

%%%

QUESTION ONE

%%%

=====

Part (a)

=====

=====

MODULE cat_mod

IMPLICIT NONE

CONTAINS

! *****

FUNCTION getmat(m,n)

!**** Function to input a matrix from the keyboard. The number of rows

!**** (m) and the number of columns (n) are input arguments to the

!**** function

INTEGER, INTENT(IN) :: m,n !**** Dummy declaration

REAL, DIMENSION(m,n) :: getmat !**** Local Declaration

!** Local Declarations

INTEGER :: i

DO i=1,m

PRINT '("Enter matrix row :",i2)',i !** Prompt for row number

READ*,getmat(i,:) !** Read in row

ENDDO

END FUNCTION getmat

! *****

SUBROUTINE outmat(mat)

!**** Subroutine to output a matrix to the screen.

REAL, DIMENSION(:,:), INTENT(IN) :: mat !** Dummy declaration

!** Local Declarations

INTEGER :: i

DO i=1,SIZE(mat,1)

PRINT*,mat(i,:)

ENDDO

END SUBROUTINE outmat

! *****

FUNCTION hconcat(mat1,mat2)

!** Concatenate matrices horizontally

!** Told in class to check for conformity of matrix operations

REAL, DIMENSION(:,:), INTENT(IN) :: mat1

REAL, DIMENSION(:,:), INTENT(IN) :: mat2

!** Local Declarations

INTEGER :: m,i

```
REAL, DIMENSION(SIZE(mat1,1), SIZE(mat1,2)+SIZE(mat2,2)) :: hconcat

m=SIZE(mat1,1)

IF (m==SIZE(mat2,1)) THEN
  !** Check matrices conform for this operation
  DO i=1,m
    hconcat(i,:)=(/ mat1(i,:), mat2(i,:) /)
  END DO
ELSE
  PRINT*,"Matrices do not conform for hconcat"
ENDIF

END FUNCTION hconcat

! *****

FUNCTION vconcat(mat1,mat2)
  !** Concatenate matrices vertically
  !** Told in class to check for conformity of matrix operations

  REAL, DIMENSION(:,,:), INTENT(IN) :: mat1
  REAL, DIMENSION(:,,:), INTENT(IN) :: mat2

  INTEGER :: n,j

  REAL, DIMENSION(SIZE(mat1,1)+SIZE(mat2,1), SIZE(mat1,2)) :: vconcat

  n=SIZE(mat1,2)

  IF (n==SIZE(mat2,2)) THEN
    !** Check matrices conform for this operation
    DO j=1,n
      vconcat(:,j)=(/ mat1(:,j), mat2(:,j) /)
    END DO
  ELSE
    PRINT*,"Matrices do not conform for hconcat"
  ENDIF

END FUNCTION vconcat

! *****

END MODULE cat_mod

=====

Part (b)

=====

PROGRAM concat
!* Main program unit code to horizontally
!* concatenate two conforming matrices.

USE cat_mod

IMPLICIT NONE

!** Declare required data structures
INTEGER, PARAMETER :: m=3,n=4,k=2
REAL, DIMENSION(m,n) :: mat1
```

```

REAL, DIMENSION(m,k) :: mat2
REAL, DIMENSION(m,n+k) :: mat3

```

```

! ** Read matrices in from the keyboard.
mat1=getmat(m,n)
mat2=getmat(m,k)

```

```

! ** Output matrices to the screen
CALL outmat(mat1)
CALL outmat(mat2)

```

```

! ** Concatenate matrices
mat3=hconcat(mat1,mat2)

```

```

! ** Output result to the screen
CALL outmat(mat3)

```

END PROGRAM concat

=====

%%%

QUESTION TWO

%%%

```

=====
Part (a)
=====

```

To make use of the FORTRAN "USE" statement ie to make all entities of a module called 'matrix' available to an external program unit simply place

USE matrix

at the top of the program unit.

To restrict the entities use the 'only' attribute ie.

USE matrix, ONLY : gauss, power

Name conflicts can be dealt with by using the 'USE' rename facility. ie

USE matrix, power => pow1

renames "pow1" in the module to power locally

(or)

USE matrix, ONLY : power => pow1

where pow1 is the MODULE entity and will be used as pow1 in the unit using the matrix module

```

=====
Part (b)
=====

```

=====

```
PROGRAM quad_complex
```

```
!*** Program to calculate the roots of a quadratic
!*** using the more general COMPLEX data type
```

```
IMPLICIT NONE
```

```
REAL      :: a,b,c, xturn, yturn
COMPLEX  :: root1,root2,sqdiscrim
LOGICAL  :: check
```

```
! ** Enter the coefficients a,b,c
PRINT*, "Enter A, b, and c of the &
      &polynomial ax**2 + bx + c:"
```

```
READ*,a,b,c !*** Read in coefficients from keyboard
```

```
check= .NOT. a==0 !** Set check to .TRUE. if a valid quadratic
```

```
IF (check) THEN
```

```
! ** Calculate the sqrt of discriminant as a complex number
sqdiscrim=SQRT(CMPLX(b**2 - 4.0*a*c))
```

```
root1=(-b + sqdiscrim)/(2.0*a) !** Calculate root1
root2=(-b - sqdiscrim)/(2.0*a) !** Calculate root2
```

```
PRINT*, 'The roots are:'
PRINT*, "Root1 : ", "Real Part=", REAL(root1), &
      " : Imaginary part=", AIMAG(root1)
PRINT*, "Root2 : ", "Real Part=", REAL(root2), &
      " : Imaginary part=", AIMAG(root2)
```

```
! ** Calculate the quadratics turning point
```

```
xturn=-b/(2*a)
yturn=-b*b/(4*a)+c
PRINT*, "Turning Point = (x,y) = (", xturn, ", ", yturn, ")"
```

```
! ** Calculate Max or Min point of
```

```
IF (a .LT. 0) THEN
  PRINT*, "Turning point is a maximum"
ELSE
  PRINT*, "Turning point is a minimum"
ENDIF
ELSE
  PRINT*, "This is not a valid quadratic"
ENDIF
```

```
END PROGRAM quad_complex
```

```
=====
```

```
=====  
Part (c)(i)  
=====
```

```
=====
```

```
FUNCTION mulvecmat(vec,mat)
```

```
!**** Function to premultiply mat with vec
```

```
!**** Use assumed shape arrays for dummy arrays****
```

```
REAL, DIMENSION(:), INTENT(IN) :: vec !**** Dummy declaration
```

```
REAL, DIMENSION(:,,:), INTENT(IN)  :: mat  !*** Dummy declaration

!***** Local Declarations *****
INTEGER  :: j,m,n
REAL, DIMENSION(SIZE(mat,2))  :: mulvecmat

!***** Find out the matix size for the DO loop limit *****

n=SIZE(mat,2)
m=SIZE(mat,1)

!**** Perform the multiplication

IF (SIZE(vec) == m) THEN
  DO j=1,n          !*** For each element of mulvecmat
    mulvecmat(j)=SUM(vec*mat(:,j))
  ENDDO
ELSE
  PRINT*,"Size mismatch in mulvecmat"
ENDIF

END FUNCTION mulvecmat

=====

=====
Part (c)(ii)
=====

=====

FUNCTION mulmat(mat1,mat2)
!**** Function to input two matrices [mat1] & [mat2] and check if
!**** [mat1]*[mat2] is a valid matrix multiplication. If it is valid the
!**** matrix product [mat1]*[mat2] is returned.

REAL, DIMENSION(:,,:), INTENT(IN)  :: mat1
REAL, DIMENSION(:,,:), INTENT(IN)  :: mat2

INTEGER  :: m,n,k,i,j,p

REAL, DIMENSION(SIZE(mat1,1),SIZE(mat2,2))  :: mulmat

m=SIZE(mat1,1) ; n=SIZE(mat1,2) ; k=SIZE(mat2,2)

!**** Perform the matrix multiplication
!**** using three DO loops

IF (SIZE(mat2,1) == n) THEN
  DO i=1,m          !*** For each row of mulmat
    DO j=1,k        !*** For each column of mulmat
      mulmat(i,j)=0  !*** initialise to zero
      DO p=1,n
        mulmat(i,j)=mulmat(i,j)+mat1(i,p)*mat2(p,j)
      ENDDO
    ENDDO
  ENDDO
ELSE
  PRINT*,"Size mismatch in mulmat"
ENDIF

END FUNCTION mulmat

=====

=====
Part (c)(iii)
=====
```

```
=====
```

```
=====
```

```
mulmat2(i,j)=SUM(mat1(i,:)*mat2(:,j))
```

also note we no longer need to initialise "mulmat(i,j)=0" as there will no longer be a running summation in the code.

```
=====
```

```
=====
```

```
Part (c)(iv)
```

```
=====
```

No it could not because the "mulmat" function will only accept 2-dimensional arrays as arguments.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

QUESTION THREE

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
=====
```

```
MODULE ode_mod
```

```
!****
!**** Module file to hold Milne's method for solving
!**** ordinary differential equations.
!****
```

```
IMPLICIT NONE
```

```
CONTAINS
```

```
!*****
```

```
SUBROUTINE rk4(y,x,h)
```

```
!**** Does step of Runge-Kutta 4th Method ****
```

```
! *** Dummy declarations ***
```

```
REAL, INTENT(IN) :: h
```

```
REAL, INTENT(INOUT) :: y,x
```

```
REAL :: k1,k2,k3,k4
```

```
!**** Calculate k values ****
```

```
k1=h*ode(x,y)
```

```
k2=h*ode(x+h/2,y+k1/2)
```

```
k3=h*ode(x+h/2,y+k2/2)
```

```
k4=h*ode(x+h,y+k3)
```

```
!**** Calculate y(x) ****
```

```
y=y+(k1+2*k2+2*k3+k4)/6
```

```
x=x+h
```

```
END SUBROUTINE rk4 !***** [8 Marks]
```

```
!*****
```

```
SUBROUTINE hamm(y,x,h)
```

```
!***** Does one step of Hamming's Method *****
```

```
! *** Dummy declarations ***
```

```
REAL, INTENT(IN) :: h
```

```
REAL, INTENT(INOUT) :: y,x
```

```
! *** Local declarations ***
```

```
REAL :: py,y_in
```

```
INTEGER, SAVE :: count=1
```

```
REAL, SAVE :: oy1,oy2,oy3
```

```
y_in=y !** Make a copy of y_{n}
```

```
If (count < 4) THEN !** If not enough starting values
```

```
CALL rk4(y,x,h) !** Do Runge-Kutta 4th order
```

```
count=count+1 !** Increment count
```

```
ELSE !** Do the Hamming's method
```

```
py=oy3+4.0*h*(2*ode(x,y)-ode(x-h,oy1)+2*ode(x-2*h,oy2))/3
```

```
y=(9*y-oy2)/8+3*h*(ode(x+h,py)+2*ode(x,y)-ode(x-h,oy1))/8
```

```
x=x+h !** increment x
```

```
ENDIF
```

```
oy3=oy2 !** Update y_{n-3}
```

```
oy2=oy1 !** Update y_{n-2}
```

```
oy1=y_in !** Update y_{n-1}
```

```
END SUBROUTINE hamm !***** [10 Marks]
```

```
!*****
```

```
FUNCTION ode(x,y)
```

```
!** Function to return the value of the the differential
```

```
!** equation for a given x and y
```

```
!*** Dummy declarations
```

```
REAL, INTENT(IN) :: x,y
```

```
!*** Local declarations
```

```
REAL :: ode
```

```
ode=y-x
```

```
END FUNCTION ode !***** [2 Marks]
```

```
!*****
```

```
END MODULE ode_mod
```

```
=====
```