


Computational Physics
Dr. Apratim Chatterji
Dr. Prasenjit Ghosh
Department of Physics
Indian Institute of Science Education and Research, Pune

Lecture - 04
Introduction To Fortran Part-2
Session-B

(Refer Slide Time: 00:16)

ARRAYS

- Array is an ordered sequence of many instances of the same data type
- Equivalent of a vector or a matrix
- Declared using the **DIMENSION** keyword:
`<DTYPE>, DIMENSION(<id>) :: <var>`
- <id> is a comma separated list describing the index range in each dimension:
 - A single value n means that n values in that dimension with indices {1,2,...,n}
 - Values n:m means (m-n+1) values in that dimension, with indices {n, n+1,...,m}



$\begin{matrix} u_1 \\ u_2 \\ u_3 \end{matrix}$ } each component of a vector
 $u(3)$ } array location

$m(3,3)$ } 1:3
 $m(3,3)$ } 1:3

Now, in this last part of this lecture what we are going to do is, we are going to talk about arrays. So, again the idea of an array is if you have a set of numbers which are similar type or which contains similar type of information, then instead of defining a variable for each of the numbers I can set them in a form of a array by using just a 1 variable type. So, by definition an array is an ordered sequence of many instances of the same data type. Say for example, you can think of that you have a you have need to write a vector . So, a vector has three components x y and z.

So, you can do it in two ways. So, you can declare say for example, v 1 as 1 variable, v 2 as 1 variable, v 3 as another variable. So, each contents each component of a vector. So, basically what you are doing, you are using a 3 variables here, but the same thing you can store using just 1 variable say I call it as v and I declare it as an array containing 3 numbers. So, this is what the idea of using an array is. So, it contains it stores the same

information, it stores all the individual components of a vector, but now I use just 1 variable instead of 3 variables here .

So, similarly same thing also applies when we try to store matrix say for example, are 3 cross 3 matrix. So, the way one uses this is; so basically array food has one needs to mentioned two things; one is what type of data you are storing in that array that is determine given by this data type, then you need to also specify the dimension of the array which is done by using this dimension word keyword in the Fortran and then you give the variable name. So, one thing you note is that t this contains the dimensions So, basically what it is a its a comma separated list describing the index range in each dimension.

so what I mean to say is suppose I want to store a 3 cross 3 matrix. So, I define the matrix by m. So, now, I have I have to give 3, so I the way I will declare that array is m 3; 3 . So, what it means is that along one direction I have 3 elements and along the other direction I have 3 elements now this if I write it this way. So, what the code will assume is that the counter on 3 goes to 1 2 3 and similarly for this it will also go to 1 2 3.

So, instead what I can also do is if I want to go the this index from 0 to 2 I can write it as m equals to 0 colon 2 comma 0 colon 2 in this fashion. So, what it means is that basically. So, this one has three numbers that is m is my upper one n is my lower one, so m minus n 2 minus 0 is 1 is 2 plus 1. So, that tells me three numbers and similarly in this way. So, the number of values stored in this part is 3 in this example and 3 in this example.


(Refer Slide Time: 04:32)

Examples

```
! Declare v1 to be a vector of 5 reals
! with indices 1, 2, 3, 4, 5
REAL, DIMENSION(5):: v1

! Declare v2 to be a vector of 4 reals
! with indices -1, 0, 1, 2
REAL, DIMENSION(-1:2):: v2

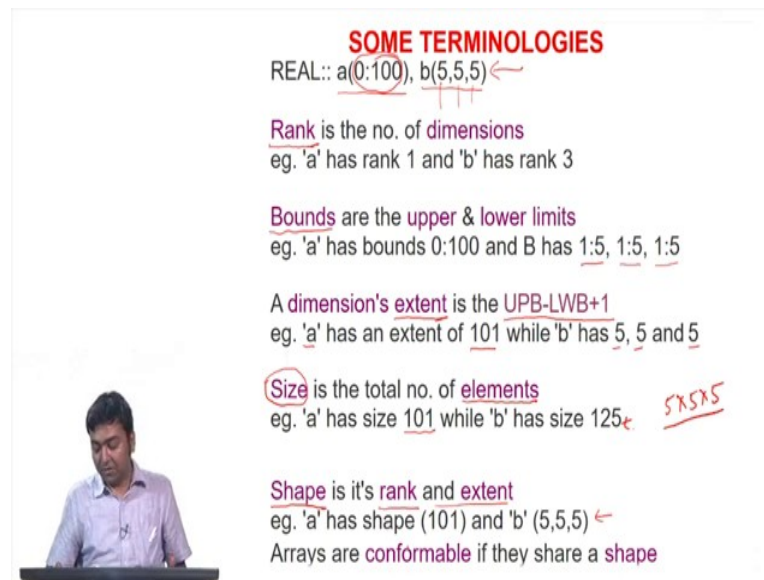
! Declare m1 to be a 2x2 array of integers
! with indices 1, 2; 1, 2
INTEGER, DIMENSION(2,2):: m1
```



So, here are some examples in which one can declare variables. So, for example, here I have declared an array which I am calling v 1 which are the dimension 5 and it is a real type of array. Similarly, now in this case the indices will be going from 1 to 5, but I do not want to do that, I want to declare an array with a indices minus 1, 0, 1 and 2. So, then I change this part of my declaration in this fashion.

So, I give a lower limit which starts from minus 1 and a upper limit which starts from 2 and going by the previous formula, so 2 minus minus 1 plus 1 this will give me 4. Similarly, I can declare I want to declare 2 cross 2 array of integer numbers so, where the indices grow from 1 to 2 in a along each direction, so, I do it do it in this fashion.

(Refer Slide Time: 05:28)



SOME TERMINOLOGIES

REAL:: a(0:100), b(5,5,5) ←

Rank is the no. of dimensions
eg. 'a' has rank 1 and 'b' has rank 3

Bounds are the upper & lower limits
eg. 'a' has bounds 0:100 and B has 1:5, 1:5, 1:5

A dimension's **extent** is the $UPB-LWB+1$
eg. 'a' has an extent of 101 while 'b' has 5, 5 and 5

Size is the total no. of elements
eg. 'a' has size 101 while 'b' has size 125 ← $5 \times 5 \times 5$

Shape is its rank and extent
eg. 'a' has shape (101) and 'b' (5,5,5) ←

Arrays are **conformable** if they share a shape

So, there are some terminologies which is associated with arrays. So, let us look at these and let us start with this particular example. So, here I have in this line I have two error declarations; one is given by a which has just one set of indices which goes from 0 to 100 and another given by b which has three sets of indices which each of which goes from 1 to 5.

Now, one important thing about the array is called the rank of the array which tells us how many dimensions are present in the array? For example, for case a the rank will be just 1 because it has just 1 dimension, in contrast for b the rank will be 3 because it has 3 dimensions. Then there is something called the bounce of an array which tells us the lower and the upper limits of the array along each dimension. So, for example, for a the bound will be 0 to 100 while for b along each dimension the bound is 1 to 5.

Now, the dimensions extent tells us how many elements are present along that particular dimension and this is given by the formula which is the upper bound minus the lower bound plus 1. So, for this particular array, so, the number of dimensions will be 101, 100 plus 0 plus 1 and while for the array b each dimension has an extent of 5. Then another character terminology to characterize an array is its size.

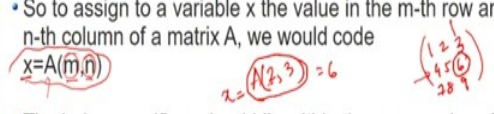
So, size is basically contains the total tells us how the main elements are there in the array. So, for example, here array a has 101 elements, array b along each dimension has 5 elements. So, you can think of 3 dimensional array as a cube in which the numbers are


arranged in a cubic fashion. So, a cube; so, 5 into, so each dimensions has 5, so, 5 into 5 into 5. So, total 125 elements are present in the array.

Then there is something called the shape and a shape of an array is defined by its rank and extent. So, for example, a has a shape 101 and b has a shape 5, 5, 5 and like matrices. So, arrays also needs to be conformable; so, arrays are conformable when they have the similar shape.

(Refer Slide Time: 08:12)

Accessing array elements

- We access a particular element of an array via its subscript values
- So to assign to a variable x the value in the m-th row and n-th column of a matrix A, we would code $x=A(m,n)$

- The index specifiers should lie within the ranges given in the declaration
- Sub-array can be accessed using a range-specifier (as in a declaration).



Now, how does one access an element in an array? So, that is done by its subscript values. So, we access a particular element of an array using the subscript values. So, for example, to assign a variable x the value in the mth row and the nth column of a matrix A we could use a statement like this.

So, basically what this statement means is that in the matrix element A in the mth row and the nth column. So, suppose if I have a 3 cross 3 matrix and I write a I consider A as 2 and 3. So, I have a 3 cross 3 matrix containing say 1 2 3 4 5 6 7 8 9; 9 elements and now I want to pick up this particular element and assign it to this variable x. So, what it means is that the program will go to the mth row that is m here is 2. So, I go to the second row and from here from and the column I go to the third column. So, basically this is the element that will go to x.

So, these numbers m and n these are called the index specifier of an array and one should be careful that the arrays the index specifier should learn should lie within the ranges given in the declaration.

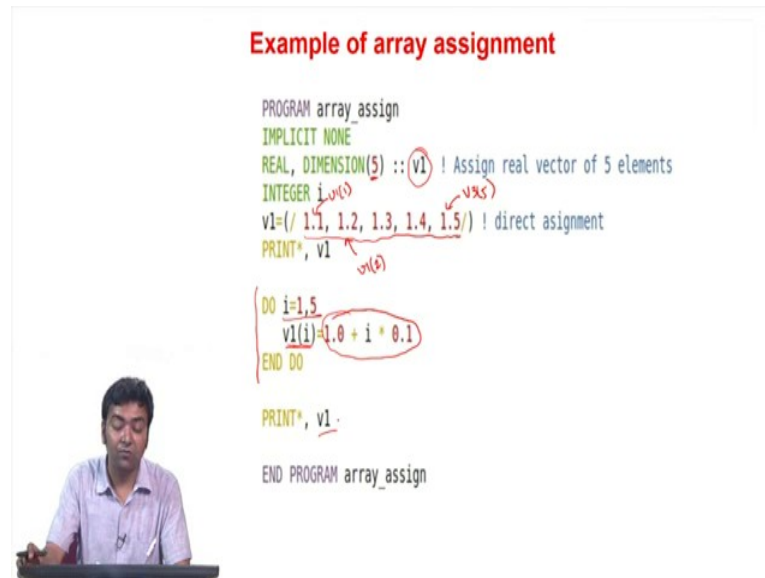
(Refer Slide Time: 09:49)

Example of array assignment

```
PROGRAM array_assign
IMPLICIT NONE
REAL, DIMENSION(5) :: v1 ! Assign real vector of 5 elements
INTEGER i ! Assign integer index
v1=(1.1, 1.2, 1.3, 1.4, 1.5) ! direct assignment
PRINT*, v1

DO i=1,5
  v1(i)=1.0 + i * 0.1
END DO

PRINT*, v1
END PROGRAM array_assign
```



Finally one can also use sub-array by specifying a certain range along a particular dimension as given in the declaration. So, this is an example of a code which assigns an array. For example, I have here declared a variable which is an array which five elements in a particular dimension and like one way I can do the assignment is in this fashion. So, I by hand assign each of the elements of the array. So, here say for example, v this will be v 1 1 this will be v 2 sorry v 1 2 and so on and so forth and this will be v 3 5 element of the array.

But if you have I mean this is for this type of assignment is feasible only when you have a few numbers, but if you have say an array which in which use to a million numbers, then one cannot do the assignment in this way and then one needs to find an automated way to generate the values or assign the values in to the individual elements of an array and this can be done in the following way. So, in this case what we have done is we have used a do statement. So, I have five elements in the array, so my the counter in my do loop goes from i equals to 1 to 5 and then for each index of the array I assign the value given by this mathematical operation.

And then once this assignment is done here I am printing the output on the screen. So, what you can do is you can try out the this program and try to see how the output is done and I try to understand how the assignment is done.

(Refer Slide Time: 11:17)

Example of array assignment

```
PROGRAM arrar_assgn_2
IMPLICIT NONE
INTEGER, DIMENSION (2,2):: matrix
INTEGER::i,j
->DO i=1,2
->DO j=1,2
matrix = i+i+j-2
END DO
END DO
PRINT*, matrix
END PROGRAM arrar_assgn_2
```

matrix(j,i) = i+i+j-2

So, so far in the last example was assigning an array which contains a vector now in this example we will assign an array which contains a matrix. So, now, for the matrix we have 2 dimensions present here. So, basically to do an automated assignment we need to have to do loops. So, that is why I have uses a nested do loop here. So, do i equals to 1 to 2 do j equals to 1 to 2 and then the matrix element will be assigned in this fashion.

So, this what it will do is it will each time this loop is executed, it will assign all the values; it will assign this value to all the matrix elements if you want to assign different values then you can write this in this fashion matrix j comma i equals to i plus 1 plus i plus i plus j minus 2. So, this the way I mean writing it in this way it will assign a different values to each of the elements otherwise, if we had written it in this fashion then every time this statement is executed what would happen is all the elements of the matrix would have been reassigned with the new value of this expression.

(Refer Slide Time: 12:37)

Array operation


Most built in operators/ functions acts element by element on arrays

```
REAL, DIMENSION(50):: v1, v2, v3
v1=SIN(v2)+EXP(v3)
REAL, DIMENSION(50):: v1, v2, v3
LOGICAL, DIMENSION(50):: flag
flag=(arr1 > arr2 .OR. arr3 < 0.0)
```

Handwritten notes:

$v(40) = \sin(v2(40)) + \exp(v3(40))$

$\{ \text{DO } i = 1, 50$
 $v(i) = \sin(v2(i)) + \exp(v3(i))$
 END DO



So, one can do also operations which one can instead of using a do loop one can if one tries to do the same operation and all the elements of the array one can also do that. For example, suppose here I have three arrays v 1, v 2 and v 3. So, what I want to do is, so, my v 2 and v 3 I know what are the array elements and I want to create a new array v 1 in which each element of this new array contains the sign of the corresponding element in the array v 2 plus the exponential of the corresponding array in v 3.

So, basically what I mean is suppose if I am looking at v say 40 the 40th element in the array v this would be sin of the 40th element in array v 2 plus the exponent of the 40th element in the array v 3. So, one way to do that is I have I executed a do loop do i equals to say 1 to 50 end do then my v i will be sin v 1 i plus exponential v 2 i. So, this is one way to do it, but you can do it in a much more sophisticated way.

So, all these three sets of instructions you can directly replace by this one single line. So, basically what it means is you can I mean just by using this one expression you can do these same operations on all the elements of the array at one shot. So, similarly thing also applies for a logical array if you have one.

(Refer Slide Time: 14:43)

Array intrinsic function

SUM(x[,n]) ! Sum of all elements of x


PRODUCT(x[,n]) ! Product of all elements of x

TRANSPOSE(x) ! $X_{ij} \Rightarrow X_{ji}$

DOT_PRODUCT(x,y) ! $\sum_i x_i \cdot y_i \Rightarrow z$ ✓

MATMUL(x,y) ! $\sum_k x_{ik} \cdot y_{kj} \Rightarrow z_{ij}$

Second dimension of x must match with first dimension of y



Apart from these there are some array intrinsic functions. Suppose if you are interested to know the sum of all the elements present in an array x. So, what one can do is, one can use this Sum variable. So, what this Sum variable will do is, it will take each of the individual elements of the array and then gradually and then add them up and in the output it will return the sum of them numbers.

Similarly, you can use product to compute the product of the number of the all the individual elements of that array, you can compute a use the function called transpose which will compute basically which will redefine your array elements in this form. So, x_i and x_i j_{th} element will be assigned to the x_j i_{th} one, so same concept like a transpose of a matrix.

If you have two vectors we just stored in an array you can use something some function called the dot underscore product which will give you the number which will return a number which contains the vector product you can also do multiplications with between 2 matrices or 2 arrays in using this matmul intrinsic function, but one should be careful that the second dimension of x. So, these two should be matching the second dimension and the first of x and the first dimension of y.

(Refer Slide Time: 16:12)


SIMPLE I/O OF ARRAYS

Arrays can be included in input & output. They are expanded in array element order

```
REAL, DIMENSIONS(3,2):: array1  
READ*, array1
```

It is same as

```
REAL, DIMENSIONS(3,2):: array1  
READ*, array1(1,1),array1(2,1),array1(3,1), &  
array1(1,2),array1(2,2),array1(3,2)
```




So, this is sort of say again similar to my vector algebra. So, here we have some simple examples of input output arrays. So, one can you can include input output the arrays in your input output statements also. So, for example, here you have and this example you have an array 1 whose dimensions are given by this these two numbers and then you read the element in this fashion. So, the first element is assigned with the indices 1, 1 the second element is 2, 1; 3, 1 so on and so forth.

(Refer Slide Time: 16:41)

ALLOCATABLE ARRAYS

Sometimes the size of the array may not be known Beforehand. In those cases use the key words ALLOCATABLE, ALLOCATE and DEALLOCATE

```
PROGRAM allocate_eg  
IMPLICIT NONE  
REAL, DIMENSION(:), ALLOCATABLE:: vec  
INTEGER:: n, i  
  
PRINT*, "Enter the size of the array"  
READ*, n  
  
ALLOCATE( vec(n) )  
DO i=1,n  
vec(i)=ACOS(-1.0)**i  
END DO  
  
PRINT*, vec  
DEALLOCATE(vec)  
END PROGRAM allocate_eg
```



Now, often it is. So, what so far we have seen is that whenever we have an array. So, we need to know a priori that what are the dimensions of the array? But many cases it may not be possible okay. So, what does one do in those cases? So, in those cases what one uses a different type of arrays which are called allocatable arrays. So, what the allocatable arrays does or allows you to do is that it allows you to dynamically allocate or reserve memory to store a set of numbers in an array and then also be allocated.

So, basically it consists of three keywords; first one is allocatable. So, this part comes in the declaration part, then this part this allocate comes in the program where you basically in the main part of the body where you allocate the dimensions to the to the particular array which you have declared as allocatable.

And then once you are done using the data of that which is stored in that array and you do not want to use it or use the data again, then what you can do is you can de-allocate the array. So, let us see how one does that. So, for example, here I have a program where I am trying to allocate some values to a vector, but suppose I do not know what is the size of a vector. So, what I will do is if I had known the size of the vector I would have put a number to this dimension indicating the range.

But since I do not know it I put colon here in this line and I declare I use this allocatable statement to tell the compiler that at present I do not know what is the size of this vector. So, the thing which I want to do is I want my the user to determine the size of this vector. So, in order to do so, I need I put in a print statement here which I say "enter the size of the array". So, when the user sees this while running the program, then he or she will write some number which will the value of which will determine what is the dimension of my array. So, this is stored in the value n.

So, then I use the allocate attribute in my program and I assign the vector to have the dimension of n. So, once this is done, so, then now what I can do is I can use a for example, as I have done here I use a do loop and assign to each element of an of this array the following variable. So, once I am done I want to give to the user the values which are which has been generated and stored in this way in this variable 'vec'. So, I ask the code to print this out.

Now once I have I am done with this vector with the value stored in this vector I do not want to store them back. So, what I ask the code is again to deallocate the memory. So,

in the deallocation process one should remember that one loses the information which is already stored in there.

(Refer Slide Time: 20:02)

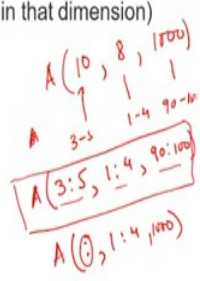

ACCESSING ARRAY SUBSETS

It is possible to access any sub-array

Syntax: must supply a comma-separated list of index descriptors (one for each dimension)

Each index descriptor may be (in addition to a single value or colon separated pair):

- A colon (returns all the elements in that dimension)
- A colon preceded by a value:
→ $array(i)$ is equivalent to $array(i:UBOUND(array, 1))$
- A colon followed by a value:
 $array(j)$ is equivalent to $array(LBOUND(array, 1):j)$



Now, often it might happen is that one can one is interested in sort of accessing the dimensions a certain subset of the array instead of the whole array at one shot. So, that is also possible in Fortran. So, the syntax typically consists of a comma separated list of index descriptors for each dimension. So, for example, suppose if I have an array which has say 10, 8, 1000 .

Now, instead of accessing this whole thing what I want is to array access along this dimension I access one to access the elements which are labeled from 3 to 5, here from 1 to 4 and here from 90 to 100. So, how will I do that? So, I will do it in the following way. So, for each dimension I specify the array the range, so 3 to 5 then for the second one 1 to 4 and then the third one 90 to 100.

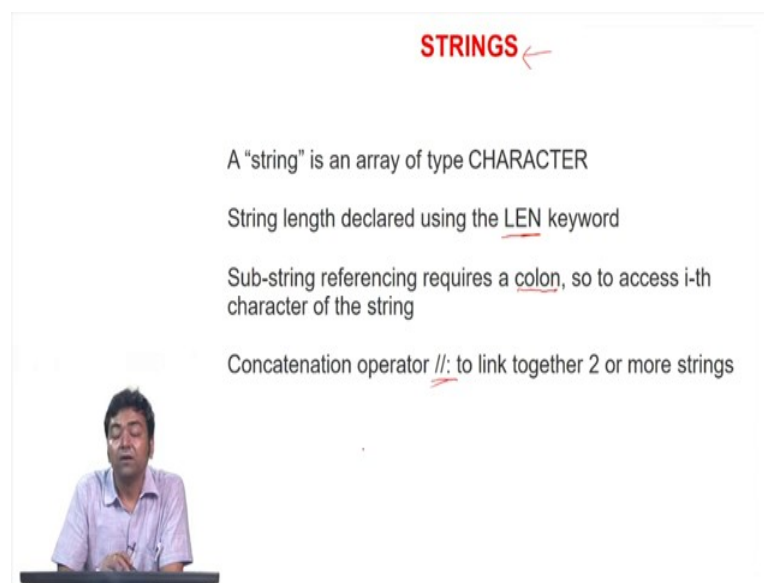
So, when I call this array in this fashion only the elements which are stored in this range will be accessed. So, this is what is written down here. So, each index descriptor may be say in addition to a single value. So, one can use a colon which returns, so instead of doing this you can. So, instead of this for example, what we can I can also ask think of his I want all the 10 elements of this array.

So, what I can do is, when I call this I use a colon; I can use a colon and then I say I want only from 1 to 4. So, this will go from 1 to 4 and then say this I say I go for 1000. So, basically what this colon tells this for this particular dimension I want all the elements present then. So, one can also say from a certain somewhere in the middle of the array to the upper bound when can also want to access those elements or from the lower bound of the array to a to a certain particular value i. So, this is done by in this fashion.

So, basically what you do is you say and in this array. So, suppose I have a 1 dimensional array variable which I called as array 'i' and so, I say that 'i' within the bracket instead of writing the total number of elements in that in that along the dimension I just write i colon. So, what it and then I do not write anything after the colon.

So, what it will do is, so whatever this value i is set to. So, from this value to the upper bound of the array all the elements will be accessed. Similarly, if I write it the other way around here, so I leave the space before the colon empty I do not write anything here and after the colon I write 'i'. So, what it will do is it will access the elements from the lowest bound to the ith element depending on the value of i. So, this is how one can array access subsets of array.

(Refer Slide Time: 23:29)



STRINGS ←

- A "string" is an array of type CHARACTER
- String length declared using the LEN keyword
- Sub-string referencing requires a colon, so to access i-th character of the string
- Concatenation operator //: to link together 2 or more strings

Now, an array; so, these arrays which you are dealing in so, far these are typically used for numbers they are either, they can be explore they can be integers, they can be real

numbers or they can be imaginary numbers, but what happens or how does one acts create an array containing characters. So, that is done by this type of arrays called strings.

So, string basically if by definition is an array of characters and like the previous cases the arrays which we saw has dimensions this length of the array is given by dimensions in case of strength it is given by the length keyword. Also there is a requirement for referencing sub string then one uses a colon to access the ith character of the string, one can also use this concatenation operator to link basically to connect 2 or more screens strings together.

(Refer Slide Time: 24:26)

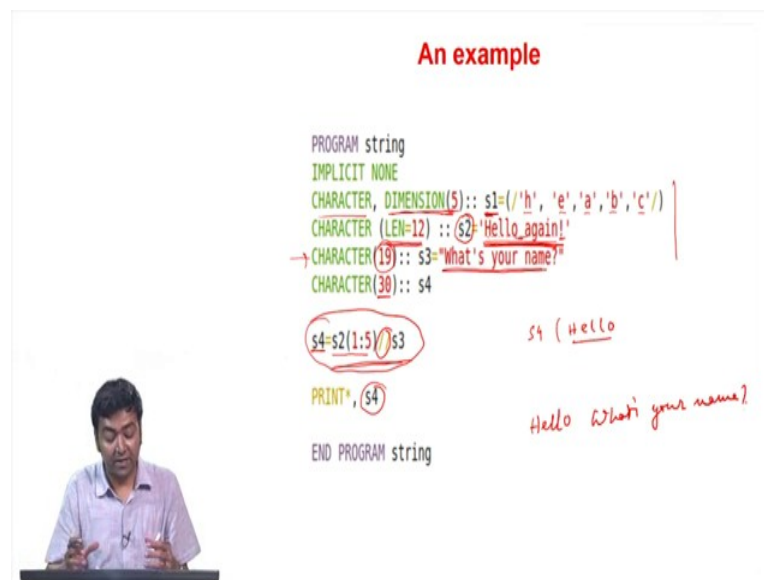
An example

```
PROGRAM string
IMPLICIT NONE
CHARACTER, DIMENSION(5):: s1=('h', 'e', 'l', 'l', 'o')
CHARACTER (LEN=12) :: s2='Hello again!'
CHARACTER (19):: s3='what's your name?'
CHARACTER (30):: s4

s4=s2(1:5)//s3
PRINT*, s4
END PROGRAM string
```

s1 (hello)

Hello what's your name?



So, here is an example of a program where one can use this type of string based arrays. So, one way is you define a variable name which is given which you declare as a character type and you give a it assign it a dimension of 5 and then you separately save these 5 letters. So, these 5 letters will be saved separately, but suppose if one wants to write this statement or wants to store this statement hello again. So, how does one do that in that case one uses this length statement in. So, so if you look at it. So, here there are 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; 10 letters, 10 alphabets one space and one exclamatory sign.

So, basically to store this whole thing as a single array as I have done here in this program in this variable 's2' we need the length of this to be 12. So, that is why I have defined it sizes a length equals to 12. Similarly, you can also in the similar fashion to

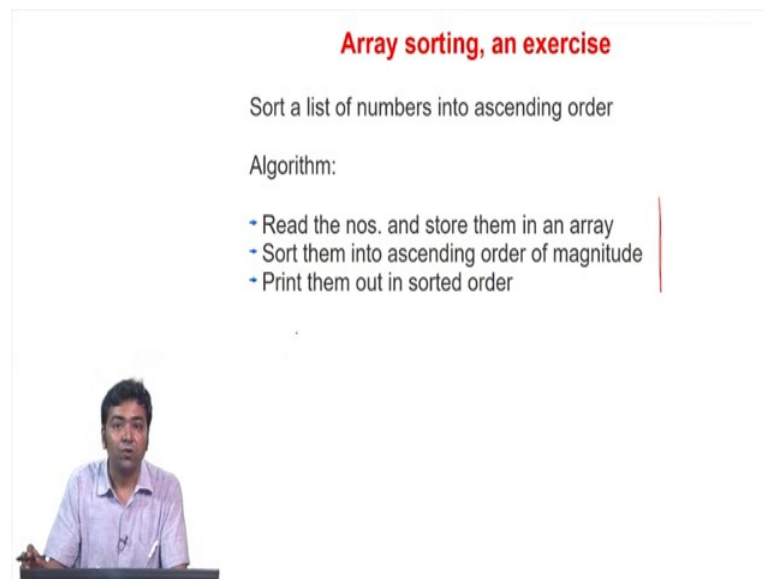
store this you need a character array which is length 12. So, this is another way of declaring the size of the character or array.

So, the thing is that these three stores arrays which contains characters instead of numbers and these are the three different ways typically people use to assign the dimension and this is done. So, what I am doing here in this statement is, so from the s2 array, so I am creating a new array s4 which is assigned the following values.

So, from the s2 array that is from these letters which are written here I take number 1 to 5, so basically, 1 2 3 4 5. So, my 's4' this stores the following numbers the following letters h e l l o and then I connect these elements in the s2 array with the elements of this 's3' array. So, what it will happen is. So, this and this is through this concatenation operator. So, the remaining, so I have here 5 elements my array s4 can store 30 elements, so the remaining 25 elements will be used to store this thing.

So, if you print out the s4 statement, so what you will find is something like this hello what is your name? So, this is how one can connect partake a part of one array another part of a second array connect together and form a new array .

(Refer Slide Time: 27:26)



Array sorting, an exercise

Sort a list of numbers into ascending order

Algorithm:

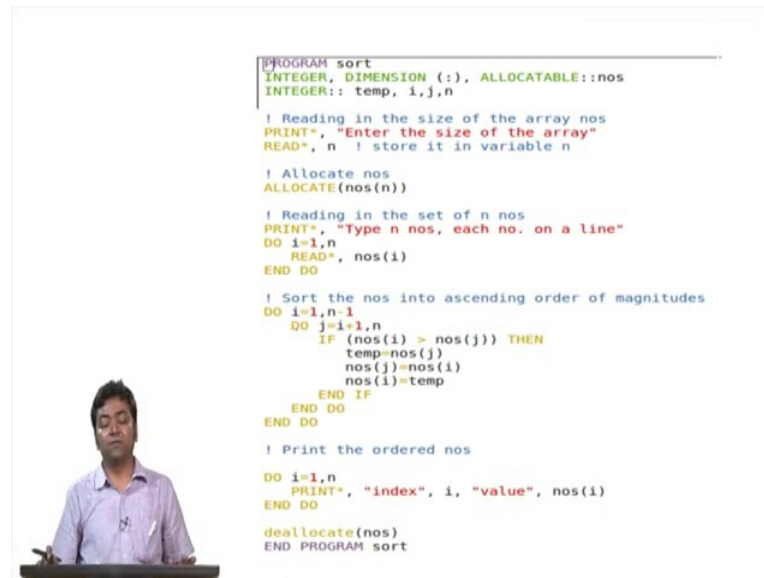
- Read the nos. and store them in an array
- Sort them into ascending order of magnitude
- Print them out in sorted order

The slide also features a small video inset at the bottom showing a man in a light blue shirt speaking.

So, with this we come to an example here to which I have given the solution in this slide, but I mean I would encourage you to use it as a practice exercise. So, the idea is given a set of numbers you need to write a program which will sort them in an ascending order,

so the algorithm is given briefly here. So, first you need to read the numbers from given by the user. So, you need to store them in an array, then you need to sort them into an ascending order and then print out the sorted array.

(Refer Slide Time: 28:05)



So, this is the program, but I would encourage you to write the program yourself and see how one can get a feel of the things. So, with that we are done with the second part of the introduction to Fortran module.

Thank you.