Fortran - Arrays

Arrays, Multidimensional Array, Dynamic Arrays

Spring 2017

Victor Eijkhout and Charlie Dey

Arrays

a definition

a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.

Arrays

An example

```
program array1
implicit none
integer :: i
real, dimension(5) :: A = (/ 1, 2, 3, 4, 5 /)

do i=1,5
   print *, A(i)
end do

end program array1
```

What's different from C/C++?

- index starts at 1
- `()`'s instead of `[]`'s
- explicit declarations require '`/`' at the beginning and end of the series

Arrays

Reading and writing

```
program array2
implicit none
integer :: i, n=5
real, dimension(n) :: A

do i=1,5
   A(i)= i*i
end do

do i=1,5
   print *, A(i)
end do

print *, A

end program array2
```

What's different from C/C++?

- index starts at 1
- `()`'s instead of `[]`'s
- explicit declarations require '`/`' at the beginning and end of the series
- you can reference an array by the array variable.
- size of an array can be a parameter

Exercise 1.

Write a program that creates an array of 100 random numbers between 0 and 100
Run the following code, modify it so it meets the exercise criteria

```
// this code generates an array of 100 random numbers
program test_random_number
implicit none
real :: r(100)
    call random_number(r)
    print *, r
end program
```

Arrays

As an argument to a Function

```
real function average(n, x)
integer              :: n, i
real, dimension(n)   :: x
real                 :: sum

sum = 0.

do i=1, n
  sum = sum + x(i)
enddo

average = sum / real(n)

end function average
```

```
program with_fct
! Declaration of variables
! Read data (n,a) from a file

! Calculate Average
aver = average(n, a)    ! Function
                        !   call
! Read more data (n2, a2)
open …; read …; close …

! Calculate Average again
aver2 = average(n2, a2)

end program
```

Exercise 2.

Using your random array generators,
Write 2 functions that take an array as an argument

- one function that finds the maximum value and the index of the maximum value
- one function that finds the minimum value and the index of minimum value

Exercise 3.

Using your random array generator,
Write a function that takes an array and 2 index locations and swaps the values of the array at the 2 index locations.

Exercise 4.

Using your random array generator,
Write a function that will sort your randomly generated array from smallest to largest, by traversing your array and swapping values of adjacent indices if a(i) > a(i+1)

How can you test that your array is sorted?

Exercise 5.

Using Exercise 4, write a test function which will take your "sorted" array as an argument and tests it to verify that the array is indeed sorted, this function will return a logical.

Multi-dimensional arrays

the definition from C/C++ (Row major)

```
int a [ 3 ] [ 4 ];
```

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Arrays

Run this code.

```
program array3
implicit none
integer :: i, j, k=0
integer, dimension(5,5) :: A

do i=1, 5
   do j=1, 5
      k = k + 1
      A(i,j)= k
   end do
end do

print *, A

end program array3
```

It will print all of Row 1, then Row 2, then Row 3, all unformatted.

Arrays

Run this code.

```
program array3
implicit none
integer :: i, j, k=0
integer, dimension(5,5) :: A

do i=1, 5
   do j=1, 5
      k = k + 1
      A(i,j)= k
   end do
end do

print *, A

end program array3
```

It will print all of Row 1, then Row 2, then Row 3, all unformatted.

you should see something like:

```
 1           6          11          16          21           2
 7          12          17          22           3           8
13          18          23           4           9          14
19          24           5          10          15          20
25
```

Arrays

Run this code.

```
program array3
implicit none
integer :: i, j, k=0
integer, dimension(5,5) :: A

do i=1, 5
   do j=1, 5
      k = k + 1
      A(i,j)= k
   end do
end do

print *, A

end program array3
```

It will print all of Row 1, then Row 2, then Row 3, all unformatted.

you should see something like:

```
 1           6          11          16          21           2
 7          12          17          22           3           8
13          18          23           4           9          14
19          24           5          10          15          20
25
```

What is this telling us?

Arrays

Run this code.

```
program array3
implicit none
integer :: i, j, k=0
integer, dimension(5,5) :: A

do i=1, 5
   do j=1, 5
      k = k + 1
      A(i,j)= k
   end do
end do

print *, A

end program array3
```

It will print all of Row 1, then Row 2, then Row 3, all unformatted.

you should see something like:

```
 1           6          11          16          21           2
 7          12          17          22           3           8
13          18          23           4           9          14
19          24           5          10          15          20
25
```

What is this telling us?
Fortran is Column major!

```
 1           6          11          16          21
 2           7          12          17          22
 3           8          13          18          23
 4           9          14          19          24
 5          10          15          20          25
```

Arrays

More about array, multidimension arrays.

```
integer, parameter      :: n = 3
integer, parameter      :: m = 7

real, dimension(n,m)    :: a
real, dimension(0:2,4,8) :: b
```

- Ordered collection of elements
- Each element has an index
- Index may start at any integer number, not only 1
- Array element may be of intrinsic or derived type
- Array size refers to the number of elements
- The number of dimensions is the rank
- The size along a dimension is called an extent
- Array shape is the sequence of extents

```
size of a: 21
rank of a: 2
extent of a, first dimension: 3
shape of a: (3,7)

size of b:96
rank of b:3
extent of b, first dimension: 3
shape of b: (3,4,8)
```

Multi-dimensional arrays

recall from C++

```
#include <iostream>


using namespace std;

int multiplyByC(int arr[][4], int rows, int cols, int
C)
{

   for (int i = 0; i < rows; i++)
   {
      for (int j = 0; j < cols; j++)
      {
         arr[i][j] *= C;
      }
   }
   return 0;
}
```

```
int main ()
{
   int a[3][4];

   for ( int i = 0; i < 3; i++ )
      for ( int j = 0; j < 4; j++ )
         a[i][j] = i+j;

   multiplyByC(a, 3, 4, 5);

   for ( int i = 0; i < 3; i++ )
      for ( int j = 0; j < 4; j++ ) {
         cout << a[i][j]<< endl;
      }

   return 0;
}
```

Multi-dimensional arrays

in Fortran.

```
subroutine multiplyByC(arr, ans, rows, cols, C)
implicit none
integer :: rows, cols, C
integer, dimension(rows, cols) :: arr, ans

   ans = arr * C

end subroutine
```

```
program mArray
implicit none
integer :: i, j
integer, dimension(3,4) :: a, b

do i=1, 3
   do j=1, 4
      A(i,j) = i+j
   end do
end do

call multiplyByC(a, b, 3, 4, 5);

print *, b


end program
```

Exercise 6.

Write a subroutine that creates a 100x100 identity matrix , a matrix where the diagonal values are 1's and the rest of the values - the upper and lower triangles - are 0's

Exercise 7.

- Using your random number generator, create 2 random 100x100 matrices.
- Write a subroutine that multiplies the 2 matrices together and puts the result in a third matrix.
- Test your matrix multiplication subroutine by multiplying your random matrix with the same size identity matrix, the result will be the same as the original matrix.

Matrix Multiplication Algorithm:

- Input: matrices A and B
- Let C be a new matrix of the appropriate size
- For i from 1 to n:
  - For j from 1 to p:
    - Let sum = 0
    - For k from 1 to m:
      - Set sum ← sum + A[i][k] × B[k][j]
    - Set C[i][j] ← sum
- Return C

Simple Array Syntax

```
real                    :: x
real, dimension(10)     :: a, b
real, dimension(10,10)  :: c, d

a       = b
c       = d
a(1:10) = b(1:10)
a(2:3)  = b(4:5)
a(1:10) = c(1:10,2)
a       = x
c       = x
a(1:3)  = b(1:5:2) ! a(1) = b(1)
                   ! a(2) = b(3)
                   ! a(3) = b(5)
```

```
a = c(:,1)    ! 1st column
a = c(:,5)    ! 5th column
a = c(1,:)    ! 1st row
a = c(5,:)    ! 5th row
```

- Variables on the left and the right have to be conformable in size and shape
  - i.e. number of elements and rank
- Scalars are conformable
- Strides can be used