```
unit cmmnds2;
  interface
     uses
MemTypes,QuickDraw,OSIntf,ToolIntf,PackIntf,PrintTraps,PasLibIntf
,
              globals,help,text,cmmnds1,pusherr,pushStack,
                 evaluate;

                 procedure postError;
                 procedure postfix;
                 procedure includeRule;


implementation



procedure includeRule;
(*************************************************************)
(*                                                           *)
(*  resets a 'rule number'(all excluded rules if INCLUDEALL) *)
(*  back to active status.                                   *)
(*                                                           *)
(*************************************************************)
var low,high,nz,i:longint;
begin
  if (buffer[nextc]='t') or (buffer[nextc]='T') then
     (************************)
     (*  temporary theorem   *)
     (************************)
     begin
       nextc:=nextc+1;
       if (buffer[nextc]='t') or (buffer[nextc]='T') then nextc:
=nextc+1;
       if buffer[nextc]=blk then nextc:=nextc+1;
       if buffer[nextc]='a' then
             begin
               low:=1;
               high:=ntt;
             end
            else
              if not(buffer[nextc] in digit) then error(7)
                  else
                    begin
                      readNum(high);
                      if errcode = 0 then
                         if (high > 0) and (high <=ntt) then low:
=high
                                   else error(6);
                    end;
```

```
       if errcode = 0 then
         for i:=low to high do
          if tempactive[i] then writeln(sysm:1,' TT-',i:2,' is
already active.')
                    else
                      begin
                          write(sysm:1,' TT-',i:2,': activated.');
                          tempactive[i]:=true;
                          savesw:=true;
                          evalTempRule(i);
                          if errcode = 0 then changes(i);
                          if cflag then writeln;
                        end;
      end
    else
      (*****************************)
      (*    regular theorem        *)
      (*****************************)
      begin
        if buffer[nextc]='a' then nextChar(chc)
                             else chc:=op;
        if errcode = 0 then
          begin
            if chc=blk then
               while (nextc <= numc) and (errcode = 0) do
                 begin
                   readNum(i);
                   if errcode = 0 then
                     if (i < 0) or (i > cnumrules)  then error(6)
                      else
                      begin
                        savesw:=true;
                        if activerule[i] then
                           writeln(sysm:1,' Theorem ',i:3,':
already active.')
                         else executerule(i);
                      end;
                 end
              else
               begin
                 if chc <> 'a' then error(7)
                 else
                   begin
                     nz:=1;
                     for i:=1 to cnumrules do
                        if not(activerule[i]) then
                           begin
                             nz:=0;
                             savesw:=true;
                             executerule(i);
                            end;
                     if (nz=1) and (errcode = 0) then
```

```
                        writeln(sysm:1,' All Theorems were
already active.');
                    end;
                end;
            end;
     end;
end;


procedure postError;
(*********************************************************)
(*                                                       *)
(*     error in postfix.  delete the theorem and exit.   *)
(*                                                       *)
(*********************************************************)
var i:integer;
begin
  if allowable in [0..6] then
   begin
      write(sysm:1);
      case allowable of
          0: writeln(' Character should be =, <, or >.');
          1: writeln(' Operand or left paran. expected.');
          2: writeln(' Operator or right paran. expected.');
          3: writeln(' Missing left paran.');
          4: writeln(' Missing right paran.');
          5: writeln(' Invalid character.');
          6: writeln(' Invalid name: ',name:6);
       end;
    end;
  write('      ');
  for i:=3 to numc do
                    begin
                      write(buffer[i]:1);
                      buffer[i]:=blk;
                    end;
  writeln;
  buffer[nextc]:='^';
  write('      ');
  for i:=3 to nextc do write(buffer[i]:1);
  writeln;
  deleteTempTheorem(ntt);
  if errcode = 0 then
    begin
      pfix:=0;
      error(100);
    end;
end;

procedure postfix;
(*********************************************************)
(*                                                       *)
```

```
(*       accept a temp. theorem in infix notation        *)
(*       and convert it to postfix notation.             *)
(*       place the string in TRULES.                     *)
(*                                                       *)
(*********************************************************)
label 99;
var tstacku,opstk:array[1..20] of longint;
    lson,rson,value,clson,crson,cvalue:array[1..20] of longint;
    vi,otop,nxtnd,k,tree,root,lorr,varn,a,b,c,opnode:longint;

           procedure traverse(root:longint);
           (*****************************************)
           (*                                       *)
           (* traverses the left subtree of ctree in   *)
           (* 'lson,rson,root' order(postfix) and      *)
           (* and places the result in trules.         *)
           (*                                       *)
           (*****************************************)
           begin
             if root <> 0 then
                  begin
                    traverse(clson[root]);
                    traverse(crson[root]);
                    trmax:=trmax+1;
                    trules[trmax]:=cvalue[root];
                  end;
           end;

          procedure copytree;
          (*************************************)
          (*                                   *)
          (* copies the original operator tree     *)
          (* for next variable evaluation. also    *)
          (* computes which branches the desired   *)
          (* variable is in,i.e. 1 for left and    *)
          (* 0 for right.                          *)
          (*                                   *)
          (*************************************)
          begin
            clson:=lson;
            cvalue:=value;
            crson:=rson;
            k:=0;
            otop:=2;
            while k <> varn do
              begin
                if (lson[otop] = 0) and (value[otop] > 0) then k:
=k+1;
                otop:=otop+1;
              end;
            root:=otop-1;
            lorr:=0;
```

```
            while otop < tree do
              begin
                if lson[otop] = root then
                       begin
                         lorr:=2*lorr+1;
                         root:=otop;
                       end
                     else
                       if rson[otop] = root then
                             begin
                               lorr:=2*lorr;
                               root:=otop;
                             end;
                otop:=otop+1;
              end;
            end;

procedure transformtree;
(*****************************************)
(*                                       *)
(*   move tree around to isolate variable *)
(*   varn on the right branch of the     *)
(*   root node of ctree.                 *)
(*                                       *)
(*****************************************)
  begin
    copytree;
    while crson[crson[tree]] <> 0 do
           begin
             a:=clson[tree];
             opnode:=crson[tree];
             b:=clson[opnode];
             c:=crson[opnode];
             if odd(lorr) then
                    begin
                      clson[opnode]:=a;
                      clson[tree]:=opnode;
                      crson[tree]:=b;
                      case cvalue[opnode] of
                          1000: cvalue[opnode]:=1001;
                          1001: cvalue[opnode]:=1000;
                          1002: cvalue[opnode]:=1003;
                          1003: cvalue[opnode]:=1002;
                        end;
                    end
                  else
                    begin
                      crson[tree]:=c;
                      clson[tree]:=opnode;
                      if odd(cvalue[opnode]) then
                            begin
                              clson[opnode]:=b;
```

```
                              crson[opnode]:=a;
                              if cvalue[tree]=1005 then
cvalue[tree]:=1006
                                 else
                                   if cvalue[tree]=1006 then
cvalue[tree]:=1005;
                            end
                          else
                             begin
                               clson[opnode]:=a;
                               crson[opnode]:=b;
                               cvalue[opnode]:=cvalue[opnode]+1;
                             end;
                    end;
            lorr:=lorr div 2;
          end;
   trmax:=trmax+1;
   ntemptt:=ntemptt+1;
   tspoint[ntemptt]:=trmax;
   trules[trmax]:=cvalue[crson[tree]];
   traverse(clson[tree]);
   trmax:=trmax+1;
   if cvalue[tree]=1005 then trules[trmax]:=1006
      else if cvalue[tree]=1006 then trules[trmax]:=1005
             else trules[trmax]:=1004;
end;

procedure poppush(x,ls,rs:longint);
(*********************************************)
(*                                           *)
(*     puts next value in the postfix string.   *)
(*     combines two subtrees into one with x    *)
(*     being its root and places the new tree   *)
(*     on a stack.                           *)
(*                                           *)
(*********************************************)
begin
  trmax:=trmax+1;
  trules[trmax]:=x;
  value[nxtnd]:=x;
  rson[nxtnd]:=rs;
  lson[nxtnd]:=ls;
  otop:=otop-1;
  opstk[otop]:=nxtnd;
  ttop:=ttop-1;
  nxtnd:=nxtnd+1;
 end;

(*************************************)
(*                                   *)
(*   main body of postfix routine.     *)
(*                                   *)
```

```
(************************************)
begin
  pfix:=1;
  readName;
  if errcode = 0 then validName(1,x);
  if errcode = 0 then
    begin
      if (pfix < 1) or (x < 1) then pfix:=0;
      if x < 1 then error(1)
               else if pfix < 1 then error(2);
     end;
  if errcode = 0 then
    begin
      ntemptt:=ntemptt+1;
      ntt:=ntt+1;
      tsp[ntt]:=ntemptt;
      nxtnd:=1;
      otop:=2;
      ttop:=2;
      poppush(x,0,0);
      tspoint[ntemptt]:=trmax;
      vi:=0;
      while (op=blk) and (nextc <= numc) and (errcode = 0) do
nextChar(op);
      if errcode = 0 then
        begin
          allowable:=0;
          if not(op in [eql,lte,gte]) then postError;
          if errcode <> 0 then goto 99;
          case op of
            eql: tstacku[1]:=1004;
            lte: tstacku[1]:=1005;
            gte: tstacku[1]:=1006;
          end;
          allowable:=1;
          while (nextc <= numc) and (errcode = 0) do
             begin
               while (buffer[nextc]=blk) do nextc:=nextc+1;
               if buffer[nextc] in letter+digit then
                   begin
                     if allowable=2 then postError;
                     if errcode <> 0 then goto 99;
                     z:=nextc;
                     if buffer[nextc] in digit then
                          begin
                            readNum(x);
                            x:=-x;
                          end
                       else
                         begin
                           readName;
                           if errcode = 0 then validName(1,x);
```

```
                          if errcode = 0 then
                            begin
                              if (pfix <= 0) or (x < 1) then
                                  begin
                                    allowable:=6;
                                    nextc:=z;
                                    postError;
                                    if errcode <> 0 then goto
99;
                                  end;
                              vi:=vi+1;
                            end;
                        end;
                    if errcode = 0 then
                       begin
                         if nextc <= numc then nextc:=nextc-1
                            else
                              if not(buffer[nextc-1] in
letter+digit) then
                                    nextc:=nextc-1;
                         otop:=otop+2;
                         poppush(x,0,0);
                         ttop:=ttop+1;
                         allowable:=2;
                       end;
                  end
                else
                  if nextc <= numc then
                    begin
                      nextChar(op);
                      if errcode = 0 then
                        begin
                          nextc:=nextc-1;
                          if not(op in
[plus,minus,mult,divd,lparn,rparn]) then
                                begin
                                  allowable:=5;
                                  postError;
                                end;
                        end;
                    if errcode = 0 then
              case op of
                  plus,minus:
                       begin
                         if allowable =1 then postError;
                         if errcode <> 0 then goto 99;
                         while tstacku[ttop] < 1004 do

poppush(tstacku[ttop],opstk[otop-1],opstk[otop]);
                         ttop:=ttop+1;
                         if op=plus then tstacku[ttop]:=1000
                                    else tstacku[ttop]:=1001;
```

```
                          allowable:=1;
                        end;
                    mult,divd:
                          begin
                            if allowable =1 then postError;
                            if errcode <> 0 then goto 99;
                            x:=tstacku[ttop];
                            while (1001 < x) and (x < 1004) do
                                    begin

poppush(x,opstk[otop-1],opstk[otop]);
                                      x:=tstacku[ttop];
                                    end;
                            ttop:=ttop+1;
                            if op=mult then tstacku[ttop]:=1002
                                       else tstacku[ttop]:=1003;
                            allowable:=1;
                          end;
                    lparn:  begin
                              if allowable=2 then postError;
                              if errcode <> 0 then goto 99;
                              ttop:=ttop+1;
                              tstacku[ttop]:=1007;
                            end;
                    rparn:
                        begin
                          if allowable=1 then postError;
                          if errcode <> 0 then goto 99;
                          while tstacku[ttop] < 1007 do
                             begin

poppush(tstacku[ttop],opstk[otop-1],opstk[otop]);
                                if ttop=1 then
                                       begin
                                         allowable:=3;
                                         postError;
                                         if errcode <> 0 then goto
99;
                                       end;
                             end;
                          ttop:=ttop-1;
                        end;
                  end;  (* end case *)
                if errcode = 0 then nextc:=nextc+1;
              end;
          end;
      if errcode = 0 then
        begin
          allowable:=4;
          while ttop > 0 do
            begin
              x:=tstacku[ttop];
```

```
            if x=1007 then postError;
            if errcode <> 0 then goto 99;
            poppush(x,opstk[otop-1],opstk[otop]);
          end;
        tree:=nxtnd-1;
        tempactive[ntt]:=true;
        all:=false;
        pfix:=0;
        for varn:=1 to vi do transformtree;
        savesw:=true;
        evalTempRule(ntt);
        if errcode = 0 then changes(ntt);
      end;
    end;
  end;
99: end;

end.
```