

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


**Lecture - 02**  
**Introduction To Fortran Part-1**  
**Session-B**

(Refer Slide Time: 00:16)

Execution part (steps 3-6)

```
PRINT*, 'Type length in m'  
READ*, input_m  
output_cm = 100.0*input_m  
PRINT*, 'Length in cm=', output_cm
```

Input and output statements



Now, we come back to this part. So, we have seen how to assign variables. Now we will see how to do the input and the output statements.


(Refer Slide Time: 00:25)

**INPUT/OUTPUT STATEMENTS**

Fortran **READ** statement **reads** user supplied **inputs**. Inputs can be read either from a file or from command line  
Eg. **READ\***, input\_m (reads from command line)

Fortran **PRINT** statement **writes the results**. Outputs can be written either in a file or on the screen  
Eg. **PRINT\***, 'Length in cm=', output\_cm (writes on screen)

In both the above examples the input (output) is read (written) format free. One can also read and write data in a formatted form.



So, there are a certain keywords in Fortran which allows us to read inputs either from a screen or from a file. So, the Fortran keyword for that is READ. Similarly it also allows us to write something or the write the output of our program on the screen. So, a Fortran keyword for that is PRINT. So, here is an example of how to read it. So, for example, if I have this line in my program. So, READ star comma input underscore m what it will do is once the user types the number on the screen and presses enter, the code will read the number and store it in this valuable in this variable input underscore m.

Similarly, when I try to write the output of my program here where we which in this case is the length converted into meter. So, I can give us print statement for in the following way. So, PRINT star comma and then I put within codes we in so, this is character array where I just write length in centimetre equals to. So, this tells me or the user who is seeing the output what is the quantity that I am writing out and then I write the I given a comma followed by the variable name which gives the results on the screen.

So, one thing you note that in this both this cases here I have a put a star here which denotes that it is a written in and read in the free format, but one can also do write it or read the data in a formatted form in Fortran.

(Refer Slide Time: 02:08)

**FORMATTED OUTPUT**

Format specifier in parentheses  
Eg. `PRINT '(i5)', j`

For **INTEGER** `nix`


For **CHARACTER** `nax`

For **REAL** in **floating point** `nfx.y`

For **REAL** in **exponential** `nex.y`

n = no. of real/integer/character  
x = field width  
y = no. of characters reserved for fractional or exponential part

Same principle applies for formatted input



And this is done in the following way. So, typically formats are written after the print statement. So, for example, here I have PRINT and then within this inverted commas and within the bracket I write the format statement.

So, for different types of variables one has different ways to write the format statement. So, the first one so, the general form of the format statement for an integer will be written in this. So, the first letter here n denotes how many integer numbers I want to write. The second one 'i' tells me that the numbers which I am going to write in on the screen these are all.... I am sorry these are all integers and then the 'x' tells me the field width the compiler or the program will store to print one such numbers.

Similarly, for so, the format is same for everything just what will change as we go from integer to character is that instead of 'i' here I use 'a'. Now it becomes a bit tricky when we go for a floating point operation. So, in the floating point operation the first 2 parameters remain same. I am still telling the number of variables I want to print on the line and the second one is I am characterizing it by a floating point real number f does that.

Now in the third category that is the filled width is now divided into 2 parts x dot y. So, x tells the total field width for I want to reserve for a given number and then y tells the compile the code how many characters of the fractional part or needs to be printed out of

this total x. Similarly for the real exponential instead of 'f' I replace here for 'e' the same principles also applied for the formatted input.

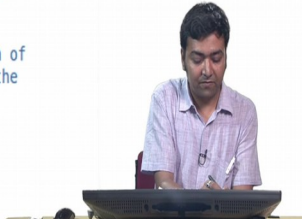
(Refer Slide Time: 04:17)

**FORMATTED OUTPUT: EXAMPLE**

```
PROGRAM format_spec
IMPLICIT NONE
INTEGER:: i=12345
REAL:: j=5.693849284
REAL, DIMENSION (5):: v=(/ 1.1,1.2,1.4,1.6,1.9 /)

PRINT '(i5)', i ! integer in field width of 5
PRINT '(f10.8)', j ! real no. each has a width 10 of
!which 8 characters reserved for
!fractional part
PRINT '(5f8.3)', v !5 real no. each has a width 5 of
!which 3 characters reserved for
!fractional part
PRINT '(e10.3)', j !Exponential no. each has a width of
!10 of which 3 are reserved for the
!fractional part
PRINT '(i5 f15.8)', i, j ! both integer & real
END PROGRAM format_spec
```

*0.58938... x 10<sup>10</sup>*



So, now we will look into an example of how one writes a formatted output. So, here I have a example program where I am trying to write out a couple of numbers on the screen the numbers are integers then they are integers, real numbers and so on and so forth. So, the way this is done is. So, here I have declared a variable I which I have declared as integer and I have set it to 1 2 3 4 5.

Similarly, I have a variable j which I have declared as a real variable and then I have assigned it to this real number. So, if you look at this real number. So, this has 1 2 3 4 5 6 7 8 9 10 11 12. So, basically needs 12 places to display in the screen then I have another array variable which contains consists of 5 such real numbers, that is why I have given the dimension 5. We will learn about array in the later part of this module.

So, the print statement the formatted print statements appeared as follows; the first print statement which I have here tells me that I want to write an integer which will occupy 5 places and their integer variable name is i. Similarly for j which is a real number. So, I use this format statement which tells me f that it is a real number and then it is states me that I want to reserve 10 places for that and this 8 of the number 8 after this dot tells me that I will write 8 places after decimal. So, basically what if you remembered so, this j is

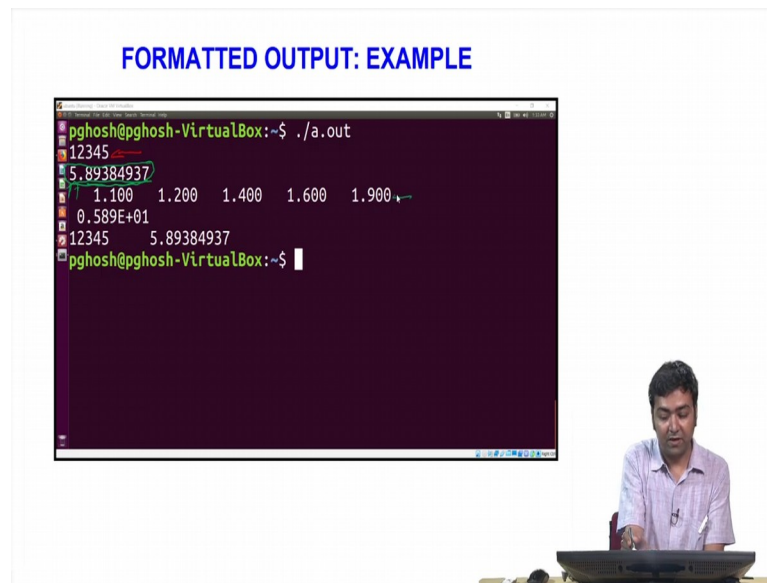
to represent this whole number  $j$  here I need to 15 places instead here I am just assigning 10 places.

So, we will see what happens and very soon. Similarly I can I write all the arrays of this stored all the numbers stored in this array  $v$ . So, as I we have seen here that it has 5 numbers stored. So, I use the following format statement. So, basic first I tell how many numbers I want to print. So, that is 5 tells me that tells the code that I have to print 5 numbers then I tells that it is a real number with this  $f$  and then I say that for each num such number I store 8 places out of which 3 places are for exponential after decimal.

Similarly, I want to write the same number  $j$  here as uniform of a exponential number and that is why I have used  $e$  in the format statement, again I stored 10 places for the number out of which 3 places will be used for the exponential or the fractional part and we can also write for example, in the same line real and integer numbers. So, how do we do that? So, the format statement is for that.

So, say suppose I first want to write my the display the numbers stored in my variable  $i$  which is a real which is an integer variable and then the number stored in  $j$  which is the real variable. So, in the format statement first I write  $i 5$ . So, basically what it tells me is that this is an integer and I reserve 5 places for it and then I give a space here and then this is followed by the format statement for my real part which I say it is a  $f$  which are denote by  $f$ . I reserve 15 places for it out of which 8 will be for fractional part. So, what will be the output of this program?

(Refer Slide Time: 07:52)



So, if I run if I compile and run this program. So, this is what I will I get on the screen. So, the first number here in the screen that is this 1 2 3 4 5. So, if you go back to the program we will see that the first print statement is we have asked the code to print the number stored in the integer variable i and for 5 places.

So, that is precisely what it has done. Now, the second print statement which was there in the code is I write down the output the number stored in j which is and I use 10 places for that and out of that 8 will be after the fractionation and as I told you before that as we counted before that the number j requires a total of 12 places to print it.

So, what the code will do is. So, and also we would like I would like to point out here that after the decimal point. So, we need 1 2 3 4 5 6 7 8 9 and 10, 10 places to write down this whole part fractional part , but instead we have allocated here 8 places for that same thing. So, what the code will do is it will write the it will the, what the code will do is it will basically write the code will write the first 8 digits after decimal.

So, that is 4 6 8. So, till this point and typically rounds off after the last decimal place and this is what we precisely see here in the in this output statement which I have taken the screenshot from my laptop. So, I have total 10 places. The number j has been expressed using 10 places out of which there is one place for the decimal point and then one is the part before the decimal and then 8 places after the decimal.

Similarly if we go back here; so, for the array v we have said that we will write 5 numbers here and then these are all real numbers and we replace 3 places after decimal. So, this is what precisely what we see here given in this line of the output and then the second last print statement where we are trying to write this j in an exponential format.

So, what we have is we have stored 3 places for the fractional part. So, that is why so, if we take this number. So, the fraction in exponent form it will read as 0.5938 into 10 to the power 1. So, this will be in the exponent form and since we have kept 3 places after decimal here to print it.

So, that is why it is printing 0.589 with the then e to the power 01 and then the final print statement here which we have is a mixture of integers and a fractional number and that is what the code is printing out. So, this is the way we write the print statement in a or write the output on the screen on the display screen using a formatted for in a formatted form using the print statement.

(Refer Slide Time: 11:04)

### FILE INPUT/OUTPUT

File I/O in Fortran works via "unit numbers"

Unit nos. for files are issued by using the OPEN keyword

I/O is done using WRITE (similar to PRINT) and READ

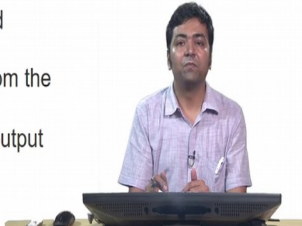
WRITE(<unit\_num>, <fmt>) vars

READ(<unit\_num>, <fmt>) vars

Unit nos. are released using the CLOSE keyword

unit\_num=5, is the default value for reading in from the command line

unit\_num=6, is the default value for printing the output on a screen



Now, we can also, but this will be very infrequently used, but the most common use of the input output statement is we typically write the output of the program into files. So, what one needs to do is one needs to have several files with some given file names which the user decides or the programmer decides and then there are some numbers which are associated. These are called typically unit numbers which are associated with a particular file. So, the unit numbers for files are issued by the using the open keyword.

So, this open keyword along with the number unit number of that particular file tells the Fortran code that we need to open this file with this associated with this particular unit number and the input output is typically done using the write statement. Remember when we put it on the screen we use a print statement, but in the file to put the output in a file we use a write statement here and the Read is done by again as the Read statement we saw.

So, the format is the same way as the print statement. So, for example, write we use WRITE then we give the unit number which is my file identifier then I give my format statement and then the list of variables. So, remember the number of format statements here and the number of variables here those things should be matched. Similarly in the same way you can use the READ statement to read data from a file. So, you give Write Read and within parenthesis you write the unit number here and also the format statement.

So, this here this format statement should be in the same format as the data is present in the file which one wants to read in. If there is a mismatch of the format statement the code typically complains and then you write the list of variables you want to read in from that particular file. So, once you have done with the reading and writing of the file. So, what is typically done is one uses this close keyword to close the file and regarding then unit numbers. So, there are 2 things which one needs to be careful of.

So, first of all one should not ever use unit number 5 and 6 to name to connect the name of the file with that particular unit number. The reason for that is 5 is the default value unit number equals to 5 that is the default value for reading in from the command line. Similarly unit number equals to 6 is the default value for printing the output on a screen.



(Refer Slide Time: 13:43)

### FILE INPUT/OUTPUT: AN EXAMPLE

```
PROGRAM output_file
IMPLICIT NONE
INTEGER:: u=7, ios ! unit no, input output status == 0 is OK
REAL:: j=5.8938492847
REAL, DIMENSION (5):: v=(/ 1.1,1.2,1.4,1.6,1.9 /)


! ampersand sign does line continuation

OPEN(UNIT=u, IOSTAT=ios, FILE='myfile.txt', STATUS='new', &
     ACTION='write')

IF (ios .eq. 0) THEN
WRITE(u, '(5f5.1)') v
CLOSE(u)
ELSE
PRINT '(a30)', "Error: file not open"
END IF

END PROGRAM output_file
```

IOSTAT +ve => previous read have some error  
IOSTAT=0 => previous read has been executed correctly  
IOSTAT -ve => end of file



So, here we have an example of how to write in a formatted way in a input file. So, if we look these variables here are the same the numbers as in the previous example which we saw. But, the difference is in the previous example we are trying to write it on the screen here we will try to write it on the file.

So, what do we need to do. So, first of all we need to have a file. So, we need a file name, we need to have a unit number associated with the file and then we need to ask the code to open the file then we need to ask the code to write into the file and we need to close the file and then we come to the end of the program.

So, how this is done is. So, for the unit number we have used integer variable which we have set to the value 7. So, basically and I have the output file I have given the name as myfile dot txt. So, as I mentioned in the previous slide. So, I ask the compiler to the program to open the file using this open statement then after the open statement I have the unit number this IOSTAT is gives me the status of the file. So, IOSTAT typically can take 3 types of variables.

So, what it tells is that the previous time when we one started to read this file there is some error. IOSTAT equals to 0 means the previous read has been executed correctly and IOSTAT equals to negative means that there is the end of the that the that the file has ended there are no more lines to read. So, basically this variable is gives what is the status of the file so, but this is not so important and also and then we need to specify the

file name this is done with the file attribute here this one file equals to and then within single inverted commas I give one file name and then I need to mention that what is the status of the file is this a old one which is already there or is this a new one.

So, for that one uses this status statement and if since we are creating the new file here. So, we have assigned it to status equals to new and we can also tell the code what to do with this particular file. So, that is whether are we interested in reading data from that file or writing data from that file. So, in this case we will be writing data on to the file and. So, that we do with this action variable. So, action so, we said action equals to write. So, then this part of the program is where we are writing the data.

So, before writing the data so, what we do is we check what is the status of the files. So, if I we will come to this if statement here. So, basically what this means is that if this IOSTAT variable is set to 0 then only we will write and if it is not set to 0 we will not write anything and we will print this error message on the screen. If this file is there is no issue with this file then what it will do is it will write. So, then it will go into this line of the program the roller it will execute this step. So, what it will do is write on..... So, this is as I said the file unit number and which is equals to 7 here in my case and we associate this unit number with this file.

So, this file is already open with the help of this open statement. The program the code knows ,the computer knows, that we need to write. So, it will go to this file containing the unit number 7 it will read in this format statement. So, basically what I am doing is I am writing the variable v here the number stored in this array v. So, since as I mentioned in the previous slide that this has 5 numbers.

So, I am using a format statement which tells me that tells the code that there are 5 numbers to be printed in this line and then since these are real numbers. So, I am using f then I am using 5 places for that and then one place after decimal. So, once this statement is executed then what we need to do is we need to close the file. So, we use the close command and we are done with the job.

So, this is how one writes output in the in a formatted fashion. So, if one wants to read in some data which is already existing in the file. So, what one needs to do is one basically needs to replace this write statement with a read statement and this formatting statement

would should be the same should contain the same formatting attributes as the those of the data present in the file from which we are going to read.

So, with this we come to the end of this first part of this Fortran lecture. So, where we have seen how to read how to; so, what is a typical structure of a Fortran program and how to end and how to assign different types of variables how to do mathematical operations and how to write and read data on the screen and on the file.

Thank you very much.