

Computational Chemistry & Classical Molecular Dynamics
Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology – Bombay

Lecture - 09

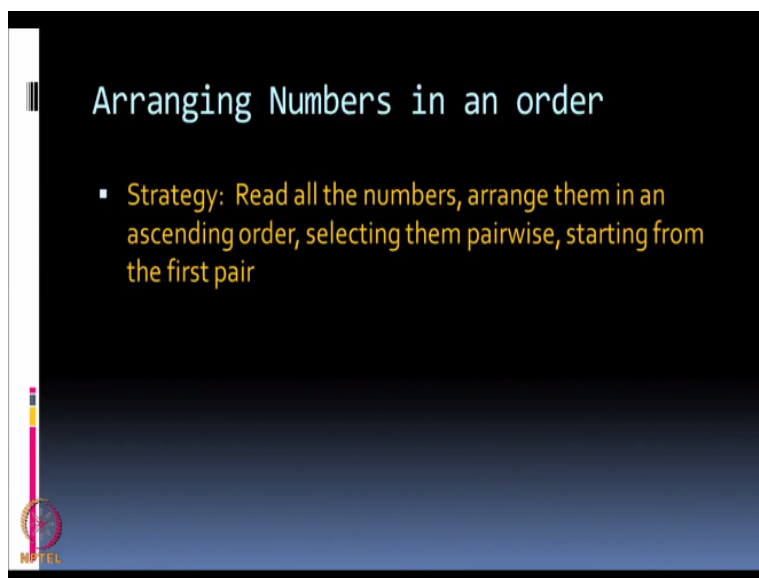
Programming Techniques 6. Functions and Subroutines, Arranging Numbers in as Ascending Order

Hello and welcome again. In our last class, what we did was we discussed the little bit about format statements. What the format statements do for you is to enable you to write your input as well as output in a specific arrangement in your file right. You can compartmentalize your columns into different ways, your different lines, so that you can read it in a systematic way and it is useful when you have large number of variables to be written or read.

And we also started a program for arranging numbers in an ascending order. So what we said that if you want to exchange two numbers a and b, you just cannot make $a=b$ and $b=a$ because if you do this then both a and b will have the same value. The only way you do it would be to assign a temporary variable in which you store the initial value of a, then say $a=b$ and then $b=\text{the temporary name into which you had stored the value of a}$.

So to arrange several numbers in an ascending order what is our strategy? The strategy is let me make it bigger.

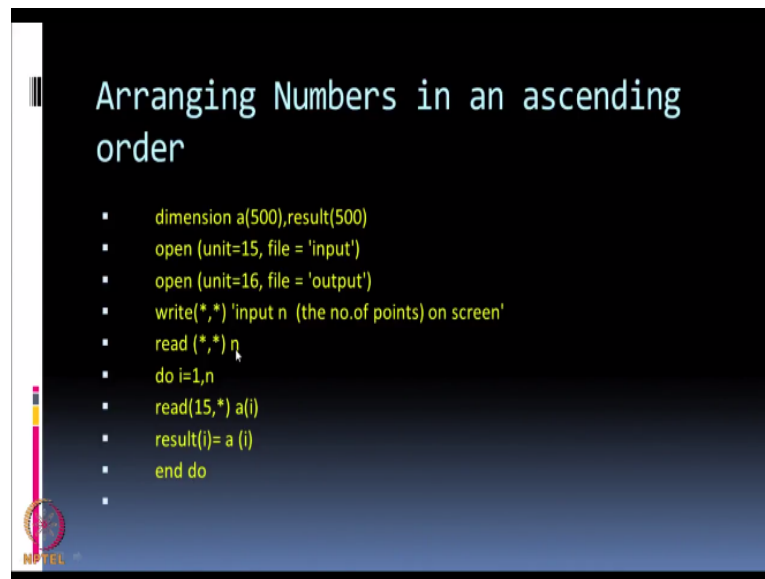
(Refer Slide Time: 01:32)



So the strategy would be to read all the numbers then arrange them in an ascending order, selecting pairwise, starting from the first pair then your final array would be a set of numbers

in an ascending order. So what is the first thing we need to do? We need to read all the numbers, so that is what you will do now.

(Refer Slide Time: 01:48)



So how you will read all the numbers? So to do that first thing is I have a dimension statement. It is an array variable, so there is an array a of 500 values and result which is also an array of 500 values. I will use this result array to store my a values in the result, so that I can work with the result keeping the a array unchanged. I am going to read all my data from unit 15, so what have I had open unit=15 file=input.

What this does for me? The file input is associated with the number 15 in the program. Similarly, the file output is associated with line number with number 16 in the program. So before I start reading from the files, I want to know how many numbers are there to be read. So I will write in my program this particular line, write star, star input n the number of points on the screen, number of points means number of data points to be read.

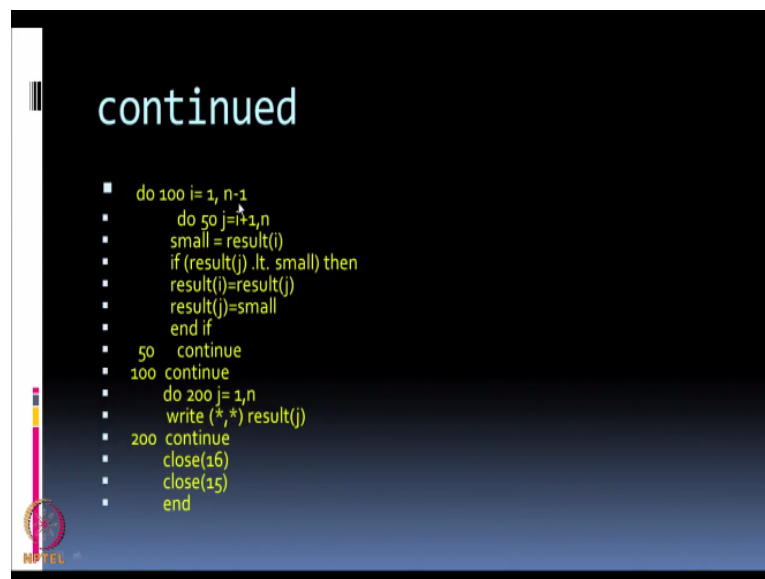
So when the program executes this particular line, it expects you to give that input n, so read star, star n. That means this n which is the number of data in your file will be read from the screen. Once this number is read, I want to write all the values of that array a*the array result. So what this do loop does for me, do i going from 1 to n, n is already read from the screen. So do read 15, star ai, so from file 15 which is input it will read the value of ai.

Then, the result i is set=ai okay, there is a small error here. There should be no space between this a and i because if there is a space the program will not like it so since there is no space

between a and this bracket this also should be a space i bracket complete a bracket i bracket complete. There should be no space, so make sure when you execute, do not leave any spaces when it is not indicated in my original variable end do.

So this do loop reads all the value as of ai from your file number 15 and in the result variable it puts all the values of a into that result variable. I am going to work with the result variable and keep the a intact because I do not want to lose the original order. So I am keeping the information of the original array intact.

(Refer Slide Time: 04:37)



So now let us see what is the, the next thing will be I want to arrange them. So now I am going to use two do loops for this. There is now outer do loop do 100 i going from 1, n-1. If n is the number of data points, my i will take only values up to n-1 okay. So and do 100, what is the meaning of this do 100? Repeat all the statements up to this 100 continue. How many times? Start with 1, 2, 3, up to n-1.

So my second do loop starts from j going from i+1 to n. The second do loop is up to line number 50, do 50 means repeat everything up to line 50. How many times? Start with j=i+1 and all the way up to n. So if i is 1, this second 1 will start with i+1 which is 2. So why is it I am doing this because if I have read the first number, I need to compare that number with the second number.

So I cannot compare the number with itself, so there is a difference in the do loop. The first loop goes all the way up to n-1, the second one starts with 1 index>i and ends with n. So this

is my two do loops. So now what I am going to do, my first variable is I, so this small I have written result i is my first value of the result because $i=1$, that value I keep in this array called small then I compare result j with small.

Because small is already result i, so if result j what is j first time? $j=2$. So if result j which is second one if it is $< \text{small}$ then I want to exchange these two because I want to arrange them in an ascending order. The smaller ones will be written in an earlier value of the array and the larger ones will automatically go into larger values. So $\text{small} = \text{result } i$. If result j is $< \text{small}$ then result i becomes result j because result j was a smaller number.

And now result j is small because small is original value of result i. Remember, I already discussed in the last class, if you do not have this small as a second temporary variable if I say $\text{result } i = \text{result } j$ and $\text{result } j = \text{result } i$ then both result i and result j will have the same value. So rather than exchanging I would have made both of them equal so I need this temporary variable small so that I can distinguish between the initial values of result i and the final value, so this if statement ends here.

If the result i is already $< \text{result } j$, no exchange occurs but if the result i is $> \text{result } j$ then the greater value is put into a larger array variable. So it ends this do loop. Both the 50 do loop ends here, so at the end of the first do loop what will happen is that result i will have the smallest value among $j=2$ to n. So the first value will be the smallest value. So in the next loop now $i=2$, so when $i=2$ j will go from 3 to n.

So when $i=2$ small will be result of 2 then it will exchange 2 and 3, 2 and 4, 2 and 5 until the result 2 will be the smallest value between 2 and n. So the first loop gives the smallest value in result 1, the second loop will give the next smallest value in result 2, the third loop will give the third smallest value in result 3 and so on and so on until all the smaller values are arranged in an ascending order from 1 to $n-1$.

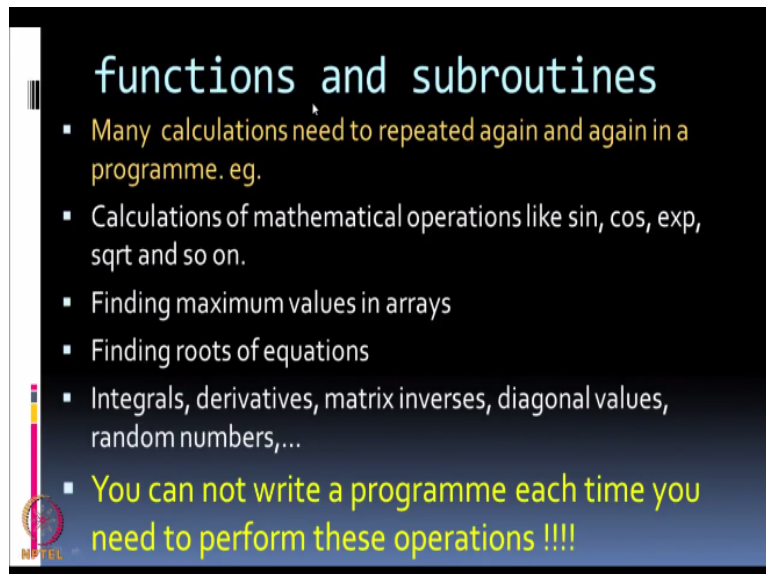
And if I do up to 1 to $n-1$ automatically the last value would be the largest value, so the last loop will finish without doing anything because already the last value is larger. So in the last loop, I have this $n-1$ if n is 10 let us say it will be 9th, so the 9th value will be compared with 10, but 10 is already the larger value because the 9th one is the smallest among the first 10, so the loop will end.

And what I will do at the end of these do loops is write my result on the screen now. In this case, do 200 j going from 1 to n write star, star result j. Instead of writing on the screen, suppose I want to write it in unit 16, I will say write 16, star result j. This way I will write it in file which is 16. So instead of just writing in arbitrary format, remember the second star represents the format.

Instead of star suppose I have some format, suppose I want to write all those numbers in a given format then I will say write 16, say 200 result j and that 200 will be the format number. That format could be say E12.4 that we considered and if they were all integers I will use an integer format. So you can write in whichever format you want and we have already discussed the format statement.

So this is how that program will execute and we will also practice this in our practical session. We will practice this in our practical session.

(Refer Slide Time: 10:09)



The slide has a dark blue background with a light blue gradient at the bottom. The title 'functions and subroutines' is in a light blue, sans-serif font. Below the title is a list of six bullet points, each preceded by a small square icon. The first bullet point is in orange, and the last one is in yellow. A small logo is visible in the bottom left corner of the slide.

functions and subroutines

- Many calculations need to be repeated again and again in a programme. eg.
- Calculations of mathematical operations like sin, cos, exp, sqrt and so on.
- Finding maximum values in arrays
- Finding roots of equations
- Integrals, derivatives, matrix inverses, diagonal values, random numbers,...
- You can not write a programme each time you need to perform these operations !!!!

So at this point what I want to do, I want to discuss a new topic which are categorized as functions and subroutines. Now what are these functions and subroutines? So you will know that in your calculator in your scientific calculator there are functions called cos, sin, log, exponential. These functions are already built into your calculator. So suppose your angle is 30 and you want cos of this 30, all you do is to press that cos and the cos value will come on the screen.

So the same thing you can do in programs as well. So you may want your own function, you may want $\cos^2 x$ somewhere, you may want $\sin^2 x$, you may want some powers of functions, so you will be able to write your own functions based on your own requirements. So as I have said here many calculations need to be repeated again and again in a program. See what are the examples, sin, cos, exponential and so on, square root.

These are already there in your fortran compiler but suppose you want to calculate some Legendre polynomials; these are polynomials which will come when you study orbitals. So the calculator does not have these functions nor your program has that function, so you will have to write a subprogram which calculates that function for you. Another thing you may want to do you may want to find the maximum value in a given array.

You may want to find roots of some equations, you may want to calculate integrals, derivatives, matrix inverses, diagonal values of matrices, all these you need again and again and every time you need a matrix inverse, you cannot write a separate program. So if you write a program which can be used again and again, so that is what a subroutine is. Subroutine is a part of the program which is in a sense separate in itself but it will link to some main program.

So just as in your calculator the functions cos, sin, log they are all part of the calculator. Whenever you want that cos you just press cos you will get it. So in the same way your program will have the main program which does all the operations you want and functions and subroutines which will give you whatever you want as a part of that program separate part of the program okay.

So now again what is the reason. I have mentioned here; you cannot write a program each time you need to perform these operations okay. So therefore now we will see what is the structure of this function and subroutine?

(Refer Slide Time: 12:37)

```

▪ program for sinnew
▪ write (*,*) 'input the value of variable x in degrees'
▪ read (*,*) x
▪ x=deg*pi/180.0
▪ y=sinnew(x)
▪ z=sin(x)
▪ write (*,*) 'x, sin(x), sinew(x) =', x, z, y
▪ end
▪ FUNCTION sinnew(x)
▪ sum=x
▪ term=x
c copy all the lines of the sine (exponential) programme that we discussed earlier
▪ sinnew = sum
▪ return
▪ end

```

So now I want to illustrate this using this function called sin new okay. So sin new is a new function, not your calculator function okay. So the program starts I will say write star, star input the value of the variable x in degrees. So as soon as you execute this program, the statement input the values of variable x in degrees will appear on the screen. Then, this read star, star x will read the value of x in degrees.

Now many of these programs use the value in radians. So first thing you do calculate $x = \text{degree} * \pi / 180$. What this will do? It will calculate the value of x in radians now. So once this is done, the next line is y is=sin new x. Now this is a function, remember I mentioned that there are functions and subroutines. The moment you say y is=sin new x, so this program will look for this sin new okay.

So what is this sin new? So I will end this program by this end statement. Then, I am writing this function statement. So whenever this is a new variable which the program does not know, it will understand that this is some function of x, so it will look for this function. So how does this function work? Function sin new of x, this is a function statements. So it is a part of the program, separate part of the program.

So function sin new x, so what will this function do? So when you say function sin new, it knows the value of x, so it will calculate this sin function through a Taylor series. Remember, in our earlier program just see this line, you had written a program for an exponential function e to the x. How did we calculate e to the x? e to the x was $1 + x + \frac{x^2}{2 \text{ factorial}} + \frac{x^3}{3 \text{ factorial}}$ and so on.

So what is the sin function now? $\sin x$ is $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$, so sin function is like an exponential function except that the sine of alternate terms are + and -. So similarly if you already have that program just as we have an exponential program, we can write a sine program. All those lines will be written here, so once you call this function, it will calculate all the values.

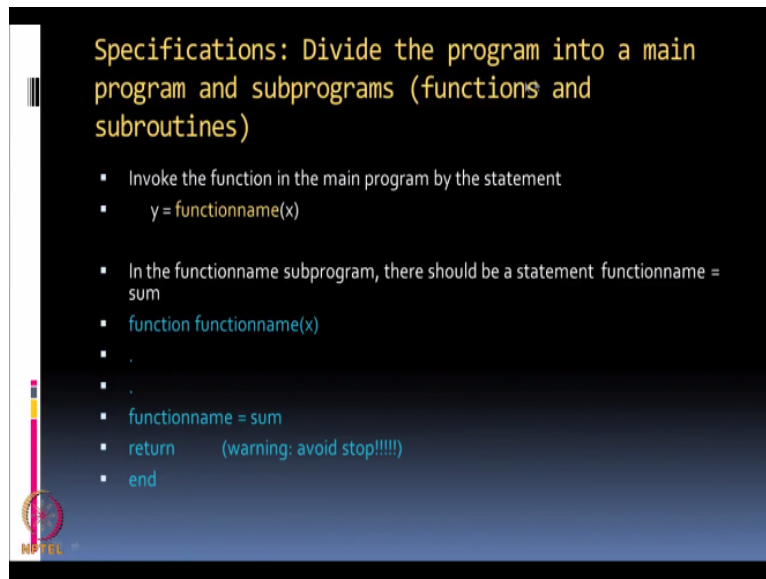
And the sum of all those will be your sine function that is $x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$ and so on that is your sum, calculate that sum, equate $\sin_{\text{new}} =$ to that sum return and end. So what does the program do? $y = \sin x$ goes to this function, calculates all these values, equates \sin_{new} to be the value of the function and returns that to y value. So now what is y? y is nothing but it is a sin function calculated.

Why I have called it new? Because you have written your own program to calculate the sin function but we also know that fortran also has a sine function built into it just as your calculator has a sine function built into it, so y is a sine function calculated by you using your functions of program and z is the value of $\sin x$ calculated from the fortran compiler. So now what you will do as an output?

Write star, star x $\sin x$ and $\sin_{\text{new}} x$, on the screen you want to write the original value of x, the value calculated from the fortran compiler and y which is your own function. So it also allows you to check whether whatever program you have written is as good as the built in program in the fortran compiler or your program may be better or your program may be worse as well so it allows you to compare.

So in this case, sine was already known, so this is a way how you will calculate a function using a subprogram. So let us now summarize again what this function is okay.

(Refer Slide Time: 16:52)



Then, so what have we done here, we have divided the program into a main program and subprograms. Now what is that main program? Now you see here, in the main program, program for sin new everything up to the end is a main program. Whatever comes after the end, these are functions or subroutines. Right now, we have considered one function, now suppose you have two more functions.

So if you have two more functions, so after z let us say a=that some other function, b is=some other function, the other functions will come after the first function, you can have any number of functions in a program. So just that you have functions, you have also subroutines. We will consider that shortly. Now let us first summarize what is the structure of this function.

You have already divided the program into a main program and subprograms. Subprograms are functions and subroutines. We have already discussed the function. So how does a function work? Invoke the function in the main program by the statement `y=function name x`. Remember, we wrote `y is=sin new of x` instead of `sin new` it could be function name. This function name is any name, any function.

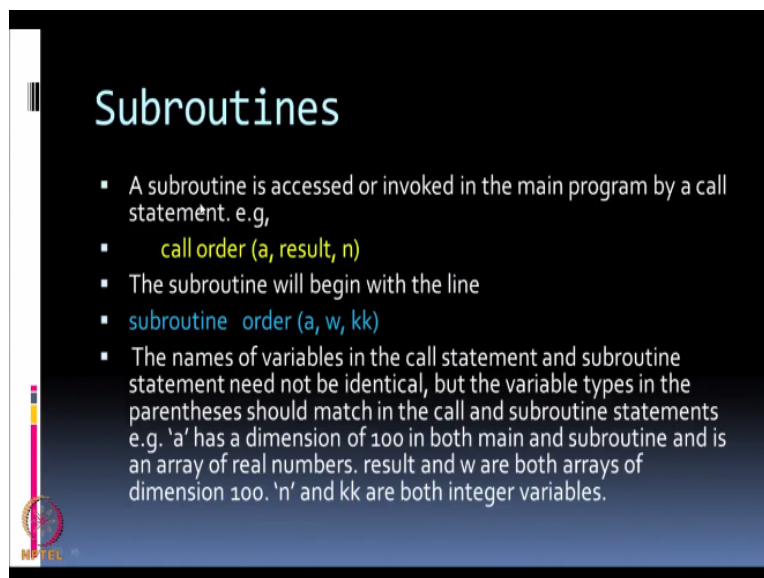
So this is how you will invoke the function in your main program and in the function name subprogram, the starting line will be `function=function name x`. In our earlier case, it was `sin new x`, so in the function name, it will use the value of `x` to calculate everything and final statement in the function would be `function name=sum`, that was the actual value and return and end. So whenever you invoke, what do you mean by invoke?

You are basically accessing or calling that value. So when you use this statement, it will go to this function name, do all the things and return with the actual value of the function, it will return. So remember I have written here return and not stop or end because if you say stop it might stop the entire program. So remember that in functions or other subprograms, you will not have a stop.

Stop means everything will stop okay. So return and end, this is the structure. So what is the main difference between a function and main program? In the main program, you will have stop and end. In the function, you have return and end and in the function, function name is associated with the value. The function name is associated with the value, so that is the nature of the function.

And the function will always return to you one value but many times suppose you want to invert a matrix, the result of your operation is an entire inverse matrix. The inverse matrix will be end by end, so you cannot use a function for large number of data. So therefore what you use is called a subroutine okay.

(Refer Slide Time: 19:41)



Subroutines

- A subroutine is accessed or invoked in the main program by a call statement. e.g,
`call order (a, result, n)`
- The subroutine will begin with the line
`subroutine order (a, w, kk)`
- The names of variables in the call statement and subroutine statement need not be identical, but the variable types in the parentheses should match in the call and subroutine statements e.g. 'a' has a dimension of 100 in both main and subroutine and is an array of real numbers. result and w are both arrays of dimension 100. 'n' and kk are both integer variables.

So what is a subroutine now? Subroutine is accessed or invoked in the main program by a call statement. In this case, I said call order a result n. So this particular subroutine whenever you have a call statement it is a subroutine not a function. In a function, it was y=function name into bracket x. So in a subroutine the difference is subroutine is always called it is called by a statement call.

So call order a result n. So how will the subroutine begin now? In the case of a function, it began as function, function name of x. The subroutine will begin as the subroutine; subroutine order a w kk. So let us again summarize what I have done. Whenever you want to invoke a subroutine, subroutine is like a subprogram which comes at the end of the main program. It will have a large number of lines okay.

So when I say call order that program will go to the subroutine order, do all the calculation and return to the main program okay. So now we will discuss remember we have already discussed a program to arrange numbers in an ascending order. So what I will do in this particular lecture is to use this arranging numbers in an ascending order is a subroutine not as a main program.

So before I go to that let us make certain comments about subroutines and functions. So in the function and subroutine, there will be again several variables. So the names of the variables in the call statement and subroutine, names and variables in the program can be very, very different okay but whatever is passed on to the subroutine that is now look at the statement, call order a result n.

So the subroutine should understand what is a, what is result, what is n. These 3 are the things which are linking the subroutine to the main program. So these should be having the same meaning, but everything else could be different okay, so variable types in the parentheses should match okay. If a has a dimension 100, suppose this a has a dimension 100 in the main program, then in the subroutine also a should have a dimension 100 okay.

If result has a dimension 10, subroutine also should have the same dimension and whatever is the type of n suppose n is an integer in the main program, in the subroutine also it should be an integer. So these should be matched, the rest there is complete freedom.

(Refer Slide Time: 22:26)

More Details on subroutines

- When the calculations are done in the subroutine, it closes with a return and end statement and not just an end statement or a stop and end statements, i.e., return statement is a must in a subroutine. (newer compilers may not require return!!)

So now let us look some more details now okay. So when the calculations are done in the subroutine, it again closes with a return and end statement not like a stop. Remember, in the function also you had a return and end, subroutine also has a return and end that is how you will do it. So there should be no stop and end statements but newer compilers may not require even a return.

So you may have newer compilers where you say you write the subroutine and just return okay. So in the practical session, we will see this.

(Refer Slide Time: 23:03)

Details on subroutines

- The line numbers and variables in the subroutine are independent of the line numbers and variables in the main program
- Only variables passed through the parentheses of the subroutine statement are common between the main program and the subroutine (or variables passed through common statement).

Now I will look at some detail now okay. Some more details, so one of the question is in the main program we have several line numbers and several variables. For example, line number 10 will be in the main program, variable a b c d will be in the main program but in the

subroutine these variables can in principles be very, very different. You can have the same variables in the main program as well as subprogram.

But unless they are passed through the parentheses they will have a different meaning. So what is this meaning now, only variables passed through the parenthesis of the subroutine statement are common between the main program and the subroutine okay. So there are two ways of doing it, one is through the statement of the subroutine. So look at this, when I say call order a result n, this a result and n were common between subroutine and this main program.

These 3 are common; the rest can be very different, so one way to pass information from subroutine to the main program is through all the variables in the parentheses of the call statement, one through the parentheses but suppose you have many, many variables you want to pass through it becomes too much write 20 or 30 things in this bracket, so there are other ways of doing which is called a common statement.

So that is what is written here. So you can convey information between main program and subroutine through the things that are written in the parentheses one option. The other option would be through a common statement. We will also discuss a common statement okay. That is my next discussion. So I again summarize either you have a common statement or you pass the variables through the parentheses. Two options to convey between the main program and the subroutine okay.

(Refer Slide Time: 24:55)

Main program for arranging numbers in an ascending order

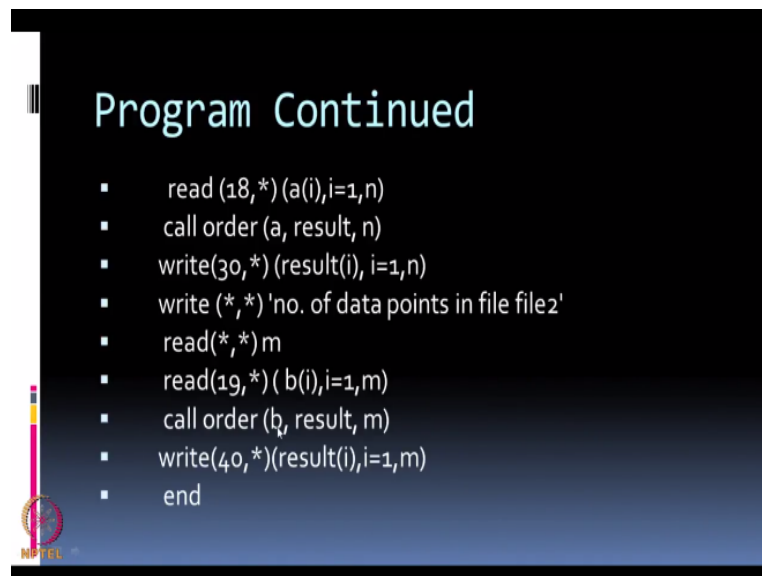
- program main
- c this program reads numbers from two files and arranges them in an
- c ascending order
- dimension a(100), b(100), c(100)
- dimension result(100)
- open(unit=18, file= 'input1')
- open(unit=19, file= 'file2')
- open(unit=30, file='result1')
- open(unit=40, file='result2')
- write(*,*) ' no. of data points in file input1='
- read (*,*) n

So now let us now arrange numbers in an ascending order using a subroutine okay. I do not want the main program to do it; I want a subroutine to do it. So what is the difference? So the difference is my main program will be so I again I have a b and c, these are 3 arrays. Suppose I want to arrange both a and b in an ascending order or suppose I want to arrange both a b and c in an ascending order.

I do not want to write the program 3 times that I wrote earlier. So I want to write a subroutine which arranges the numbers in an ascending order and call that subroutine 3 times. So that is the main advantage, write the program once and call it as many times as I want in the main program. So again so there are 4 objects here a b c are dimension variable, result is 100, so what I will do?

Open 18 19 30 these are all units, 18 will open unit input 1, 19 is file 2, 30 is result 1, 40 is result 2 and so on, write number of data points in file input 1. So what will this read now? Read n from the screen, n will tell me the number of data points from in the screen.

(Refer Slide Time: 26:16)



So what it will do? Now read 18, star ai i going from 1 to n, suppose n is 20 from that file 18 it will read all the 20 data points. Now I will say call order a result n, so a was my array, result is a final array into which it is in order and n is the number of data points. Call order, so it will arrange them in an ascending order, write 30, star result i, i going from 1 to n. It will write the result. In this case, it will write on the screen.

Now I want to arrange data in file 2 in an ascending order, so what do I need to do? How many data points are there in file 2? Read that m and read all the data from that file 19 now because this is a second file again call order. So now first time it arranged all the things in an ascending order of variable a. Next time, it will arrange b in an ascending order, b was read from a different file, it called a subroutine again.

When you call the subroutine the second time, I had b in place of a, result is the same, m is in place of n. So it is a different number of data points, different array and I want to arrange them in the ascending order and write result 1, n. So it will arrange the file b and the program will be over. So the next thing, the next slide will tell me how the subroutine is written. So I will what I will do I will conclude this lecture here.

Next time, again I will begin what is the concept of a function and a subroutine. This is an important concept, it takes a little practice to understand, then how I write the subroutine, how I communicate between the main program and the subroutine and then we will repeat that and show you how the subroutine looks like. In this case, the subroutine is to arrange numbers in an ascending order.

So we will discuss different types of subroutines and functions in the next class. So I will conclude here. Thank you.