Turbo Pascal

編程技巧

曹文辉 张卫 编著

原子能出版社 Atomic Energy Press

Turbo Pascal
编程技巧

曹文辉 张卫 编著

原子能出版社

(京)新登字 077 号

内容简介

本书系统地介绍了 Turbo Pascal 语言提供的多种功能、编程方法及技巧。对每一个功能，除了详细阐述其基本概念和具体实现外，还尽可能给了每一例程的编程技巧和调用说明。本书概念清楚，例程丰富，适合于 Turbo Pascal 初级学习者，也适合于开发 Turbo Pascal 的高级编程员。

本书适用于使用 Turbo Pascal 各版本的程序设计者，可供从事计算机系统开发与应用开发的科技工作者参考。

图书在版编目(CIP)数据

Turbo Pascal 编程技巧/曹文辉，张卫编著. —北京：原子能出版社，1995.4
ISBN 7-5022-1251-5

I.T… Ⅱ.①曹…②张… Ⅲ.PASCAL 语言-程序设计 Ⅳ.TP312PA

中国版本图书馆 CIP 数据核字(94)第 09870 号

原子能出版社出版　发行
责任编辑：李　涛
社址：北京市海淀区阜成路 43 号邮政编码：100037
原子能出版社印刷厂印刷　新华书店经销
开本：787×1092 1/16 印张：27 字数：671 千字
1995 年 4 月北京第 1 版　1995 年 4 月北京第 1 次印刷
印数：1—1500
定价：34.70 元

前　言

自从美国 Borland 公司 1985 年推出 Turbo Pascal 语言以来，就一直风靡计算机界。它的友好的编程环境(IDE)，高速的编译能力和提供的各种编程软件包，使许多微机编程者为之倾倒。它还具有易读、易写、易移植的特点，深受广大用户欢迎。Turbo Pascal 除保留了标准 Pascal 的结构外，又进一步扩展了该语言的处理能力。它不仅提供了高效的数值运算功能，还提供了系统的低级调用手段，在动态内存管理、文件管理、图形图像处理、输入输出等方面功能也很强大，许多用户已利用它成功地开发了各种系统软件和应用软件。

本书适用于使用 Turbo Pascal 4.0，5.0，6.0 版本的程序设计者，为读者提供有关 Turbo Pascal 最新颖、最全面的参考资料，涉及了软件开发的各个方面——界面设计、访问 DOS 和 BIOS 服务，与各种语言的接口，图形、工具箱的应用，面向对象程序设计等等。每部分都提供了大量实用而富有技巧的实例，便于读者理解与掌握，并能直接加入到自己的程序中。

本书共分十一章，第一章介绍了 Turbo Pascal 的概念与特点；第二、三、四、五章介绍了 Turbo Pascal 的 BIOS 和 DOS 服务，中断与通讯，扩展内存，及文件操作；第六章介绍怎样设计友好的用户界面，包括键盘控制、窗口、菜单以及命令行；第七章介绍面向对象的程序设计；第八章介绍 Turbo Pascal 与其他语言的混合编程，包括 Turbo Pascal 与汇编接口，Turbo Pascal 与 FoxBase 数据通信；第九章介绍怎样使用 Borland 公司提供的两个通用工具箱；第十章介绍图形程序的设计；第十一章介绍 Turbo Pascal 6.0 的新软件包 Turbo Vision 的应用。

在本书编写过程中，曾得到赵岳、于烈、赵剑平及其他朋友们的支持和帮助，在此表示衷心的感谢。书中不当之处，敬请读者赐教。

编　者

1994 年 6 月

目　录

前　言
第一章　Turbo Pascal 基本概念与特点 …… (1)
1.1　程序结构 …… (1)
1.2　数据类型 …… (17)
1.3　变量与常量 …… (26)
1.4　过程与函数 …… (33)
1.5　程序和单元 …… (40)

第二章　BIOS 和 DOS 服务 …… (45)
2.1　BIOS 服务 …… (45)
2.2　DOS 服务 …… (60)
2.3　未公布的 DOS 服务 …… (69)

第三章　中断与通讯 …… (72)
3.1　使用中断 …… (72)
3.2　编写中断管理器 …… (74)
3.3　PC 远程通讯 …… (75)

第四章　使用扩展内存 …… (86)
4.1　扩展内存工作原理 …… (86)
4.2　LIMEMS 单元 …… (88)
4.3　与扩展内存相关的编程问题 …… (91)

第五章　Turbo Pascal 文件 …… (97)
5.1　文件句柄概念 …… (97)
5.2　Turbo Pascal 文本文件 …… (97)
5.3　磁盘文件和缓冲区 …… (103)
5.4　类型文件 …… (104)
5.5　无类型文件 …… (107)
5.6　类型文件和无类型文件过程 …… (109)
5.7　删除和修改文件名 …… (112)

第六章　界面设计 …… (114)
6.1　视频单元 …… (114)
6.2　键盘单元 …… (124)

6.3 窗口 …… (129)
6.4 菜单 …… (140)
6.5 命令行分析 …… (156)

第七章 面向对象的程序设计 …… (165)
7.1 对象的概念 …… (165)
7.2 继承性 …… (166)
7.3 对象的定义 …… (166)
7.4 方法 …… (168)
7.5 封装 …… (173)
7.6 关于对象的进一步编程 …… (174)
7.7 小结 …… (189)

第八章 Turbo Pascal 与其它语言的接口 …… (190)
8.1 嵌入式汇编程序 …… (190)
8.2 Turbo Pascal 与 Turbo Assembler 的接口 …… (206)
8.3 Turbo Pascal 与 FoxBase 的数据通信 …… (228)

第九章 Borland 公司提供的 ToolBox …… (241)
9.1 数据库工具箱(DataBase ToolBox) …… (241)
9.2 使用数值方法工具箱 …… (264)

第十章 图形程序设计 …… (282)
10.1 Graph 单元 …… (282)
10.2 图象处理与图象文件 …… (318)
10.3 图形显示中鼠标的应用 …… (339)
10.4 用绘图仪进行图形输出 …… (373)

第十一章 编写 Turbo Vision 应用程序 …… (388)
11.1 第一个 Turbo Vision 应用程序 …… (388)
11.2 桌面,菜单条和状态行 …… (389)
11.3 打开窗口 …… (393)
11.4 窗口特性 …… (395)
11.5 创建一个对话框 …… (403)
11.6 其它对话框控制 …… (412)
11.7 标准对话框 …… (412)
11.8 建议和忠告 …… (413)
11.9 演示例程 …… (416)
参考文献 …… (424)

第一章　Turbo Pascal 基本概念与特点

Pascal 语言是 70 年代初期由 Niklaves Wirth 为讲授程序设计而设计的。因此 Pascal 特别适合作程序设计初学者使用的语言，而且对于已有使用其它语言进行程序设计经验的用户，学会 Pascal 更是易事。

Turbo Pascal 作为 Pascal 语言家族的一员，功能齐全，而且体现了当代程序设计语言的特点——易读，易写，易移植。Turbo Pascal 除保留了标准 Pascal 语言鲜明的结构外，还进一步发展了这种语言的处理能力，因而深受用户欢迎。Turbo Pascal 以应用为对象，向程序设计者提供了功能齐全的各种程序手段。自 1985 年 Borland 公司推出 Turbo Pascal 以来，Borland 公司几乎每隔二年推出一个新版本。目前 Turbo Pascal 最新版本为 6.0。Turbo Pascal 6.0 在 Turbo Pascal 5.5 的基础上进一步扩展了面向对象的程序设计，提供了功能强大，面向对象的软件开发工具，提供了功能完备的嵌入式 InLine 汇编器等等。可以说 Turbo Pascal 是目前用于微机程序设计最有成效的开发环境和工具.

为使用户走向 Turbo Pascal 程序设计之路，本章将介绍 Turbo Pascal 语言的基本概念，并展示它们在程序中的使用方法。

1.1　程序结构

Pascal 是一种结构化语言，它由严格定义的各部分组成，每个部分都有特定的含义。Turbo Pascal 程序主要包括程序头、数据部分和代码部分，其组成如图 1.1 所示：

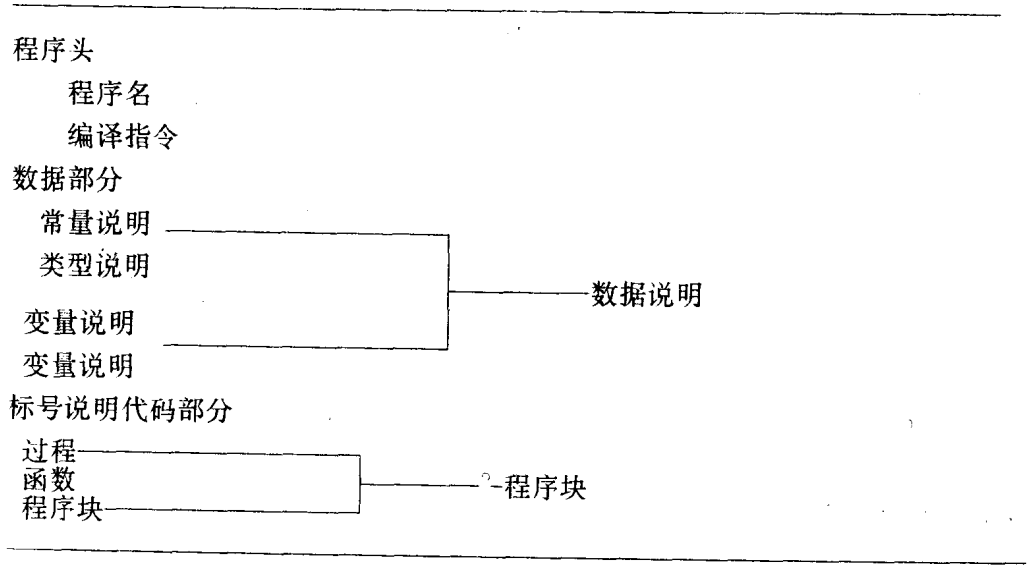

图 1.1　Turbo Pascal 程序的结构

1.1.1 程序头

一个 Turbo Pascal 程序的开始两行一般由程序名和编译指令组成，它们都是任选的，但为了程序文档完备起见，最好能包括它们。

作为程序的第一行，程序头只不过是标识了程序的名字以及程序中是否使用输入输出。一个典型的程序头如下：

```
Program ProgName(Input,Output);
```

注意，Turbo Pascal 允许你在程序名后添加一个参数表。这是从标准 Pascal 那里继承下来的，它在标准 Pascal 里是必需的，但 Turbo Pascal 可以忽略。

程序第二行包含编译指令，它在 Turbo Pascal 程序里扮演重要的角色，控制着错误检查和输入/输出，虽然初学者可以不管这些编译指令，但高级程序员必须懂得怎样使用这些选择项以充分发挥 Turbo Pascal 效率。

Turbo Pascal 编译器提供了很多选择项，使用它们可更加容易地编程和调试。编译指令可大致分为三类：开关指令、参数指令和条件编译。

开关指令

开关指令具有打开或关闭 Turbo Pascal 的特殊性功能，如输入/输出错误检查，堆栈检查和数据对齐。它们之所以称为开关指令是因为它们只有两个状态：接通和关闭。

开关指令由大写或小写的单字母标识。如 S 指令控制堆栈检查，R 指令设置范围检查，I 指令指定输入/输出错误检查等等。允许或禁止一个编译指令的格式为一个 $ 号后跟编译指令，可以是一个正号(允许)或者是一个负号(禁止)；这些字符被限定在注释定界符内，或者是带 * 的圆括号或者是花括号。下面是一些有效编译指令语句的例子。

```
(*$I-*)
{$i-}
{$s+,v-,r+,a+}
```

前两个语句是相同的，都禁止输入/输出错误检查。从这个例子可以看出，可使用两种类型的注释定界符：花括号或者带 * 号的圆括号。指令的大小写是不重要的。第三个例子一次说明了四个编译指令，它允许堆栈溢出检查，禁止变量字符检查和允许范围检查及按字对齐数据。

Turbo Pascal 允许用两种方式设置编译指令。最简单的方式是从 Options/Compiler 菜单上设置。在这个菜单里的指令设置影响所有程序和单元的全局缺省值。换句话说，Turbo Pascal 如没有在源代码里说明编译指令，则使用 Options/Compiler 设置。

有两种类型的开关指令：全局指令和局部指令。顾名思义，全局指令从头到尾影响整个程序的编译，必须在程序单元的开始予以说明(必须在第一个 Uses，常量，标号，类型，过程，函数或 Begin 关键字前面)，局部指令可在程序的任何位置出现，它只影响这个指令后面的程序。例如，你可在一个过程的开始打开范围检查局部指令，然后在过程的结束关闭这个指令。

★ 对齐数据(全局)Align Data

当数据在偶数地址(也称为字边界)对齐后，8086 微处理机系列可快速访问内存。当对齐数据编译器有效时({$A+})，Turbo Pascal 确保每个大于 1 个字节长的变量和有类型常数都在偶数地址开始。在字边界上，对齐数据加快了数据存取，但也增加了存贮数据需要的内存

量，因为必须在必要的地方插入"死"字节以确保变量在字边界开始。如果关心的是程序使用的内存量，可禁止这个指令({$A-})。

★ 布尔变量求值(局部) Boolean Evaluation

Turbo Pascal 支持两种类型的布尔变量求值——完全布尔变量求值和短路布尔变量求值。为理解二者之间的差别，考虑下面的例子：

```
If (a<b) And (b>c) Then
  Begin
    .
    .
  End;
```

上面的布尔语句是两个由 And 连接的单独比较表达式组成。完全布尔变量求值时，Turbo Pascal 在分支前将测试两个比较表达式。但是，如果 a 大于等于 b，实际上就没有必要测试是否 b 大于 c。在短路求值时，Turbo Pascal 只测试能确定整个表达式的结果就可以了。在特定情况下，短路求值能相当可观地加快程序的运行。进行短路求值，使用{$B-}，选择完全求值，使用{$B+}。

★ 调试信息(全局) Debug Information

允许调试信息指令({$D+})，指示 Turbo Pascal 产生可执行指令匹配到源文件位置所需要的信息。这些信息允许逐行检查一个程序或者当出现运行错误时定位源程序。Turbo Pascal 向.TPU 文件的结尾添加调试信息，使文件增大。但这并不影响可执行代码的速度。

禁止这个指令，就不能单步执行一个程序，但能节约磁盘空间，缩短编译时间。一般来说，应该允许调试信息指令。

★ 仿真(全局)Emulation

8087 数字协处理器芯片比 8088/8086 具有显著的运算优点。但并不是所有的计算机都配置有 8087，允许时，仿真编译指令允许访问 8087 数据类型而不管是否配置有这个数字芯片。

当仿真被允许后，Turbo Pascal 检查是否安装了 8087 芯片。如果有，程序将使用这个协处理器的处理能力；如果没有的话，程序将使用主微处理器实现 8087 运算。除非需要特别精确的数学结果，不然最好禁止仿真指令。

★ 强制远调用(局部) Force Far Calls

Turbo Pascal 支持多个代码段——每个单元一个代码段，主程序有另外一个代码段。在一个单元或程序里发生的函数和过程调用称为段内调用或近调用；当一个单元的语句调用另一个单元内的过程时，调用越过了代码段边界，因此就称为段间调用或者远调用，近调用比远调用做的工作要少，远调用，由于牵涉到不止一个代码段，做的工作要多，执行速度也慢。Turbo Pascal 知道什么时候使用远调用和近调用。然而有时需要强制一个过程为远调用。

必须强制使用远调用的环境一般来说是非常高级，必要时可通过允许 F 编译指令强制一个过程为远调用。演示如下：

```
Program TestFarcall;
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
{$F+} Procedure Farcall; {$F-}
Begin
```

```
End;
{**************************}
Begin
  Farcall; {This would normally be a near call}
End.
```

很明显，过程 Farcall 和程序结束时的调用处于同一个代码段内，这意味着这个过程应该被执行为近调用。F 编译指令通过强制这个过程为远调用改变了这种情况。

★ 输入/输出检查(局部) Input/Output Checking

I 编译指令，它用来检查程序里的 I/O 错误，用语句{$I+}表示允许。I/O 错误可能是最常见的 Turbo Pascal 错误类型，也是最危险的错误之一。

当允许输入/输出检查指令后，一个 I/O 错误会产生一个运行错误并停止程序。然而允许 I 编译指令并不是处理 I/O 错误的最好方式。一个更为有效的方式是用语句{$I-}禁止 I 指令，用户自己来俘获 I/O 错误。

★ 局部符号信息(全局) Local Symbol Information

局部符号信息指的是局部于一个单元或过程的变量和常量信息，Turbo Pascal 通常并不保存这些符号的信息，这样在调试对话中不能看到或者改变它们的值。通过允许局部符号信息编译指令{$L+}，Turbo Pascal 产生并保存关于所有局部变量的信息以供调试程序时使用。

局部符号信息编译指令和调试信息指令以下面的方式共同工作。当禁止调试信息时({$D-,L-})或者{$D-,L+}，Turbo Pascal 不为调试目的保存任何信息，L 编译器也不起作用。当允许 D 指令，禁止 L 指令时({$D+,L-})，Turbo Pascal 只为全局指令和常数存放调试信息。当 D 指令和 L 指令都允许后({$D+,L+})，Turbo pascal 为全局和局部符号保存信息。

★ 数值处理(全局) Numeric Processing

Turbo Pascal 提供了两种类型的浮点运算处理：正常方式和 8087 方式。正常方式支持 6 字节数据类型。8087 方式提供了另外四种浮点数据类型和使用 8087 数字芯片的能力。使用{$N-}编译器指令选择正常方式，使用{$N+}选择 8087 方式。

N 编译指令和 E 编译指令联合使用，当二者都允许时({$N+,E+})，即使没有装配 8087 协处理器，程序也使用 8087 方式，如果装配了这个芯片，程序将它用于浮点操作。否则程序将使用仿真程序，它提供了同样的精度级别，但速度比较低。当在 8087 方式下没有允许仿真指令时({$N+,E-})，程序将只能在装配有 8087 协处理器的机器上运行。

★ 覆盖代码产生(全局) Overlay Code Generation

如果要在一个覆盖文件里引用一个单元，必须允许覆盖代码产生编译指令({$O+})，这告诉 Turbo Pascal 产生把这个单元管理为覆盖所必需的代码。

★ 范围检查(局部) Range Checking

Turbo Pascal 里的大多数数据类型都有限制。例如，一个字节容纳不下大于 255 的值，一个具有 5 个元素的数组不能容纳 6 个元素。定义为 String[20]的字符中不能容纳 21 个字符。任何违反这些限制的操作都会产生一个范围错误。

当允许范围检查后{$R+}，Turbo Pascal 产生代码以检查所有数组下标和赋值是否出界。出现一个范围错误后，Turbo Pascal 产生一个运行错误，停止这个程序。

范围检查显著地增加编译后程序的大小，执行速度变慢。

★ 堆栈溢出检查(局部) Stack Overflow Checking

当程序调用一个过程或者函数时,Turbo Pascal 从局部变量堆栈里分配内存,允许堆栈溢出检查编译指令告诉 Turbo Pascal 插入必要的代码以确保在堆栈里有足够的内存来容纳这些局部变量。如果没有足够的内存,程序会停止并出现一个运行错误。如果禁止 S 编译指令,将不做任何检查,当程序用完堆栈内存时,计算机就要失灵。堆栈检查花费时间、增加程序的可执行代码,因此只有在开发和调试程序时才能用 S 编译指令。

★ 变量字符串检查(局部) Var String Checking

当允许变量字符串检查时,Turbo Pascal 对传递给过程和函数的字符参数进行严格的检查。

```
Program TestVarStr;
Type
  Str30 = String[30];
Var
S: String[20];
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
Procedure ChangeS(Var S:Str30);
Var
  i:Integer;
Begin
For i:=1 To 30 Do
  s[i]:=Chr(Ord(s[i] + 1);
End;
(* * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
Begin
S:= 'adc';
ChangeS(s);
WriteLn(s);
ReadLn;
End.
```

在这个程序里变量 S 说明为 String[20],传递给过程 ChangeS 的参数类型为 String[30],如果允许了变量字符串检查({$V+}),这个程序将通不过编译,因为参数类型不匹配。

参数指令

不像开关指令,参数指令没有明显定义的接通/关闭状态。这些指令指明在编译期间使用的文件名和分配给程序的内存大小。

★ 包含文件(局部)Include File

一个包含文件是一个源文件,它作为另外一个源文件的一部分被编译。如果你没有为 I 指令指定扩展符,Turbo Pascal 假设它为. PAS。

为了理解它是怎样工作的,下列过程 ProcA 包含在名为 MAININC. PAS 的文件里。主程序文件使用包含编译指令,把 MAININC. PAS 的代码插入到程序里。当一个程序太大不能一次放进 Turbo Pascal 编辑器里或者当你希望通过改变包含文件的值来改变作为一个整体的程序时,一般都使用包含文件。

```
(* Contents of file MAININC.PAS *)
```

```
Procedure ProcA;
Begin
WriteLn('ProcA');
End;
(* Program Using MAININC.PAS *)
Program Main;
{$I MainInc}
Begin
ProcA;
ReadLn;
End.
```

★ 连结目标文件（局部）Link Object File

如果用汇编语言编写 Pascal 程序的子程序，需要把这个汇编程序的目标文件和 Pascal 程序连结起来。它是由连结目标文件编译指令 L 后跟目标文件名而实现的。如：

```
{$L MyProg.obj}
```

★ 内存分配大小（全局）Memory Allocation Sizes

内存分配大小(M)编译指令提供了对程序里栈和堆所使用的内存量。这个指令后跟三个用逗号隔开的数值，分别代表栈的内存量和堆的最小量和最大量。例如指令{$M 30000，1000,5000}分配给栈 30,000 字节，分配给堆的最小量和最大量分别为 1000 和 5000，分配给栈的内存量必须从 1024 到 65520，可分配给堆 0 到 655360 字节。

★ 覆盖单元名（局部）Overlay Unit Name

覆盖单元名编译指令(O)后跟单元名并指示 Turbo Pascal 将该单元包含在覆盖文件里，在这个指令中命名的单元必须用{$O+}编译指令进行编译以允许它被覆盖。覆盖单元编译指令必须放在 Uses 子句的后面，说明如下：

```
Program OverTest;
Uses
  Unit1,
  Unit2,
  Unit3,
  {$O Unit1}
  {$O Unit2}
```

在这个程序的说明段里，Unit1 和 Unit2 被命名为包括在覆盖文件里，Unit3 不能被覆盖。

条件编译

Turbo Pascal 的条件编译指令允许在同一个文件里保持一个程序的不同版本。通过改变某些定义，就可编译某些代码段而把其它藏起来。条件编译的关键是条件符号，下面的程序将演示条件符号是怎样定义并用来控制编译的。

```
Program ConditionExample;
{$DEFINE TEST}          (* Define symbol *)
{$IFDEF TEST}           (* If' symbol defined,do this··· *)
{$R+,S+}
{$ELSE}
```

```
                    (* If symbol defined,do this··· *)
{$R-,S-}
{$ENDIF}          (* End of conditional compilation.  *)
Begin
{$UNDEF TEST}    (* Undefine the symbol *)
{$IFDEF TEST}    (* If symbol defined,do this... *)
WriteLn('TEST DEFINED'); (* If symbol not defined,do this... *)
{$ELSE}
WriteLn('TEST NOT DEFINED');
{$ENDIF}              (* End of conditional compilation.  *)
ReadLn;
End.
```

程序通过定义符号 TEST 开始，第一次使用 TEST 时，如果 TEST 已被定义就允许 R 和 S 编译指令，如果 TEST 没有被定义就禁止它们。在关键字 Begin 后，TEST 的定义被解除，取消了前面的{$DEFINE TEST}指令，因为 TEST 已不是被定义的，WriteLn('TEST NOT DEFINED');这一行被编译。下面描述 Turbo Pascal 提供的条件编译指令。

★ 定义(DEFINE)

使用这个指令来定义一个条件符号。任何依赖被定义符号的代码都将被编译，但只限于出现 DEFINE 指令的文件里的代码。如果在主源文件里使用 DEFINE 指令，在包含文件和单元文件里不使用，那么只影响主源文件；其它文件不带 DEFINE 指令进行编译。定义一个全局编译符号的唯一方式是用 options/compiler 菜单上的条件定义功能。

★ UNDEF

本指令对 DEFINE 指令取反。一旦一个符号用 UNDEF 指令使用过，依赖于那个符号的代码都不被编译。

★ IFDEF

如果一个命名的条件符号被定义，本指令指示 Turbo Pascal 编译代码。

★ IFNDEF

如果一个命名条件符号没有被定义，本指令指示 Turbo Pascal 编译代码。

★ IFOPT

可以根据另外一个编译指令来控制编译。例如，如果写了{IFOPT R+}，那么只有当允许 R 编译指令时，才能编译代码。

★ ELSE

当第一个条件不为真时，在作为分支的任何 IF... 指令(IFDEF，IFNDEF，IFOPT)后使用这个指令。

★ ENDIF

本指令标志条件编译序列结束。在 ENDIF 语句后面出现的任何代码都不依赖于条件符号。

有效地使用编译指令是通向高级程序员的重要一步。你必须懂得怎样使用这些指令以发挥 Turbo Pascal 的最大效率。

1.1.2 数据部分

在 Turbo Pascal 里，全局变量、常数、标号和用户定义的数据类型是紧接着程序头和全局编译指令之后说明的，局部变量在过程和函数里说明，但遵循同样的基本规则。

常量定义

很多程序都有一些从不改变的值，使用这些常量标识符简化了程序并且使它们容易维护。对于在整个程序里都得到反映的变化，只须改变这个常数的值。

Turbo Pascal 保留字 const，标志常量定义块的开始。

Turbo Pascal 里有两种常量类型：无类型的和有类型的。无类型的常量用下面语法说明：一个紧跟等号的标识符，一个文字值(数值或文本)和一个分号，如下列所示：

```
CONST
DaysPerweek = 7;
HoursPerDay = 24;
Message = 'Good Morning';
```

这些常量称为无类型的，因为没有给出它们的类型定义。

有类型常数是在标识符和等号之间插入了类型定义，如下例所示：

```
CONST
  DaysPerweek : Integer = 7;
  Message : Strign[20] = 'Good Morning';
  Interest : Real = 0.14;
```

在前面的无类型常数例子中，Turbo Pascal 将用串文字 'Good Morning' 替换每个标识符 Message。

类型定义

在用 Turbo Pascal 保留字 Type 开始的类型定义块中，可定义你自己的数据类型，然后可用它们说明变量。类型定义的一般形式是标识符后面跟一个等号，一个数据类型和一个分号，如下所示：

```
Type
  Paytype = (Salary,HourlyRate);
  Customer = Record
               Name : String[30];
               Age : Integer;
               Ineome : Real;
             End;
  MaxString = String[80];
  Namelist = Array[1.100] of String[30];
```

第一个数据类型 PayType 是具有两个值的枚举标量。第二个数据类型 Customer 是一包含三个域的一个记录类型，MaxString 代表一个有 80 个字符的字符串变量，NameList 是一百个分量的字符串数组，每个字符串包含 30 个字符。

变量说明

变量是命名的内存区域。一个变量说明块，由保留字 Var 开始，后跟变量标识符(变量的

名字)，一个冒号和数据类型(例如，VAR i：Integer；)。

可以使用标准 Turbo Pascal 数据类型，如布尔型，实数型，整数型和在类型段里创建的用户定义类型。变量说明的格式基本上和 Type 定义里使用的格式相同，但是标识符后面跟的是一个冒号而不是一个等号。

```
Var
  i,j,k : Integer;
  x,y,z : Real;
  BeylondLimit : Boolean;
  Ad : AdType;
  Book : Record
    Title : String[20];
    TotPages : Integer;
    Text : Array[1..10000] of String[20];
    End;
```

变量 Book 的说明使用了记录类型，使得你在一个变量里包含不止一个数据元素。

标号说明

标号用来标识一个程序里的位置，通过使用 Goto 语句和一个标号，可强制程序流从一个位置跳转到另一个位置。Turbo Pascal 把标号的范围限制在同一个过程块里。

标号说明块由保留字 Label 开始和由以逗号分隔的标识符组成，用分号结束，如下所示：

```
Label
  EndOfProgram,NextStep;
```

当在程序里使用时，标号后面跟有一个冒号(例如：EndOfProgram :)，标号后的语句在 Goto 语句转向那个标号时就被执行。下面的程序演示了一个有效的标号和 Goto 语句用法。

```
Program GotoTest;
Var
i,j: Integer;
a : Array[1..100] of Integer;
(****************************************)

Function Found : Boolean;
Label JumpOut;
Begin
For i:= To 100 Do
Begin
For j:= 1 To 100 Do
  Begin
  If a[i,j]< 0 Then
    Begin
    Found:=True;
    Goto JumpOut;
    End;
  End;
  End;
  Found:=False;
```

```
JumpOut;
End;

(*********************************)

Begin
FillChar(a,sizeof(a),0);
a[50,50]:=-1;
WriteLn(Found);
ReadLn;
End.
```

Found 函数在一个整数数组里寻找一个负值。如果找到了一个负值，函数应该返回 TRUE，否则返回 FALSE。当需要从一个嵌套的循环里跳出时，Goto 指令是一个好的选择，如果不用 Goto 指令将显著地增加子程序的复杂性。

1.1.3 代码部分

Turbo Pascal 程序的第三个主要部分是代码部分。它是程序的最大部分，包括使得程序工作的一步一步的指令。

代码段总是包含一个程序块并常常包含过程和函数。由 Turbo Pascal 保留字 Begin 和 End 限定的程序模块包含给变量赋值，创建逻辑分支和调用其它过程及函数等等一些指令。如图 1.2 所示：

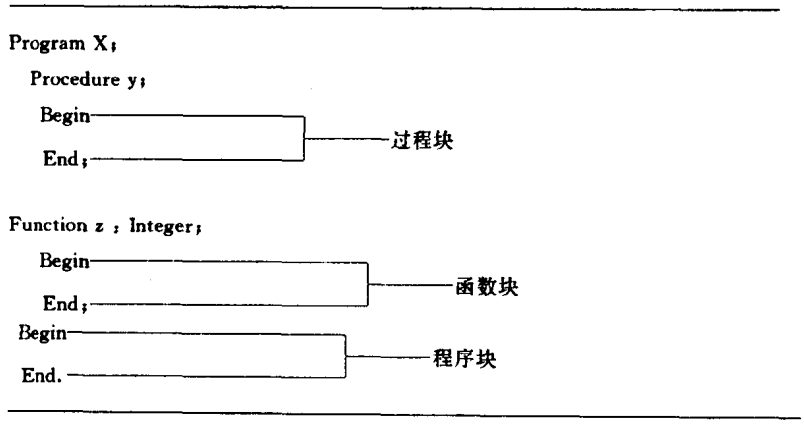

图 1.2 代码段的组织

图 1.3 的程序计算一组雇员的周工资，它演示了一个 Turbo Pascal 程序的所有基本特点。程序头和编译指令在顶部，数据段定义常数和数据类型并说明变量和标号。过程和函数在代码段定义。

```
Porgram Payroll; ←——————序头
(*$v-,I-,S-*) ←——————编译指令
```

```
Const ←————————————常量定义
  BonusRate = 0.07;
  Employees = 60;

Type ←————————————用户定义类型定义
EmployeeType = Record
  Name : String[30];
  Id : Integer;
  HourlyRate : Real;
  HoursWorked : Integer;
  GetsBonus : Boolean;
  TotalPay : Real;
  End;

MaxStr = String[255];
Var ←————————————变量说明
  Employee : Array [1..Employees] of EmployeeType;

I :Integer;
S : Macstr;
Label ←————————————标号说明

  Endofprogram;
  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)

  Procedure Calcpay(var Employee : Employeetype);
  Function Calcbonus(pay : real) : real;
begin
calcbonus :=pay + (pay * bonusrate);
end;
begin
  employee.totalpay :=employee.hoursworked *
                      employee.hourlyrate;
  if employee.getsbonus then
  employee.totalpay := calcbonus(employee.totalpay);
end;
    (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
  procedure writereport ( employee : employeetype);
  begin
  writeln('name:     ',employee.name);
  writeln('total pay:   ',employee.totalpay:0:2);
  writeln;
  end;
  (* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *)
   begin
  for i :=1 to employees do
```