Programming and Data Structure

Dr. P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture # 01

Introduction

Good morning everybody. Today we will begin our first lecture in the study of the course in title programming methodology and data structures. You will observe from the title that there are two aspects, one is called programming methodology and the other is data structures but these two issues are quite integrated in nature and they are go hand to hand in solving any problem for which a computer program is required to be developed. Now the core issue in all these aspects which require both programming methodology and data structuring lies at one point. The first point is that you are given a problem and given this problem, you have to solve it efficiently.

So the issue at hand in most tasks of computer science is the issue of problems solving. Unless we understand how to solve problems on a computer quite efficiently and what are the steps to do it, it will be very difficult to understand what programming methodology and data structuring means because these are two derived items of the system of problem solving. Now the first thing that we have to remember about problem solving is that we are solving the problem on a computer and the computer has got a particular structure, a particular architecture and we have to remember that we are going to solve problems on that issue.

On the other hand, problem solving is not merely a process of programming. It is a process of developing the solution to a problem which is expressed in a natural form. For example when somebody gives a problem, find the maximum of n numbers. This problem is posed in a natural language. It is our duty and tasks to convert this problem posed in a natural language to a computer which will solve the problem. If we had technology which could understand natural language itself then we would not require this series of tasks of programming or the concept of programming methodology and data structures would have been automated already. So since we are having a programming language or we are having a computer which can understand only a restricted form of language and the problem is posed in a natural language, it is our task to convert this problem into the language of the computers, so that it works efficiently.

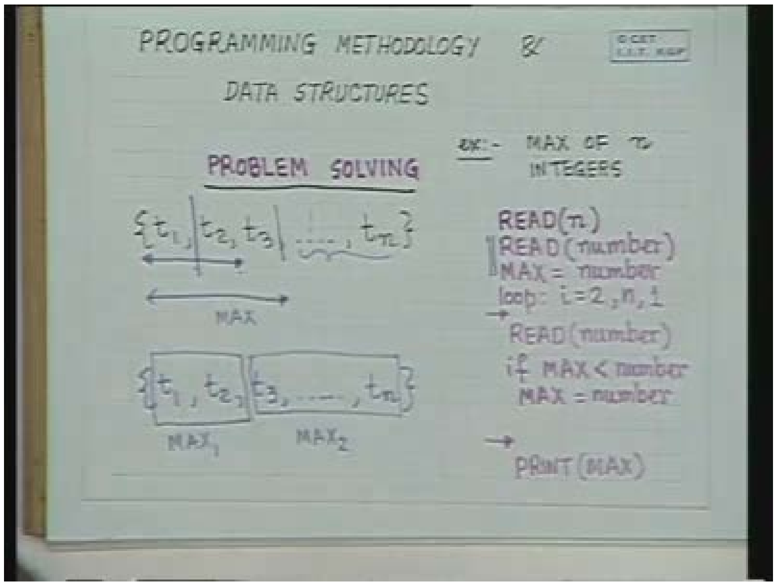
So this is the core and we shall call it problem solving. So our task is to understand the technique of problem solving. The technique of problem solving is not very easy to understand, it's not a very simple task and it requires a lot of issues other than peer programming. The first issue which and the first aspect about problem solving is that when a problem is expressed in an English language, find the maximum of n numbers. There are no steps which are told that this is the steps that you have to follow to find the maximum of n numbers. Then why does the question of steps comes in. The question of step comes in because of the architecture of the computer itself. The computer is unable to understand our visual expression, you cannot give a diagonal of n numbers to a computer like you can give it to a human being, who will just observe it and give you the answer. So the question of steps is an inherent issue which is related to the architecture of the computer.

Therefore the first aspects of problem solving is to convert it to a sequence of steps. And how do you convert it to a sequence of steps is something that we do ourselves. So in other places you will read terms like algorithm design, terms like programming methodology, terms like data structuring. These are all related to converting the problem which is given to you to the sequence of steps which can be executed in the computer. It is very difficult to define and understand and present an automated way of saying that this is the theory of problem solving. There is no well defined theory of problem solving till date in which you will express the problem in a given language like English or any other natural language and the problem will automatically be solved on the computer. Partial solution to such automated problems solving has been achieved in artificial intelligence. But it’s a long way to go before we understand, what actually automated problem solving is because it is possibly one of the tasks and one of the goals of computer science itself to understand what automated problem solving is.

Now problem solving as we understand it is a mixture of techniques, intuition and other aspects which have form a body of knowledge. Now this body of knowledge has to be used and understood in order to really solve individual problems at hand. Our task in this course will be to study this body of knowledge through various forms so that we are able to use it and develop new techniques in future. Why it is difficult to express what automated problem solving is. There are several ways in which we can try and impart this knowledge to others and that is what I will be trying to do in this whole lecture.

Now let us take an example. As I said before the example that we will take is maximum of n integers. Now all of us would, initially those who got programming experience what immediately come up with a solution which looks like this. you will say READ (n) and then you will read the n numbers and in an iterative loop, you would find the maximum and that’s what you do in something like this, read the first number and then initialize the maximum to this number and then in a loop with an index i equal to 2 to n in steps of one. What you would do is you would read the next number and then you would compare, if this number is greater than the maximum which is temporarily stored here, you would update the maximum to this number.

(Refer Slide Time 13:26 min)



PROGRAMMING METHODOLOGY & DATA STRUCTURES

C.CET I.I.T. KGP

PROBLEM SOLVING

ex:- MAX OF n INTEGERS

```
READ(n)
READ(number)
MAX = number
loop: i=2,n,1
    READ(number)
    if MAX < number
        MAX = number
PRINT(MAX)
```

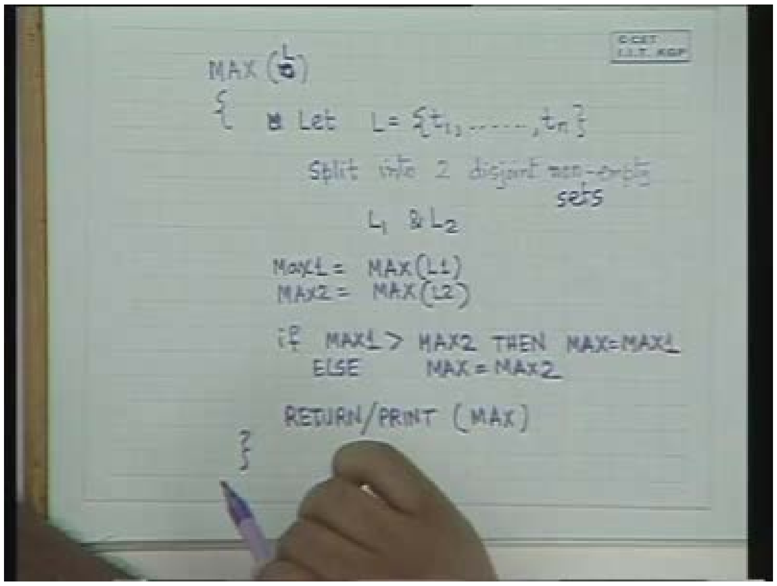
$\{t_1, t_2, t_3, \ldots, t_n\}$
MAX
$\{t_1, t_2, t_3, \ldots, t_n\}$
MAX_1
MAX_2

This is what most of us who have written some programs in any language before would initially do. That is you would say that if MAX is less than number, this loops starts from here and continues to the certain point. And if max is less than number, you would say MAX is equal to number and this loop would continue from 2 to n because the first number has already been read and at the end of the loop, you would just print the value of max. But this is not what I mean by problem solving. this is the solution expressed in a language which has got that is we have converted to a sequence of steps but given this problem, we have converted to this sequence of steps because all of us knew how to do it exactly. Given a new problem, how do you tackle it or given if this problem were given to a person for the first time, how would that person tackle. So for that we need to develop certain thinking skills, it's not a question of whether you know it. It is a question, it is a style of thinking. It is a style of thinking which we called problem decomposition and that's what I will try and explain by this example today and we will go on to more complicated example and see step by step how solutions to problems can be efficiently achieved.

For example let us have, let us take a deeper look at how we solved this problem. If we take it from a mathematical or a diagrammatic point of view you are given n numbers t_1, t_2, t_3 and so on t_n. And what was the problem solving approach here? The problem solving approach here was we first tried to solve the problem up to here and said that if this was the only problem that was to solved, this is the value of the maximum which we did here. And then once we have solved this problem, we try to solve the problem from beginning to here and then once we solve this problem, we try to solve the problem up to here. That is at any point of time, we have already solved the problem in this loop up to t_i that is at t_i we have a solution of the problem in the value MAX. And this is the sequence of steps that we have followed, we have partially solved the problem and then we continued to solve the problem more and more till we reach the end of the unsolved portions of the problem. This is the technique known as problem decomposition. This is the very simple problem and decomposition here is quite trivial in some sense but there are other aspects.

Once you understand that this problem has been decomposed in this way, you will immediately stop thinking that this problem could have been decomposed in other ways as well. How? For example, what I could have done was I could have solved this problem and obtain a value called MAX_1. I could have solved this problem independently and obtained a value called MAX_2 and the solution to the whole problem would be the larger of the two. So rather than decompose it step by step like this, we could have decomposed it in any other way and we would have got the solution to the problem. That is we could have said that to solve the maximum of n numbers, what do you do is you form initially let L be all the n numbers t_1 to t_n. You break it up into two steps, split into two disjoint non-empty sets L_1 and L_2. These two sets would be disjoint and non-empty, each would be non-empty and it would be disjoint. There is no point keeping common elements because you will be wasting your time, trying to solve the common parts. You could have done, that is also a part of decomposition.

(Refer Slide Time 16:07 min)



```
MAX (L)
{  Let  L = {t_1, ....., t_n}
        split into 2 disjoint non-empty
                                 sets
            L_1 & L_2
     MAX1 =  MAX(L1)
     MAX2 =  MAX(L2)
     if  MAX1 > MAX2 THEN  MAX=MAX1
       ELSE        MAX = MAX2
     RETURN/PRINT (MAX)
}
```

C-CET
I.I.T. KGP

But here it is unnecessary at this point and useless to decompose it. And then you do MAX1 is sorry this is L, MAX1 is MAX of L_1. MAX2 is MAX of L_2 and then if MAX1 is greater than MAX2 then MAX is equal to MAX1 else MAX is equal to MAX2 and you can return or you can print the value of MAX. This is not a program but this is just an intuitive decomposition which says that, given a set of elements, split it up into any two parts, split it up into any two parts, solve these two sub problems and then from the solutions to the two sub problems obtain the solution to the original problem.

Now you will see that the program that we have written here is a special case of this general decomposition. So when you solve a problem, the first part which comes in problem solving is what I would term, these are the broad steps one of the guidelines which can be used in problem solving. There are no hard and fast rules, these guidelines can be broken to obtain better solutions in some problems but these are the general techniques which can be used to obtain solutions to a large number of problems.

The first is initial solution generation that is sorry, you have to try and solve the problem which has been posed to you in some form. And at this point in initial solution generation, there is something called inherent solution in a person's mind. That is you must be able to solve this problem in your own mind before you can really solve it on the computer. Now solving it in your own mind is something which is vague and in order to make it more concrete, we would say we need to decompose the problem in some way. Decomposition inherently gives you a sequence of steps but it gives you a very broad outline in that sequence of steps. Therefore the first thing is initial solution generation.

For example we could call such a solution as an initial solution that we have generated. Now given an initial solution, you will see that this initial solution actually encompasses a lot of possible final solutions. In this simple example, the possible final solution will differ depending on how you have split it. We will see other situations in which they will be more complex issues in developing a final solution. We shall come to them step by step and how to generate an initial solution there is no hard and fast rule. We shall learn it through experience and problems solving. And once we get what is called the hang of it, we will be able to handle possibly solved several problems in our own ways.

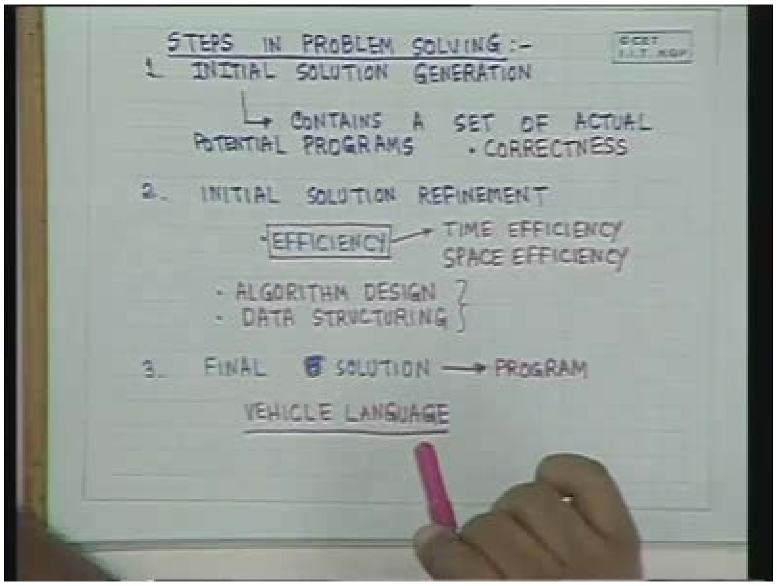
So getting an initial solution and this initial solution contains a set of actual programs. These are not programs, this we can call it potential programs. So this is the first step. The second step is called initial solution refinement that is given the initial solution, you would like to find out which final solution you would take and to decide that what are the conditions to decide which is the final initial solution that you take. The main criterion is efficiency. Here at the initial solution, the condition that we have taken care of and that is the prime condition in any problems solving is correctness. The initial solution must be correct; it must be able to generate a correct answer to a problem. Those solutions which do not generate correct answers are absolutely useless. Therefore the issue of correctness is tested and is validated at this step first. And whenever we do a refinement, we will try to maintain this correctness and the main issue in refinement is efficiency.

Efficiency means two things, one is time efficiency and the second is space efficiency. We have to try and make sure that I have programs run as fast as possible that was the need for computers to make things run fast and get them automated and space efficiency is required because space is at a premium. Today memory technology is such that we can have lot of memory but still there are problems where the space will be a premium. We will not have sufficient memory to solve the problem, if we want to use certain techniques and there will be a space time trade off that is there will be situation in which to reduce this space, you have to increase the time or to improve the time we may have to increase the space.

So given an initial solution, we have to choose from the set of potential programs, the final solution. And the criteria for choosing will be efficiency. This is where the steps of, the standard steps which we call programming methodology and data structures coming. At this point, if it does not coming directly here is an issue of problem decomposition which is something like say to give you an example, nobody can teach you how to solve problems in geometry. Somehow you solve them through experience of previous solutions, so through some theorems that you know and through something which we call intelligence which we do not understand properly.

Similar things are available here. You have to have an insight into the problem, we have to know several problem solution techniques and a combination of these with your own intelligence will be able to give you the initial problem. And two people may generate two totally different initial problems for a particular given problem at any time. But once you are given an initial problem, there are certain techniques which are now quite standard and well known.

(Refer Slide Time 28:22 min)



STEPS IN PROBLEM SOLVING :-

1. INITIAL SOLUTION GENERATION

$\hookrightarrow$ CONTAINS A SET OF ACTUAL POTENTIAL PROGRAMS

• CORRECTNESS

2. INITIAL SOLUTION REFINEMENT

- $\boxed{\text{EFFICIENCY}} \rightarrow$ TIME EFFICIENCY, SPACE EFFICIENCY

- ALGORITHM DESIGN
- DATA STRUCTURING

3. FINAL SOLUTION $\longrightarrow$ PROGRAM

VEHICLE LANGUAGE

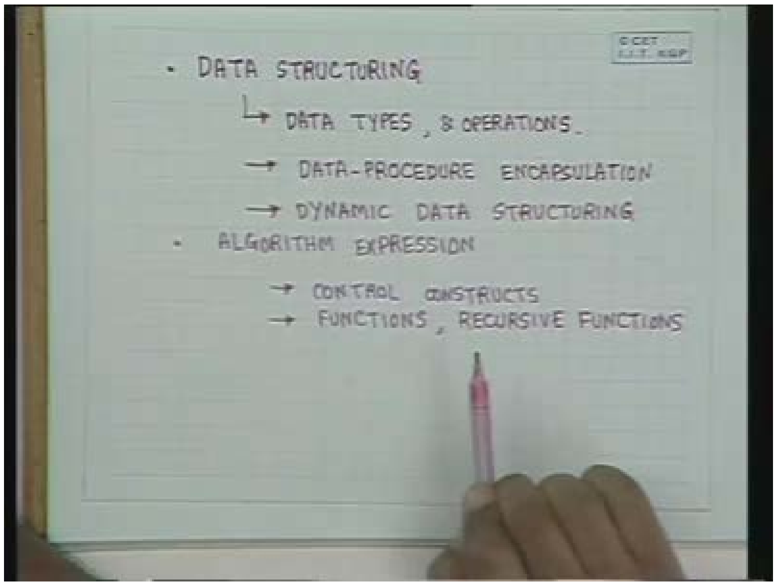
These form part of programming methodology and data structures. And here we come across the terms, the design of the algorithm that is the selection of the exact sequence of steps from the set of potential programs or solutions and the organization of your data, whether for example here we will look at it as a set. How will this set be stored, how will the set be organized? Once you have decided to split it into equal halves say then how will the two, this set be organized. As an array, as a list, as what? Will we not keep anything, will we just read it as and when required? So data structuring goes hand to hand with algorithm design to give you the final solution. So we have got the final solution which we term as the most efficient that we could design from this initial solution.

Now comes the step of programming. Given this final solution which we have expressed in some sequence of steps, in some language of our own, some people call it pseudo code, some people call it flow chart, whatever you call it. Now we are to write a program and when we come to write a program, we come across the vehicle language. So this has to be converted to a program. Now, given the situation, the circumstances that are provided to us we will be using vehicle language which is available.

Now among vehicle language, if language is, if you are given a choice then you will use that language which provides you as much closeness to the process of pseudo code or the process in which you have written your program, you have developed your initial solution. So while natural language is mainly unreachable at present, programming languages are being made more and more sophisticated so that they can take into account what are called high level features. At the base of the computer everything runs within a machine language but since we do not want to program in a machine language, is not that you cannot write the program in a machine language. It’s very tedious and difficult to translate an algorithm which we have expressed in a mere English language in a sequence of steps into a machine code. So, people in computer science are developing languages which can understand and realize and translate more and more structured forms into machine code. Such languages, as you all know are called high level languages.

And these high level languages require several features which makes it quite useful for writing out our programs. And these features are related to data structuring, algorithm design specially problem decomposition. So some of these languages which are high level languages provide features for data structuring. For example you have got a lot of declared types, all of us know arrays. You can have several data types. For example if a language provides you a set data type then it would be more convenient for you to use that set data type rather than implement it as an array or linked list. So data type and operations, several languages provide more and more abstract data types and operations.

(Refer Slide Time 31:57 min)



- DATA STRUCTURING
  - → DATA TYPES, & OPERATIONS
  - → DATA-PROCEDURE ENCAPSULATION
  - → DYNAMIC DATA STRUCTURING
- ALGORITHM EXPRESSION
  - → CONTROL CONSTRUCTS
  - → FUNCTIONS, RECURSIVE FUNCTIONS

We shall see something called data procedure encapsulation which is another data structuring technique which is provided by certain languages, especially the object oriented languages. And related to algorithm design that is the exact sequence of steps, these languages provide you control constructs if, then, else, for, while, all of us know that and for decomposition you have got functions. you can decompose a problem into several functions. And among these functions, that is a very special type of function which is required for which the language may or may not have it. Look at this, this function calls itself. This function calls itself. This, when a function calls itself, we call this idea, we call such functions recursive functions.

Recursion will be a very big role in problem decomposition because when you decompose a problem into sub problem, you would like to solve the sub problems using the same technique that you would solve the original problem. So recursive functions, programs like Fortran do not provide recursive functions whereas others do provide like C, Pascal do provide the recursive functions. So, if a vehicle language just provide recursive functions then it makes easier for the programmer to write programs. And finally we have the concept of dynamic data structuring which is very useful in writing a solution to several problems.

Dynamic data structuring means that you would require some space at a certain kind of time and you would ask for that space, organize it and then when it is over, you can release that space. So that you can make use of available space as and when required rather than declaring the whole amount at the beginning. for example if I ask to find the maximum of n numbers and never told you what is the value of n, then you would find it very difficult to declare. Suppose I asked to sort n numbers, you have most of us know that we require an array of size n.

Now if nobody told you the value of n apriori or even the maximum value of n, it would be difficult in most programming languages to declare the value of n, to declare an array of size n. In Fortran, you cannot do it but many languages will provide you with dynamic allocation, facilities for allocating and deallocating data dynamically during the program. So when such facilities are available, it becomes much easier to translate their final solution into a program. So first we need to know what this vehicle language provides and how can be used automatically in problem solving. And then we will see how to solve problems using such a vehicle language. In this course our vehicle language will be the language C. And what we will do during our study here is that we will study C, the feature of the language C as well as the steps in problem solving hand in hand and solve several example problems one after another and also learn C as we go about solving it.

The objective of this course is not merely to learn C and not merely to write that data structures in C. The objective of this course is to understand what programming methodology also means and what data structuring means and how they go hand in hand in solving problems. So that will be the objective of this course. Initially we will start with learning a bit of C language and seeing what constructs are available in C language because unless the vehicle language is known to us, it will be very difficult to solve the, get the final solution because starting from here if you cannot end up here, if you end up in between anywhere, it does not mean much because it is our task to write out the final program. And even from the translation from a solution to a final program may take some effort depending on the language which is available to us. And the language C has to be studied because it does provide several interesting features which can be automatically used from our final solution. That is our final solution itself can be expressed in a high level which has got a reasonable one to one correspondence with the language C in many situations.

Therefore our study of C will also be a study of problem solving in some sense. Therefore the objective is to understand what problem solving is. The language C will be a vehicle by which we will understand it and the issues which we will require to understand or what is problem decomposition and what is meant by initial solution refinement. That is what this course will be about and that is what we shall study from the next class onwards. Thank you.