x,y,z);
    PointSet=array [cood] of real;
var
    p:PointSet;
    r:Cood;
begin
    writeln('Enter array p:');
    for r:=x to z do
        read(p[r]);
    writeln;
    for r:=z downto x do
        write(p[r]:4:1);
    end.
```

运行结果:

```
Enter array p:
1.1 2.2 3.3 ↵
3.3 2.2 1.1
```

(2) 当数组下标是字符类型, 元素为子域类型时, 代码如下:

```
program e2 (input,output);
type
    Char26=array [char] of 0..255;
var
```



```
tab26:Char26;  
Alfa,i:Char;  
begin  
  for i:='a' to 'z' do  
    tab26[i]:=ord(i);  
    write('Input a letter:');  
    readln(Alfa);  
    for i:='a' to 'z' do  
      if tab26[i]=ord(Alfa) then  
        writeln(Alfa;3,ord(Alfa):4);  
      end  
    end  
  end  
end.
```

运行结果：

```
Input a letter: d ↵  
d 100
```

(3) 当数组下标是子域类型,元素为字符类型时,代码如下:

```
program e3 (input,output);  
var  
  u,d:array [1..26]of char;  
  i:integer;  
begin  
  for i:=1 to 26 do  
    read(u[i]);  
    writeln;  
  for i:=1 to 26 do  
    begin  
      d[i]:=chr(ord(u[i])+32);  
      write(d[i])  
    end;  
  end.  
end.
```

运行结果：

```
ABCD...YZ ↵  
abcd...yz
```

(4) 当数组下标是子域类型,元素为枚举类型时,代码如下:

```
program e4 (input,output);  
type  
  wt=(No,Yes);  
var  
  a:array[0..1]of wt;  
  i:integer;
```



```
begin
    Write('input code:');
    readln(i);
    case i of
        0:a[i]:=No;
        1:a[i]:=Yes;
    else
        writeln('Error !')
    end;
case a[i] of
    No:write('No');
    Yes:write('Yes');
end;
end.
```

运行结果:

```
input code:1 ↵
Yes
```

由此可见,对数组的访问是通过访问某一个数组元素进行的,而对某一个数组元素的访问又是通过下标变量访问的。元素类型决定能否直接通过 read、write 过程语句来读/写其值(枚举类型不能直接用 read/write 过程语句)。数组变量在内存中存储方式是以元素为单位顺序存放的。有两个以上数组时,按其定义顺序依次排列(如图 11.1 所示)。数组变量访问分随机访问和顺序访问两种。随机访问是随机处理某一个元素;顺序访问是当对数组所有元素统一处理时,通常要从数组的第一个元素开始,依次处理数组的每个元素,直到最后一个元素结束。

2. 应用举例

【例 11.1】 键盘输入 10 个整数,输出其中最大值和最小值。其代码如下:

```
program maxmin(input,output);
const
    n=10;
type
    range=array [1..n]of integer;
var
    a:range;
    max,min,i:integer;
begin
    Write('input number:');
    for i:=1 to n do
        read(a[i]);
    writeln;
    max:=a[1];
```



```
min:=a[1];
for i:=2 to n do
  if a[i]>max then
    max:=a[i]
  else if a[i]<min then
    min:=a[i];
write('max=',max,' min=',min);
end.
```

运行结果:

```
input number: 6 4 9 5 7 4 8 2 3 6 ✓
              max=9   min=2
```

3. 下标越界及检查

数组使用常容易出现的错误是数组元素的下标超出在对数组类型定义时规定的界限。数组的使用不能越界,只能在定义类型时所定义的范围内引用其下标,否则一旦越界将侵占相邻存储单元,错误地改变其他存储单元的内容就会造成严重的错误。例如在程序的变量说明部分有如下说明:

```
a:array[1..3] of integer;
b:array[1..5] of integer;
```

而在语句部分有

```
for i:=1 to 5 do
  b[i]:=i;
for j:=1 to 5 do
  a[j]:=j+1;
```

此时,如果不在程序的开始部分加编译器指示{\$R+}进行编译检查,在j循环内将给a[1]、a[2]...a[5]赋值。虽然a[4]、a[5]已经超出数组定义的界限,但此时程序尚能正常运行,而且没有任何出错信息。若忽略数组下标越界错误,b[1]、b[2]的值为1和2,见图11.2(a)。但是,数组变量在内存中存储方式是以元素为单位顺序存放的,并且先定义a放在b前面,所以该程序数组下标越界出错后,执行的结果是把a[4]、a[5]的值放在a[3]后面的存储单元b[1]和b[2]中,从而改变b[1]、b[2]的原值使之分别变成5和6,如图11.2(b)。这种错误是很危险的,因为它直接影响了程序运行结果。而编译程序时并没有错,错误只在运行时出现。

由下标越界引起的错误常使用户费很多时间去找源程序的错误,而又很难找出错误之所在。

因此设计程序时必须注意,尽量避免下标越界。

TURBO PASCAL 中提供了检查下标越界的方法,即在程序首部后加编译器指示{\$R+},使系统在运行时进行下标越界检查,当越界时给出越界出错信息,但增加运行时间。建议调试时加入{\$R+},而调试结束后撤消检查(缺省时为{\$R-},表示不进行检查)以保证运



行速度。能否把数组越界消灭在调试运行之前,这是我们在程序设计时应特别考虑的问题。对此没有一个普遍通用的措施,只能在实践中根据具体情况进行具体分析采取相应措施。例如,在下面两程序段中,写成右边的为好。

a[1]	2	a[1]	2
a[2]	3	a[2]	3
a[3]	4	a[3]	4
b[1]	1	b[1]	5
b[2]	2	b[2]	6
b[3]	3	b[3]	3
b[4]	4	b[4]	4
b[5]	5	b[5]	5

(a)
(b)

图 11.2 数组下标越界示意图

```

:
var
  a:array[1..100] of real;
  i:integer;
  :
  :

for i:=1 to 100 do
  a[i]:=0.0;
  :

```

```

:
const
  m=1;n=100;
var
  a:array [m..n] of real;
  i:integer;
  :
  :

for i:=m to n do
  a[i]:=0.0;
  :

```

这样在程序中当改变数组元素个数时,只须改动 const 中的 m 和 n 就行了。因而可避免由于遗漏修改而造成下标越界的错误。

另外,在输入数据时,以下语句:

```

a:array[1..3] of integer;
:
readln(i,a[i]);

```

是危险的。因为当读入 a[1]为 1000 时,应在键盘上先给 i 一个 1,再给 a[1]键入 1000;如果由于马虎而错打成 1000 1,这就使数组 a 的下标越界,从而造成通常总是正常运行的程序而不能正确运行。如果此时加上判断语句,就可避免这种错误。譬如改成:

```

:
read(i);
if (i>=1) and (i<=3) then
  readln(a[i])

```



```
else  
    writeln('subscript out of range');
```

就不用担心越界了。

此外,使用编译器指示{ \$R+ }也可检查子域类型上、下界越界问题。

11.1.2 多维数组

1. 多维数组的类型定义

由于一维数组元素的类型可以是任何数据类型,那么当它仍然是一维数组类型时,也就构成了二维数组类型。二维数组类型定义的一般格式为

<类型标识符>=array [T1] of array [T2] of <元素类型>

其中,T1、T2 为下标类型。也就是在数组类型定义中,下标类型个数为两个时,称为二维数组。二维数组类型的定义也可简化成如下格式:

<类型标识符>=array [T1,T2] of <元素类型>

二维数组类型可以描述矩阵或矩形表之类的二维空间数据。例如:

```
type  
two=array[1..5,1..3] of real;  
var  
    a:two;
```

表示数组 a 有 5 个元素,而每个元素又都是一个一维数组 array [1..3] of real,可看成是由 5 行 3 列数据构成的矩阵,即由 15 个元素组成的矩形表。其元素分别为

a[1,1]	a[1,2]	a[1,3]
a[2,1]	a[2,2]	a[2,3]
a[3,1]	a[3,2]	a[3,3]
a[4,1]	a[4,2]	a[4,3]
a[5,1]	a[5,2]	a[5,3]

这里 a[i,j](1≤i≤5,1≤j≤3)表示 a 的第 i 行、第 j 列处的元素,其值为一个实数,被存到计算机内存中的连续存储单元中。但因计算机的存储器是一维的,要把二维或更高维数组元素存放内存中,通常先按行、后按列的方向存放。这里依次为 a[1,1]、a[1,2]、a[1,3]、a[2,1]、a[2,2]…a[5,1]、a[5,2]、a[5,3]。

类似的,数组类型定义中,下标类型个数为 n(n≥2)时,称为多维数组(n 维数组)。多维数组的一般定义格式为

<类型标识符>=array[T1] of array[T2]of... array[Tn] of <元素类型>

或简化为<类型标识符>=array[T1,T2,T3,...,Tn] of <元素类型>

n 维数组类型是对 n 维空间数组的描述。当 n=3 时,它描述的是三维空间的立方体。例如:

```
type
```



```
ThreeDimention=array[1..3,1..2,1..3] of real;
var
    b:ThreeDimention;
```

说明 **b** 是具有 **ThreeDimention** 类型的数组。**b** 是有 3 个元素,而每个元素又都是 2 行 3 列的二维数组。于是数组 **b** 是共有 $2 \times 3 \times 3 = 18$ 个元素的三维数组。它在计算机内存中的存放顺序为 **b**[1,1,1]、**b**[1,1,2]、**b**[1,1,3]、**b**[1,2,1]、**b**[1,2,2]、**b**[1,2,3]、**b**[2,1,1]、**b**[2,1,2]、**b**[2,1,3]、**b**[2,2,1]、**b**[2,2,2]、**b**[2,2,3]、**b**[3,1,1]、**b**[3,1,2]、**b**[3,1,3]、**b**[3,2,1]、**b**[3,2,2]、**b**[3,2,3],即对多维数组是按先第一下标、后第二下标、再第三下标的行优先顺序依次存放的。

一般说来,对于多维数组,除下标个数比一维数组多外,所有的语句要求和限制全和一维数组相同,不再赘述。

例如,不用条件语句,直接形成并打印出 n 阶单位方阵,可使用如下程序:

```
program unitmatrix(input,output);
const
    n=5;
var
    a:array[1..n,1..n] of 0..1;
    i,j:integer;
begin
    for i:=1 to n do
        begin
            for j:=1 to n do
                begin
                    a[i,j]:=(i div j) * (j div i);
                    write(a[i,j]:3);
                end;
                writeln;
            end;
        end;
end.
```

这里之所以不使用条件语句就能达到使矩阵 **a** 的主对角线元素为 1 而其他元素为 0 之目的,是因为对 **a**[*i*,*j*]采用表达式: $(i \text{ div } j) * (j \text{ div } i)$ 赋值,当 $i=j$ 时, $i \text{ div } j=1, j \text{ div } i=1$,因而 $a[i,j]=1$;而当 $i>j$ 或 $j>i$ 时,都有 $a[i,j]=0$ 。因为此时或者 $i \text{ div } j=0 (i<j)$,或者 $j \text{ div } i=0 (j<i)$ 。程序运行后,输出结果如下:

```
1  0  0  0  0
0  1  0  0  0
0  0  1  0  0
0  0  0  1  0
0  0  0  0  1
```



2. 多维数组的应用举例

【例 11.2】 编一个 n 阶方阵转置的程序。

矩阵转置的含义是将矩阵的行列交换,但主对角线上的元素不变,如:

$$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix} \text{ 转置为 } \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$$

假设用数组 a 描述矩阵,则转置实为 $a[1,2]$ 与 $a[2,1]$ 对换, $a[1,3]$ 与 $a[3,1]$ 对换, $a[2,3]$ 与 $a[3,2]$ 对换,即 $a[i,j]$ 和 $a[j,i]$ 交换。其程序如下:

```
program ex(input,output);
const
    max=10;
type
    index=1..max;
    matrix=array[index,index] of integer;
var
    i,j,n:index;a:matrix;temp:integer;
begin
    write('Input n=');readln(n);           {读入阶数}
    for i:=1 to n do                       {输入矩阵}
        begin
            for j:=1 to n do
                read(a[i,j]);
            readln;
        end;
    for i:=1 to n                         {转置操作}
        for j:=i+1 to n do
            begin
                temp:=a[i,j];
                a[i,j]:=a[j,i];
                a[j,i]:=temp;
            end;
    for i:=1 to n                         {输出转置矩阵}
        begin
            for j:=1 to n do write(a[i,j]:4);
            writeln;
        end;
    end.
```

运行结果:



Input n=3 ✓

1 2 3 ✓

4 5 6 ✓

7 8 9 ✓

1 4 7

2 5 8

3 6 9

由于转置矩阵行列交换而主对角线上的元素不变,因此为了防止已交换过的两个元素再次被交换,转置操作中控制列号的变量 j 是从 $i+1$ 循环到 n ,而不是从 1 循环到 n 。另外,也可以通过 $b[j,i] := a[i,j]$ 再生成一个转置矩阵。

11.1.3 字符数组和字符串类型

1. 字符数组

由前面两节及其例子不难看出,数组不论下标有几个,类型如何,其元素类型都相同。如果数组元素类型为字符类型,称为字符数组。如变量 `pcode`,`qcode` 均是字符数组变量。例如:

```
type
    Code = array [1..26] of char;
var
    Pcode: code;
    Qcode: array [1..26] of char;
```

【例 11.3】 输入一行字符,若超过 15 个,则输出前 15 个字符,(不含空格符),否则用 ‘*’ 补齐 15 个字符。其程序如下:

```
program trim15(input,output);
type
    s = array [1..15] of char;
var
    t: s;
    i: integer;
begin
    for i:=1 to 15 do
        if not eoln then
            repeat
                read(t[i])
            until t[i]<>' ';
            else t[i]:='*';
    writeln;
    for i:=1 to 15 do
        write(t[i])
```



end.

运行结果:

```
i am a student ✓  
i amastudent * * * *
```

程序中, `eofln` 是 1、5、4 中布尔类型中提到的标准函数, 当读到行结束符(回车符)时, `eofln` 为 true, 此时输入字符读完; 若循环未结束, 则双分支语句执行 `else` 后的 `t[i] := '*'`, 即数组 `t` 中剩余元素值将被赋值为 `'*'`。

字符数组元素可以使用 `read`、`write` 过程被直接读/写以及赋予单个字符。字符数组变量可以直接使用 `write` 过程, 但仍不能直接使用 `read` 过程。例如, 对上例的数组变量 `t`, 如下语句:

```
read(t[i]); write(t[i]); t[i] := succ('a'); write(t);
```

都是合法的, 也就是可以用 `write(t)` 语句替换

```
for i := 1 to 15 do  
  write(t[i]);
```

但不能用 `read(t)` 语句替换

```
for i := 1 to 15 do  
  read(t[i]);
```

另外字符数组元素也可以与其他字符型变量相互赋值。例如当 `Ch` 被说明为字符型变量时, `Ch := t[i]` 就是合法的赋值语句。

2. 字符串类型

在前面一些程序里, 特别是在 `write` 过程中, 为了输出提示字符, 已不止一次地使用过字符串常量。例如, `'total sum'`, `'Input number'` 就是两个字符串常量。字符串常量其值为一对撇号之间的字符序列, 但不包括撇号。一个字符串里包含的字符个数称为字符串长度。例如上面第一个串长为 9 (含空格字符), 第二个串长是 13。当两个撇号之间不含任何字符时, 称为空串。空串的长度为 0。空串和任何字符串变量类型相容, 并可为其赋初始值。

为了处理字符串方便, TURBO PASCAL 提供了一种新的数据类型: 字符串类型。字符串类型是构造类型, 在很多方面和字符数组类型有相同之处。例如, 都靠下标引用其中的元素(字符), 都可以被读/写、赋值或当作过程或函数的形参, 此时其类型又都只能以类型标识符的形式出现在形参表中。然而, 它们之间还存在一些差别。其中最主要的两个差别是: 字符串中的字符个数(即字符串长度)可以从 0 到所指出的上限间动态变化, 而数组元素的个数是固定不变的; 在内存中的存储方式不同; 字符串变量可以直接用 `read`、`write` 过程进行读/写, 而数组变量一般不能(字符数组变量可以用 `write` 过程)。

(1) 字符串类型定义

字符串类型定义的一般形式为



<类型标识符>= string [长度];

string 为保留字, 整型常数指明了字符串的最大长度, 其值不能超过 255。当长度及其外方括号省略时, 表示长度为 255。例如:

```
type
    fileName= string[15];
    texLine= string[80];
var
    name:FileName;
    book:TexLine;
```

定义了最大长度分别为 15 个字符和 80 个字符的两个字符串类型。被说明为字符串类型的变量称为字符串变量, 上例中的 Name、Book 都是字符串类型变量。

字符串变量在内存中所占的空间是它的最大长度加 1B。第一个字节用来存放所给字符串变量的实际有效长度(<最大长度), 可用 0 下标去访问。例如对上述说明的 Name 字符串变量, ord(Name[0]) 即指出了 Name 字符串中实际字符的个数。

字符串变量中的实际有效字符, 从第二个字节开始存放, 每个字符占 1B。在实际长度值加“1”以外的字节单元中是随机信息。

字符串变量为其中每个字符都规定了相应的下标。字符串中实际字符的下标从 1 到字符串的长度值, 即通过字符串下标可以访问字符串中单个字符。这可以通过在字符串变量标识符后面加上用方括号括起的整型下标表达式来实现, 这一点很像字符数组。

(2) 字符串运算

由字符串常量、字符串变量、字符串函数引用(值为串类型)和字符串运算符组成的表达式称为字符串表达式。利用字符串表达式可以对字符串数据进行运算或操作。

字符串的运算符主要是字符串连接运算符“+”。所谓连接运算就是把第二个字符串拼接到第一个字符串的后面, 结果是一个新的字符串。加号可以进行字符串的连接。尽管也可以用后面将要介绍的连接函数 Concat 完成同样的操作, 但运算符“+”通常更为方便。如果连接后得到新串的长度超过 255, 那么将会产生运行错误。例如:

```
'TURBO'+ 'PASCAL' = 'TURBOPASCAL'
'123' + ',' + '456' = '123.456'
'A' + 'B' + 'C' + 'D' = 'ABCD'
```

关系运算符 <、<=、=、>=、>、<> 可以用于字符串数据间的比较。关系运算符的优先级低于连接运算符的优先级。字符串关系运算的结果是一个布尔值。当两个字符串进行比较时, 是从左到右逐个比较两个字符串中每个字符的 ASCII 码, 直到找到不相同字符为止。此时 ASCII 码值大的字符所在串就大。如果两个字符串长度不同, 且直至较短串的最后一个字符处两者都相同, 则认为较长字符串值大, 较短字符串值小。只有当两个字符串长度相等且对应字符完全一致时, 这两个字符串才被认为是相等的。总之, 字符串的比较是按字典顺序进行的, 排在字典前头的字符串值较小, 后面的字符串值较大。



例如,在下列表达式中:

```
'ABC'='ABC'; 'ABC'<'ABCDE'; 'ABCMN'<>'ABCDN';
'A123'<'ABCEF'; 'ABCMN'>'ABCDNP'; 'Pascal Compiler'<'Pascal compiler';
'a'>'8'; '2'<'12'
```

前七个取值为 true,而第八个取值为 false。对第六个表达式因为左边字符串中的 C 是大写字母而右边字符串中的 c 是小写字母,在 ASCII 字符表中,小写字母排在大写字母之后,所以它的值为 true。而对第七个表达式,串首字符“2”应在“1”之后,所以关系式不成立,取 false 值。

赋值运算符用来把一个字符串表达式的值赋给一个字符串变量。当然被赋值的变量其最大长度要大于等于字符串表达式值的字符个数。所赋的字符串表达式可以是一个字符串常数,也可以是一个已赋过值的字符串变量,还可以是一个字符串函数引用。例如,在执行语句:

```
Age:='fiftieth';
Line:='Mary happy returns on you'+Age+'birthday';
```

后,Line 的最后值是 Mary happy returns on you fiftieth birthday。

如果字符串变量被赋予过长的值(即赋予一个比定义说明时最大长度还多的字符),那么多出的字符将被截掉。如在上面例子中,如果 Age 被定义成属于类型 string[5],那么执行上述赋值语句后 Age 中只含有最左边的五个字符“fifti”。

【例 11.4】 输入若干个英文单词,按字母顺序打印这些单词(所有单词全用大写或全用小写)。

每个单词可以看做一个字符串,对字母来说大小顺序与字典顺序一致,因此本题实质上是对字符串进行排序的问题。前面已介绍过比较法、冒泡法排序,在此将介绍选择法排序。

算法步骤如下:

- ① 假设 a[1] 中的字符最小,记下此最小字符串的位置 p=1。
- ② 依次把 a[p] 和 a[i](i 从 2 变化到 n) 进行比较,每次比较时,若 a[i] 中的字符串比 a[p] 中的字符串小,则把 i 的值赋给 p,使 p 总是指向最小字符串的位置。
- ③ 把 a[p] 和 a[i] 中的字符串对调,最小字符串就在 a[i] 中了。
- ④ 重复以上算法,第二遍从第 2 个到第 n 个字符串中选出最小字符串放入 a[2] 中;第 3 遍从第 3 个到第 n 个字符串中选出最小字符串放在 a[3] 中……此过程重复 n-1 次后,即把 n 个字符串按从小到大顺序排好了。其程序如下:

```
program exmple8(input,output);
const
    n=10;
type
    str=string[n];
var
    a: array[1..n] of str;
    t:str;
```



```
i,j,p:integer;
begin
  writeln('please input word:');
  for i:=1 to n do
    readln(a[i]);
  for j:=1 to n-1 do
    begin
      p:=j;
      for i:=j+1 to n do
        if a[i]<a[p]then
          p:=i;
      t:=a[p];a[p]:=a[j];a[j]:=t
    end;
  writeln('please output word:');
  for i:=1 to n do
    writeln(a[i])
  end.
```

运行结果：

```
please input word;
world ✓
book ✓
:
student ✓
please output word:
book
student
:
world
```

从上面例子可以看出字符串类型数据不仅可以放在读/写过程语句中输入/输出,也可以被赋值,还可以作为其他构造类型数据的基类型构造其他类型数据。

(3) 字符串类型和字符数组类型的关系

在 TURBO PASCAL 中,字符数组可以出现在字符串表达式中。在这种情况下,字符数组被转换成具有该数组长度的字符串。因此,字符数组可以用和字符串同样的方式进行比较操作,并且只要长度相同,就可以把字符串常数赋给字符数组;而字符串变量和字符串表达式计算得到的值不能赋给字符数组。因为它们是不同类型的且字符数组中的元素个数是固定的,而字符串变量和字符串表达式计算出来的值其长度是可变的。

另外,字符数组可以赋给字符串变量。当数组元素个数少于字符串变量所能容纳的最大长度时,取数组全部字符;当多于时,多余的字符被截掉。

其次,字符串变量和字符数组一样都可通过下标访问其中的单个字符。因此,可以把字符



串变量看成是带有长度字节(由零下标指示)的字符数组。

(4) 字符串过程和字符串函数

在 TURBO PASCAL 中提供了如下与字符串类型有关的标准过程和函数,以方便字符串数据的处理和应用。

① Delete 过程

格式: Delete(st,Pos,Num)

功能:该过程从 st 字符串中第 Pos 位置开始删去个数为 Num 个字符的子字符串,其中 st 是字符串变量,Pos 和 Num 都是整型表达式。如果 Pos+Num 的值超过了字符串的长度,即企图删除字符串尾以外字符时,则只有字符串内字符被删掉,Pos 的值在 1~255 之间;如果 Pos>255,便产生运行错误。如 st:='ABCDEFGH';Delete=(st,2,4);将使 st 变为'AFG'。

② Insert 过程

格式: Insert(obj,target,Pos);

功能:本过程把字符串 obj 插入到字符串 target 的 Pos 位置之前。obj 是一个字符串表达式,target 是一个字符串变量,Pos 是整型表达式。如果 Pos 比 target 长度大,那么 obj 就连到 target 之后。如果插入的结果长于 target 的最大长度,多出字符将被截掉,target 中只含有最左面那些字符。如果 Pos 在 1~255 之外,则产生运行错误。例如 st:='ABCDEFGH';那么 Insert('XX',st,3)将使 st 变为'ABXXCDEFGH'。

③ Str 过程

格式: Str(value,st);

功能:该过程把 value 的数值转换成字符串存放在 st 中。其中,value 是一个具有整型或实型的写参数,st 是一个字符串变量。写参数是类似于 write 过程中的可含有域宽和小数位数限制的表达式参数。

例如,I 值为 1234,那么 Str(I;5,st)使 st 值变为'1234',而 x 值为 2.5E4 时,那么 Str(x;10;0,st)使 st 值变为'25000'。

④ Val 过程

格式: Val(st,var,code);

功能:Val 过程把字符串表达式 st 转换成对应的整型或实型数值(根据 var 的类型而定),并把这个值存放在 var 中。st 必须是一个表示数值的字符串,并符合数值常数的形成规则。首尾空格均不得出现。var 必须是整型或实型变量,code 必须是一个整型变量,作为过程执行的出错检查代码。如果没有检测出错误,变量 code 置为 0;否则置为第一个出错字符位置,同时转换结果值为无意义。

例如,st='123',则 Val(st,I,Result)将使 I 值变为 123,Result 值为 0。如果 st 内含有非数值字符,例如'25x',则 Val(st,I,Result)将使 I 值无定义,且 Result 值为 3,指出错误位置是在第三个字符位置上。如果 st 值为'25.4E3',且 x 值是一个实型变量,那么调用 Val(st,x,Result)将使 x 值为 25400,Result 值为 0。

⑤ Copy 函数

格式: Copy(st,Pos,Num);

功能:Copy 函数返回 st 字符串中一个从位置 Pos 处开始的,包含 Num 个字符的子字符



串。这里 `st` 是一个字符表达式, `Pos` 和 `Num` 是整型表达式。如果 `Pos` 值大于 `st` 字符串长度, 那么返回一个空串。如果试图得到字符串尾部以外的字符, 即如果 $\text{Pos} + \text{Num} > \text{st}$ 的长度, 那么只返回字符串内字符; 如果 `Pos` 在 $1 \sim 255$ 之外, 也会引起运行错误。如果 `st` 值为 'ABCDEFGH', 那么:

<code>Copy(st, 3, 2)</code>	返回值为 'CD'
<code>Copy(st, 4, 10)</code>	返回值为 'DEFG'
<code>Copy(st, 4, 2)</code>	返回值为 'DE'

⑥ Concat 函数

格式: `Concat(st1, st2, ..., stn);`

功能: 函数 `Concat` 把它所有自变量表示的字符串按所给出的顺序连接起来, 并返回连接后的值。自变量 `st1, st2, ..., stn` 均是字符串表达式, 彼此用逗号隔开。如果结果字符串长度大于 255, 则产生运行错误。它可为连接运算符“+”所取代并且更为方便。例如 `st1` 的值为 'I am', `st2` 的值为 'student', 则 `Concat(st1, 'a', st2)` 返回 'I am a student'。

⑦ Length 函数

格式: `Length(st);`

功能: `Length` 函数返回字符串表达式 `st` 的长度值, 即字符串 `st` 中字符的个数。结果类型是整型。例如, `st` 值为 'pascal compiler' 则 `Length(st)` 返回值为 15。

⑧ Pos 函数

格式: `Pos(obj, target);`

功能: `Pos` 函数返回字符串 `obj` 在字符串 `target` 中的第一次出现的位置号。`obj` 和 `target` 均是字符串表达式, 值的类型为整型。字符串中第一个字符位置号为 1, 如果 `target` 中没有匹配的字符串, `Pos` 返回 0 值。

例如 `st` 的值为 'see you tomorrow' 那么 `Pos('you', st)` 返回值 5, `Pos('seek', st)` 返回值 0。

这里一定要完全匹配, 即只有 `target` 中完全包含有 `obj` 字符串时, 才认为是完全匹配。在完全匹配情况下, 返回的是字符串 `obj` 的第一次出现时其中第一字符在 `target` 中的位置。

下面举例说明字符串函数的应用。

【例 11.5】 随机输入一个长度不超过 255 的字符串, 倒置后输出。字符串读入到字符串变量 `letters` 中。其程序如下:

```
program Inverse(input, output);
var
  Letters: string [255];
  i, len: integer;
  ch: string [1];
begin
  writeln('Enter a string to letters:');
  readln(Letters);
  len := length(Letters);
  for i := 1 to len do
    begin
```



```
ch:=copy(Letters,len,1);  
delete(Letters,len,1);  
insert(ch,Letters,i);  
end;  
writeln(Letters)  
end.
```

运行结果：

Enter a string to letters:

ABCDEFGH ✓

HGFEDCBA

本程序是使用 Copy 函数、Delete 过程、Insert 过程实现倒置的,并且仍放在字符串变量 Letters 中。每次循环把尾字符复制到 ch 中,然后将字符串尾字符删除,再将 ch 中字符插入到字符串的第 i 位前。请读者考虑,每次循环将 ch 中字符插入到字符串第 1 位前,则字符串将发生什么变化?

DP 11.2 集合类型

1. 集合类型的定义

集合是由具有某些共同特性的元素构成的一个整体。在 PASCAL 语言中一个集合的元素必须属于同一种数据类型,该类型称为集合的基类型。集合类型由用户在类型定义部分中自己定义,定义形式如下:

```
type  
类型标识符 = set of 基类型;
```

其中, set、of 为保留字,标志着集合类型。基类型可以是整数类的子域、字符类型、布尔类型、枚举类型和子域类型。基类型表示集合中元素的类型。

下面是集合类型定义的例子:

```
type  
CharSet = set of Char ;  
BoolSet = set of boolean ;  
Week = (SUN, MON, TUE, WED, THU, FRI, SAT) ;  
Ch = A .. Z;  
Int = 0..255 ;  
WeekSet = set of Week ;  
ChSet = set of Ch ;  
IntSet = set of Int;
```

在上面的类型定义中,集合类型 CharSet 所代表的是由 ASCII 码字符集构成的集合,基类



型是字符类型。集合类型元素包括显示字符和控制字符；集合类型 BoolSet 所代表的基类型是布尔类型的集合；集合类型 BoolSet 包括有 false 和 true 两个元素。WeekSet 的基类型是枚举类型 Week；ChSet 和 IntSet 的基类型都是子域类型，其元素分别为字符 A~Z 和 0~255 的所有整数。

在进行了上面的类型定义之后，标识符 CharSet、BoolSet、WeekSet、ChSet 和 IntSet 代表了集合类型，接着可以在变量说明部分进行变量说明，说明某变量属于某一集合类型。变量说明部分如下：

```
var
    CSVar: CharSet ;
    BSVar: BoolSet ;
    WSVar: WeekSet ;
    ChSVar: ChSet ;
    IntSVar: IntSet ;
```

上面的类型定义部分和变量说明部分也可以合并写为

```
var
    CSVar: set of char ;
    BSVar: set of Boolean;
    WSVar: set of (SUN, MON, TUE, WED, THU, FRI, SAT) ;
    ChSVar: set of A .. Z ;
    IntSVar: set of 0 .. 255 ;
```

TURBO PASCAL 规定一个集合的元素个数最多不得超过 256 个。基类型的序数值必须在 0~255 范围内。实数类型不能做为集合类型的基类型，否则将造成错误。例如：

```
type
    Int = set of integer;
    Int1 = set of -10..10;
    Int2 = set of 200..300;
    Rs = set of real;
```

在上例中，集合类型 Int 的基类型为整数类型是错误的，因为整数类型的取值范围为 -32768~+32767 的全体整数，其元素个数超过了 256 个，序数值也超出了 0~255 的范围；集合类型 Int1 和 Int2 也是错误的，它们的元素个数虽然没有超过 256 个，但是其序数值超出了 0~255 的范围；集合类型 Rs 的基类型是实数类型，因而也是错误的。

集合类型的一个元素在内存中占用一位。由于其元素个数不能超过 256 个，所以一个集合类型变量在内存中占用的字节数最多不会超过 32B。

2. 集合类型的运算规则

(1) 集合类型的赋值运算

集合类型变量的取值是其基类型元素的一个集合。集合类型变量的取值范围是包括空集合在内的所有子集，即集合类型变量的值可以是由集合中若干元素构成的集合，或者是不包含



任何元素的集合——空集合。空集合可以属于任何集合类型。集合类型的值的表示方法是用方括号括起来的元素序列,元素之间彼此用逗号隔开。空集合是用左方括号和右方括号表示。例如,三原色为红、黄、蓝三色,若定义为集合类型,定义如下:

```
type
    Color = ( Red, Yellow, Blue );
    BasicColor = set of Color ;
var
    BCV1, BCV2, BCV3 : BasicColor ;
```

集合类型 BasicColor 的取值为下列集合:

```
[ ];
[Red];
[Yellow];
[Blue];
[Red, Yellow];
[Yellow, Blue];
[Red, Blue];
[Red, Yellow, Blue];
```

下面列举几个赋值运算的例子:

```
BCV1 := [ ];
BCV2 := [Red, Yellow, Blue];
BVC3 := [Blue];
```

BasicColor 类型的基类型有 3 个元素,故其取值个数为 $2^3 = 8$,前面已列举了其全部的取值。一个集合如果包含有 n 个元素,则该集合的取值个数为 2^n 个,其中包括一个空集合。

集合中元素之间没有次序,只要元素相同,不论次序怎样即为同一集合。例如:

[Red, Yellow]和[Yellow, Red]

为同一集合。集合中的元素不能重复出现,相同元素只能出现一次。

对于程序中定义的每个顺序类型或标准类型中的顺序类型,不论是否定义了它为某集合类型的基类型,总可以用该顺序类型的某些值作为集合的元素,表示成集合的形式。这种形式的集合可以出现在 PASCAL 的任何程序中。例如:

```
[1, 2, 5], [A, B, C], [false, true]
```

又如,程序中定义了下面的枚举类型:

```
Month = (JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV,
DEC) ;
```

则不论是否定义了类型

```
set of Month ;
```

下列集合的值都是合法的,即



[JAN, FEB, DEC],[AUG, APR, OCT, FEB]

对于元素序数连续的集合,可写为子域的形式,例如:

[1, 2, 3, 4, 5]

可以表示为

[1..5] ;

[FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP]

可以表示为

[FEB..SEP] ;

[a, b, c,d, A, B, C, D]

可以表示为

[a..d,A..D]。

(2) 集合类型的关系运算

集合类型的关系运算有以下五种:

= 集合相等

<> 集合不等

>= 集合包含

<= 集合被包含

in 集合成员检查

集合类型的关系运算的结果为布尔类型。

关系运算符=和<>分别用来测试集合的相等和不等。

设有集合 A 和集合 B,若集合 A 和集合 B 的所有元素都相同,则称集合 A 和集合 B 相等;否则集合 A 和集合 B 不等。

若 A、B 两个集合相等,则表达式

A=B

的值为 true;否则其值为 false。

若 A、B 两个集合不等,则表达式

A<>B

的值为 true;否则其值为 false。

例如,类型定义和变量说明部分如下:

type

BasicColor=set of (Red,Yellow,Blue)

var

BCVar1, BCVar2,BCVar3, BCVar4; BasicColor ;

若 BCVar1 和 BCVar2 赋以下列值:

BCVar1 :=[Red, Yellow, Blue] ;



```
BCVar2 := [Blue, Yellow, Red] ;
```

则表达式

```
BCVar1 = BCVar2
```

的值为 true。

若 BCVar3 和 BCVar4 被赋值为

```
BCVar3 := [Red, Yellow] ;
```

```
BCVar4 := [Red, Blue, Yellow] ;
```

则表达式：

```
BCVar3 <> BCVar4
```

的值为 true。

关系运算符 $>=$ 和 $<=$ 分别用来测试集合的包含和被包含。

设有集合 A 和集合 B, 若集合 B 的所有元素全是集合 A 中的元素, 则称集合 A 包含集合 B; 或称集合 B 被集合 A 包含。

当集合 A 包含集合 B 时, 表达式

$A \geq B$ 或 $B \leq A$

的值为 true; 否则为 false。

例如, 沿用前面的类型定义和变量说明部分, 当执行下列赋值运算后, 即

```
BCVar1 := [Red, Yellow, Blue];
```

```
BCVar2 := [Blue, Red];
```

则表达式

```
BCVar1 >= BCVar2 ; 和 BCVar2 <= BCVar1 ;
```

的值为 true。

集合的另一种关系运算是集合成员检查, 其运算符为

in

利用成员检查运算, 可以测试某元素是否是某一集合中的元素。例如:

type

```
Week = (SUN, MON, TUE, WED, THU, FRI, SAT) ;
```

var

```
WV1, WV2: Week ;
```

当进行下列赋值运算后, 即

```
WV1 := TUE ;
```

```
WV2 := SUN ;
```

则表达式



WV1 in [MON..SAT];

的值为 true;表达式

WV2 in [MON..SAT];

的值为 false。

必须注意在集合的关系运算中,其运算对象的基类型必须是类型相容的;否则将出现错误。例如:

[1...5]=['A'..'F'] ;

9 in [A..B] ;

这两个表达式中,其运算对象的基类型是类型不相容的,所以上面的两个表达式均是错误的。

利用集合的成员检查运算,可使某些复杂的表达式简化,并且增加了程序的可读性。例如检查某字符类型变量 Ch 的值是否为元音字母时,如果不用集合运算符 in,其语句应为

if (Ch='a') or (Ch='e') or (Ch='i') or (Ch='o') or (Ch='u') then <语句>;

利用集合的关系运算符 in,可改写为

if Ch in ['a','e','i','o','u'] then <语句>;

又如检查某整数类型变量 M 的值是否在 1~100 之间,语句

if (M>=1) and (M<=100) then <语句>;

可改写为

if M in [1..100] then <语句>;

可见采用集合的成员关系检查运算,在处理这类问题时是很简便的。

(3) 集合的并、交、差运算

在 PASCAL 语言中,对集合类型还提供:

+ 并运算

* 交运算

- 差运算

三种集合运算,其运算结果为一集合。

集合的并运算。设集合 A 和 B 是两个相同类型的集合,集合 A 和 B 的并运算记作:

A+B

其运算结果仍为一集合,所得集合是由属于集合 A 或者属于集合 B 的全部元素组成,即集合 A 中的元素加上集合 B 中的与 A 不同的那些元素组成的集合。例如集合:

A:['M','N','O','P','Q','R'] ;

B:['O','P','S','T'] ;

则

A+B:['M','N','O','P','Q','R','S','T'] ;



利用集合的并运算,可以向一个集合增加一些元素。

集合的交运算。设集合 A 和 B 是两个相同类型的集合,集合 A 和 B 的交运算记作:

$A * B$

其运算结果仍为一集合,所得集合是由既属于集合 A 又属于集合 B 的所有元素组成,即是由两个集合中都具有的元素组成的集合。例如集合:

A: ['M', 'N', 'O', 'P', 'Q', 'R'];

B: ['O', 'P', 'S', 'T'];

则

$A * B$: ['O', 'P'];

如果参加运算的两个集合没有相同的元素,则它们的交运算的结果为空集合。例如集合:

A: ['M', 'N', 'O', 'P', 'Q', 'R'];

B: ['S', 'T', 'U', 'V'];

则

$A * B$: [];

利用集合的交运算,可以检查两个相同类型的集合是否有相同的元素。例如,检查某基类型为字符类型的集合 E_n 中有哪些元音字母,可以利用集合的交运算:

$['a', 'e', 'i', 'o', 'u'] * E_n$

所得集合中的元音即为 E_n 中的元音字母。

集合的差运算。设集合 A 和 B 是两个相同类型的集合,集合 A 和 B 的差运算记作:

$A - B$

其运算结果仍为一集合,所得集合是由属于集合 A 而不属于集合 B 的那些元素组成,即是由集合 A 中的元素减去集合 B 中那些与 A 相同的元素后所有元素组成的集合。例如集合:

A: ['M', 'N', 'O', 'P', 'Q', 'R'];

B: ['O', 'P', 'S', 'T'];

则

$A - B$: ['M', 'N', 'Q', 'R'];

如果集合 A 和集合 B 没有相同的元素,则集合 $A - B$ 还与集合 A 相同。例如集合:

A: ['M', 'N', 'O', 'P', 'Q', 'R'];

B: ['S', 'T', 'U', 'V'];

则

$A - B$: ['M', 'N', 'O', 'P', 'Q', 'R'];

利用集合的差运算,可以从一个集合中删除一些元素。

(4) 集合类型数据的输入/输出

集合类型数据不能直接用 read 和 write 过程输入/输出,而必须采用间接的方法。

对于集合类型数据的输入,一般是按照其基类型的输入方法逐一输入其每一元素,同时通过集合的并运算将其各个元素追加到集合类型变量中。

集合类型是一种在数据处理中很有用的构造数据类型。它与数组一样,是由同类型的元



素组成的,不同的是元素的排列顺序是无关紧要的。而且对集合可执行一些有效的快速运算操作。例如,对数组增加或删除一个元素是比较麻烦的,首先要找到插入(或删除)位置,插入(或删除)之前还要将自插入(或删除)元素以后的元素移动位置,这就需要编制一段程序来执行这些操作。而对于集合,只须使用集合运算并或差,再用一个赋值语句就可以进行增加或删除元素的操作。又如,要查找一个元素在数组中是否存在,需要编写一个查找过程来完成这一任务;而对于集合,则只要使用关系运算符 `in` 进行测试,根据测试结果是 `true` 或 `false` 即可判断元素是否在集合之中。当然数组与集合在程序中各有用处,它们并不能完全相互代替,在具体应用中要注意数组元素的有序性和集合元素的无序性及无重复性。

DP 11.3 记录类型

11.3.1 记录类型的定义

平时所说的记录是指一组相关的数据。其中每一个独立的数据称为记录的一个项。它们有的是数值的,有的是非数值的,按某种要求的次序结合为一个整体。相关整体是说,对这组数据要结合起来看才有应用意义。如数据对:

```
Xcoordinate:integer;
Ycoordinate:integer;
```

表示一个二维坐标点。而数据对

```
TelNo:integer;
Name:string[20];
```

表示某人的电话号码。该人的名字用字符串型数据 `Name` 表示,而对应的电话号码用整型数据 `TelNo` 表示,对这对数据分开解释将失去意义。再如下面的一组数据:

```
Day:1..31;
Month:(JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC);
Year:1990..2100;
```

表示英语日期。由子域类型的日、枚举类型的月份及子域类型的年度三项数据组成,合在一起可表示 1990—2100 年度的任何一天。

如果说坐标点的一对数据,由于具有相同的数据类型,还可以用数组类型表示;那么对电话号码、日期的各数据项,由于类型各异,便无法用数组类型表示了。怎样把它们统一成为一个整体的数据类型呢? PASCAL 语言规定用一对保留字 `record` 和 `end` 做括号,把各数据项(可以是相同的数据类型,也可以是不同的数据类型)括在一起,项间用分号分隔符“;”隔开,组成了一个新的构造类型——记录类型。上面三组数据可表示为

```
Coordinate=record
    Xcoordinate:integer;
    Ycoordinate:integer;
```



```
        end;
    Phone=record
        TelNO:integer;
        Name:string[20];
    end;
    Months=(JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC);
    Dates=record
        day:1..31;
        Month:Months;
        Year:1990..2100;
    end;
```

上面 `Coordinate`、`Phone`、`Dates` 三个类型标识符定义了一种新类型。可以看出这样的定义格式是很顺乎自然的。它实现了对不同类型数据项的组合统一。

记录类型的一般格式为

```
<类型标识符>=record
    <标识符表 1>:<类型 1>;
    <标识符表 2>:<类型 2>;
    :
    <标识符表 n>:<类型 n>;
end;
```

我们把记录类型定义中夹在 `record` 和 `end` 间用分号隔开的部分,称为记录段。在记录段内冒号左边的标识符称为记录的成分或域,右边的类型说明用以说明该成分的类型,标识记录域的标识符称为域标识符。如 `day`、`month`、`year` 为记录类型 `Dates` 的域,`TelNo`、`Name` 为记录类型 `Phone` 的域。具有相同类型的域,可以放在同一个记录段内说明,构成标识符表。如下面的记录类型定义:

```
type
    Func=record
        x,y:integer;
        Fun:real;
    end;
```

其中, `x`、`y` 是记录类型 `Func` 的两个成分,但由于它们具有相同的数据类型而被写在一个记录段内。

另外,也可以采取直接在变量说明部分说明记录变量类型的方法。例如:

```
var
    u:record
        x,y:integer;
        Fun:real;
    end;
```



11.3.2 记录的嵌套

当一个记录类型的某一个域的类型也是记录类型的时候,就说发生了记录的嵌套,看下面的例子。

用计算机对图书借阅工作进行管理,为每个学生建立一个记录。记录中除应有学生的自然情况登记,如姓名(name) 性别(sex)和所在系(dept)等项外,还应有借书卡片项(card)。借书卡应能描述所借的书名、借书的日期等。暂假定每位学生只有一张借书卡片,请定义对应的数据描述形式。

先按总体数据结构要求,从简单项建起。根据书卡内容所含项目,设定书卡类型为

```
card=record
    borrow:boolean;
    bdate:Dates;
    Name:string[50];
end;
```

在 Card 记录类型的各域中,Name 记录所借书的名字,采用字符串类型;bdate 域的类型由类型标识符 Dates 说明。Dates 已在 11.3.1 中定义过。可见记录中域类型的说明,可以用以前定义过的类型标识符或者标准类型来说明。Borrow 域要特别说明一下。它被用来作为一个标志,以标志整个记录的有效性,所以选用 Boolean 类型。用其 true 值代表有效,false 值代表无效。这种标志(flag)是计算机中常采用的方法。稍了解计算机存储数据原理的人就会知道计算机内数据的“空白”状态不像一张白纸,它总是有值的(计算机内存的字节总表示某一个值);但这一值不一定有效,如程序没进行初始化前的变量的随机数值就是无效的。当要表明一组相关值有效时,只要用一个特定单元标明也就可以了。Borrow 域的设置就是一个例子。

Card 是记录类型,而其 Bdate 域本身又是一个记录类型,这就发生了记录的嵌套现象。为把学生的自然情况同其拥有的书卡结合成一个整体,进一步做下面的定义:

```
student=record
    name:string[20];           { 学生姓名 }
    SecretCode:integer;        { 自给定密码 }
    Dept:string[5];           { 所在系代码 }
    Cards:card;                { 书卡 }
end;
```

这样定义 Student 类型后,就把学生的各情况统一在一个记录里,具有了完整的意义。这里定义的记录类型 student 的 cards 域是记录类型 card,而 card 的 Bdate 域又是记录类型 Dates,我们称这种嵌套记录又嵌套的情况为两级嵌套。在 TURBO PASCAL 中这种嵌套最多可达九级。

11.3.3 记录类型变量及其访问

如前所述,定义一种数据类型的目的是为说明某些变量属于这一类型,进而对这一变量进



行访问。被说明为记录类型的变量称为记录类型的变量,可以在程序的变量说明部分用记录类型标识符说明,例如:

```
var
    Today,tomorrow:dates;
    Students:Student;
```

其中 Today、tomorrow 被说明为 Dates 记录类型变量,Students 被说明为 Student 记录类型变量。例如:

```
var
    Today,Tomorrow:record
        day:1..31;
        Month:(JAN,FEB,MAR,APR,MAY,JUN,JUL,AUG,SEP,OCT,NOV,DEC);
        year:1990..2100
    end;
```

用这种方法说明的记录变量 Today、Tomorrow 与前面的分开说明方法是一样的。但考虑类型的一致性时还要视具体情况决定采用哪种说明方式。

和数组类型变量的内存分配方式比较,数组是以元素(即带下标变量)为单位分配内存单元;而记录类型则是以记录变量的域为单位分配内存单元。由于数组元素的类型都是相同的,所以每个元素分配的内存单元一样多。记录类型变量的各个域的类型可能不同,所以每个域分到的内存单元就可能不一样多。和数组变量的相邻下标变量在内存中相邻分配的情况类似,记录变量的相邻域在内存中也是相邻分配的。图 11.3 示出了由前面定义的 Card 类型定义的变量的内存分配示意图(设每横格代表一个字节)。

简单变量可通过其标识符直接访问,即简单变量标识符可直接出现在表达式中(读取访问)或赋值语句的左端(写入访问)。记录变量的结构是复合构造的,访问时除要给出记录变量标识符外,还要指明要访问的域。如果域是嵌套的,则还要进一步指明域中的域名,直至最后的域是所要访问的成分。

记录变量的成分用域命名符来进行标识。这种用域命名符对记录成分的访问方法有时称点域法。例如,访问前面定义的记录类型变量 Today 的 Day 域,并赋值 21,则语句:

```
Today.day:=21;
```

是合法的。要输出记录变量 Students 的 Cards 域的 Day 域值到显示器上,下面的过程调用语句:

```
write(Students.cards.Bdate.day:10);
```

是合法的。对域命名符操作的合法性要依据其类型来决定,例如:

```
write(Students.cards.Bdate.Month);
```

语句调用有语法错误。因为域命名符 Students.cards.Bdate.Month 的类型是枚举类型,而枚举类型变量是不能直接用 Write 语句输出的。可见,域命名符可以像变量一样进行访问,其类

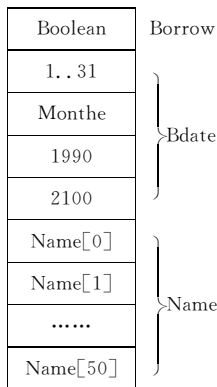


图 11.3 Card 类型记录变量的内存分配图



型为最后一个域所属的类型,其操作受最后一个域所属类型限制。

记录的成分是相对的,这是由记录的嵌套结构造成的。如前面例子中的记录类型变量 Students,域命名符 Students. cards 为一个成分,而 Students. cards. Bdate 也为其中一个成分。由于 Bdate 仍是记录类型,域命名符 Students. cards. Bdate. year 仍是其一个成分。为今后叙述的方便,我们把记录类型中域类型仍为记录类型的域标识符也称为记录变量,以区别非记录类型的域的域标识符。

【例 11.6】 用 student 类型为元素类型,定义一个数组变量,代码如下:

```
StudentRec:array[1..100] of student;
```

这一数组相当于 100 个学生的借阅证,现要把第 i 号借阅证发给一个新生,设该新生的情况为

姓名:zhang xinhua;

自给密码:1122;

所在系:Z9119;

请写出登记的对应程序段。

用点域法写出为

```
studentrec[i].name:='zhang xinhua';
studentrec[i].secrecode:=1122;
studentrec[i].dept:='Z9119';
studentrec[i].cards.borrow:=false;
```

最后一条语句的作用是把该新生的书卡置成“空”,即未借书的状态。

和过程、函数的嵌套结构类似,记录类型的嵌套也可划分为层次。把夹在说明某记录变量的记录类型 record 与 end 间的各记录段,称为同层记录段。同层记录段的标识符称同层域标识符或同层的记录成分。这些记录成分被称作该记录变量的直属成分。对记录成分仍为记录类型的,其直属成分被称做下层成分,依次类推。

如对 student 类型,name、secretCode、Dept、cards 为其同层的直属成分,属第一层;而 borrow、Bdate、Name 又为 Cards 的直属成分,为第二层成分;类推 year、month、Day 又为第三层记录成分。

记录变量划分为层次后,也有域标识符的作用域问题。

仔细观察 student 类型定义,我们发现,在第一层的各域中有域名为 Name,而在 Cards 变量的域内还有一个域标识符为 Name。这样的定义是否发生了重名,即两次定义呢?我们说这里没有发生两次定义。因为 PASCAL 规定,同层 record 与 end 之间直属的域标识符不可同名;但不在同层直属域的域标识符可以同名。像下面这段变量说明:

```
var
  a:integer;
  b:record
    a:boolean;
    b:record
```



```
a:char;  
d:real;  
end;  
c:record  
a:1..20;  
e:set of 1..20;  
end;  
end;
```

是符合 PASCAL 语法的,虽然标识符 `a` 已在定义中出现了四次,但由于它们没出现在同层 `record` 与 `end` 之间的直属域,故没有重名情况。访问时用 `a`、`b`、`a.b`、`b.a`、`b.c`、`a` 等域命名符可以很清楚地区别开。

记录类型变量是由其各成分类型构造的,所以一般讨论记录类型变量的运算是无意义的,只能就其某成分类型进行运算。

11.3.4 开域语句

1. 开域语句的意义

有了 11.3.3 中介绍的点域法,可以用域命名符对记录类型变量的各成分进行访问,其中也包括对嵌套记录类型变量里层各层次诸成分的访问。但我们看到 11.3.3 例【11.6】的 4 条赋值语句前面都缀有标识符 `studentrec[i]`,写起来不方便,占用了源程序的空间,且不易阅读。能否提供一种说明方法,声明这条语句的域标识符都是属于变量 `Studentrec[i]` 的,而不在每条语句上重复书写了呢? PASCAL 提供的开域语句实现了这一需求。

开域语句的一般格式为

```
with [记录变量 1] [,记录变量 2]...[,记录变量 n] do <语句>;
```

其中, `[记录变量 i]` 为记录类型变量标识符,或类型仍为记录类型的域标识符。

格式中 `with`、`do` 为保留字,语句部分可以是简单语句,也可以是复合语句或空语句。

开域语句的作用是把记录类型变量的“墙”层层打开,直至最后一个域标识符。“墙”内的各直属域可由语句部分直接访问。也就是说,在开域语句的语句部分将直接书写所访问的域标识符,而无须再写上“前缀”。由于有效区只在语句部分,所以称这些语句为开域语句的受限语句。这样上例中的语句用开域语句书写就简化成了下面的形式。

```
with studentrec[i] do  
begin  
  Name:=‘Zhang Xinhua’;  
  SecretCode:=1122;  
  Dept:=‘Z9119’;  
  Cards.borrow:=false;  
end;
```

注意:这里的 4 条语句被写在同一复合语句中,因而都在受限范围内。若不加 `begin`、`end`,则只有 `name:=‘Zhang Xinhua’`;合法。其中第四条语句由于要访问第二层的 `Borrow` 域,而



开域语句只打开了第一层“墙”，所以还要对里层用点域法，写成 `Cards.borrow` 的形式。如果要访问第二层的多个成分，是否还可用一次开域语句呢？即开域语句的受限语句本身是否还可为一个开域语句？PASCAL 语言允许这样做，这就是开域语句的嵌套用法。

2. 开域语句的嵌套

11.3.2 节中介绍了记录类型的嵌套结构，和这种结构对应的开域语句访问方法是开域语句的嵌套结构。例如当我们要把 21、Jan、1991 这一借书日期赋给记录变量 `studentrec[i]` 的借卡域中的借书日期域中的诸直属域时，可用嵌套开域语句：

```
with StudentRec[i] do
  with Cards do
    with Bdate do
      begin
        day:=21;
        Month:=Jan;
        Year:=1991;
      end;
```

这里出现了三级嵌套，为书写方便，PASCAL 语言允许这种嵌套换一种书写方式：

```
with StudentRec[i],Cards,Bdate do
  begin
    day:=21;
    month:=Jan;
    year:=1991;
  end;
```

两种方式的运行结果相同。嵌套开域语句的一般格式为

```
with u1 do
  with u2 do
    .....
  with un do s
```

或

```
with u1, u2,...,un do s
```

其中， $u_i (i=1, 2, \dots, n)$ 为记录变量，即它可以是在变量说明部分说明的记录类型变量标识符，也可以是类型仍为记录类型的域标识符。 s 为受限语句，可以是简单语句，也可以是复合语句或空语句。

在使用开域语句时，要注意以下几点：

(1) 保留字 `with` 和 `do` 之间的标识符 $u_1, \dots, u_i, \dots, u_n$ 必须是记录变量，其中 u_i 必须是在变量说明部分说明了的记录变量。这些记录变量的排列顺序将影响受限语句中标识符的归属，因而不能随意排列，TURBO PASCAL 对其规定了如下原则：

① 记录变量 u_i 之前各记录变量 $u_1, \dots, u_j, \dots, u_{i-1}$ ，不可是 u_i 内层直属或嵌套所属的记录变量，即内层记录不能在外层记录变量之前出现；



② 允许在一个开域语句中同时对几个记录类型变量进行开域处理,即 $u_1, \dots, u_i, \dots, u_n$ 记录变量序列可同时有几个是在变量说明部分说明的记录变量。

下面的例子,有变量定义:

```
var
  rec1:record
    A1:real;
    A2:record
      B1:boolean;
      B2:record
        C1:integer;
        C2:25..30;
      end;
    end;
  A3:record
    B1:set of 1..12;
    B2:array[1..20] of integer;
  end;
end;
rec2:record
  A1:(male,female);
  A2:record
    B1,B2:1..200;
  end;
end;
Norec:integer;
```

则下面的开域语句

```
with rec1,rec2,A2 do S;
with rec1,A2,A3 do S;
with rec1,rec2,A2,B2 do S;
```

都是合法的。它们在一个语句中同时打开了两个记录类型变量的限制域。

一般来说,当同时打开几个记录类型变量的记录域时,总可以在 $u_1, \dots, u_2, \dots, u_n$ 序列中找出若干个子序列 $v_1, v_2, \dots, v_p$ 满足 v_1 为在变量说明中定义的记录类型变量, $v_2, \dots, v_p$ 顺次为该变量内层的各记录变量。

(2) 针对开域语句的记录变量序列受限语句 S 中的标识符声明有明确的规则:从最后一个记录变量 u_n 的直属域开始查找其所属各个域的标识符,如果有同名,就认为受限语句中的标识符声明该域;如无名者,就对记录变量 u_{n-1} 继续进行同样的查找,依次可能一直到 u_1 。如查找到 u_1 也未遇同名域,则认为该标识符是过程部分局部标识符或程序的全局标识符,否则是未定义的标识符。如开域语句:

```
with rec1,rec2,A2 do
```



```
begin
    B1:=35;
    A1:=female;
    A3.B1:=[1,2,3];
    Norec:=8
end;
```

由记录变量系列知 B1 应是 rec2 的一个域标识符而不是 rec1 的一个域的域标识符。因为,对受限语句中的 B1 标识符,应从 A2 的直属域开始找,显然是指其 B1 域,用点域法对应写出为 rec2. A2. B1;同理受限语句中的 A1 对应 rec2. A1;A3. B1 对应 rec1. A3. B1;Norec 是指记录类型变量 rec1、rec2 之外定义的整型变量。

如果把上述开域语句的记录变量改变次序为“rec2, rec1, A2”,那么同一受限语句内的标识符就另有所指了,对应关系为

```
B1:rec1. A2. b1;
A1:rec1. A1;
A3. B1:rec1. A3. B1;
Norec;
```

显然,这样调换后赋值语句将有类型不匹配的问题。从这两个例子可以看出寻找受限语句内标识符所属域的一般方法。

(3) with 语句在开始执行受限语句前,记录变量已确定下来,因而受限语句中对成分的访问也是指这一确定的记录变量。对于记录变量:

```
StudentRec:array[1..200] of student;
```

下列语句:

```
i:=15;
with StudentRec[i] do
begin
    Name:='Zhang Xinhua';
    i:=16;
    SecretCode:=1122;
    Dept:='Z9119';
    Cards. Borrow:=false
end;
```

由于 with 语句前 i 已被赋值 15,所以记录变量已被确定为数组的 15 号元素。其受限语句中的操作只对应 15 号元素的诸域,尽管 i 又被重新赋值为 16,但 1122、'Z9119'、false 等量仍被赋给了数组第 15 号元素的对应域中。

【例 11.7】 输入 30 人 2 门课程成绩,输出每个人的平均分,代码如下:

```
program sum(input,output);
const
    n=30;m=2;
```



```
type
    stu=record
        Name:string[8];
        C:array[1..m] of real;
        Ave:real;
    end;

var
    A:array[1..n] of stu;
    i,j:integer;
begin
    Writeln('Input data:');
    for i:=1 to n do
        with a[i] do
            begin
                read(c[1],c[2]);
                readln(name);
                ave:=(c[1]+c[2])/2;
            end;
        writeln('name c[1] c[2] ave');
    for i:=1 to n do
        writeln(a[i].name:8,a[i].c[1]:6:1,a[i].c[2]:6:1,a[i].ave:6:1);
    end.
```

运行结果：

```
Input data:
98 99 lihong ✓
78 89 wangli ✓
...
name   c[1]   c[2]   ave
lihong  98.0   99.0   98.5
wangli  78.0   89.0   83.5
```

DP 11.4 文件类型

文件作为系统提供的手段,为程序和计算机外部设备相互传输数据开辟了一个通道。文件可以是磁盘文件,也可以是某种逻辑设备的标准文件,如打印机、键盘等。显示器、键盘等逻辑设备作为一种标准文件 output、input,在程序首部的参数表中进行说明,大家已有所了解。磁盘文件作为程序和磁盘存储器交换数据的手段,解决了计算机原始数据和加工结果的长期、大量保存的问题。磁盘文件的定义、操作方法是本节介绍的主要内容。

和数组类型不同,文件类型虽具有有限个数的元素,具有相同类型的成分,但它元素的个



数是不固定的。

本节首先介绍磁盘文件的逻辑结构、文件组织方式,然后介绍磁盘上三种文件:顺序文件、TEXT 文件和无类型文件。最后,为使读者从统一观点了解文件,介绍大家曾接触过的标准文件。

TURBO PASCAL 的文件系统与标准 PASCAL 有较大差别,但笔者认为它的理解和使用更有其方便之处。

11.4.1 磁盘文件的逻辑组织

磁盘是一种可由计算机读/写数据的磁记录设备。和计算机内存储器比较,磁盘可以长期而且大量地(数千兆字节)保存信息,因此又称为外存储器。但由于磁盘的机械运动部件速度受限,计算机和它的信息交换速度远低于和内存的交换。我们不讨论磁盘存储信息的物理机理和过程,只通过逻辑关系使读者理解操作方法。

磁盘以文件为单位存放和管理一组相关数据,而文件又通过文件名由计算机系统使用。文件的命名方法随计算机系统不同而有所差异。在 IBM PC 的 DOS 操作系统支持下,文件名由文件名和扩展名两部分组成,两部分之间用‘.’号分开,如 TEST.PAS、EXAMPL.COM 等。规定文件名最多由 8 个字符,扩展名为 3 个字符组成。字符不做大小写区别,都看成是大写的(详见 IBM DOS 手册)。

磁盘文件的最小逻辑单元是字节,一个文件是一组顺次排列的字节序列。字节序列以一文件结束标记来结尾,如图 11.4 所示。

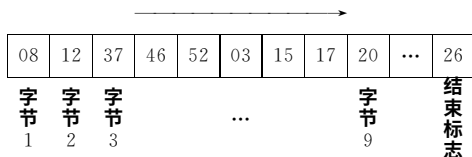


图 11.4 磁盘文件字节排列逻辑图

这一字节序列除存放位置是在磁盘上而不是在计算机内存中以外,在逻辑上和内存中字节序列并无两样。程序中一个变量在内存中占有一组连续的几个字节,字节的多少随变量的类型而异。文件中的字节序列也同样,几个连续的字节构成文件的一个元素。这一元素内字节的排列可以和某变量内存字节排列相同,因而文件中的元素完全可以和程序中的变量一一对应。文件的字节再组合为文件元素后,文件就可以在逻辑上看成元素的序列。依文件字节序列划分为元素的方法不同,TURBO PASCAL 中的磁盘文件分为三类。

1. 随机磁盘文件(或称类型文件)

当磁盘文件元素序列中的每个元素都含有相同个数的字节,且元素内字节的排列次序与某类型变量在内存中的字节排列次序一致时,该文件被称为随机磁盘文件,即随机文件的字节序列按某类型变量所含字节数等量分隔成一个个的元素。变量的类型称为该随机文件的元素类型。



如图 11.4 的文件字节序列按下面的记录类型,即

```
record
    X:0..255;
    Y:0..255;
end;
```

以元素类型来划分,则成图 11.5 所示的元素序列。

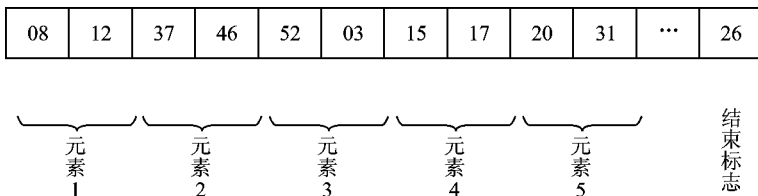


图 11.5 磁盘文件元素的组成

这样我们认为该文件的元素 1 为 $X=08$ 、 $Y=12$;元素 4 为 $X=15$ 、 $Y=17$ 。

为了能对随机文件的任意一个元素进行操作,如读出或写入新的值等,文件的管理系统设置文件指针。图 11.6 表示文件指针指向了第一个文件元素。文件指针总指向元素的首字节。

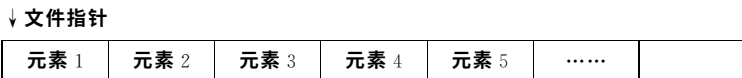


图 11.6 磁盘文件元素排列逻辑图

由于随机文件元素字节数相同,文件管理系统很容易通过跳过的字节数来决定指针指向第几号元素,然后再按指针指向来对元素进行操作。这样看来随机文件很像数组,元素很像数组的元素,指针很像数组的下标,但操作方法上有两个重大差别:首先数组可整组拷贝,文件只能逐个元素地读/写;其次,数组定义时元素个数已固定,而随机文件可在文件尾增添新的元素,因而所含元素的个数是变化的。

2. 正文文件

和随机文件的组织方法不同,正文文件的元素不定长度,每个元素内字节解释成一个字符的 ASCII 码,元素间用 ASCII 码中的回车符 CR(ODH)分隔(有的用 CR/LF 两个字符分隔),如图 11.7 所示(文件中字节数据为十六进制)。图中把 FIRST LINE 与 SECOND 间用 CR 分成两个元素。FIRST LINE 为第一元素,含 10B,第二个元素含 6B。由于每个元素所含字节数不同,系统不能用跳过的字节数来计算所指向的元素,而只能顺次读取,直至见到分隔符(CR)。CR 控制字符的下一相邻字节为下一元素的首字节,也即下一元素的开始。正文文件的元素常被称做一行。

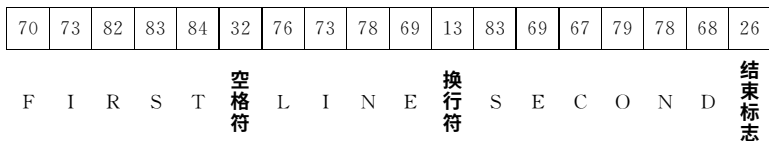


图 11.7 正文文件组织方法



3. 无类型文件

无类型文件是 TURBO PASCAL 的一个特有类型。它的一个元素含 128B, 不对元素做任何意义上的解释。不像随机文件要把元素解释成某种类型, 也不像正文文件那样把元素解释成文字行。

无类型文件对于计算机的底层应用十分方便, 如拷贝一个文件, 或把内存中的一块数据存入磁盘等。

11.4.2 随机文件

1. 随机文件的类型定义

前一节已介绍了随机文件的逻辑结构。它的元素具有一定的数据类型, 这些类型与前几节讲述的具有相同的组织方法。元素依编号顺序排列, 设有文件指针可以随机地指向任一元素。

显然对于不同使用目的文件, 其元素类型将是不同的。程序人员必须在使用文件前对其元素类型进行指明。这就是文件类型定义。它的一般格式是:

〈随机文件类型标识符〉 = file of 〈元素类型〉

其中, 元素类型可以是除文件类型和指针类型以外的任何类型。

和其他数据类型定义的目的相同, 文件类型定义的目的也是为了去说明某变量属于该类型。被说明的变量称为文件变量。

【例 11.8】 文件变量的说明:

```
type
    Coordinate = record
        X, Y: real;
    end;
    Person = record
        Name: string [20];
        Birthday: Date;
        Sex: (male, female);
        Marrital: (Married, Single);
    end;
    FPerson = file of Person;
var
    FCoord: file of Coordinate;
    VPerson, FPerson;
```

例中定义了 FCoord、VPerson 两个类型不同的文件变量。可以看出文件变量的说明方法和其他类型变量的说明方法类似, 可以用文件类型标识符说明, 也可以用变量说明与文件类型定义结合的方法说明。



在文件类型的定义中有一点要提请读者注意,就是从类型一致的角度考虑,文件的元素类型最好用类型标识符,不用直接定义的方法。因而下面的文件类型定义:

```
var
    Fcoord: file of record
        X,Y: real;
    end;
```

是不好的,而应改为

```
type
    Coordinate= record
        X,Y: real;
    end;

var
    Fcoord: file of Coordinate;
```

这一要求是必要的。它能保证文件的元素类型与文件交换数据的内存变量类型一致。如再定义:

```
Vcoord: Coordinate;
```

之后, Vcoord 才能与 Fcoord 的元素交换数据。没有类型一致的保证,程序将有类型不匹配的错误,从而不能通过语法检查。

对文件中元素的访问操作与对其他变量的访问操作有很大的差别。一个完整的文件访问操作,要经过打开文件、设置指针、读或写文件元素和关闭文件几个步骤。这几步操作是通过 PASCAL 提供的标准过程来实现的。下面将分几步介绍。

2. 打开一个随机文件

前面介绍过变量经类型说明后,系统在内存中就为其开辟了一个区域,程序语句按照变量名就可以直接访问了。文件的元素不是在内存,而是在磁盘上。前面说过,磁盘上的文件是按文件名管理的。文件名是按规则排列的字符串。所以,在访问某个名字的磁盘文件之前,要把文件变量和要访问的某文件名联系起来。标准过程 Assign 实现了这一联系。其形式为

```
Assign(Filvar, str);
```

其中, Filvar 是文件变量; str 是字符串表达式,它的值是某磁盘文件的文件名(可以包括路径名)。例如:

```
Assign(VPerson, 'A: RecPerson. DAT');
```

该过程执行后,在程序的后续语句中,凡出现文件变量 VPerson 的时候,都认为是 A 驱动器根目录下的 RecPerson. DAT 文件,直至该文件被关闭。

对文件元素进行读/写访问开始前,还要打开文件。这里分两种情况:

一种是磁盘上已存在这个文件,比如一个图书管理系统,把各学生的记录已存在磁盘上,当要从中读取某个学生的记录时,要先打开它。这个文件就是已存在的文件。打开已存在的



文件的过程格式为

```
Reset(Filvar);
```

其中, `Filvar` 是某类型的文件变量,且在调用该过程前已通过 `Assign` 过程与某名字的磁盘文件联系起来。如前所述,这个名字的文件已在磁盘上存在。

该过程的作用是正式打开程序和磁盘文件交换信息的通道,随后可通过读/写过程读/写文件元素内容。该过程将指针初始化为指向第一个元素。另一种是打开一个新文件,也就是建立一个磁盘上事先不存在的文件。建立新文件的过程一般格式为

```
Rewrite(Filvar);
```

其中, `Filvar` 是文件变量,调用前要与所要建的文件名字用 `Assign` 过程联系。`Rewrite` 过程也将这文件的指针初始化为指向第一个元素(空元素)。

可见打开一个已存在文件的过程为

```
Assign(Filvar, <已存在的文件名>);  
Reset(Filvar);
```

打开一个新文件的过程为

```
Assign(Filvar, <新文件名>);  
Rewrite(Filvar);
```

打开文件时要注意下面几个问题:

- (1) 文件变量的类型要与磁盘文件元素类型一致才有意义;
- (2) 使用 `Reset` 过程时,一定有已存在的与 `str` 同名的磁盘文件,否则将出错(参见 11.4.6 I/O 检查);
- (3) 使用 `Rewrite` 过程时,若已有同欲建文件同名的文件存在于磁盘上,则该过程将置空旧文件的内容,所以要特别注意;
- (4) TURBO PASCAL 规定可同时打开 16 个文件(若欲同时打开更多的文件,详见有关资料)。

3. 随机文件的读/写

随机文件读/写操作前要首先按前节的要求打开文件,然后才能进行读/写操作。

把数据写入文件的过程形式为

```
Write(Filvar, Fvar);
```

其中, `Filvar` 为已打开过文件的文件变量; `Fvar` 为与文件的元素类型一致的变量,是一个内存变量。

该过程的功能通过两个动作实现。

- (1) 将变量 `Fvar` 的内容存至文件指针当前所指的元素位置上;
- (2) 指针下拨一个元素位置,即指向下一个元素,如图 11.8 所示。

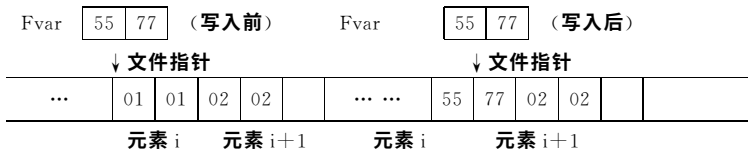


图 11.8 随机文件元素的写入过程

写文件过程还有下面的形式:

```
Write(Filvar, Fvar1, Fvar2, ..., Fvarn);
```

它等效于

```
Write(Filvar, Fvar1);  
Write(Filvar, Fvar2);  
:  
Write(Filvar, Fvarn);
```

读文件过程是把文件上保存的某个元素数据读到内存变量中来,供计算机加工处理。读文件过程的一般形式为

```
Read(Filvar, Fvar);
```

其中, **Filvar** 是已打开的文件变量, **Fvar** 是与文件变量的元素类型一致的内存变量。

过程首先将文件当前指针指向元素的内容赋给内存变量,然后将指针下拨一个元素位置,如图 11.9 所示。



图 11.9 随机文件元素的读出过程

读文件过程的另一形式:

```
Read(Filvar, Fvar1, Fvar2, ..., Fvarn);
```

它等效为过程语句序列:

```
Read(Filvar, Fvar1);  
Read(Filvar, Fvar2);  
:  
Read(Filvar, Fvarn);
```

从上面介绍的读/写过程的功能看, **Read** 和 **Write** 过程提供了内存变量与外存元素之间交换数据的手段。计算机不能直接操作文件上的元素数据,而是将它们交换到类型一致的内存变量中后,通过对内存变量的操作间接完成对文件数据的操作。

前面已介绍的 **Reset** 或 **Rewrite** 过程对指针的作用都是初始化为指向第一个元素,而



read 或 write 过程只能将指针下拨一个。当要访问文件中的任意一个元素,即实现随机访问文件元素时,怎样设置指针呢? TURBO PASCAL 给出了拨指针过程满足了这一要求。其形式为

```
Seek(Filvar,n)
```

其中, Filvar 为已打开的文件变量, n 为欲指向的文件元素的编号。注意第一个元素的编号为 0, 顺次为 1、2... 所以 n 号元素是文件的第 n+1 个元素。这一点要特别注意。n 可以是整型表达式。

Seek 过程的功能是将文件指针拨向第 n 号元素, 成为当前指针位置。当 n 大于文件元素的实际编号时将由系统返回错误(见 11.4.6 I/O 错误一节)。当要求文件长度大于整型值 maxint 时, 可采用长整型文件处理。

4. 关闭文件及文件处理函数

当对文件访问完毕后, 必须关闭文件。关闭文件过程的形式为

```
Close(Filvar);
```

Filvar 是已打开的文件变量。Close 的功能一方面为文件加结尾标志 CtrlZ(ASCII 码为 26), 切断文件变量与具体文件名的联系; 另一方面对操作系统修改文件目录, 反映新文件的变化状态信息, 如将文件长度、修改日期等写入目录。注意对文件操作后一定要执行关闭过程调用。

上面介绍了对文件的基本操作及其实现过程。下面再介绍 TURBO PASCAL 提供的部分标准函数。这些函数为灵活处理文件带来了很多方便。

(1) 文件指针尾位置判断函数

函数形式为

```
Eof(Filvar)
```

该函数的类型为 Boolean, 其中 Filvar 为已打开的文件变量。当文件指针指向文件末尾元素时, 函数返回 true; 而当文件指针不在尾部时, 函数返回 false。

(2) 文件指针当前位置函数

函数形式为

```
FilePos(Filvar)
```

该函数为 Integer 类型。其中 Filvar 为已打开的文件变量。函数返回当前文件指针指向的元素编号。注意: 其值加 1 才是元素真实位置。

(3) 文件长度函数

函数形式为

```
FileSize(Filvar)
```

其中, Filvar 为已打开的文件变量, 函数类型为整型。函数返回文件的长度值(尾元素编号加 1)。利用它和 Seek 函数结合, 能将指针拨到文件尾, 例如:



```
Seek(Filvar,FileSize(Filvar));
```

当文件的参数不能用整型数表示时,即超过 `maxint` 时,TURBO PASCAL 提供了扩充文件函数。

`LongFilesize`、`LongFilePos`、`Longseek` 三个函数,其功能与前面介绍的 `Filesize`、`Filepos`、`Seek` 相对应,只是对应的整型参数或函数值改为实数,以扩大文件容纳的数据容量。

(4) 文件截断过程

过程形式为

```
Truncate(Filvar);
```

该过程截去文件当前指针位置以后的全部记录。

11.4.3 text 文件

当正文文件与磁盘设备相连时,称其为 `text` 文件。`text` 文件的组织方法已在 11.4.1 节中介绍。由于是将英文字符用 `CR`(或 `LF`)分隔成行,所以 `text` 文件存放英文文本最合适,也被称为磁盘文本文件。TURBO PASCAL 用标准标识符 `text` 来标识这种文件类型,可直接说明文件变量 `Filvar` 为 `text` 文件类型。例如:

```
Filvar:text;
```

在随机文件中使用的 `assign`、`reset`、`rewrite`、`close` 过程也适于 `text` 类型文件。其功能和使用方法是一样的,不再赘述。

由于逻辑结构的不同,`text` 文件本身有一些特殊的访问方法与随机文件不同。一个最大的差别在于 `text` 文件元素是字符个数不固定长度的“行”。系统没有办法通过计算来拨动指针到任意行首,因而只能一行行地访问。

下面介绍的 `text` 文件操作的过程和函数是假定 `text` 文件变量已通过 `assign`、`reset`(或 `rewrite`)过程打开,不再一一说明。

(1) `readln(Filvar)`

该过程只能用于 `text` 文件(和前面说过的标准文件)。它的功能是文件指针从当前位置跳至下一行(即下一元素)的开始,即跳过包括下一个 `CR/LF` 在内的所有字符,使下一步的读或写操作从下一行的首字符开始。

(2) `Eoln(Filvar)`

这是一个布尔型函数,也只能用于 `text` 标准文件类型。但 `text` 文件和标准文件的用法略有差别,在 `text` 文件中若当前指针的下一字符为行分隔符或文件结束符,则返回 `true` 值;否则将返回 `false` 值,即在 `text` 文件中 `Eoln` 函数有向前看的能力。在标准文件中则只根据指针当前字符值决定 `Eoln` 的值,无向前看的能力。

(3) `Eof(Filvar)`

同 `Eoln` 函数类似,在 `text` 文件中,当指针位置下一字符为文件结尾符时,返回 `true` 值;否则返回 `false` 值。

(4) `Read(Filvar,var)`

该过程形式上与随机文件的 `read` 过程完全一样。但在随机文件中,要求 `var` 的类型必须



与文件的元素类型一致才有意义。text 文件中 var 的类型要复杂一些。它可以是字符类型变量、字符串类型变量或数值类型(包括整型和实型)变量。

当 var 为字符类型变量时,执行 Read 过程将把文件指针指向的字符赋给 var,然后指针下拨一个字符。当下一字符为控制字符 Ctrl Z(ASCII 码为 26)时,Eof 和 Eoln 同时为 true。当下一字符为控制字符 CR(ASCII 码为 13)时,Eoln 为 true。除上述情况,每读一个字符 Eof 及 Eoln 均为 false。

若 var 为字符串类型变量,和上面的字符类型变量的情况又有所不同。文件将从当前文件指针下的字符开始依次赋给字符串,直至字符串变量被赋满(等于字符串变量定义的长度)或行结束符 CR/LF(13/10)被遇到或文件结束符 Ctrl Z(26)被遇到才停止赋值。

当 var 为数值类型(包括整型和实型)变量时,read 过程将把 text 文件中的字符串转变为数值常数赋给变量 var。text 文件中表示数值的字符串间由空格符(ASCII 码为 32),格式 TAB、CR 或 LF 分隔。read 过程在对 text 文件按数值读取时,指针将跳过这些控制字符读取数字字符串。数字字符串的长度不超过 30,字符串后面必须有分隔符。

text 文件的读入过程 read 对 Eoln 及 Eof 的影响与标准文件 input、output 略有不同。

(5) Write(filvar, var)

write 过程是 read 过程的逆过程。和 read 过程一样,当 Filvar 为 text 型文件时,var 变量可以是字符类型、字符串类型和数值类型(包括整数类型和实数类型)。当 var 为数值类型变量时 write 过程自动将其转为对应的字符串,然后再写入 text 文件。过程将 var 的值从当前指针位置开始依次写入文件中。

(6) Writeln(Filvar)

该过程在文件指针当前位置上写上一个回车符 CR,有的系统为 CR/LF,然后指针下拨一个位置。

(7) Read(filvar, var1, var2, ..., varn)

Read(filvar, var1, var2, ..., varn);

等效于以下语句序列:

Read(Filvar, var1);

Read(Filvar, var2);

:

Read(Filvar, varn);

(8) Readln(Filvar, var)

Read(filvar, var1, var2, ..., varn);

等效于以下语句序列:

Read(Filvar, var);

Readln(Filvar);

(9) Write(Filvar, var1, var2, ..., varn)

Read(filvar, var1, var2, ..., varn);

等效于以下语句序列:



```
Write (Filvar, var1);
Write (Filvar, var1);
...    ...
Write(Filvar, var1)
(10) Writeln(Filvar, var)
Read(filvar, var1, var2, ..., varn);
```

等效于以下语句序列:

```
Write (Filvar, var1);
Writel (Filvar);
```

也与类似的语句序列等效。

(11) SeekEoln(Filvar), SeekEof(Filvar)

这两个函数过程是 text 文件类型固有的。Seek Eoln 使文件指针跳过若干字节, 寻求到第一次遇到的 CR 字符后, 返送 Eoln 为 true 值, 并将指针指向下一个字符; 若遇不到, 则返送 Eoln 为 false 值。SeekEof 与 SeekEoln 相似, 只不过它寻找文件结束符, 并对 Eof 返送结果值。

(12) Append(Filvar)

它与 Reset(Filvar)过程起的作用是相同的, 都是打开一个文件; 但它使文件指针指向文件尾, 下一个操作只能是添加新元素。

text 文件与随机文件在操作上又一很大差别在于: 随机文件打开后可读, 也可写, 即对读/写操作可交替进行; 而 text 一类的正文文件, 则打开后只能读或只能写, 且由打开文件的方式决定。

如执行 reset 过程后, 下一步只能进行读操作; 执行 rewrite 或 append 过程后就只能进行写操作了。

下面用例子来说明 text 文件的使用范围及操作方法。

【例 11.9】 将从键盘上输入的英文文本变换成大写字符后存在磁盘文件上, 遇到 Ctrl-Z 字符结束输入。要求磁盘文件名为 ENG1.TXT。

程序的过程是从键盘上读入一行后, 实现转换, 再把该行存入打开的 text 文件, 直至收到从键盘上输入的 Ctrl-Z 符号。代码如下:

```
program Exmp;
{ $B- }
var
  Filvar:text;
  Ch:char;
  Line: string [80];
begin
  Assign(Filvar, 'ENG1.TXT');
  rewrite(Filvar);
  writeln('Input English/Text please:');
  Line:= ' ';
```



```

read(Ch);
while Ch<> # 26 do
begin
  if Ch<> # 13 then
    Line:=Line+Ucase(Ch)
  else
begin
  readln;
  writeln(Filvar,Line);
  Line:= ' ';
end ;
  read(Ch);
end ;
close(Filvar);
end.

```

编译器指示 { \$ B- } 是为使输入符合标准 I/O 规则。Ucase(Ch) 是函数, 将 Ch 字符变成大写后回送。故 Line:=Line+Ucase(Ch) 将逐一读入的字符并在一起直到见回车符后成一行写入 text 文件中。

为结束文件行的输入, while 语句的结束条件为 Ch= # 26 (Ctrl-Z)。

11.4.4 无类型文件

TUBRO PASCAL 提供了直接访问磁盘文件低级通道的无类型文件。对有类型文件的随机文件、text 文件等如不考虑其存放在磁盘上字节序列的逻辑解释, 数据的物理存储实际上只不过是字节序列。这样它和内存的物理单元便可一一对应了。无类型文件用每 128 个连续的字节做为一个记录的组织方法, 它就可以和任何类型的内存变量(当把它看成内存中的若干字节时)交换数据了。

用无类型文件可以处理任何类型的已存在的文件。如 FileT.TXT 为 text 型文件, FilR.REC 为某类型的随机文件, 都可以把它们按无类型文件打开并操作。

无类型文件的类型标识符为 file, 对变量的说明如下:

```
var Datafile:file;
```

无类型文件的操作在 assign、reset、rewrite 几个过程的使用上与前述的几种文件类型相同。它采用了两个特殊的读/写过程: blockread 和 blockwrite。

在做这些操作之前, 也应使文件变量与某具体磁盘文件名相联系。

(1) 成块读过程, 形式为

```
blockread(Filvar, Fvar, Recs)
```

或

```
blockread(Filvar, Fvar, Recs, result);
```

其中, Filvar 为无类型文件变量标识符; Recs 是整型表达式, 表示要传送的记录数, 每记录为 128B。Fvar 是任何类型的变量, 但要注意 Fvar 在做 Read 操作时必须能容纳足够的字节数。



传送从 Fvar 变量所占用的第一个字节开始依次进行。

这里的一次读操作,实际读回的记录数在 Result 参数中回送。

读操作完成后,指针指向下一个要读的记录的首字节。

(2)成块写过程,形式为

blockwrite(Filvar,Fvar,Recs)或

或

blockwrite(Filvar,Fvar,Recs,Result);

其中,Filvar 为无类型文件变量;Fvar 是要被写入的变量;Recs 为整型表达式,表示要写入的记录数,每记录为 128B;Result 为整型的变量,返回实际写入的记录数(注:当实际写入的字节数大于 0 而小于 128B 时,Result 回送 1),指针的下拨也是以 128B 为单位的。写入的过程从 Fvar 变量所占空间的首字节开始。

(3) Eof、FilePos、Filesize 和 Seek 等在前面几种文件类型中使用的过程和函数在无类型文件中仍然可用,但要注意这里是以记录(128B)为单位的。如 Seek(Filvar,2)是将文件指针指向第三组 128B 的首字节。

下面通过例子说明无类型文件的使用。

【例 11.10】 将一个磁盘文件拷贝到另一个名字的磁盘文件上,这里不管文件类型如何。

因为题目只要求拷贝,故对源文件(即被拷贝文件)的类型不必关心,而将它看成是无类型文件。代码如下:

```
program FileCopy(input,output);
const
    RecSize=128;
    BuffSize=200;
var
    Source, Dest: file;
    SourceName, DestName: string [14];
    Buffer: array [1..RecSize,1..BuffSize] of byte;
    RecsRead: integer;
begin
    write('Copy from:');
    readln(SourceName);
    assign(Source,SourceName);
    reset(Source);
    write('To:');
    readln(DestName);
    assign(Dest, DestName);
    rewrite(Dest);
    repeat
        BlockRead(Source, Buffer, BuffSize, RecsRead);
        Blockwrite(Dest, Buffer, RecsRead);
```



```
until Recsread=0
    close(source);
    close(Dest)
end.
```

显然这个程序可对任何类型的文件进行操作,也可看出无类型文件在这类操作中带来的方便。

在上面的程序中我们注意到:BlockRead 过程中的参数 RecsRead 回送实际读到的记录数,而下面的 Blockwrite 过程将 RecsRead 作为输入参数,把实际读入 Buffer 缓冲区中的记录写入目标文件中,这样即使一开始源文件为空也没关系。

另外,请读者注意:Buffer 作为与无类型文件交换数据的缓冲区要留有足够存放所传送字节数的空间,即 Buffer 的容量:Recsize * Buffsize 大于或等于块读/写中的 $128 * \text{Recs}$ 。无类型文件传送受 PC DOS 操作系统的限制,每次传送的字节数最多不能超过 64KB,即读/写时的记录长与记录数的乘积要小于 64KB。

11.4.5 标准文件

当正文文件与计算机的键盘、显示器、打印机、串行接口等外部硬件设备相连接而进行输入/输出时,称为标准文件。

标准文件的使用方法不像磁盘文件那样规范,它有时使用专门的文件变量,如 Input、Output(专门用于键盘的输入和显示器的输出);有时使用专门的设备文件名,如 Con、LPT1 等;有时使用专门的连接过程,如 Assignrt。读者在学习时要注意其具体用法。

标准文件的具体输入/输出过程,不论其连接何种外设,仍使用 Read、Readln 和 Write、Writeln 过程来实现,其使用语法也与在磁盘文件中的用法相同。对于含 Input、Output 标准文件变量的情况,常常省略这两个文件变量不写,以方便程序的书写。

read(x)等效于 read(Input, x);
write(x)等效于 write(Output, x)。

在和计算机连接的设备中,有些设备的驱动程序已写在 DOS 操作系统, TURBO PASCAL 对这类设备的操作要借助 DOS 操作系统的功能,所以从 TURBO PASCAL 的角度称这类设备为 DOS 设备。TURBO PASCAL 中还支持一部分 DOS 系统不支持的或不完全支持的设备,称这类设备为文本文件设备。

1. DOS 设备

TURBO PASCAL 支持的 DOS 设备常见的有如下几种:

(1) CON 设备

CON 设备是指操作台控制设备。在文件设备中输出将送至显示器,输入从键盘上获得,除非特殊声明(如使用 Crt 单元),Input、Output 就定向在 CON 设备上,即在程序中不做任何声明(缺省状态下)的情况下,程序运行前自动执行下列操作:

```
Assign(Input, 'CON');
```



```
Reset(Input);  
Assign(Output,'CON');  
Rewrite(OutPut);
```

程序结束后,Input 和 Output 变量自动关闭,取消了和 DOS 支持的 CON 设备的连接。

(2) LPT1、LPT2、LPT3 设备

这是和 DOS 系统连接的打印机设备专用文件名。由于对行式打印机的输出不同于显示器,所以不能用标准文件变量 Input 和 Output,要用用户自己定义的正文文件类型的文件变量。LPT1、LPT2、LPT3 为打印机专用文件名,若系统只连接了一个打印机,通常是指 LPT1,也可以用 PRN 代替。下面的一段程序:

```
var  
    LT:Text;  
begin  
    Assign(LT,LPT1);  
    Rewrite(LT);  
    write(LT,'Hello');  
    close(LT);  
end.
```

是使用打印机输出的一个例子。程序运行后,将在行式打印机上输出 Hello 字样。

若用 Reset 标准过程试图打开 LPT 这样的设备,系统会立即遇到文件结束符。

(3) COM1、COM2 设备

为两个串行通信接口的专用文件名,可以用 AUX 代替 COM1 做专用文件名。该设备是可读/写的设备,通过串行接口进行数据的接收和发送。

2. 文本文件设备

TURBO PASCAL 中提供的 CRT 标准单元中的 CRT 设备就是一种文本文件设备。尽管它也有对键盘和屏幕的操作,但因它直接利用了 BIOS 或直接输出到显示缓存区,所以与 DOS 设备中的 CON 设备比较,具有更高的速度和更大灵活性,且 CRT 单元中还提供了屏幕的颜色控制和窗口功能等是十分有用的。

与 DOS 设备不同,文本文件没有保留设备文件专用名,通过预定义 Assign 过程建立这种连接,如 Crt 单元中的 AssignCrt。

只要在程序语句中使用 Crt 单元,Crt 单元中的初始化代码便使 Input 和 Output 标准文件变量指向 Crt,而不定向在 DOS 的标准 I/O 文件,即在程序开始前要执行下述定向语句:

```
AssignCrt(Input);  
Reset(Input);  
AssignCrt(Output);  
Rewrite(OutPut);
```

除非有新的定向命令,在程序结束前不会改变这种指向。



当 Crt 指向 Output 时,下列控制代码有特殊意义,即

- #7——响铃(从机内发出“嘟”的一声);
- #8——退格键(光标向左移一个字符位置);
- #10——换行(光标移到下一行);
- #13——回车(把光标移至窗口的左边)。

当从指向 Crt 的 Input 文件中读取文本时,一次读入的是一行字符,在一行组织完(见回车符 Enter)之前,所有已输入的字符存于缓存区中,这一期间可使用编辑键:

- BackSpae(或 Ctr+S)——删除上次输入的字符;
- ESC(或 Ctrl+A)——删除整个输入行;
- Enter——结束输入行,并在缓冲区中存储行结束符;
- Ctrl+D——重复上次输入行的一个字符。

这里 Ctrl+D 的功能要稍做解释:所谓上次输入行是指程序中上一个 readln 语句读入的一行,当又执行新的 readln 语句时组织当前编辑行。当按下 Ctrl+D 键时,将把上次输入行的第一个字符放在本次输入行光标所在位置,就像输入了一个字符一样;如再按下 Ctrl+D 键,将把上行的第二个字符放至本行光标所在位置,依次类推。例如:

- 上行曾输入:ABCDE ↵;
- 本行输入:1⊕2⊕3(⊕代表键入 Ctrl+D);
- 本行实际显示为:1A2B3。
- Ctrl+F 重复上次输入的一行;
- Ctrl+Z 终止输入;产生一文件结束符。

TURBO PASCAL 还提供了由用户自己定义特殊外设驱动程序以便直接利用 read、write 等 text 类型的标准过程的方法。有关单元的功能及使用方法,请参阅技术手册。

11.4.6 I/O 检查

在前几节叙述有关文件操作中,可能出现一些错误情况,如使用 reset(Filvar)过程,而 Filvar 所联系的磁盘文件名在实际磁盘上并不存在。这是很容易发生的,因为有可能用户把磁盘文件名输入错,或盘片插错。再比如 Seek(Filvar,n)中表达式 n 的值大于 Filvar 所代表的文件的长度等等。

程序运行中,如果有这样的错误发生会出现什么情况呢?一般情况下运行的程序会终止,程序控制权返回操作系统。这种终止程序有时是不方便的。为此 TURBO PASCAL 提供了一个 Ioresult 函数,在编译指示器的配合使用下,当出现错误的时候,程序能根据 Ioresult 的返回值决定去向,而不退回系统。

编译指示器 I 用于 I/O 出错时的处理方式,当采用{\$I+}时(缺省值)每当执行 I/O 操作时就调用一次 I/O 检查例行程序。程序若发现此次 I/O 操作有错,则终止程序运行并在显示错误类型信息后返回操作系统。

当选用{\$I-}时,取消 I/O 检查。这时不执行运行中的检查例程,I/O 出错时也不终止程序,但将挂起进一步的 I/O 操作,直至程序中出现对 Ioresult 函数的调用。Ioresult 的调用一方面能返送 I/O 错误信息(只有等于零时为正常,其余值为错误代码);另一方面清除错误



情况对 I/O 下一步继续操作的挂起状态。

【例 11.11】 打开一个存在的文件,如有名字的错误则重新让用户输入名字,直至正确为止。代码如下:

```
repeat
    write('Enter name of Input file:');
    readln(InFileName);
    assign(InFile, Infile Name);
    { $ I- }
    reset(InFile);
    Ok:=Ioresult; { $ I+ };
    if Ok<>>0 then
        writeln('Can Not find file', Infile Name)
    until Ok=0;
```

从程序段中看出由于使用了 Reset 过程,若输入的文件名有误,则 IOresult<>>0,因而程序将重新返回输入文件名。

DP 11.5 指针类型与动态变量

前面介绍了 PASCAL 语言中的简单类型和构造类型。当在程序中要使用变量时,必须在程序的变量说明中对它进行说明。运行此程序时,首先在内存的数据段中为这些全局变量分配内存单元,然后再执行主程序体的语句部分;程序运行结束后,变量自动消失,并释放出所占用的内存单元。这些变量有着共同的特点,即必须在相应程序体的 var 说明中说明。var 说明确定了变量的类型和标识符,变量的类型也决定了为变量分配内存时所占用的内存的大小;这些变量在程序体运行前已分配,运行时不再改变,运行结束后自动消失,并释放所占内存单元。我们称具有这些特点的变量为静态变量。

那么能不能根据需要在程序运行期间生成变量呢?这是 PASCAL 语言允许的,但必须通过特殊标准过程向系统提出申请。我们称这种动态生成的变量为动态变量。动态变量不需要在程序的说明部分中说明,它可在程序执行部分的任一点生成,并且不局限于过程或函数,一旦生成就不会自动消失。动态变量的撤消也是通过相应标准过程实现的。由于动态变量不出现在 var 说明中,因此它们是无名的,不能直接用标识符访问这种变量,而需要通过指针间接访问。这也是静态变量与动态变量的又一区别。

动态变量的生成、访问与撤消都需要借助于指针类型的变量。这一节将介绍指针类型及动态变量的生成、访问和撤消。

11.5.1 指针类型

由于动态变量并未出现在 var 说明中,因而无法直接用标识符访问它们。在 PASCAL 中,通过指针类型的变量访问动态变量。假设有如下定义:



```

type
    RealPtr = ^real;
    NameRec = record
        Name: string[20];
        Age: 0..100;
    end;
    NamePtr = ^NameRec;
var
    p1, p2: RealPtr;
    q1, q2: NamePtr;
    r: ^char;

```

RealPtr 和 NamePtr 是指针类型。其中 RealPtr 的基类型是实数类型, NamePtr 的基类型是记录型 NameRec。从这个例子中可以看出指针类型的语法如图 11.10 所示。

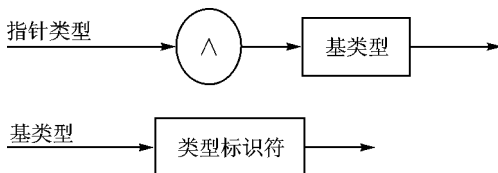


图 11.10 指针类型的语法图

变量 p1、p2、q1、q2、r 全是指针类型变量。其中 p1、p2 用来指向实数类型的动态变量, q1、q2 用来指向记录型(NameRec)的动态变量, r 用来指向字符类型的动态变量。

那么, 指针类型变量指向某某类型的动态变量代表什么意义呢?

变量在计算机内存中占据一定数量的存储单元, 这些单元的起始地址就称作该变量的指针。因而, 指针类型变量是用来指向动态变量的这句话意味着指针类型变量中存放的是该动态变量在内存中的首地址。例如, 如果变量 p1 已经指向了一个实数类型的动态变量, q1 已经指向了一个 NameRec 类型的动态变量。指针类型变量和动态变量之间的联系可用图 11.11 表示。

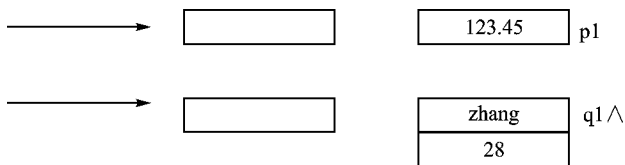


图 11.11 指针变量和动态变量的关系图

在前面的类型定义中, NameRec 和 NamePtr 的定义位置能否交换一下呢? 回答是可以的。这是 PASCAL 语言中允许违反“先定义后引用”原则的唯一的特例。

不难看出, 指针类型的变量也必须在程序的说明部分里加以说明, 因为指针型变量是静态变量, 并且每个指针类型变量一经说明便只限于指向某种固定类型的动态变量。

在 PASCAL 中, 指针可按下列方式使用:



- 被赋值；
- 作为参数；
- 作等与不等的关系运算；
- 用来访问动态变量。

11.5.2 new 标准过程与指针变量

前面的类型及变量定义仅说明了 `p1`、`q1` 等变量是指针类型变量,它们可以指向相应的动态变量。但由于没有为动态变量申请内存空间,也未给指针变量赋予指针(动态变量的首地址)值,这时的指针变量还没有指向哪个具体的内存区域,因而这些指针变量还不能用于访问动态变量。

PASCAL 语言中的标准过程 `new` 可以实现为动态变量分配存储空间和让指针变量指向该空间首地址的操作。标准过程 `new` 的使用方法如下:

`new(指针变量)`

例如,可以通过语句:

```
new(p1);
```

使系统在内存中申请一个 `real` 类型的存储区,并把该存储区的首地址放入指针变量 `p1` 中。这时,`p1` 就指向了一个具体的 `real` 类型的动态变量。假如,设法让该动态变量的值为 123.45,那么指针变量 `p1` 与该动态变量的关系就如图 11.11 所示。同理,可以通过语句:

```
new(q1);
```

让指针变量 `q1` 指向一个具体的 `NameRec` 类型的动态变量。用语句:

```
new(r);
```

让指针变量 `r` 指向一个具体的 `char` 类型的动态变量。

应当指出的是,`new` 标准过程既为动态变量分配了内存空间,又为指针变量赋了值(所分配的动态变量的首地址);但它并没有对动态变量赋值,即指针变量所指向的动态变量中尚无具体的内容(可以认为其值是不确定的)。如果想使用动态变量,涉及对动态变量的访问等问题。

11.5.3 动态变量的访问

在 11.5.1 节中定义的指针变量 `p1`、`q1`、`r` 等,在 11.5.2 节中为它们进行了赋值(用 `new` 过程实现)之后,可通过以下方法访问动态变量,即

```
p1^:=3.14159;
```

```
r^:=‘B’;
```

```
q1^.Name:=‘Zhao’;
```

```
q1^.Age:=30;
```

执行以上语句之后,指针变量和相应的动态变量如图 11.12 所示。

对动态变量除了可以赋值操作之外,它还可以做与其类型相适应的其他操作,例如:

```
readln(p1^);
```

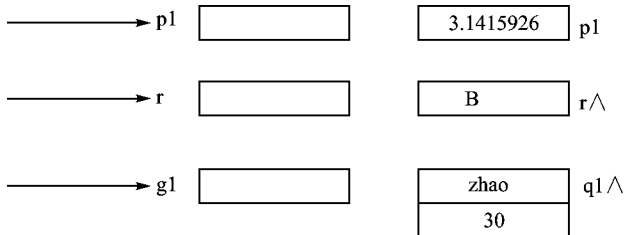


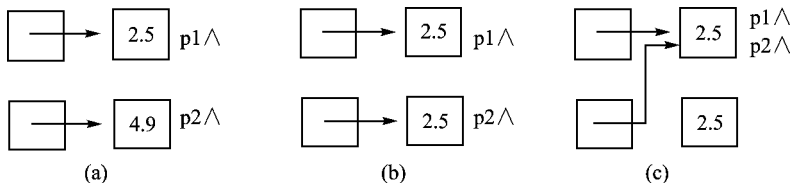
图 11.12 动态变量的赋值

```
writeln(p1^);
if (p1^ > 1.825) then
p1^ := p1^ + 72.50;
```

指针变量除了可通过 `new` 过程赋值之外,两个类型一致的指针变量之间可以相互赋值。在 11.5.1 节中定义的 `p1` 和 `p2` 都是 `RealPtr` 类型(指向 `real` 类型)的变量,它们之间可以互相赋值。假设有以下语句序列:

```
new(p1);
p1^ := 2.5;
new(p2);
p2^ := 4.9;
```

这时数据存储情况如图 11.13(a)所示。

图 11.13 `p2 := p1` 与 `p2^ := p1^` 的区别

如再执行 `p2^ := p1^`, 情况变为图 11.13(b)所示。若在情况图(a)下执行 `p2 := p1`, 则情况如图 11.13(c)所示, `p2` 的值得到了修改,使它也指向 `p1` 指向的动态变量,并与原动态变量脱离了联系,以后将无法通过 `p2` 访问原动态变量。但该动态变量并不消失,仍将占用内存,系统将无法收回它占用的内存重新利用,所以程序设计中应尽量避免这种情况发生。程序中可用另一类型一致的指针变量先把 `p2` 的值记录下来,不妨设为 `p3`,即先做 `p3 := p2`;然后再执行 `p2 := p1`,这样值为 4.9 的动态变量就不会丢失了。

两个类型不一致的指针变量之间不允许互相赋值,如 `q1 := p1`;是不合法的语句。但在 PASCAL 中有一个预先定义的指针类型常量 `nil`,它不指向任何动态变量,通常称作空指针。作为一个特例,`nil` 与所有类型的指针变量赋值相容,可以作为任何指针变量的值。例如:

```
p1 := nil;
q1 := nil;
```

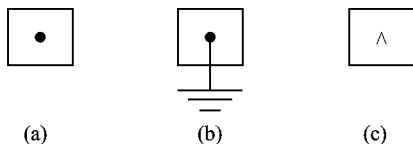


图 11.14 空指针

空指针通常用来初始化一个指针。空指针可用“·”指示,如图 11.14(a)所示;也可用地线符号表示,如图 11.14(b)所示;而在数据结构的理论讨论中往往用符号 $\wedge$ 表示空指针,如图 11.14(c)所示。当访问一个动态变量 $p \wedge$ 时,若 p 值未定义或为 nil 时,访问结果是不确定的。

11.5.4 dispose 标准过程

由于系统可供 `new` 过程使用的空闲区是有限
的,不断地使用 `new` 过程创建动态变量就会造成该空闲区全被占满,从而导致程序运行错误。
标准过程 `dispose` 可以完成撤消动态变量的任务,其使用方法为

`dispose(指针变量);`

因而,可以调用 `dispose` 标准过程释放那些不再使用的动态变量的内存空间,以便于系统再利用。
如语句:

`dispose(p1);`

将释放指针 $p1$ 所指向的动态变量所占的内存区域。该语句被执行之后, $p1$ 所指向的内存空间已被系统收回,再做 $p1 \wedge := 4.5$;这样的操作是非法的。一般情况下,应在语句 `dispose(p1)` 之后跟随操作 $p1 := \text{nil}$;表示 $p1$ 不再指向任何动态变量,以避免那些非法操作。



习题与思考题

- 11.1 利用一维数组计算一个班级 30 名同学某门课的平均分、最高分及最低分。
- 11.2 设有 5 阶 2 维方阵,计算所有对角线上的元素之和。
- 11.3 编程从一个字符串中移去所有的空格,并紧缩后输出。
- 11.4 输入一系列字符,统计其中元音字母的个数。
- 11.5 输入一个班级同学的学号、姓名及某门课成绩,按成绩由高到低排序后,输出该班的成绩单。
- 11.6 将 11.5 题中的学生成绩单存入文件 "student.dat" 中。
- 11.7 建立一个单向链表,存放某个班级所有同学的姓名及成绩。



第 12 章

面向对象程序设计基础

基础篇完整地介绍了 PASCAL 程序设计的一般方法。归纳一下程序设计的过程大致分为两个步骤：第一步是先把要解决问题的特征用一组数据结构来表示，然后把解决问题的办法用若干组过程（或函数）实现。数据和过程总体构成了程序。由于这些过程（或函数）要对这组数据进行操作，所以在编写过程（或函数）的过程中有些还要对数据结构进行修改。这种设计方法的核心是功能的分解。它将数据结构和过程（或函数）作为两个实体分别对待，而且在设计过程中着重于解决问题的操作过程，所以称它为面向过程的程序设计方法。这种方法把数据结构和过程分解成两个实体。这两个实体的关联性给设计过程，特别是程序的维护带来了很大的麻烦。比如，当某一个局部过程需要修改数据结构（这在程序设计过程中是会经常发生的），作用到这组数据基础上的其他过程就都要做相应的修改；反之，当数据结构发生变化也要引起过程和函数的修改。尽管 PASCAL 用局部变量及模块化结构作为有益的补充，但面向过程设计方法的数据与过程相分离很难解决现实世界中特性与行为相依存的问题。

面向对象的程序设计 OOP(Object-Oriented Programming)是一种新的设计思路，或者说提出了一种新的程序设计范型。这种设计方法的成功在于它抓住了模拟现实世界事物特征和行为的统一性原则，把数据和作用在其上的操作构成一个整体——对象。面向对象去研究程序的组织，从而使其比以前的结构化程序更加结构化，也比以前的数据抽象和信息隐蔽更为抽象化和模块化。

支持面向对象程序设计范型的高级语言开始于 20 世纪 60 年代的 simula 67, TURBO PASCAL 5.5 版本开始具有面向对象的程序设计功能，特别是 Delphi 3.0 更是基于面向对象程序设计这一基础，又兼容 Windows 窗口界面要求，向用户提供了一个良好的 OOP 模块。

本章将结合 PASCAL 中提供的 OOP 功能介绍，引出对象的概念，介绍面向对象程序设计主要特性：封装性、继承性、多态性，通过对这些特性的学习，将会掌握 OOP 的设计思路，体会到面向对象程序设计代码的可再用性及神奇的扩展功能。



12.1 对象类型的定义

对象是 OOP 设计的一个基本概念，它来源于现实世界，抽取了日常所说的对象的基本特性。日常所说对象是构成世界的基本单元，可以是一件物品，可以是一种动物，还可以是一件事，例如一张桌子、一只猫、一次储蓄。随考察问题的角度不同，对象可以简单，也可以复杂；复杂的对象可由若干简单对象构成。现实世界中的对象都是可描述的，同时又以能相互区别而存在。在程序设计中把具有下列三个特征的描述整体称为对象。



- (1) 有一个名字,用以标识并区别于其他对象。
- (2) 用一组数据来描述它的属性。
- (3) 用一组操作来表示它的行为。

比如,储蓄这件事,可以进行如下描述:

储户姓名;

账号;

现有金额;

存入操作;

取出操作。

例中前三项是状态的描述,后两项是作用这组状态之上的操作,用程序的术语,它构成了数据单元和过程单元的组合体,是一类具体的对象。

一组具有相同属性构成和操作行为的对象构成了类。类是创建对象的样板,包含对象属性的描述和操作的说明。有了类才可以创建对象,对象是类的一个实例(instance)。这种类与对象的关系很像我们过去熟悉的类型与变量的关系,只不过在那里只定义了数据结构,而没有在类型说明中定义操作。

下面看 TURBO PASCAL 中支持 OOP 的一个新的数据类型——对象类型的定义。

先来看一个新的类型定义的例子。它被用来描述屏幕上的位置(Row,Column)及有关位置的初始化、取得当前位置的坐标值等操作。其代码如下:

```
type
  Location=object
    Row,Column:integer;
    procedure Init(newrow,newcol:integer);
    function GetRow:integer;
    function GetCol:integer;
  end;
```

该例定义的是 TURBO PASCAL 5.5 版提供的新数据类型——对象类型。Location 为该类型标识符。看上去它像记录类型,但这里保留字 object 代替了 record。类型定义中 Row、Column 的定义与记录类型的域相似,称为对象的数据域。和记录类型定义不同的是,这里定义了三个过程和函数的首部,称为方法(method)声明。从例中看出:数据的声明在方法声明之前,这是 TURBO PASCAL 对象类型定义的原则之一。

对象类型中声明的每一个方法都要在该类型定义的后面(主程序前的外层上的任何位置)定义对应的程序体,以实现方法的具体功能,称其为方法体。Location 类型中定义的三个方法的方法体定义如下:

```
procedure Location. Init(NewRow,NewCol:integer);
begin
  Row:=NewRow;
  Column:=NewCol;
```



```
end;  
function Location. GetRow; integer;  
begin  
    GetRow := Row;  
end;  
function Location. GetCol; integer;  
begin  
    GetCol := Column;  
end;
```

可见,方法体的定义和过程、函数说明的形式完全一样,只是要求方法体的定义的首部要与该方法在对象类型中声明的形式一样,且在方法体的过程或函数名标识前,要冠以对象类型的类型标识符及圆点加以限定。如 `Location. Init` 是指 `Location` 对象类型中的 `Init` 方法。对方法体首部采用限定标识符的原因,是 TURBO PASCAL 规定对象类型(当然包括方法域中方法的声明)一定要在程序或单元的外层位置,即对象类型的定义不能在过程、函数或方法体的说明段中去说明;而方法的实现部分定义常常在程序说明部分的后面或单元的实现部分。冠以限定符不仅能为编译程序连接提供指示,也增加了程序的可读性。

对象类型定义的语法图如图 12.1 所示,图中的部分语法,如继承、`private`、`virtual` 等将在后面几节逐一介绍。

定义了对象类型之后,就可以在变量声明段中声明某变量属于对象类型。我们称该变量为此对象类型的一个实例(Instance)。例如:

```
var  
    Alocation: Location;
```

`Alocation` 为 `Location` 对象类型的一个实例。

在对象类型的声明中,方法的形参表中可以列入正在说明的对象类型,尽管类型的说明还没完成。例如:

```
Rect = object  
    a, b: integer;  
    procedure Copy(var R: Rect);  
    procedure Move(c, d: integer);  
end;
```

方法 `Copy` 的形参 `R` 的类型为 `Rect`,而 `Rect` 正是其所在对象类型的标识。

对象类型作为一个类型,可以去定义其他结构类型。如可以用对象类型作为一个记录类型的域类型、数组类型的元素类型,但它不能作为文件类型的元素类型。

前面只在形式上说明了对象类型的定义方法,后几节将结合实例对对象类型的功能和使用方法进行解释。

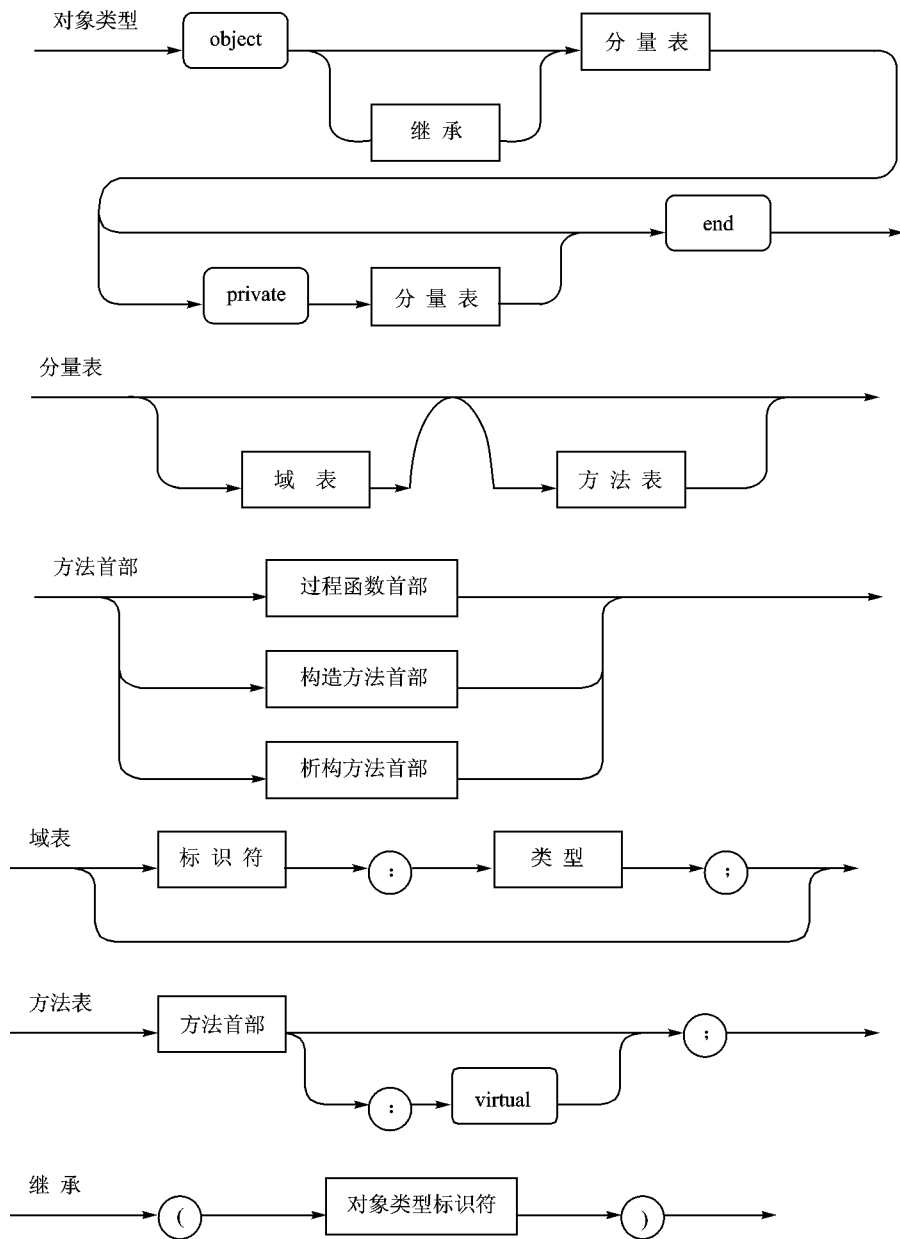


图 12.1 对象类型语法图

对象类型不仅形式上把数据域与代码(指方法)封装在一起,而且方法体与其所属的对象类型的数据域共享同一作用范围。这一特点称为对象类型的封装性(encapsulation)。前节 Location.Init 方法体内使用了语句“Row:=NewRow;”,Row 是在 Location 类型的数据域中



定义的,在 location 所属的方法体 Init 中被直接引用,说明它们之间有共同的作用域。做了这样的规定之后,带来的必然结果是,对象类型的数据域不能与本对象类型所属的方法的形式参数、方法体中的局部变量重名。

有了对象类型的封装性,对象类型的数据和操作该数据的方法成了一个整体——对象(object)。摆在我们面前的不再是被动的数据集和独立的程序,而是一个具有特性(数据)和行为(方法)的事物模拟整体。可以通过向对象类型的实例发送“消息”来让对象实例执行一次“模拟”动作。所谓发送消息就是对对象实例的某一个方法进行一次调用。由于对象的方法本质上仍是一个过程,所以方法的调用和过程的调用是一样的,只是在过程或函数名前要冠以对象实例的名字加以限定。例如:

```
Alocation. Init(20,20);
```

就是让对象实例 Alocation 初始化至坐标(20,20)位置上的一个方法调用。

只要在对象类型中提供足够的方法,完全可以不让对象外的过程去直接操作对象实例的数据域,为结构化程序设计提供了数据更加隐蔽、功能模块更加明确的手段。程序设计人员不必担心方法的增多会影响将来程序代码的冗余,事实上灵活的 TURBO PASCAL 编译程序只把被调用的方法代码加入到目标码中。

这种封装性是 OOP 程序设计提倡的一种规范,但 TURBO PACASL 并没有像其他语言那样把这一规范强加于程序设计者。它允许用与记录类型变量访问数据域类似的方法访问对象类型的数据域。记录类型的开域语句或点域法在对象类型的实例中也适用。如欲对 Alocation 实例的 Row 域赋值 25,可以用下面的语句:

```
Alocation. Row:=25;
```

或

```
with Alocation do
```

```
Row:=25;
```

实现。我们已经知道,在对象方法体中访问对象的数据域时,不必做开域或点域的声明。这是由于在调用方法时,隐含完成了 with myself do method 的操作,把对象的数据和方法链接到了一起。每次对方法调用时都传递了隐含参数 Self,从而实现了 with 语句的效果。Self 参数是一个指向被调用方法的对象实例的 32 位指针。

在特殊情况下,参数必须显式地给出:

```
type
  MousePos=record
    Row,Column:integer;
  end;

procedure Location. GotoMouse(mouse;MousePos);
begin
  with Mouse do
  begin
    self. Row:=Row;
```



```
self.Column:=Column;  
end;  
end;
```

处在 Mouse 记录类型的开域语句,冠以 Self 限定符可以区别 Location. Row 和 Mouse. Row。

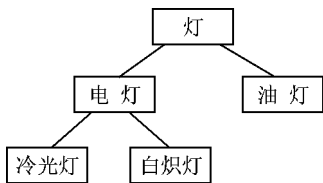
按图 12.1 所示的语法,在对象类型中保留字 private 将对象类型中定义的数据和方法分成两部分,如下面的对象类型定义形式:

```
type  
  NewObject=object  
    FieldsCom;           {公共数据部分}  
    MethodsCom;          {公共方法部分}  
  private  
    FieldsPri;           {私有数据部分}  
    MethodsPri;          {私有方法部分}  
  end;
```

处在保留字 private 之后定义的数据和方法为对象类型的私有部分。对象类型私有部分分量表的定义格式与公有部分分量表的定义格式相同。对象类型的私有部分只有在本对象声明的范围内可以访问,这样避免了本对象类型处的其他部分对私有部分进行引用,给用户提供了隐蔽的手段。

DP 12.3 对象类型的继承性

我们把特征结构相同且具有相同行为的对象归为同类。同类中的对象以其每项具体特征的差异表现个体的存在。当在某类的特征结构中增加特征或新的行为时,就会出现新的对象组,因而产生新类。比如,把具有能发光、为人类活动场所提供照明两特征的物品分类可以产生灯类;如果再考察发光能源一项特征,就可能出现电灯及油灯等新类;如果在电灯类中再增加发光方式特征,又可能产生冷光灯和白炽灯新类,如图 12.2 所示。



增加特征所产生的新类显然和原类之间有密切的关系,它包括了前类的全部特性,因此称其为前类的子类;而把前类称为它的父类。如电灯是灯的子类,而电灯是冷光灯和白炽灯的父类。把子类包含父类全部特征的性质称为子类的继承性。

图 12.2 分类

TURBO PASCAL 提供了对继承特性的支持。

前节定义的 Location 类型说明了点在屏幕上的位置情况。如果还要描述该位置上点的可见性,采用记录的惯用方法将其定义为

```
Rpoint=record
```



```
Pos:Location;
Visible:Boolean;
end;
```

这里显然产生了一个后继的新类,它除有描述位置功能之外,还用该位置的可见性描述,总体代表一个点。为简化这种书写, TURBO PASCAL 给出了专门后继的定义方式,并规定了相应产生的一些必要功能。首先给出如下形式的定义:

```
Point=object(Location)
    Visible:Boolean;
end;
```

保留字 `object` 之后括号内的类型 `Location` 是新定义的 `Point` 类型的直接前辈。这样定义后,对象类型 `Point` 既从对象类型 `Location` 中继承了 `Row` 和 `Column` 数据域,又扩充了自己定义的数据域 `Visible`,用以说明该点在位置(`Row`、`Column`)上是否可见。这里强调对象继承的数据域和自己扩充的数据域是同层的,和 `Rpoint` 记录类型相同,它相当于下面的记录类型:

```
Opoint=record
    Row,Column:Integer;
    Visible:Boolean;
end;
```

如果还要把 `Point` 扩展成圆,就把 `Point` 的位置定为圆心,把 `Visible` 视为对圆的可见性,这时利用对象的继承性,扩展为

```
Circle=object(point)
    Radius:Integer;
end;
```

其中, `Radius` 为新增数据域,用以描述圆的半径。

称 `Point` 和 `Circle` 类型为 `Location` 类型的后辈类型,而称 `Point` 类型为 `Location` 类型的直接后辈类型。称 `Location` 和 `Point` 为 `Circle` 类型的先辈类型,而称 `Point` 为 `Circle` 的直接先辈类型。后辈类型不仅可以与其先辈类型共享数据域,还可以享用其先辈类型的方法,当然也可以定义新的方法来扩充功能。后辈类型享用先辈类型中定义的数据域、方法的过程称为对象类型的继承(inheritance)。

对象类型对数据域的继承性,表现在方法不仅和其所在类型定义的数据域有共同的作用范围,还与其一切先辈类型(包括直接先辈类型和非直接先辈类型)中定义的数据域有共同的作用范围,就好似所有先辈类型中定义的数据域是在本对象类型中定义的一样。因此,方法的形参、方法内的数据与后辈类型中扩充的数据及其一切先辈类型域相互间不能重名。看下面的例子:

```
Point=object(Location)
    Visible:Boolean;
    procedure Init(InitRow,InitCol:Integer);
    procedure Show;
```



```
procedure Hide;  
    procedure MoveTo(NewRow,NewCol:integer);  
end;  
end;
```

Point 的 Show、hide 方法的方法体定义为

```
procedure Point. Show;  
begin  
    Visible:=true;           {置该点为可见}  
    PutPixel(Row,Column,GetColor); {取当前颜色在 Row、Coiumn 处画点}  
end;
```

该方法是在屏幕上显示点。其中 GetColor 是 TURBO PASCAL 提供的标准函数,取前次画像素的颜色;PutPixel 为标准过程,在坐标 Row 和 Column 处按给定颜色画点。代码如下:

```
procedure Point. hide;  
begin  
    Visible:=false;  
    PutPixel(Row,Column,GetBKColor);  
end;
```

该方法是在屏幕上消掉点。其中 GetBKcolor 为标准函数,取屏幕的背景颜色值。用背景色画点,实质上是消去该点。

从这两个方法体中的语句看,它们不仅使用了自己类型中定义的数据域 Visible,而且直接使用了先辈类型 Location 中的数据域 Row 和 Column,却没有使用任何 With 语句或点域法去声明。可见对象类型对先辈数据域的继承为方法的设计带来了很大的方便。

对象类型的继承性还表现在对先辈类型中声明的方法的继承上。

前面已经介绍过对象类型实例调用对象类型中的方法的格式为

〈实例标识符〉.〈方法名及实参表〉

还有一种调用对象类型方法的途径,就是从方法体中调用本类型中或先辈类型中定义的方法。这时可能本对象类型或先辈类型还没有定义实例,其调用格式为

〈对象类型标识符〉.〈方法名及参数表〉或〈方法名及参数表〉

数据域的继承原则不允许在本对象类型中扩充与其所有先辈类型重名的数据域,而方法的继承性并无此限制。就是说本对象类型中定义的方法允许与其先辈类型中的方法重名。出现重名时,后辈类型中的方法就覆盖了前辈类型的方法。如 Point 类型中也定义了 Init 方法,这就与其父辈类型 Location 中的 Init 方法重名。其方法体的定义为

```
procedure Point. Init(InitRow,InitCol:integer);  
begin  
    Location. Init(InitRow,InitCol);  
    Visible:=false;
```



```
end;
```

由于 Point 中定义的 Init 方法已覆盖了先辈类型 Location 中的 Init, 所以 Point. Init 方法体中再调用先辈类型中的重名方法时, 就必须冠以先辈类型名标识符来限定以区别, 代码如下:

```
Location. Init(InitRow, InitCol);
```

几乎每一对象类型都有初始化方法。定义类型的初始化方法时, 直接采用继承先辈类型的初始化方法, 而只对本对象类型的特殊问题分别初始化是个好习惯。它有利于前辈类型的修改, 不影响其后辈类。

再来看 Point 中的 MoveTo 方法体:

```
procedure Point. MoveTo(NewRow, NewCol: Integer);
begin
    hide;
    Row := NewRow;
    Column := NewCol;
    Show;
end;
```

这个方法的用途是把当前的点隐蔽掉, 点的坐标移到新位置, 然后再在新的位置上显示。这一过程相当于移动一个点的位置。

方法体中引用了 hide 和 show 方法。和 Init 的引用不同, 没有对 hide 和 show 所属的类型用限定符说明。这两个方法是不是指本类型中的定义的方法? Point 的先辈类型 Location 中定义的 GetRow 和 GetCol 方法可否在 Point 类型的方法定义中引用或从 Point 的实例中引用? 我们将在下节详细回答这些问题。

DP 12.4 静态方法

现在再扩展一个 Point 的子类型 Circle。它以 Point 中的点做为圆心, Point 中点的可见性为圆的可见性, 又增加了半径 Radius 来描述一个圆的特性。其代码如下:

```
Circle = object(Point)
    radius: integer;
    procedure Init(InitRow, InitCol, InitRadius: Integer);
    procedure Show;
    procedure Hide;
    procedure MoveTo(NewRow, NewCol: Integer);
end;
```

对应的方法声明体为

```
procedure Circle. Init(InitRow, InitCol, InitRadius: Integer);
begin
```



```
Point. Init(InitRow, InitCol);  
Radius := InitRadius;  
end;  
procedure Show;  
begin  
Visible := true;  
Graph. Circle(Row, Column, Radius);  
end;  
procedure Hide;  
var  
TempColor; words;  
begin  
TempColor := Graph. GetColor;           { 暂存当前颜色值 }  
Graph. SetColor(GetBKColor);             { 当前色置成背景色 }  
Visible := false;  
Graph. Circle(Row, Column, Radius);       { 画圆, 使用当前颜色 }  
Graph. SetColor(TempColor);               { 恢复当前颜色 }  
end;
```

其中,用 Graph. 冠名的函数或过程是 TURBO PASCAL 在 GRAPH. TPU 单元中的标准函数或过程。GetColor 为无参函数,返回当前画笔的颜色值;GetBKColor 取回当前屏幕背景颜色值;SetColor 为按参数值设置当前画笔颜色;Circle 则为一作图过程,以其参数表中(Row, Column)位置为中心,以 Radius 为半径,以当前的颜色画一个圆,代码如下:

```
procedure Circle. MoveTo(NewRow, NewCol; Integer);  
begin  
Hide;  
Row := NewRow;  
Column := NewCol;  
Show;  
end;
```

把 Circle 类型中定义的方法与其父类 Point 类型中的方法做比较,发现有方法定义上的重名。前面说过子类与父类有相同的数据作用域,因而不允许重名出现;但子类的方法定义却允许重名。

Circle 类型中定义的方法 Circle. Init 引用了 Point. Init 去初始化圆心的位置,对自己的新增数据域 Radius 又进行了置初值操作。这样 Circle. Init 方法的形参表比 Point. Init 的形参表多了一项,这种覆盖是容易理解的。Circle. show 和 Circle. hide 方法要覆盖 Point 中的这两个同名方法是因为它们要显示或消隐一个圆而不是一个点,方法的操作显然不同。读者对比阅读一下方法的实现部分就可以清楚地看出。

问题在于 MoveTo 方法。Circle. MoveTo 与 Point. MoveTo 的形参表、方法、程序体完全相同,Circle. MoveTo 没有对 Circle 类型的特有数据或方法进行任何操作或引用,还有什么必



要重写一遍去覆盖先辈类型中的方法呢？这要从 TURBO PASCAL 编译处理方法调用的原则谈起。

当从对象类型实例调用方法或从对象类型的方法体中引用方法(设这种引用是带有对象类型限定符的引用)时,编译首先从对象实例所属类型或限定符所指类型的方法域开始查找。若有同名方法,则认为是所指的方法并调用执行;若该类型方法域中无同名方法,则到其直接先辈类型中去作同样的查找,依次向前辈类型中类推。若到了顶点的对象类型(即该类型无先辈类型)仍未发现所找的同名方法,则输出错误信息:未定义此方法。如果从方法体中引用方法没有指明方法所属类型(即无对象类型限定符),那么编译就从引用方法的语句所在方法体应属的类型的的方法域开始向前辈类型做前述方法相同的查找。

假设定义了如下三个对象类型实例:

```
Alocation;Location;  
APoint;Point;  
ACircle;Circle;
```

按上述方法调用原则,Acircle. GetRow 方法调用应该从 Circle 类型开始顺次向先辈类型查找首先遇到的同名方法,结果真正调用的是 Location. GetRow 方法。Acircle. Show 方法调用的应该是 Circle. Show 方法,显示一个圆。Apoint. show 方法调用的是 Point. show 方法,显示一个点。来自 Point. MoveTo 方法体中的引用 hide 应是指 Point. hide 方法,而来自 Circle. MoveTo 方法体中的引用 hide 则是调用 Circle. hide 方法。简言之,当对象类型树中有重名方法定义时,编译总是引用本对象类型的同名方法(如有定义)或其最近的先辈类型中定义的同名方法。

回到前面的覆盖 MoveTo 问题,如果在 Circle 类型中不拷贝 Point 中的 MoveTo 方法,当发生 Acircle. MoveTo 的调用时,将找 Point. MoveTo 去执行,而这时 MoveTo 中的 hide、show 调用显然是指 Point. show 和 Point. hide,结果是移动一个点而不是圆。即使 Circle 中定义了 Show 和 Hide 方法覆盖了 Point 的这两个方法,也无济于事。因为编译不会向后辈类型中去查找方法。如果 Circle 类型中定义了 MoveTo 方法情况就不同了,尽管是照原样从 Point 类型中拷贝过来。当执行 Show 和 Hide 方法引用时,到 Circle 的方法域中去查找,指的是 Circle. Show 和 Circle. Hide,起到了消隐圆,移动圆心,再显示圆的移动作用。

至此,定义的方法都是静态方法。编译程序对静态方法的处理,是在程序运行前就从方法调用开始一直连接到要执行的过程或方法体,把它们的代码装入目标码文件中,因而也称前期装配。

编译程序的前期装配,使那些程序中未被调用的方法代码不装入连接,减少了冗余。因此,在源程序中设计足够多的方法并不见得增加目标代码的体积,却为程序的扩展带来了方便。前期装配使程序有很高的运行速度也是它的一个值得重视的优点。

继承性在一定程度上改变了兼容性的规律,它表现在下述三个方面。

(1) 在对象类型的实例间,后辈类型实例可向其先辈类型实例赋值。例如:

```
Alocation:=ACircle;
```



合法,而

```
ACircle:=APoint;
```

不合法。

(2) 在对象类型的指针间,指向后辈类型的指针可向其先辈类型的指针赋值。例如:

```
var
  PPoint:^Point;
  PCircle:^Circle;
```

定义后可以有

```
PPoint:=PCircle;
```

(3) 在对象类型的形参和实参间,实参为形参的后辈类型时,可以代入形参来调用方法。这一功能大大扩展了类型一致的限制。

总之,对象类型的兼容性是后辈到前辈的。

对象类型的继承性定义可以体现在单元的不同部分之间,如在单元的接口部分定义一种对象类型,而在同一单元的实现部分定义其子类型;还可以体现在两个单元间,如 B 单元调用 A 单元,那么可以在 B 单元中定义 A 单元中定义的对象类型的子类型。

DP 12.5 虚方法和多态性

先介绍虚方法的形式定义,然后介绍它的用法及其支持多态性的原理。

在对象类型的方法域中方法声明的后面,加上保留字 `virtual` 后,该方法就被声明为虚方法。

以前几节的对象类型为基础,定义一个含有虚方法的对象类型树,代码如下:

```
Location=object
  Row,Colnm:Integer;
  procedure Init(InitRow,InitCol:Integer);
  function GetRow:integer;
  function GetCol:integer;
end;
Point=object(location)
  Visible:Boolean;
  Constructor Init(InitRow,InitCol:integer;
    InitVisible:Boolean);
  Destructor Done;Virtual;
  procedure Show;Virtual;
  procedure Hide;Virtual;
  procedure MoveTo(NewRow,NewCol:Integer);Virtual;
```



```
end;  
Circle=object(Point);  
    Radius;Integer;  
    Constructor Init(InitRow,InitCol;Integer;  
                    InitVisible;Boolean;  
                    InitRadius;Integer);  
    procedure Show;Virtual;  
    procedure Hide;Virtual;  
    procedure Fill;Virtual;  
end;
```

其中,在 Point 对象类型中的 Done、Show、Hide、MoveTo 方法声明后有保留字 Virtual,故为虚方法。同样 Circle 对象类型中的 Show、Hide、Fill 也为虚方法。

定义中保留字 Constructor 声明了一种特殊的方法称为构造方法。含有虚方法的对象类型必须定义构造方法,且构造方法不能定义为虚方法。

保留字 destructor 定义了又一种特殊方法,称析构方法。它在堆上分配的动态对象类型实例采用 Dispose 过程释放空间时使用。析构过程一般以声明为虚方法为好。这两种方法的深一步含义及使用,后面将进一步说明。

声明为虚方法的方法体的定义与静态方法是一样的。虚方法的调用语法与静态方法也是一样的。

有关虚方法要注意下面几点:

(1) 一旦对象类型中定义了虚方法,那么其后辈类型中的所有同名方法也必须定义为虚方法。也就是说,静态方法不能覆盖虚方法,如 Show 和 Hide 在 Point 和 Circle 类型中都定义为虚方法。

(2) 子类型中虚方法的覆盖与静态方法的覆盖不同。虚方法不但要求与被覆盖方法的方法名相同,还要求形参表相同,即覆盖的虚方法只能是方法的程序体部分不同。

(3) 虚方法的调用语句虽与静态方法的调用相同,但一定要保证在虚方法调用前,调用该虚方法所属类型的构造方法,否则将会引起系统的死锁。当然,含有虚方法的对象类型中一定要定义构造方法。编译的语法检查在发现含虚方法的对象类型中未定义构造方法时也会提示错误信息。

细心的读者会发现:这节定义的方法树中,Circle 类型中没有定义 MoveTo 方法去覆盖 Point 类型中的 MoveTo 方法。那么,当调用方法时,会不会仍发生前节静态方法中的问题?回答是不会,而这正是虚方法的特点所在。为搞清虚方法的这一特点,必须搞清虚方法调用的执行过程,看编译程序是如何处理虚方法的调用的。图 12.3 是对象类型的数据格式,对象各节的实例将按此格式分配内存。

对象类型的数据域分配与记录类型相似,数据按其在对象类型中定义的次序排列。

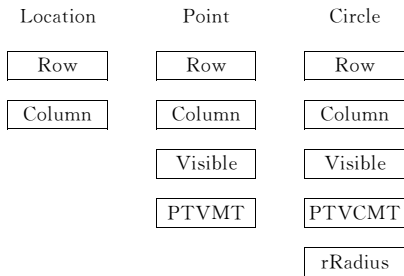


图 12.3 对象类型的数据格式

数据域分配的字节数视数据类型而定。如 Row 为 Integer 类型,将分配 2B。如有父辈类型,则所有被继承的数据域在前,本对象类型新声明的数据域在后。如果本对象类型中包含虚方法,编译还将在数据域中分配一个 16 位的指针域 PTVMT,指出所属对象类型的虚方法表 VMT(Virtual Method Table)在数据段中的偏移量。对象类型树中第一个定义虚方法的类型的 PTVMT 排在数据域之后;其后的子对象类型的 PTVMT 域都处在相同的位置,新定义的数据域将往后依次排列。如 Circle 的数据格式,其 PTVMT 是从 Point 中继承过来的(内容是 Circle 的 VMT 偏移量,与 Point 的 PTVMT 内容不同),而它自己的新定义数据 Radius 排在其后。

每个对象类型有一个虚方法表,简称 VMT 表。VMT 表在程序数据段的初始部分。该表由编译系统自动建立,不能由程序直接操作。以 Point、Circle 类型为例,其 VMT 表的结构如图 12.4 所示。Location 类型没有 VMT 表,因为它不含虚方法。

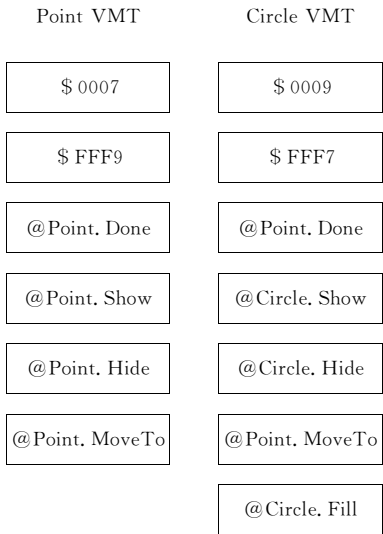


图 12.4 VMT 结构图

图中每格为一组数据,每组数据占 2B。第一组数据是相关对象实例在内存中应分配的字节数(含 PTVMT 域),如 CircleVMT 表中第一字为 \$ 0009,正是图 12.3 中 Circle 数据格式所占的字节数。Row、Column、Radius 各占 2B,Visible 占 1B,PTVMT 占 2B,共 9B。该数据使用 New、Dispose 过程为对象实例申请、释放动态空间时,提供应请求或释放的字节数。TURBO PASCAL 提供的 SizeOf 函数可以返回对象实例的这一值。例如,ACircle 为 Circle 类型的一个实例,那么函数 SizeOf(ACircle)将返回数值 \$ 0009。

VMT 表的第二个数据字是第一个字的补码,主要用于检查对象实例是否调用过构造方法,以决定虚方法调用的合法性。

VMT 表从第三组数据开始是按对象类型声明中虚方法的次序排列的指针(两字长),指向对应虚方法代码的入口地址,可按此指针转去执行虚方法。TypeOf 函数可指向对象类型 VMT 的指针。



每个含虚方法的对象类型都有一个虚方法表。下面比较一下两相互有继承关系的含虚方法的对象类型的虚方法表之间的关系。Point 的先辈类型中未曾定义虚方法,因此 Location 类型没有虚方法表,Point 类型的 VMT 中只有 Point 类型自己定义的虚方法入口地址指针。Circle 类型的虚方法表由三部分组成:从 Point 中继承的虚方法的指针,Circle 中定义的覆盖 Point 的虚方法指针,自己定义的新的虚方法指针。如 @Point. Done 和 @MoveTo 是从 Point 类型中继承的虚方法的指针,@Circle. Show 和 @Circle. Hide 是 Circle 中自己定义的且已覆盖了 Point 中的同名虚方法,@Circle. Fill 则是自己定义的新增的虚方法。

前面说过,对虚方法调用之前要先调用构造方法,否则将引起系统的死锁。因为构造方法的程序语句 begin 开始前自动执行了一段初始化代码,将对象类型的 PTVM 域的指针值赋给了调用对象虚方法的实例。注意:一个对象类型的各个独立的实例必须用单独的过程初始化,以便分别赋以 VMT 指针值;而这一赋值不能用对象实例间的赋值来替代。系统设置了 { \$R+ } 编译器指示,以检查每个虚方法调用的合法性,缺省状态为 { \$R- }。由于这一检查减低了运行速度,故应在调式成功后取消这一检查。

执行了对象实例的构造方法,实例就含有了其所属类型的 VMT 指针,当执行有虚方法调用语句:

<实例标识符>. <虚方法名标识符>

时,就将在自己的 VMT 表中找虚方法入口地址。方法中调用虚方法的过程也是如此的查找方法。

一般的过程和函数的调用是通过把实在参数压入堆栈的方法来进行参数传递的,方法的调用与其只有一点不同,就是在此基础上再将对象类型实例的指针(32 位)作为一个附加参数(即前面曾提到过的 Self 参数)压入堆栈。这一 Self 隐含参数为编译实现对象类型的封闭性提供了基础。结合图 12.3 和 12.4 举一方法调用所产生的代码实例,将使我们更清楚地了解这一过程。

设 PP: ^Point, 那么 PP^. Show(20,30) 调用所产生的代码为

```
Mov ax,20
Push ax;           {将第一个实参入栈}
Mov ax,30
Push ax;           {将第二个实参入栈}
Les di,pp;         {将 PP 指针传入 ES:DI}
Push Es
Push Di;           {ES、DI 入栈作为 Self 参数}
Mov di,ES;{di+6}; {将 Point 实例的 PTVM 域值,即 Point VMT 表在数据段的偏移量送 DI}
Call DWORD DTR[di+8];
                  {调用 Point.Show 方法,当方法调用结束返回时,要将参数和其他参数一起退栈}
```

比较 12.4 和 12.3 节中定义的静态对象类型 Point、Circle 与本节中定义的含虚方法的对象类型 Point、Circle,发现本节中定义的 Circle 类型中并未重新定义 MoveTo 方法去覆盖其先辈类型中的同名方法,那么会不会发生 12.4 节中所说的静态方法继承的问题呢?不会的,因为把 MoveTo 中引用的 Show、Hide 方法定义为虚方法后问题就解决了。如有实例:



```
var
  Apoint:Point;
  Acircle:Circle;
```

当实例 ACircle 调用 Acircle.MoveTo 之前,要先调用 Acircle.Init 构造方法,这时 Acircle 已含有 Circle 类型的 VMT 指针,以后凡是以 Acircle 为限定符的虚方法调用便都从 Circle 类型的 VMT 表中寻找虚方法的代码入口。回到上面的问题,尽管调用 Acircle.MoveTo 方法时,由于 Circle 类型中没有对 MoveTo 方法重新定义,即没有覆盖,因而必须到其先辈类型 Point 中去执行 Point.MoveTo 方法。但由于 Point.MoveTo 中的 Show、Hide 已定义为虚方法,执行时它将从 Circle 的虚方法表中去查入口。从图 12.4 中看到这时 Show 和 Hide 的入口为 @Circle.show 和 @Circle.Hide,是在 Point 对象中执行的 Circle 体中的方法。这种情况与前期装配的例子有本质差别。我们看到,MoveTo 方法虽然在 Point 类型中定义,但由于 Show 和 Hide 方法已定义为虚方法,MoveTo 调用 Show 和 Hide 时与静态方法不同,是连接 Point.Show、Point.Hide 还是连接 Circle.Show、Circle.Hide,在程序运行前是无法确定的,要视调用 MoveTo 的实例而定。如果是 Apoint.MoveTo,就连接 Point.Show;如果是 Acircle.MoveTo 就连接 Circle.Show。我们称这种连接为后期装配。虚方法的定义为后期装配提供了基础。

MoveTo 方法可由 Point 类型及其后继类型 Circle 共享。但 Point 的实例以点的特征来享用 MoveTo,产生点的移动;而 Circle 的实例以圆的特征来享用 MoveTo,产生圆的移动。也就是说,对象类型层次结构中的某动作方法可以被各层次的实例共享,而每个层次的对象实例又以自己的特有方式来享用实现这一动作,这种特性称对象类型的多态性。

对象类型中定义的 Destructor、Done 称为析构过程,将在 12.7 中介绍。

DP 12.6 对象类型的可扩展性

常规的程序设计习惯于应用过程、单元等手段把过去写好的程序括进来扩展自己。对象类型中虚方法支持下的后期装配功能使我们能够把后写的程序用来扩展已写好的程序。为区别常规的扩展方法,把这种扩展功能称为对象类型的扩展性。

从前节讨论的 MoveTo 方法中可以了解到对象类型的扩展性。如果 Point 类型及其所属 MoveTo 的方法是在单元 A 中定义且已编译过。这时的 MoveTo 方法只能移动一个点到新的位置。如在单元 B 中定义 Point 的后继类型 Circle 并在 Circle 中定义 Show 和 Hide 虚方法来覆盖 Point 中的同名方法,那么,在单元 B 之中再使用 MoveTo 方法(请注意这时的 MoveTo 仍是单元 A 中的且已编译过的方法),就不仅可以移动点,而且可以移动圆到新的位置。A 单元的功能被 B 单元给扩展了,显然对熟悉常规方法的人来说过去的程序设计方法是无法支持的,因而 OOP 设计提供了一个崭新的方法。

在举例说明扩展性前,先将前几节介绍的对象类型整理一下,并将其写在 figures 单元中。由于其中有的方法体已在前几节中定义过,为节省篇幅这里不再重写。其代码如下:

```
unit figures;
interface
```



```

uses Graph, Crt;
type
  location = object
    Row, Column: Integer;
    procedure Init(InitRow, InitCol: Integer);
    function GetRow: Integer;
    function GetCol: Integer;
  end;
  Point = object(Location)
    Visible: Boolean;
    constructor Init(InitRow, InitCol: Integer);
    destructor Done; Virtual;
    procedure Show; Visible;
    procedure Hide; Visible;
    function IsVisible: Boolean;
    procedure MoveTo(NewRow, NewCol: Integer);
    procedure Drag(Dragby: Integer);
  end;

```

```

Circle = object(point)
  Radius: Integer;
  Constructor Init(InitRow, InitCol: Integer;
    InitRadius: Integer);
  procedure Show; Virtual;
  procedure Hide; Virtual;
end;

```

Implementation

{前几节定义过的方法体略写}

```

Destructor Point. Done;
begin
  Hide;
end;
function Point. IsVisible: Boolean;
begin
  IsVisible := Visible;
end;
function
GetDelta(var DeltaRow, DeltCol: integer): Boolean;
var
  Keychar: char;
  Quit: Boolean;
begin
  DeltaRow := 0; DeltaCol := 0;
  GetDelta := true;

```

{0 表示位置无变化}

{函数值 true 表示有位置变化值返回}



```

repeat
    kryChar:=Readkey;           {第一次读键}
    Quit:=True;
    case ord(KeyChar0 of)
        27:begin                {27 为两字节键的第一字节}
            keychar:=readkey;    {读二字节键第二字节}
            case ord(KeyChar) of
                72:DeltaCol:=-1;  {up 减列值}
                80:DeltaCol:=1;  {down 增列值}
                75:DeltaRow:=-1; {left 减行值}
                77:DeltaRow:=1;  {right 增行值}
            else
                Quit:=false;      {其他二值键略}
            end;
        end;
        13:getDelta:=false;      {回车键不增值}
    else
        Quit:=false;            {忽略任何其他键}
    end;
until Quit;
end;
procedure Point, Drag(Dragby:Integer);
var
    DeltaRow,DeltaCol:Integer;
    FigureRow,FigureCol:Integer;
begin
    show;                        {显示被拖动的图形}
    FigureRow:=GetRow;           {取图形当前位置的坐标}
    FigureCol:=GetCol;
    while GetDelta(DeltaRow,Deltacol) do
        begin
            Figurerow:=FigureRow+deltaRow*Dragby;
            FigureCol:=FigureCol+DeltaCol*Dragby;
            MoveTo(figureRow,figureCol);
        end;
    end;
end;

```

drag 方法为依按键情况拖动图形(随不同对象而异)移动的方法。参数 **dragby** 给出移动的步数。它调用 **Getdelta** 函数,从中取得 **DeltaRow**(水平增量)和 **DeltaCol**(竖直增量)。然后将增量值加入旧坐标(**Row**、**Column**)以产生新坐标点,再调用 **MoveTo** 方法将图形从旧坐标位置移至新坐标位置。

Getdelta 过程是当按下 **Up**、**Down**、**Left**、**Right** 键后返回坐标增量,按 **CR** 键退出,按其他键继续循环等待。



下面的演示程序使用 Figures 单元。程序中扩展了一个新的图形对象类型 Arc, 以 Circle 为其直接先辈类型。程序中使用 Drag 这一在 figures 单元中已编译过的方法, 拖动的却是本程序中新扩充的图形——弧。例子说明了 Drag 方法的多态性。读者仔细阅读 figures 单元和本程序可体会到对象类型的扩展特性。程序代码如下:

```
program FigureDemo;
uses Crt,Dos,Graph,Figures;
type
  Arc=object(Circle)
    StartAngle,EndAngle: Integer;
    Constructor Init(InitRow,InitCol: Integer;
                     InitRadius: Integer;
                     InitStartAngle,InitEndAngle: Integer);
    procedure Show: Virtual;
    procedure Hide: Virtual;
  end;
var
  Graphdriver: Integer;
  GraphMode: Integer;
  ErrorCode: Integer;
  Anarc: Arc;
  Acircle: Circle;
  Constructor Arc. Init(InitRow,InitCol: Integer;
                        InitRadius: Integer;
                        InitStartAngle,InitEndAngle: Integer);
begin
  Circle. Init(InitRow,InitCol,InitRadius);
  StartAngle:= InitStartAngle;
  Endangle:= InitEndAngle;
end;
procedure Arc. Show;
begin
  Visible:= true;
  Graph. Arc(Row,Column,StartAngle,EndAngle,Radius);
end;
procedure Arc. Hide;
var
  TempColor: Word;
begin
  TempColor:= Graph. GetColor;
  Graph. SetColor(GetBkColor);
  Visible:= false;
  Graph. Arc(Row,Column,StartAngle,EndAngle,Radius);
```



```
GraphSetColor(TempColor);
end;
begin {main program}
  Acircle. Init(151,82,50);
  Anarc. Init(151,82,25,0,90);           {调构造方法}
  Anarc. Drag(5);
  CloseGraph;
end.
```

该例中引入 Arc 类型作为 Circle 的后继类型,因为弧是圆的一部分,所以 Arc 类型的数据域直接继承了圆心(Row、Column),半径(radius)和可见性(visible)等,又增加了描述自己特性所必须的两个参数:StartAngle 和 EndAngle 分别表示顺时针画弧时的起始角(圆心左側方向为 0°)和终止角。在方法的继承上要认真研究一下。首先要明确扩展 Arc 类型的目的是在原来类型的基础增加一个弧。过去可以在屏幕上用拖动一个圆的功能来扩展为拖动一个弧;有了 OOP 设计方法,立刻会想到能否和其先辈类型以多态方式共享拖动这一方法。在 Circle 类型中没有 Drag 方法的定义,于是就向 Circle 的前辈类型 Point 中去查找,在那里发现了 Drag 方法的定义;我们进而去查方法体的定义,可以看出 Drag 方法的本质是 MoveTo 方法;而 MoveTo 方法在前节已经了解到它是通过把 Show 和 Hide 方法定义为虚方法后进行多态共享的。因此这里只要在 Arc 类型中有自己的 Show 和 Hide 方法并定义为虚方法,就可以共享 MoveTo,进而共享 Drag。Arc. Show 和 Arc. Hide 是很容易写出的,例中列出了这两个方法体的定义,其中引用了 Graph. Arc 标准过程。该过程的功能是在(Row,Column)点上以 Radius 为半径,以 StartAngle 和 EndAngle 为起始角和终止角顺时针方向画一弧。

通过上面的实例总结一下定义虚方法的原则:

(1) 虚方法是为对象以自己的方式共享先辈类型普遍行为提供的手段。因此在总体设计类型时要优先考虑需要多态性的行为,把它作为方法写在前辈类型中。在这个方法中把需要后继类型带入自己特性的行为写成虚方法。而且要注意“一旦为虚,永远为虚”的原则。

(2) 和静态方式比较,虚方法是程序运行后才进行装配(即后期装配),所以速度相对慢。因此程序设计是采用静态方法还是虚方法要做扩展性与时空性的权衡。

(3) 调用虚方法之前必须调用该对象的构造函数,致于析构函数只在动态对象类型实例中才是必须的。

DP 12.7 动态对象类型

在介绍动态对象类型之前,对前几节留下的几个较深入的问题再做些讨论,以作为进一步研究的基础。

12.7.1 方法调用与一般过程和函数调用的不同

这个问题是编译程序内部机理,对用户是透明的,但用户有了这方面的知识就更有利于今后的使用。



一般的过程和函数的调用是将参数入栈后, 直接开始执行过程或函数的代码; 但方法却是在做完参数入栈后, 附加一个隐含参数 `self` 入栈; 该参数是 32 位的指针, 指向方法所在的对象类型实例, 通过这个指针再行调用。例如, 一个指针变量 `P` 指向 `Point` 类型实例, 则调用方法 `P.MoveTo(10,20)` 之前, 编译将自动产生如下代码:

```
mov AX,10;           {参数入栈}
push AX;
mov AX,20;
push AX;
les DI,P;            {将 32 位指针载入 ES:PI}
push ES;             {self 参数入栈}
push DI;
```

有了这个 `self` 参数, 对该实例的数据域进行处理就相当于执行了 `With self` 开域语句, 所以称为隐含参数。

12.7.2 构造方法和析构方法同一般方法调用的不同

构造方法调用与一般方法调用的区别在于编译系统自动将类型的 `VMT` 域内容 (它是单字节指针, 指出该类型虚方法表在数据域内的偏移量) 载入实例的 `VMT` 域, 为虚方法的调用做好了准备。如 `P` 为 `Point` 类型的一个实例的指针, 调用 `P.show` 将产生以下代码:

```
les DI,P;            {将 P 指针装入 ES:DI}
push ES;            {传入 self 参数}
push DI;
mov DI,ES[DI+6];    {从实例的 VMT 域取出指针}
Call DWORD PTR[di+8]; {从 VMT 表找到 show 的代码入口}
```

从代码中看到如果没有实例的 `VMT` 域指针装入过程, 虚方法表的首址无法找到, 因而虚方法的调用不会成功。所以前面强调在调用虚方法之前, 必须调用构造方法。

而析构过程被调用结束时, 编译系统自动生成一个“后记”性代码, 做了一项特殊的工作, 就是首先它要测试 `VMT` 域的值。如果为零, 意味着以前的调用是一般的方法调用, 不需做任何工作; 如果 `VMT` 域值非零, 说明前面曾申报动态内存成功, 要到 `VMT` 表的第一字取得申请内存的字节数, 将堆中的动态内存释放。由于是“后记”性代码, 上述的功能即使析构方法自己为空, 照样完成任务。如下面形式的析构函数:

```
destructor AnObject, Done;
begin
end;
```

是常见的。从析构函数和构造函数的特殊功能分析, 可以看出有虚方法, 构造函数的调用才有实际意义, 有动态分配的对象类型实例析构函数才有作用。

12.7.3 Pascal 的动态内存管理

图 12.5 表示了 `TURBOPASCAL` 程序的内存映象, 内存的低地址从下端开始首先是



DOS 对 .EXE 文件的管理区, 俗称程序前缀 psp; 接着从程序代码段开始依次排着 PASCAL 源程序引用的各单元代码段; 以 system 系统单元代码段为结束; 然后是类型常量和全局变量区; 再向上分布三个内存区域是 PASCAL 程序编译后, 运行时使用的内存区。

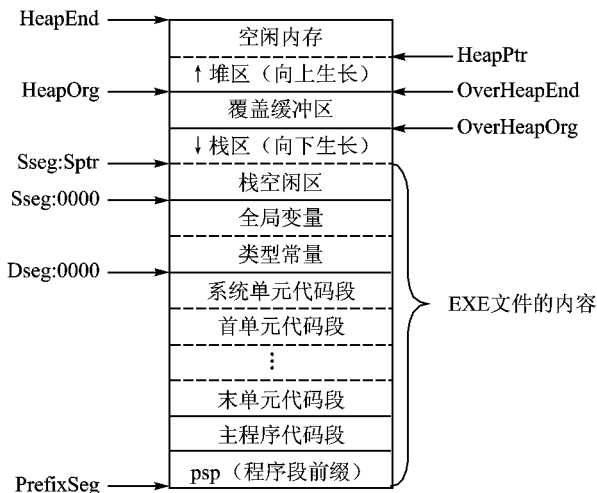


图 12.5 TURBO pascal 内存映象

栈区 从 OverHeapOrg 开始, 向下到 Sseg:0000 这一段为栈的动态使用区域, Sseg:Sptr 指示当前栈顶的位置。前面讲过的当过程和函数调用时传递参数等都要用到它。

覆盖缓冲区 从 OverHeapOrg 开始, 到 OverHeapEnd 止, 按需要为 PASCAL 提供调入单元代码, 以节省内存空间; 这类过程或函数以 Overlay 标识, 被放在外存器上, 当调用时再调到覆盖区; 调用完成后, 当新的 Overlay 过程被调入时, 再被覆盖掉。

堆区 从 HeapOrg 开始向上生长到 HeapEnd 止。它为变量或对象实例提供临时创建的空间。这项技术特别有利于变量数量随应用发生变化的情况, 如动态链表等。HeapPtr 指出当前堆顶的地址, 动态对象类型就是利用堆区来实现的。这是这节下面要讨论的内容。

TURBO PASCAL 为用户提供了标准过程或函数来使用堆区。这里介绍 New 过程和 Dispose 过程。

例如, Ptr1 为某记录类型变指针变量, Ptr2 为某对象类型实例的指针, 那么当调用 New (Ptr1) 时, 它得在堆中从低部开始分配一块内存, 其大于等于该记录类型所需的内存量; 调用后, Ptr1 保留了堆顶的地址 (假设从前没有申请过堆的内存, 因而 HeapPtr = HeapOrg), 即 HeapPtr (= HeapOrg) → Ptr1; 然后 Ptr1 + 所需内存字节数 → HeapPtr; 调用结束后, Ptr1 带回了堆中申请内存的起始地址。

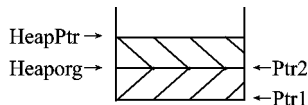


图 12.6 堆区的动态分配

如果再调用 New (Ptr2), 则在 Ptr1 申请后的基础上重复上述过程, 又为 Ptr2 指向的实例申请了相邻的又一块内存, 如图 12.6 所示。

如果有的想释放掉 Ptr2 实例所占用的内存, 则调用 Dispose (Ptr2), 有 Ptr2 → HeapPtr, 说明内存已释放。如果



有申请又从这里向上开始。

扩展的 New 还允许作为一个函数返回一个指针值,这时 New 的参数是指向对象的指针的类型,而不是指针变量自己。例如:

```
type
    CirclePtr = ^Circle;
var
    PCircle: CirclePtr;
```

则调用

```
Pcircle := New(CirclePtr)
```

由此看到 New 的参数是 CirclePtr,是一个指针类型;而 New 作为函数返回的值赋给了指针类型变量 PCircle。

有了上面的了解,可介绍动态对象类型的使用。以前介绍的对象实例是在 VAR 声明中定义所属的对象类型,这样的实例是在数据段的变量部分分配内存的。现在所谓动态对象类型是只在 VAR 中定义指针,在程序运行中用 New 和 Dispose 等过程在堆上动态分配内存,因而称为动态对象。看下面的例子,借用 12.6 节中定义的 Circle 类型:

```
var
    PCircle: ^Circle;
```

把 Pcircle 定义为 Circle 对象类型的指针变量,这时编译程序在全局变量区为其分配了一个 32 位的 Pircle 指针变量空间。Circle 对象类型实例并未分配内存空间,当执行 New(PCircle)过程后,堆区为其分配了空间,并把该空间的首地址传给了 Pcircle。当对象类型调用虚方法时,必须调用构造过程。前面介绍 Circle 的构造过程定义为

```
constructor Init(InitRow, InitCol; InitRadius: Integer);
```

因此应该按下面方法首先调用:

```
Pcircle^.Init(10, 20, 30);
```

TURBO PASCAL 提供了一个更简便的形式,来代替这两个过程,即

```
New(PCircle, Init(10, 20, 30));
```

前面讲到,动态分配的对象结束时必须调用析构过程来释放占用的内存, TURBO PASCAL 给出了类似的过程形式:

```
Dispose(PCircle, Done);
```

该过程的执行是在做完 Done 过程的实体之后用“后记性”代码从 VMT 中查出实例类型的大小,并传递给 Pcircle 执行 Dispose 释放掉占用的堆中内存。当然,如果只为释放内存这项任务,可以定义析构过程 Done 为空过程。

下面举一个动态对象实例内存分配的例子。



建造一个对象的链表,从中看出如何给新加入链表的对象接入链并申请堆上的内存,以及如何在撤掉链表时用析构过程释放堆上的内存。仍以前节定义的 Point、Circle、Arc 对象类型为基础来做。设它们的指针类型分别为 PointPtr、CirclePtr、ArcPtr。由于上述的类型中没有链表指针的数据结构,故建立一个记录类型:

```
Nodeptr:=^Node;  
Node=record  
    Item:PointPtr;  
    Next:NodePtr;  
end;
```

其中,Item 作为指向对象 PointPtr 的指针类型。我们知道对象类型的兼容性,Item 也可以作为 Circle 和 Arc 类型的指针;Next 域作为指向下一个节点的链指针。

再创建一个对象类型 List。它一方面作为链表的表头,一方面将处理链表的方法含在其中,代码如下:

```
Listptr:=^List;  
List=object  
    Nodes:Nodeptr;  
    constructor init;  
    destructor done;virtual;  
    procedure Add(item:pointptr);  
end;
```

其中,Nodes 域作为表头指针,除初始化的构造过程和析构过程外,还有接入链表的 Add 方法。

整个链表的数据结构如图 12.7 所示,List 的 Nodes 域指向链表首节点作为表头,而链表的尾节点的 Next 域置空 Nil,图中用接地表示。每个节点的 Item 域指向动态对象申请来的堆中内存首址。

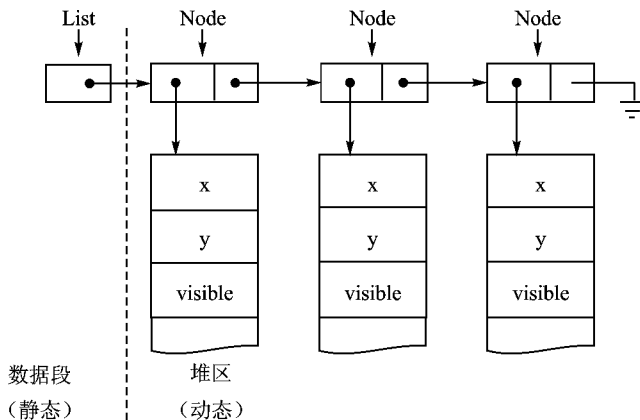


图 12.7 动态对象链表的数据结构

首先在 List 的构造函数中做一件事,其代码如下:



```

constructor List,init;
begin
    Nodes:=nil;
end;

```

把表头的指针域置空,表明链表无节点接入,处在初始状态。

接下来看如何在 Add 方法中实现节点加入链表,其代码如下:

```

procedure List.Add(item;pointper);
var
    n:Nodeptr;
begin
    New(n);
    n^.item:=item;
    n^.next:=Nodes;
end;

```

该方法是以 PointPtr 指针类型做参数,将该指针指向的动态对象实例加入到链表的头部。注意到对象类型的兼容性,Point 类型的后继类型 Circle、Arc 的实例指针也可以作为该方法的参数,也就是链表中可含有一切 Point 类型后继类型实例。方法借用了临时的局部指针变量 n,用 New(n)先为记录类型 Node 申请了堆上的一块内存,作为 Node 节点。插入到表头之后的过程,读者可自续程序。

下面再看一下析构过程释放全部链表所占空间的例子。代码如下:

```

destructor List.Done;
var
    n:NodePtr;
begin
    while Nodes <> nil do
    begin
        n:=Nodes;
        dispose(n^.item);
        Nodes:=n^.next;
        dispose(n);
    end;
end;

```

有了上述的准备,可用如下的语句:

```

Alist.Init;
Alist.Add(New(circleptr,init(400,300,30)));
Alist.Add(New(arcptr,init(500,2020,30)));

```

建立链表。

注意:New 中的第一参数 circleptr、arcptr 是对象指针类型,而不是指针本身。

也可一次消去链表,代码如下:

```

Alist.done;

```



以上只是给出了动态对象类型使用的一些关键点,要想掌握面向对象的程序设计方法,读者要动手编写程序上机实验。一些细节还请查阅相关专业书籍的说明。

DP **习题与思考题**

- 12.1 举例说明对象的概念。
- 12.2 什么是对象的封装性和继承性？
- 12.3 如何为对象定义一个虚方法,并分析虚方法是如何支持多态性的？
- 12.4 构造方法与析构方法的作用是什么？

参考文献

- 1 万兆泉等. Delphi 5 编程指南. 北京:北京希望电子出版社,2000
- 2 李峰,聂文惠. Delphi 5 实用例库与高级编程技巧. 北京:人民邮电出版社,2000
- 3 魏志强,王忠华,何敏等. Delphi 5.0 程序设计(数据库应用实务篇). 北京:中国铁道出版社,2001
- 4 唱江华,张益铎. Pascal 语言程序设计. 北京:北京航空航天大学出版社,1991