COMMERCIAL-IN CONFIDENCE

SCIENCE RESEARCH COUNCIL
RUTHERFORD & APPLETON LABORATORIES

COMPUTING DIVISION

D I S T R I B U T E D   C O M P U T I N G   N O T E   3 4 3

PERQ
TECHNICAL NOTE 3
Notes on Programming the PERQ

issued by
R W Witty

28 January 1981

DISTRIBUTION:	F R A Hopgood
	R W Witty
	D A Duce
	W P Sharpe
	C J Prosser
	J Brown
	I D Benest
	A S Williams
	RL Support/Comp Facils/PERQ/General file

1. PERQ PERFORMANCE

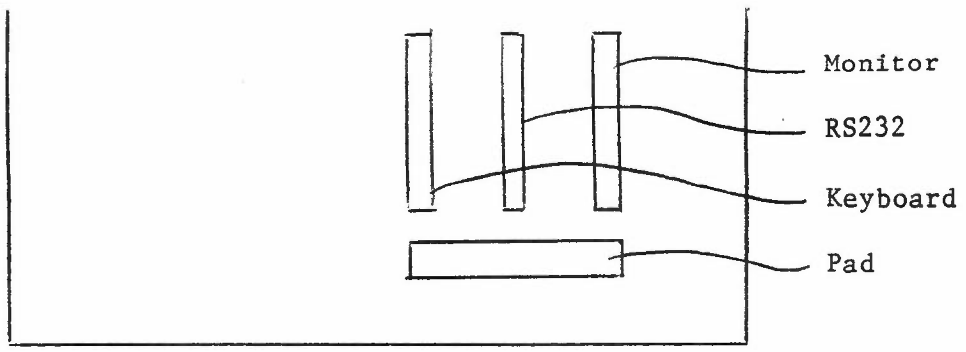
```
for i: = 1 to 1000 do
for j: = 1 to 1000 do <statement>;
```

<statement>	repetition	time	Comment
Null	1 M	12 secs	
K: = K	1 M	15 secs	3 Q codes?
K: = K + 1	1 M	15 secs	4 Q codes?
K: = K + K	1 M	15 secs	"
K: = K * K	1 M	22 secs	"
K: = K div K	1 M	27 secs	"

2. FLOPPY FORMATTING

```
Run program  TEST FLOPPY
             DD?          NO
             Bttedo?      YES
             Interface?   5
             Seed ?       56345   (a random number)
             Format ?     YES
             Test ?       Ynn
```

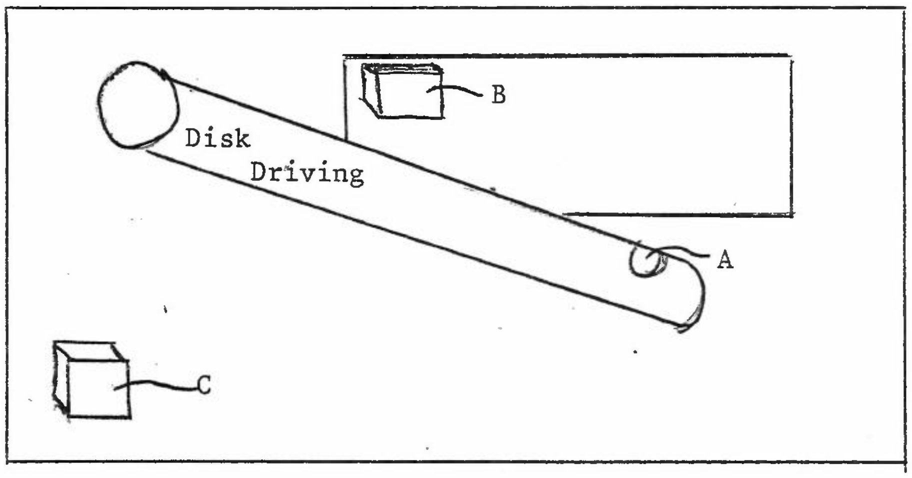
3. PLUG LAYOUT AT BOTTOM OF BACK



IMPORTANT: ALWAYS SWITCH OFF POWER BEFORE INSERTING OR REMOVING PLUG

Failure to do so can damage hardware!

4. INSTALLATION



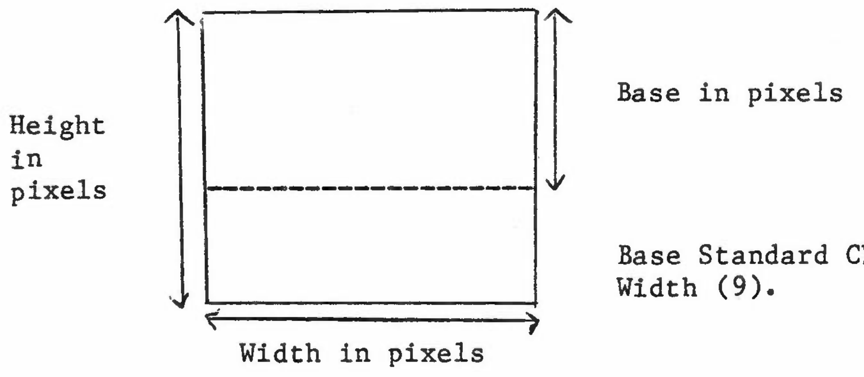
1. Remove disk transit screw A.
2. Connect +5V, +12V plug B.
3. Connect 60 Hz power plug C.
4. Check all PC boards correctly seated.
5. Check all device plugs correctly connected.
6. Switch on.
7. Leave for 2 minutes whilst disk runs up to speed.
8. Should boot automatically.
9. If not, hit boot button on back of keyboard.
10. Check LED display for 999 (ok) else error code.

ALWAYS SWITCH OFF POWER BEFORE INSERTING/REMOVING ANY PLUG

(SRC PERQ : Slot #9, Monitor SN102, Frame (by power plug) SN110, CPU serial no 124).

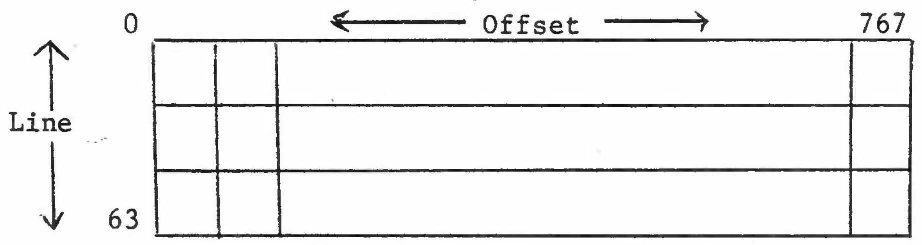
5. FONT LAYOUT

Font layout is given in Module Screen, together with standard font associated routines, cursor control and window management.



Base Standard Character is (13), Width (9).

Individual character definitions are stored as



<u>See RASTOP program</u>

Character patterns are stored in this way so that the width of the overall pattern area is the same (0 ... 767) as the screen. This is for Rasterop efficiency.

6. SEGMENTS AND POINTERS

Segment addresses are 16 bit segment numbers plus 16 bit offset within (2^{16} = 64K) segment. Hence a 32 bit virtual address is constructed. The Pascal pointer is implemented as this 32 bit virtual address. Sometimes one must use a pointer (normal Pascal usage) and sometimes one must use a segment number, segment offset form (rasterop). The relationship between them is:

$\longleftarrow$ 32 bit pointer or virtual address $\longrightarrow$

Segment number (16 bits)	Offset within segment (16 bits)

In Pascal the two equivalent forms are handled by a variant record

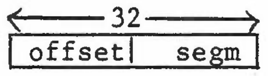
```
    Typptr = packed Record
                  case boolean of
                    true:  (segm,offset:integer);
                    false: (actptr     :somepointertype);
             end;
```

eg lines 19-22 in RASTOP program.

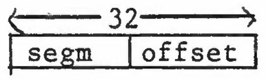
Line 145 of RASTOP shows the conventional pointer form in use (actptr) whilst line 148 shows the segment, offset form of the same virtual address.

It is important to realise that the following are not equivalent

(1) true : (segm:integer,offset:integer);



(2) true : (segm,offset:integer);



Due to the internal workings of the computer form (1) will produce the reverse ordering of the integers, and so cannot be used for virtual address equivalencing.

7. COMMAND WINDOW

The 'system' window is window zero.

8. RASTEROP

Alignment

Rasterop source and destinations must be quad word aligned because rasterop is 64 bit operation. Alignment is controlled by the extended NEW procedure. If the 2nd parameter is 4 then the data with the quad word aligned. Scan lines must be multiples of 4 words.

Efficiency

Rasterop is apparently more efficient if source and destination widths are multiples of four words and the same length. When rasteroping to/from screen it is apparently most efficient if both source and destination are 0 ... 767, ie 48 words wide. This is why font definitions are lines of 48 words (0 ... 767).

9. COMPILING LARGE PROGRAMS

Programs such as QUEEN require the magic words COM QUEEN/S:12. The default is 15. This must be reduced for large programs.

10. KNOWN COMPILER BUGS

(1) for i:= 1 to 32767 will loop for ever.

(2) end. needs a carriage return after the '.' else "unexpected eof" will occur.

(3) write(file,char) does not work if file is 'var' parameter.

(4) real numbers are not implemented. The compiler can handle them (in the latest version) but the microcode is still in preparation.

(Miles Barel has NPL validation suite. I left Unix Pascal VU tests).

11. KNOWN OPERATING SYSTEM BUGS

(1) If one does too many edits then the editor runs out of memory and crashes (eg replace all spaces in big file).

(2) DO NOT UNDER ANY CIRCUMSTANCES do a cntrl-C during the updating of a file by the editor (U) else one will have to hand patch the disk to recover.

(3) SSetCursor is suspect if $X \leq 10$.

12. Z80

Don Selza is responsible for the Z80 code. It is all in assembler. There are 2 x 4K PROMs but the highest address used is #10466. Might be feasible to put CRing code in Z80. Z80 physically located in I/O board.

13. THE WSORT PROGRAM

The WSORT program is a modification of Bill Sharpe's DSORT1 program which runs on the Terak and is a simple animation of a SORT procedure. DSORT1 has been modified in two ways to make it run on the PERQ. Firstly, the Terak intrinsic 'GotoXY' has been rewritten using the PERQ graphics intrinsics, and secondly, the WSORT procedure has now been made to run in two separate windows by illustrating the use of the rather primitive PERQ window manager. Given below is a commentary on the various changes need to make DSORT1 into WSORT.

Line 2

The import command is used to import type definitions and procedures into the user's program. It imports the module screeen (note that this is spelt with 3 e's because of a silly typing error by Three Rivers) from the file screen. Module screeen contains procedures for positioning the cursor, reading the cursor, setting the type of the cursor for creating and manipulating the windows and defining and changing fonts. See page 7 of the PERQ Operating System Programmer's Guide. Program WSORT uses 'set cursor' and 'create window', which are from module screeen.

Lines 11-15

Terak intrinsic 'move cursor routine' called 'gotoXY' is directly equivalent to the PERQ procedure 'setcursor' and so in WSORT 'gotoXY' is just expanded to be a call of 'setcursor'. The complex parameters to 'setcursor' just cope with the finer resolution of the PERQ and cope with the fact that WSORT actually runs the DSORT program in two separate windows, hence the REPT MOD 2 component which determines which window is used.

Line 83

Due to a deficiency in the existing PERQ Pascal system, all programs must include the statements 'reset(input)', 'rewrite(output)' as their first statements.

Line 85

Character #14 is the form feed character. Using the standard I/O procedure 'write' to output the form feed character clears the screen. In the context of the standard output channel, form feed clears the screen, end of line character advances to the next line, and back space erases the previous character (see module screeen).

Lines 90-94

The statements set up the extremities of the two windows in which the WSORT program will run. They are in absolute screen coordinates because the rather primitive screen manager only works in absolute screen coordinates.

Lines 96-103

This is a call to the screen manager which will create a new window which has a rectangular boarder with a black stripe at the top in which appears the title given on line 102. Its top left-hand corner will be WX1,WY1 and it will be WWD pixels wide and WXT pixels high. On line 97 the integer 1 is the window number. Window numbers are in the range 0-15. The WSORT program creates two slightly overlapping windows, one at line 96 and one at line 104.

Lines 113-126

DSORT1 program is repeated 4 times, twice in each window, creating a new window each time. After each window has been created the statement 'gotoXY (0,0) on line 124' causes the current cursor position to be moved to the origin, ie the top left-hand corner of the window. However, the screen manager is deficient in the sense that it does not automatically change the coordinate system when a new window is created. This means that anything that is written into a window must be absolute screen coordinates, hence the reason why the 'gotoXY (0,0)' which is relative to the window origin has to be transformed into absolute coordinates in the procedure 'gotoXY' so that 'setcursor' will work. See lines 11-15. Line 126, writing out the form feed character, will just clear the current window.

14. THE RASTOP PROGRAM

The Rastop program is a tutorial exercise in using the Rasterop function directly from Pascal. Rasterop is used to draw an arbitrary pattern and move it, and is also used to access and draw the standard font.

Lines 3-6

Various PERQ operating system modules are imported.

Lines 9-13

A 2-dimensional array will be used to hold the pattern which is moved around the screen by Rasterop. It is COLBIT pixels wide x ROWBIT pixels deep. Note how COLBIT is defined as a multiple of 16 x COLWORD to ensure that it is word aligned. For greatest efficiency, COLBIT should be a multiple of 64 to ensure quad-word alignment. PATLEN is a number of bits needed to store a font. Note that PATLEN is 768 (the width of the screen in pixels) divided by 16 to give the number of words in a scanline and multiplied by 64 which is the maximum of lines in the standard font definition (see the definition of fontptr in module screen.

Lines 19-22

FPTR is pointer type which shows how the 32 bit virtual address, which is the Pascal pointer, is equivalent to the 16 bit segment number plus 16 bit offset.

Lines 39-40

'Createsegment' is one of the Memory Management procedures from module memory. Its function is to return to the user program the segment number of a new segment which can be used for the user's data. The segment number of the new segment is given by the first parameter SEGNO. The three input parameters are integers which define the initial size of the segment as a multiple of 256 words, the increment by which the segment may be expanded in terms of 256 words to a maximum specified by the third parameter. Line 39 creates a new segment whose initial size is 2 x 256 words which may be incremented 512 words at a time up to a maximum of 5 x 256 words. The new segment created by 'create segment' can have data put into it via pointers which are created by an extended version of the new procedure described in the Pascal Extensions manual. It takes three parameters in the extended version. The first is the segment number to be operated on, the second parameter controls alignment, and the third output parameter is the traditional parameter, ie it is the traditional pointer parameter to be returned by the new procedure. The value of 4 ensures that the source pointer will point to an area which is quad-word aligned which is essential for Rasterop operations.

Lines 43-46

This piece of code just sets up the pattern which will be moved about by Rasterop. It defines a rectangular area which is comprised of a set of vertical stripes.

Lines 48-60

These statements set up the parameters for Rasterop. Rasterop is described in the Pascal Extensions manual. The destination for all Rasterop operations to come on the screen must use the operating system segment which contains the screen buffer. This is called SCREENSEG. Line 55 shows the use of a PERQ Pascal extension 'makepointer' which is a way of creating an arbitrary pointer from a segment number, so SCREENSEG is the segment number, 0 is the offset within the segment, and BLKPTR is the type of the pointer which is returned. Line 54 sets up the number of words in a scanline. This is 48 when dealing with the screen.

Lines 63-74

This Rasterop draws the pattern at the top of the screen and the delay at line 77 causes it to stay there for a few seconds. Note the use on line 64 of a neumonic to describe the Rasterop functions. These are imported from the module raster.

Lines 79-113

This piece of code causes the pattern to be moved from the top of the screen to the middle of the screen. It is essentially achieved by deleting the top line from the 50 which make up the pattern, and adding one to the bottom, which together with the delay at line 83 caused the pattern to move smoothly down the screen. Note how things are added and removed from the screen using the same function 'exclusive or'.

Lines 116-151

This section of the program is independent of the first section and shows how to manipulate the font. In places it contains alternative ways of doing things, hence certain things are commented out.

Line 116

This uses a procedure from the screen module to return a pointer to the current font. Note the use of the actptr discriminator to show that this is the 32 bit virtual memory conventional Pascal pointer operation.

Line 117

This is a call to the PERQ extended new operator to create a new data area of type patn (line 17) which is quad-word aligned (because the second parameter is 4) and which is in the 'default user data' segment because the segment number specified for the first parameter is 0.

Line 118

This operation copies the current font character definitions into newly created data area. Later on the code will show that the font can be painted on to the screen from either the system area of this newly created user area. This is shown in line 148 and 149. Line 130 is incorrect and should be disregarded. The Rasterop of line 119 paints the entire font up on the screen as two lines of text below which appears a lot of random dots. This is because the font actually needs less than 63 scanlines to define it all. From this operation it is clear that it is useful to define a font as being a set of rows which can be displayed across the screen as well as being efficient for Rasterop.

Lines 134-151

This loop paints each individual character on the screen just below the entire font and shows how to access individual characters. Line 136 is a delay just to allow the character to be clearly seen. Lines 139 and 140 show how the height and width of the character are used in the Rasterop operation. Line 141 is the absolute screen coordinate for the X position of the character. Line 142 shows the absolute screen coordinate in the Y direction for the character. Note how the base value is subtracted from the Y coordinate. This ensures that the Y coordinate is the bottom left position of the character. Note one uses base not height, because the height could be variable for a character with a decending tail. Line 144 'destptr' is the screen (see line 55). Line 145 is the position of the character's left-most pixel in the scan line. Line 146 computes which row of scan lines the character starts. Note that this is computed by looking up which line (first or second in a standard font) of the font the character appears in, and then multiplying this by height or number of pixels per line. Line 147 is the number of words in a scanline. This is 48 for the standard font definitions, the same as the screen. Lines 148 and 149 are alternatives. Line 148 makes a pointer to the actual system area containing the standard font. However, it uses the equivalence mechanism to refer to the current font pointer the segment and offset method rather than the Pascal pointer. The magic number octal 404 is added to the offset value because 404 words is the number of words to implement the index array which preceeds the patterns in core. If line 149 was used instead of 148 the source rectangle for Rasterop would be the copy of the system font which was made by lines 117 and 118.

15. THE QUEENS PROGRAM

This program is a modification of Janet Malone's animation of the QUEENS program which runs on the Terak. The program essentially shows how the Terak intrinsics 'gotoXY', 'drawline' and 'drawblock' can be implemented by the PERQ intrinsics 'setcursor', 'drawline' and 'rasterop'.

Lines 10-11

These two lines serve to import some definitions of segment numbers using the 'include' mechanism of the PERQ Pascal Compiler.

Line 28

The Terak program used blocks as a type which were passed to the Terak screen handling procedures. PERQ Pascal is a little bit more rigorous and requires pointers to be passed which point to blocks which have been quad-word aligned (see line 404).

Lines 29-33

This record type was defined in order to read in the photo file which gives the bit pattern to draw QUEEN. I never quite got the queen to come out right on the PERQ and I never found the reason for this, so this piece of code is suspect.

Lines 53-56

This shows how the Terak intrinsic 'gotoXY' for positioning the cursor is mimicked by the 'setcursor' routine. The 'setcursor' routine does not seem to work properly if the X coordinate is less than 10. I suspect this is just a bug as X < 10 is around one character width. X and Y are scaled and 'setcursor' works in absolute screen coordinates, not characters.

Lines 75-109

Terak intrinsic 'drawblock' is mimicked by Rasterop. Lines 89-96 essentially convert the 'drawblock' mode to its equivalent Rasterop function. The actual conversion in these few lines is probably not quite accurate as I could not remember all the exact values of 'drawblock' and Janet had very kindly used integers rather than mnemonics in her program. Line 95 shows the use of the PERQ Pascal extension to the case statement. For details of the Rasterop function see above. However, note line 104 making a pointer to the system screen buffer, and line 107 where the number of pixels in the scan line which, supplied by 'drawblock', is converted to words per scan line. The Terak procedure 'drawblock' does not take any cognisance of quad-word alignment and so the source parameter must be manipulated outside the 'draw block' procedure to ensure it is quad-word aligned. An alternative to this would be to copy the contents of the source to some quad-word aligned buffer before doing Rasterop, but this would seem to be a little slow.

Lines 114-118

The Terak delay intrinsic 'throttle' has been simulated by just wasting time.

Lines 122-145

The Terak intrinsic 'drawline' (converted to 'drawaline' here to avoid a name clash) is quite simply re-implemented using the PERQ line mechanism. This is defined in the line draw module.

Lines 187-198

The Terak routine 'fillchar' has no equivalent in PERQ Pascal and so lines 195-198 simulate its effects. These are a little tricky because the version on the PERQ comes out as the inverse of the Terak, ie everything that is black on the Terak is white on a PERQ. Note how lines 195-198 operate on 32 x 32 arrays to ensure word alignment.

Lines 208-211

These two commands clear the screen and set the Terak screen buffer to be 'screen'. In the PERQ version of the standard write command to clear the screen and the system screen buffer is always used rather than the user defined screen buffer, hence the use of 'makeptr (screenseg)' in the Rasterop operations.

Lines 229-235

This little bit of jiggery pokery reads in the Terak photo file which defines the bitmap of a Queen. I never did quite get this queen to come out properly on the PERQ screens. I am sure there is something wrong with this bit of code. The queen always came out with what looked like the left third of it missing which seemed to suggest some kind of word alignment problem and I never solved it. I am sure it is some stupid error somewhere.

Lines 400-404

Reset and rewrite must appear in every program. Line 401 clears the screen, and line 404 ensures that the various buffers used in Rasterop operations are quad-word aligned and appear in the user's default data segment segment number 0.

APPENDIX 1

NAME	EXTENSION	DEPARTMENT	ROOM
Barel, Miles	37	Software	302
Broadley, Bill	40	Hardware	307
Cercone, Richard	27	Drafting	205B
Crenner, Jim	22	Test Office	107
Eckenrode, Chuck	23	Test Area	106
Fredkin, Edward	34	C. E. O.	209
Gasbarro, Jim	41	Hardware	306
Gilmour, Tom	27	Drafting	205B
Glass, Bill	38	Software	301
Grandey, Connie	21	Assembly Area	104
Hahn, Andy	20	Shipping/Receiving	103
Harrington, John	25	Accounting	204
Heller-Hendrick, Jean	21	Assembly Office	105
Heller, Sandy	21	Assembly Area	104
Horner, Jake	21	Assembly Area	104
Horner, Michelle	20	Stockroom	103
Howell, Jeff	24	Manufacturing	204
Janets, Lynne	26	Bookkeeping	203
Letterle, Bruce	29	Personnel	201
Letterle, Karen	0	Receptionist	
Manko, Jack	30	Production	206
Marks, Diane	33	Secretary	208
Myers, Brad	37	Software	302
Newbury, Paul	35	President	210
Pe[illegible]er, Jeff	26	Software Support	203
Reddy, Pradeep	41	Hardware	306
Reid, Jill	20	Stockroom	103
Rose, John	40	Hardware	307
Rosen, Brian	31	Vice President	211
Scelza, Don	36	Software	303
Schultz, Dorothy	21	Assembly Area	104
Software Lab	39		308
Steele, Guy			
Stewart, Rege	23	Test Area	106
Strait, John	36	Software	303
Vavra, Terry	23	Test Area	106
Visnich, Donna	27	Drafting	205B
Williamson, Theresa	21	Assembly Area	104

APPENDIX 2

```
PROGRAM wsort(input,output);
imports screeen from screen; {ed}
const   max=11; firstrow=1; cx=35; cy=23; pfac=5; {ed}
var     n,srccol,dstcol,srcpos,dstpos,
        win,
        i,j,count,searchcount,a,b,total,rept :integer;
        col                             :array[1..3] of integer;
        list                            :array[1..max] of integer;
        wx1,wy1,wwd,wht                 :array[1..2]   of integer;
        change          : boolean;
procedure gotoxy(X,Y:integer);  {ed}
begin
        ssetcursor(X*10+wx1[(rept mod 2)+1]+30,
                   Y*13+wy1[(rept mod 2)+1]+50);
end;


procedure writetotal;
begin
        total:=total+1;
        gotoxy(cx+24,cy);
        write(total:2)
end;

procedure delay(m : integer);
var     i,j       : integer;
begin
         for i:=1 to m   do j:=0 ;
end;


procedure flash(column, row : integer) ; {draw attention to list item}
var     x,y     : integer;

begin
        x:=col[column]-4; y:=firstrow+(row-1)*2;
        gotoxy(x,y); write(chr(7),'>>>'); gotoxy(x,y);
        delay(2500*pfac);
        write('   ')
end;

procedure shift(n,srccol,srcpos,dstcol,dstpos : integer);
var     i,m,x1,x2,y1,y2       :integer;
begin
        m:=20; y1:=firstrow+(srcpos-1)*2;
        y2:=firstrow+(dstpos-1)*2;
        if dstcol>=srccol then
        begin
                x1:=col[srccol];
                x2:=col[dstcol];
                for i:=1 to n do
                begin
                        gotoxy(x1,y1); write(' ');
                        gotoxy(x2,y2); write('*');
                        delay(1*pfac);
                        x1:=x1+1; x2:=x2+1
                end
        end
        else
        begin
                x1:=col[srccol]+n-1;
                x2:=col[dstcol]+n-1;
                for i:=1 to n do
        begin
            gotoxy(x1,y1); write(' ');
            gotoxy(x2,y2); write('*');
            x1:=x1-1; x2:=x2-1;
            delay(1*pfac)
        end;
        gotoxy(x2,y2)

    end
end;

function next(n : integer) : integer;
{returns next nonnull position in list AFTER n}
begin
    next:=n+1;
    if n+1 <=max then if list[n+1]=0 then next:=next(n+1);
end;

begin
    reset(input); rewrite(output);  {ed}

    write(chr(#14));




     wx1[1]:=0;          wy1[1]:=0;
     wwd[1]:=700;        wht[1]:=500;

     wx1[2]:=50;         wy1[2]:=320;
     wwd[2]:=700;        wht[2]:=500;

     CreateWindow(
                 1,
                 wx1[1],
                 wy1[1],
                 wwd[1],
                 wht[1],
                 'win one'
                 );
     CreateWindow(
                 2,
                 wx1[2],
                 wy1[2],
                 wwd[2],
                 wht[2],
                 'win two'
                 );

     for rept := 1 to 4 do
      begin
      {ChangeWindow((rept mod 2)+1);}
      win:= (rept mod 2) + 1;
      CreateWindow(win,
                  wx1[win],
                  wy1[win],
                  wwd[win],
                  wht[win],
                  'window');

                  gotoxy(0,0);

      write(chr(#14));  {ed}
    col[1]:=10;
    col[2]:=30;
    col[3]:=50;
```

```
 list[1] :=3;
 list[2] :=4;
 list[3] :=2;
 list[4] :=5;
 list[5] :=1;
 list[6] :=4;
 list[7] :=8;
 list[8] :=6;
 list[9] :=9;
 list[10] :=10;
 list[11] :=7;
 for i:=1 to max do
         shift(list[i],1,i,1,i);
 total:=0; gotoxy(cx,cy); write('Number of comparisons = ');
 count:=max {must deal with all one at a time};
 repeat
 begin
         {find shortest}
         searchcount:=count-1 ;
         a:= next(0) {first non null list entry};
         b:=a;
         shift(list[a],1,a,2,a) {pick out};
         while searchcount<>0 do
         begin
                 delay(1000*pfac);
                 b:=next(b) {next non null entry};
                 shift(list[b],1,b,2,b) {pick out};
                 delay(1000*pfac);
                 change:=list[b] < list[a];
                 if change then
                           begin shift(list[a],2,a,1,a){putback a};
                                 a:=b
                           end
                           else shift(list[b],2,b,1,b){putback b} ;
                 searchcount:=searchcount-1;
                 writetotal
         end;
         delay(1000*pfac);

         {move shortest to end of sorted list}
         flash(2,a);
         shift(list[a],2,a,3,max-count+1);
         list[a]:=0 {delete from list};
         count:=count-1;
         delay(2500*pfac)
 end;
 until count=0;




end {of outer window changing loop} ;

end.

OK,
program rastop(input,output);

imports memory   from memory;
imports linedraw from linedraw;
imports raster   from raster;
imports screeen  from screen;


const
   rowbit=50;
   colwrd=20;
   colbit=colwrd*16;
   patlen=(768 div 16)*64;

type blk = packed array[1..rowbit,1..colbit] of boolean;
     blkptr = blk;
     patn   = array[0..patlen] of integer;
     patnptr = ^^patn;
     fptr = packed record case boolean of
             true:(segm,offs:integer);
             false:(actptr:fontptr);
            end;
  var
   segno,
   i,j,
   func,wid,hgt,
   destx,desty,destl,
   sorcx,sorcy,sorcl  :integer;
   destptr,sorcptr    :blkptr;
   cfptr   : fptr;
   {cfptr  : fontptr;}
   patptr : patnptr;

begin
   reset(input); rewrite(output);

   write(chr(#14));

   CreateSegment(segno,2,2,5);
   new(segno,4,sorcptr);


   for i:=1 to rowbit do
     for j:=1 to colbit do
       if odd(j) then sorcptr^^[i,j]:=true
                 else sorcptr^^[i,j]:=false;

   func:=0;
   wid :=colbit;
   hgt :=rowbit;

   destx:=100;
   desty:=1;
   destl:=768 div 16;
   destptr:=makeptr(screenseg,0,blkptr);

   sorcx:=0;
   sorcy:=0;
   sorcl:=colwrd;
   {sorcptr}


   RASTEROP
           (RXor,
              wid,
              hgt,
              destx,
              desty,
              destl,
              destptr,
              sorcx,
              sorcy,
              sorcl,
              sorcptr);


    for i:=1 to 30000 do j:=j;

    for desty:=1 to 500 do
      begin


      for i:=1 to 500 do j:=j;

      RASTEROP
             (RXor,
              wid,
              1,
              destx,
              desty,
              destl,
              destptr,
              sorcx,
              sorcy,
              sorcl,
              sorcptr);

      RASTEROP
             (RXor,
              wid,
              1,
              destx,
              desty+rowbit,
              destl,
              destptr,
              sorcx,
              sorcy,
              sorcl,
              sorcptr);



      end;


    cfptr.actptr:=GetFont;
    new(0,4,patptr);
    for i:=0 to patlen do patptr^^[i]:=cfptr.actptr^^.pat[i];
    RASTEROP
             (RRpl,
              768,
              63,
              0,
              desty,
              destl,
              destptr,
              0,
              0,
              48,
              {makeptr(0,cfptr.actptr^^.pat,LinePtr)}
```

```
            patptr
            );

    for i:=0 to #177 do
       begin
       for j:=1 to 30000 do destx:=destx;
       RASTEROP
           (RRpl,
            cfptr.actptr^^.index[i].width,
            cfptr.actptr^^.height,
            350,
            (desty+100)-cfptr.actptr^^.base,
            destl,
            destptr,
            cfptr.actptr^^.index[i].offset,
            cfptr.actptr^^.index[i].line*cfptr.actptr^^.height,
            48,
            makeptr(cfptr.segm,cfptr.offs+#404,fontptr)
            {patptr}
            );
       end;


end.

OK,
PROGRAM QUEENS;


imports linedraw from linedraw;
imports screeen from screen;




const
{$I Segnumbers}


TYPE TSCREEN = RECORD
         CASE INTEGER OF
             1 : (BITS : PACKED ARRAY[0..239] OF
                         PACKED ARRAY[0..319] OF BOOLEAN);
             2 : (CHRS : PACKED ARRAY[0..9599] OF CHAR)
     END;
     BLOCK = RECORD
         CASE INTEGER OF
             1 : (BITS : PACKED ARRAY[0..63] OF
                         PACKED ARRAY[0..63] OF BOOLEAN);

             2 : (CHRS : PACKED ARRAY[0..512] OF CHAR)
     END;

     blkptr = ^^block;
    queenin = record
                case integer of
                1:(bits:packed array[0..31,0..31] of boolean) ;
                2:(chrs:packed array[0..127] of char)
              end;


VAR DONE,SAFE : BOOLEAN;
          I,N : INTEGER;
            A : ARRAY[0..7] OF BOOLEAN; { rows }
            B : ARRAY[0..14] OF BOOLEAN; { / diagonals }
            C : ARRAY[-7..7] OF BOOLEAN; { \ diagonals }
          SOL : ARRAY[0..7] OF 0..7; { row posn. for each column }
       SCREEN : TSCREEN;
        BLACK,
  CHECK,QUEEN : blkptr;
          ANS : CHAR;
         quin : queenin;






procedure GOTOXY(x,y : integer);
   begin
      ssetcursor(x*10+10,y*13+500);
   end;


procedure FILLCHAR({blk:block.chrs;}
                   a,b:integer);
begin
end;

```

```
procedure UNITWRITE(a:integer; var scrn:tscreen; b:integer);
begin
end;






   PROCEDURE DRAWBLOCK(VAR SOURCE : blkptr;
                           SRCROW,
                             SRCX,
                             SRCY : INTEGER;
                       VAR   DEST : TSCREEN;
                           DSTROW,
                             DSTX,
                             DSTY,
                             CNTX,
                             CNTY,
                             MODE : INTEGER);

   var
      func:integer;
      begin
       case mode of
         0:func:=0;
         1:func:=1;
         2:func:=6;
         3:func:=4;
       otherwise:func:=mode;
       end;

       RASTEROP(func,
                cntx,
                cnty,
                dstx,
                dsty,
                48,
                makeptr(screenseg,0,Lineptr),
                srcx,
                srcy,
                (srcrow div 16),
                source);
      end;




    PROCEDURE THROTTLE(TICKS : INTEGER);
    var i,j:integer;
    begin
       for i:=1 to ticks*900 do j:=0 ;
    end;



    PROCEDURE DRAWALINE(VAR RANGE   : INTEGER;
                        VAR SCREEN : TSCREEN;
                          ROWWIDTH,
                            XSTART,
                            YSTART,
                            DELTAX,
                            DELTAY,
                          PENSTATE : INTEGER);
```

```

     var ltyp:Linestyle;
     begin
       case penstate of
         1:ltyp:=Drawline;
         2:ltyp:=EraseLine;
       otherwise:
           ltyp:=XorLine;
       end;
       Line(ltyp,
            xstart,
            ystart,
            xstart+deltax,
            ystart+deltay,
            makeptr(screenseg,0,LinePtr));
     end;





     PROCEDURE INTRO;
BEGIN
    WRITELN(
'                         THE QUEENS PROBLEM');
    WRITELN(
'                         ******************');
    WRITELN(
'    Given an N x N checkered board, the aim is to place N queens on the');
    WRITELN(
'board so that no queen may take another (i.e. none are on the same');
    WRITELN(
'diagonal, horizontal or vertical line).');
    WRITELN(
    'This may be solved recursively - for each column of the board, the');
    WRITELN(
'queen is moved down each square until a position is found where she is');
    WRITELN(
'safe. If there is no possible square in a column then we return to the');
    WRITELN(
'previous column and choose a new square - this is known as ');
    WRITELN(
'''backtracking''.');
    WRITELN(
'    When the last queen is placed safely on the last column then this is');
    WRITELN(
'a possible solution, but if all combinations of squares have been tried');
    WRITELN(
'and failed then there is no solution.')
END; { intro }
PROCEDURE INITIALISE;
TYPE BLOK = FILE OF queenin;
VAR DUMMY,
    I,j,X,Y : INTEGER;
    CROWN : BLOK;
    PROCEDURE SETCHECK; { creates a block of 24 x 24 checkered dots }
    BEGIN               { and sets up blank square }
        {
        FOR I := 0 TO 11 DO
        BEGIN
            FILLCHAR(CHECK.CHRS[I*8],4,85);    sets '. . .'
            FILLCHAR(CHECK.CHRS[I*8+4],4,170)  sets ' . . '
        END;
        FILLCHAR(BLACK,128,0)
        }
        for i:=0 to 31 do for j:=0 to 31 do
```

```
            if odd(j) then check^^.bits[i,j]:=odd(i)
                      else check^^.bits[i,j]:=not odd(i);
        for i:=0 to 31 do for j:= 0 to 31 do black^^.bits[i,j]:=false;
    END; {SETCHECK}
BEGIN { initialise }
        DONE := FALSE;
    {initialise arrays}
        FOR I := 0 TO N-1 DO SOL[I] := 0;
        FOR I := 0 TO N-1 DO A[I] := TRUE;
        FOR I := 0 TO 2*(N-1) DO B[I] := TRUE;
        FOR I := -(N-1) TO N-1 DO C[I] := TRUE;
    {draw board}
        {
        FILLCHAR(SCREEN,9600,0);
        UNITWRITE(3,SCREEN,63);
        }
        SETCHECK;
        FOR Y := 0 TO N-1 DO
        BEGIN
            IF ODD(Y) THEN X := 1
                      ELSE X := 0;
            REPEAT
               DRAWBLOCK(CHECK,32,0,0,SCREEN,320,24*X+125,24*Y+36,24,24,0);
               X := X+2
            UNTIL X > N-1
        END;
        FOR I := 0 TO 2 DO
        BEGIN
            DRAWALINE(DUMMY,SCREEN,20,122,33+I,24*N+5,0,1);
            DRAWALINE(DUMMY,SCREEN,20,122,36+24*N+I,24*N+5,0,1);
            DRAWALINE(DUMMY,SCREEN,20,122+I,36,0,24*N,1);
            DRAWALINE(DUMMY,SCREEN,20,125+24*N+I,36,0,24*N,1)
        END;
    { get queen character }
        RESET(CROWN,'rob.QU.FOTO');
        quin := CROWN^^;
        CLOSE(CROWN);
        for i:=0 to 31 do
          for j:=0 to 31 do
            queen^^.bits[i,j]:=quin.bits[i,j];

    { draw N queens above board }
        FOR I := 0 TO N-1 DO
            DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,I*24+125,9,24,24,0);
        THROTTLE(25);
    { write explanation }
        GOTOXY(0,14);
        WRITELN('LINES - SHOW WHICH QUEEN');
        WRITELN(' ':8,'WOULD TAKE THE NEW');
        WRITELN(' ':8,'QUEEN IF HER SQUARE');
        WRITELN(' ':8,'IS NOT SAFE.');
        WRITELN('**BACKTRACK** - THERE IS');
        WRITELN(' ':8,'NO SAFE SQUARE FOR');
        WRITELN(' ':8,'THIS COLUMN, SO WE');
        WRITELN(' ':8,'MOVE THE QUEEN IN');
        WRITELN(' ':8,'THE PREVIOUS COLUMN.');




    END; { initialise }
PROCEDURE TRYCOL(J:INTEGER);
VAR I,K,X,Y:INTEGER;
    PROCEDURE LINE; { draws lines showing if queen can be taken
                      if placed on this square }
    VAR DUMMY,DX,DY,X,Y : INTEGER;
```

```
BEGIN { line }
    IF NOT(A[I]) { rows }
    THEN BEGIN
        DRAWALINE(DUMMY,SCREEN,20,125,I*24+48,24*N,0,3);
        THROTTLE(25);
        DRAWALINE(DUMMY,SCREEN,20,125,I*24+48,24*N,0,3)
    END;
    IF NOT(B[I+J]) { / diagonals }
    THEN BEGIN
             DX := (N - (ABS(I+J-N+1)))*24;
             DY := -DX;
             IF I+J < N
             THEN BEGIN
                 X := 125;
                 Y := (I+J+1)*24+36
             END
             ELSE BEGIN
                 X := (I+J-N+1)*24+125;
                 Y := N*24+36
             END;
             DRAWALINE(DUMMY,SCREEN,20,X,Y,DX,DY,3);
             THROTTLE(25);
             DRAWALINE(DUMMY,SCREEN,20,X,Y,DX,DY,3)
     END;
     IF NOT(C[I-J]) { \ diagonals }
     THEN BEGIN
         DX := (N - (ABS(I-J)))*24;
         DY := DX;
         IF I-J < 0
         THEN BEGIN
             X := (J-I)*24+125;
             Y := 36
         END
         ELSE BEGIN
             X := 125;
             Y := (I-J)*24+36
         END;
         DRAWALINE(DUMMY,SCREEN,20,X,Y,DX,DY,3);
         THROTTLE(25);
         DRAWALINE(DUMMY,SCREEN,20,X,Y,DX,DY,3)
     END
END; { line }
PROCEDURE WIPEOUT(J,I:INTEGER); { removes queen character from square }
BEGIN
    IF ODD(I+J) THEN { black square }
        DRAWBLOCK(BLACK,32,0,0,SCREEN,320,24*J+125,24*I+36,24,24,0)
                ELSE { grey square }
        DRAWBLOCK(CHECK,32,0,0,SCREEN,320,24*J+125,24*I+36,24,24,0)
END; { wipeout }
PROCEDURE SETQUEEN; { sets row and diagonals as covered }
BEGIN
    A[I]:=FALSE; B[I+J]:=FALSE; C[I-J]:=FALSE
END; { setqueen }
PROCEDURE REMOVEQUEEN; { releases row and diagonals, signals
                         backtracking has occurred }
VAR K : INTEGER;
BEGIN
    A[I]:=TRUE; B[I+J]:=TRUE; C[I-J]:=TRUE;
    GOTOXY(0,J+2);
    WRITE('**BACKTRACK**');
    FOR K := 1 TO 12 DO
    BEGIN
        DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,24*J+125,24*I+36,24,24,2);
        THROTTLE(6)
    END;
    GOTOXY(0,J+2);
```

```
        {WRITE(CHR(29))}
        for k:=1 to 13 do write(' ');

    END; { removequeen }
    PROCEDURE LASTCOLUMN; { a solution has been found. If another
              solution is not required then the flag DONE is set }
    BEGIN
        GOTOXY(0,9);
        WRITELN('WOULD YOU LIKE ANOTHER');
        WRITELN('SOLUTION FOR THIS BOARD? (Y/N)');
        READLN(ANS);
        GOTOXY(0,9);
        WRITELN(CHR(29));
        WRITELN(CHR(29));
        WRITELN(CHR(29));
        IF ANS <> 'Y' THEN DONE := TRUE
                      ELSE { remove queen }
                          A[I]:=TRUE; B[I+J]:=TRUE; C[I-J]:=TRUE
    END; { lastcolumn }
    PROCEDURE MOVEDOWN; { moves queen down one square of board }
    BEGIN
        WIPEOUT(J,I);
        DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,24*J+125,24*I+48,24,24,3);
        THROTTLE(6);
        WIPEOUT(J,I);
        WIPEOUT(J,I+1);
        DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,24*J+125,24*(I+1)+36,24,24,3);
        THROTTLE(6)
    END; { movedown }
BEGIN { trycol }
    { remove queen from side of board and place on first square }
        THROTTLE(25);
        DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,J*24+125,9,24,24,2);
        DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,24*J+125,36,24,24,3);
    { write next line of solution }
        GOTOXY(0,J+1);
        WRITELN('(',J+1:1,', )');
    I := 0;
    REPEAT
        { fill in row of solution }
            GOTOXY(3,J+1);
            WRITE(I+1:1);
        SAFE:= A[I] AND B[I+J] AND C[I-J];
        IF SAFE
            THEN BEGIN
                     SETQUEEN;
                     SOL[J]:=I;
                     IF J < N-1
                         THEN BEGIN
                                  TRYCOL(J+1);
                                  IF NOT DONE THEN REMOVEQUEEN
                              END
                         ELSE LASTCOLUMN
                     END
             ELSE LINE;
             { move queen down 1 square, or replace above board }
                 IF (I < N-1) AND NOT DONE
                 THEN MOVEDOWN
                 ELSE IF (I = N-1) AND NOT DONE
                      THEN BEGIN
                          WIPEOUT(J,I);
                          DRAWBLOCK(QUEEN,32,0,0,SCREEN,320,J*24+125,
                                    9,24,24,0);
                          { clear solution line }
                              GOTOXY(0,J+1);
                              WRITE(CHR(29));
                          THROTTLE(25)
                      END;
             I := I + 1
    UNTIL (I=N) OR DONE
END; { trycol }
BEGIN { main }
    reset(input);  rewrite(output);
    write(chr(#14));
    gotoxy(0,0);

    new(0,4,queen); new(0,4,check); new(0,4,black);

    INTRO;
    REPEAT
        WRITELN('PLEASE TYPE IN N, THE SIZE');
        WRITELN('OF THE BOARD (BETWEEN 1 AND 8)');
        READLN(N);
        IF (N<1) OR (N>8)
        THEN WRITELN('BOARD SIZE SHOULD BE BETWEEN 1 AND 10')
        ELSE BEGIN
                 {PAGE(OUTPUT);} write(chr(#14));
                 INITIALISE;
                 GOTOXY(0,0);
                 WRITELN('SOLUTION FOR ',N,' BY ',N,' BOARD :');
                 TRYCOL(0);
                 IF NOT DONE THEN WRITELN('NO SOLUTION')
        END;
        GOTOXY(0,30);
        WRITELN('DO YOU WANT TO CONTINUE? (Y/N)');
        READLN(ANS)
    UNTIL ANS <> 'Y'
END. { main }

OK,
```