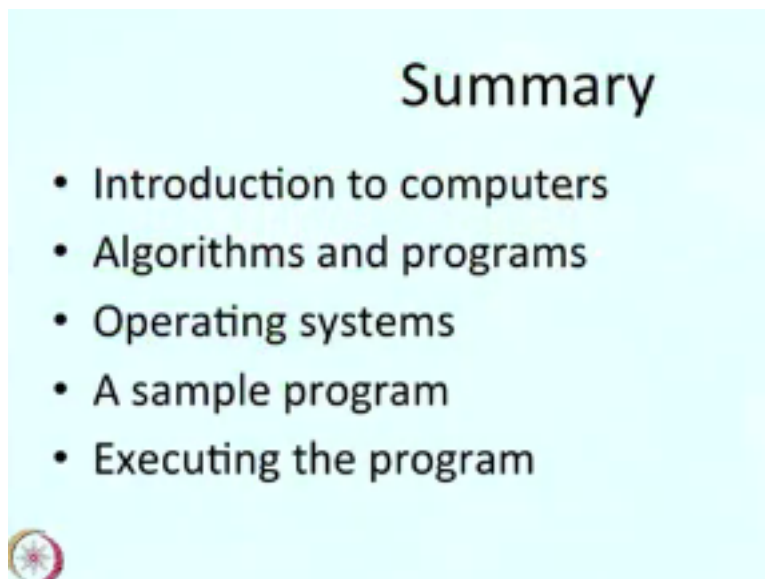


**Computational Chemistry & Classical Molecular Dynamics**  
**Prof. B. L. Tembe**  
**Department of Chemistry**  
**Indian Institute of Technology- Bombay**

**Lecture – 02**  
**Writing Simple Programs: Compilation and Execution**

**(Refer Slide Time: 00:28)**



## Summary

- Introduction to computers
- Algorithms and programs
- Operating systems
- A sample program
- Executing the program

Hello and welcome to the next lecture before I start, let me summarise what we did in the earlier lecture, in the earlier lecture, we gave an introduction to computers and outline why we really need to use them for calculations in chemistry, we discussed algorithms and we discussed the simple program, we talked of operating system not too much about them but we are going to use the Linux operating system not Windows.

Because Linux is the free software and a very, very powerful software that the sample program we did last time but we did not execute the program, now what is the meaning of execute the program? The program that we wrote was in the Fortran language, so do execute that it has to be converted into machine language into an executable file, okay, so that I will explain how to do that, so that process is called compilation.

**(Refer Slide Time: 01:12)**

## Compilation and execution

- Compilation: Converting your program file in a programming language into an executable file.
- `f77 prog.f` or `gfortran prog.f`
- The name of the executable file is `a.out`
- To run the program, type `./a.out`



Compilation is converting your program file in a programming language, it could be any language; Fortran, C, Pascal, basic and you want to convert that into an executable file, an executable file is the one which actually executes, okay so there is a command for that the command is `f77 program.f`. Here, that `f77` is a command and `prog.f` is your program file which was written in a Fortran language.

In some situations, where `f77` is not working, you may have to give `gfortran`, so `gfortran` is another compiler, `gfortran` is a compiler very much like `f77`, some systems may want `gfortran`, some systems may ask for `gf77`, both these are commands in that computer, so using that command, you convert that `prog.f` into an executable file. Now, the name of that executable file is `a.out`.

So, let us not worry at this time all the details, the main thing to learn here is that `f77` or `gfortran` is a command which converts your `prog.f` into an executable file called `a.out` and how do I run the program; simply type, `./a.out`, so you please remember this `./a.out`, all the programs that you compile it will be converted into a file called `a.out`, you and you execute it by typing `./a.out`. There are other ways of doing it but we will do it as we go along, okay.

**(Refer Slide Time: 03:08)**

# A sample program

```
c  my first program (this is a comment card/line)
   program myprog      (this is an optional line)
c  input an integer
   read(*,*) n
c  sum integers from 1 to n
   msum = 0
   do 10 i = 1,n        ( this is a do or for loop, for repeated operations)
     msum = msum + i
10  continue
   Write (*,*) n, msum
   stop
   end
c  use all small case letters for convenience
```



Once you type a.out, the computer will ask you to input that number m, so let us see what that program was, the earlier program was; you read any integer, sum all the members up to that integer and write the sum on the screen, so once you compile and type a.out, it will ask you to this value n from the screen because that is what this \*.\* means, read n from the screen, sum all the integers up to the value n through this do statement.

A do statement is nothing but it will repeat all the operations till this 10 continue until the final value of this integer variable is reached, i is an integer variable it will go from 1 to n and it will sum all the values then finally, write that thing on the screen; \*.\* refers to screen, so in the program you read n, wrote the program, compile the program, when you execute, it will read n from the screen, do all the calculation and file; finally, write n and msum on the screen.

**(Refer Slide Time: 04:14)**

## Summary

- Introduction to computers
- Algorithms and programs
- Operating systems
- A sample program
- Executing the program



(Refer Slide Time: 04:16)

## Review of Last Lecture

- Introduction to computers
- Algorithms and programs
- Operating systems
- A sample program
- Compiling a programme: `f77 prog.f`
- Executing a program `./a.out`



Once you get `n` and `msum` on the screen, your program work is over, okay, so we talked of compilation and so I have already reviewed what all I have said in the last time.

(Refer Slide Time: 04:22)

## Types of Files

- Fortran Files: prog.f, qchem.f, stats.f ....
- C program files: prog.c, nmr.c,....
- Data files: kinetics.dat, mydata1,...
- Executable files: ./a.out
- prog.exe
- **How to compile programs**
- f77 prog.f This creates a file ./a.out
- gcc prog.c " "
- **To run a program, just type ./a.out**



Now, let us discuss what are the types of files, we already talked of Fortran files, what were the Fortran files; prog.f, qchem.f, stats.f, any Fortran files will have extension .f, so instead of Fortran suppose you had use the C language, if you had use the C language, you will call prog.c, nmr.c, all the C programs will have an extension of .c. Now, you can have other extensions also suppose, you have lots of data, so to identify a file as a data file, you may just want to call it with an extension .dat; dat, kinetics.dat, mydata1.

You can give many, many names but it is always good to give a name which gives you an idea what you are trying to do, so what are executable files? You already knew ./a.out, this is an executable file, you can instead of ./a.out, you can call it a prog.exe, you can give many, many names but executable files will have .exe extension, remember I already told you that when you do f77 prog.f, it creates a file called a.out.

When you; if you want to compile a C program, it will be gcc prog.c but to run the program again, it is ./a.out. Remember again, you write a program in the language of your choice, you convert it into an executable file and execute the executable file by typing ./a.out, so this is what we did earlier, write a program, compile the program and execute it, okay. Now, again let us summarise what we did.

**(Refer Slide Time: 06:19)**

## Algorithms, Flow Charts and Programs

- For our purpose, these are three representations of the problem that we need to solve, the last one, being the most useful
- **Unambiguous sequence of steps/statements** for carrying out any task
- A flow chart outlines or presents the algorithm in such a way that the **flow of control of execution** (recall traffic signals and police men) is clearly visible.



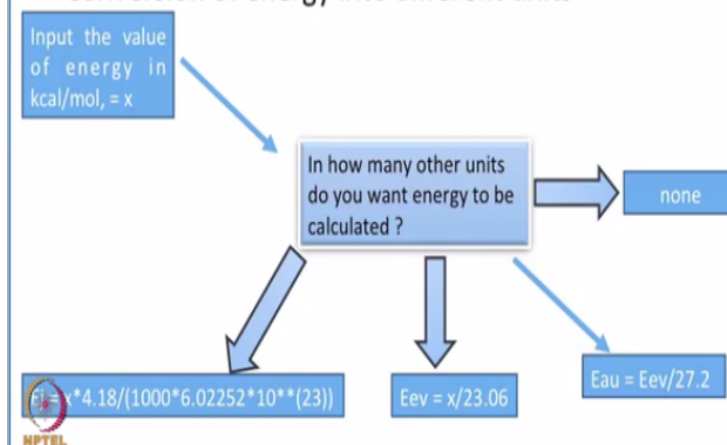
We talked of algorithms, remember last time, we talked of algorithms, what are algorithms? These are nothing but representation of the program in a very unambiguous way, okay so whatever program you want to solve like the last one, okay, it is a very simple program, you want to have unambiguous sequence of steps or statements for carrying out the task. So, often when you write an algorithm, it is a long sequence of steps.

So and it is easy to get confused when you have a long number of steps, so it may be better to write it as a chart that is called a flow chart. What a flow chart does; it outlines or represents the algorithm in a pictorial way, so that the control of flow is easily visible. For example, what are traffic lights, what traffic lights do? It says that green means go, red means stop, yellow means be careful, so program flowchart is just like a collection of traffic signals.

**(Refer Slide Time: 07:19)**

## Example of a flow chart

- Conversion of energy into different units



So, let us consider this particular example of this flowchart, what I want to do in this case is I am given energy in one particular unit and I want to convert that energy into different units for example, let us say that the value of energy is given in kilocalories per mole, let us call that x, so if you have energy in kilocalories per mole, I want to convert that into say, electron volts, in to say atomic units and into say, in Joules.

So, I want to convert my given energy into different units and this is a common requirement in chemistry because you are; you get energy in some units, you want it in some other units, you can of course use a calculator but if a program which converts from one unit to all other units at one shot, you have got all the answers, so this is what I will do in the next program, get the input in some unit kilocalories per mole and write the output in different units.

So, this is the flowchart I have an input, the arrow says I will read and go to this, what this oval shape tells me in how many units, do you want the energy to be converted, okay if you do not want to convert, say none and the program will stop but if you want to convert in electron volts, in atomic units and in Joules, you want to do in 3 units and what we will do in the next slide is to convert this into all these 3 units.

**(Refer Slide Time: 08:42)**

## A program for the previous task

```
• C  read the value of energy in kcal/mol
•  read (*,*) x
•  eev = x / 23.06
•  eau = eev/27.06
•  ej = x * 4.18 / (1000 * 6.02252 * 10 **23)
•  write (*,*) 'energy in kcal/mol, ev, au, j'
•  write (*,*) x, eev, eau, ej
•  stop
•  end
```



And this will also allow us to know how programs are written, how the operations in program are done like multiplication, division, okay so remember, now I want to write a program actually, it is the program for the previous task but of course converting units from one to the other is the precious thing because you need to do it all the time. So, first one is a comment card, C is the comment, read the values of energy in kilocalories per mole, so the first line says read into brackets, \*, \* bracket complete x.

You know from last lecture that anything into brackets \*, \* start means read object x from the screen, so the next line it says, eev is  $x/23.06$ . What this line is doing; if you divide kilocalories per mole by 23.06 that is a conversion factor to go from kilocalories per mole to electron volt, so the first line converts your energy from kilocalories per mole to electoral volts, so you got energy in this line in electron volts.

The next line says if you want the energy in atomic units, you divide the electron volts by 27.06, 27.06 electron volts is one atomic unit of energy, the atomic unit of energy is called Hartree, so by dividing electron volts by 27.06, you got energy in atomic units which is nothing but a Hartree. The third line says I want to convert that into Joules now, earlier I had kilocalories per mole now, I want to convert that to Joules.



To convert that to Joules, I am multiplying kilocalories per mole by 4.18, so that converts into kilojoules per mole and divide that by 1000 because I had already in kilocalories, so when I divide by 1000, I get in kilojoules, now it is kilojoules per mole. So, when I divide kilojoules per mole by the Avogadro number, I get just in kilojoules, so I want to go from kilocalories per mole to Joules, to do that I am multiply by 4.18, divide by 1000, multiply that by Avogadro number.

Remember that this Avogadro number is in the denominator because I have an object per mole and I divide number per mole, so per mole and per mole cancels, so what I get it energy in Joules, so once I do this calculation, I want to write all these results on the screen the way it is done, write, \*,\* that means again on the screen now, I have done something different from the past. I have put inverted colon and energy in kilocalories per mole; ev, au, j.

And this bracket should not be bracket but it should be that inverted colon, remember I have a one colon here, it should instead of this bracket, it should be a colon, so whatever is written between 2 colons, it is just printed on the screen. So, what this line does; it writes all these things energy mole etc. on the screen, remember this is not a bracket, it is a colon just as I have this starting colon, I will have starting colon here, so it will write on the screen.

And the next line says write \*,\* x, eev, eau and ej, so this is the line which really writes all the 4 quantities on the screen. Now, why I have written these energy and all this in quotation because once I write this in the second line, this energy in kilocalories per mole will be x, in ev will be eev, in au would be eau and ej would be j, so this is how I am writing the heading, it is like a tabular format.

So, I can give the output as the tabular format, okay, this is just the starting point of how I write anything other than variables on the screen, so all those will be written in a quotation mark, so once this is done, I say stop and end, so this was the second program not a precious but previous task I converted energy from kilocal per mole to different units, so this is one application. So, now let us see some different functions in Fortran.

In the first lecture, I did discuss several functions, now let us say, I will go line by line, so that you will know what each line is doing.

(Refer Slide Time: 13:13)

## Elementary functions in Fortran

$$y = x + z / c + a * b$$

- $p = \sin(\text{theta}) + \text{alog}(\text{conc})$
- $q = \exp(-\text{energy} / (\text{boltz} * \text{temp}))$
- $r = d ** f$
- $t = \text{sqrt}(23.0)$
- $a = a + b$



$$z = \text{acos}(y)$$

This says  $y = x + z$  divided by  $c$  and it is just a division operation,  $/$  is a division operation,  $z$  is divided by  $c + a * b$ ;  $a * b$  means  $a$  is multiplied by  $b$ , so in this line, there is an addition, there is a division and there is again a addition and a multiplication, so this is the first line. Now, second one is  $p = \sin$  of  $\text{theta}$ ;  $\text{theta}$  is some angle, I want to calculate  $\sin$  of that angle +  $\log$  of concentration.

Remember in kinetics, we use logs of concentration,  $\log$  of  $a/a - x$  versus  $t$ , it gives you the rate constant, so this is  $\text{alog}$  of concentration. Then the next line;  $q = \exp$ ; this is an exponential function,  $e$  raised to something, exponential \* bracket energy divided by Boltzmann constant \* temperature, you may know that  $Kt$  is like an energy, so if I want something exponential of -energy divided by  $Kt$ , this is what I am going to do.

So, finally suppose I want some object  $d$  raised to power  $f$ , so when you want to raise an object to power  $f$ , so it is  $r = d ** f$ , this is raising a variable to the power  $f$ , so this is another operation, it is a function; exponential function. Then square root is a standard thing;  $t = \text{square root of } 23$ , I am just calculating square root of 23 and the last line which is not visible, it says  $z = a \cos y$ , so this inverse function, remember you have a  $\cos$  function.

Similar to cos function, there is a cos inverse function, most inverse functions are written as a cos, a sin, a tan, these are all inverse function, so that is a; these are now some of the elementary functions. So, now let us summarise what we did so far when we go to the next point.

**(Refer Slide Time: 15:12)**

## Summary of second lecture

- Reviewed the first lecture
- Types of files, compilation and execution
- Algorithms, flowcharts and program
- **Next Lecture**
- Algorithm for the sine function
- Calculation of the sine function
- Main ingredients of programs
- Convergence and iterations



We reviewed the first lecture, discussed types of files, discussed compilation and execution, we discussed, what is an algorithm, we wrote a simple flowchart; we wrote another program and executed that. So, what I want to do now, I want to write an algorithm for the sine function and I want to calculate the sine function then discuss, what are main ingredients of programs, we have written already programs.

But these programs just to certain elementary operations but a program will have different aspects, then we will also talk of convergence and iterations, so convergence and iterations are very important concepts in programming, this will become more and more apparent, as we go along, okay. So, what we will do now; we consider the sine function, we want to calculate the sine function.

**(Refer Slide Time: 16:05)**

---

## An Algorithm for calculation of the sine function

- $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + x^9/9!$
- How to construct an algorithm for the series?
- Observe the trends in the terms...
- 1) as n (no of terms) increases, terms become small; i.e, the series may converge
- 2) Each successive term can be obtained from the previous term by multiplying by  $x^2$  and



So, we want now an algorithm for the sine function up to now, algorithm was just some new word for you, so now I am actually giving an algorithm to calculate the sine function, so you know that many functions can be expanded in Taylor series. What is the Taylor series? It is a power series that is a function is expressed in terms of power suffix, so here sine function  $\sin$  of  $x$  is  $x - x^3/3 \text{ factorial} + x^5/5 \text{ factorial} - x^7/7 \text{ factorial} + x^9/9 \text{ factorial}$ , next will be  $-x^{11}/11 \text{ factorial}$ .

So, alternate signs will keep on changing sign and how far I have to go; I have to all the way, it is an infinite series, okay. Now, what I want to do is; to construct an algorithm for this series. The first question you will ask is that suppose, I go on summing large number of terms, will this remain finite or will this become infinite? So, by the saving graces, your denominators are factorials.

So, I have 9 factorial, 11 factorial, 13 factorial as  $x$  becomes larger and larger, these factorials are very, very fast rising function, if you for example let us say, 100 factorial; 100 factorial you cannot even calculate on your calculator because the number is too large, so factorials are very, very rapidly rising function and plus you are also having  $+$  and  $-$  sign, some terms are adding, some terms are subtracting so, therefore that also ensure that the series will converge; converge means, it will give you a finite number.

So, on a computer you can only calculate those functions, which are finite, if it is infinite you will get an error, okay. So, now what are our observations of these terms as  $n$ , the number of terms increases, terms become smaller and smaller, so the series I am saying it may converge in this case, we surely know that it converges because you can calculate the sine of any angle, it is a finite number.

Because you know, the sine value can take only between 0 and 1, we know that okay and the other observation is each successive term can be obtained from the previous term by multiplying by  $x^2$ , now you see, let us see, first term was  $x^3/3!$ , how do I get the next term; multiply by  $x^2$  and divide by 4 and 5, I had a  $5!$ , how do I get  $7!$ ; by multiplying by 6 and 7.

How do I get the next term? Change the sign, remember from first term, from second term to third term, I changed the sign, so next time again I change the sign, so  $x^5$  was there, I multiply by  $x^2$ , change the sign and multiply by 6 and 7, so I got  $7!$ . How do I get the next term? Multiply by  $x^2$ , change sign and divide by 8 and 9, so each term increases the factorial by 2 numbers and power by 2.

So that is how I can calculate each new term, I can calculate in terms of the old term, so that is called the recursion relation. A recursion relation is a relation which allows me to calculate newer terms in terms of the old terms, okay so that is what we are going to use, so that is what my algorithm is, so let us see what is an algorithm now. Algorithm is a set of steps, unambiguous steps which allows me to do the calculation.

**(Refer Slide Time: 19:35)**

## Algorithm for $\sin(x)$

- Get the value  $x$  in radians
- Initialize the value of sum (of terms)
- Calculate successive terms
- Add successive terms to the sum
- If the value of the sum does not change by more than say,  $10^{-10}$  by adding a new term, stop the addition of successive terms
- Print the value of the sum and the no. of terms



So, what I have done here, first step; get the value of  $x$  in radians, okay, so we want  $x$  in radians and not in degree that is how the sine function works, initialise the value of the sum, this is the sum of terms, I initialise the value to some value, let me call it sum, then calculate successive terms, okay and after you calculate the successive terms, add the successive terms to the sum, then how many times will you add, so this is where the key to the algorithm is.

In the value of the sum does not changed by more than let us say  $10^{-10}$  that is when the new term and the old term do not changed by  $10^{-10}$  that is it is a one digit in the 10th place, so then I stop adding successive terms, okay. So, once my terms becomes very, very small, smaller than some object, then I stop the calculation and print the value of the sum and the number of terms, this is my algorithm.

**(Refer Slide Time: 20:36)**

## Programme to calculate sin (x)

```
• C program sine x
• write(*,*) 'input the value of theta in deg'
• x = theta * 3.14159265 / 180.0
• sum = 0.0
• term = x
• denom = 1.0
• nterm = 1
• 10 sum = sum + term
• denom = denom + 2.0
• term = - term * x * x / (denom * (denom-1))
• nterm = nterm + 1
• if ( abs (term) .gt. 10**(-10) ) go to 10
• write(*,*) 'nterm, sum, fortran sine fn
• ftsnfu = sin (x)
• write(*,*) nterm, sum, ftsnfu
• stop
• end
```



So that is what now, is done in this program, so now this is a program to calculate sin x, so again we will use the same do loop, so in this case we cannot use a do loop because we do not know how many terms we have to add, so we have to use a slightly different trick, so that is an if statement, so if statement allows you to go from one point to another based on a condition, so let us again go linearly step by step.

First line is comment program sine x, okay this really is a program to calculate sine x, now I will write something on the screen remember, I told you whatever is written between 2 quotation marks, it is written on the screen, write \*, \* quotation mark input the value of theta in degrees, again quotation marks, so whatever is in this, so when you execute the program, the first thing you will see is input the value of theta in degrees, so that you know how to give a value of theta.

So, once that theta is given, okay, so I am saying next line should be which is missing here; read, \*, \* theta, so after this write, the next line should be read \*, \* theta and it is only when you read that theta, then I will convert x into radians,  $x = \text{theta} * 3.1415265$ , this is the value of pi divided by 180, so I have got the value of x in radians by multiplying theta/pi and dividing by 180. Remember, between this write statement and this x, there is a line; read \*, \* theta that was missing here.

So, in a way sometimes, it is good to miss something, so that you will start questioning, okay, so whenever you hear many things you have to practice and do it, computer programming cannot be learnt just by listening or hearing or seeing, you have to execute it and you will execute in after 1 or 2 days, you will execute all these programs and show you on the screen, so that you know that what we are saying actually happens on the computer terminal.

So, initialise the value of  $x$  in radians. I start  $\text{sum} = 0$ , so this is my first, I have initiated  $\text{sum} = 0$ ,  $\text{term} = x$  and  $\text{denominator} = 1$ , remember the first term was  $\sin x = x$ , okay and  $n\text{term} = 1$ , so this is the first term. So what is the new value of the sum now?  $\text{Sum} = \text{sum} + \text{term}$ ;  $\text{term}$  was  $x$  and  $\text{sum}$  was 0, so in this line 10, this 10 is a line number, remember all these lines from first till last, they all start in the 7th column that was one of our key ingredients of programs.

Only line numbers are written to the left of the 6th column, first 5 columns I can use for line number and the first column would be for a comment card, this was the comment card, now it is a line number, so what have I done here; I have said  $\text{sum} = \text{sum} + \text{term}$ , the first sum will be same as  $x$ , now see the next line,  $\text{denominator} = \text{denominator} + 2$ , okay, so earlier my denominator was 1, now the denominator has become 3.

So, the next term I am calculating now, the next term will be  $-\text{term}$  multiplied by  $x * x$  divided by  $\text{denominator} * \text{denominator} - 1$ , so what was denominator here; denominator was 3, so what this is  $3 * 2$ , you remember that the first term was denominator was 1 factorial, the third term it is 3 factorial, so by doing this  $\text{term} = - \text{term} * x \text{ square}$  by this, by; I have got the second term that is  $-x \text{ cubed} / 3 \text{ factorial}$ , this was my second term, okay.

Then, so since I calculated one more term, the number of terms has increased by 1, so I will say  $n \text{ term} = n \text{ terms} + 1$ , so I have got 2 terms now and my this term is  $-x \text{ cube} / 3 \text{ factorial}$ . Now, what I ask; I ask the question is the value of the term  $< 10^{-10}$ , if the value is much smaller, then I go to line 10, if it is not smaller, if it is large, okay, then if it is greater, then I go this line 10, if it is less my calculation is over, okay.



So, this is an, if statement and if statement allows you to branch, what is the meaning of branch? If some condition is satisfied, you go to line 10, if it is not satisfied you go to the next line, so if statement is like your traffic signal, if something is red, you do not go, if something is green, you go, so green signal here means go to the next line, so see this line again, if absolute value of term .gt; .gt; is the comparison.

So, it says is the absolute value of term  $> 10^{-10}$ , if it is greater that needs I have not converged, so I should go back to 10, so I go back to 10,  $\text{sum} = \text{sum} + \text{term}$ , what did I do: add x and it is  $-x^2/3$  factorial. The next i row, denominator = denominator + 2, so denominator was 3 last time, now it has become 5 because I have added 2 to 3, then now term, this is the third term now, is the second term multiply by x square.

So, this term has already x cube, now it is x to the 5th power and divided by denominator which was already 3 factorial okay, now these denominator is 5;  $5 * 4$ , okay, the 3 factorial is already in this term of the last one, now I multiply by 4 and 5, so what I have is x to the  $5 / 5$  factorial, again increase the number by 1, again compare whether the term is  $< 10^{-10}$ , if it is  $< 10^{-10}$ , you stop, if it is more than  $10^{-10}$ , go back to 10, again calculate.

Now, you have calculated 3 terms, what is the fourth term? In the fourth term, the denominator is be 7; 7 would be now term is; this term is x to the  $5/5$  factorial, put a  $-$  sign, then x square, so it is  $-x$  to the seventh, there is already a 5 factorial here multiplied by 7 and 6, so I got the new term;  $-x$  to the  $7 / 7$  factorial. So, I keep on repeating and now the term will be 4, so this way I have calculated 4 terms.

Again ask the question; is the term  $> 10^{-10}$ , if it is  $> 10^{-10}$ , go to line number 10, if not come to the next one, so suppose that absolute value the term is  $< 10^{-10}$  that means, I have calculated enough terms, so that I do not have to calculate anymore, so then I write \*, \* again on the screen, n term that is the number of terms, sum fortran sin function, it will write this on the screen that I have to have a quotation mark here, purposely missing, so that you will write the quotation mark.

So, `ftsinu`, this is a fortran sin function =  $\sin x$ , so what I will do now; I will write n term sum and fortran sin function, so next time I will explain you what this fortran sin function is because in the present program, we calculated our own sum, which is with our program, fortran also has its own functions, it has a library, this  $\sin x$  is a fortran function, what we calculated sum is our sin function, so we will calculate our sin function with the fortran sin function.

And it allows us to learn whether our program is good or not, so what we have done today is that we wrote another program which was for a sin function gave the algorithm for it and in this algorithm, each new term is calculated in terms of the old, we added the terms and if the function was less, if the last was less than sum number, we stop the calculation. As an assignment here, I have used absolute of term that is absolute value of term.

I did not used just the term, I use the absolute value of term, so you think about it, why I have us the absolute value, so that next time when you come, we will start with this again, okay. So, I hope you have a learnt a little bit now what an algorithm is, what a program is, how to write a sequence of steps, what are line numbers, where you start the program lines from the seventh column and this theme will be using throughout in our course, okay. I will stop here and continue with other programs in the next class. Thank you.