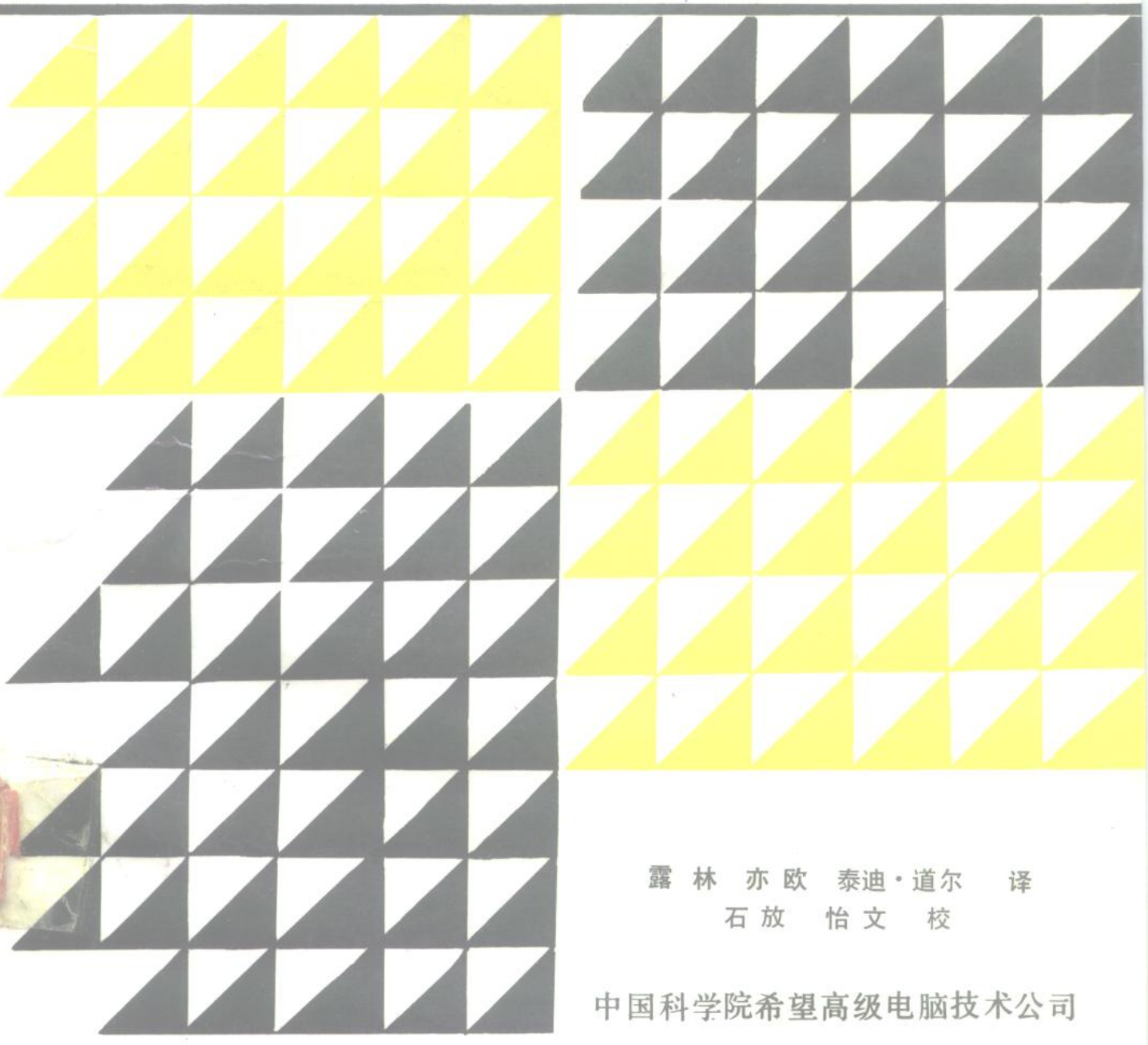
5.0版

TURBO PASCAL
高级编程指南



露林 亦欧 泰迪·道尔 译
石放 怡文 校

中国科学院希望高级电脑技术公司

Turbo Pascal 5.0

高级编程指南

露林 亦欧 泰迪·道尔 译
石放 怡文 校

中国科学院希望高级电脑技术公司

一九九〇年五月

前　言

<<Turbo Pascal 5.0 5.5 高级编程指南>>是<<Turbo Pascal 5.0 用户手册>>、<<Turbo Pascal 5.0 参考手册>>的后续。<<Turbo Pascal 5.0 用户手册>>、<<Turbo Pascal 5.0 参考手册>>是 Turbo Pascal 系列学习的基础，着重介绍如何使用集成环境，特别是功能独特且强大的内部调试，介绍解决内存紧张问题的透明的覆盖功能；介绍丰富的标准子程序的用法；论述基本的编程方法与调试技巧。而<<Turbo Pascal 5.0 5.5 高级编程指南>>为用户在上述基础上取得进一步提高提供有效时机，它讨论内存驻留、传递过程类型参数、与汇编的接口、程序段前缀(PSP)结构、未公布的 DOS 和 BIOS 服务；介绍用户界面的设计方法、窗口菜单的新概念等高级专题，还涉及程序设计的单元化方法、通用工具箱的设计与使用。丰富的内容拓宽您的软件开发技能，精巧的例程使您快速领会精髓。先进的知识、高级的技巧既实用又易于学习。

在编译过程中，曲建平同志作了大量辅助工作，给予了大力协助。张志辉，翁敬农，颜蓉等同志也做了不少的工作，在此一并表示谢意。

由于时间短促、加上水平有限，错误在所难免，欢迎指正。

译者

1990.5.

《Turbo Pascal 5.0高级编程指南》简介

Stephen K.O'Brien

《Turbo Pascal 5.0高级编程指南》充分阐述Turbo Pascal 5.0的性能技术，包括：如何编写安全的常驻内存程序，如何传递过程类型参数，如何与汇编接口等等。讨论Turbo Pascal 5.0功能强大的工具箱。在上述各专题中，例举了丰富、精致的样本程序以分析说明概念的使用方式，便于用户快速理解掌握。

《Turbo Pascal 5.0高级编程指南》的组成：

第一部分　Turbo Pascal 5.0的重要特性

涉及单元的使用，高级过程调用、条件编程；提供显示和键盘控制的示例单元，论述了开发用户界面时窗口，菜单的使用，以及命令行的分析方法。

第二部分　提供帮助提高编程技能的专题讨论。

介绍了扩展内存的利用方法，BIOS和DOS中断服务的调用；完整地论述了编写安全的常驻内存程序的诀窍。更重要的是提供了使用鼠标器的示例单元和程序，揭示了前所未有的程序前缀段（PSP）的奥秘，使得Turbo Pascal 5.0覆盖了目前Pascal应用的所有方向。在这部分还别致地说明了Turbo Pascal 5.0与汇编的接口，说明了其重要的应用：回归分析。

第三部分　Turbo Pascal 5.0通用工具箱

数据库工具箱能对文件的任意数据类型进行排序；通过创建股票价格表的例程说明图型工具箱的功能和使用方法；通过使用二元编辑器创建数据库，介绍编辑工具箱；通过创建一个完整的回归分析实例讨论数字处理工具箱的使用。

第四部分 Turbo Pascal 5.0用户界面

讨论Turbo Pascal 5.0及其以上版本，阐述了新的用户界面，说明如何使用集成调试器，如何创建和使用覆盖文件。

本书以简洁的语言、丰富的内容，向用户展示了Turbo pascal 5.0强大功用。既有概念的说明又有精确的例程，确是Turbo Pascal 5.0用户的良好指南。

目 录

第三部分　使用Ｂｏｒｌａｎｄ工具箱

第一部分
强有力的工具

第一章　编程指南

Turbo Pascal 是一个功能很强的编译器，能控制个人计算机的各个方面。尽管如此，程序员还是希望有更高的速度，更好的用户界面，更多的灵活性等。

在这一章里，介绍了如何使用一些工具和技术，为编程提供方便。其中的一些问题在《Turbo Pascal 用户手册》中没有讨论，或者没有深入地讲解。而这里通过实例，讨论了这些重要的技术，并介绍具体的编程方法。

§1.1 使用单元

单元是能与 Turbo Pascal 程序分开来编译的一组 Turbo Pascal 过程和函数。因为单元是单独编译的，所以使用单元的程序编译得快。不仅如此，一个单独的单元能够应用于许多不同的程序。这样加强了程序间的一致性，从而使程序的维护变得容易。

有效地使用单元，可能是 Turbo Pascal 编程中最重要的一个方面。但是用户如果刚接触 Turbo Pascal，则可能会发现单元的概念是很难掌握的。

§1.1.1 单元结构

一个单元由接口部分和实现部分组成。在接口部分，可以进行数据类型和变量说明、引用其他单元、定义过程和函数的头部。实现部分则由接口部分定义的过程和函数相应的体、局部变量以及局部过程组成。

§1.1.1.1 接口部分

正象它的名称所表达的含义一样，接口部分是单元与其他单元或程序“接口”的部分。换句话说，一个程序如果使用了一个单元，那么它就能访问该单元的接口部分所定义的所有变量、数据类型和过程。

接口部分不包含完整的过程和函数，仅是他们的头部。而过程或函数体在实现部分定义。在程序中使用一个单元只需知道怎样调用单元中的过程，而不需知道过程是怎样实现的。过程头和它的可执行代码的分离增强了程序的模块化，提高了编程处理效率，因而这种分离很重要。

下面是一个单元接口部分的例子：

```
Unit UnitX;       (* Units begin with a Unit Name. *)
Interface         (* Declares beginning of Interface section. *)
Uses CRT;         (* Uses the Crt Unit. *)
Var
  Xi:Integer;     (* Declares a global variable Xi as Integer *)
Procedure CalcXI(Var Xparam:Integer);
                  (* Declares a procedure named CalcXi which    *)
                  (* accepts an integer parameter.              *)
```

在单元中，首先定义单元名字。在这个例子中单元名字是 UnitX。

第二个语句是一个关键字 Interface，它表明接口部分的开始。随后有三个语句。第一个语句，“Uses CRT;”，表示该单元要使用 Turbo Pascal 的 Crt 单元（Crt 单元是 Turbo Pascal 标准库的一部分）。Uses 语句允许 UnitX 调用 Crt 单元中的任何一个过程和函数。下一个语句，“Var Xi: Integer;”声明了一个叫 Xi 的整型变量。由于 Xi 在接口部分中声明，因此它可被使用 UnitX 的任何单元或程序引用。而在实现部分中声明的变量，局部于该单元，因此不能被其他单元或程序存取。

在该例子的接口部分中，声明了一个叫 CalcXi 的过程，该过程接收一个整型参数 Xparam。请注意这里只声明 CalXi 的头部，在实现部分中定义过程体。

§1.1.1.2 实现部分

实现部分是一个单元中的实际工作部分，含有接口部分所定义 过程的可执行语句。接口部分中命名的过程必须定义相应的过程体。　　在实现部分也可以定义在接口部分中不出现的过程和变量，但是它们将只在这个单元内有效。换句话说，任何只在实现部分中声明的对象是局部的，在单元外面无法存取。

因为在实现部分中声明的一切对象在作用域上是局部的，所以实现部分的改变对其他单元和程序来讲是“不可见的”。因此，修改一个单元的实现部分，并不需要重新编译其他使用该单元的单元—只需重新编译被改变了的单元和使用这些单元的程序。然而，如果对接口部分做了修改，所有使用该单元的单元和程序必须重新编译，甚至可能要修改。

下面例子说明 UnitX 的实现部分的一般格式:

```
Implementation   (* Declares Implementation Section *)

Function DoCalc(x,y : Integer) : Integer;
(************************************************)
(* This function is local to UnitX and cannot be *)
(* accessed outside the unit.                    *)
(************************************************)
begin
DoCalc := x*(y div 2);
end;

Procedure CalcXI(Var Xparam : Integer);
(************************************************)
(* This defines the operation of the procedure   *)
(* declared in the Interface section.            *)
(************************************************)
begin
Xparam := DoCalc(Xparam,Xparam+1);
```

```
end;

End.    (* End of the Unit *)
```

UnitX 的实现部分包含四个部分。首先，语句 Implementation 定义实现部分的开始。

在实现部分中声明的第一项是函数 Docalc。Docalc 在接口部分中没有声明，因此它的作用域是单元 UnitX。换言之，使用 UnitX 的程序不能调用 Docalc。因此其他单元或程序可以定义同名 Docalc 函数而不会冲突。

下一项是过程 CalcXi 的执行代码，CalcXi 在接口部分中有声明。请注意：Turbo Pascal 要求实现部分中的过程头与接口部分中的过程头准确匹配。如果过程头不匹配，该单元编译不能通过。但可在实现部分省去参数表—Turbo Pascal 将接口部分中的参数表作为缺省参数。因而，如此声明实现部分是合法的：

```
Procedure CalcXI; (* Parameter List is optional *)
```

单元用保留字 End 作结束语句。注意单元结束用句号而不是分号。

§ 1.1.2 在程序中使用单元

若程序要使用单元，则必须有一个带有该单元名字的 Uses 语句。下面的例程用来说明使用的方法：

```
Program TestUnitX;

Uses UnitX;       (* Provides access to UnitX *)

Begin Xi := 1;          (* Uses variable defined in UnitX *)
CalcXi(Xi);       (* Uses procedure defined in UnitX *)
WriteLn(Xi);
End.
```

只要简单地声明一下"Uses Unitx; "，程序就能使用 Unitx 接口部分中所声明的所有变量、数据类型和过程。

注意：这个程序首先给 Xi 赋一个初值，Xi 是 Unitx 中定义的全局变量；然后调用过程 Calcxi，它也是在 Unitx 中定义的。用这种方式引用单元使单元的内部操作同程序本身分离开来，从而简化了程序。

§ 1.1.3 值初始化

可以在程序一开始就对某些变量初始化。在刚给出的例程中，Xi 通过语句 Xi:=1 显式初始化。另外一种方法是在单元内部初始化。如要在 Unitx 中对 Xi 初始化，可以将实现部分的结尾修改成：

```
Begin (* Begin Unit initialization. *)
Xi :=1;
End.     (* End of the Unit *)
```

修改后的实现部分含有一个 **Begin** 语句和一个 **End** 语句。当使用 **Unitx** 的程序开始执行时，**Turbo Pascal** 先执行在该单元 **Begin** 语句和 **End** 语句之间出现的语句。把这些过程隐藏在单元内部是为确保初始化总能兑现。值得注意的是单元初始化代码只在程序开始时执行一次。

在前面的例子中只有一个执行语句"**Xi:=1;**"。单元的初始化部分可以含有任意多的语句—甚至包含完整的程序!

§1.1.4 域冲突

前面已提及，单元接口部分的变量可以被其他单元或程序引用。但如果使用该单元的程序声明了一个同名变量或过程，就会发生冲突。这种情况下，程序引用其本身代码中声明的变量或过程，而不是单元中的同名对象。

为了说明如何发生冲突，试分析下面的单元：

```
Unit Conflict;

Interface

Var
  i:Integer;

Procedure WriteI;

Implementation

Procedure WriteI;
Begin
WriteLn('i=',i);
End;

Begin
i:=0;
End.
```

名为 **Conflict** 的单元声明了一个全局整型变量 **I**，还声明了过程 **WriteI**，用来显示 **I** 的值。该单元将 **I** 值初始化为 0。　　下面的例程使用单元 **Conflict**。可以看到，该程序也声

明了一个整型变量 I，并且在程序开始时已被赋值为 100。

```
Program TestConflict;
Uses Conflict;
Var
  i:Integer;
Begin
i := 100;
WriteI;
End.
```

当这个程序调用 WriteI 时，会显示什么值呢？答案是：0。程序中的语句 I:=100 访问的是该程序内部的变量，而非单元的变量。而过程 WriteI 使用单元内部的变量，因此显示的值是单元内部变量 I 的值 0，而不是程序中变量 I 的值 100。

这种冲突会在程序中产生难以捉摸的错误。在编程时必须注意消除任何此类可能的冲突。

§1.1.5 单元名和文件名

通常，单元名与该单元所在的文件名是相同的（例如，名为 Unitx 的单元应该存放在名为 UNITX 的文件中）。当单元与文件名字不匹配时，必须在 Uses 语句中使用如下所示的{$U}编译指令：

```
Uses {$U Filename) UnitName;
```

{$U}编译指令告诉 Turbo Pascal 在文件 FILENAME 中寻找单元 UnitName。

§1.2 高级过程调用

在 Turbo Pascal 中使用过程和函数是轻而易举的事。先定义，后通过名字调用。下面介绍的两项高级技术—将过程做为参数和间接调用过程—增加了调用过程和函数的方法。

§1.2.1 传递过程参数

作为 Turbo Pascal 程序员，也许很习惯于将参数值传递给过程。参数是过程用于产生某种预期结果的数据值。然而把过程做为参数行不行呢？在 C 语言中这是一个经常使用的强有力的方法，遗憾的是在 Turbo Pascal 中不直接支持过程参数。将过程作为参数不仅能增强程序的灵活性而且能减少完成某些任务所需的代码量。

§1.2.2 过程指针

当将一个过程当作参数传递时，实际上是传了一个包含该过程地址的指针。在 Turbo

Pascal 中用@操作符来取得过程地址。例如存取过程 Proc1 的地址，可这样写：

@Proc1

要传递一个过程参数，只要声明一个类型指针。难点在于怎样用这个地址调用相应的过程。可通过 in-line 代码使用地址参数调用过程来解决。下面是例程 CallProc：

```
{$F+} Procedure CallProc(Sub:Pointer); {$F-}
Begin
inline($FF/$5E/$06);
End;
```

该过程将一个参数 sub 作为类型指针。sub 参数包含一个地址，指向要调用的过程。CallProc 过程体由一条 in-line 语句实施长调用。　　注意：为 CallProc 设置{$F+}编译指令，以确保 Turbo Pascal 产生带有段和偏移量的地址。在 Turbo Pascal 中为程序和每一个单元提供了独立的代码段。因而被调用的过程可能会在另外一个代码段中，所以需用长调用。忽略定义 CallProc 为长调用很可能会导致系统崩溃。

下面的程序可说明怎样使用 CallProc：

```
Program ProcCallDemo;
Uses CRT;

{$F+} Procedure CallProc(Sub:pointer); {$F-}
begin
inline($FF/$5E/$06);
end;

Procedure Proc1;
begin
Writeln('Executing Proc 1');
end;

Procedure Proc2;
begin
Writeln('Executing Proc 2');
end;

begin
ClrScr;
CallProc(@Proc1);
CallProc(@Proc2);
end.
```

程序通过将 Proc1 和 Proc2 的地址传给 CallProc 调用这两个过程。虽然直接调用 Proc1 和 Proc2 同样能完成，但在一些特殊的应用场合传送过程参数才会显出真正价值。例如，用户声明一个指针数组，其每一个元素指向一个不同的过程，这样就可以把每一个过程当作数组的一个元素来访问。该方法可使许多编程问题简化。

§ 1.2.3 间接过程调用

前面介绍的传递过程参数的方法虽是一个强有力的技术，但同时又有一个缺点—不能传递带有参数的函数或过程。要调用带有参数的函数或过程需要采用一种稍有区别的方法。例如，可如下编写比较两个字符串，判断一个串是大于、小于还是等于另一个，并返回一个整型值的例程:

```
Function CompareString(x,y:String): Integer;
Begin
If x < y Then
  CompareString := -1
Else If x > y Then
  CompareString :=1
Else
  CompareString :=0;
End;
```

函数 CompareString 接受两个串变量 X 和 Y，进行比较，最后返回一个表示它们在字母表中相对位置的整数。间接调用这个函数，要做两件事。首先要声明一个全局指针变量，指向函数 CompareString。其次，要声明一个有相同参数表的函数，以便用 in-line 代码实施间接调用:

```
Var
  ProcPtr : Pointer; (* This must be a global variable. *)

{$F+}Function Compare(x,y:String):Integer;{$F-}
InLine($FF/$1E/ProcPtr);
```

同函数 CompareString 一样，Compare 接受两个串参数。不同之处在于: Compare 将控制转到 ProcPtr 所指向地址的函数。在调用 Compare 之前，必须把 CompareString 的地址存放在 ProcPtr 中。

下面例程说明如何实现间接过程调用。注意: Compare 和 CompareString 都要将{$F+}编译指令激活，以保证 Turbo Pascal 产生必要的远程调用。

```
Program TestCall2;
```

```
Var
  ProcPtr:Pointer;

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

{$F+}Function Compare(x ,y : String) : Integer;{$F-}
InLine($FF/$1E/ProcPtr);

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

{$F+}Function CompareString(x ,y : String) : Integer;{$F-}
begin
If x<y then
  CompareString := -1
else if x>y then
  CompareString := 1
else
  CompareString := 0;
end;

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

begin
ProcPtr := @CompareString;
WriteLn(Compare('ABC','DEF'));
end.
```

§1.2.4 间接调用与提前声明

在 Turbo Pascal 中提前声明允许先定义一个子程序的头，然后再定义实际的子程序代码。使用间接程序调用不仅具有同样的功能而且具有更大的灵活性。例如，一个程序从文件中读取数据，然后处理，用最基本的方法可编成：

```
Procedure FileProcessor;
  begin
    ReadFile;
    ProcessFile;
  End;
```

FileProcessor 过程假设 ReadFile 已预先声明。即或者在 FileProcessor 之前定义 ReadFile，或者提前声明 ReadFile 而后在程序中定义过程体。这两种情形下，程序都能正确