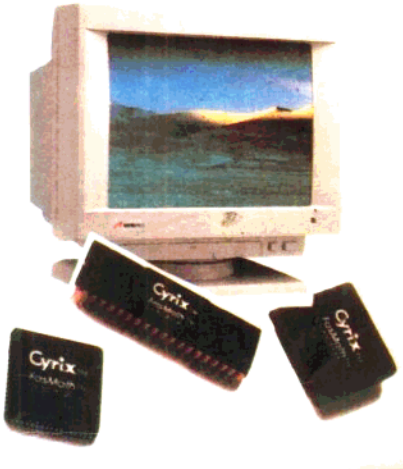
Turbo Pascal 6.0

编程技术与实例



北京希望电脑公司

Turbo Pascal 6.0
编程技术与实例

刘明 张恒 编

北京希望电脑公司

内容简介

本书系统地介绍了 Turbo Pascal 的最新版本 Turbo Pascal 6.0 的语言特性和程序设计技术和技巧。全书分为十四章，内容包括 Turbo Pascal 6.0 的基本概念、类据类型、控制结构以及其它语言特性，包括集成开发环境、工具箱、视频显示和软、硬中断高级编程，面向对象编程以及 Turbo pascl 最重要的增强－面向对象应用工具 Turbo vision 的应用实例。

本书内容丰富、完整，具有大量程序示例，深入浅出，易于掌握。既是高等院校学生的极好的计算机语言教学参考书，又是计算机系统和应用开发人员极好的指南手册。

前　言

N. Wirth 教授设计的 Pascal 语言是公认的定义严格，结构严谨，并且支持结构化程序设计方法学的一种高级语言，目前它是计算机专业的首选计算机教学语言，而且几乎国内外所有的大、中、小、微型计算机上都有它的编译器。

Turbo Pascal 是最流行的 Pascal 系统，是美国 Borland 公司推出的，可以在 PC DOS、MSODOS、CP/M80、CP/M86、OS/2 等操作系统支持的 IBM PC/XT、AT、286、386 PS/2 及其兼容机上运行，许多评论家认为 Turbo Pascal 已是事实上的微机 pascal 标准。

Turbo Pascal 有许多优良的特性，它设计精巧、用户界面友善，编译速度的运行效率高。它完全遵从 N. wirth 的标准 Pascal，也遵从 ISO 7185－1983 国际标准，而且还进行了大量的扩充，如利用计算机的硬件特性，存取绝对地址变量、内嵌汇编(机器)代码，结构常量等等，还提供了大量视频控制能力，如图形、颜色、窗口及声音等等。

另外，Turbo Pascal 具有一个集编辑、编译、连接、运行，求助，调试于一身的集成软件开发环境，支持重叠窗口、鼠标、菜单、对话框等交互手段。

Turbo Pascal 还支持可以分别编译的单元，有利于大型软件的开发。而且从 5.5 版本起，扩充了面向对象程序设计的设施，在 Turbo Pascal 6.0 中又有进一步改进和增强，尤其还提供了面向对象的应用工具 Turbo Vision，使人们可以充分表达模块化、信息隐藏、抽象代码共享等软件工程思想。

本书系统地介绍了 Turbo Pascal 的最新版本 6.0 的各种语言特点和优良特性，并通过大量程序实例阐明 Turbo Pascal 的高级编程技术和技巧，是一本难得的全面、丰富的 Turbo Pascal 6.0 程序设计教程和编程指南。

本书分为十四章，分别给出了基本概念、数据类型、语句和控制结构、集成开发环境，内存管理、文件、过程和库、工具箱、视频显示控制、软、硬中断处理、程序调试、面向对象编程等内容，最后一章还包括一个面向对象应用工具 Turbo Vision 的应用实例。本书内容详实，结构完整，深入浅出，易于掌握。

本书在编写整理的过程中，得到了多方面的帮助与支持，尤其要感谢李京同志为提高成书质量所作的大量工作。

本书的编写者虽然都是长期从事程序设计工作，并对操作系统，编译结构以及各类高级语言设施的特性有过一些研究，但由于时间短促，难免在选材和叙述上存在各种不足，欢迎读者指评指正。

目　录

第一章　Turbo Pascal 编程的基本概念

Pascal 程序设计语言是目前应用最广泛的计算机语言之一。它是由瑞士的沃斯(Nirlaus Wirth)教授于一九七一年开发出来的。其命名是为了纪念波兰数学家 Pascal。开发 Pascal 语言的目的之一是为了培养程序员，使他们能够熟练地运用结构化程序设计的技能。

在传统 Pascal 语言的基础上，Borland 进一步把它发展为丰富多彩的语言－Turbo Pascal 语言。除具有传统 Pascal 的特点外，Turbo Pascal 还有各种各样的工具集。今天 Turbo Pascal 已经成为微机世界的标准 Pascal。

§1.1　Turbo Pascal 程序的一般形式

Pascal 是一种结构化的语言，每一部分都有特定的功能。它有严格的变量说明，程序结构和控制流程的规则。结构化程序强调把程序分解为易于管理的小块，再把这些小块按照相互关系结合到一个程序 。

GOTO 语句通常会破坏程序的结构化，这条语句允许程序员在程序中跳到任何地方。这样增加了程序的灵活性，但也使程序结构变得复杂，可读性差，不易调试。Pascal 语言提供了丰富的程序控制结构语句，尽量减少 GOTO 语句的使用。这样既增加了程序的可读性，也有助于减少非结构化程序流程带来的不可预测的错误结果。

Turbo Pascal 程序的一般形式为：

- 程序头
  - 程序名
  - 编译指令
- 数据部分
  - 常量说明
  - 类型说明
  - 变量说明
  - 标号说明

  （以上为数据说明）
- 代码部分
  - 过程
  - 函数
  - 程序块

  （以上为程序体）

Pascal 程序可以只包括其中的部分，数据说明部分不是必需的，但程序体是必不可少的。程序体 以 Begin 开始，以 End 结束。程序从第一个 Begin 开始执行，到最后一个 End 语句结束，但遇到 Halt 语句除外，每个语句之后用分号与下一个语句分开，但 Begin 语句之后和 End 语句之前不用分号。下面是一个程序例子：

```
Program Progname;
    Uses CRT;
```

```
Var
  Number1,
  Number2,
  Number3;
Begin
Clrscr;
Write (' Enter a number:');
ReadLn(number1);
Write('enter another number:');
ReadLn(number2:);
number3 : = number1+number2;
WriterLn;
WirteLn('Number1+Number2=', number3:10:3);
End.
```

这个程序是从键盘分别输入两个数，然后输出这两个数的和。

§1.2 Turbo Pascal 与标准 Pascal

标准 Pascal 有许多优点，但不能满足目前商业应用的发展需要。它缺少有效的输入输出功能，没有串类型等，都不利于它的使用。Turbo Pascal 是在标准 Pascal 上更新了的语言，它既有标准 Pascal 的逻辑结构，又有一个扩展工具集。

Pascal 通常称为强类型语言。不同类型的变量不能混淆。在赋值语句中，左右两边的变量类型必须匹配。例如，变量 i 定义为整型变量。则下面的语句为非法的

```
i:=1.0+2;
```

因为值 1.0 是实型数，不能用于整型数的赋值。这不符合强类型语言的宗旨。强类型规则有助于避免程序设计中的错误。Turbo Pascal 中的强类型要求不如标准 Pascal 严格。但仍需遵守下列规则：

* 实型数不能直接用于整型变量或字节的赋值语句，必须先用 trunc 和 Round 标准函数把它们转换为整型数。
* 只有在长度和类型都一致时，一个数组才可对另外一个数组赋值。
* 传递给过程或函数的变量类型必须与过程说明的类型一致。但对于串类型可以用{$V-}编译指令取消。

一般来讲，强类型有很多好处。但有时侯不使用强类型，问题更好处理。使用类型的强制转换可以解决这个问题。类型的强制转换就是把一种类型的变量暂时作为另外一种类型处理。例如，有时可能需要求出字符的 ASCII 值。直接把一个字符的值赋给一个整型变量是不允许的。但用类型强制转换可以做到。如

```
i:=Integer(c);
```

其中 i 是整型变量，C 是字符型变量，这条语句把 C 的二进制值作为整型数赋给 i。指针型变量被解释为整型变量和一个串变量。类型强制转换技术对指针特别有用，这样它可以用于任何类型的数据。

§1.3 程序结构

Turbo Pascal 程序主要由三部分组成：程序头，数据部分和代码部分。各部分有其特定的功能。

§1.3.1 程序头与编译指令

Turbo Pascal 程序的前二行一般为程序名和编译指令。程序名也可以没有，但是为了归档方便最好使用程序名。程序名标识了程序的名字以及在程序中是否使用了输入输出。典型的程序头如下：

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

程序名后可以带一个参数表，但对于 Turbo Pascal 程序不是必需的。

Turbo Pascal 编译器可带许多选择项，指导编译器如何工作。这些选择项称作编译指令。编译指令在 Turbo Pascal 程序中起着重要作用。它们控制着各种错误检查和输入/输出控制。要充分发挥 Turbo Pascal 的能力，就必须懂得如何使用选择项。编译指令大致分为三类：开关指令，参数指令，条件指令。

开关指令

开关指令打开或关闭 Turbo Pascal 的某项特殊功能。开关指令由单个字母标识，大写字母与小写字母相同。如 S 指令控制堆栈检查，R 指令设置范围检查等等。开关指令的格式为一个 $ 号后跟一个编译指令及一个正号(允许)，或负号(禁止)，这些指令用两种注释定界符限定。这两种定界符是带 * 号的圆括号或花括号。如：

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

等都是有效的编译指令。前两个语句是相同的，都是禁止输入

输出错误检查。第三个例子说明了四个编译指令：允许堆栈溢出检查S+，禁止变量字符检查 V-，允许范围检查 r+，及按字对齐数据 a+。在这个例子中，只在第一个指令前有 $ 号。

编译指令的设置有两种方式。最简单的方式是由 Options

Compiler 菜单设置。这样设置的指令是所有程序和程序块的缺省值，所有未在源代码行说明的编译指令，都使用 Options

compiler 设置的值。

在程序头上说明的开关指令是全局性的，它们影响整个程序的编译。它们必须在第一个 Uses，常量，标号，类型，过程，函数或 Begin 之前，否则为局部性的。局部指令可以出现在程序中的任何地方，它们只影响以后的程序。

{A} 数据对齐

当数据对齐指令有效时({$A+})，Turbo Pascal 保证每个大于 1 字节长的变量和有类型常数都在偶地址开始。这样对齐后，8086 系列微处理机可以快速访问内存，但也增加了存贮数据所需的内存量。因为数据在字边界对齐后，有些字节会成为"死"字节。若程序使用的内存量必须考虑时，可禁止这个指令{$A-}。

{B} 布尔变量求值

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。短路布尔变量求值时，只要能确定整个表达式的结果时，就不再继续运算。在很多情况下，这可以大大加快程序的运行。例如：

(a<b)And(b>c)

表达式，使 a 大于 b 时，不论 b 是否大于 c，其结果都是假。这时短路布尔变量求值即结果。便用{$B-}进行短路布尔变量求值，使用{$B+}，进行完全布尔变量求值。

{D}调试信息

使用允许调试信息指令{$D+}，Turbo Pascal 将产生可执行指令在源文件中定位所需要的信息。使用这些信息可以逐行检查一个程序，或者在出现运行错误时确定错误的位置。这些调试信息加在。TPU 文件的后面，使文件增大，但不会影响可执行代码的速度。若使用禁止指令{$D-}，则不能单步执行程序，但可稍稍缩短编译时间并节省一些磁盘空间。

E 仿真

有的计算机带有 8087 数字协处理器，另外一些计算机则没有。如果一个程序需要 8087 的数字精度，并且这个程序有可能在不带 8087 协处理器的计算机上运行。当允许仿真{$E+}时，Turbo Pascal 检查是否安装了 8087 芯片。如果有，程序将使用协处理器。如果没有，程序将用主处理器实现 8087 运算。自然如果没有 8087 协处理器，计算将用较长时间，但是程序可在任何一种机器上运行。除非需要特别精确的数学结果，最好使用禁止仿真指令{$E-}。

F 强制运程调用

Turbo Pascal 支持多个代码段，每个单元一个代码段，主程序另有一个代码段。在一个单元或程序中调用函数或过程为近程调用，不需要改变代码段。一个单元中的语句调用另外一个单元中的过程为远程调用。近程调用要做的工作较少。远程调用牵涉到不止一个代码段，做的工作较多，执行速度也慢。Turbo Pascal 可以判断应该用近程调用还是用远程调用。有时侯在可以做近程调用的地方，需要强制一个过程为远程调用。必须强制使用远程调用的环境是非同寻常的。必要时，可使用允许强制远程调用指令{$F+}，使一个过程为远程调用。

I 输入/输出检查

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。它们使看似正常运行的程序产生不可预测的结果。{$I+}语句就是用于检查程序中的 I/O 错误。使用这条指令后，出现 I/O 错误将产生运行错误并停止执行程序。{$I+}指令不是处理 I/O 错误的最好方法。更有效的方法是使用{$I-}，用户自己发现 I/O 错误。

L 局部符号信息

局部符号信息是指局部于一个单元或过程的变量和常量信息，Turbo Psacal 一般不保存这些符号的信息，因而在调试对话中不能看到或者改变它们的值。{$L+}语句使 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。L 指令与调试信息指令 D 共同使用时，结果如下：当禁止调试信息时，L 编译指令不起作用，Turbo Pascal 不保存任何调试信息。当允许 D 指令，禁止 L 指令，即({#D+，L+})，Turbo Pascal 保存所有符号的调试信息。

N 数值处理

Turbo Pascal 处理浮点运算有两种方式：正常方式和 8087 方式。正常方式为 6 字节数据，不使用 8087 协处理器。8087 方式提供另外四种浮点数据类型，使用 8087 协处理器。{$N

—} 语句选择正常方式，{$N+} 选择 8087 方式。N 指令可以与 E 指全共同使用。当两个指令都有效时，即使没有 8087 协处理器，程序也可以用 8087 方式处理，即使用{$N+，E+}时，若装了 8087 芯片，程序就用它进行浮点操作，否则使用仿真程序。仿真程序具有相同的精度，但速度较慢。若禁止仿真指令{$N+，E—}，程序只能在装有 8087 协处理器的计算机上运行。

O　覆盖代码产生

在一个覆盖文件中引用一个单元时，必须使用{$O+}指令。它告诉 Turbo Pascal 产生管理这个覆盖单元所需的代码。这时不是必须覆盖这个单元，只是使覆盖成为可能。反之，除非 O 指令有效，否则不能覆盖这个单元。

R　范围检查

Turbo Pascal 的大部分数据类型都有限制。一字节数据的最大值为 255。5 个元素的数组不能有 6 个元素。定义为十个字符的字符串不能有十一个字符。若超出了限制会产生一个范围错误。当使用{$R+}指令后，Turbo Pascal 产生代码检查所有数组下标和赋值是否出界。发现范围错误后产生一个运行错误，停止执行程序。如果使用{$R—}，所有的出界赋值和下标都不报告，其结果可能是灾难性的。

在一个好的程序中不应出现范围错误，但在程序开发的早期却是很难避免的。所以在开发程序阶段最好使用允许范围检查指令，在程序基本完成时禁止这一指令。在允许 R 指令时，编译后的程序长度会明显加大，执行也慢。使用时应该注意。

S　堆栈溢出检查

程序调用一个过程或者函数时，是从局部变量堆栈中分配内存，{$S+} 指令让 Turbo Pascal 加上必要的代码，确保堆栈有足够的空间容纳这些变量。内存不够时，程序将停止并产生一个运行错误。若使用{$S—}指令，当堆栈内存没有了时，计算机将崩溃。堆栈检查要占用时间，增加可执行代码，因此只在开发调试阶段才使用堆栈检查指令。

V　字符串变量检查

当使用{$V+}时，Turbo Pascal 对传递给过程和函数的字符串参数进行严格的检查。若不进行检查，参数类型不匹配时，有可能产生不可预测的结果，如毁坏内存。进行字符串变量检查能发现隐蔽的字符串错误。禁止字符串变量检查要保证程序不会产生意想不到的破坏。

参数指令

参数指令与开关指令不同，没有明显的接通

关闭状态。这些指令表明编译时使用的文件名和程序所需内存的大小。

I　包含文件

在编译一个源程序时，I 指令使另外一个文件成为这个源程序的一部分。如果包含的文件没有指定扩展名，Turbo Pascal 假定为. PAS. 当一个程序太大，不能一次放到 Turbo Pascal 编辑器中，或者希望通过改变包含文件来修改程序的一部分时，可以使用包含文件指令。例如：

```
INC. PAS 文件
Procedure    Proc1
Begin
...
End
```

主程序文件

```
Program Main;
{$I Inc}          包含文件指令
Begin
Proc1;            执行包含文件
...
End
```

在这个例文中，编译程序把 Inc.PAS 作为主程序的一部分编译。

L　链接目标文件

使用 L 指令可以把汇编语言目标程序与 Pascal 程序链接起来。这样可以在 Pascal 程序中使用汇编语言编写的子程序。这条指令是在 L 后面接目标程序文件名。

M　内存分配

M 指令指示程序中堆栈和数组各自所用的内存量，这条指令的格式是在 M 后接三个数字，分别用逗号隔开。第一个表示堆栈所用的字节，第二和第三个表示数组的最小量和最大量。例如，{$M 10000，300，5000} 表示分配给堆栈 10000 字节，分配给数组的最大字节为 5000，最小字节为 300。堆栈分配的内存量在 1024 到 65520 之间。数组分配的内存量为 0 到 655360 之间。

O　覆盖单元名

O 指令的格式是在 O 后面接一个单元名。Turbo Pascal 将把这个单元包含在盖文件中。覆盖单元指令 0 必须用在 Uses 字句的后面。用 0 指令命名的单元必须用 {$+}编译指令编译，否则不允许覆盖。

条件编译

使用条件编译，可以在一个文件中保留程序的不同版本。只要改变某些定义，就可以指示 Turbo Pascal 编译某些代码段，而把另外一些段隐藏起来，条件编译的关键是条件符号，即定义一个符号，控制条件编译的进程。下面的例子怎样进行条件编译。

```
Program Condtion;
{$DEFINE CON}                    (*定义 CON*)
{$IFDEF CON}                     (*如果 CON 已定义*)
{$R+,S+}
{$ELSE}                          (*如果 CON 未定义*)
{$R-,S-}
{$ENDIF}                         (*条件编译结束*)?
Begin
{$UNDEFCON}                      (*解除 CON 定义*)
{$IFDEF CON}                     (*如果 CON 已定义*)
WriteLn('CON DEFINED');
```

```
{$ELSE}                                    (*如果 CON 未定义*)
WriteLn('CON NOT DEFINED');
{$ENDIF}                                   (*条件编译结束*)
```

程序在开始时定义了符号 CON。在第一个条件编译处，由于符号 CON 已经定义，则 R 指令和 S 指令都有效，而 R 指令和 S 指令禁止的语句被隐藏起来。在关键字 Begin 之后，CON 的定义解除，这时仅编译 WriteLn('CON NOT DEFINED')；一行，而 WrteLn('CON DEFINDE')；不进行编译使用条件编译指令后。可以最大发挥 Turbo Pascal 的效率。下面介绍 Turbo Pascal 提供的条件编译指令。

DEFINE　定义

这条指令定义一个条件符号。所有依赖被定义符号的代码都将被编译，但只限于出现本指令的文件中的代码。例如，在一个主源程序文件，一个包含文件和一个单元文件中。在每个文件中都使用条件编译指令。如果主源文件中使用了 DEFINE 指令，在包含文件和单元文件中不使用，则只影响主源文件，其它文件中不受主文件中 DEFINE 指令的影响。定义全局编译符号只能用 Options

Compiler 菜单上的条件定义功能。

UNDEF　解除定义

本指令与 DEFINE 指令相反。一个符号用 UNDEF 指令解除定义后，依赖于这个符号的代码都不编译。

IFEDF

如果条件符号已定义，本指令后面的代码将被编译。

IFNDEF

本指令与 IFDEF 相反，如果条件符号未定义，后面的代码将被编译。

IFOPT

本指令使得 Turbo Pascal 可以根据另外的编译指令控制编译。如{IFOPTR+}，使编译程序在 R 指令有效时编译代码。

ELSE

作为 IFDEF ，IFNDEF，和 IFOPT 的条件分枝，当条件为假时，编译本指令后面的代码。

ENDIF

本指令表示条件编译的结束，其后的任何代码都不依赖于条件符号。

§1.3.2　数据部分

在 Turbo Pascal 程序中，程序头和全局编译指令后是数据部分，包括全局的变量，常量，类型和标号。局部变量在过程和函数中说明，但它们遵守同样的基本规则。

常量说明

常量在以 CONST 开始的块中定义。在 Turbo Pascal 中有两种常量：无类型的和有类型的。无类型常量定义的格式为

CONST

<标识符>=<常量>;

常量可以是一个数值或字符及字符串。在标识符和等号之间加上类型定义，即为有类型常量的定义。下面是几个常量定义的例子：

```
CONST
WEEK=7;
Message='Hello! ';
TAX:Real=25.00;
Hours : Integer=24;
MONTH:string[15]='september';
```

例子中前两个是无类型常量，后三个是有类型常量。对于一些在整个程序中不改变的值用常量标识符表示，这样简化了程序，也使程序易读，便于维护。

无类型常量和有类型常量在 Turbo Pascal 中是不同的。在编译程序时，每个无类型常量标识符都要用其值替换；而有类型常量标识符仅在数据段中定义一次，程序中使用有类型常量都是使用其副本。无类型常量要占用较多的内存。而有类型常量更象是一个已经初始化的变量，因此可以改变其值。如果要确保一个常量在程序中不变，应使用无类型常量。

类型说明

在 Turbo Pascal 中可以定义自己的数据类型，然后用它们说明变量。类型定义在以 TYPE 开始的块中定义。类型定义的一般形式是。如：

```
type  <标识符>=<数据类型>;
Type    payday=(salary, Hours);
  Employee=Record
           Name:String[20];
           Number:In teger;
           sal(ary:Real)
           End;
  Maxstring=string[80];
  NameList=Array[1..100]of string[20];
```

用户自己定义数据类型是 Pascal 的一大优点。在后面要做进一步的讨论。

变量说明

变量是命名的内存区域。变量的说明在以 Var 开始的块中，变量说明的格式为 Var

<标识符>:数据类型;

类据类型可以是 Turbo Pascal 标准数据类型，如布尔型，实型和整型，或在类型段中建立的用户定义的类型。例如：

```
Var
  i, j, R:Integer;
  x, y, z:Real;
  a, b, c:Boolean;
  Ad:payday;
```

标号说明

标号标识程序中的一个位置，与 GOTO 语句共同使用，可以把控制从程序流中的一个位置强制跳到另外一个位置。对结构化语言来讲，使用 GOTO 语句不是一种好的方法。因此，Turbo Pascal 把标号的范围限制在同一过程块中。使用标号和 GOTO 语句还是有好处的，它可以增加程序的灵活性。特别是当需要从嵌套较深的循环中跳出时，使用 GOTO 语句不失为一种好的方法，否则将明显增加程序的复杂性。其格式为

```
Label
  <标识符>,…标识符<>;
```

§1.3.3 代码部分

代码部分是 Turbo Pascal 程序中最大的部分，由使程序工作的指令组成。代码段包括一个程序块，由 Begin 开始，以 End 结束。在 Turbo Pascal 中，首先执行的程序模块是在程序的尾部。例如：

```
program exampla;
  procedure Exa1;
  Begin                          ┐
  End;                           ┘   过程块
  FuncTion Exa2: Integer;
  Begin                          ┐
  End;                           ┘   函数块
Begin                            ┐
End                              ┘   程序块
```

Pascal 被称为模块结构化语言，它的每一条语句都属于一个代码模块。模块可以嵌套，即在一个过程中嵌套另一个过程。简单的程序可以只有一个模块－程序模块。在一个过程中嵌套的过程对另一个过程中嵌套的过程是不可见的。这样在不同的过程中可以嵌套两个过程名相同的过程而不会行起混乱。过程名的作用域遵守以下规则：

* 一个程序可调用其所在块或嵌套于这个块中的任何子块中的过程。
* 当程序在较高层子块中说明了另一个具有相同名字的过程时，上一条规则无效。

在其它一些语言中，过程说明的顺序没有关系。但在 Turbo Pascal 中，一个过程未加说明之前不能调用(向前说明例外)。在 Turbo Pascal 程序中，最后到达程序块。程序块可能只有几个过程调用。因此程序可以看作一个连续进化的过程。程序的优先级是根据这样一个逻辑：由简单构造复杂。这虽然给程序中的过程规定了顺序，但有时可能要调用一个未说明的过程，这可以通过向前引用说明来实现。其格式为：

Procedure<过程名>; FORWARD;

FORWARD 是关键字。这里的过程只是一个过程头。整个过程在后面说明。这个指令通知编译器有一个前面未说明的过程。下面是一个向前引用过程的例子。

```
Program Prog1;
Var
  I: Integer;
  procedure StepB(i:Integer); FORWARD;
  Procedure StepA(i:Inleger);
```