5.2 用户指令参数菜单

Import（输入）按钮可以输入原先开发的子程序。这些子程序里一般会有功能说明（如图 5-4、图 5-5 所示）。

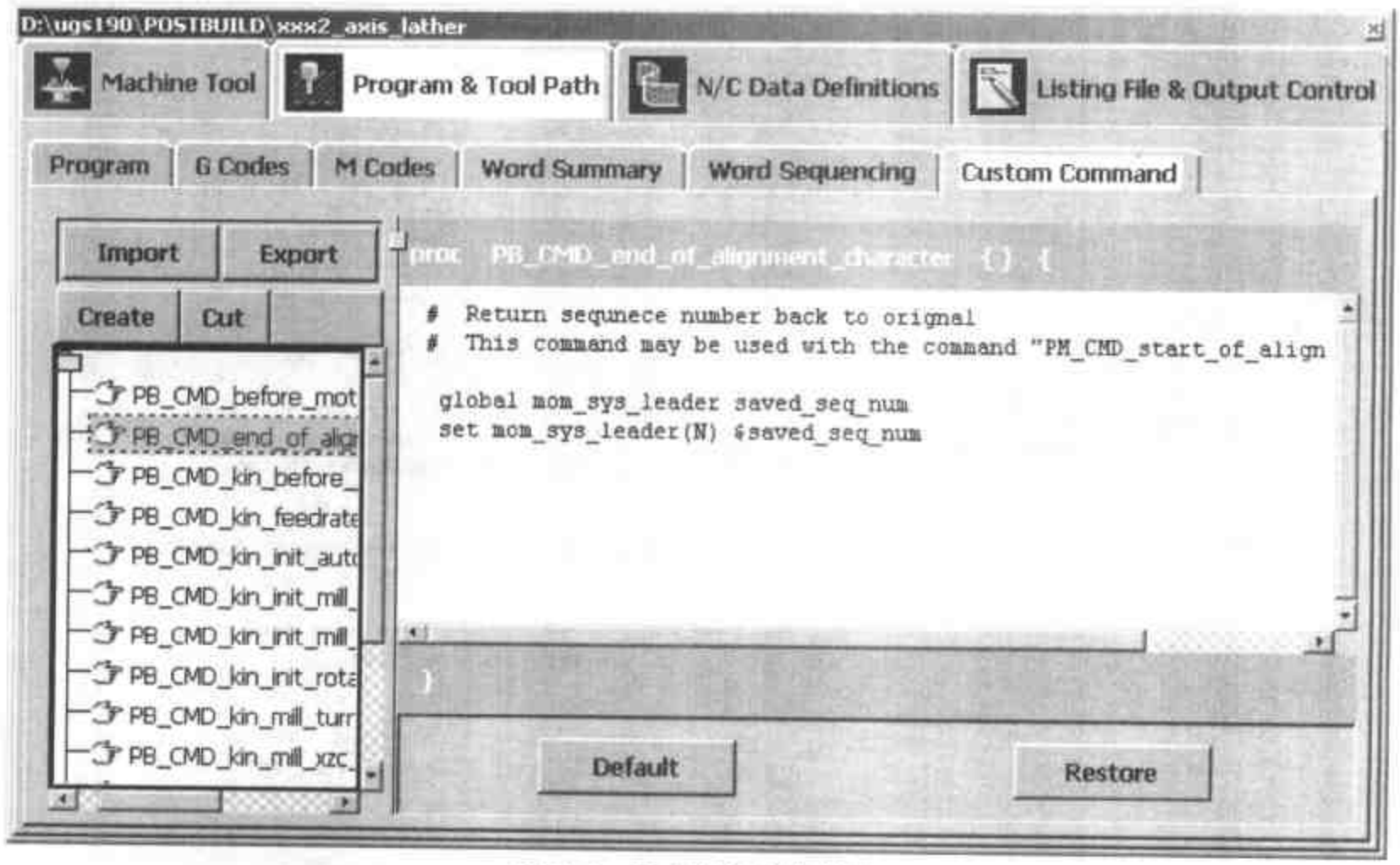

图 5-4 用户指令参数菜单

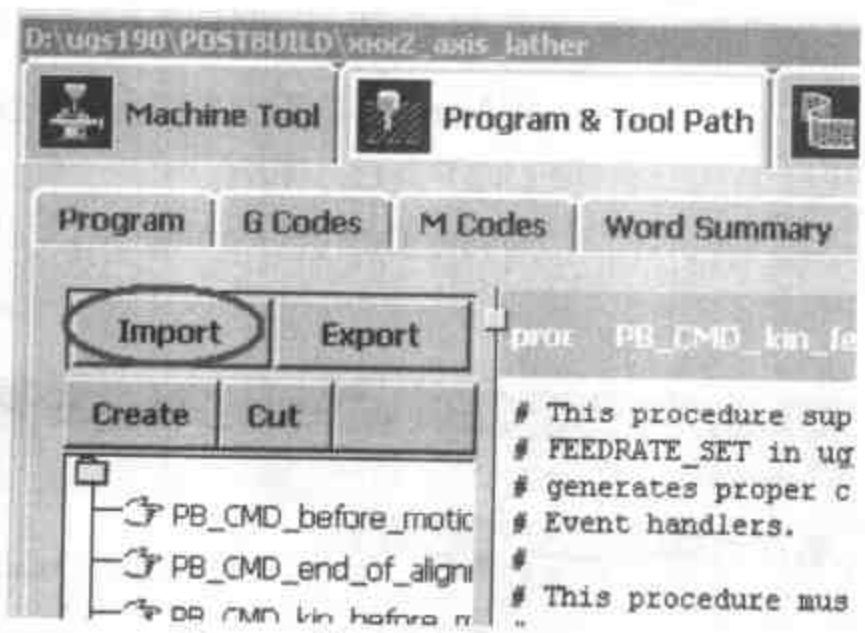

图 5-5 输入原先开发的子程序

单击 Import 后，显示文件选择对话框。默认路径是 custom_command，里面有一些已

经写好的子程序。也可以浏览其他目录，输入用户自己定义的程序。如果前面提到的语法检查开关打开，所有输入的程序都要做 Tcl 语法检查（如图 5-6 所示）。

图 5-6 选择要检查的文件

在 custom_command 目录里的文件以 pb_cmd 开头，后缀是.tcl。

单击 Export（输出），可以把创建或修改的用户指令子程序以文件的形式输出。把常用的指令保存下来，便于以后使用（如图 5-7 所示）。

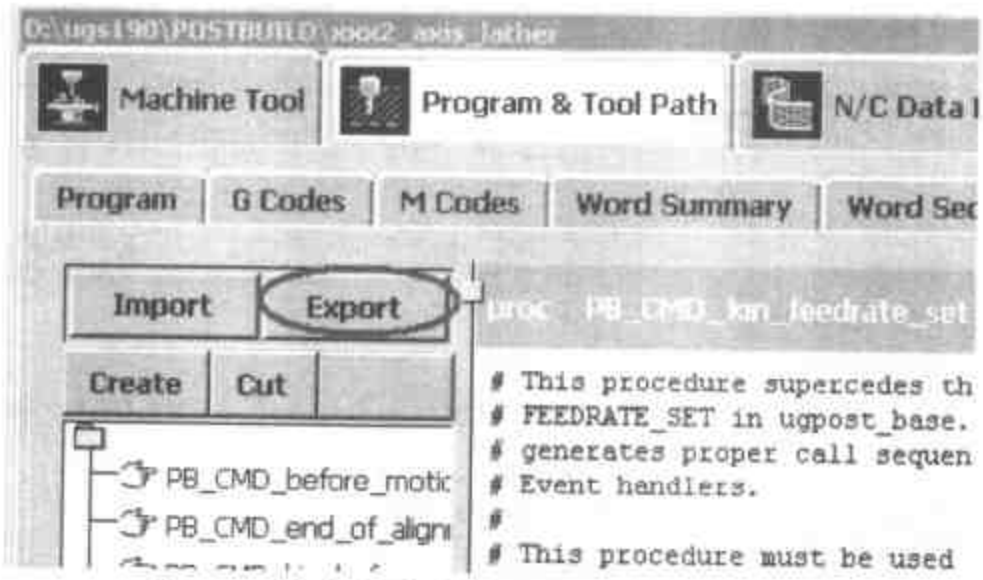

图 5-7 输出创建或修改的用户指令子程序

也可以在记事本里以文本的格式写好程序，再剪贴到 Custom Command 窗口里。可以用鼠标右键选 Paste 粘贴，如图 5-8 所示。

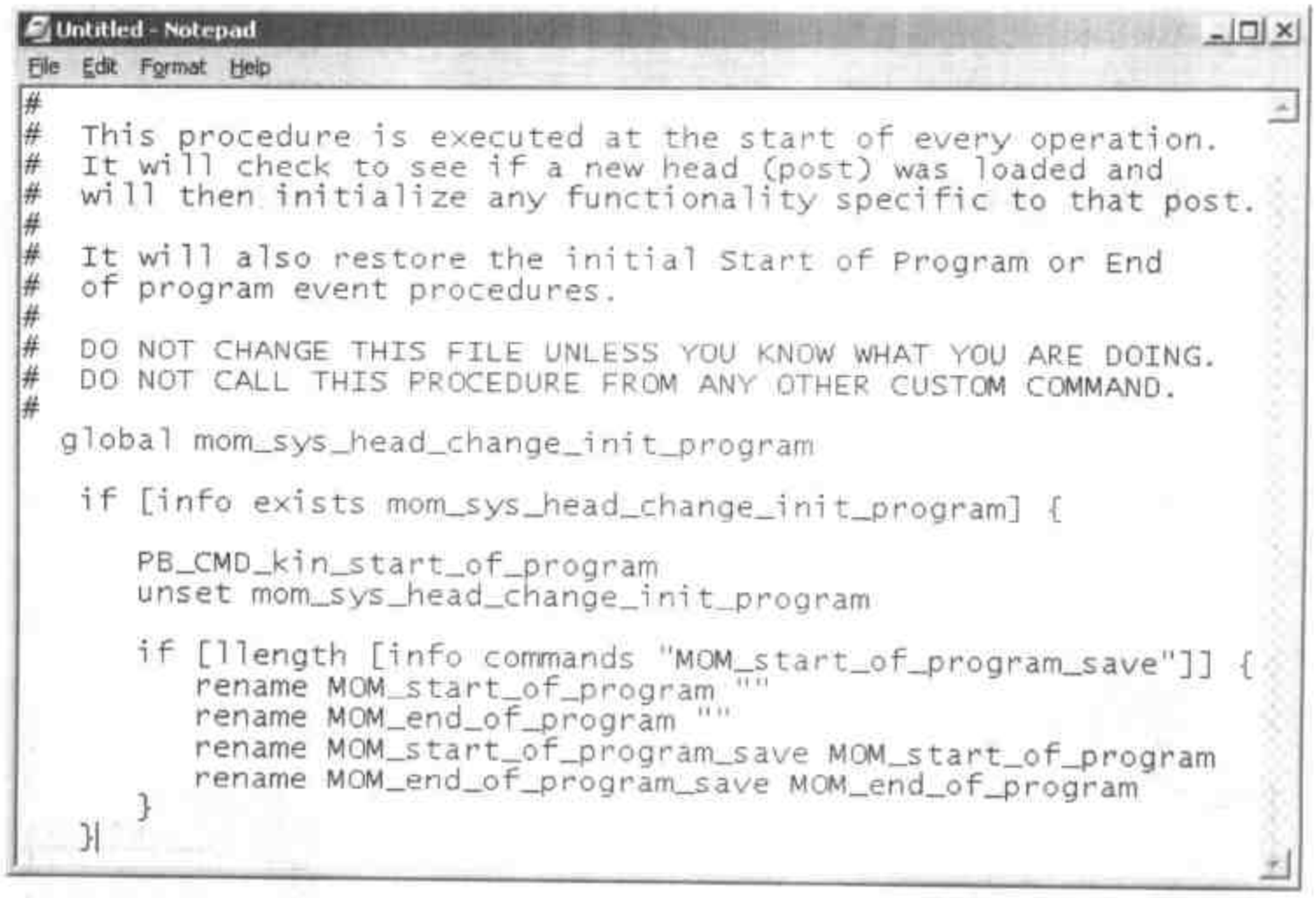

图 5-8 利用记事本（Notepad）建立用户指令

练习：用户指令

在这个练习里用输入一个用户指令，定义在处理的 NC 程序末尾显示 NC 程序的大

小；然后从一个文本文件中剪切一段程序，粘贴到用户指令窗口里，定义 NC 程序头的输出格式；最后把创建的用户指令放到合适的位置，以便得到所需的 NC 程序头尾。

第 1 步 进入 Post Builder 的 Program and Tool Path ,再进入 Custom Command。

- 在 Post Build 对话框里选择 Program and Tool Path。
- 选择 Custom Command（如图 5-9 所示）。

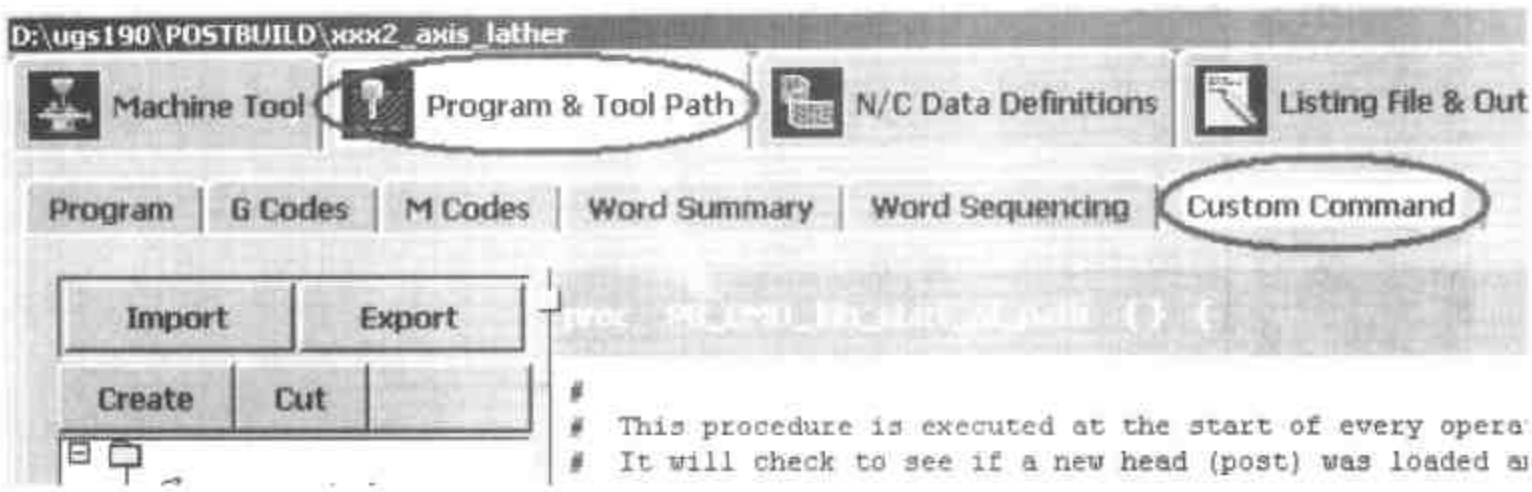

图 5-9 选择用户指令（Custom Command）

第 2 步 输入一个用户指令程序，这个程序定义输出 NC 文件的大小。

- 选择 Import（如图 5-10 所示）。

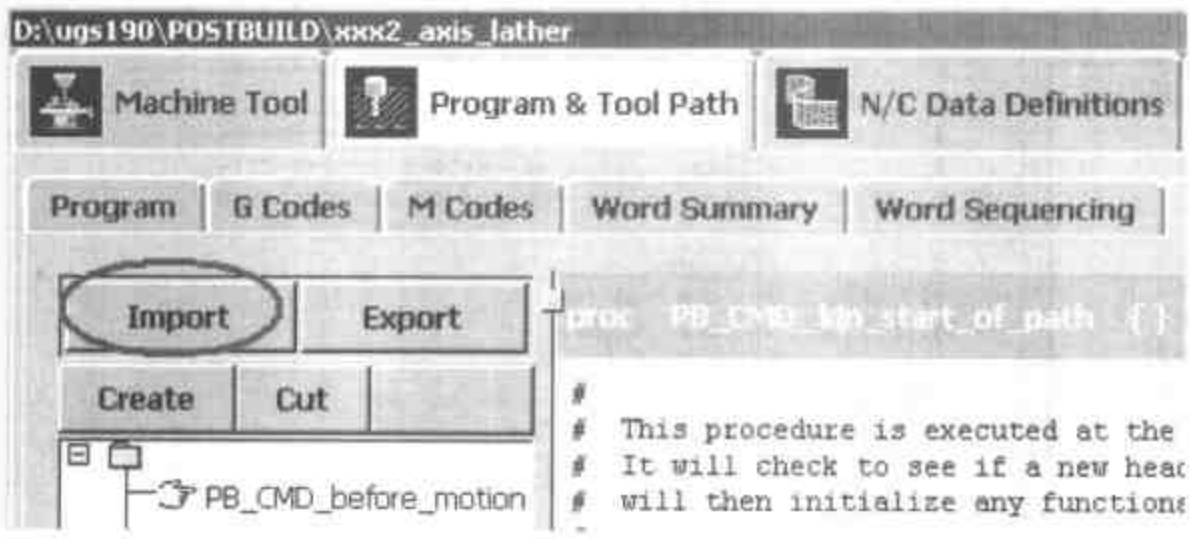

图 5-10 选择输入（Import）

显示文件选择对话框。

- 选择 pb_cmd_tape_length.tcl 文件（如图 5-11 所示）。
- 选择 Open。

输入的用户指令程序显示在 Custom Commands 窗口里（如图 5-12 所示）。

- 单击 OK。

PB_CMD_ptp_size 子程序名高亮显示在 Custom Command 左面的结构窗口里，内容显示在右面的参数窗口里。（如图 5-13 所示）

第 3 步 用剪贴的方式创建一个定义 NC 程序头的用户指令子程序。

- 如有必要选择 Custom Command。
- 在左面结构窗口里选择 PB_CMD_before_motion （如图 5-14 所示）。

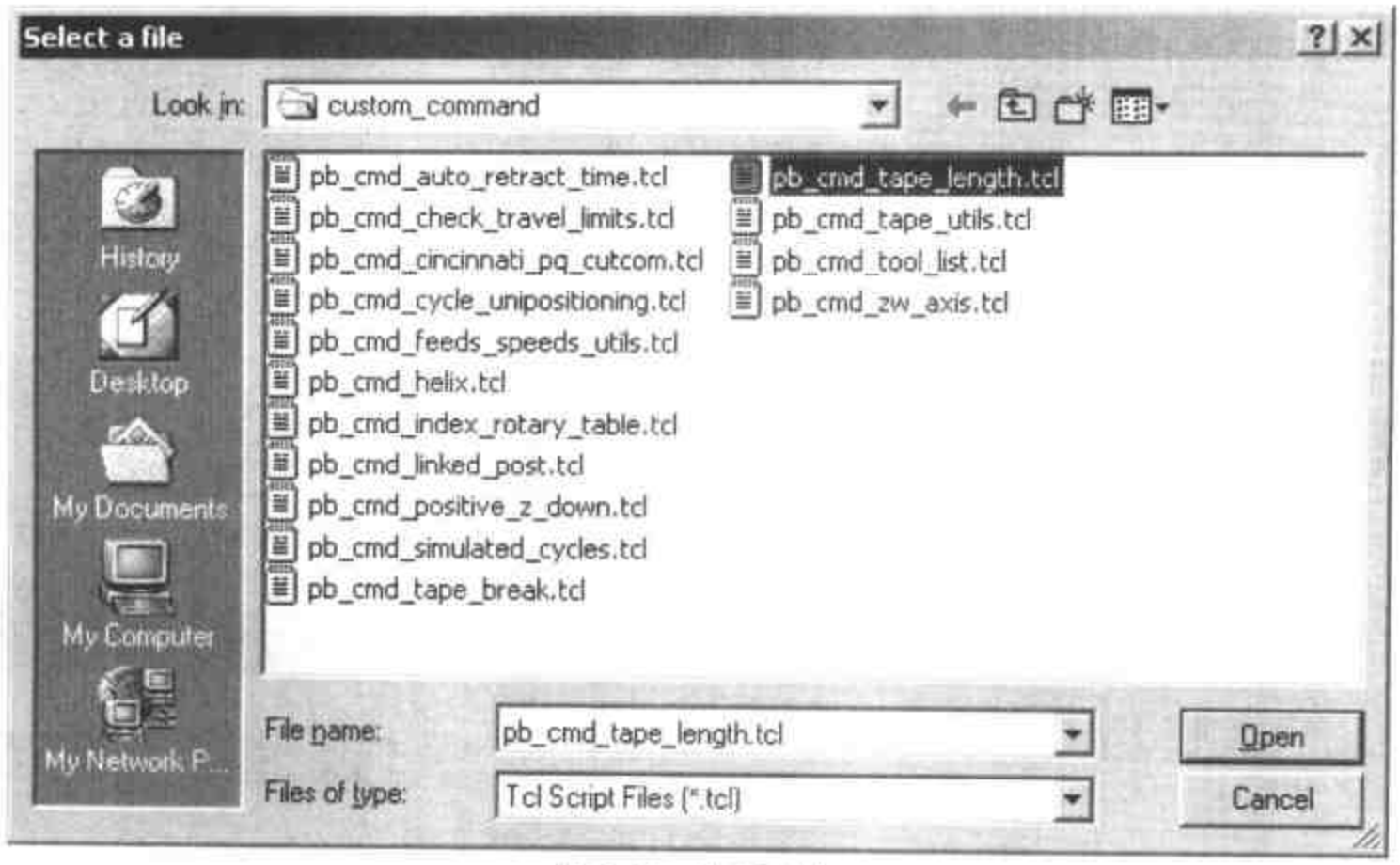

图 5-11　选择文件

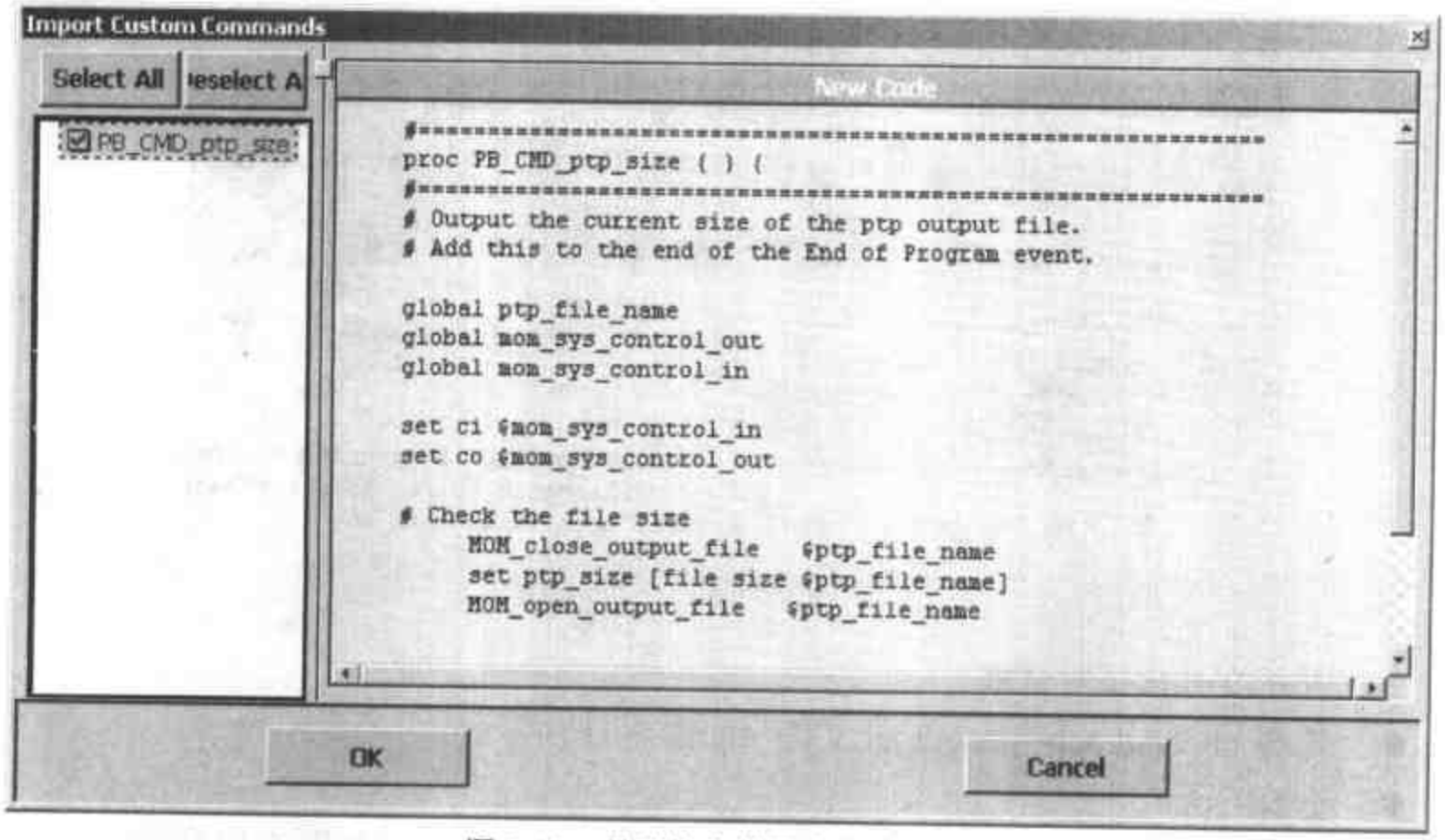

图 5-12　显示输入的用户指令程序

- 单击 Create。
  产生了一个新的指令 PB_CMD_before_motion_1（如图 5-15 所示）。
  重命名为 PB_CMD_header。
- 在刚产生的 PB_CMD_before_motion_1 上（PB_CMD_右面任何位置）左击（如图 5-16 所示）。

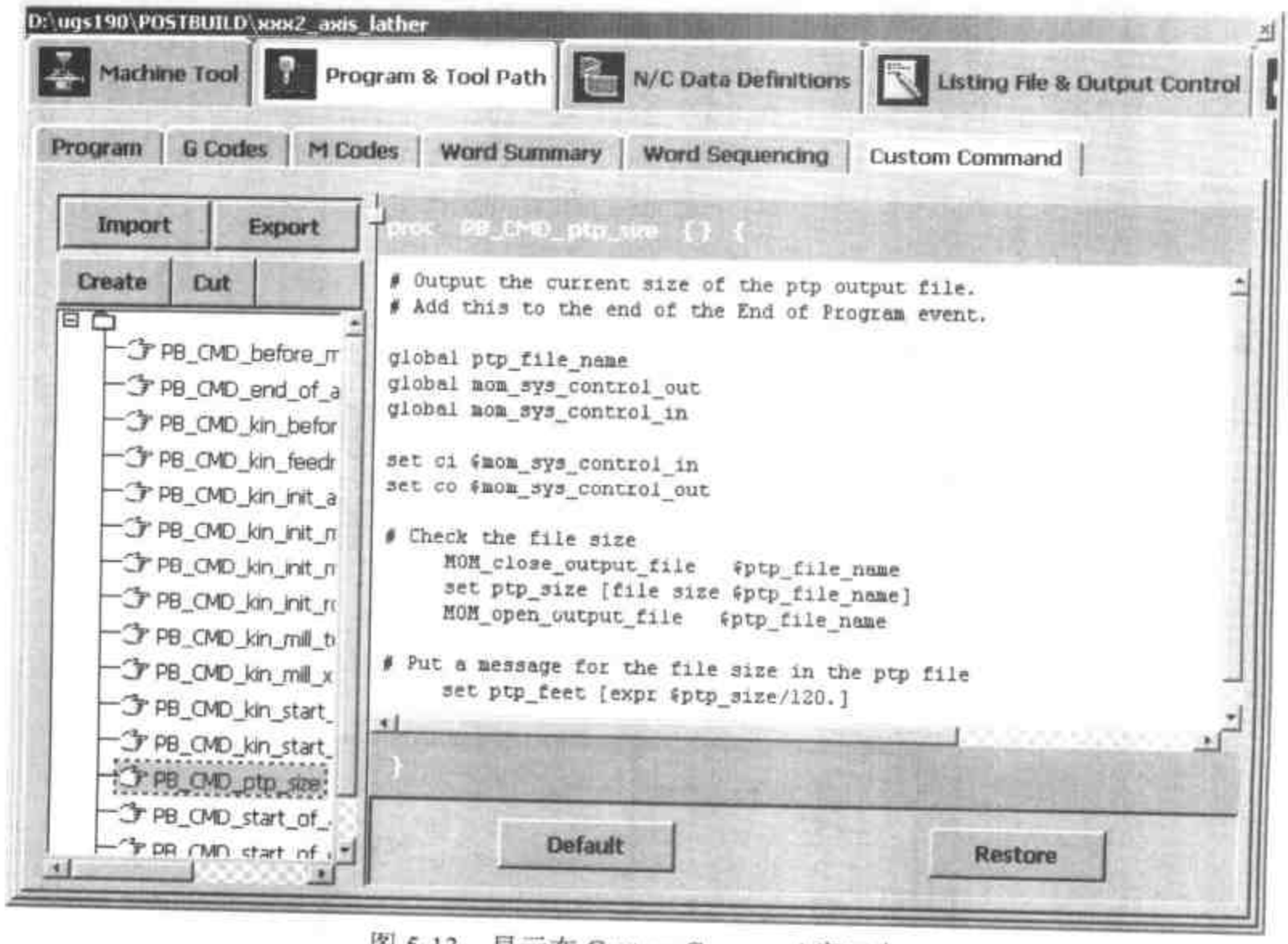

图 5-13 显示在 Custom Command 窗口中

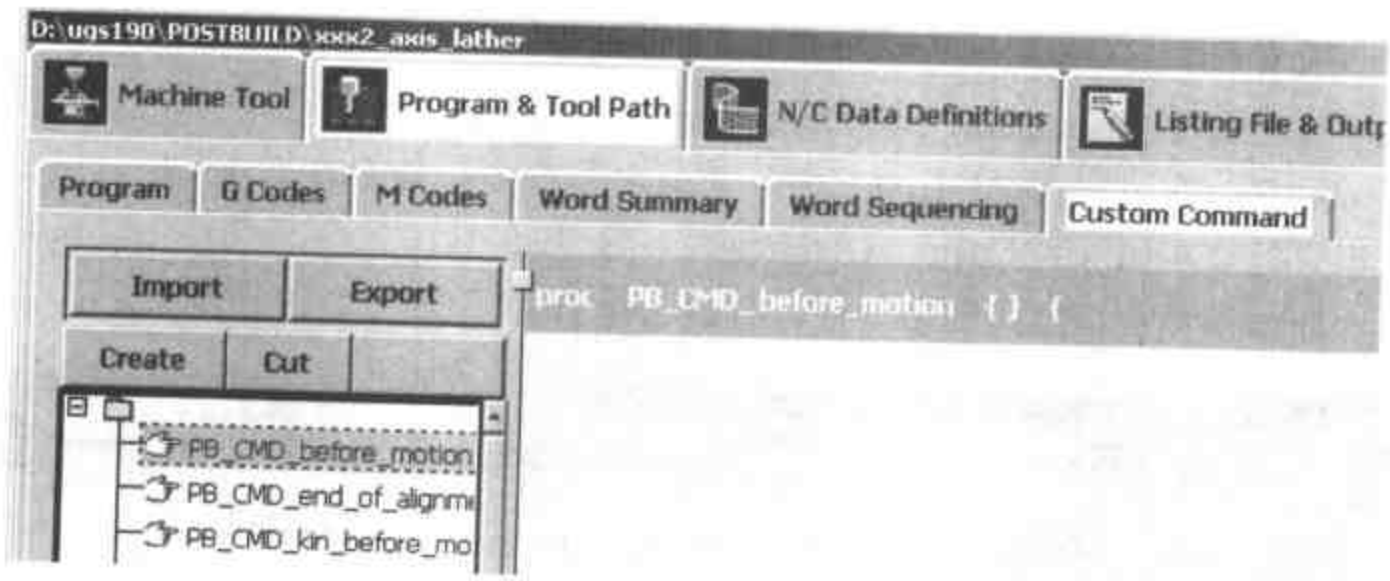

图 5-14 选择已存指令

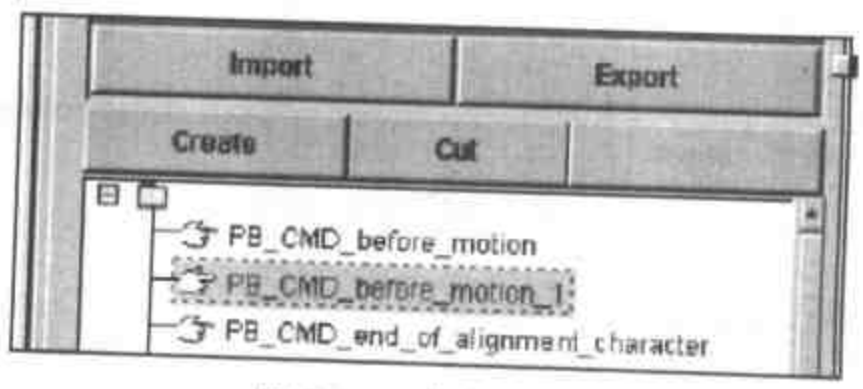

图 5-15 建立新指令

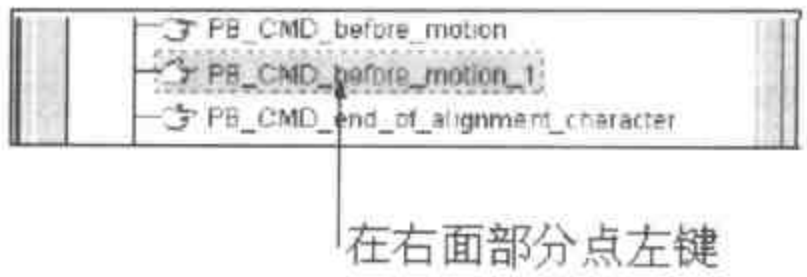

图 5-16　点左键（MB1）

- 输入名称 header，按回车键（如图 5-17 所示）。

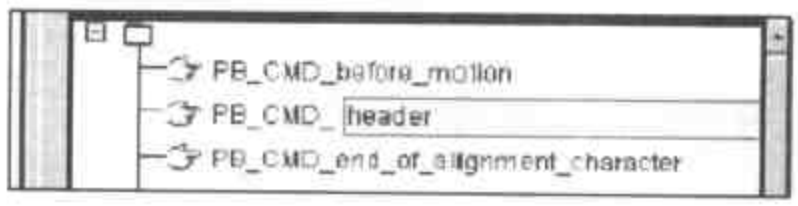

图 5-17　输入指令名

指令名改成 PB_CMD_header，并高亮显示（如图 5-18 所示）。

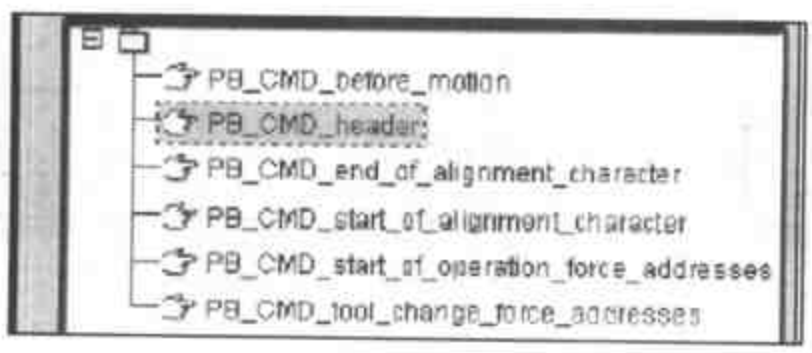

图 5-18　修改后的指令

第 4 步　在文件 pbt_custom_command_1.txt 中选取数据。

打开 pbt_custom_command_1.txt 文件，选取全部内容，按 Ctrl+C 复制。

第 5 步　把文本插入到 Custom Command。

- 在右面参数窗口里点亮第一行，右击选择 Paste（如图 5-19 所示）。

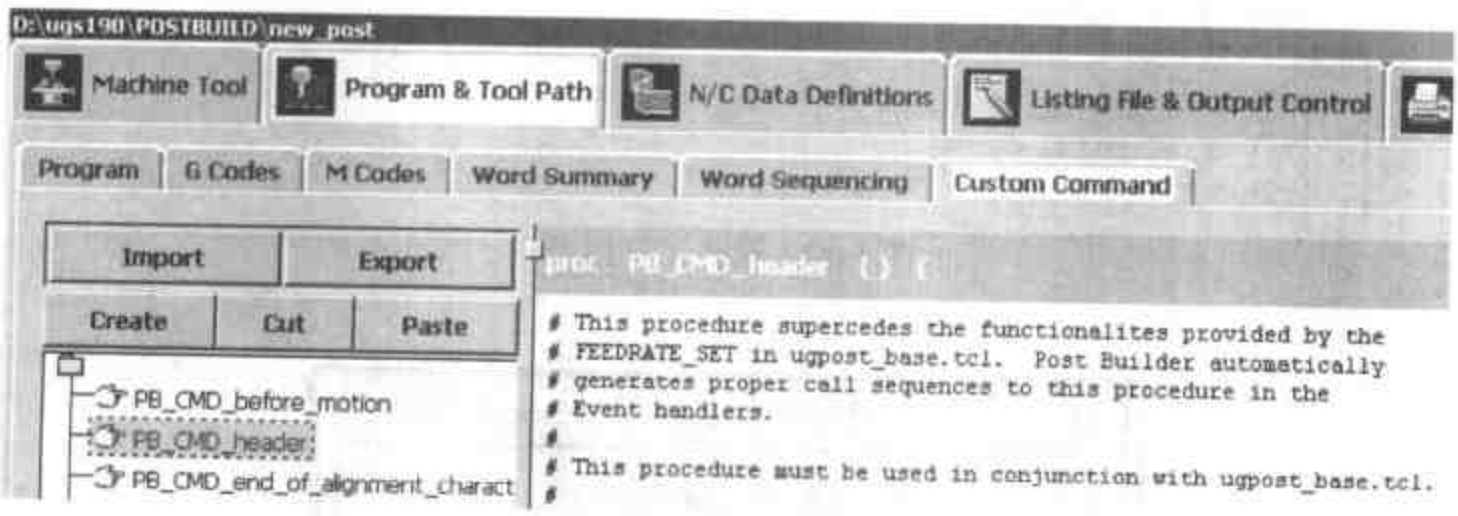

图 5-19　粘贴文本文件内容到新的用户指令

复制的内容插入到后面的参数窗口里。

刚才创建了计算程序长度的用户指令，又用不同的方法创建了定义程序头的用户指令。现在需要给这两个用户指令定位，以便在 NC 程序的前面输出程序头的信息，在后面

输出程序长度信息。

第6步 将前面产生的用户指令加入后处理程序。

- 选择 Program，在左面结构窗口里选择 Program Start Sequence。
- 在 Add Block 里选择 PB_CMD_header，拖拽到 Mom_set_seq_off 前面（图 5-20 所示）。

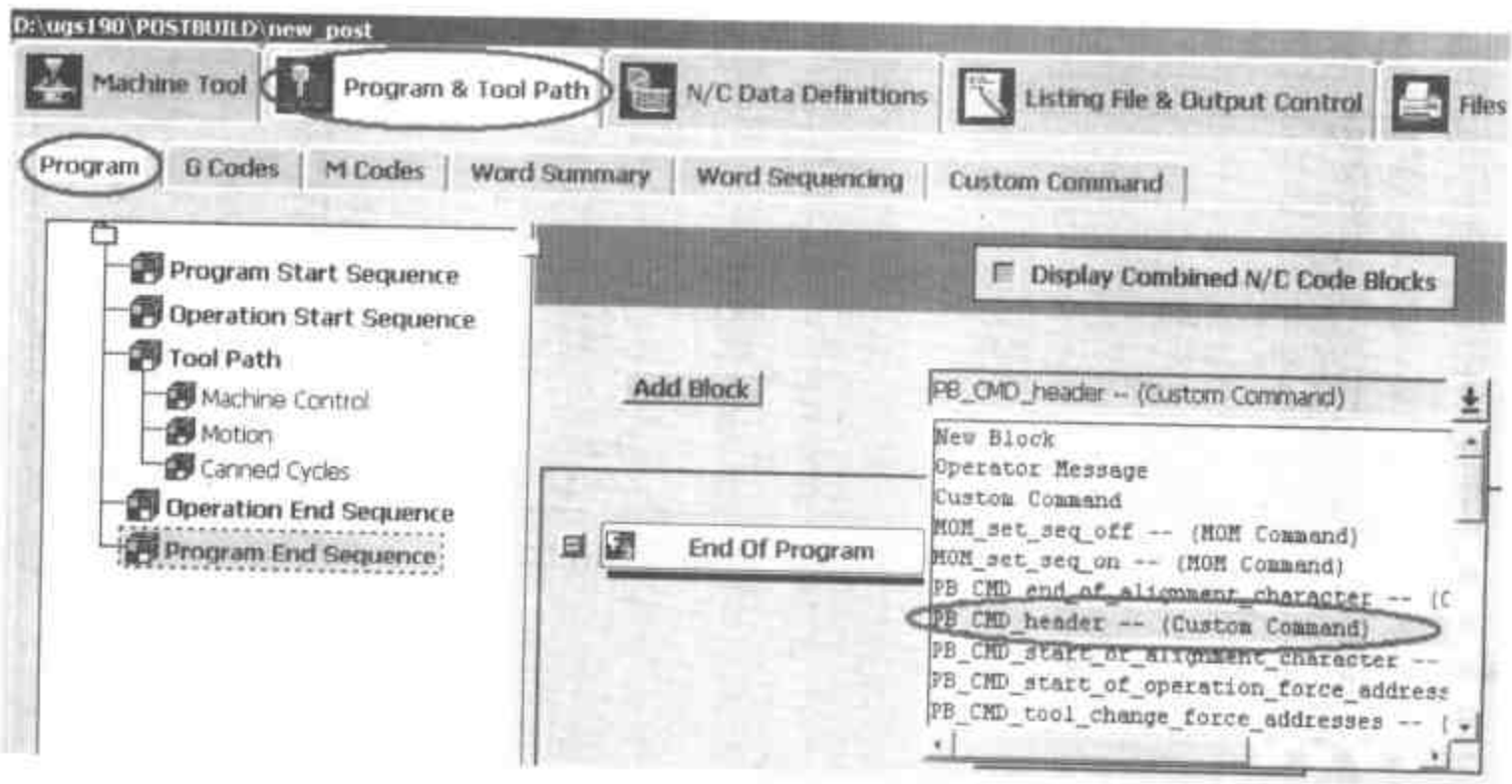

图 5-20 加用户指令到后处理程序

程序行 header 放在这儿可以保证定义的程序头出现在 NC 程序的前面。现在来定义计算程序长度的指令位置。

- 点亮 Program End Sequence。
- 在 Add Block 里选择 PB_CMD_ptp_size，拖拽到最后一行的后面（如图 5-21 所示）。

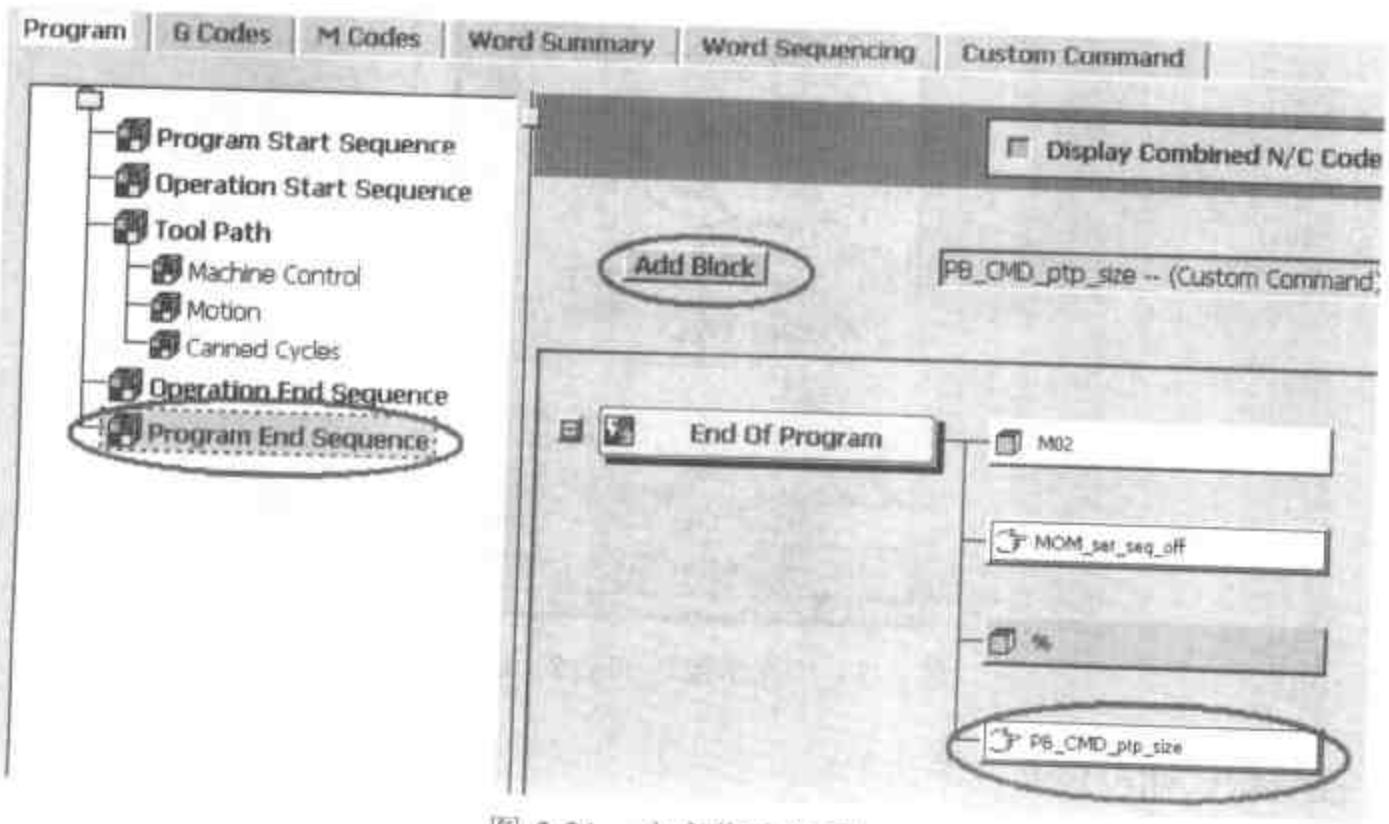

图 5-21 定义指令位置

已经把计算长度的程序行插入到合适的位置。现在来检验这些指令的执行情况。

第 7 步　保存后处理并用 pbt_mill_test.prt 检验。

- 进入 Manufacturing 应用模块。
- 在工序导航树中展开 T12345-A，选择 FACE_MILLING 操作（如图 5-22 所示）。

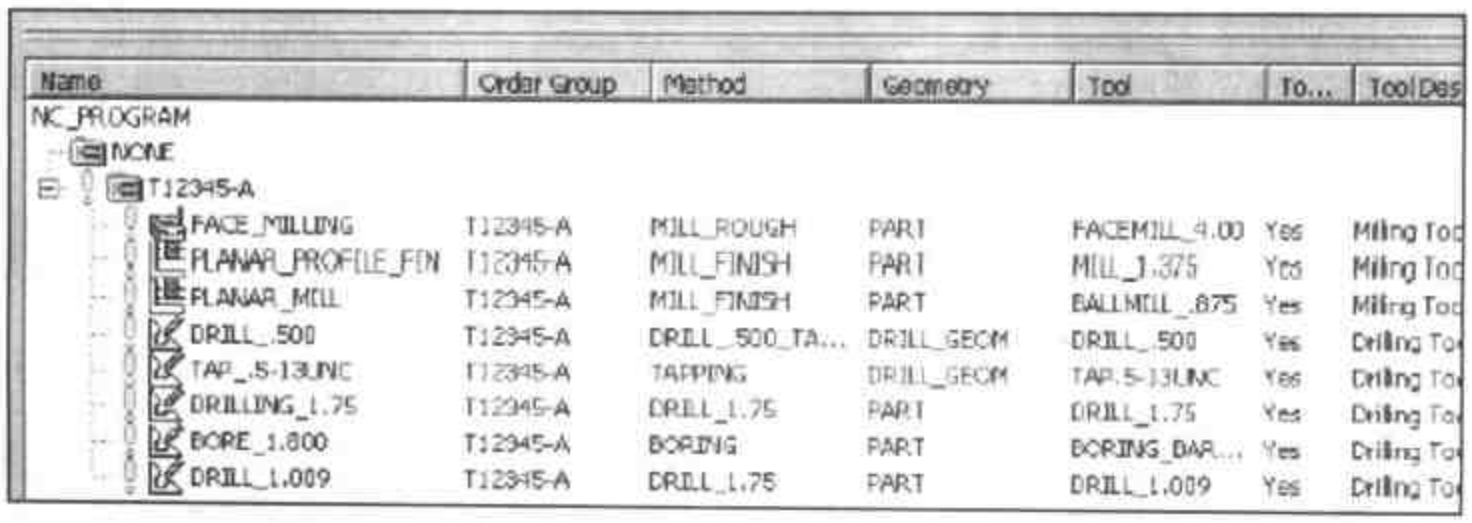

图 5-22　选择操作 FACE_MILLING

- 选择后处理图标（如图 5-23 所示）。

图 5-23　选择后处理图标

- 点亮刚修改的后处理文件（如图 5-24 所示）。

图 5-24　找修改过的后处理文件

- 接受默认输出定义，单击 OK。
- 检验输出结果。

处理结果如图 5-25 所示。

```
(*************************************)
(* Program          : \pdt_student_home\Parts\                    *)
(* Created By       :                *)
(* Creation Date    : Wed Mar 20 17:35:35 2002          *)
(* Post Processor   : mill      *)
(* UGPost Tcl File  : ***my_post.tcl*)
(* Machine ptp file : \pdt_student_home\Parts\                    *)
(*************************************)
#
N0005G70G90G40G17G94
N0010G28G91Z0.0
:0015T001M06
N0020G90G00X-3.Y-3.S0M03
N0025G43Z6.
N0030X7.Y-3.1
N0035Z4.
N0040Z2.1
N0045G01Z2.F13 M08
N0050Y-1.2
N0055X-1.3
N0060Y9.2
N0065X15.2
N0070Y-1.2
N0075X7.
N0080Y1.777
N0085X1.777
N0090Y6.223
N0095X12.223
N0100Y1.777
N0105X7.
N0110Y3.577
N0115X3.577
N0120Y4.423
N0125X10.423
N0130Y3.577
N0135X7.
N0140G00Z4.
N0145Z7.
N0150X-1.Y4.
N0155Z2.
N0160X-10.Y-10.
N0165M02
#
( PTP file size = 1590 bytes   13.3 feet )
```

图 5-25 后处理结果文件

练习结束。

本章小结

本章主要学习了以下内容：

用户指令提供了一个有效的方法来定义特殊的用户输出格式，有两种途径实现：

- 从 UG 提供的用户指令库中提取；
- 创建一个新的用户指令来满足处理需求。

第 6 章　用于 UG/Post Builder 的 Tcl 语言

【目的】

本章介绍用于 UG/Post 和 UG/Post Builder 的 Tcl 语言基础知识。

【目标】

通过本章的学习，可以：

- 理解 Tcl 语言的基本结构和语法；
- 编写简单的 Tcl 程序和用户指令。

本章包括以下练习：

练习 6-1：Tcl 基础；

练习 6-2：编写一个简单子程序；

练习 6-3：Tcl 流程控制。

6.1　Tcl　语　言

Tcl（Tool Command Language）是一个交互式解释性计算机语言。在 UG 中用于 UG/Post（后处理）、Process Assistants（CAM 过程辅助）、Shop Documentation（车间工艺文档）、clsf（刀轨文件）的生成、UG/Post Builder 等。

Tk 是 Tcl 的另一部分，可以看做是 Tcl 的一个标准扩展工具，提供用户图形界面，如按键、复选框、滚动条等。

Tcl 最初是由 John K. Ousterhout 在伯克利 California 大学开发的。Tcl 是一个公共软件，目前由 Scriptix 公司维护支持。

下面的信息有助于获取和安装 Tcl（在 Window 2000 和 Unix 平台上）：

- 在浏览器上进入下面网址：

  www.scriptics.com/software/tcltk
- 选择 Tcl8.3，按提示步骤下载（注意选择适当的平台）。

6.2　Tcl 指令结构

构造 Tcl 程序的最小元素是指令。这些指令指示 Tcl 解释执行各种任务。

指令由指令名（Name）、选项（Option）和执行内容（Argument）组成。

- Options 提供给解释程序更详细的执行指示；
- Argument 是指令要处理、更改或使用的具体内容。

概括地讲一个指令由 3 部分组成：指令名、指令选项和执行内容。

一个典型的指令语法如图 6-1 所示。

Command Name　Option　Argument 1　Argument 2　Argument N

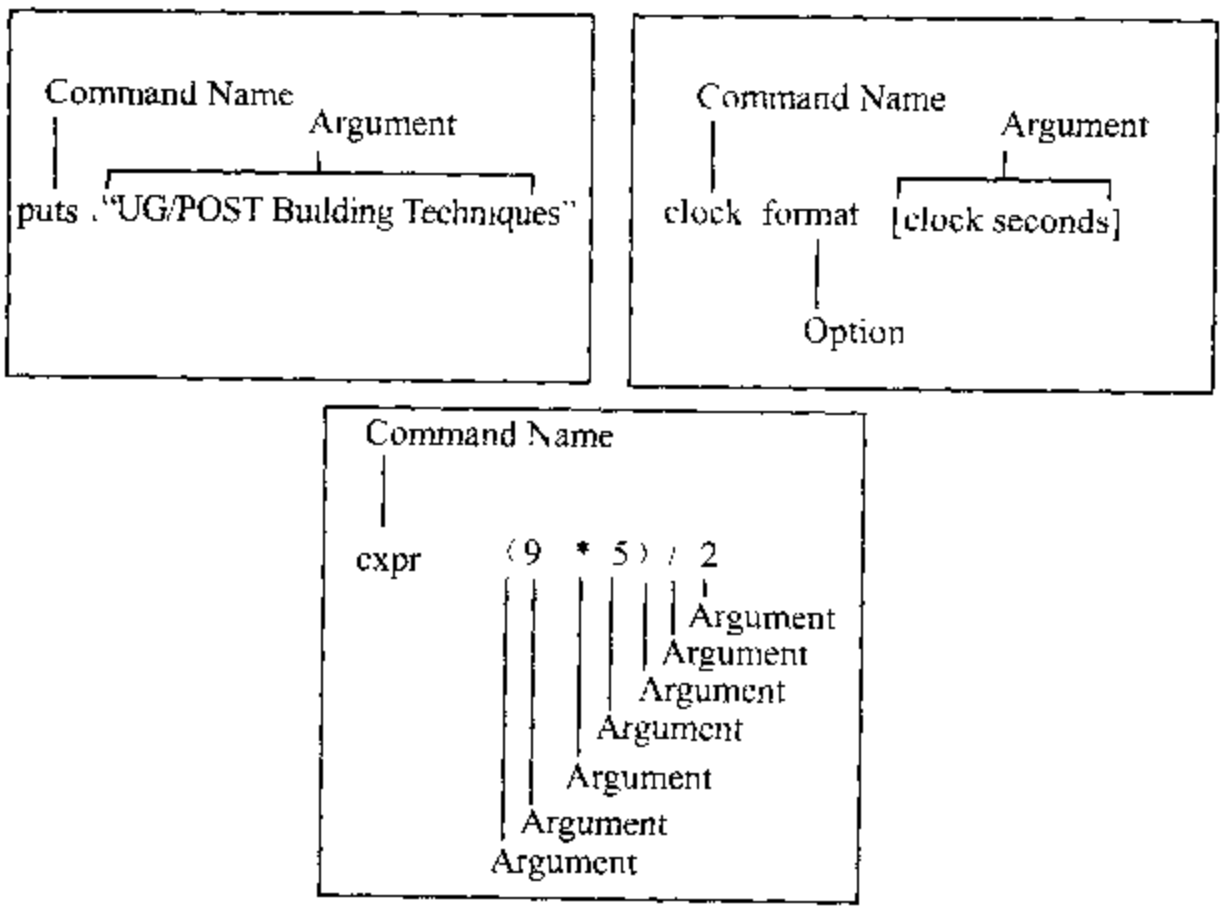

图 6-1　指令语法

6.3　Tcl 语 句

Tcl 语句由一系列指令组成，指令之间由“换行”或分号分开。

注释是在行的前面加#号。图 6-2 所示的例子是一个包含注释和指令的语句。

Tcl 程序的词之间是用“空格”、“tab”键或“分号”隔开的，但下列情形除外：

- 表示字符串的双引号；

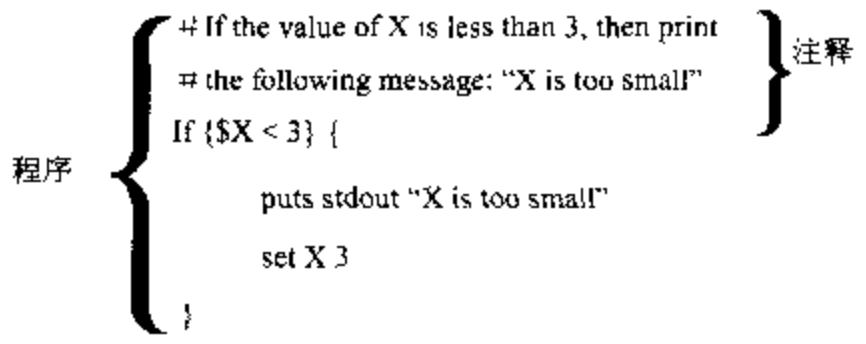

图 6-2 Tcl 程序例

```
set X "this is a word which has spaces"
```

- 大括号：

```
set Y {nested {} braces}
```

- 反斜杠：

```
set Z word\ with\\]$\ and\ space
```

- 替换不改变词的结构：

```
set abc "two words"
set b $abc
```

b 等于字符串 "two words"。

6.4 Tcl 变 量

Tcl 变量可以看做是一个有名字的信息储存器。一个变量既可以储存数字也可以储存字符。数字可以是一个表达式，也可以是一个数据；字符可以是由字母、数字或符号组成的字符号串，也可以是由三者混合组成。

变量名一定要以字母开头，后面跟字母和数字，中间不能有空格。变量名的大小写是区分的。如：

- X;
- x;
- This_is_a_Variable;
- Pre_drill;
- PRE_DRILL。

所有变量值以字符储存，有一些特殊功能来执行以字符表示的数字的计算。

Tcl 可以有局部变量和全局变量。局部变量只在子程序中使用，子程序结束，变量删除；全局变量相反，在主程序和子程序中都起作用。

6.5　数学表达式

Tcl 数学表达式用 expr 指令来执行，表达式用到的基本运算有：

- +　加；
- -　减；
- /　除；
- *　乘。

当然还有更复杂的三角函数、对数等操作。

6.6　变 量 定 义

大多数计算机语言像 FORTRAN 和 C，变量用“=”定义，如：XYZ=49、XX=5*5。在 Tcl 中，变量用 set 指令定义。

```
set XYZ 49
set XX [expr 5*5]
```

变量的代入用在变量之前加$表示。作用是让指令使用变量值而非变量名。如 A=2，让 A 乘以 10，于是用表达式写成 expr A*10。这样 Tcl 会解释成字符 A 乘以 10 而不是 A 的值乘以 10。要让 Tcl 正确理解，必须在变量名前加$，写成 expr $A*10。

变量代入的例子如图 6-3 所示。

指令	结果
set a Uni	a=Uni
set b graphics	b=gralphics
set c ab	c=ab=Unigraphics
set b 32	b=32
set a b	a=b
set a $b	a=32
set a $b+$a+$b	a=$b+$a+$b=32+32+32
set a $b.3	a=$b.3=32.3
set a $b9	无效，因为 b9 不是一个变量

图 6-3　变量代入例

Import（输入）按钮可以输入原先开发的子程序。这些子程序里一般会有功能说明

（如图 6-4 所示）。

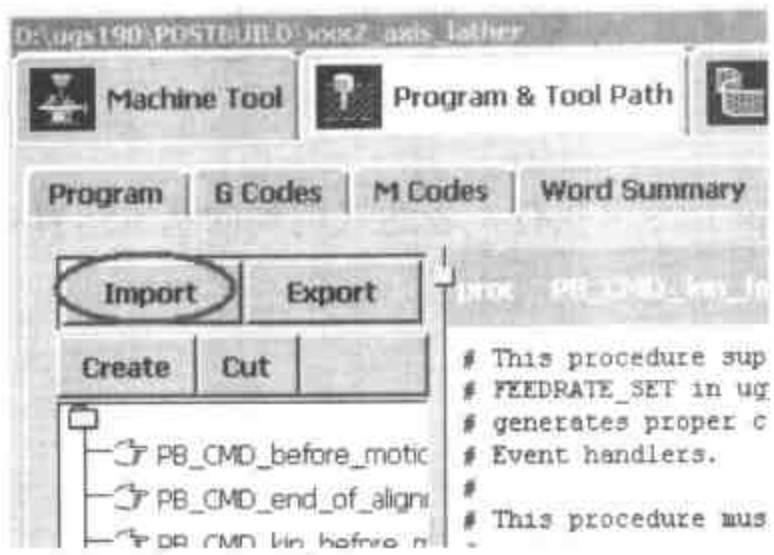

图 6-4 利用输入（Import）按钮

单击 Import 后，显示文件选择对话框。默认路径是 custom_command，里面有一些已经写好的子程序。也可以浏览其他目录，输入用户自己定义的程序。

练习 6-1：Tcl 基础

在这个练习里启动 Tcl 执行窗口，编写和执行 Tcl 指令。

第 1 步 启动 Tcl 执行窗口。

- 在菜单栏选择 Start → Programs → Tcl → Tclsh（如图 6-5 所示）。

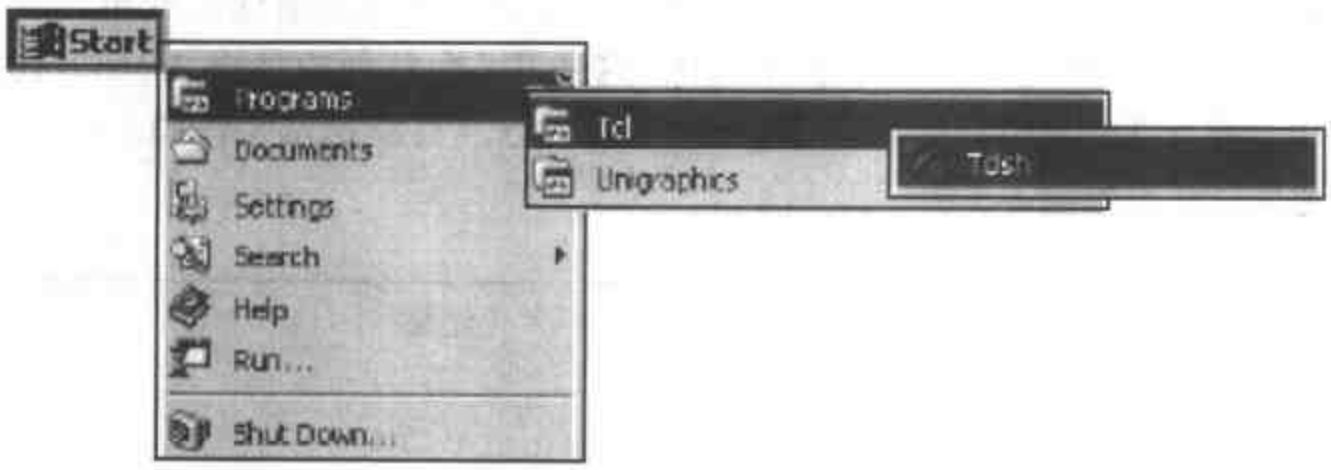

图 6-5 启动 Tcl 执行

- 显示 Tcl 执行窗口（如图 6-6 所示）。

在 Tcl 执行窗口里可以执行单行指令，也可以执行储存在一个文本文件中的程序，提示符是%。

第 2 步 输入并计算下式：

$$4+(5\times6)+(7\times28)/4$$

- 输入下式：

$$\text{expr } (4+(5\times6)+(7\times28)/4)$$

回车，得到结果是 68。

第 3 步　设置下列变量。

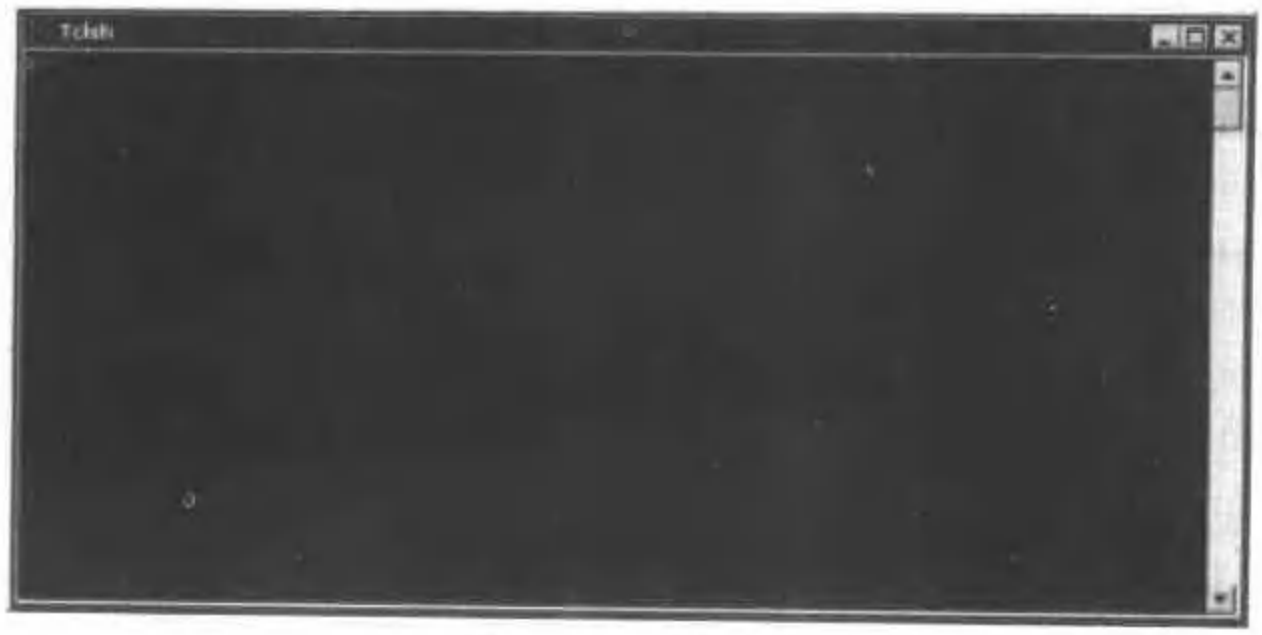

图 6-6　Tcl 执行窗口

```
mom_sys_zaxis_limit=9999.9999
```

- 输入下式：

```
set mom_sys_zaxis_max_limit 9999.9999
```

回车。

```
PI=3.1415926536
```

- 输入下式：

```
set PI 3.1415926536
```

回车。

```
DEGRAD=PI/180
```

- 输入下式：

```
set DEGRAD [expr $PI/180]
```

回车。得到结果 0.01745329252。

这里用$表示 PI 的值除以 180。

练习结束。

6.7　Tcl 子程序和函数

Procedures（子程序）可以看做是子程序。如果在程序的不同地方需要执行相同的功

能，用子程序编写简便而易于修改。

子程序结构：

```
proc    procedure name      arguments      body
```

- proc：Tcl指令，表示后面是一个子程序。如：proc sub1 x {expr $b-1}。
- arguments：变量。子程序内的变量可以有默认值。
- arguments可以是局部变量也可以是全局变量。

前面已经提到，Tcl有局部变量和全局变量，局部变量在子程序内使用。子程序被调用时，变量产生；调用结束，变量被删除。全局变量正相反，在调用程序和子程序中都有效，而且用同一个变量名。

提示：建议子程序名用大写字母，在编写程序时便于查找。

子程序例子如下：

```
#************************************************************
proc  PB_CMD_ptp_size{ } {
#************************************************************
# Output the current size of the ptp output file.
# Add this to the end of the End of Program enent.

global ptp_file_name
global mom_sys_control_out
global mom_sys_control_in

set ci $mom_sys_control_in
set co $mom_sys_control_out

# Check the file size
MOM_close output_file $ptp_file_name
set ptp_size [file size $ptp_file_name]
MOM_open_output_file $ptp_file_name

#Put a message for the file size in the ptp file
set ptp_feet [expr $ptp_size/120.]
MOM_output_literal "$co PTP file size = $ptp_size byres\
[format "%5.1f" $ptp_feet] feet $ci"

}
```

Functions（函数）与子程序功能相同，不同之处是要返回一个值。程序最后必须有返回指令。

6.8　Tcl I/O 输入/输出

用 puts 指令可以在 Tcl 执行窗口输出字符，如：puts stdout “Post Processor”，会在屏幕输出“Post Proccssor”。要输出一个变量的值，就在变量名前加“$”，格式是：puts stdout $name。

用 gets 指令可以在 Tcl 执行窗口输入字符，如：gets stdin var，会把输入的字符放到 var 变量中。

要输出特殊字符如$、\、tab、LF 等，要在这些字符前加“\”。

- 输出：\　　指令：puts stdout “\\”。
- 输出：$　　指令：puts stdout “\$”。
- 输出：tab　　指令：puts stdout “\t”。
- 输出：LF　　指令：puts stdout “\013”。

练习 6-2：编写一个简单子程序

在这个练习里编写并测试一个计算圆周长的简单子程序，然后把结果显示在 Tcl 执行窗口中。可以用记事本或写字板编写源程序。

第 1 步　用记事本或写字板编写源程序。

- 打开记事本或写字板。

第 2 步　编写一个计算圆周长的子程序，并返回结果。

- 先写一个子程序的头，子程序名为 CIRCUMFERENCE。输入下面的内容：

```
#************************
proc CIRCUMFERENCE {} {
#************************
#
# Output the circumference based on the input diameter
#
```

- 建立一个全局变量 diameter，以便在子程序外部也能使用这个变量，接着输入：

```
global diameter
```

- 建立变量 PI 和 circ 分别表示圆周率和周长，接着输入：

```
set PI 3.1415926536
```

```
set circ [expr $PI * $diameter]
```

注意在变量前面加了S。

- 输出周长值，接着输入：

```
puts "Circumference is $circ"
```

- 结束子程序，加入提示和圆直径的输入，以及子程序调用，接着输入：

```
# end the procedure
}
#inquire for diameter value
puts stdout "Enter Diameter Value"
#Get input from the keyboard
gets stdin diameter
#execute the procedure
CIRCUMFERENCE
```

第3步　保存文件，在Tcl窗口里执行，

刚才编写的程序如图6-7所示。

```
#------- ----------------------------------------
proc CIRCUMFERENCE{}{
#----------------------------------------------
# Output the circumference based on input diameter

global diameter

set PI 3.1415926536
set circ[expr $PI * $diameter]
puts "Scire"
#
# End of procedure
#
}

puts stdout "Enter Diameter Value"
gets stdin diameter
CIRCUMFERENCE
```

图6-7　计算周长的子程序

- 在记事本或写字板的工具条上选 File → Save As，把文件保存到用户目录下，命名为 circum.tcl。
- 在 Tcl 窗口里输入下面内容执行程序。

```
source "pathname filename"
```

pathname 是路径，在“\”的前面加“\”，变成“\\”；filename 是 circum.tcl（如图 6-8 所示）。

图 6-8　输入.tcl 文件

- 在看到提示“Enter Diameter Value”后输入 15，返回值是：47.123889804。
- 现在修改 circum.tcl 文件，把输出结果改成只显示小数点后三位。

练习结束。

6.9　Tcl 流程控制

流程控制，就是条件转移和循环控制，是特殊的 Tcl 指令，控制程序执行的流向。用条件转移可以控制程序执行哪些指令而跳过哪些指令；用循环控制可以控制程序重复执行某些指令。这两种指令都有一个判断真、假的语句，如果是真，执行一个流程的指令；如果是假，执行另一个流程的指令。这些流程控制的综合运用可以处理编写后处理时遇到的任何复杂情况。

条件语句 if 用如下判断方式：

- ==　等于；
- !=　不等于；
- >　大于；
- >=　大于或等于；
- <　小于；
- <=　小于或等于。

条件语句 if 结构例如下：

```
if {$mom_sys_cir_vector = ="Vector —Arc Center to Start"}{
   set mom_prev_pos($cir_index) 0.0
   set mom_pos_arc_center($cir_index) $pitch
```

```
} elseif {$mom_sys_cir_vector = ="Vector - Arc Start to Center"}
set mom_prev_pos($cir_index) $pitch
set mom_pos_arc_center($cir_index) 0.0
} elseif {$mom_sys_cir_vector = ="UnsignedVector --Arc Center to
  Start"}{
set mom_prev_pos($cir_index) 0.0
set mom_pos_arc_center($cir_index) $pitch
} elseif {$mom_sys_cir_vector = ="Absolute Arc Center"} {
set mom_pos_arc_center($cir_index) $pitch
}
```

循环语句 for 组成：

- 指令名；
- 控制循环次数的记数变量初始值；
- 判断语句，判断为真时执行循环体；
- 记数变量增量；
- 循环体。

循环语句 for 结构例子如下：

```
#*********************************************
proc VMOV { n p1 p2 } {
#*********************************************
#
# n, p1 and p2 are variables that are passed through to the proc
# when the procedure is called
#
 upvar $p1 v1; upvar $p2 v2
# "upvar" substitutes the value of p1 to the v1 variable
#
for {set I 0} {$I < $n} {incr I} {
  set v2($I) $v1($I)
 }
}
```

循环语句 while 组成：

- 指令名；
- 判断语句；
- 循环体。

循环语句 while 结构例子如下：

```
#*********************************************
```

```
proc ROTSET { angle prev_angle dir kin_leader sys_leader } {
#*********************************************
#
# This procedure will take an input angle and format for a specific
# machine.
# angle = angle to be output
# prev_angle = previous angle out. It should be mom_out_angle_pos
# dir can be either MAGNITUDE_DETERMINES_DIRECTION
# or SIGN_DETERMINES_DIRECTION
# kin_leader = leader (usually A,B or C) defined by postbuilder
# sys_leader = leader that is created by rotest. It could be C-.
#
 upvar $sys_leader lead
while {$angle < 0.0} {set angle [expr $angle + 360.0]}
while {$angle >= 360.0} {set angle [expr $angle - 360.0]}
if {$dir = = "MAGNITUDE_DETERMINES_DIRECTION"} {
 while {[expr abs($angle - $prev_angle)] > 180.0} {
   if {[expr $angle - $prev_angle] < -180.0} {
   set angle [expr $angle + 360.0]
   } elseif {[expr $angle - $prev_angle] > 180.0} {
    set angle [expr $angle - 360.0]
    }
   }
   }elseif {{$dir = = "SIGN_DETERMINES_DIRECTION"} {
   set del [expr $angle - $prev_angle]
   if {($del < 0.0 && $del > -180.0) || $del > 180.0} {
    set lead "$kin_leader-"
   } else {
    set lead $kin_leader
   }
  }
  return $angle
 }
```

条件匹配语句 switch 组成：

- 指令名；
- 指令选项；
- 匹配字符变量；
- 匹配字符集；
- 执行体。

条件匹配语句 switch 结构例子如下：

```
switch $mom_rotation_mode {
  NONE {
    set angle $mom_rotation_angle
    set mode 0
  }
  ATANGLE {
    set angle $mom_rotation_angle
    set mode 0
  }
  ABSOLUTE {
    set angle $mom_rotation_angle
    set mode 1
  }
  INCREMENTAL {
    set angle [expr $mom_pos($axis) + $mom_rotation_angle]
    set mode 0
  }
}
```

练习 6-3：Tcl 流程控制

在这个练习里编写一个主程序和两个子程序。主程序是让用户计算矩形或圆的面积。第一个子程序是输入矩形的长和宽并计算面积。第二个子程序是输入半径、定义圆周率 PI 并计算面积。

第 1 步　用记事本或写字板编写源程序。

- 打开记事本或写字板。

第 2 步　编写一个输出结果的子程序命名为 RESULTS。

- 建立子程序 RESULTS。程序如下：

```
#****************************************
proc RESULTS { } {
#****************************************
#
global area
#
if {$area > "0"} {
    puts stdout "Area is [ format "%2.2f" $area] "
```

```
  } else {
   puts stdout "Invalid Entry!"
  }
}
```

第 3 步　建立计算圆面积子程序。

- 建立一个计算圆面积的子程序，程序名为 CIRCLE。接着输入：

```
proc CIRCLE { } {
#************************************
#
# Calculate the area of a circle
#
global area
#
set PI [expr 2.0* asin(1.0)]
#
# input the radius value
#
set radius 0
puts stdout "Enter the radius value"
gets stdin radius
set area [expr $PI * $radius *$radius]
RESULTS
}
```

第 4 步　建立计算矩形面积子程序。

- 建立计算矩形面积子程序 RECTANGLE。接着输入：

```
#************************************
proc RECTANGLE { } {
#************************************
#
# Calculate the area of a rectangle
#
global area
#
# set variables for length and width; input values for the variables
```

```
#
set length o
set width o
puts stdout "Enter length"
gets stdin length
puts stdout "Enter width"
gets stdin width
set area [expr $length * $width]
RESULTS
}
```

第5步　建立主程序。

- 建立主程序，任务是区分要计算面积的类型。

```
set in ""
puts stdout " Select c for circular area, or r for rectangular area"
gets stdin in
If {$in = = "Q" || $in = = "q" || $in = = "quit"} exit
switch $in {
      r { RECTANGLE }
      c {CIRCLE }
      R { RECTANGLE }
      C {CIRCLE }
      Default {puts stdout "Invalid Entry! "}
}
```

第6步　保存文件在Tcl窗口里执行程序。

- 在记事本或写字板的工具条上选择 File → Save As，把文件保存到用户目录下，命名为 results.tcl。
- 在 Tcl 窗口里输入下面内容执行程序。

```
source "pathname filename"
```

pathname 是路径，在“\”的前面加“\”，变成“\\”；filename 是 results.tcl。

- 在看到提示“Select carea for circular area, or rarea for rectangular area”时，输入“carea”，然后回车。
- 在看到提示“Enter the radius value”时，输入“15.5”，然后回车。

得到结果：754.767635025

练习结束。

6.10　Tcl　格　式

Tcl 提供一系列指令处理 lists（列表）。列表是对一些对象的归类，并给这些对象冠以一个名字。对列表的操作包括增加、访问、查找、排序等。

建立列表用 list，增加对象用 lappend，访问对象用 lindex，替换对象用 lreplace，插入对象用 linsert。

6.11　Tcl 和 UG

作为对 Tcl 功能的扩展，用于 UG 的 UG/Post（后处理）、Process Assistants（CAM 过程辅助）、Shop Documentation（车间工艺文档）、UG Libraries（数据库）、User Defined Features（用户自定义特征）等。

Tcl 可以调用 UFUNC，UFUNC 可以调用 GRIP。Tcl 便于使用，扩展后可与 C 语言结合。目前 Tcl 用在 CAM 模块的 UG/POST 和 Shop Documentation。

Post Builder 的 Custom Command 可以让用户插入自己编写的 Tcl 子程序。Post Builder 提供程序名和头、尾。

Post Builder 提供常用的功能子程序，每一版本都会增加一些新的子程序。用户仍需要自己定义程序体，以控制输出格式来满足特殊需求。

本章小结

Tcl 语言是一种解释性执行语言，用于 UG/Post 和 UG/Post Builder。在 UG/Post Builder 的 Custom Command 中可以插入用户定义的 Tcl 子程序，以满足机床/控制系统的特殊要求。

本章主要学习了 Tcl 语言的：

- 变量；
- 数学表达式；
- 子程序；
- 输入/输出；
- 流程控制。