6. Arrays
• Declaring arrays

real, dimension(100) :: a

real :: a(100)

real :: a(1:100) ! same as a(100)

real :: a(-39:60)

integer, parameter :: isize = 100
real :: a(isize), b(isize), c(isize)

character(len=20), dimension(100) :: last_name
• Initialization

```
real :: a(5)
do i = 1, 5
  a(i) = i
end do
```

```
real :: a(5)
a = (/ 1.0, 2.0, 3.0, 4.0, 5.0 /) ! array constant
```

```
real :: a(5) = (/1.0, 2.0, 3.0, 4.0, 5.0/)  
```

```
real :: a(5) = (/ (i, i = 1, 5) /) ! implied DO loop
```

```
integer :: a(25)=(/ ( (0,i=1,4), 5*j, j=1,5 ) /)
a(i) = 0 0 0 0 5 0 0 0 0 10 0 0 0 0 15 0 0 0 0 20 0 0 0 0 25
```

```
real :: a(100)= 1.0
```
Example: Finding the largest and smallest values in a data set.

```fortran
program extremes
! Find the largest and smallest values in a data file
implicit none
! List of parameters:
integer,parameter :: max_size = 100   ! Max size of data set
! List of variables:
integer :: input(max_size)    ! Input values
integer :: ilarge             ! Pointer to largest value
integer :: ismall             ! Pointer to smallest value
integer :: j                  ! Do loop index
integer :: nvals              ! Number of data in data set

! Read nvals in fort.1 and check if the number <= max_size
read(1,*) nvals
size: if (nvals <= max_size) then
! Read data
  in: do j = 1, nvals
      read(1,*) input(j)
  end do in

! Find the smallest and largest values
  ismall = 1
  ilarge = 1
  do j = 2, nvals
    if (input(j) > input(ilarge)) ilarge = j
    if (input(j) < input(ismall)) ismall = j
  end do

>Continued on next page...
```
! Write out
write(*,110) input(ilarge)
110 format('The largest value is:',I4)
write(*,111) input(ismall)
111 format('The smallest value is:',I4)

else size
! (nvals > max_size)
   write(*,120) nvals, max_size
   120 format('Too many input values: ',I6,' > ',I6)
end if size

end program

---

```fort.1
10    nvals
922   input(1)
13    input(2)
-4    ...
33
-12
14
689
67
0   ...
92   input(nvals)
```
Exercise

A data file, which contains the grades of a quiz, is as shown below. In the file, the first column is the name of the student, the second column is the ID number of the student, and the third column is the grade. Write a Fortran program that will:

1. Read the total number of students took the quiz and the name, ID and grade of each student from a data file `grade.dat`, such as the one shown below,
2. Compute the averaged grade of the quiz,
3. Compute the standard deviation of the grades,
4. Compute the averaged grade of the students who pass the quiz (grade >= 60),
5. Count the number of students who fail the test (grade < 60),
6. Find the lowest and highest grades and the students' names and ID's.
7. Sort the records according to descending order of the grade.

Output all of your results after the end of the file `grade.dat`.

An example of data file `grade.dat`:

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wuting</td>
<td>9912345</td>
<td>98</td>
</tr>
<tr>
<td>Scott</td>
<td>9800001</td>
<td>85</td>
</tr>
<tr>
<td>Tsai13</td>
<td>9724680</td>
<td>37</td>
</tr>
<tr>
<td>John</td>
<td>9514248</td>
<td>60</td>
</tr>
<tr>
<td>BJ4</td>
<td>9800002</td>
<td>100</td>
</tr>
<tr>
<td>KoP</td>
<td>9723573</td>
<td>59</td>
</tr>
<tr>
<td>MaYJ</td>
<td>9144444</td>
<td>0</td>
</tr>
</tbody>
</table>
6a. Additional Features of Arrays
Whole Array Operation and Array Subset

Ex:

```fortran
real :: a(10)
real :: b(10)
real :: c(10)
...
do i = 1, 10
   c(i) = a(i) + b(i)
end do
```

```fortran
real :: a(10)
real :: b(10)
real :: c(10)
...
c = a + b
```

Ex:

```fortran
real :: a(1:4)=(/1.,2.,3.,4./)
real :: b(5:8)=(/5.,6.,7.,8./)
real :: c(101:104)
c = a + b
write(*,*) c
```

output: 6.00   8.00   10.00   12.00
I/O of Arrays

• Implied DO loop

```plaintext
do i = 1, 5
    write(*,*) a(i)
end do

write(*,*) a(1), a(2), a(3), a(4), a(5)
write(*,*) (a(i), i = 1, 5)
write(*,*) (i, 2*i, 3*i, i = 1, 3)
write(*,*) ((i, 2*i, 3*i), i = 1, 3)
write(1,100) (i, i = 1, 5)
100 format(5I3) 1 2 3 4 5
100 format(3I3) 123 45
100 format(I3) 1 2 3 4 5
```

compilation error
• Nested implied DO loop

```
write(*,100) ( (i, j, j=1,3), i=1,2 )
100 format(2I3)
```

Output:
```
1 1
1 2
1 3
2 1
2 2
2 3
```

```
write(*,100) ( (i, j, i=1,2), j=1,3 )
100 format(2I3)
```

Output:
```
1 1
1 2
2 1
2 2
2 3
1 3
```

```
write(*,100) ( (i, j, j=1,3), i=1,2 )
100 format(3I3)
```

Output:
```
1 1 1
1 2 1
2 1 3
2 2 2
2 2 3
2 3 3
```

```
write(*,100) ( i, j, j=1,3, i=1,2 )
write(*,100) ( (i, j), j=1,3, i=1,2 )
write(*,100) ( ( (i, j), j=1,3), i=1,2 )
```

Compilation error
Two-dimensional Arrays (Rank 2)

• Declaring arrays

Ex:  
real :: a(3,4)  
real, dimensional(3,4) :: b  
real :: c(0:3,-1:4)  
integer, dimensional(0:100,20) :: d  
integer, dimensional(-3:3,10) :: e

• Storage

real :: a(3,2)

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a(1,1)</td>
<td>a(1,2)</td>
<td>a(2,1)</td>
<td>a(2,2)</td>
</tr>
<tr>
<td>a(3,1)</td>
<td>a(3,2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

However, the memory allocation for array a is:

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a(1,1)</td>
<td>a(2,1)</td>
<td>a(3,1)</td>
<td>a(1,2)</td>
</tr>
<tr>
<td>a(2,2)</td>
<td>a(3,2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

i.e. the sequence of storage a(i,j): allocate i first, and then allocate j.
• Initialization with ASSIGNMENT statements

Ex: integer :: istat(4,3)
do i = 1,4
  do j = 1,3
    istat(i,j) = j
  end do
end do

Ex: integer :: istat(4,3)
istat = (/1,1,1,1,2,2,2,2,3,3,3,3/)
a 1×12 vector is assigned to 4×3 array istat, and results in compilation error

Ex: integer :: istat(4,3)
istat = reshape((/1,1,1,1,2,2,2,2,3,3,3,3/),(/4,3/))
reshape converts the 1×12 vector into a 4×3 array

output = RESHAPE( (/value1,value2,.../), (/m,n/) )
• **Initialization with type declaration**

```fortran
integer, dimension(4,3) :: &
    & istat(4,3) = reshape((/1,1,1,1,2,2,2,2,3,3,3,3/),(/4,3/))
```

• **Initialization with READ statements**

If the data stored in the file *fort.99* is: 1 1 1 1 2 2 2 2 3 3 3 3

```fortran
integer :: istat1(4,3), istat2(4,3), istat3(4,3)
integer :: istat4(4,3), istat5(4,3)

do j=1,3
    do i=1,4
        read(99,*), istat1(i,j)
    end do
end do

read(99,*), ((istat2(i,j), i=1,4), j=1,3)

read(99,*), istat3

do i=1,4
    do j=1,3
        read(99,*), istat4(i,j)
    end do
end do

read(99,*), ((istat5(i,j), j=1,3), i=1,4)
```
• **Whole array operation:**

Most of the intrinsic functions operate component-wise on arrays.

```fortran
real, dimension(100,200) :: A,B,C
...
do j=1,200
do i=1,100
   C(i,j) = A(i,j) + B(i,j)
end do
end do
...

real A(100), C(100)
...
do i=1,n
   C(i) = sin(A(i))
end do
...

integer A(100), B(100)
...
A=0        ! set each element of A to zero
B=(B+1)/2  ! add 1 to each element of B, then take half the result
...
But \( C = A \times B \) multiplies corresponding elements in \( A \) and \( B \). It does NOT do matrix multiplication.

\[
\text{real, dimension}(100,200) :: A,B,C \\
\ldots \\
C = A \times B \\
\ldots
\]

\[
\begin{align*}
A &= \begin{bmatrix} 2 & 3 & 5 \\ 1 & 7 & 4 \end{bmatrix} & B &= \begin{bmatrix} 5 & 4 & 1 \\ 2 & 2 & 3 \end{bmatrix} \\
A + B \times A &= \begin{bmatrix} 2 & 3 & 5 \\ 1 & 7 & 4 \end{bmatrix} + \begin{bmatrix} 10 & 12 & 5 \\ 2 & 14 & 12 \end{bmatrix} = \begin{bmatrix} 12 & 15 & 10 \\ 3 & 21 & 16 \end{bmatrix}
\end{align*}
\]

- `matmul(A,B)` returns an array that is the matrix multiplication of \( A \) and \( B \). \( A \) and \( B \) must be conformable for multiplication.
- `dot_product(A,B)` returns a scalar that is the dot product of \( A \) and \( B \).
- `transpose(A)` returns an array that is the transpose of \( A \).
Allocatable Arrays

- **Static memory allocation**
  
  Explicit-shape array
  
  The size of the array has been set at compilation time, and cannot be changed.

  ```
  real, dimension(4,5) :: a
  ```

- **Dynamic memory allocation**
  
  Dynamically sets the size of the array to just large enough for current use.

  ```
  real, allocatable, dimension(:, :) :: b
  ...
  ...
  allocate(b(1000, 0:1000))
  ...
  ...
  deallocate(b)  
  ```

  An array of $1000 \times 1001$ will be allocated at execution time.

  The array is no longer available for use.
Example:

Read in a rank-2 array from a data file and calculate the sums of all the data in each row and of each column in the array.

In the input data file, the first line contains two integers which specify the dimensions of the array. The elements in each row of the array appear on a line (i.e. a record) of the data file.

Input file:

```
2 4
-24.0 -1121.  812.1  11.1
35.6  8.1E3  135.23 -17.3
```

Output:

```
Sum of row 1=  -321.80
Sum of row 2=  8253.53
Sum of column 1=    11.60
Sum of column 2=   6979.00
Sum of column 3=    947.33
Sum of column 4=   -6.20
```
program array_example
implicit none
real, allocatable, dimension(:, :) :: a ! Data array to sort
character(len=20) :: infile ! Input data file name
character(len=20) :: outfile ! Output data file name
integer :: m, n ! Dimensions of the array
integer :: i, j ! Loop indices
real :: sum_row ! Sum of elements in a row
real :: sum_col ! Sum of elements in a column

! Get the names of the input and output files
write(*,11)
11 format(1X,'Enter the filename to read the data:')
read(*, '(A20)') infile
write(*,12)
12 format(1X,'Enter the file name to write the results:')
read(*, '(A20)') outfile

! Open output data file
open(unit=2, file=outfile, status='replace', action='write')

! Open input data file and read the dimensions of the array
open(unit=1, file=infile, status='old', action='read')
read(1,*) m, n

! Allocate memory
allocate (a(m,n))  

Continued on next page...
! Read in the data
do i = 1,m
    read(1,*) (a(i,j), j = 1,n)
end do

! Loop to sum elements in a row
do i = 1,m
    sum_row = 0.0
    do j = 1,n
        sum_row = sum_row + a(i,j)
    end do
    write(2,120) i, sum_row
    120 format('Sum of row',I4,'=',F12.2)
end do

! Loop to sum elements in a column
do j = 1,n
    sum_col = 0.0
    do i = 1,m
        sum_col = sum_col + a(i,j)
    end do
    write(2,110) j, sum_col
    110 format('Sum of column',I4,'=',F12.2)
end do
end program
Exercise

Matrix Multiplication:

If a matrix $[A]$ is an $(N \times L)$ matrix, and matrix $[B]$ is an $(L \times M)$ matrix, then the product $[C] = [A] \times [B]$ is an $(N \times M)$ matrix whose elements are given by the equation:

\[ c_{i,k} = \sum_{j=1}^{L} a_{i,j} \times b_{j,k} \]

where $j = 1$ to $N$ and $k = 1$ to $M$.

Write a FORTRAN program that reads two matrices of arbitrary size from the input data file and compute the multiplication using the formula above. An example input data file for the matrices:

\[
[A] = \begin{bmatrix}
1.3 & 2.5 \\
3.6 & -2 \\
3.05 & -0.03
\end{bmatrix} \quad [B] = \begin{bmatrix}
2.0 & -0.2 & 3.4 & 38.9 & 23.9 \\
12.4 & 2.7 & -7.1 & 1.2 & 2.4
\end{bmatrix}
\]

is given as follow:

```
3  2
  1.3  2.5
  3.6  -2.0
  3.05  -0.03

2  5
  2.0  -0.2  3.4  38.9  23.9
 12.4  2.7  -7.1  1.2  2.4
```

In the data file, the two integers (e.g. 3 2 and 2 5 in the example data file) specify the number of rows and columns of the matrix.
• Use **open** and **read** statements to input the data file. The filename of the data file is input interactively as in the example.

• Use **allocatable arrays** to hold the input matrices and the resulting output matrix. You do not need to output the resulting matrix, but print out the elements of the matrix $c_{i,j}$ where $i = j$ on screen, so you can check if your computed results are correct. For example, the output of the above two matrices is:

  33.6  
  -6.12  
  10.583  

• Note that you must compute the whole $[C]$ matrix, then output $c_{i,j}$ where $i = j$. Don't write a program which just computes $c_{i,j}$ where $i = j$. 