Memory Problems

Problem

Read 4 test scores (1 per line) and 1) find the average 2) print the average, 3) and finally print the 4 scores; (in this order).

Solution:

```
          :
          :
    READ(9,100) score1
    READ(9,100) score2
    READ(9,100) score3
    READ(9,100) score4
    ave = ( score1 + score2 + score3 + score4) / 4.0
    WRITE (10,200) ave
    WRITE (10,201) score1
    WRITE (10,201) score2
    WRITE (10,201) score3
    WRITE (10,201) score4
          :
```

Now repeat the problem above, but for 400 test scores.

Solution is **NOT** trivial!

Memory/Variables Problems

Any problem that involves accesssing large sizes of data values in primary memory to perform calculations or output will lead to the same type of memory/variable accessing problems that cannot be practically solved using only simple variables.

Arrays

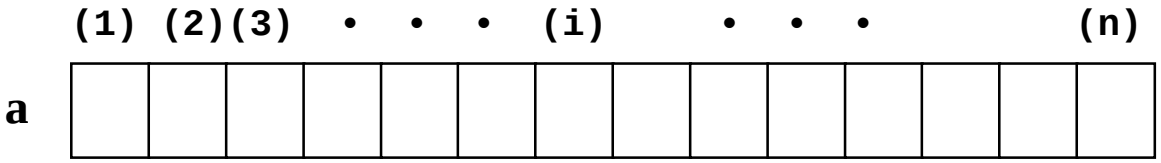
Definitions:

collection of variables that all share the same name and the type; can be thought of as a list or table

Array : sequential list of variables of the same type.

Mathematics: a_1 a_2 a_3 • • a_i • • a_n

FORTRAN: a(1) a(2) a(3) • • a(i) • • a(n)



Elements have same properties as simple variables of the same type.

Individual array elements are treated just like simple variables.

However, the entire array itself is NOT!The array name can appear by itself in certain cases !

INDEX SCORE access

```
     +----+
1    |89| score(1)
     +----+
2    |78| score(2)
     +----+
3    |92| score(3)
     +----+
4    |84| score(4)
     +----+
       :
       :
```

the 1 is a subscript or index

The name of the whole array is SCORE.

We distinguish between individual elements of the array by giving a subscript: 1, 2, 3, etc.

Array Declaration

Declaration Examples

INTEGER, PARAMETER :: maxscores =400, maxcredits=5

INTEGER, PARAMETER:: maxstus=5, numemps=75

INTEGER:: score(maxscores), credits(maxcredits), idnum(maxstus)

REAL:: weight(maxstus), wages(numemps)

INTEGER::arr(-2:4), b_arr(3,4)

INTEGER, DIMENSION(2)::x_arr

INTEGER, DIMENSION(2:4):y_arr

Accessing Array Elements:

A subscript is used to access an array element.

Subscript must be an integer expression or constant

```
      score(1) = 0
score(2) = score(1)
```

array initialization

```
DO i = 1, maxscores
    score(i) = 0
  END DO
```

array summation

```
sum = 0
DO  i = 1, maxscores
sum = sum + score(i)
    END DO
```

Remember: Array subscripts:

1) integer (expression)

2) must be in the proper range: lower bound to upper bound

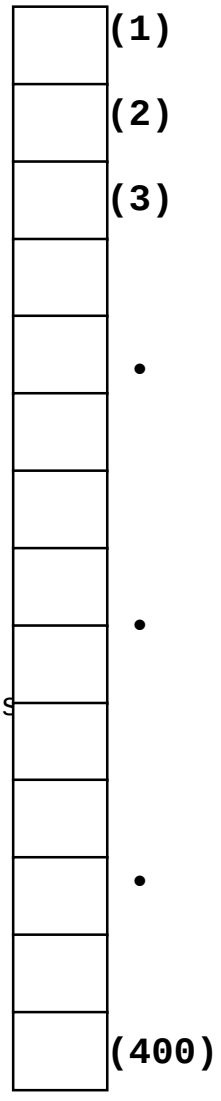
Memory/Variables Solution

Problem

Read 400 test scores (1 per line) and 1) find the average 2) print the average, 3) and finally print the 400 scores, (in original order).

Solution:

```
PROGRAM example
IMPLICIT NONE
INTEGER :: count, total
INTEGER, PARAMETER :: maxscores = 400
INTEGER :: score(maxscores)
REAL :: average
total = 0
! input scores and sum
DO   count = 1, maxscores
     READ(9,..) score(count)
     total = total + score(count)
END DO
! compute average & print
average = REAL(total) / REAL(maxscores
WRITE(10,..) average
! output scores
DO  count = 1, maxscores
    WRITE(10,..) score(count)
END DO
STOP
END PROGRAM example
```



Array Input

Array Input Examples

Given the following **array declarations**:

```
INTEGER ::  score(300), x(5), y(5), i, j
REAL ::  charge(8)
```

Array Input Code	**Resulting Actions**

`DO  i = 1, 300`
`   READ(9,..) score(i)`
`END DO`
	reads 1 value from each of the next 300 lines

`DO     j = 1, 5`
`   READ(9,..) x(j), y(j)`
END DO
	reads 2 values from each of the next 5 lines

Automatic Array Input Examples (Short-List Techniques)

Auto-Array I/O can only be used to input/output the entire array in sequence of the subscripts without other I/O, (of other variables or data) between the array elements.

	`READ(9,250) charge`	
`250`	`FORMAT ( 8F6.2 )`	reads 8 values from 1 line better to specify as (8(F6.2,TR1))
	`READ(9, 260) charge`	
`260`	`FORMAT (F6.2)`	reads 8 values from 8 lines
`260`	`FORMAT (2F6.2)`	reads 8 values from 4 lines
`260`	`FORMAT (4F6.2)`	reads 8 values from 2 lines

Array Output

Previous Output Example

```
      DO count = 1, 400
         WRITE (10,300) score(count)
      END DO
300   FORMAT (' ',I5)
```

Outputs 1 value per line, 400 lines.

Implied DO-Loop

The number of values input/output due to an implied do-loop is dependent upon the number of values per line specified in the format statement.

Abbreviated notation for Array I/O.

Specified on the same read/write line as the array.

Enclosed in parenthesis, with the array reference(s), omitting the DO.

Example:

```
      WRITE (10, 300) (score(count), count = 1, 400)
300   FORMAT (' ', I5)
```

Count serves as the LVC & must an integer variable.

The above implied DO outputs 1 value each on 400 lines.

If the following format statement had been used:

```
300   FORMAT (' ', 4(I5,TR1))
```

The implied DO-loop would output 4 values per lines on 100 lines.

Implied Input Loops

The following code segment shows an implied DO-loop with a read statement:

```
      INTEGER ::  day(15), idnum(15), i
      READ (9, 200) (day(i), idnum(i), i = 1, 15)
200   FORMAT (I1, TR1, I3)
```

The implied DO-loop would input 2 values per line from the next 15 lines.

Standard Deviation

Std. Dev. Formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N} (x_i - \mu_x)^2}{N}}$$

σ	standard deviation
x	represents data values
N	is the number of values
μ	is the mean or average

Std. Dev. Program

```
PROGRAM standev
INTEGER :: i
INTEGER, PARAMETER :: size=100
REAL :: vals(size), sum, avg, difsum, stdev

READ (9, ?) (vals(i), i= 1, size)

sum = 0
DO   i = 1, size
     sum = sum + vals(i)
END DO
avg = sum / size

difsum = 0
Do   i = 1, size
     difsum = difsum + (vals(i) - avg)**2
END DO

stdev = SQRT( difsum/size)
```

input values

find avg

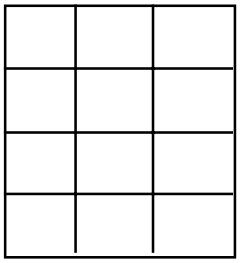
determine sum of the squared differences

compute std dev

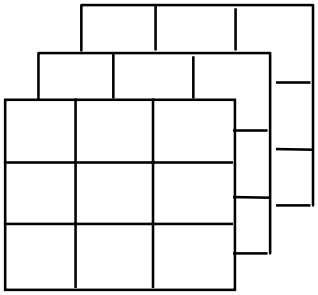
Multi-Dimensional Arrays

An array can be declared with multiple dimensions.

2 Dimensional

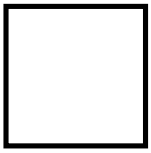


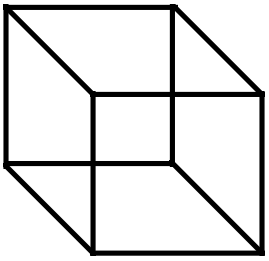
3 Dimensional

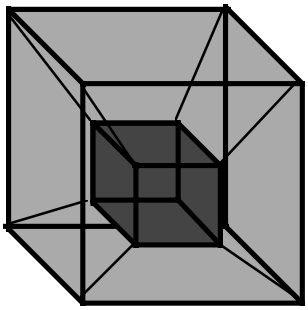


Multiple dimensions get difficult to visualize graphically.

-







Two-Dimensional Arrays

Viewed as a table (rows, columns) of data

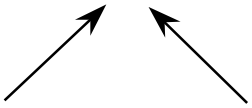
Given the following **Population** Data Table:

	columns	**1**	**2**	**3**	**4**
		1990	1991	1992	1993
Blacksburg	1	23.0	24.0	24.5	28.0
C'Burg	2	8.0	9.0	7.0	8.5
Radford	3	4.0	4.5	5.2	5.6
Pulaski	4	18.1	19.2	20.1	20.2
Dublin	5	8.6	7.6	6.0	2.1

rows thousands

Consider this info to be stored in an array POP declared as:

real **POP(5,4)**



rows cols

What is the population of ..

Radford in 1991 ?

4.5 thousand= **POP(3,2)**

Christiansburg in 1992 ?

7.0 thousand= **POP(2,3)**

2-D Array

Nested DO-Loops

2-D arrays are processed using a DO-loop with an inner DO-loop

Given the declaration:

REAL :: pop(5,4)

Examples:

Print out the array, one number per line row by row:

```
DO    r = 1, 5
      DO     c = 1 ,4
             write(10,..) pop(r,c)
      END DO
END DO
```

Read one number per line, column by column

```
DO     c = 1, 4
       DO   r = 1, 5
            read(9,..) pop(r,c)
       END DO
END DO
```

READ will be executed as ..

pop(1,1)
pop(2,1)
pop(3,1)
pop(4,1)
pop(5,1)
pop(1,2)
pop(2,2)
etc.

2-D Array (cont)

Examples (cont)

What if each line of the input data has a row of the data ?

How can we read it into the array properly ?

```
DO  r = 1, 5
      read(9,..) (pop(r,c), c = 1, 4)
END DO
```

What if all the data is on one line ?

READ(9,..) pop Short-List method (Do NOT do it this way).

READ(9,..) ((pop(r,c), c = 1, 4), r = 1, 5)

read all columns for each row

read all rows

Population Table Problem

Problem: Write a program to read in the city population data file and compute the population averages for each year, for every city and for all of the cities for all of the years.

For each year determine the average population for all of the area cities.
For each city determine the average population during the given years.
Determine the average city population for all of the given years.

columns		**1**	**2**	**3**	**4**	**5**
		1990	1991	1992	1993	**avgs**
Blacksburg	1	23.0	24.0	24.5	28.0	
C'Burg	2	8.0	9.0	7.0	8.5	
Radford	3	4.0	4.5	5.2	5.6	
Pulaski	4	18.1	19.2	20.1	20.2	
Dublin	5	8.6	7.6	6.0	2.1	
averages	6					

rows

thousands

overall average

Declare an "**extra**" row & column in the table to store the averages.

Population Table Program

```
      PROGRAM POPULOUS
      IMPLICIT NONE
      INTEGER :: row, col
      INTEGER, PARAMETER :: cities = 6, years = 5
      REAL :: popdata(cities,years), yearsum(years)
      REAL :: yearcitysum,citysum

      OPEN ( 9, FILE = 'CITYPOP.DAT')
      OPEN (10, FILE = 'POPAVG.RST' )

      DO row=1, cities-1
         READ (9, 100) (popdata(row,col) , col=1, years-1)
      END DO
      yearcitysum = 0.0
      DO yr = 1, years-1
         yearsum(yr) = 0.0
      END DO
      DO row=1, cities-1
         citysum = 0.0
         DO  col= 1, years-1
            citysum = citysum + popdata(row,col)
            yearsum(col) = yearsum(col) + popdata(row,col)
         END DO
       yearcitysum = yearcitysum + citysum
        popdata(row, years) = citysum / REAL(years-1)
      END DO
      DO  yr = 1, years-1
         popdata(cities, yr)= yearsum(yr) / REAL(cities-1)
      END DO
      popdata(cities, years)= yearcitysum / REAL((cities-  &
                                 1)*(years-1))
      DO row=1, cities
         WRITE (10, 700) (popdata(row,col) , col=1, years)
      END DO
      CLOSE(9)
      CLOSE(10)
      STOP
100   FORMAT ( 4(F5.1, TR1) )
700   FORMAT ( 5(F5.1, TR1) )

      END PROGRAM POPULOUS
```

Population Table Program

Typing in the input given in the table on slide 9.12:

```
 23.0  24.0  24.5  28.0
  8.0   9.0   7.0   8.5
  4.0   4.5   5.2   5.6
 18.1  19.2  20.1  20.2
  8.6   7.6   6.0   2.1
```

The given program produces the (woefully unlabelled) output:

```
 23.0  24.0  24.5  28.0  24.9
  8.0   9.0   7.0   8.5   8.1
  4.0   4.5   5.2   5.6   4.8
 18.1  19.2  20.1  20.2  19.4
  8.6   7.6   6.0   2.1   6.1
 12.3  12.9  12.6  12.9  12.7
```

Initialization & Operations (Fortran 90 Style)

EXAMPLES:

```
INTEGER, DIMENSION(10):: arr = (/1,0,9,5,6,4,2,3,8,2/)
INTEGER:: i
INTEGER, DIMENSION(5):: b_arr = (/ (0, i = 1, 5) /)
INTEGER, DIMENSION(10) :: c_arr = (/0,0, (i, i = 3, 7), 0, 5,3/ )

INTEGER:: x_arr(20), y_arr(30)
x_arr=(/ (0, i = 1, 10)/ )
y_arr=(/(2*i, i = 1, 30, 2)/)
y_arr=(/(i, i = 2, 30, 2)/)
```

AGGREGATE OPERATIONS ON ARRAYS

Fortran 90 language supports aggregate (element-by-element) operations on arrays provided that the arrays and scalars involving in an operation are conformable to each other and the operation is defined between two conformable objects.

EXAMPLES:

```
! The following declaration will declare and set all elements of arrays x,
! y,  and z to zeros.
REAL, DIMENSION(20)::x=0.0, y=0.0, z=0.0
REAL :: array_max
                :
! The following statement will multiply 10 to every element of x.
       x = 10 * x
! The following statement will multiply corresponding elements of x and
! y and set the corresponding elements of z
       z = x * y
! The following statement will assign the maximum of x, y, and z to
! array_max by using the intrinsic function MAX.
                array_max = MAX (x, y, z)
```