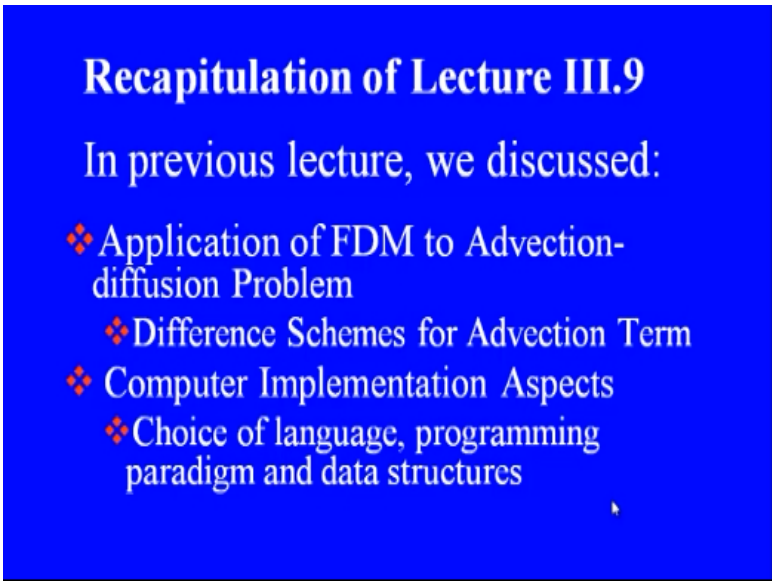
Computational Fluid Dynamics
Dr. Krishna. M. Singh
Department of Mechanical and Industrial Engineering
Indian Institute of Technology – Roorkee

Lecture - 19
Computer Implementation of FDM for Steady State Heat Diffusion Problems

Welcome back again to finite difference method. In this module, we are now left with applications of difference method and we would focus particularly on computer implementation of finite difference method for solution of one dimensional heat conduction and this should enable you to write finite difference based CFD code for solving high dimensional problems as well.

(Refer Slide Time: 01:02)

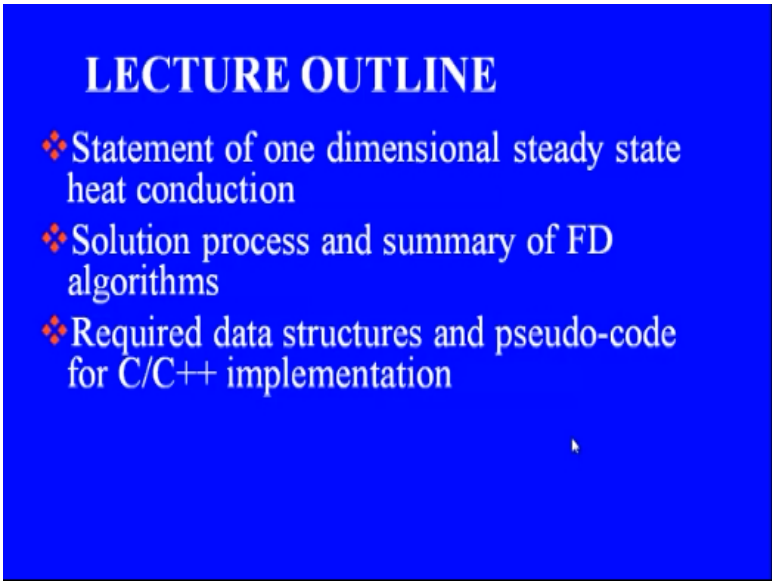


So let us have a recap what we did in the previous lecture. We discussed application of FDM to advection diffusion problem with an outline of different difference schemes for advection term which can take care of the oscillatory behavior and then we discussed some general aspects of computer implementation of CFD algorithms. In particular, we discuss the choice of languages, programming paradigm and data structures which are particularly suited to CFD programming.

Now in this lecture we would take up one simple application that is computer implementation of finite difference method for steady state heat conduction problem. So we will systematically proceed from our finite difference algorithm to pseudo code to implementation in C and C++ for a simple problem. So the stepwise procedure should enable you to use the

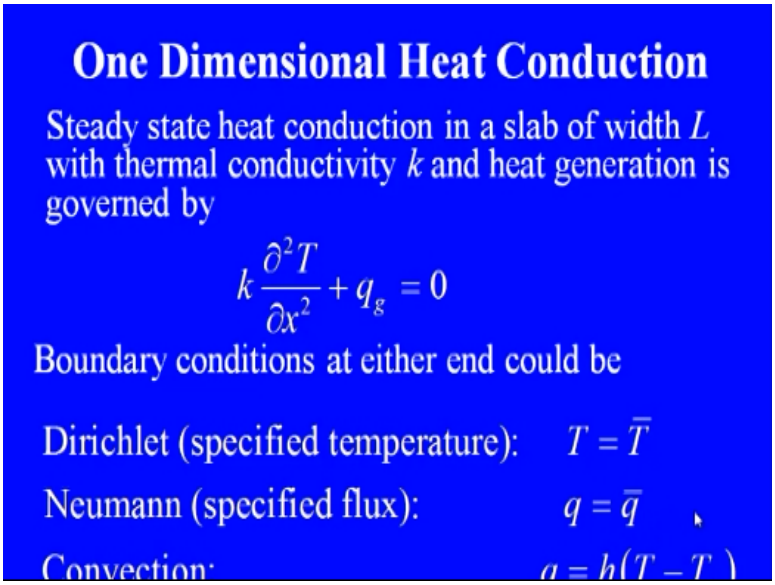
same approach for writing your own code for more complex problems.

(Refer Slide Time: 02:15)



So outline of lecture we will just have a relook at the statement of one dimensional heat conduction equation and then we would summarize or recap of the solution process and the summary of finite difference algorithm then we will discuss required data structures which we need for this particular problem and the pseudo code for C, C++ implementation and if time permits we should also start off with actual program code.

(Refer Slide Time: 02:45)

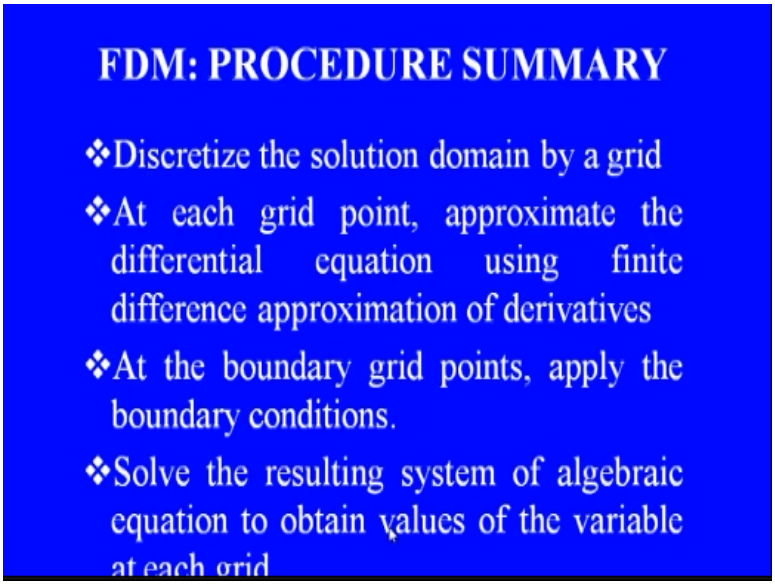


Now let us have a relook at bit general statement of steady state heat conduction in a slab of width L with thermal conductivity K which is assumed to be constant and heat generation. So this is governed by differential equation K d2 T/dx square +Qg=0. Now this Qg could be depended on temperature. Now boundary conditions either end specifically when we are

writing a program we would like to make more general and we should apply or we should provide the possibilities of different types of boundary conditions at either end.

So at either end we could have Dirichlet boundary conditions that is specified temperature T=T bar where T bar would be some specified value. We could also called Neumann boundary conditions which simply mean the flux specification q=q bar where this Q is K times gradient of T and we can have conductive boundary conditions Q=hT-Ta or this Q is –K times gradient of T from Fourier's law.

(Refer Slide Time: 04:00)



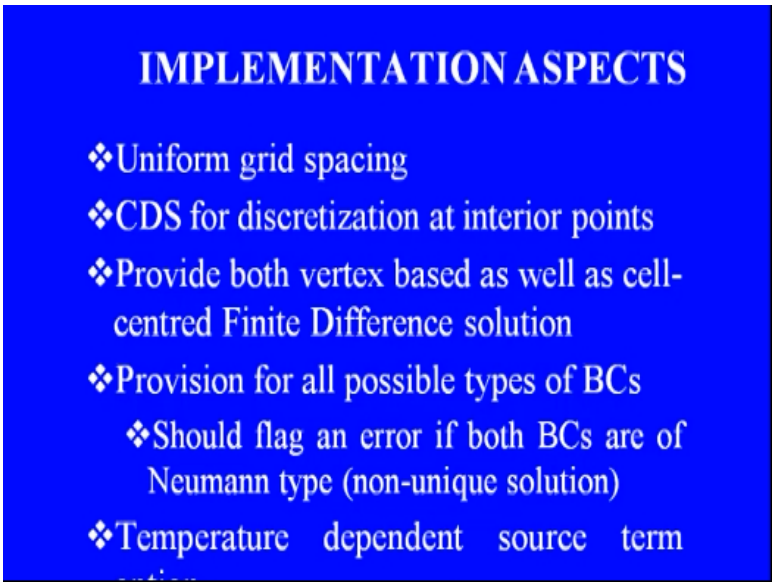
Let us have a review of this FDM procedures which we discussed right in the beginning. We need to describe the domain by a grid that would be your first step. Same thing we should translate into your computer implementation as well and at each grid point we have to approximate our differential equation using finite difference approximation of derivatives. So this has to be translated into a code segment which gives us the appropriate matrix elements and right hand side vector coefficient.

Then once we have generated the equations at all the interior points we need to apply our boundary conditions at the boundary points and apply the boundary conditions wherein we have to modify the matrix coefficient for boundary nodes taking care of this specified boundary conditions and then once you have done that and we have got our algebraic system of equation solver system of equations.

So basically this step-by-step procedure tells us modeler decomposition which should make

of our code like each step which we have outline here could be made a separate function routed or module whichever term you prefer.

(Refer Slide Time: 05:20)

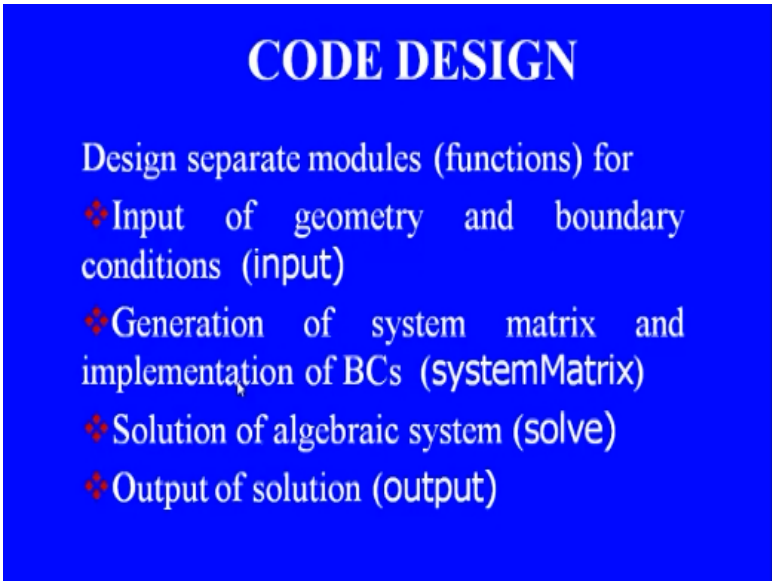


And now in our implementation that example which we are taking up in the class we will consider a uniform grid spacing extension to non uniform grid spacing should we fail a trivial and would be left as extension exercise to you and we will focus on central difference scheme for discretization of governing we have got a second hour derivative. So use CDS for discretization at interior points and we would now like to provide two options in our code.

The one which is based on normal finite difference method where the vertexes of the cells finite difference cells they are our computational nodes or the other one which we discuss that instead of the vertexes we will consider cell centered as the finite difference grid points. So we will provide both the options in our code. And as I mention earlier we want to write a code wherein we can have a difference mix of boundary conditions.

So we have to provide a provision for all possible types primarily 3 types which we would have discussed in our problem specification and please remember our heat conduction equation it should flag of an error if both BCs of Neumann type because in this case we might end in non unique solution. And we should also provide for an option of temperature dependent source term.

(Refer Slide Time: 06:50)

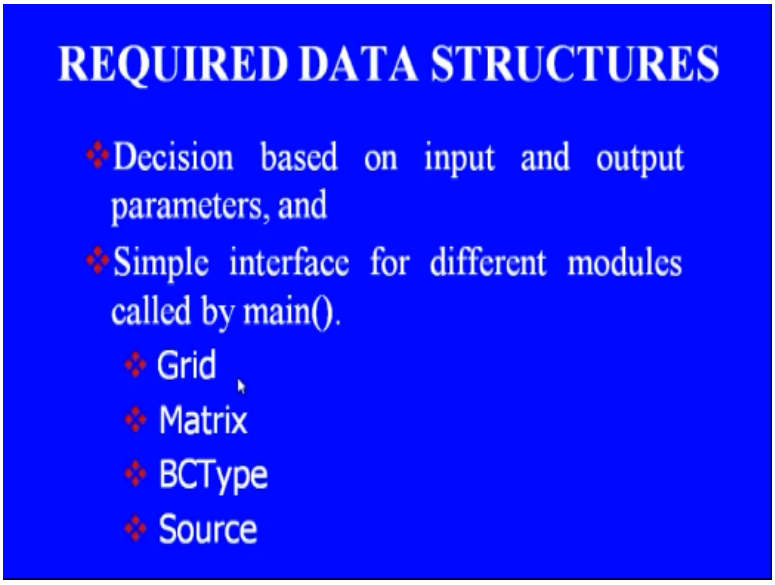


So let us say that how do we proceed for designing our code. So finite difference code for solving this problem. So it contains one module for input of the geometry and boundary conditions. First call that we are going to call this module simply as input. You can choose any of those appropriate names which you might find and see. Similarly, we need 800 module which will generate the system matrix and the right hand side taking care of the implementation of boundary conditions. So let us call this module a system matrix.

This module may itself invoke separate modules for implementation of boundary conditions for the two types of finite difference approximation or approaches which we had from which that we are going to implement in the code. And then we should have one module which should solve our tridiagonal system of algebraic equations and then we should also have a separate module which will output the solution in a desired format.

Now before we can design our modules we should also decide what sort of data structures we are going to use.

(Refer Slide Time: 08:10)

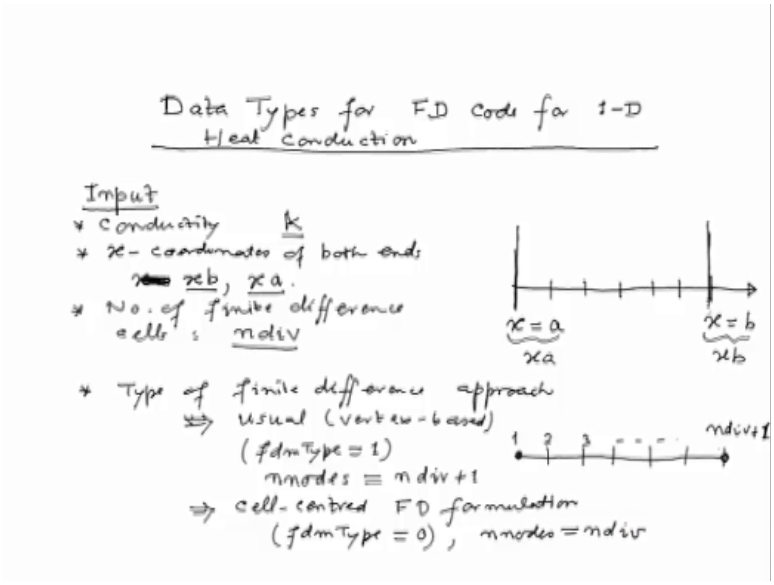


Now the decision of data structures were based on what are the inputs and output parameters which this code is required to handle. That is what input we should provide to solve our problems and what output do we expect from our code and the second important consideration is that we would like to have a simple interface for different modules when it is called by the driver function which in C or C++ is called a main function.

In Fortran you might call it main program. So suppose these are our grid now what sort of different data structures you might have. We can define one data structure for a grid related things that is what are the end points of our domain number of grid points we would like to have number of computational nodes we would have and so on. Let us divide one simple data structure called grid.

And similarly for handling of system matrix which is essentially a tridiagonal matrix. We can define a new data type called matrix. Boundary conditions we could have different types or different mix of boundary conditions. So let us define one data structure which we would call BC type and we want our source term to be temperature dependent it could be of linear function let say of temperature. So we should specify or we would provide the specification by a new data type let call this as source data type.

(Refer Slide Time: 09:50)



Now let us have a look at each one of these data types that what will these data types contain in detail. So data types for finite difference code for 1-D heat conduction. So what are the input data which we had the end points of the grid let us call it X= A and X=B the number of division which we might have so that becomes one more input. So what are our inputs let list them. And then we are going to put these inputs in appropriate data types. So our inputs or conductivity. Let us use the same symbol in our code as well.

We will call it K. X coordinates of both ends. Like this 1-D heat conduction we can think of as if we are dealing with heat conduction across the slab one end as given by the X coordinate X=A and other end is given by X coordinate X=B or in code just use this prefix X=XB for the right end and XA for the left end. For this we will call XA this end we would use the symbol XB.

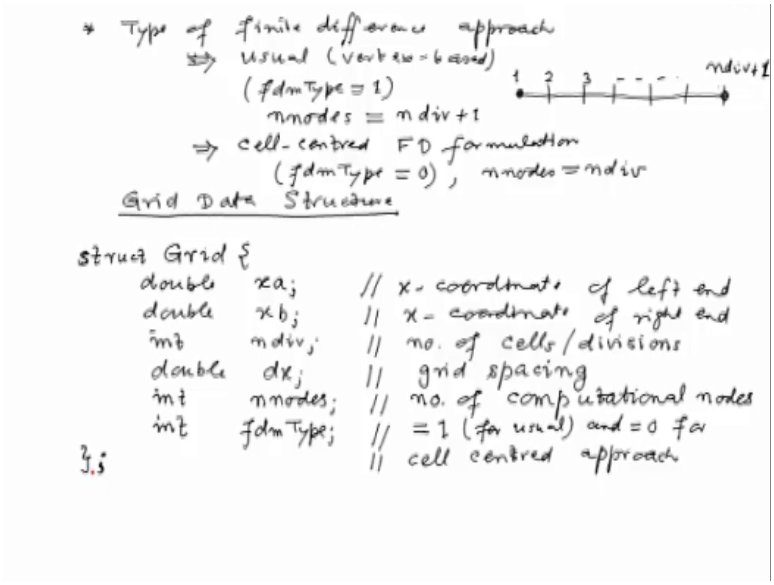
Number of divisions or number of finite difference cells so let us use a variable N div. So these are to be specified at geometric and metal property inputs what else we would need in terms of geometry. Another thing which we need to specify for our grid specification as a type of finite difference approach that is we are dealing with usual vertex based where in we will have our if there are N div number of divisions.

The total number of nodes will be from 1, 2, 3 and so on N div+1. So we can denote this by 1 parameter let us call it FDM type. FDM type=1 and in this case number of finite difference nodes N nodes=N div+1 or the second approach could be our cell centered finite difference formulation. Now this we can flag of by this parameter FDM type=0 and in this case N

nodes=N div.

I have particularly chosen this FDM type=1 and 0 with a specific reason so that we can set in our code that N nodes=N div+ FDM type.

(Refer Slide Time: 15:08)



Now we are ready to formulize our grid data structure. Let us call grid itself starting with capital G as usual convention which we follow in C or C++ that whenever we introduce a new type the name is started with the capital letter. We can use the key word class or struct let use struct type structure grid. Now what are elements of the grid? First the 2 end points coordinates so double XA this is X coordinate of left end.

Double Xb this is X coordinates right end. Next we should put in number of divisions that is definitely an integer. So N div number of cells or divisions. We will assume a uniform grid. So we will have a single parameter which denotes our grid spacing DX which we will use very often. So we can introduce a new variable of the same name. So double DX this gives us the grid spacing which other input data we should put as part of grid.

The number of nodes is also related to our grid types. So let us put the number of computational nodes. Let call that int as N nodes. So this is number of computational nodes. The finite difference approach should also be put as a part of this. That is again integer parameter int FDM type. Now this is=1 for usual and=0 for cell centred approach. Do you think we are missing anything else any other grid related parameter which we should put as a part of this data type.

So we have already taken care of the coordinates the number of divisions, number of nodes Dx that is grid spacing total number of computational nodes and the finite difference types so I think that is where we should close it. So this is an encapsulation of all information which is related to finite difference grid for this one dimensional problem. Do you have put this template is fairly generic in the sense in the sense if we had a 2-d grid on a rectangle.

We can easily extend this template by adding the suitable dimension, the suitable coordinates like double YA and YB to denote the extents similarly let say int Y div which will give us number of divisions in Y direction double Dy which will gives us the mesh spacing in Y direction and so on. So this should help you writing further program for a multi dimensional problem.

(Refer Slide Time: 20:22)

Next data structure which we should take up is that of boundary condition type. So let us move on to the fresh page. Now we anticipate 3 types of boundary conditions and we should provide separate integer codes for all the 3. So our 3c conditions we have primarily 3 types Dirichlet and we should use integer code for it. Suppose you use code 0 for Dirichlet boundary conditions. And what data we need specify temperature let us call that or denote that by TB.

The second one is Neumann boundary condition that is specified flux. Let use an integer code 1 for it and the specified data is basically the heat flux. Third case is the convective boundary condition let use an integer code for it as 2 and what all data we need for it. We need that

convective heat transfer coefficient H and ambient temperature TA. So the data structure which we define for BC type should have all these data which we need for any of these boundary conditions.

So that is we should have one integer flag 1, 2, 3, 4 these 4 double type variables. Struct BC or let us make it bit more longer name BC type int type which simply gives us the code for a BC specified double TB which is specified temperature double QB represents specified heat flux double H which stands for convective heat transfer coefficient double TA ambient temperature which is required in specification of convective boundary conditions.

There is a purpose in writing these names and these comments which I have written separately. In our actual code also we will type exactly in the same fashion so that anyone any user who wants to have a look at our code knows that what each symbol stands for and they can easily correlate with their problem statement. So it is always a good idea to provide ample comments whether it is type specifications or in our modules or functions to make work code understand to a human programmer.

And it will facilitate the maintenance, debugging and extension of our code. Okay we have handled the grid, we have handled boundary conditions. The next thing the next issue which we had is the handling of the discrete finite difference system and there we can have 2 choices that is to say we have got all the matrix and let us keep all of them together. So the next thing which we want to handle is our discrete finite difference algebraic system.

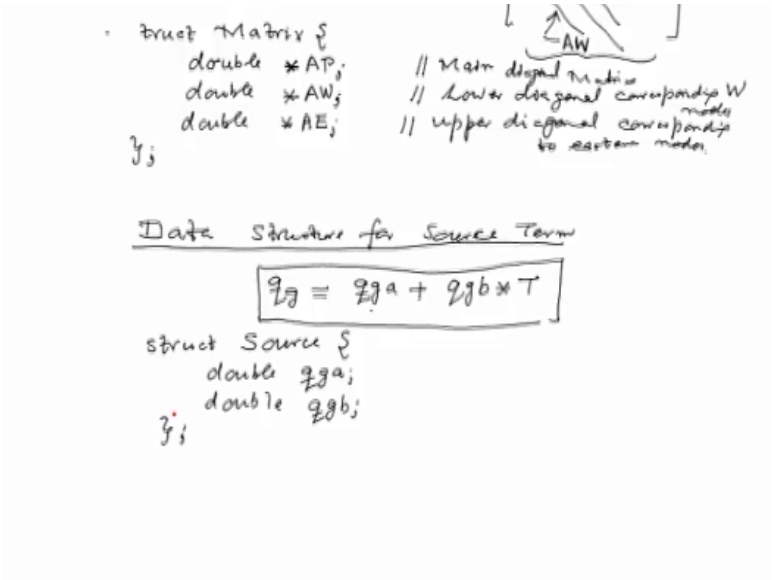
So it was given by A*T where T stands for the temperature denotes=QB which is what right hand side. Now remember this A was given for 1-D problem this A has got a tridiagonal structure. There are only 3 diagonals in A which are non zero and this main diagonal we had use the symbol Ap. For upper diagonal which corresponds to denote I+1 we have used the symbol Ae and for the lower diagonal where lower diagonal stands for matrix A.

This stands for the collection of the coefficient which we have used symbol Aw. So let us use this same symbols in designing a data structure for representation of all the entries in our finite difference algebraic system and let us call that is a matrix. So basically we need to provide a data structure which encapsulates Ap, Ae, Aw, T and Qb all of them together. Here we can make A time the choice if you want to.

We can keep Ap and Ae and Aw this which form this 2 dimensional matrix such a part of one data structure. We can leave T and Qb which are simple vectors alone. So we can decide on either choice and other choice is equally good. So now let us opt for the second choice. We will call it system matrix only which will be stored in that matrix or which will be store the entries of our discrete algebraic system A.

T and Qb would be left as a standalone vectors, but if you do that we will have to make sure that we pass these separately. So we must have an extra argument which should be supplied to our relevant modules. So struct matrix. Now we do not know how many number of nodes user smart is specified. So we would like to provide the user with a flexibility of providing any number of grid points when a solution process starts. So we are not going to have any fixed arrays in our designs.

(Refer Slide Term: 30:02)



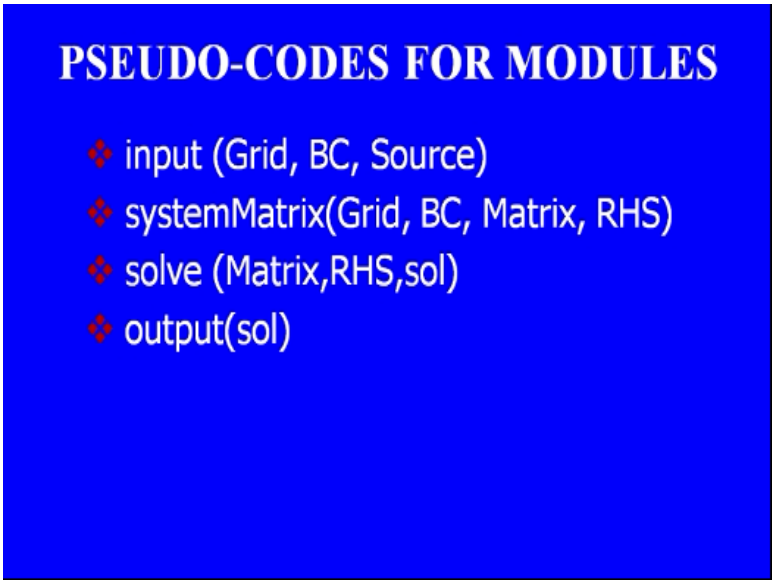
So let us use then dynamically allocated arrays which would be represented by pointer in the struct matrix double **Ap. This is our main diagonal so this is your main diagonal **Aw lower diagonal corresponding to western node or i-1th node and **Ae which corresponds to the eastern neighbour corresponding to eastern nodes. For T and B we will have simple one dimensional arrays those also will be dynamically errors.

Next thing we have left the source term. We said source would be slightly more generic than it would not be just a constant thing. So let us introduce one more data structure for handling the source term. Source was given by Qg. Let say that we use 2 symbols we call it QGA+

QGB*T. That is the source is linearly depended on temperature So the data structure which we are going to provide this should have this two double entries QGA and QGB.

So struct source double QGA double QGB. Okay those are few who might be looking at this code pieces for the first time and may not be very well conversant with C or C++. This double basically represents a double precision or what we call double precision real number and intestines for integers. So now we have got most of data structures designed which we need for our code implementation.

(Refer Slide Time: 33:35)



Now we should go for the pseudo codes for modules. So whenever we want to write a program the very first step is to write a pseudo code. Pseudo code may be what we call simple description in everyday English language which can be easily translated into our computer code lines is what we call meanwhile pseudo code. And this should be done before we go to type in our code in a development environment or in an editor.

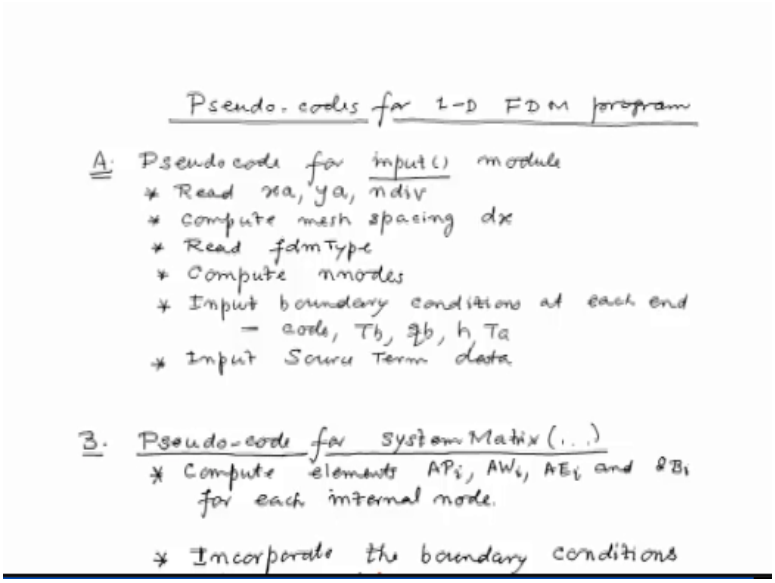
So we should first decide what different algorithms we are going to use for, write down those algorithms decide on the data structures and then write pseudo codes for each modules then translate that pseudo code for that module in a code, test it and then combine all the modules together to get our final code. So now these are all modules which have decided we are going to put in.

The first module we call it input which will read the information or input data for grid boundary conditions and sources and the second one which we said was our system matrix to

which we would provide our grid data BC type matrix and right hand side vector and the third module which we had was solve to which we provide our matrix right hand side vector and then the solution vector so it will return the solution in our solve vector.

And the fourth module was our output module. We will provide solve as input parameter and to print our solution in a specified format. So now let us have a look at the pseudo code for each of these modules before we proceed for actual typing this code in a development environment or into editor.

(Refer Slide Time: 35:50)



So first we will take up the case of this pseudo code for1-D FDM program. First we will take up the case of the pseudo code for input module. So what is input module supposed to do. It is supposed to ask the user to input all the required data. So first think and our code should prompt users to input certain set of data this is what we will write here read Xa, YA and number of divisions.

This should be the first input which this module should ask from the reader and based on this input it should compute mesh spacing Dx. Then it should read you should ask the user or prompt the user to input the finite difference type. So read FDM type and once we have got this FDM type next thing is compute number of computational node. And nodes what else we need input module.

So this is the input and the corresponding competition which give us basically basic geometry. Next we have to ask the user input boundary conditions at each end and what sort

of data we expect. We expect the user to input a code Tb Qb, H and Ta. The data which is not required for a given boundary condition should be just in that place the user should simply input 0. So we should prompt that information, but suppose we have our code is 0 we need only Tb data remainder Qb, H and Ta for that user should put simply input 0 or if you want we can also built that information in our code that which ones to read.

So we can prompt user to only input Tb or Qb or H and Ta. So that decision we can make because this simple refinement to our code which we can incorporate while coding our input module. The next set of input which we need is that related to the source term. So input source term data. So that completes our pseudo code for input module. Now each one of the task which we listed that is what in my pseudo code that write in plain English which each statement or a set of statement was supposed to do.

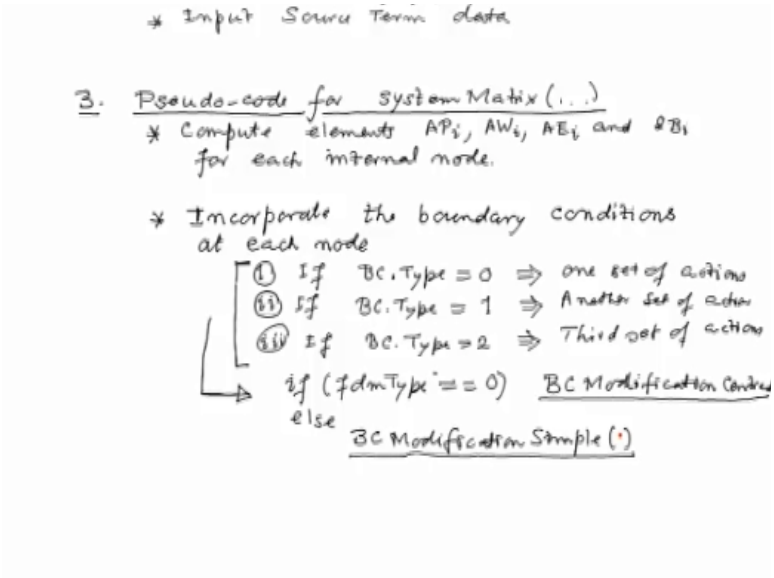
So read Xa, Ya, N div and this can be translated into a single line of the code. Similarly this compute spacing of TX is again get translated into single line of code read FDM type a single line of code and so on. Whereas these ones input boundary conditions at each node this might lead to a block of the code. There might be a few lines of code here. The same holds good for input source term data, but nevertheless when we hold these refinements if there is something which requires many more a set of action to be performed that can be refined further.

That can be extended further before we translate that into a code. So our pseudo case in that case would look a bit longer which we will see in our next module. Okay so next module is our system matrix. Pseudo code for module system matrix. Okay what are the main task which we need here we got all the geometric information, the source term information and that information has to be supplied in this parameter list to this modules system matrix.

First task would be compute elements API, AWI, AEI and QBI for each internal node. So that maybe basically formed just a particular line or particular rho of our algebraic system. Next incorporate the boundary conditions at each node. Now here this single statement has to be refined further because we may have different set of boundary conditions and at different modifications would occur in each case.

So the details has to be further expanded.

(Refer Slide Time: 44:01)

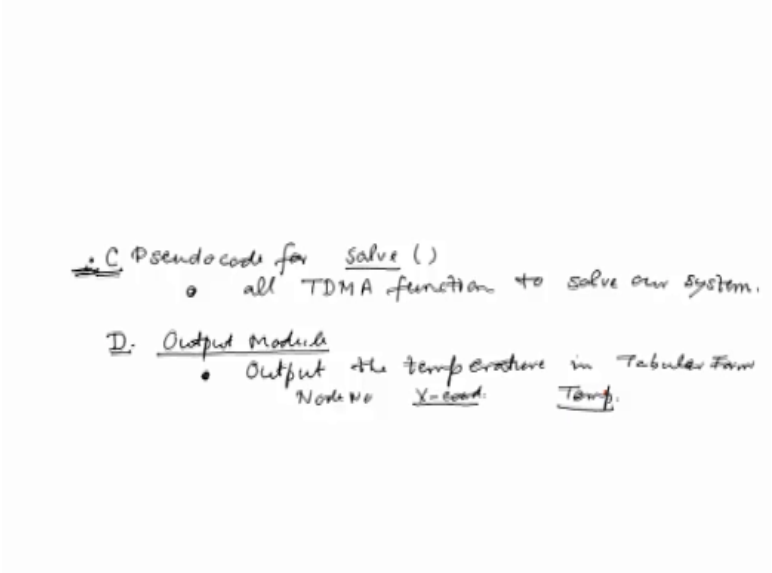


So that we will take each of these nodes and we will have 3 distinct set of modifications. One set that if BC type. Type=0 you got one set of actions or modifications. These we are going to elaborate later in bit more detail so that we can translate each of these into actual code lines or BC dot type=1 another set of actions and third BC type=2 third set of actions. Now please remember that incorporation of these boundary condition would also depend on of finite difference approach which we had chosen.

So what we should do is transfer these incorporations in 2 different sub modules which corresponds to 2 different finite difference methodology system. So basically we are going to put these modifications in separate module if FDM type=0 which stands for self centered formulation called BC modification center. So this is a new module. We will look at its detailed pseudo code in the next lecture else that is our FDM type=1.

So we will have DC modification simple. So we will have a detailed look at the pseudo code for these 2 sub modules later on.

(Refer Slide Time: 47:00)



So for the time being let us complete the sequence. The next one which we had was solved pseudo code for solve module. We would basically call an external function we have not yet learnt the solution of algebraic equations so we will assume that we have already got from somewhere algorithm or sub routine based on tridiagonal matrix algorithm. So this all will simply have one line in a pseudo code called TDMA function to solve our system and D we have got our output modules the pseudo code for it is again very simple one line.

Output the temperature in tabular form. Specifically, we would like our program to output node version information that is node number X coordinate and temperature. So we would like to have this tabular form. Okay so in this lecture we would stop at this point. In the next lecture, we will have a detailed enumeration of the sub modules of system matrix the pseudo code for that.

And we will have look at full code translated version of these pseudo code in actual C or C++ code and we will try and solve two example problems live in the class.