```
 1:
 2: ****************************************
 3: ./part6.f90
 4:
 5: PROGRAM part6
 6:
 7:   USE part6_mod
 8:
 9:   IMPLICIT NONE
10:
11:   INTEGER, PARAMETER :: m=4,n=4
12:   REAL, DIMENSION(m,n) :: mat1
13:   REAL, DIMENSION(n,m) :: mat2
14:
15:
16:   CHARACTER(LEN=*), PARAMETER :: fname='mat.txt'
17:   CHARACTER(LEN=*), PARAMETER :: fname2='mat.dat'
18:
19:   !** We assume a file "mat.txt" has been created and exists
20:   !** inside the current working directory.
21:
22:   mat1=getmatf(m,n,fname)   !*** Read in matrix from txt file
23:   mat2=transpose(mat1)      !*** Take the transpose and place in mat2
24:   CALL outmatf(mat2,fname)  !*** Write out transpose to same file
25:
26:   mat1=0  !*** Destroy mat1
27:
28:   CALL outmatf_u(mat2,fname2) !*** Output mat2 to binary file
29:   mat1=getmatf_u(m,n,fname2)  !*** Read from binary file into mat1
30:
31:   CALL outmat(mat1) !*** output new mat1 to the screen
32:
33:
34:
35: END PROGRAM part6
36:
37: ****************************************
38:
39:
40: ****************************************
41: ./part6_mod.f90
42:
43: MODULE part6_mod
44:
45:   IMPLICIT NONE
46:
47:   CONTAINS
48:
49:   ! ************************************************************************
50:
51:   FUNCTION infnorm(vec)
52:   !** Calculates the infinity norm of vector vec.
53:     !*** Dummy declarations
54:     REAL, DIMENSION(:), INTENT(IN)  :: vec
55:     !*** Local declarations
56:     REAL :: infnorm
57:
58:     infnorm=MAXVAL(ABS(vec))
59:
60:   END FUNCTION infnorm
61:
62:  ! ************************************************************************
63:
64:   FUNCTION twonorm(vec)
65:   !** Calculates the Euclidean norm of vector vec.
66:     !*** Dummy declarations
67:     REAL, DIMENSION(:), INTENT(IN)  :: vec
```



```
68:     !*** Local declarations
69:     REAL :: twonorm
70:
71:     twonorm=SQRT(SUM(vec**2))
72:
73:   END FUNCTION twonorm
74:
75:  ! ************************************************************************
76:
77:   FUNCTION cont(y,x,tol,max_iters)
78:   !*** Returns a logical type. If .TRUE. then the tolerance has been met or
79:   !*** the maximum number of iterations has been exceeded.
80:
81:     !*** Dummy variables
82:     REAL, DIMENSION(:), INTENT(IN) :: y,x
83:     REAL, INTENT(IN) :: tol
84:     INTEGER, INTENT(IN) :: max_iters
85:     !*** Local declarations
86:     LOGICAL,DIMENSION(2) :: cont
87:     INTEGER, SAVE :: iter=0
88:
89:     cont(1)=(infnorm(ABS(y-x)) < tol)
90:     cont(2)=(iter>max_iters)
91:     iter=iter+1
92:
93:   END FUNCTION cont
94:
95:   ! ************************************************************************
96:
97:   FUNCTION power1(mat,x,tol,eigv,flag,max_iters)
98:   !**** Func. to calc. the dominant eigenvalue and corresponding normalised
99:   !**** eigenvector of the (n) by (n) matrix [mat] the initial guess is the
100:  !**** input vector (x) and the method is considered to have converged if
101:  !**** absolute error is less than the given tolerance (tol). The
102:  !**** eigenvalue is returned in (eigv) and the eigenvector is
103:  !**** returned in x
104:
105:    !**** Dummy arguments
106:    REAL, DIMENSION(:,:), INTENT(IN)  :: mat
107:    REAL, DIMENSION(:), INTENT(INOUT)  :: x
108:    REAL, INTENT(IN) :: tol
109:    REAL, INTENT(OUT) :: eigv
110:    LOGICAL,DIMENSION(:), INTENT(OUT) :: flag
111:    INTEGER :: max_iters
112:
113:
114:    !**** Local Declarations
115:    REAL, DIMENSION(SIZE(x))  :: y
116:    INTEGER :: power1
117:    INTEGER, DIMENSION(1) :: loc
118:
119:    power1=0
120:    flag=.FALSE.
121:    !**** Normalise initial estimate
122:    x=x/infnorm(x)
123:
124:    DO                              !** Iterate while "flag" is .TRUE.
125:      y=mulmatvec(mat,x)            !** Calculate new eigenvector estimate "y"
126:      loc=MAXLOC(ABS(y))            !** Location of largest magnitude in ABS()
127:      eigv=y(loc(1))/x(loc(1))      !** Estimate eigenvalue
128:      y=y/infnorm(y)                !** Normalise new eigenvector estimate
129:      flag=cont(y,x,tol,max_iters) !** Decide if another iteration is needed
130:      x=y                           !** Update eigenvector "x" for next iteration
131:      power1=power1+1
132:      IF (flag(1) .OR. flag(2))  EXIT !** EXIT loop if any flag is true
133:    ENDDO
134:
```

```
  135:    END FUNCTION power1
  136:
  137: ! ********************************************************************
  138:
  139: FUNCTION getmat(m,n)
  140: !**** Function to input a matrix from the keyboard. The number of rows
  141: !**** (m) and the number of columns (n) are input arguments to the
  142: !**** function
  143:
  144:   INTEGER, INTENT(IN) :: m,n       !**** Dummy declaration
  145:   REAL, DIMENSION(m,n) :: getmat   !**** Local Declaration
  146:
  147:   INTEGER :: i
  148:
  149:   DO i=1,m
  150:     PRINT '("Enter matrix row :",i2)',i !*** Prompt for row number
  151:     READ*,getmat(i,:)                 !*** Read in row
  152:   ENDDO
  153:
  154: END FUNCTION getmat
  155:
  156: ! ********************************************************************
  157:
  158: SUBROUTINE outmat(mat)
  159: !**** Subroutine to output a matrix to the screen.
  160:
  161:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat  !*** Dummy declaration
  162:
  163:   INTEGER :: i
  164:
  165:   DO i=1,SIZE(mat,1)
  166:     PRINT*,mat(i,:)
  167:   ENDDO
  168:
  169: END SUBROUTINE outmat
  170:
  171: ! ********************************************************************
  172:
  173: FUNCTION mulmat(mat1,mat2)
  174: !**** Function to input two matrices [mat1] & [mat2] and check if
  175: !**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
  176: !**** matrix product [mat1]*[mat2] is returned.
  177:
  178:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat1
  179:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat2
  180:
  181:   INTEGER :: m,n,k,i,j,p
  182:
  183:   REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2))  :: mulmat
  184:
  185:   m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)
  186:
  187:   !**** Perform the matrix multiplication
  188:   !**** using three DO loops
  189:
  190:   IF (SIZE(mat2,1) == n) THEN
  191:     DO i=1,m               !*** For each row of getmat (mat3)
  192:       DO j=1,k             !*** For each column of getmat (mat3)
  193:         mulmat(i,j)=0      !*** initialise to zero
  194:         DO p=1,n
  195:           mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
  196:         ENDDO
  197:       ENDDO
  198:     ENDDO
  199:   ELSE
  200:     PRINT*,"Size mismatch in mulmat"
  201:   ENDIF
```



```
  202:
  203: END FUNCTION mulmat
  204:
  205: ! ********************************************************************
  206:
  207: FUNCTION mulmatvec(mat,vec)
  208: !**** Function to input a matrix [mat] and a vector [v] check if matrix
  209: !**** vector multiplication is valid  w.r.t. their sizes. If it is then
  210: !**** this function returns the matrix vector product.
  211:
  212:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  213:   REAL, DIMENSION(:), INTENT(IN)  :: vec
  214:
  215:   INTEGER :: m,n,k,i,j
  216:   REAL, DIMENSION(SIZE(mat,1))  :: mulmatvec
  217:
  218:   m=SIZE(mat,1) ; n=SIZE(mat,2) ; k=SIZE(mat,1)
  219:
  220:   !**** Perform the matrix multiplication using three DO loops
  221:   IF (n==k) THEN
  222:     DO i=1,m
  223:       mulmatvec(i)=0
  224:       DO j=1,k
  225:           mulmatvec(i)=mulmatvec(i)+mat(i,j)*vec(j)
  226:       ENDDO
  227:     ENDDO
  228:   ELSE
  229:     PRINT*,"Size mismatch in mulmatvec!"
  230:   ENDIF
  231:
  232: END FUNCTION mulmatvec
  233:
  234: ! ********************************************************************
  235:
  236: FUNCTION transmat(mat)
  237:
  238:   !**** Dummy arguments
  239:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  240:   INTEGER :: m,n,i,j
  241:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat !*** Return
  242:
  243:   !**** Findout the no. of rows and cols in the matrix
  244:   m=SIZE(mat,1) ; n=SIZE(mat,2)
  245:
  246:   !**** Perform the transpose using two DO loops
  247:   DO i=1,n             !*** Loop over rows in mat
  248:     DO j=1,m           !*** Loop over cols in mat
  249:       transmat(i,j)=mat(j,i)
  250:     ENDDO
  251:   ENDDO
  252:
  253: END FUNCTION transmat
  254:
  255:
  256: ! ********************************************************************
  257:
  258: FUNCTION transmat2(mat)
  259:
  260:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat
  261:
  262:   REAL, DIMENSION(SIZE(mat,2),SIZE(mat,1))  :: transmat2
  263:
  264:
  265:   transmat2=RESHAPE(mat,(/SIZE(mat,2),SIZE(mat,1)/),ORDER=(/2,1/))
  266:
  267: END FUNCTION transmat2
  268:
```

```
269: ! *****************************************************************
270:
271:
272:
273: FUNCTION getmatf(m,n,filename)
274: !**** Function to input a matrix from a text file. The number of rows
275: !**** (m) and the number of columns (n) are input arguments to the
276: !**** function along with the name of the file
277:
278:   INTEGER, INTENT(IN) :: m,n              !**** Dummy declaration
279:   CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
280:
281:   REAL, DIMENSION(m,n) :: getmatf   !**** Local Declaration
282:   INTEGER :: i
283:
284:   !***** Open the file passed in as the string "filename" on unit one
285:
286:   OPEN(UNIT=1,FILE=filename, FORM="FORMATTED",STATUS="OLD",ACTION="READ")
287:
288:   DO i=1,m !**** Do for each row
289:     READ(UNIT=1,FMT=*) getmatf(i,:) !*** Read in row at a time
290:   ENDDO
291:
292:   CLOSE(UNIT=1) !***** Close the file
293:
294: END FUNCTION getmatf
295:
296: ! *****************************************************************
297:
298: SUBROUTINE outmatf(mat,filename)
299: !**** Function to output a matrix to a text file. The matrix to be
300: !**** written and the name of the file are the arguments
301:
302:
303:   CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
304:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat   !**** Dummy Declaration
305:
306:   INTEGER :: m,i
307:
308:   m=SIZE(mat,1)
309:
310:   !***** Open the file passed in as the string "filename" on unit one
311:
312:   OPEN(UNIT=1,FILE=filename, FORM="FORMATTED",STATUS="REPLACE",ACTION="WRITE")
313:
314:   DO i=1,m !**** Do for each row
315:     WRITE(UNIT=1,FMT=*) mat(i,:) !*** Write out row at a time
316:   ENDDO
317:
318:   CLOSE(UNIT=1) !***** Close the file
319:
320: END SUBROUTINE outmatf
321:
322:
323: ! *****************************************************************
324:
325:
326:
327: FUNCTION getmatf_u(m,n,filename)
328: !**** Function to input a matrix from a binary file. The number of rows
329: !**** (m) and the number of columns (n) are input arguments to the
330: !**** function along with the name of the file
331:
332:   INTEGER, INTENT(IN) :: m,n              !**** Dummy declaration
333:   CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
334:
335:   REAL, DIMENSION(m,n) :: getmatf_u  !**** Local Declaration
```



```
336:
337:   !***** Open the file passed in as the string "filename" on unit one
338:   OPEN(UNIT=1,FILE=filename, FORM="UNFORMATTED",STATUS="OLD",ACTION="READ")
339:
340:   READ(UNIT=1) getmatf_u !*** Read WHOLE array
341:
342:   CLOSE(UNIT=1) !***** Close the file
343:
344: END FUNCTION getmatf_u
345:
346: ! *****************************************************************
347:
348:
349: SUBROUTINE outmatf_u(mat,filename)
350: !**** Function to output a matrix to a binary file. The matrix to be
351: !**** written and the name of the file are the arguments
352:
353:
354:   CHARACTER(LEN=*), INTENT(IN) :: filename !**** Dummy declaration
355:   REAL, DIMENSION(:,:), INTENT(IN)  :: mat   !**** Dummy Declaration
356:
357:   !***** Open the file passed in as the string "filename" on unit one
358:
359:   OPEN(UNIT=1,FILE=filename, FORM="UNFORMATTED",STATUS="REPLACE",ACTION="WRITE")
360:
361:   !*** Write out whole array as no need now to write row by row
362:   !*** as we can not look at binary files like we can text files
363:
364:   WRITE(UNIT=1) mat !*** Write out WHOLE array
365:
366:   CLOSE(UNIT=1) !***** Close the file
367:
368: END SUBROUTINE outmatf_u
369:
370:
371: END MODULE part6_mod
372: ****************************************
373:
```