

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

Lecture - 08

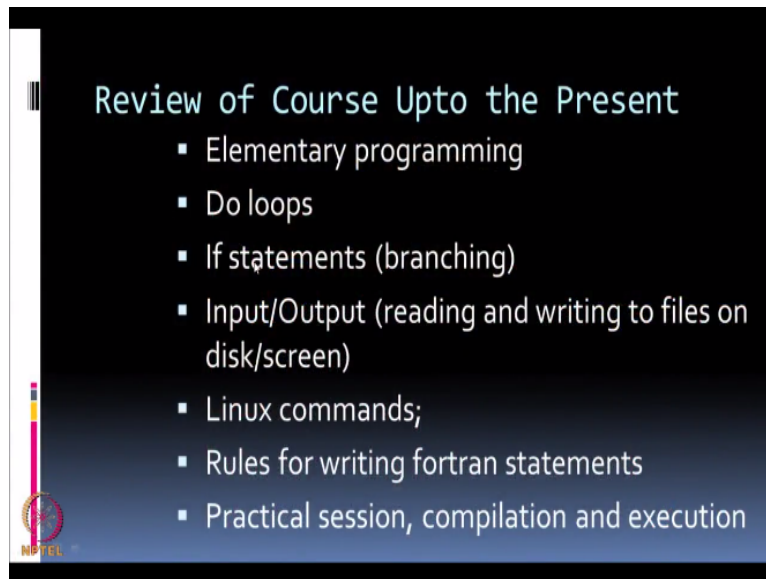
Programming Techniques 5. Formats, Functions and Subroutines

Hello and welcome to today's lecture. So before I start with today's material, let us review what all we have done so far in our course. First thing that we did we did elementary programming. What is elementary programming? Whatever formulae you have you want to calculate them using the computer. So for that you need a language which converts all your mathematical language into a programming language.

For example, e to the x , in normal writing we write e raised to power x , in a programming line there is nothing like raising something to the power. So we write it in the program as x into bracket x complete bracket. So these are simple translations of formulae into a programming language. So I have been using fortran as a programming language. So in addition to the simple formula translation, what we need is the special techniques that are relevant to the programming language.

One of them was is do loops. So what do the do loops do? Do loops allow you to do an operation any number of times as you want, so the statement was do 10 i going from 1 to n. So that means all the statements up to line up to a statement defined by line 10 will be executed n number of times. Those were the do loops.

(Refer Slide Time: 01:38)



Review of Course Upto the Present

- Elementary programming
- Do loops
- If statements (branching)
- Input/Output (reading and writing to files on disk/screen)
- Linux commands;
- Rules for writing fortran statements
- Practical session, compilation and execution

Then, we had if statements. What do the if statements do for you? It is a condition, if $a > b$ you do something, if $a < b$ you do something else, if $a = b$ you do something else. So this is basically branching. Your program is going from top to bottom and at some stage if you want to branch, you will use an if statement. So then the crucial thing was input and output. What are these inputs and outputs?

Suppose you want to read a lot of data into your program, you can always read it from the screen but reading from the screen can be very tedious because suppose you have 100 numbers to read and you make a mistake in the 94th number, you will have to type all the things again. So instead of reading and writing from the screen, it is better to read and write from files.

So we also discussed how to open a file from a program using an open statement, open and unit numbers. We have done that also reading and writing from files. Then, most of our operations are in a Linux operating system okay like when you compile a program with gfortran. So this is a compiler in Linux operating system. So you use Linux operating system to compile and execute your programs.

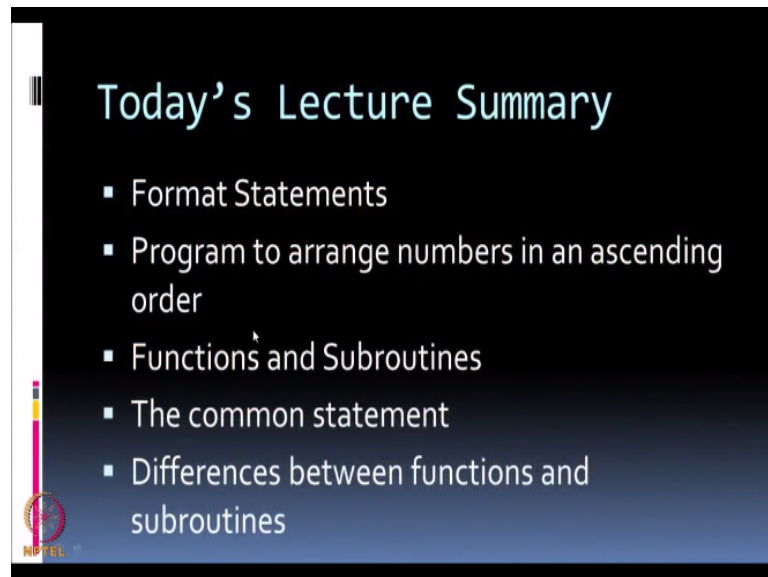
Then, since we are using fortran as a language, there are several rules for writing fortran statements. One of the first rules is that all the characters in the line begin from the 7th column okay. The first 6 columns have to be left blank and if one line continues into the other line, you put a in the 6th column you write 1, 2, 3, or some characters so that it continues to the second line.

Then, line numbers were written in the first 5 columns. Remember, all line numbers are in the first 5 columns and one of the other rules was that all variables in the program which start with i, j, k, l, m, n these are all integers and all the variables that begin with a, b, c, d, e, f, g or x, y, z they are all real numbers. So the programming language gives you this facility to make some numbers, integers, make some other numbers real, so that you do not have to declare all these statements.

In other programming language, before you start writing the program you have to declare everything. So fortran enables you to have this default declaration statements. Then, finally we had one practical session. What we did in the practical session was we executed a simple program. Then, we learnt about compilation, we learnt about execution. There were some errors also in the program.

We will correct that error then we take up the next practical session. So what we will do today that is outlined in my next slide.

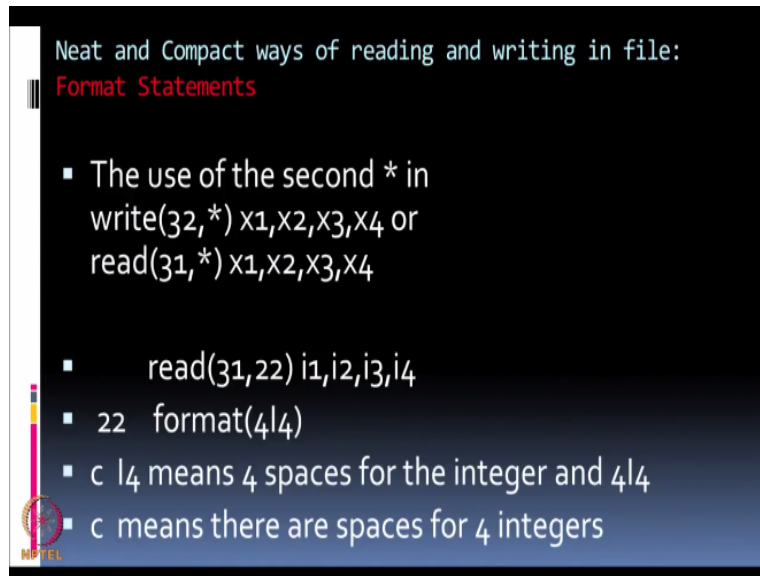
(Refer Slide Time: 04:24)



So what we will do? Today, we will discuss a little bit about format statements. Then, we will consider a program to arrange numbers in an ascending order. So this is one of the uses of programming, you can arrange things the way you want. Then, we will discuss the important concepts of functions and subroutines. Functions and subroutines are subprograms which makes your program stronger by compartmentalizing all your tasks into different units.

Just like an institution works with several departments, your different programs are like different departments. Each one will do its own function and return it to your main coordinating program. Then, there is also a common statement, we will illustrate that. It is very much of use when you do subroutines and functions. Mainly, it is used for subroutines. Then, we will discuss differences between functions and subroutines. So let us now begin with a format statement.

(Refer Slide Time: 05:27)



Neat and Compact ways of reading and writing in file:
Format Statements

- The use of the second * in `write(32,*) x1,x2,x3,x4` or `read(31,*) x1,x2,x3,x4`
- `read(31,22) i1,i2,i3,i4`
- `22 format(4I4)`
- `4` means 4 spaces for the integer and `4I4`
- `4` means there are spaces for 4 integers

So you remember that when we wrote something or write something from the screen, we said write star, star x1 x2 x3 x4. So what was the meaning of star, star? Star, star means the whole thing was writing on the screen or the whole thing was read from the screen. Now in this statement what I have is write 12, star x1 x2 x3 x4. So what this does? This file this 32 this is a number this is connecting with some file number.

So remember open unit=32 file=output.dat or open unit=31 file=input.dat. We have seen those open statements which connect a number to a file name in your directory. So this way all the output as well as input can be written in a file. So that your whole screen remains uncluttered because if you write 1000 lines on a screen, your screen can only see some 20 or 30 lines and you will miss all the earlier data.

So reading and writing from files also we have studied but now we want to study what is the format because the second star here, the second star will refer to a line number. Now look at this line, what this line says is read 31, 22 i1 i2 i3 i4, so this is my read statement. So this is

going to read information from file which is connected with this number 31, that is the file with which I am connecting, 31 is the file.

You have opened some unit with some file name which is connected with 31, 22 is a line number, look at this. I have 22 as a line number that 22 gives you the format in which the data is written. So in this case my data is i1, i2, i3, i4. It is going to read these 4 numbers. These 4 numbers are integers because I begin with i, each variable begins with i here i1, i2, i3, i4. So since there are 4 numbers, so I am giving this format, format 4I4.

What this 4I4 means? I4 means each integer has 4 spaces to write the number. This 4 means there are 4 integers like that. So 4I4 would mean I can write 4 numbers in that line and each number will have 4 spaces okay. So that is what I have written here, I4 means there are 4 spaces for the integer and since there are 4 integers, I will give 4I4 as my format. Format really is the structure in which you want the input to be read or the output to be written.

So it gives you a better control over your entire line space in your input or output. Otherwise, if you instead of 31, 22 suppose I had said 31, star that means I can give those 4 integers in 4 different lines. I may be able to separate them by comma, so when I give a star here I am reading from the screen and the total format is arbitrary whereas if I give a format here then I cannot write arbitrarily I should give exactly in the format given.

It is like in a classroom we have 10 chairs in a row that means these only 10 people can sit in this row, not the 11th person. So like that format gives you an arrangement of your line into which you can write the information or from which you can read the information. So this particular statement I considered it was for integers. Now suppose I have real numbers, how will I read real numbers or write real numbers?

(Refer Slide Time: 09:23)

Formats for real variables

- `write(32,24) x1,x2,x3,x4,x5`
- 24 `format(5E12.4)`
- c each number has 12 places, 4 after the
- c decimal point. There are 5 places for 5
- c numbers
- c `-20.4433E-05233.7435E+04.....`

That is given in the next line. Now look at this, this particular statement I say write 32, 24 x1 x2 x3 x4 x5. So these x1 x2 x3 x4 x5 are all real variables. So real variables means what everything will have a decimal point. This we call as real number. An integer has no fraction, so it is only a full number. So whereas a real number has a fraction, so that fraction I can write in many ways. I can write it as 1.23 or I can also write in an exponential format.

That exponential format is useful because suppose I have a number which is 0.0000 with 22 0s then followed by some numbers. Then, it is not possible for me to write so many 0s in my output. So it is best to write it in the exponent format, so that is what we are going to do. So these are written, so since these are 5 numbers, my format is given in line number 24. So I say write 32, 24 and 24 is the format statement, 24 format 5E12.4.

So what is the meaning of this 5? This 5 means there are 5 numbers which are going to be written and E12.4 it means that it is an E format. E format is an exponential format, 12.4 now we need to understand what is this 12.4? 12 means total 12 spaces are allotted to that number and 0.4 that means there are 4 numbers after the decimal point. Look at this last line. This last line says `-20.4433E-05`.

So these are the 12 spaces, since I have 5 numbers I have given 5 times 12, so each number takes 12 places. Now let us see how the 12 places have been taken – is one place, 2 is one place, 0 is third place, point after the decimal points you have 4433. So 4 after the decimal point, one space for the decimal point, 0 and 2 these take up two more, then this –sign, so -20 point, these take up 4 values, 4433 takes 4 values, then E-05 these take up 4 spaces.

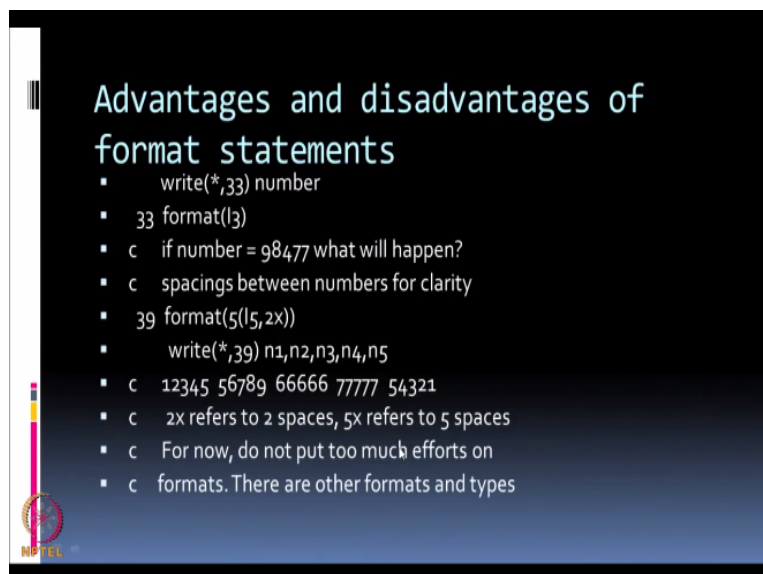
This tells me that this number is -20.4433×10 to the power of -5 . This E really means 10 to the power. So 10 raised to -5 , so my first number is $-20.4433\text{E}-05$, my second number is 233.7455E to the $+04$. So what is the meaning? The second number is 233 point. That means there are 233.7435×10 to the $+4$, so which means what this format allows you to do, it allows you to keep only 4 numbers after the decimal point.

And since there are 5 numbers like that first takes up 12 spaces, second one take up 12 spaces and so on so there will be total 5 numbers. So the whole thing will take up 12×5 5×12 that is 60 spaces. So this is how it will write. If you had instead written write 32, star x1 x2 x3 x4 x5 it would have written first number and given some arbitrary spaces of the first number, then the second number, then the third number.

The computer will give its own format whereas here you have arranged them in your own format. So while this is good there is one problem. One problem is that between the first number and the second number there is no space to make a distinction between two numbers. I really we would have write $-20.4433\text{E}-05$. Then, suppose I had one or two spaces here then I could write the second number.

Then, again some spaces some more numbers, so it would be a better arrangement like when you arrange several chairs in a row you keep some space between the chairs. So how do I do that? That is indicated in the next slide.

(Refer Slide Time: 14:00)



Advantages and disadvantages of format statements

- `write(*,33) number`
- `33 format(l3)`
- c if number = 98477 what will happen?
- c spacings between numbers for clarity
- `39 format(5(l5,2x))`
- `write(*,39) n1,n2,n3,n4,n5`
- c 12345 56789 66666 77777 54321
- c 2x refers to 2 spaces, 5x refers to 5 spaces
- c For now, do not put too much efforts on
- c formats. There are other formats and types

So look at this now. I have illustrated many things in this slide okay. Now let us before I discuss the slide let us see what are the advantages and disadvantages. Advantages is that you can write it in a very nicely ordered format just as you arrange your room very nicely you can arrange. The disadvantage is that when you input the data you should know what the format is because if you do not know the format in write in different ways, it will take some arbitrary values.

So the way you give input you should give exactly the way the format is written. So the only disadvantage is you should remember exactly what the format was but with practice it is not a major issue because you will have some each one would have his own standard formats, you can read your data in that format, you can write in that format. So you will get used to it. So only disadvantage is you have to remember, advantage is the nice arrangement.

Now let us take this particular next example. The next example says write star, 33 number. Now what is the meaning of the first star? First star says that you will be writing it now on the screen and not to a file. So write and what is the format for that number? Format i3, so the meaning of i3 is there are only 3 spaces given to that particular number. So you will write that number in that format in 3 spaces.

Now let us take this example, suppose 33 is the line number and format i3 is the format into which you want to write the number. Now look at this next one, c means comment card. We all know that in any fortran program you begin everything from the 7th column. In case there is a character c in the first column, it is a comment. Now what is the meaning of this comment? Suppose the number I want to write has the size of 98477.

So I want to write it in i3 format, there will be a total confusion because you have said that the format can have only 3 characters, 3 spaces but the number is very large. So computer will be in deep trouble which number should it write. Should it write 477 or should it write 984, both would be wrong. So therefore the format should be such that it accommodates all possible numbers that your output can be.

If the output has very large numbers, so then you should give a format which will have so many spaces as many are needed by the number okay. So this is one point, so one of the disadvantage of a format statement like i3 is that if the number is more than 3 digits you

cannot use this format. So all this you have to keep track when you give your format. Now we will come back to my earlier thing.

Now suppose I want spaces between numbers for clarity. Remember, we wrote 5 numbers earlier. Now let us also write 5 integers this time, let my 5 integers be n1 n2 n3 n4 and n5. These are my 5 integers, I want to write them on the screen and I want to write them in a format such that between every two numbers there is two spaces. So now I have given this format statement.

As I mentioned before, the format statement always begins with a line number and this line number should be in the first 5 columns. So I have given 39 in the first 5 columns then the format into bracket 5 into bracketed I5, 2x bracket complete, bracket complete. So one formula you should remember or one thing you should know. As many write brackets are there you need so many left brackets. So what I have done since there are 5 numbers, I have given this outer 5.

Then into bracket that is I5, 2x. What is the meaning of I5, 2x? That means I5 is the space required for the integer, 2x, 2x means 2 blank spaces and bracket complete. So what is in the inner bracket? I5, 2x that means 5 spaces for the integer and two blank spaces bracket complete and 5 times to the whole thing. So that means you will have 5 numbers, each number will have 5 spaces and 2 blank spaces after the number.

So all this is enclosed in the outer bracket. So the format is there are 5 numbers to be written and how do you write each number, first 5 digits are your numbers and the next two are the blank spaces. So that is my format statement and the next line is write star, 39. So this 39 is the line number, the format statement number can either come before this write statement or after the write statement.

There is no rule that the line number should be after the write or before the write, you can write the format statement wherever you want. A good strategy would be to write all the format statement at the end of the program so that it does not clutter your program significantly. So now let us see how it will write these 5 numbers. The way it would write these 5 numbers are I am just since it is a program, I have given a comment the way it would write the first 5 numbers 1, 2, 3, 4, 5 and two blank spaces.

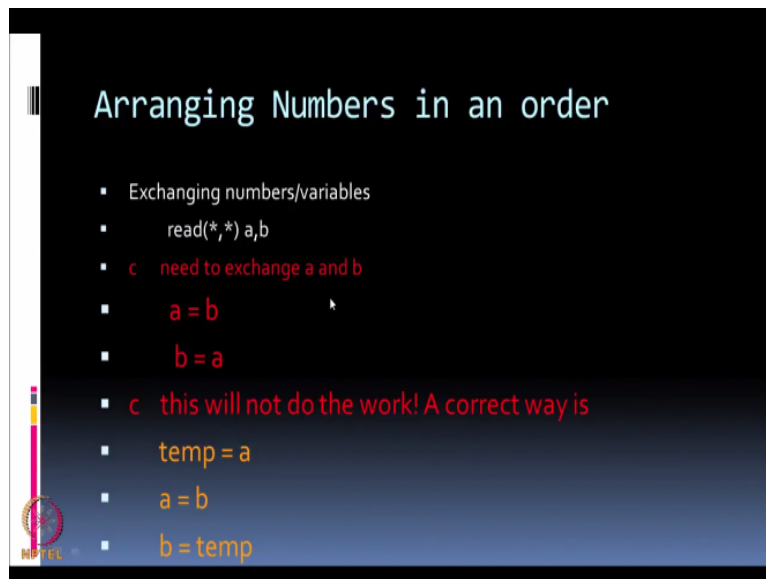
Then, the next 5 numbers, two blank spaces, the next number, then next number written in 5 spaces, two blank spaces, next number 2 blank spaces, next number and 2 blank spaces. In this case, all the 5 numbers were 5 digit numbers. Suppose some number is a 2 digit number, then it would use suppose instead of 77777 I had only 77, the way the program should write it will not write anything in the first 3 and it will write 77 in the last two columns.

So you do experiment with this, what I recommend to you, you write a program, use different formats, so we have discussed only two formats. The E format for real numbers and the integer format. There are other formats as well but we need not worry too much because if you know one format to write real numbers and another format to write integers that will suffice that will be sufficient for most of our work.

But sometimes you may want to write characters into the output. Then, what you have is called a character format. So as and when we need it, we will use that character format in reading as well as some times you may want to read a name of a person into a program, you may want to write name as an output, so there are other formats, we will use it as and when we need it or you can just look through how to write characters in fortran.

Now to summarize again what we did 2x refers to the 2 spaces after the number and instead of 2 spaces suppose I wanted 5 spaces between numbers, I will read 5x okay. So this is what is the summary of the format statement. So from now on what we will do, we will not put too much effort into format or types of format. Our main purpose is the logic in the programming, so we will concentrate not too much on formats but go through the logic of programming.

(Refer Slide Time: 21:11)



Arranging Numbers in an order

- Exchanging numbers/variables
- `read(*,*) a,b`
- `c` need to exchange a and b
- `a = b`
- `b = a`
- `c` this will not do the work! A correct way is
- `temp = a`
- `a = b`
- `b = temp`

So the next thing I want to do, I want to discuss a program. It is a very simple program but there is a main idea in this program. We want to arrange numbers in an ascending order, some order, it could be ascending order, it could be descending order okay. So now before we go into this let us see what is the logic when you exchange two variables. Suppose I want to have a number a and b, there could be some values for a and b.

I want to exchange the values of a and b, suppose a is 4 and b is 3, I want a situation where a becomes 3 and b becomes 4. So how do I do it in a program? So what is the first thing you may want to do suppose I want to exchange a and b. The simple idea would be `a=b` and `b=a`. This is how a normal person will do but there is a major problem with this. Let us understand what is the problem.

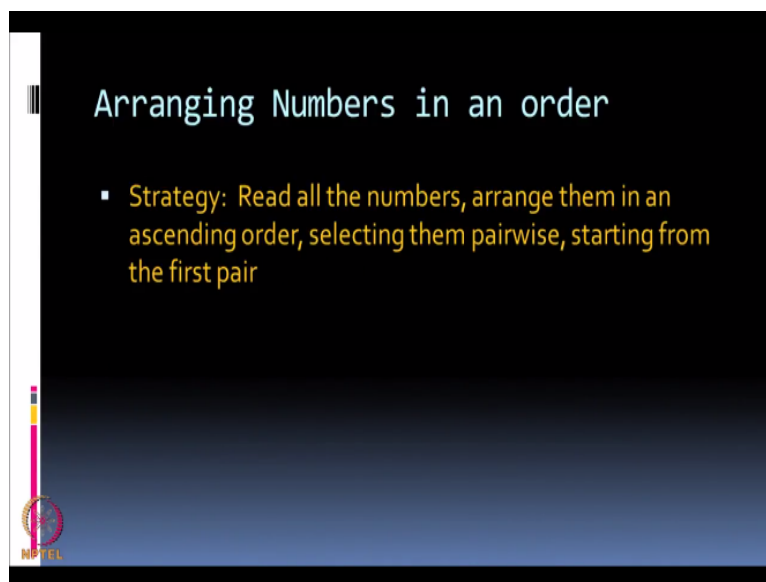
So let us say `a=3` and `b=4`, then my first thing would be `a=b`. So what it has done? a was already 3 but when I say `a=b` b was 4, so it makes `a=4` okay. So in the next line says `b=a`, now `b=a` but a is already become 4, so what this will do, it will make both a and `b=4`. So the whole idea that you wanted to exchange a and b is lost because whenever you write a statement `a=b` what it does is it replaces whatever original value of a by the value of b.

So since b was 4, a has become 4, now next line, next b also=`4`. So rather than exchanging anything, all you would have done you have made it both a and b as 4. So this is not a correct way to do it. So the correct way to do it is to use a temporary variable. So let us say a was 3 and b was 4. Now I said `temp=a` so that means this temp has the original value of a into that variable, so temp is 3.

Now next line is $a=b$, so b was 4 now a becomes 4. Now in the next line is $b=temp$, now b becomes 3 because $temp$ was already storing the original value of a . So what does this illustrate? Whenever you want to exchange two objects in a program whatever you want to exchange one of them you save in a temporary location or a temporary variable, then you exchange them so that the final result now a and b would have a different order than the original.

So remember whenever you want to exchange, the first variable you have to save it in some temporary location or as a temporary variable okay. So now this idea is very crucial. I shall use this idea to arrange numbers in an ascending order. Now that is my next program. So let us start arranging numbers in an ascending order.

(Refer Slide Time: 24:02)



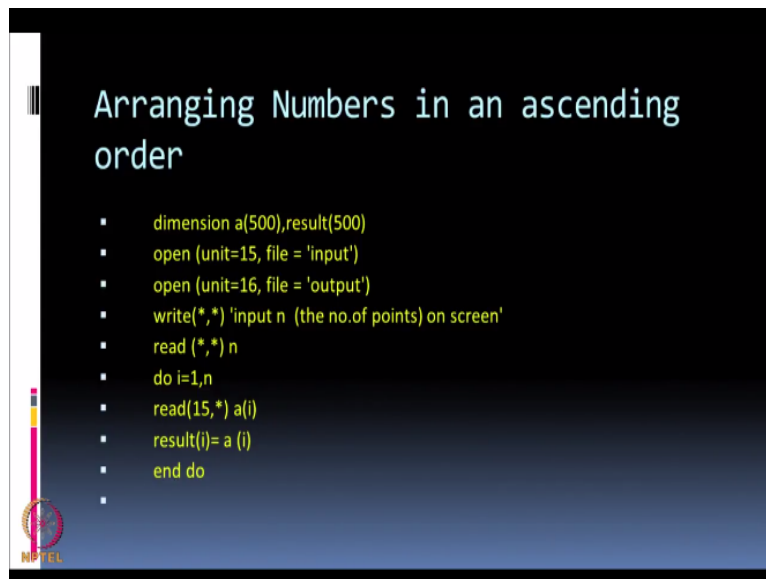
So what is the strategy before you actually arrange? Let us see what is the strategy. What I will do, strategy would be first you have to read all the numbers because you want to arrange them in some order. So you have to reach them in all the numbers. So how will you reach so many numbers, you read them in an array. Remember we have already discussed arrays, array variables.

You will read all of them in an array because in an array each number will be assigned with one of the array variables. Suppose there are 10 variables in an array a_1 a_2 a_3 up to a_{10} . Then, these 10 are the variables. Then, I can arrange a_1 in place of a_2 , or a_{10} in place of a_3 . I

can arrange easily in a do loop. So I need to use a dimension variable to read all the numbers. Then, to arrange them in an ascending order I shall use them select them pairwise.

Because at the same time I cannot select all the numbers, so I will take two at a time and if I want in an ascending order I will start keeping the lower numbers in the lower values of the array and higher numbers in the higher values of the array. So I will keep on pairwise I will compare, exchange them so that the final thing would be entire sequence arranged in an ascending order.

(Refer Slide Time: 25:25)



So let me now start the program. So this is my program to arrange numbers in an ascending order. So what I will do here, first line would be I need to declare a dimension. So dimension a500 result 500, I just have what I have done I have given an extra result as a variable so that I want to keep the original array as it is and in the result I want to rearrange numbers okay. So since I want to read several numbers and I want to read it from a file.

I do not want to read from the screen because as we already saw there if there are 100s of numbers there could be mistake in typing so I will open a unit, I will call it 15 file=input. Then, I will open another file, I call unit=16 file=output. So now I have an input file, I have an output file, now I want to read all these numbers. So before once the program executes it will write on the screen input and the number of points the number of numbers on the screen.

So first thing the program once you this line, on the line you will see input n the number of points on the screen so then the program will wait for your input. So your input say n, so let

us say $n=10$, you want to arrange these numbers in an ascending order. So 10 numbers you want to read from this 10 is read from the screen. Now you will read all these numbers from that file file=15 okay.

So this is just the part for reading the numbers from that file. So in the next class, I will execute the logic for arranging the numbers in ascending order. We will conclude this session now and in the next session I will begin with this particular slide and we will review what we have done and we will start again with the session of arranging numbers in an ascending order.