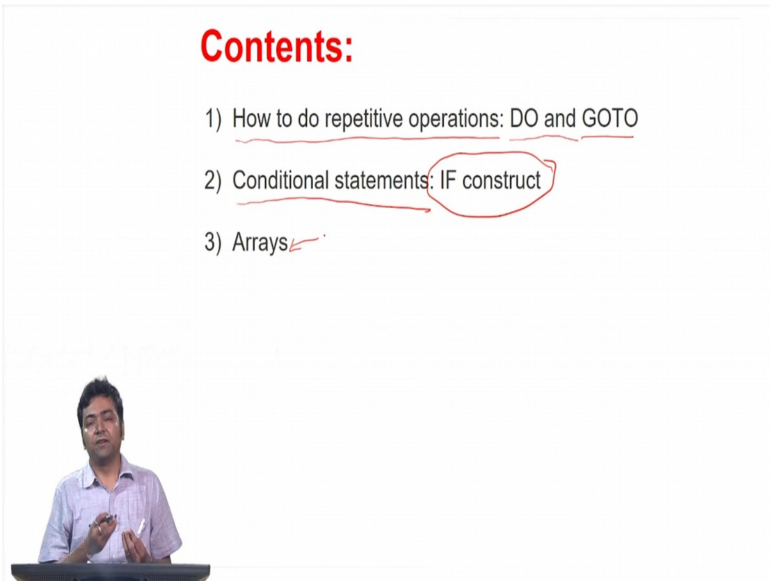
Computational Physics

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

Lecture - 03
Introduction To Fortran Part-2
Session-A

Welcome back. So, this is the second part of the Fortran module introduction to the Fortran module.

(Refer Slide Time: 00:24)



Contents:

1) How to do repetitive operations: DO and GOTO
2) Conditional statements: IF construct
3) Arrays

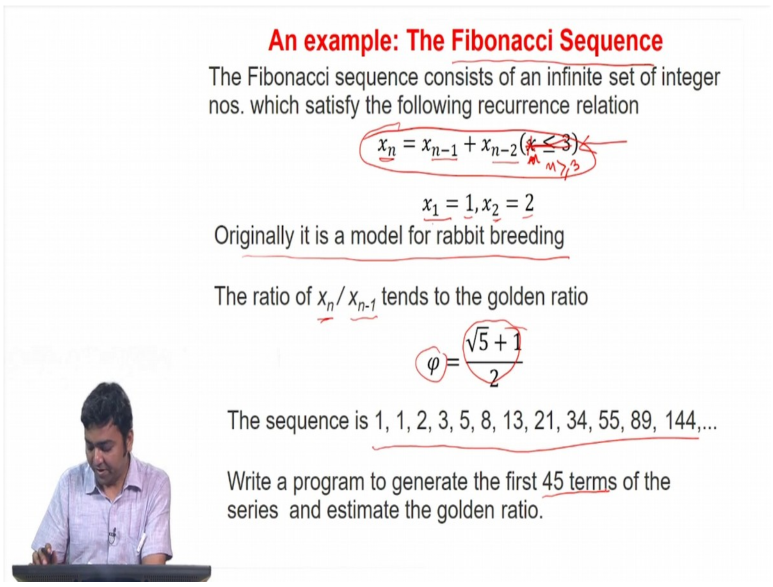
And here we will learn the following 3 things. So, when one writes a code this is something general this is nothing specific about Fortran code. So, one needs to repeatedly do ask the code to do some stuff or a set of instructions, execute a set of instructions. Also one needs to in order to do that what in Fortran the first topic we learned in this part is how to do repetitive operations and these are typically done by these 2 variables the DO and the GOTO statements.

Another thing which we often come across when writing a program is for example, we want to execute a certain statement only if certain conditions are satisfied. So, we will also see in this part how to use conditional statements in Fortran using the IF construct. And, the third thing which we will cover in this part is, suppose I have a large number of

I mean a large set of numbers which contains similar information and I. So, in order to store them we need to resolve space and also we need to have different types of variable names.

So, but for the same type of numbers when I have a large set of such numbers then what instead of using individual variable name for each such numbers I can use something which is called an array and we will see what it is. So, basically the purposes of an array is it helps us in reducing the usage of a large number of variable names, ok.

(Refer Slide Time: 02:15)



An example: The Fibonacci Sequence

The Fibonacci sequence consists of an infinite set of integer nos. which satisfy the following recurrence relation

$$x_n = x_{n-1} + x_{n-2} (\cancel{x \leq 3}) \; n > 3$$

$$x_1 = 1, x_2 = 2$$

Originally it is a model for rabbit breeding

The ratio of x_n / x_{n-1} tends to the golden ratio

$$\varphi = \frac{\sqrt{5} + 1}{2}$$

The sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144,...

Write a program to generate the first 45 terms of the series and estimate the golden ratio.

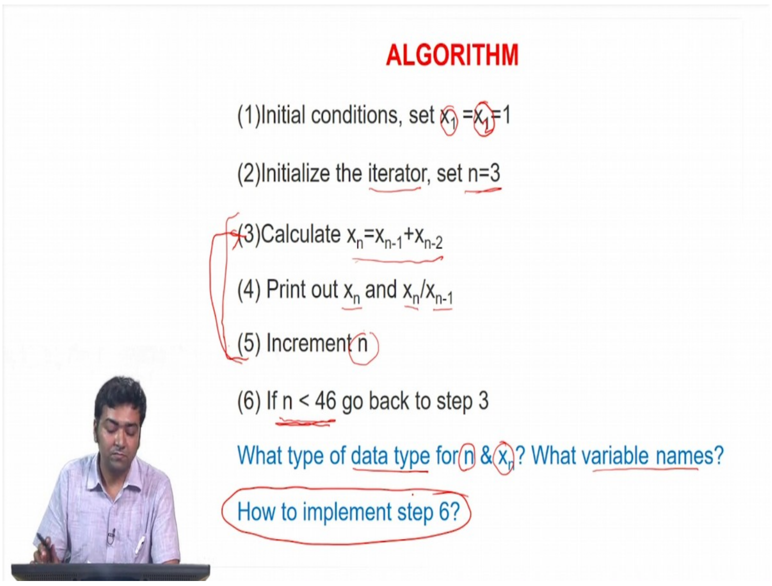
So, to see how to do these things we start with an example. So, what we are going to do in this example is we are going to generate a set of numbers and such a set of numbers. So, for example, 1 1 2 3 this way and such a set of numbers is called the Fibonacci sequence. And, it is a very famous sequence and it is this sequence is used originally for modelling the rabbit reading, you can find it in spirals in sea shells the spiral formation, all those things can be explained using this Fibonacci sequence.

So, the basic idea of Fibonacci sequence is its contains a series of numbers, it is an infinitely large series and it what it the way the series is generated is using this recurrence relation . So, the what this recurrence relation tells is that to get or to the nth number in this series we need to know the n minus 1th number in the series and n minus 2th it number in the series, we add them we get the nth number. So, puts a restriction on the value of x in the number.

So, basically we start from sorry this will not be x this will be n. We start from n less than, I am sorry this is...... So, the restriction is n greater than or equal to 3. I am sorry for the mistake in this slide and my x 1 and x 2 that is n equals to 1 and n equals to 2 these are already set to 1 and 2 and another interesting property of this series is that the ratio of x n and x n minus 1 this always gives this number which is called the golden ratio 5.

So, what we are going to do is we are going to write a program that generates the first 45 terms of the series and estimate the golden ratio. So, how do we do that?

(Refer Slide Time: 04:22)



ALGORITHM

(1)Initial conditions, set $x_1 = x_1 = 1$

(2)Initialize the iterator, set n=3

(3)Calculate $x_n = x_{n-1} + x_{n-2}$

(4) Print out x_n and x_n/x_{n-1}

(5) Increment n

(6) If n < 46 go back to step 3

What type of data type for n & x_n? What variable names?

How to implement step 6?

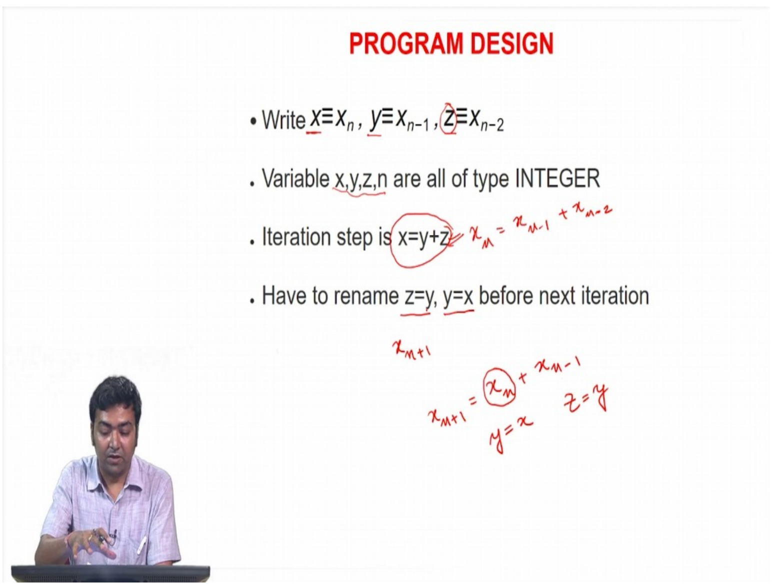
So, here is the algorithm. So, we will set the initial condition that is we will set x 1 equals to 1 which is my initial condition and sorry.... x 1 equals to 1 and x 2 is also equals to 1 which is the initial condition then we initialize the iterator because, to generate the third number we need to use the x 1 and x 2 then to generate the fourth number we need to perform the same set of operations mentioned in this equation here.

But by using now x 2 and x 3 and so on and so forth we need to do this stuff repeatedly. So, that is why we need to have an iterator which can do this iterative job and the start and we need to have a counter also. So, that will tell us how many times this iteration is done. So, we set that iterator to n equals to 1 and then the third step is we calculate that expression the nth number in the series and then we print out the nth value and the ratio of the n and n minus 1 and then once we are done with that we increase the iterator.

So, basically a n will now be set to n plus 1 and what we do is. So, we as I mentioned we want the first 45 numbers in the series. So, once the iterator is incremented. So, the largest value of n should be 45. So, if it goes to so, this step from 3 to 6 this will be repeated only as long as n is less than 46 and once we have run this increment. So, what we need to tell the program is that you carry on this till n takes a value with the value of n is less than 46 and what you need to do is still that value once you reach this point you need to go back to step number 3 and do the same process.

So, how does the program design look like? So, the questions here is what will be the data type for n and x n?. What variable names we can use and the new thing which we are going. So, these two we have already seen in the previous part and the new thing which we are going to learn in this example is how to implement step 6.

(Refer Slide Time: 06:57)



PROGRAM DESIGN

- Write $x \equiv x_n$, $y \equiv x_{n-1}$, $z \equiv x_{n-2}$
- Variable x,y,z,n are all of type INTEGER
- Iteration step is x=y+z $x_n = x_{n-1} + x_{n-2}$
- Have to rename z=y, y=x before next iteration

x_{n+1}

$x_{n+1} = \boxed{x_n} + x_{n-1}$ $\quad z=y$

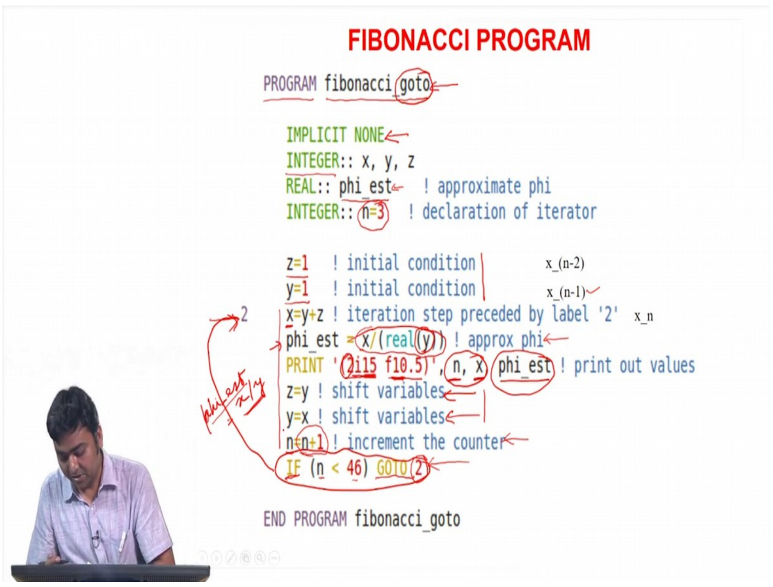
$y=x$

So, the program design will look as following. So, we will use so, at a time at a given instant of time if you see we always need 3 numbers x n x n minus 1 and x n minus 2. So, and all these numbers are integer numbers.

So, basically what we need is 3 variables we will be using here x y and z where x I have assigned to in x I will store x n my new number y I will store x n minus 1 the previous number and at z I will store x n minus 2 which is the second previous number. And, since these are all integers I have set x y n all are as integers time and the iteration step which I need to do the math that is the mathematical operation which I need to do repeatedly is x equals to y plus z that is sum of x n. So, this is nothing, but you have x n equals to x n minus 1 plus x n minus 2. So, this is the step which I am doing.

So, now once this is done so, for the second step; so, that is suppose I have done x n now I want to do x n plus 1. So, for x n plus 1 what I will be having is x n plus 1 equals to x n plus x n minus 1. So, what it means is that. So, what was earlier x n minus 1 will now become x n for the nth step. So, what we need to do is we need to set y to x as I have mentioned here and what was earlier x n x n minus 1 for the n plus 1th step this will be x n minus 2. So, and x n minus 2 I am storing in the variable z. So, what it means is that I have to set z to y as I have mentioned here.

(Refer Slide Time: 09:01)



FIBONACCI PROGRAM

```
PROGRAM fibonacci_goto

   IMPLICIT NONE
   INTEGER:: x, y, z
   REAL:: phi_est     ! approximate phi
   INTEGER:: n=3    ! declaration of iterator

   z=1   ! initial condition            x_(n-2)
   y=1   ! initial condition            x_(n-1)
2  x=y+z ! iteration step preceded by label '2'   x_n
   phi_est = x/(real(y)) ! approx phi
   PRINT '(2i15 f10.5)', n, x, phi_est ! print out values
   z=y ! shift variables
   y=x ! shift variables
   n=n+1 ! increment the counter
   IF (n < 46) GOTO (2)

END PROGRAM fibonacci_goto
```

So, this is the basic design of the program. I have the example program here. So, what it will look like is the following. So, again as I mentioned in the previous lecture.

So, I start the program with this program thing and then I give a name of the program. So, the program generates the Fibonacci series using this new Fortran function which is a goto. So, as usual I still the code I tell the machine that do not assume anything about the type of the of the variables I am using then since I am dealing with only integer numbers I am declaring x y z as integer variables; however, the ratio one should remember is a real number. So, I am storing the ratio of x n and x n minus 1 in this variable called phi underscore estimate est which is a real type and then n is my declaration of the iterator which I set to 3 because. So, my x 1 and x 2 are always 1.

So, once I have declared my variables now I come to the main part of the program. So, first I set the initial conditions. So, remember z as I mentioned here also z is my n minus 2 and y is my n minus 1. So, the for the first two numbers of the series are always 1. So, I have set z equals to 1 and y equals to 1 as the iteration condition and then I proceed to compute the value of x that is the nth number in the series which is nothing, but the sum of n minus 1 and n minus 2 numbers and that is what I have done.

So, basically in the variable x here I am assigning the sum of the variables y and z. So, once that is done what I have done is now I am estimating here the ratio which is stored in this variable phi. So now, notice one thing. So, here my x and y both are integers. So, had I written instead of writing it like this had I written that phi underscore estimate is equal to x by y. So, this division would have been a integer division because x and y are always integer x and y are integers here. So, I would not get the correct estimate of the phi.

So, instead what I have done is I have converted one of the numbers so, in this case y to a real number. So, that now this mathematical operation which is a division will be done in terms of will be a real of mathematical operation instead of an integer operation then once I am done with estimating computing the value of the phi. So, I want to print 3 things here. So, first is the iteration number that is which nth which number of the series is this which is denoted by n then I print the number x and then the estimate of the ratio of n and n my n minus 1.

So, what I want to print is I want to print 2 integer numbers and 1 real number. So, that is why in the in the form and I want to print it a formatted form. So, that is why within the format statement I have here since I am having 2 types of numbers. So, I have 2 format statements one for integer part and one for the real part and for the integer part I am printing out 2 integer numbers. So, I am using 2 here and then I am keeping a space of reserving a space of 15 for each of them and then for the estimate of phi I am reserving a space of 10 and I am writing 5 places after decimal.

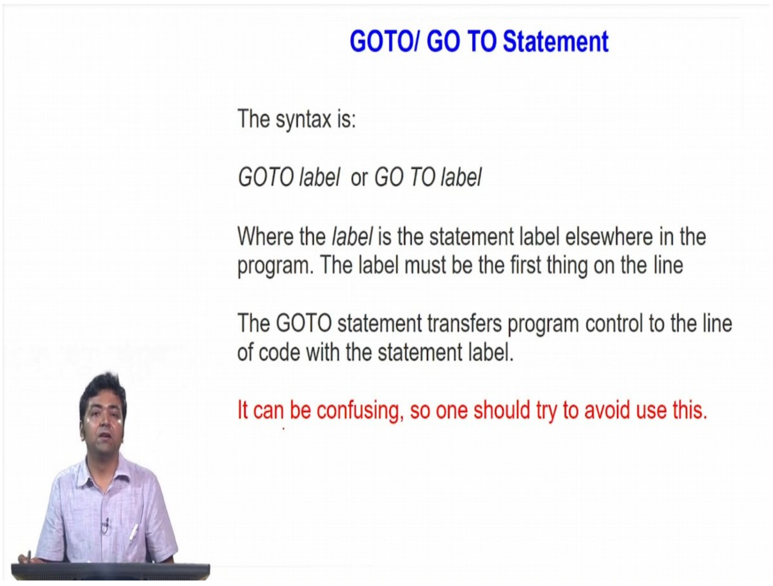
So, once I have printed this value on the screen. So, now, I need to reset my variables. So, what was earlier y that is x n minus 1 will now become x n minus 2 and I need to store it on z. So, here in this statement in this line what I am doing is I am reassigning the value of y to the variable z and in a similar fashion I am also reassigning the value of x to the where to the variable y.

So, once this reassignment is done. So, I have my new sets of n minus 1 and n minus 2. So, once these are done then I reset I increase the counter by 1. So, this now becomes n 2. So, I reassign n by using this assigning statement basically I am adding a value of 1 to the previous value of n. So, once it is done then I come to this part of. So, this is the new part which we are going to learn in this. So, here I have 2 statements one is a conditional if statement and another is a goto statement.

So, what this statement does is the code checks whether the value of n is less than 46. So, as long as the value of n is less than 46 whenever controller comes to this line what it will do is it will go to a line which is labelleded by this number 2. So, let us say where in the program I have labelleded this line by 2. So, we can see that here is my leveller.

So, what this will do is that when the moment the controller comes here it the moment the machine reaches here this part of the code this last line it goes back to the controller is taken back to this part of the code and then the whole thing is again executed repeatedly.

(Refer Slide Time: 14:51)



GOTO/ GO TO Statement

The syntax is:

GOTO label or *GO TO label*

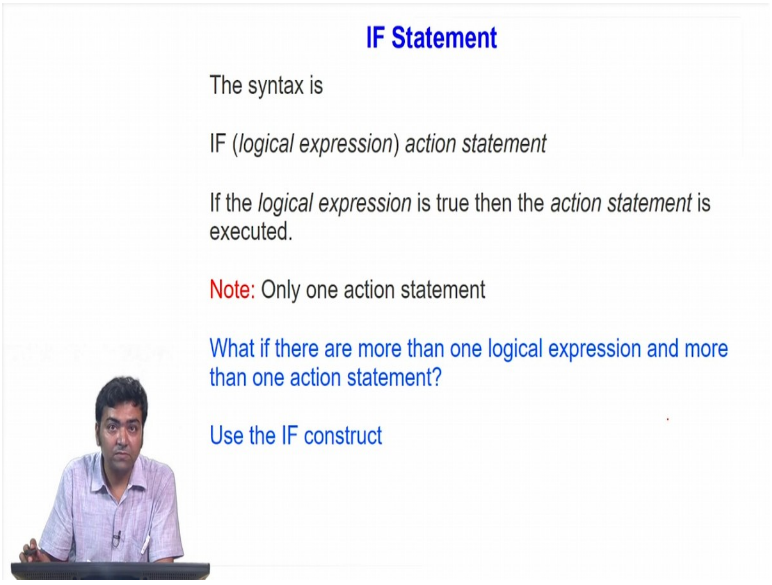
Where the *label* is the statement label elsewhere in the program. The label must be the first thing on the line

The GOTO statement transfers program control to the line of code with the statement label.

It can be confusing, so one should try to avoid use this.

So, this is one way of doing a task repeatedly in a program or telling the computer how to do a task repeatedly in the program. So, that is by using the GOTO label. So, you can write either together go to this is the syntax or you can write in a separate way and also you should have the label which comes in the at the beginning of the line, but it is not always I mean this is one way to do it, but it is not the most efficient way and you can make mistake in putting the label. So, if I mean for all practical purposes one should try to avoid this thing.

(Refer Slide Time: 15:25)



IF Statement

The syntax is

IF (*logical expression*) *action statement*

If the *logical expression* is true then the *action statement* is executed.

Note: Only one action statement

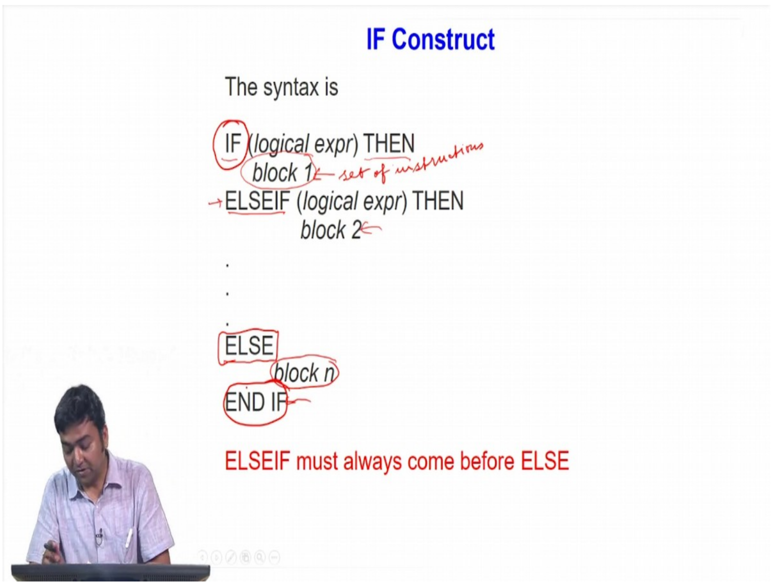
What if there are more than one logical expression and more than one action statement?

Use the IF construct

So, another important attribute of the Fortran which we saw is the IF statement. So, the syntax is very simple. So, what IF statement does is it allows us to do a perform a logical task. So, the syntax is very simple I have if then in the bracket I need to give a logical expression which if satisfied the code will do this following action.

So, here in this way the writing it, the IF statement in this way we can perform only one action if the condition is satisfied. Now, suppose if we have to perform a set of actions if when the logical expression is satisfied then how do we do that.

(Refer Slide Time: 16:12)



IF Construct

The syntax is

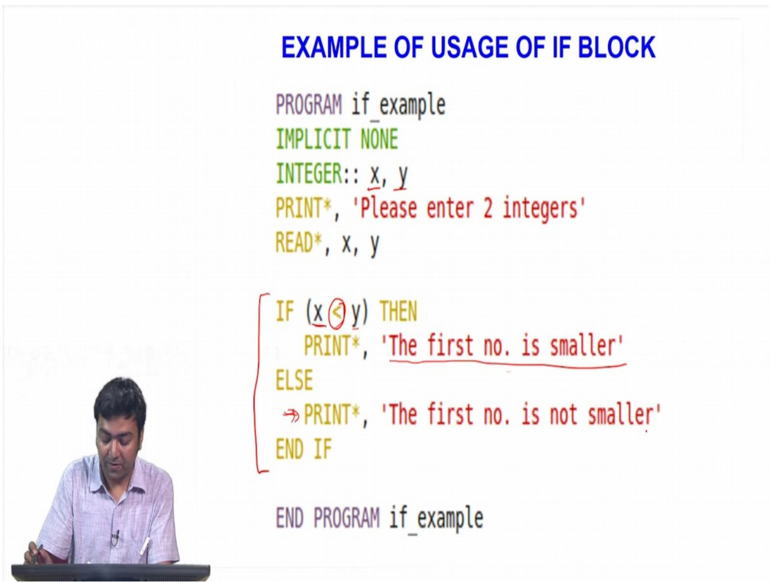
```
IF (logical expr) THEN
   block 1 ← set of instructions
ELSEIF (logical expr) THEN
         block 2
.
.
.
ELSE
     block n
END IF
```

ELSEIF must always come before ELSE

So, in order to do that we use what is called the IF construct block. So, the syntax is in the following. So, you start with IF and THEN you have your logical expression. So, if that logical expression is satisfied then you have this then you go to block 1. So, block one basically contains a set of instructions ; if however, this logical statement is not satisfied then you can come to the next condition that is by using this ELSEIF statements. So, the syntax is again similar ELSEIF then the logical expression then the if this logical expression is satisfied then it comes to block 2 and so on and so forth.

So, in this way one can put several else if blocks within the full IF and the END IF, but remember that the last block should be preceded by the ELSE statement. This you should be careful of. So, once this is done then you end the construct loop the IF construct with a END IF. So, basically the point is, one should remember that if you have a IF THEN you must have a END IF. So, this is something which one needs to be careful of.

(Refer Slide Time: 17:37)



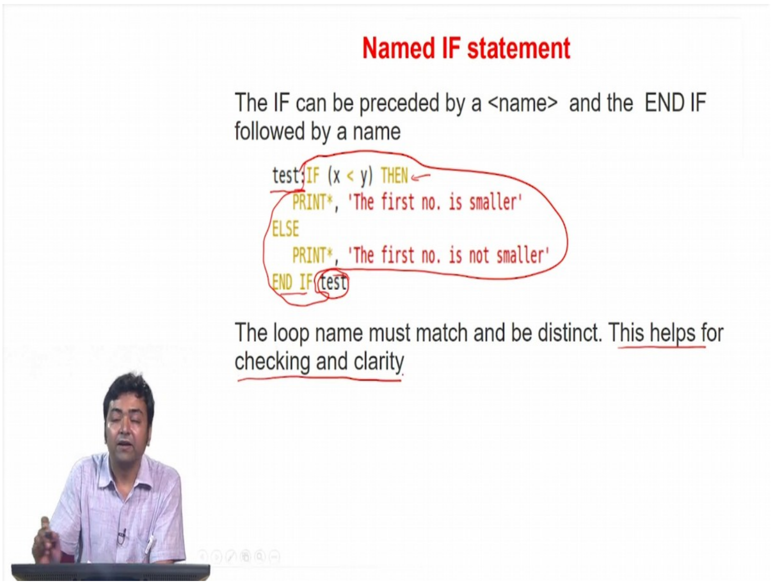
EXAMPLE OF USAGE OF IF BLOCK

```
PROGRAM if_example
IMPLICIT NONE
INTEGER:: x, y
PRINT*, 'Please enter 2 integers'
READ*, x, y

IF (x < y) THEN
   PRINT*, 'The first no. is smaller'
ELSE
   PRINT*, 'The first no. is not smaller'
END IF

END PROGRAM if_example
```

So, for example, this program shows the usage of a IF block. So, I have 2 variables x and y and what I want to do is these variables are typically numbers and I want to check and I want to check whether if x is less than y then I write down that the first number is smaller and if I and if it is not so then the second number is smaller. This is what I want to do. So, how one does it one does it through the IF statement.

So, I use the IF construct in the following way I say check whether x is. So, this is my conditional statement I check whether x is less than y. If this condition is satisfied then the code will what we will do is we will print this line on the screen. If this condition is not satisfied that is if x is greater than y then what the code will do is it will not execute this block it will come to this block and then it will write this statement that the first number is not smaller and then the program will stop.

(Refer Slide Time: 18:39)


Named IF statement

The IF can be preceded by a <name> and the END IF followed by a name

```
test: IF (x < y) THEN
   PRINT*, 'The first no. is smaller'
ELSE
   PRINT*, 'The first no. is not smaller'
END IF test
```

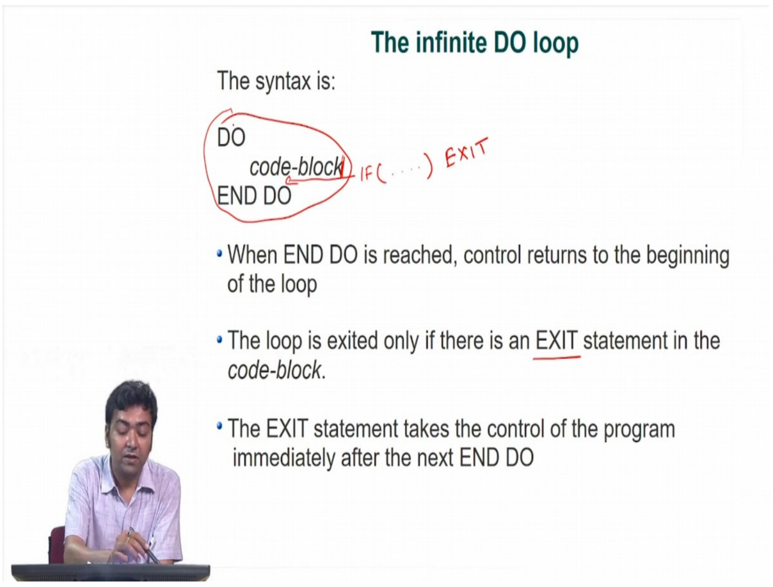
The loop name must match and be distinct. This helps for checking and clarity

Also in the program what might happen is you can have several and thousands of these if statements. So, to help you go help the programmer or the person who is writing the program to go through easily. So, what one can use is also something called the named if statement that is for if each block which you are using you assign it to a certain name.

So, this is done by the following way. For example, suppose I want to assign a name to this same block. So, what I do is this is my the Fortran program which is doing the this is my IF construct. So, just before the IF construct I put the label. So, the label is written in the following way.

You give a name to the label followed by a colon and then you start your if construct and once you are done. So, after the end if you again repeat the name of the labeel; so, the advantage of this is, this gives a clarity in your program and helps you in checking mistakes or debugging

(Refer Slide Time: 19:44)



The infinite DO loop

The syntax is:

```
DO
    code-block   IF(....) EXIT
END DO
```

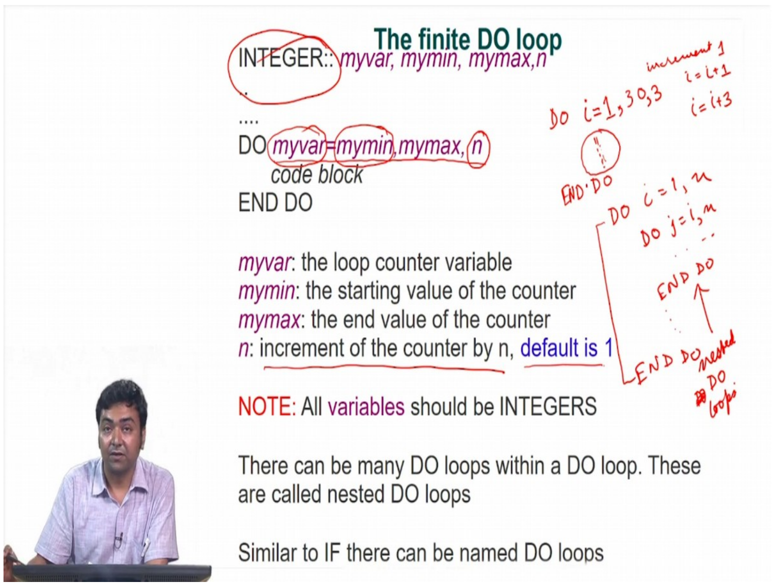
- When END DO is reached, control returns to the beginning of the loop
- The loop is exited only if there is an EXIT statement in the *code-block*.
- The EXIT statement takes the control of the program immediately after the next END DO

So, this so, till now we saw how to execute conditional statements and then also we saw one I would say crude way of doing a set of instructions repeatedly. So, a better and a sophisticated way to do it is using what is called the DO loop. Now the DO loops are of 2 types. So, one which in I am showing in this slide this is a infinite DO loop. So, the code the syntax is the following. So, you start with a DO and then you have a code block which carries the set of instructions that one needs to execute repeatedly and then you have a END DO.

So, if you have a something like this in your program. So, this the program will never stop. So, it will continue executing it infinitely. So, basically what you need to do is you need to tell the call the program when to stop and that is done by a exit statement. So, inside this code block you should put an exit steps statement something like this. If some condition is satisfied then exit to prevent infinitely carrying on with the set of operations in the loop.

So, the moment this condition is satisfied what the controller will do it you do is it will come out of the END DO loop.

(Refer Slide Time: 21:04)



The finite DO loop

```
INTEGER:: myvar, mymin, mymax,n
.
....
DO myvar=mymin,mymax,n
    code block
END DO
```

myvar: the loop counter variable
mymin: the starting value of the counter
mymax: the end value of the counter
n: increment of the counter by n, default is 1

NOTE: All variables should be INTEGERS

There can be many DO loops within a DO loop. These are called nested DO loops

Similar to IF there can be named DO loops

You can also have finite DO loop that is you can restrict or the number of times you want to execute a set of statements and the syntax for this is very similar to the infinite DO loop only the new part which is present here is this 3 ,4 set of integers.

So, basically the first one this my variable here ‘myvar’ this acts as a counter. So, for example, I can write say if ‘i’ is my counter. So, then I tell the code how many times from which value the counter should start and till what value it should go.

So, I can say for example, the counter should start from ‘i’ which is my the minimum value here this variable and say it should go to 30 and then I have my set of instructions here and then I put the Enddo. So, in this case after every time this set of instructions which are present here is executed the counter the increment is 1. So, basically i is said to the code automatically sets i to i plus 1, but suppose I do not want that I want the increment to be 3.

So, basically what I want is the counter should go from i equals to 1 to i plus 3. So, how do I do that then I add another number here which I say is 3. So, this is my this number here. So, this is basically the increment of the counter. So, this tells the code that you increase the counter by this number n and as I mentioned before the default value for the same is say 1; another thing we should keep in mind is all these 4 variables which are using that is my the counter, the minimum value of the counter, the maximum value of the counter and the increment all these should be integer numbers and similar to the if block you can have many DO loops nested in one inside the other for example, suppose I have a DO loop i equals to 1 to n then END DO.

Then I can have another I can start another DO loop we which starts from a different counter say i i to n and I have a set of operations like that in this. So, you can have several DO loops inside one main DO loop. So, these are called nested DO loops. So, so far we have seen how to execute conditional statement and also how to pass the program or the computer to do a set of instructions repeatedly using the if construct, the goto statement and the DO loops.