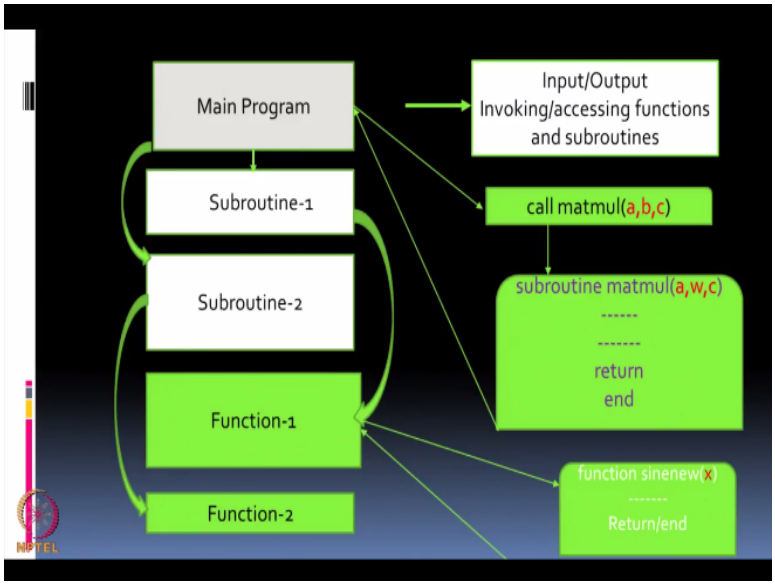
Computational Chemistry & Classical Molecular Dynamics
Prof. B. L. Tembe
Department of Chemistry
Indian Institute of Technology – Bombay

Lecture - 10
Programming Techniques 7. Functions and Subroutines, and the Common Statement

Hello and welcome to today's lecture. In the last lecture, we introduced you to the format statement as well as functions and subroutines. So we will review the functions and subroutines with some examples today and also introduce a very useful statement called the comment statement which allows you to connect the main program and the subroutines. So after completing the subroutines and functions, then we will start numerical methods.

And numerical methods are the main methods which are useful in doing several computations in chemistry. So let me begin with summary of the functions and subroutines.

(Refer Slide Time: 00:55)



So we saw that there is a main program. You first write all the statements in the main program and the main program will deal with input, output and invoking or accessing functions and subroutines. So the main program will have all the input and output and it will finally end with a stop and end statement and the main program can call the subroutines. So I have given here first you complete the main program.

Then, you have subroutine 1, subroutine 2, function 1, function 2, so there can be several subroutines and functions in a program. So now the main program will call one of the

subroutines, so take this example. Here the main program will call subroutine called matmul a, b, c. So this statement called matmul a, b, c is in the main program and the moment this statement comes in the main program, the moment this line comes the program control will go to the subroutine.

So the call statement was call matmul a, b, c. So the moment this statement is executed it will go to this subroutine. So the subroutine begins with the statement subroutine matmul a, w, c. Remember here, I have a, b, c but in the subroutine I have called it a, w, c. The actual names will not matter. Instead of a, b, c in the main program, I can call them a, w, c, so here the a in the matmul is the same as a in the main program.

So w in the subroutine will be now b of that main program and c is the same as c. So what it means is that the arrangement of objects should be in the same order in the main program and the subroutine but the names can be different. In the same way, in the subroutine I can have other variables and those variables will have no connection with the main program, only the variables that are in this bracket.

In the bracket of the definition of the subroutine, these are common between the main program and the subroutine. So once this program is called, it will execute all the statements in the program and it will return and go to the main program. So now similar to calling this subroutine, I can also call a function. So how is a function called? The function is called by a statement.

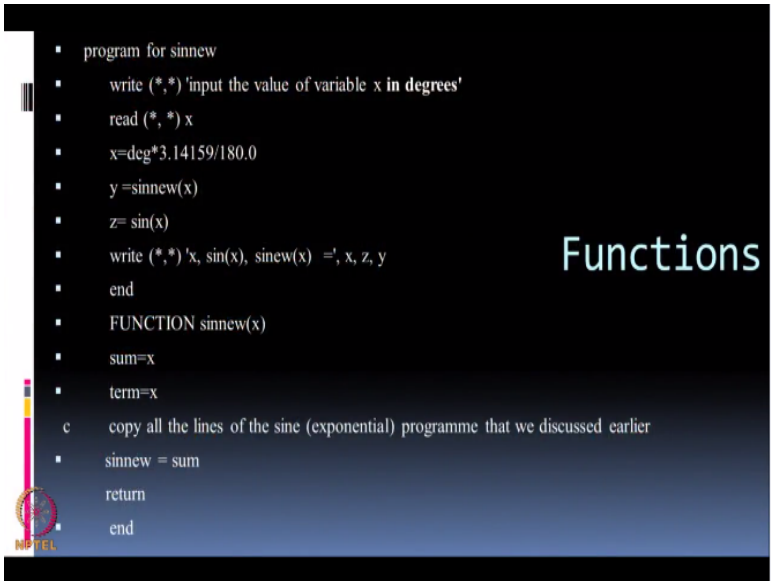
See look at the lower part of this slide, on the lower right there is called a function sin new w, x. So in the main program, I have to call it as y=sin new x. So that calling statement for the function is any variable a b c d is equated to the function sin new into bracket x. So it is accessed by that y=sin new x. Once that happens, you go to this particular function. Then, calculate the value of the function and return to the main program.

So to summarize this a subroutine is accessed by a call statement and a subroutine is accessed just by a variable being equated to the name of the function. In this case, the name is sin new x. When the function is called, it will go to that function subprogram and return the value. Now I have given here several subroutines and functions, as I said there could be any number of subroutines, any number of functions.

And any subroutine can be accessed by any subroutine, any function can be accessed on any function but from the function or subroutine you cannot access the main program. Main program is the controlling part. So from different subroutines, you can go to different subroutines that is allowed but eventually if you call subroutine 2 from subroutine 1, it will complete the job of subroutine and the control will go back to subroutine 1.

So this is how the structure of the program is, so I will take now one or two examples.

(Refer Slide Time: 05:19)

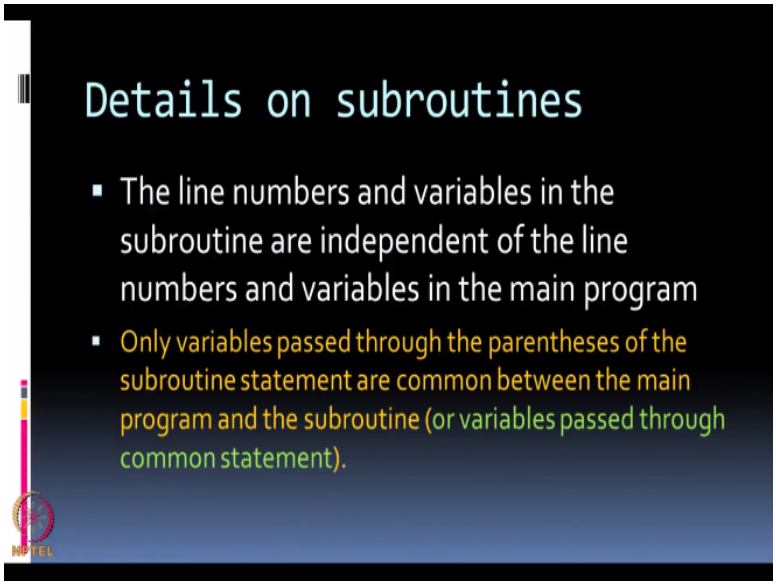


So in this example, this is a reputation of the last time. This is a function sin new. So sin new is similar to the sin function in fortran except that in the sin new I calculate the function using the Taylor expansion. So from the beginning how I will execute this program. First you will input the value of x in degrees, then convert that x into radians, then you call this function y=sin new x. This is the statement which calls that function.

The moment you call that function it goes to this function sin new x, calculates the value of sin x using this particular algorithm. Then, the control goes back here. Then, again what we are going to do, we are going to calculate that sin x using a fortran function. This sin x, see z=sin x is a fortran function, y=sin new x is a function you have written, so when you do both these tasks you will write all these values that is the value of x, value of sin x and value of sin new x on the screen.

And it allows you to compare your own function with a standard function given in the fortran library. So this is how a function is called and remember that when you call a function you return a single value whereas when you call a subroutine it can give you any number of values. So often subroutine has many more advantages than a function.

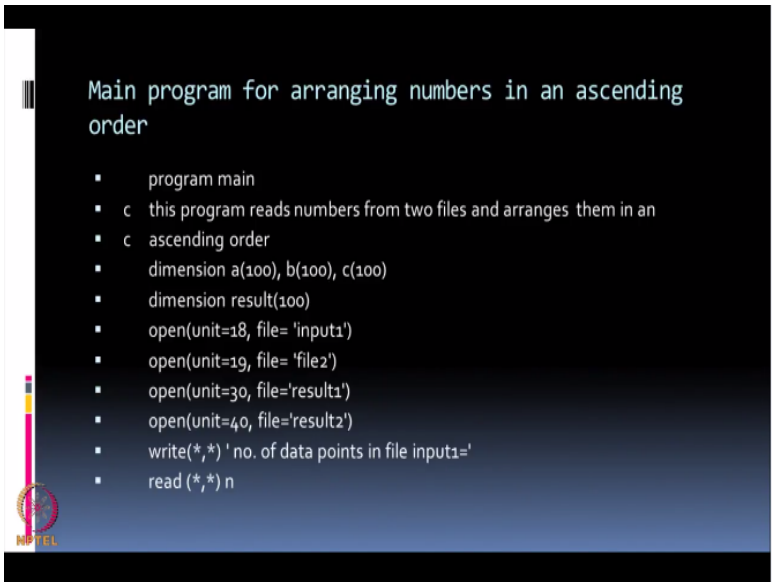
(Refer Slide Time: 06:46)



So next example would be, let us see what are the details now. Some details we will comment on subroutines. The line numbers and variables in the subroutine are independent of the line numbers and variables in the main program and only the variables which are passed through the parentheses of the subroutine statement. Remember call matmul into bracket a, b, c, only those a b c are common between the main program and subroutine.

So one way to pass information from the main program to the subroutine is through that parentheses in the subroutine statement, call statement or you can also pass variables to a common statement. We shall discuss very briefly what is the meaning of the common statement.

(Refer Slide Time: 07:25)



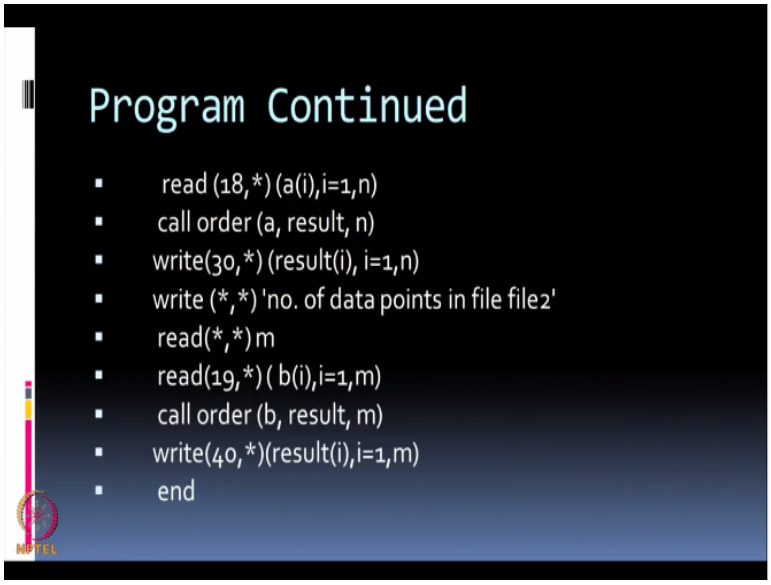
Before I do that, the program that we discussed last time it was a program to arrange numbers in an ascending order. So now what I want to do with this present program, I have several files and several numbers stored in that files and I want to arrange them in an ascending order. So the advantage of a subroutine is that I do not have to write the program twice, I have already written entire structure into that subroutine.

And I have called that subroutine as many times as I want to perform the task that I am interested. So in this case, I want to arrange numbers in an ascending order. Now look at the structure of this program. This is the main program; you do not have to have a statement program in but it is a good idea to have so that you know it is a main program. So what the program does, it has a dimensional statement.

So there are 3 arrays now a b and c, each of dimension 100 and I also have another statement dimension result 100. So that is the result of my operation of converting an array into an ascending order. Then, I will read from file 18, this is the input. I will associate this unit 18 to the file input 1, associate unit 19 or number 19 to file 2, number 30 to result 1 and number 40 to result 2.

So I have associated those 4 numbers to these files. Now the next thing the program does is it will ask you to input n, how many data points are there to be read from file input 1?

(Refer Slide Time: 09:03)



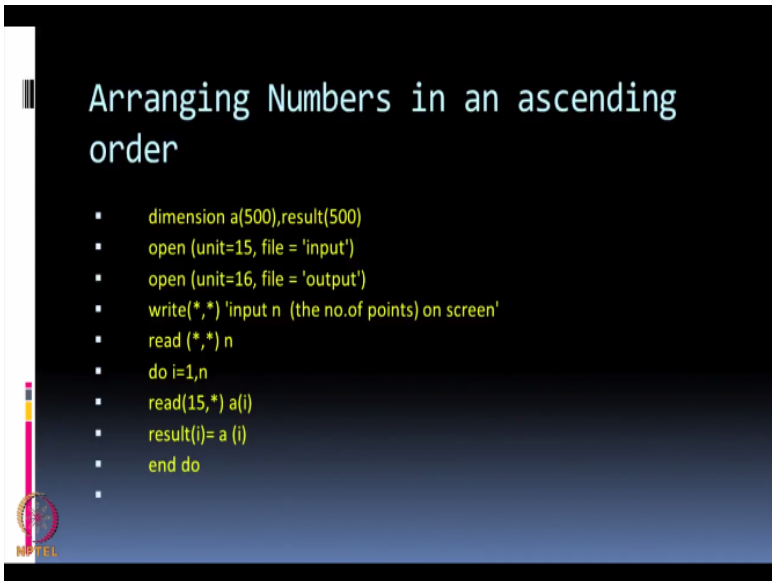
So once you read that, now it will now read all those numbers ai from 1 to n that is if n is 10 it will read 10 numbers from this particular file input 1 and once you have read those 10 numbers, now I want to call that subroutine. Now see this call order a, result, n, a is your initial array, result is the result of converting those numbers in an ascending order and n is the number of data points that you want to arrange in an ascending order okay.

So once you call the subroutine, it will go to the subroutine, arrange all the numbers in an ascending order and write them in the result. Now you see write 30, star result i i going from 1 to n, whatever array was there in a now that is arranged in an ascending order and that is stored in a variable called result and that is written now in file 30. So that will write in file 30. Now after this operation is over, you want to arrange data points in file 2.

So what does the program do now? It will write on the screen the number of data points you want to read from file 2. So it reads this m, m is the integer. Then, if m is say 10 from file 19, it will read bi i going from 1 to n. Now it is going to read variable b from file 19. So it will read as many variables that are there specified by this particular integer m and once that information is read, now I will call the same subroutine again.

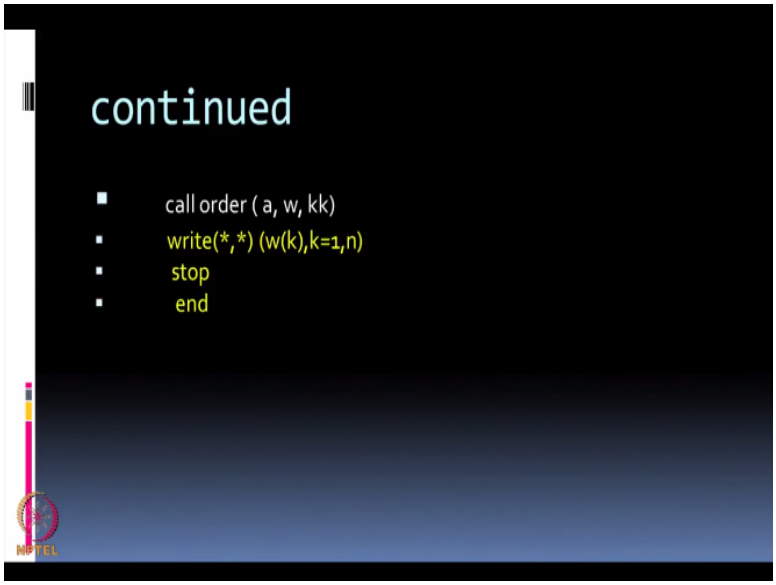
Call order b, result, m. Now what is it doing this time? This time it is not the a array which is being arranged, it is an array b that is being arranged and the result is again arranged in the result and m is the number of data points you have it. So finally it will write result in this case in file 40 that may be output 2. So what has this main program done, it has read 2 different files, arranged those numbers in an ascending order and written them in two different files.

(Refer Slide Time: 11:32)



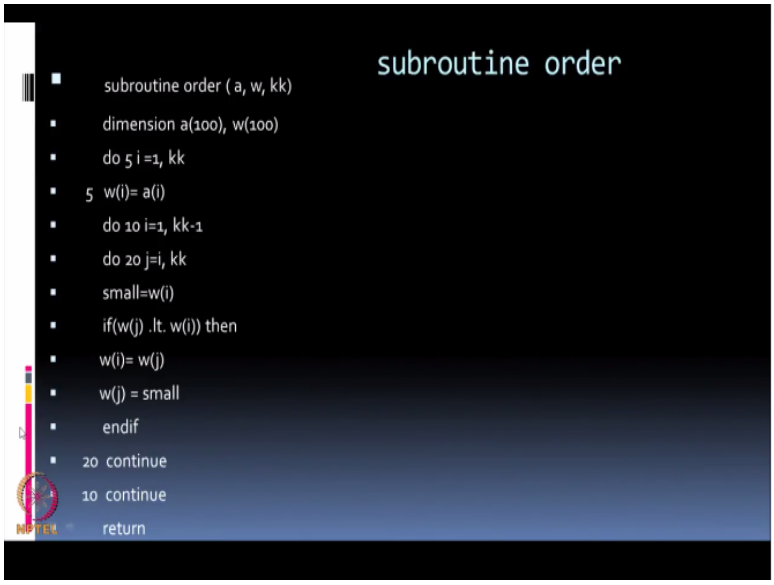
So this is the main program and the subroutine really this is what is the subroutine. So in that subroutine also you will have a dimension statement okay.

(Refer Slide Time: 11:40)



Let us see, yeah how do you call this subroutine? Call order a, w, kk and write the output.

(Refer Slide Time: 11:47)



So now this is how that subroutine will look like, see this is subroutine order. Yeah, this is subroutine order, so once the main program has a statement call order a, b, m it will call this it will go to the subroutine statement. Now the dimensions of a and w in the subroutine should match exactly the dimensions of a and w in the main program. So if they do not match, then there will be a huge problem.

So make sure that whatever are the dimensions in the main program exactly the same dimensions are there in the subroutine. So once you go to this subroutine, so now this is subroutine order, so the main program calls it by a call order a, w, kk and in the subroutine the dimensions of a and w should be identical with the dimensions of a and w in the main program.

The other point is in the subroutine, I have called this variable a, it is not necessary to call it a, I can call it any other variable, it could be b c d but this d in the subroutine will be whatever is the first variable in the call statement, w here will be whatever is the second variable in the call statement and kk will be whatever is the variable in the third statement okay. So then once I know the dimensions of a and b, a information is already containing the array a.

So what I will do first, I will write all the variables in a to variable w because w is my new variable now and in w I will arrange them in ascending order. What was our strategy to arrange numbers in an ascending order? There will be two do loops. First do loop goes from 1

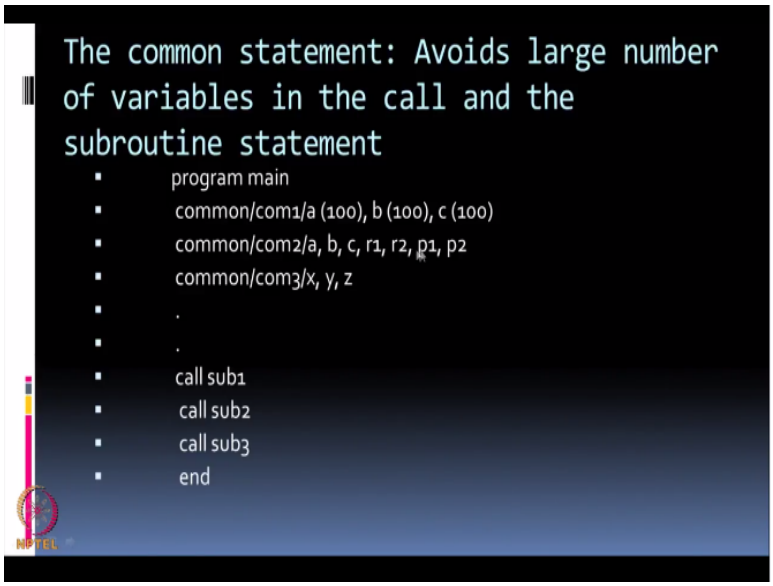
to kk-1, second do loop should actually go from i+1 to kk. So these are the two do loops, instead of i here you should have i+1.

Then, you exchange the smaller and the larger and that exchanging what you have to do you have to have a temporary variable small which allows you to store the variable wi. Then, if wj is<wi then you exchange if not you do not exchange at all. So this way you will arrange in an ascending order, then both do loops are ended here and you will return. So this is how at the end of this subroutine all the values of w will be arranged in an ascending order.

The other point I would like to make, so these arrays are of length 100 each. So when you want to arrange numbers up to 100, this program will work. Suppose you have 105 as the data, then that will not work because this allows you only to arrange 100 numbers in an ascending order. So if your data exceeds these dimensions, you will have to rewrite the program giving the size here 200, 300 and so on.

So usually it is a good idea to have sizes which are large enough so that it will satisfy your requirements of whatever task you want to perform. So this completes my discussion of how the main program calls the subroutine with the information passed through the objects in the parentheses.

(Refer Slide Time: 15:12)



Now there is another way to do what is called a common statement. Now the advantage of a common statement is that if you have a very large number of variables to be transferred from main program to subroutine, suppose you have 40 objects, so you cannot possibly write all

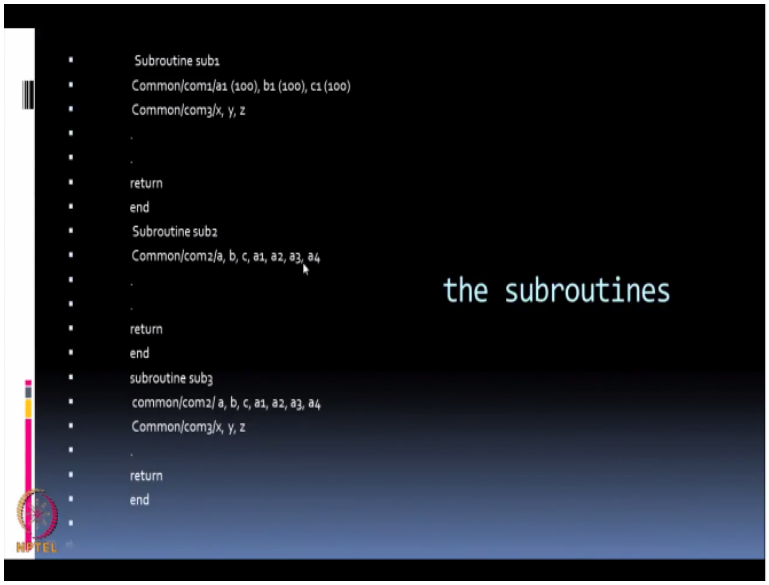
the 40 objects in a particular one particular bracket. So what you do, in the main program you have several common statements.

So look at this, this is a common, I call it com1 a100 b100 c100. So what this does? It creates a block of data that block of data will have 3 variables a b and c each of size 100. The second block of data com2, this set has a b c r1 r2 p1 p2. So the second block has 7 items. The third block has these 3 items. Now it is possible that some subroutines will require a b c, some other subroutine may require x y z, some other subroutine may require a b c r1 r2 p1 p2.

So depending on the requirement that subroutine will have those common statements. So once this is a part of the main program, then I will call subroutine 1, sub2, sub3 and end. So what is this main program doing now? It has assigned the data into 3 different common blocks. This dot, dot statements mean that you are going to read information about all these variables a b c r1 r2 and so on.

You read all the information call sub1 call sub2 call sub3 and end. Now let us see what all these sub1 sub2 and sub3 are doing.

(Refer Slide Time: 16:46)



See in this subroutine sub1, I have taken only the common com1 a1 b1 and c1. I have only taken com1 and com3, com2 has not been taken in sub1 because in this subroutine I do not require information about com2. So this subroutine will do whatever task it is required, it will take up the information between com1 and com2 from the main program, finish all the things and return to the main program.

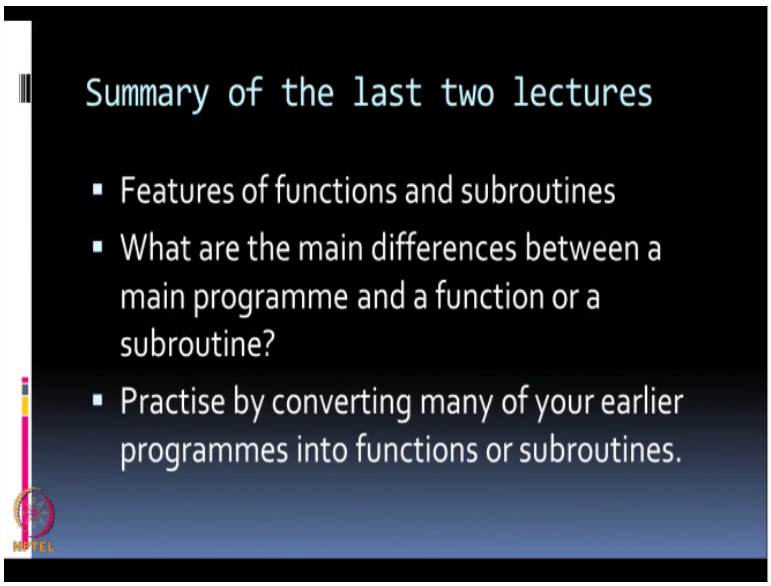
So when I call subroutine 2 from the main program, it only requires now a b c a1 a2 a3 a4. This does not require com1 or com3. So this subroutine will do all the calculations that are using only this a b c a1 a2 a3 a4. So the names a1 a2 a3 a4 need not be identical to the names in the main program but whatever is the 4th item in the com2, it will be taken by a1, 5th will be taken by a2 and so on.

The order in this will be the similar to the order in the main program. The names can be different. It is like you will have one roll number in your B.Sc., you will have another roll number in M.Sc. but you are the same person okay. So when you transfer from one place to another, information transfer is done through these variables. So your sub2 has used com2 and coming to sub3 now sub3 has used com1 and com3 similar to sub1.

So sub3 uses com2 and 3 not com1, so this uses com2 and com3, the sub2 uses only com2, sub1 uses com1 and com3. So this way using these common blocks it has the advantage that you can transfer whatever information you require to a specific subroutine and complete the task and this way you can transfer as much data as you want between the main program and subroutine and it also makes it look neater that is you just say subroutine sub2.

So you do not have to give anything in bracket, instead of giving anything in brackets, it is now all being transferred through the common statements.

(Refer Slide Time: 18:57)

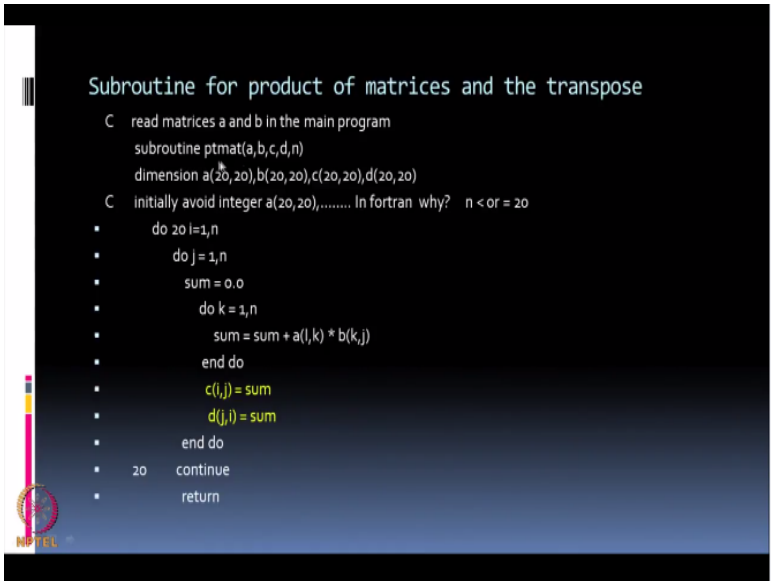


So now let us see, so what we have done so far, what is our summary now. We have discussed features of functions and subroutines and what are the main differences between a main program and a function subroutine. So the main program has lot of input and output. What does a function do? It returns exactly one value and the name of the function is important because a variable the name of the function itself is a variable.

When I say y=sin new x that sin new itself is a value of the sin function. So that will be written to the main program whereas in the subroutine any number of data can be passed on the main program to the subroutine. They are either transferred through the bracket statements in the subroutine statement or they are transferred through the common statements.

So now that I have given some examples. You can also you have several program you have written earlier, try to write subroutines using the same program that you have done. So I will take I think one more example here.

(Refer Slide Time: 19:57)



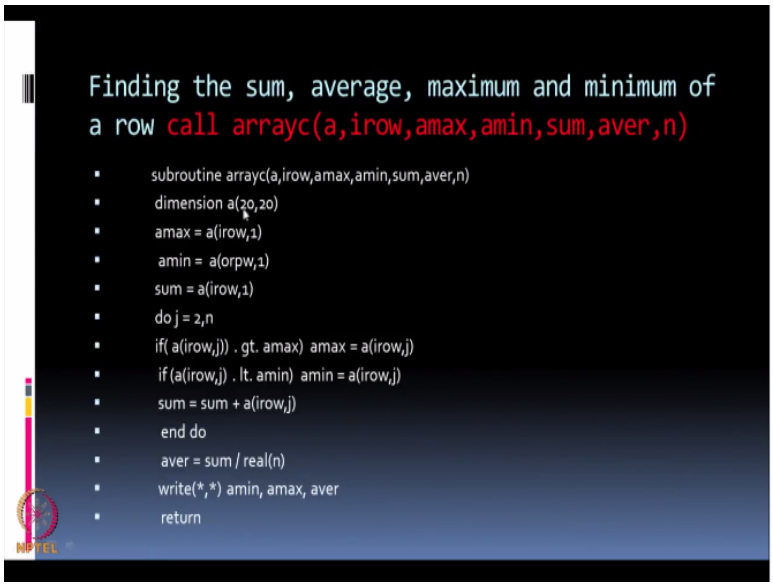
So now this is an example of a subroutine to calculate the product of matrices and transpose of the matrix. So this subroutine is going to do two functions now, calculate the product of matrices as well as the transpose. So I have called it subroutine ptmat into bracket a b c d n. So why I have called it ptmat? p for the product, t for transpose, mat for matrices. So one of a useful suggestion is that you give the names of your subroutines such that it tells you what the function is.

So this is a product and transpose of a matrix a b c d, so this is the subroutine. So once this subroutine is called a b c d are all dimension statements, so see here a20, 20 b20, 20 c20, 20, d20, 20. Both these all these 4 variables a b c d are matrices and n is just sum integer. So once you call this particular subroutine from a main program, so in the main program your array need not always be 20/20, you may have a smaller array than 20/20 but this program will not work if your array size is more than 20.

Suppose your matrices are smaller in size so that I have called it some integer here. So if your main matrices that you arranged are smaller size, the product also will be a smaller size. So what do these do loops do now, these do loops multiply matrices a and b to give you matrix c. You see here, this is a standard program to multiply matrices, here I have multiplied matrix a and b to get a product c, ci, j is your product.

Now in addition to the product of the matrices, I also want the transpose. So what is the transpose? Transpose is if i j is the element here j i is the transpose. So ci j is the main matrix, dj, i is=ci, j. So your d is the transpose of c, so this particular program it calculates for you the product of the matrices as well as the transpose and it returns both these things. Once you call this program and you get the control back to the main program, you have calculated the product as well as the transpose.

(Refer Slide Time: 22:22)



So now suppose I want to do another program. What this program is trying to do now, you transfer a matrix to a subroutine, you transfer an array to a subroutine and you want to do calculations on a given array in that subroutine. So what is the calculation you want to do?

You find the sum, average, maximum and minimum of a particular row of a matrix. So you have many matrices.

Let us say you want to find the maximum value in the ith row, maximum value and minimum value, sum as well as the average of all the numbers in the given row. So what do I have to do? From the main program, I have to call a subroutine, I have called it array c and the first one is the matrix which I want to use. Next one is i row and in that row called i row I want the maximum value, minimum value, sum of all the numbers in that particular row, average of those numbers and this n particularly is the size of that matrix.

So once I call this matrix, once I call this subroutine array c, it will go to this particular subroutine. So in the main program, it is a call statement, array c. In the subroutine, the first line is subroutine array c a irow amax amin sum average n. Whatever these objects are 1 2 3 4 5 6 7. If there are 7 objects in this column exactly 7 objects should be there and since a is a matrix I have given a dimension here, dimension a20, 20.

These other objects are not matrices, your irow is just an integer amax, amin these are real variables, sum is the real variable, aver is a real variable, n is an integer. You know that in fortran i j k l m n correspond to integers and all the other characters will refer to real variables. So it is a good idea to call all your integer variables to begin with i, m, n and so on. So the first line is the subroutine statement, is dimension 20, 20.

It is a 2-dimenisonal array. Now I want to calculate the maximum and minimum in the ith row, in the i row, i row is giving me the row in which I have to do the operations. So what is my statement here? amax is a*irow, 1, the first value of the ith row I am calling it as maximum, the same one I am calling it as a minimum also, the sum is also equated to the first element, this i row, 1 is the first element in your row called i row.

So once I do that then now I want the maximum and minimum in that row as well as the sum, so what does this, now there is a do loop, it goes from 2 to n up to the last value of the integer which correspond to the size of their row. So if that it is going from 2 to n, if their second element is>amax then amax is made to the second element. If the second element is<amin, amin is made to be the second element.

So on this compares this particular these two lines compare each of your later elements of that row and see whether it is lower than some value or larger than some value. If it is larger value, I will put it in amax. If it is a lower value, I put in amin so that at the end of my calculation I will have a maximum as well as minimum of that particular array and since I also want a sum I have made sum=sum+a row, j.

So whatever are your elements of that row, I have all added them into a variable called sum and at the end of the do loop what would have done, sum has got the sum of all the elements of that array, amin is the smallest value, amax is the largest value and I have also calculated the average=sum/real value of n. Remember your n is an integer, so when I divide I want to divide a real number by a real number.

So I have called it average=sum/real n. Now I will write star, star amin minimum value, maximum value as well as the average. What this subroutine is doing? It is calculating the average maximum, minimum and writing on the screen. So this is what my subroutine is. So what we have done by now?

We have taken several examples of functions, several examples of subroutines and we have given you how to access a function or a subroutine through a call statement or a statement where y=sin new n, it just equates variable to the name of the function. So once we understand subroutines and functions very well, it allows us to structure the program in a very systematic way, a very complicated problem can be put into several small, small tasks.

That is how exactly how suppose how does a car work, you have a driver and you have an accelerator, you have clutch, all these other systems work as subroutines whereas your driver works as the main program. So this way the program can be made more compact, all the task can be separated into functions and subroutines and then you can display your entire program in a nice flowchart. Remember, I had a flowchart in the beginning of my lecture today.

So the entire structure can be displays as the flowchart, so then you understand the logic and structure of the program. So now this will complete most of our discussions on functions and subroutines, elementary programming. Next lecture onwards we will go to numerical methods which allow us to use the power of numerical methods to do computations in chemistry. So I will conclude today's lecture here. Thank you.