```
unit globals;

  interface
       uses
MemTypes,QuickDraw,OSIntf,ToolIntf,PackIntf,PrintTraps,PasLibIntf
;

 const
                           (*******************************)
                                   (*
*)
                                   (*    For VAX:     =1000
*)
         perSecond=60;    (*                                 *)
                             (*    For Mac II   =60           *)
                                   (*
*)

(*******************************)
         prlim=10000;
         rstkmax=240;
         rpercol=20;
         cnumrules=460;
         tnumrules=40;
         mxncopiesm1=8;
         nparam=37;
         hf=0.495;
         ninst =-24;
         tdate =-24;
         ucomm =-23;
         fstep =-22;
         bstep =-21;
         versus=-20;
         recall=-19;
         batch =-18;
         ftwith=-17;
         twith =-16;
         ftrace=-15;
         thmtxt=-14;
         trase =-13;
         undo  =-12;
         dtt   =-11;
         tt    =-10;
         helpc =-9;
         list  =-8;
         endd  =-7;
         bound =-6;
         rules =-5;
         remove=-4;
         tymes =-3;
```

```
exclud=-2;
includ=-1;
nodes =1;
edges =2;
maxdeg=3;
mindeg=4;
chr   =5;
clique =6;
ncov=7;
ecov=8;
nind =9;
eind =10;
nccov=11;
eccov=12;
radius=13;
diam  =14;
genus =15;
nconn =16;
econn =17;
echr  =18;
girth =19;
circ  =20;
ncomp =21;
xnum  =22;
arbor =23;
earbor=24;
dom   =25;
bwidth=26;
thick =27;
spectr=28;
(******************)
(*  binary        *)
(******************)
    plnar=29;
    hamil =30;
    reg   =31;
    cycle =32;
    tree  =33;
    forest=34;
    connct=35;
    bipart=36;
    compl =37;
(*****************)
blank='      ';
blk5='     ';
blk=' ';
period='.';
comma=',';
semicol=';';
colon=':';
escp='!';
prompt='>';
```

```
          sysm='#';
          lte='<';
          gte='>';
          eql='=';
          udt='udt';
          nptb='nptb';
          yes='y';
          no='n';
          plus='+';
          minus='-';
          mult='*';
          divd='/';
          lparn='(';
          rparn=')';
    type strng=packed array[1..6] of char;
         char5=packed array[1..5] of char;
         strng2=packed array[1..2] of char;
         strng3=packed array[1..3] of char;
            strng4=packed array[1..4] of char;
            strng10=packed array[1..10] of char;
         relation=(lt,eq,ne,ge,gt,notRset);
         iset=set of 1..nparam;

    var

actualnumrules,infinity,numc,nextc,k1,k2,k3,k4,z1,i,j,k,x,zip,

stack,z,nbparam,nthms,tnthms,rn,cpustart,cpudiff,tcpustart,

mxncopies,ncopies,copyptr,ttop,trmax,pfix,ntt,ntemptt,allowable,

glb1,ghb1,numtables,ssglb1,ssghb1,primary,xtracopies,savez,

recallnum,compnum,ruletop,page,errcode,currentClock,oldcpuClock
                   :longint;

perror,perrorother,savesw,go,noescap,all,timeon,btch,trace,
              cflag:boolean;
          buffer:array[0..80] of char;

min,max,down,ssmin,ssmax,zeros,ssspec,spec,alphadisp,sdisp,dispor
d
            ,smin,smax,direction : array[1..nparam] of longint;
          mincopies,maxcopies,speccopies:
array[1..nparam,0..mxncopiesm1]
                of longint;
          name:strng;
          parameter:array[ninst..nparam] of strng;

lammin,lammax,slammin,slammax,sslammin,sslammax,rlb,rhb,rz,

grlb1,grhb1,ssgrlb1,ssgrhb1,numaction,ssnumaction:real;
```

```
          letter,digit: set of char;
          bparam : set of 1..nparam;
          op,chc,cart,traceop:char;
          rulestk:array[1..rstkmax] of char5;
          rulename,count,change:array[1..rstkmax] of longint;
          activerule:array[1..cnumrules] of boolean;
          tempactive:array[1..tnumrules] of boolean;
          rule:char5;
             weekdays:array[1..7] of strng10;
             months:array[1..12] of char5;
          action,ssaction: strng;
          actionrel,ssactionrel:strng2;
          biaction,ssbiaction:strng3;

echrRmaxdeg,mindegRpminus1,girthRcirc,circRnodes,pReven,pRodd,

nconnRnind,ssechrRmaxdeg,ssmindegRpminus1,ssgirthRcirc,
              sscircRnodes,sspReven,sspRodd,ssnconnRnind
                   :relation;
          relcopies: array[1..7,0..mxncopiesm1] of relation;
          actioncopies:array[0..mxncopiesm1] of strng;
          actionrelcopies:array[0..mxncopiesm1] of strng2;
          biactioncopies:array[0..mxncopiesm1] of strng3;
          grlb,grhb,numactioncopies,lmincopies,lmaxcopies:
                                  array[0..mxncopiesm1] of real;
          glb,ghb,idnumcopies,rtopcopy
                         :array[0..mxncopiesm1] of longint;
          undoable:array[0..mxncopiesm1] of boolean;
          trules : array[1..200] of longint;
          tspoint,tsp : array[1..tnumrules] of longint;
           rulemx,rulemn:array[1..cnumrules] of iset;
          lsinthm:array[1..cnumrules] of longint;



 procedure space(no:longint);
 procedure stars;
 procedure pause;
 procedure pause2(var op:char);
 procedure initBackup;
 procedure moveCurToBk;
 procedure moveBkToCur;
 procedure moveCurToCopyI(copy:longint);
 procedure moveCopyIToCur(copy:longint);
 procedure moveCopyIToBk(copy:longint);
 procedure moveForward;
 procedure newStart;
 procedure readLine;
 procedure nextChar(var op:char);
 procedure realPower(a:real;b:longint;var c:real);
 procedure power(a,b:longint;var c:longint);
 function log2(x:real):real;
```

```
 function trunk(x:real):longint;
 function root(x:real;n:longint):real;
 procedure iWriteUdt(k,i:longint);
 procedure wrtudf(i:longint);
 procedure rWriteUdt(i:real);
 procedure validName(instr:longint;var i:longint);
 procedure readNum(var i:longint);
 procedure readName;
 procedure readReal(var y:real);
 procedure pop(var y:longint);
 procedure numToChar(i:longint;var a:char);
 function convert(rule:char5):longint;
 function cpuClock:longint;
 procedure dayTime;
 procedure error(i:longint);


implementation

function cpuClock: longint;
(*********************************)
(*                               *)
(*    function to return the     *)
(*    system clock time.         *)
(*    VAX                        *)
(*        cpuClock:= clock;      *)
(*                               *)
(*    MAC II                     *)
(*        cpuClock:= tickcount;  *)
(*                               *)
(*                               *)
(*********************************)
begin
  cpuClock:=tickCount;
end;


procedure dayTime;
(*************************************************************)
(*                                                           *)
(*   prints the date and time. Uses                          *)
(*   system routines:                                        *)
(*   For the VAX:                                            *)
(*                                                           *)
(*    var mdate,stime:packed array[1..10] of char;            *)
(*    begin                                                   *)
(*         time(stime);
*)
(*         date(mdate);
*)
(*         writeln(sysm:1,' Date:',mdate:10,' Time: ',stime:10)
*)
```

```
(*                                                                  *)
(********************************************************************)
(*   For the MAC II:                                                *)
(*                                                                  *)
(*   var toDay:DateTimeRec;                                         *)
(*       year,mth,day,hour,minute,second,dayOfWeek:longint;         *)
(*    begin                                                         *)
(*      getTime(toDay);                                             *)
(*      with today do                                               *)
(*        begin                                                     *)
(*            writeln(sysm:1,' Today is:      '
*)
(*                    weekdays[dayOfWeek]:10, months[mth]:5,
*)
(*                     day:2,', ',year:4);
*)
(*         i:=hour;  if i >= 13 then i:=i-12;                       *)
(*          if i < 1 then i:=12;                                    *)
(*            write(sysm:1,' The time is:   ',i:2,
*)
(*                  ':',minute:2,':',second:2);                     *)
(*         if hour >12 then writeln('   P.M.')                     *)
(*                     else writeln('   A.M.');                     *)
(*        end;                                                      *)
(*                                                                  *)
(*                                                                  *)
(********************************************************************)
var toDay:DateTimeRec;
begin
  getTime(toDay);
  with today do
        begin
             writeln(sysm:1,' Today is:
',weekdays[dayOfWeek]:10,months[month]:5,
                     day:2,', ',year:4);
               i:=hour;
             if i >= 13 then i:=i-12;
             if i < 1 then i:=12;
             write(sysm:1,' The time is:
',i:2,':',minute:2,':',second:2);
             if hour > 12 then writeln('   P.M.')
                          else writeln('   A.M.');
           end;
(************************************************)
(*                                              *)
(*   The above code must change as according    *)
(*   to the comments.                           *)
(*                                              *)
(************************************************)
  rz:=(cpuClock-cpustart)/perSecond;
  write(sysm:1,' Elapsed time: ');
  if rz >= 60 then
```

```
        begin
            i:=trunk(rz/60);
          if i = 1 then write(i:2,' minute and ')
                   else write(i:3,' minutes and ');
            rz:=rz-i*60;
           end;
   writeln(rz:5:2,' seconds.');
end;


procedure error(i:longint);
(*************************************************)
(*                                               *)
(*       error trap procedure which can print an *)
(*       error message before it exits to the    *)
(*       mainline at label 100.                  *)
(*                                               *)
(*************************************************)
begin
  numc:=0;
  errcode:=i;
  if abs(i) in [1..19] then
   begin
     write(sysm:1);
     case abs(errcode) of
       1: writeln(' Invalid name(',name:6,')');
       2: writeln(' Not unique(',name:6,')');
       3: writeln(' Invalid numeric field.');
       4: writeln(' No backup copy available.');
       5: writeln(' Rule stack empty.');
       6: writeln(' Theorem number out of range.');
       7: writeln(' LIST statement error.');
       8: writeln(' Error in evaluation of Temporary Theorem.');
       9: writeln(' Error in postfix.');
      10: writeln(' Previous input line ignored.');
      11: writeln(' Invalid form of delete Temporary Theorem.');
      12: writeln(' Invalid Temporary Theorem number for
deletion.');
      13: writeln(' Invalid print Temporary Theorem statement.');
      14: writeln(' LOG2 parameter must be >= .5');
      15: writeln(' ROOT arguments must be positive.');
      16: writeln(' Unable to trace from this table.');
      17: writeln(' Rule Stack has been purged. Can not "remove"
or "undo".');
      18: writeln(' Invalid command name in call to HELP.');
      19: writeln(' Cannot reset.');
       end;
    end;
  writeln(sysm:1);
  if errcode < 0 then
         begin
           writeln(sysm:1,' System must be reinitialized.');
```

```
          writeln(sysm:1);
          x:=0;
        end
      else x:=-1;
end;



procedure space(no:longint);
(************************************************)
(*                                              *)
(*      clears screen.                          *)
(*                                              *)
(************************************************)
var i:longint;
begin
  for i:= 1 to no do writeln;
end;

procedure stars;
(************************************************)
(*                                               *)
(*       writes a line of astericks.             *)
(*                                               *)
(************************************************)
begin

writeln('************************************************************
**********')
end;

procedure pause;
(************************************************)
(*                                              *)
(*        to temporarily halt the screen.       *)
(*                                              *)
(************************************************)
var op:char;
begin
  if not(btch) then
    begin
      stars;
      if noescap then write(sysm:1,'  Hit return. >')
           else write(sysm:1,' To quit, type ',escp:1,' else hit
return. >');
      read(op);
      traceop:=op;
      if op <> blk then
         begin
           readln;
           if (errcode = 0) and (op = escp) and (noescap) then
                    writeln(sysm:1,' Sorry, cannot escape at this
```

```
time.');
         end;
      writeln;
      writeln;
    end;
end;

procedure pause2(var op:char);
(************************************************)
(*                                              *)
(*        to temporarily halt the screen.       *)
(*                                              *)
(************************************************)
begin
  if not(btch) then
      begin
        if noescap then write(sysm:1,'  Hit return. >')
           else write(sysm:1,' To quit, type ',escp:1,' else hit
return. >');
        read(op);
        if op <> blk then readln;
        writeln;
        writeln;
      end;
end;

procedure initBackup;
(**************************************)
(*                                    *)
(*     initializes backup data base   *)
(*                                    *)
(**************************************)
begin
  ssmin:=smin;
  ssmax:=smax;
  ssspec:=zeros;
  sslammin:=slammin;
  sslammax:=slammax;
  ssechrRmaxdeg:=notRset;
  ssmindegRpminus1:=notRset;
  ssgirthRcirc:=notRset;
  sscircRnodes:=notRset;
  sspReven:=notRset;
  sspRodd:=notRset;
  ssnconnRnind:=notRset;
  ssglb1:=0;
  ssghb1:=0;
  ssgrlb1:=0.0;
  ssgrhb1:=0.0;
  ssaction:='      ';
  ssactionrel:='  ';
  ssbiaction:='   ';
```

```
  ssnumaction:=0.0;
end;

procedure moveCurToBk;
(*********************************)
(*                               *)
(*   move current table to backup.  *)
(*                               *)
(*********************************)
begin
  ssmin:=min;
  ssmax:=max;
  ssspec:=spec;
  ssghb1:=ghb1;
  ssglb1:=glb1;
  ssgrlb1:=grlb1;
  ssgrhb1:=grhb1;
  sslammin:=lammin;
  sslammax:=lammax;
  ssechrRmaxdeg:=echrRmaxdeg;
  ssmindegRpminus1:=mindegRpminus1;
  ssgirthRcirc:=girthRcirc;
  sscircRnodes:=circRnodes;
  sspReven:=pReven;
  sspRodd:=pRodd;
  ssnconnRnind:=nconnRnind;
  ssaction:=action;
  ssactionrel:=actionrel;
  ssbiaction:=biaction;
  ssnumaction:=numaction;
end;

procedure moveBkToCur;
(*********************************)
(*                               *)
(*   move backup table to current.  *)
(*                               *)
(*********************************)
begin
  min:=ssmin;
  max:=ssmax;
  spec:=ssspec;
  ghb1:=ssghb1;
  glb1:=ssglb1;
  grlb1:=ssgrlb1;
  grhb1:=ssgrhb1;
  lammin:=sslammin;
  lammax:=sslammax;
  echrRmaxdeg:=ssechrRmaxdeg;
  mindegRpminus1:=ssmindegRpminus1;
  girthRcirc:=ssgirthRcirc;
  circRnodes:=sscircRnodes;
```

```
  pReven:=sspReven;
  pRodd:=sspRodd;
  nconnRnind:=ssnconnRnind;
  action:=ssaction;
  actionrel:=ssactionrel;
  biaction:=ssbiaction;
  numaction:=ssnumaction;
end;

procedure moveCurToCopyI(copy:longint);
(*********************************)
(*                               *)
(*   move current table to copy\th. *)
(*                               *)
(*********************************)
var i:longint;
begin
  for i:=1 to nparam do
         begin
           maxcopies[i,copy]:=max[i];
           mincopies[i,copy]:=min[i];
           speccopies[i,copy]:=spec[i];
         end;
  glb[copy]:=glb1;
  ghb[copy]:=ghb1;
  grhb[copy]:=grhb1;
  grlb[copy]:=grlb1;
  lmincopies[copy]:=lammin;
  lmaxcopies[copy]:=lammax;
  relcopies[1,copy]:=echrRmaxdeg;
  relcopies[2,copy]:=mindegRpminus1;
  relcopies[3,copy]:=girthRcirc;
  relcopies[4,copy]:=circRnodes;
  relcopies[5,copy]:=pReven;
  relcopies[6,copy]:=pRodd;
  relcopies[7,copy]:=nconnRnind;
  actioncopies[copy]:=action;
  actionrelcopies[copy]:=actionrel;
  biactioncopies[copy]:=biaction;
  numactioncopies[copy]:=numaction;
end;

procedure moveCopyIToCur(copy:longint);
(*********************************)
(*                               *)
(*   move copy\th table to current. *)
(*                               *)
(*********************************)
var i:longint;
begin
  for i:=1 to nparam do
        begin
```

```
          min[i]:=mincopies[i,copy];
          max[i]:=maxcopies[i,copy];
          spec[i]:=speccopies[i,copy];
        end;
  ghb1:=ghb[copy];
  glb1:=glb[copy];
  grlb1:=grlb[copy];
  grhb1:=grhb[copy];
  lammin:=lmincopies[copy];
  lammax:=lmaxcopies[copy];
  echrRmaxdeg:=relcopies[1,copy];
  mindegRpminus1:=relcopies[2,copy];
  girthRcirc:=relcopies[3,copy];
  circRnodes:=relcopies[4,copy];
  pReven:=relcopies[5,copy];
  pRodd:=relcopies[6,copy];
  nconnRnind:=relcopies[7,copy];
  action:=actioncopies[copy];
  actionrel:=actionrelcopies[copy];
  biaction:=biactioncopies[copy];
  numaction:=numactioncopies[copy];
end;

procedure moveCopyIToBk(copy:longint);
(***********************************)
(*                                 *)
(*   move copy\th table to backup.  *)
(*                                 *)
(***********************************)
var i:longint;
begin
  for i:=1 to nparam do
      begin
        ssmin[i]:=mincopies[i,copy];
        ssmax[i]:=maxcopies[i,copy];
        ssspec[i]:=speccopies[i,copy];
      end;
   ssghb1:=ghb[copy];
   ssglb1:=glb[copy];
   ssgrlb1:=grlb[copy];
   ssgrhb1:=grhb[copy];
   sslammin:=lmincopies[copy];
   sslammax:=lmaxcopies[copy];
   ssechrRmaxdeg:=relcopies[1,copy];
   ssmindegRpminus1:=relcopies[2,copy];
   ssgirthRcirc:=relcopies[3,copy];
   sscircRnodes:=relcopies[4,copy];
   sspReven:=relcopies[5,copy];
   sspRodd:=relcopies[6,copy];
   ssnconnRnind:=relcopies[7,copy];
   ssaction:=actioncopies[copy];
   ssactionrel:=actionrelcopies[copy];
```

```
   ssbiaction:=biactioncopies[copy];
   ssnumaction:=numactioncopies[copy];
end;

procedure moveForward;
(*****************************************************)
(*                                                   *)
(*    moves backup copies of saved data bases        *)
(*    forward one position.                          *)
(*                                                   *)
(*****************************************************)
begin
  if copyptr = 0 then copyptr:=mxncopiesm1
                 else copyptr:=copyptr-1;
  if undoable[copyptr] then
       begin
         ncopies:=ncopies-1;
         numtables:=numtables-1;
         moveCopyIToCur(copyptr);
         ruletop:=rtopcopy[copyptr];
       end
    else error(-17);
end;


procedure newStart;
var i:integer;
(*******************************************************)
(*                                                     *)
(*        initializes arrays of bounds and other       *)
(*        various values and switches. These are       *)
(*        the things that need to be set up for        *)
(*        a reinitialization.                          *)
(*                                                     *)
(*******************************************************)
begin
  for i:= 0 to mxncopiesm1 do
    begin
      glb[i] := 0;
      ghb[i] := 0;
      grlb[i] := 0.0;
      grhb[i] := 0.0;
      undoable[i]:=false;
    end;
  btch:=false;
  noescap:=false;
  trace:=false;
  savesw:=false;
  perror:=false;
  numc:=0;
  nextc:=1;
  stack:=-1;
```

```
  tnthms:=0;
  nthms:=0;
  down:=zeros;
  ncopies:=0;
  xtracopies:=0;
  numtables:=0;
  copyptr:=0;
  primary:=1;
  ruletop:=0;
  compnum:=0;
  recallnum:=0;
  initBackup;
  moveBkToCur;
  moveCurToCopyI(primary-1);
  moveCurToCopyI(primary);
end;

procedure readLine;
(*****************************************************************
**)
(*
*)
(*        reads the next line of input from the terminal
*)
(*        into a buffer.  commas are changed to blanks and
*)
(*        multiple blanks are compressed to one blank.
*)
(*
*)
(*****************************************************************
**)
var op:char;
    flag:boolean;
begin
  numc:=0;
  buffer[1]:=semicol;
  buffer[0]:=blk;
  flag:=true;
  while flag do
           begin
             read(op);
             flag:=not(eoln(input));
             if op=comma then op:=blk;
             if (op <> blk) or (buffer[numc] <> blk) then
                       begin
                         numc:=numc+1;
                         buffer[numc]:=op;
                       end;
            end;
   if numc > 0 then
      begin
```

```
        if buffer[numc] = blk then numc:=numc-1;
        if numc > 0 then readln;
      end;
   nextc:=1;
end;

procedure nextChar(var op:char);
(*************************************************************
**)
(*
*)
(*        retrieves the next character from BUFFER.
*)
(*        if none left then read a new line(RLINE).
*)
(*
*)
(*************************************************************
**)
begin
  if nextc > numc then readLine;
  op:=buffer[nextc];
  if op <> escp then
      if ((op=lte) or (op=gte)) and (buffer[nextc]=eql) then
nextc:=nextc+2
             else nextc:=nextc+1
   else error(100);
end;

procedure realPower(a:real;b:longint;var c:real);
(*****************************************************)
(*                                                   *)
(*            raise a to the b power.                *)
(*                                                   *)
(*****************************************************)
var i:longint;
begin
  i:=1;
  if b=0 then c:=1
         else c:=a;
  while (c < infinity) and (i < b) do
               begin
                 c:=c*a;
                 i:=i+1;
               end;
end;

procedure power(a,b:longint;var c:longint);
(*****************************************************)
(*                                                   *)
(*            raise a to the b power.                *)
(*                                                   *)
```

```
(*******************************************************)
var i:longint;
begin
  i:=1;
  if b=0 then c:=1
         else c:=a;
  while (c < infinity) and (i < b) do
               begin
                 c:=c*a;
                 i:=i+1;
               end;
end;

function log2(x:real):real;
(*********************************************)
(*                                           *)
(*    computes log base 2 of x               *)
(*      (only defined for x >= .5)           *)
(*    (see: Young and Gregory-A survey of    *)
(*          numerical mathematics, pg 64.    *)
(*                                           *)
(*********************************************)
var z:longint;
    rz,rlb:real;
begin
  z:=0;
  if x > 0.5 then
     begin
       while x > 1 do
         begin
           x:=x/2;
           z:=z+1;
         end;
       x:=(x-1)/(x+1);
       rlb:=x;
       rz:=x;
       x:=x*x;
       i:=1;
       while i < 8 do
          begin
            i:=i+2;
            if rz > 0.000001 then
               begin
                 rz:=rz*x;
                 rlb:=rlb+rz/i;
               end;
         end;
       log2:=z+2*rlb/0.69314718;
     end
   else error(14);
end;
```

```
function trunk(x:real):longint;
(*****************************************)
(*                                       *)
(*    truncates real numbers. For saftey *)
(*    we add a small value to x then     *)
(*    use the systems trunc function.    *)
(*                                       *)
(*****************************************)
  begin
    trunk:=trunc(x+0.001);
  end;

function root(x:real;n:longint):real;
(************************************************)
(*                                              *)
(*   a function to calculate the 'n' root of 'z'.  *)
(*   (see: Young and Gregory-A survey of numerical *)
(*         mathematics, pg 66.                    *)
(*                                              *)
(************************************************)
var i:longint;
    y,f,sum,term:real;
begin
  root:=x;
  y:=1.0/n;
  if (x <= 0) or (n <= 0) then
    begin
      writeln('x =',x:10:5,' n =',n:10);
      error(15);
    end
  else
    if n > 1 then
        begin
          x:=y*log2(x);
          n:=trunk(x);
          f:=x-n;
          if f > 0.5 then
           begin
             n:=n+1;
             f:=f-1.0;
           end;
          f:=f*0.6931471806;
          sum:=1.0+f;
          term:=f;
          i:=2;
          while (abs(term) > 0.0000001) and (i < 11) do
             begin
               term:=term*f/i;
               sum:=sum+term;
               i:=i+1;
             end;
          power(2,n,i);
```

```
          root:=i*sum;
        end;
      end;


procedure iWriteUdt(k,i:longint);
(*****************************************************************
)
(*
*)
(*        if k is infinity then write 'udt'(undetermined)
*)
(*        if k is finite but too large for i digits then
*)
(*         write 'not printable' else write the value k.
*)
(*
*)
(*****************************************************************
)
begin
  if k >= infinity then write(udt:i)
         else if k < prlim then write(k:i)
                     else write(nptb:i);
end;

procedure wrtudf(i:longint);
(*****************************************************************)
(*                                                               *)
(*           if i is infinity then write 'undef',                *)
(*           else write i.                                       *)
(*                                                               *)
(*****************************************************************)
begin
  if i < prlim then write(i:5,' ')
        else if i>=infinity then write('undef ')
                else  write(nptb:5,' ');
end;

procedure rWriteUdt(i:real);
(*****************************************************************
**)
(*
*)
(*        if i is infinity then write 'udt'(undetermined)
*)
(*        otherwise write i.  (note: this involves a real para-
*)
(*        mater, iWriteUdt is for longints.
*)
(*
*)
```

```
(*****************************************************************
**)
begin
  if i >= infinity then write(udt:6,'   ')
                   else write(i:9:3);
end;

procedure validName(instr:longint;var i:longint);
(*****************************************************************
********)
(*
*)
(*       determines whether name is a valid invariant(or
abbreviation)  *)
(*       returns pfix = -1 if not(or if not unique).
*)
(*
*)
(*****************************************************************
********)
var j,l,k,match,found,lo,hi:longint;
begin
  if name=blank then error(100)
   else
     begin
       found:=0;
       i:=0;
       lo:=ninst;
       hi:=nparam;
       if instr < 0 then hi:=-1
                    else if instr > 0 then lo:=1;
       k:=6;
       while name[k]=blk do k:=k-1;
       found:=0;
       for j:=1 to k do if not(name[j] in letter+digit) then
found:=-1;
       if found=0 then
          for j:=lo to hi do
             begin
               match:=1;
               l:=1;
               while (match=1) and (l <= k) do
                      if name[l]=parameter[j][l] then l:=l+1
                                                 else match:=0;
               if match=1 then
                     begin
                       found:=found+1;
                       i:=j;
                     end;
             end;
       if found=1 then name:=parameter[i]
             else
```

```
                begin
                  if pfix <> 0 then
                       begin
                         if found <> 1 then i:=0;
                         pfix:=-1;
                       end
                     else
                       if found < 1 then error(1)
                                    else error(2);
                end;
     end;
end;

procedure readNum(var i:longint);
(*********************************************)
(*                                           *)
(*       reads a numeric value.              *)
(*                                           *)
(*********************************************)
begin
  if nextc > numc then write(sysm:1,' Enter numeric value. ?');
  i:=0;
  nextChar(chc);
  if (errcode = 0) and (chc=blk) then nextChar(chc);
  if (errcode = 0) and (chc='u') and
     (x in [radius,diam,girth,circ]) then
     begin
       i:=infinity;
       chc:=blk;
       numc:=0;
     end
   else
     while (chc in digit) and (errcode = 0) do
        begin
          i:=10*i+(ord(chc)-zip);
          if nextc > numc then chc:=blk
          else nextChar(chc);
        end;
  if (not(chc in [blk,period])) and (pfix=0) and (errcode = 0)
then error(3);
end;

procedure readName;
(*****************************************)
(*                                       *)
(*         reads a character string      *)
(*         into 'name'.                  *)
(*                                       *)
(*****************************************)
label 99;
var i,k:longint;
begin
```

```
  name:=blank;
  nextChar(op);
  if errcode = 0 then
   if nextc <= numc+1 then
    begin
      if (op=blk) and (nextc <= numc) then nextChar(op);
      if errcode = 0 then
         begin
           i:=1;
           if op in letter then
             while (op in letter+digit) and (i < 7) do
                begin
                  name[i]:=op;
                  if nextc > numc then op:=semicol
                                  else nextChar(op);
                  if errcode <> 0 then goto 99;
                  i:=i+1;
                end
               else
                 if op='^' then
                     begin
                       x:=radius;
                       readNum(i);
                       if errcode <> 0 then goto 99;
                       if (i <> infinity) and (i > 0) then
                           begin
                             for k:= 1 to nparam do
                                  begin
                                    if (min[k] > i) or
                                       (min[k] >= infinity) then
min[k]:=i;
                                    if (max[k] > i) or
                                       (max[k] >= infinity) then
max[k]:=i;
                                  end;
                            if (lammin > i) or
                               (lammin >= infinity) then
lammin:=i;
                            if (lammax > i) or
                               (lammax >= infinity) then
lammax:=i;
                            writeln(sysm:1,' "Infinity" was
',infinity:10);
                            infinity:=i;
                          end;
                       writeln(sysm:1,' "Infinity" is now
',infinity:10);
                     end
                   else
                    while (i < 7) and (nextc <= numc+1) do
                        begin
                          name[i]:=op;
```

```
                              if nextc <= numc then nextChar(op)
                                               else i:=7;
                              if errcode <> 0 then goto 99;
                              i:=i+1;
                            end;
            end;
          end;
   99: end;


procedure readReal(var y:real);
(*************************************************)
(*                                               *)
(*        reads a real numeric value.            *)
(*                                               *)
(*************************************************)
var i:longint;
    x:real;
begin
  readNum(i);
  if errcode = 0 then
       begin
         if i=infinity then chc:=escp;
         y:=i;
         if chc=period then
            if nextc > numc then chc:=blk
               else
               begin
                 readNum(i);
                 if errcode = 0 then
                   begin
                     if i=infinity then chc:=escp;
                     x:=i;
                     while x >= 1 do x:=x/10;
                     y:=y+x;
                   end;
                end;
         if (chc <> blk) and (errcode = 0) then error(3);
       end;
end;

procedure pop(var y:longint);
(*********************************************)
(*                                           *)
(*       pops next invariant to be evaluated.*)
(*                                           *)
(*********************************************)
begin
  y:=stack;
  if stack > 0 then
      begin
        stack:=down[stack];
```

```
        down[y]:=0;
      end;
end;

procedure numToChar(i:longint;var a:char);
(*****************************************)
(*                                       *)
(*  converts a digit to an character     *)
(*                                       *)
(*****************************************)
begin
  i:=i-(i div 10)*10;
  case i of
     0:a:='0';
     1:a:='1';
     2:a:='2';
     3:a:='3';
     4:a:='4';
     5:a:='5';
     6:a:='6';
     7:a:='7';
     8:a:='8';
     9:a:='9';
    end;
end;

function convert(rule:char5):longint;
(************************************************)
(*                                              *)
(*     convert alpha string to its numeric value.  *)
(*                                              *)
(************************************************)
var i,j:longint;
begin
  if rule[2] = 'T' then i:=-(ord(rule[5])-zip)
      else
       begin
         i:=0;
         for j:=1 to 3 do
               if rule[j] in digit then i:=10*i+ord(rule[j])-zip;
       end;
   convert:=i;
end;


end.
```