Programming and Data Structure
Dr.P.P.Chakraborty
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 32
Conclusions

Hello everybody. Today, we come to the conclusions of this course. The programming methodology and data structures is a course which can go on till infinity that is no end to continue with the techniques of programming. These are evolving techniques, even nowadays newer techniques are coming up. Therefore we would like to draw clues and try to recapitulate today the salient features of the aspects of programming methodology and data structures.

At the outset I would like to reiterate one thing, we are trying to solve the problem on the computer but we must not be dictated by actually what is available in the computer today because if you are dictated by what is only available in the computer today then the process of human creativity in problem solving would go. And this process of human creativity which leads to conceptual problem solving would actually be reduced to trying to write programs in the given architecture. So programming methodology and data structuring has got two totally different aspects, one aspect is the conceptual problem solving aspect, how you would conceptually organize your information, how would a person mentally organize his or her information about the problem, how they would solve the problem. By solving we have discussed various things about solving. The second aspect is mapping it into the computer language, mapping it into a programming language.

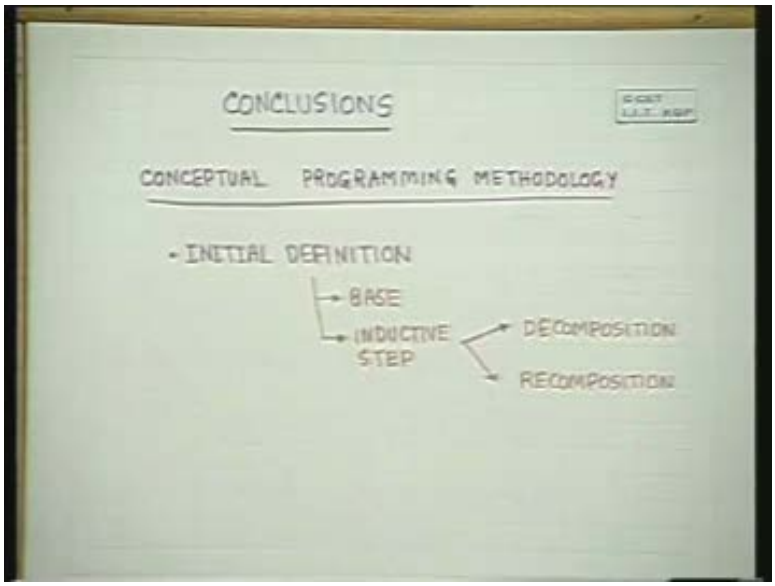
Now all programming languages will not provide you with the facilities that a human conceptual thinking would be like because we think in an abstract manner and because of this sort of abstract thinking, we have got abstract structures and conceptual components of problem solving. Now programming, the paradigm of programming has slowly evolved and due to this evolution several such conceptual structures have cropped up in new programming languages. If you look at an assembly language program, you will see that even simple assignment statements, conditional statements, expressions all of them have to be written out in a very very long and difficult manner. High level languages which provide you if then else structures, while loops, do loops and even the concept of arrays started the concept of structuring which was available in a programming language. But when a person can solve it in his or her mind, these structures conceptual structures are always available. And there is a semantic gap between the person's thinking style and actual implementation language which is provided to a person.

Programming methodology and data structures is not related to what is the implementation language only but it's definitely a very important point to understand what is the implementation language. It is important to translate our thoughts into an implementation language but it is also important to conceptually structure our thoughts in

such a manner that we are able to understand, grasp and grapple with the problem at hand in a more organized fashion.

Therefore what we have tried to discuss in this course on programming methodology and data structures is both conceptual methodology, conceptual data structuring as well as actual programming methodology and actual data structure. Coming to conceptual data structures, what did we look at the first thing in a conceptual structure? Conceptual problem solving as we said that the solution to a problem comes into the head of the problem solver in some fashion. And in order to translate it into a step by step solution on a computer certain ideas have to be put into. You cannot say that given n numbers obviously I can see 3 numbers and I know which is the maximum but just seeing them and finding out the maximum visually does not lead to a programming style or a solution to a problem. And while discussing this we came to the concept of decomposition. So, the first and the important thing about conceptual programming methodology, first the concept we discussed, the concept of the initial definition and this initial definition we said has two parts, there was a base condition, there was an inductive step.

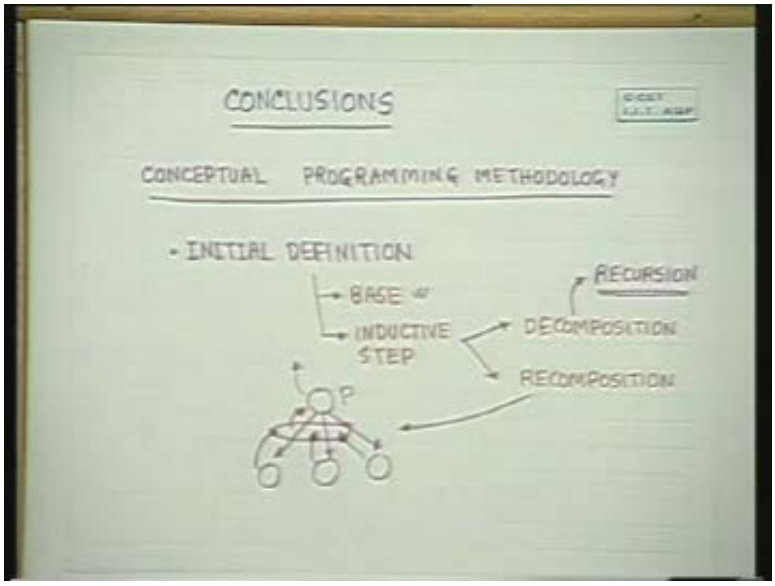
(Refer Slide Time 07:27)



And this inductive step has got two phases, phase one was decomposition into sub problems and phase two was recomposition of the sub problems. And we have solved several sub problems and obtained an initial definition, some solution to the problems. Somehow we have to obtain an initial definition and based on and we saw that in conceptual programming methodology this decomposition, in this decomposition we used the concept of recursion. That is we call the same problem again and we have done so many problems I do not wish to repeat and pick up examples. We have tried to solve every problem that way and this decomposition is sub problem. So we had a problem p either that this was true, so we returned the solution or we broke it up into sub problems. We obtain solutions to these sub problems in the same recursive manner and then these provided their solutions and then we had this recomposition step here and then we solved

the problem. So this was our initial definition, a definition by which we were able to define some solution to the problem.

(Refer Slide Time 08:42)



Now what is the advantage of such a conceptual initial definition? The advantage of recursion is or an induction is that based on recursion and induction, we can get a proof of correctness of the solution. That is when we have developed a solution which is recursive in nature, we are also able to provide with that initial definition a proof of correctness, we are guaranteed and we are able to prove that this definition is correct and that is the first step in conceptual problem solving, the first step in programming methodology that is generate an initial solution. You cannot start writing a program, if then else loop how? You are given a problem and for this problem you have to generate a solution. So you have to generate some initial solution and this initial solution has to be correct.

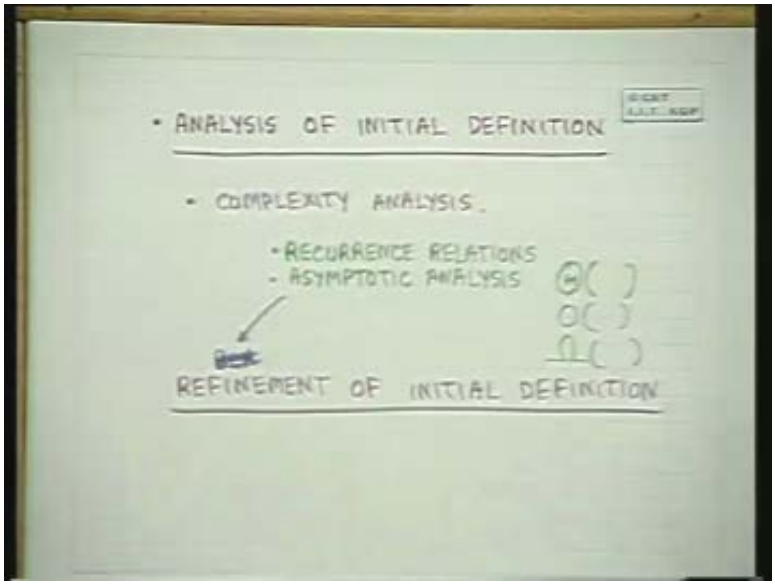
So the first step we saw in conceptual programming was generate an initial solution and check whether it is correct or not correct and give a proof of correctness. Now what are the components of this initial solution? This initial solution, the first thing we had in this initial solution was flexibility. This initial solution must not a rigid solution, this initial solution contained various aspects. We could have interchanged the sub problems, the order in which we did the sub problems did not matter also we put in a generalized conditions like split air into two sub lists, non empty sub lists l 1 and l 2 that is the flexibility indicated that the initial solution contained several alternative possibilities and we said that a initial solution has got several alternative possibilities, split a list l 1 and l2 into two non empty lists or generate 5 sub problems. Now you can do them in any order, now which order is the best?

So the initial solution is actually not a solution but a set of solutions combined together in an initial recursive wrap around definition, so that was the first thing. The main criteria

here are 2, one is you have got some solution to the problem which can be translated into a step by step definition and secondly you know that what inside it that the solution is correct. So in conceptual programming methodology that is the first thing. Here all the data elements none of them are pin pointed out, nobody says that this should be a array, every data element her is general enough and kept as we have done together as a set. so the second point is flexibility of definition is the first point, second point is data is kept in most general form a set, a list.

You don't say it's an array, it's a linked list of pointers etc etc, we just keep it in a general concept. So after we generate this initial definition which is the nucleus of your problem solving, next came the phase of refinement. So the second phase was analysis of initial definition. In analysis of initial definition the first thing that we discussed was complexity analysis and in complexity analysis we argued out various things.

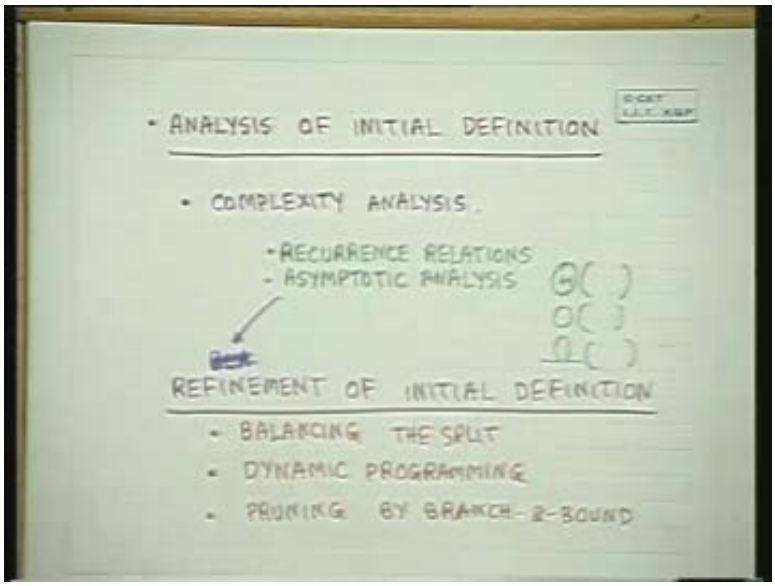
(Refer Slide Time 16:08)



One is given an inductive definition, complexity analysis can be done by using recursive equations or recurrence relations and we use recurrence relations to obtain the complexities. So the first thing that we used here was recurrence relations. Next we argued out that since we are working on many computers and each of them has a different speed, the size of the input matters and we have put in lot of arguments to argue out that getting exact values is actually not very meaningful. In order to understand why one algorithm will be better than another approach, why making one choice would be making than another, you remember we have number of choice points in our initial definition which is the flexibility and each of these choices would lead to a different complexity and we have seen that in order to make a very proper definition, a very proper analysis, we cannot go in for exact analysis because the values of speeds of compute another issues are there and there we came to the concept of asymptotic analysis.

In asymptotic analysis we had the ordered notation, the order, reorder, the lower bound rotations and based on this asymptotic analysis we tried to pick and choose which of this design strategies we will use. That is based on asymptotic analysis, we could make certain choices and these choices were related mainly to how to balance the split.

(Refer Slide Time 17:00)

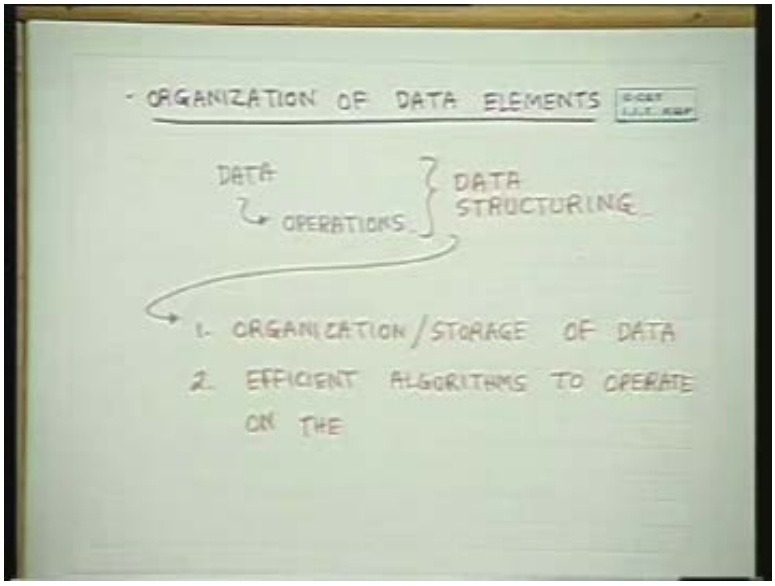


So based on this analysis of the initial definition, we moved on to the concept of refinement. In refinement of initial definition we saw how to modify the initial definition to obtain solutions, the exact solutions. One of them based on this was balancing the split. Second we saw identical sub problems being solved only once by the concept of memorizing which we saw the concept of dynamic programming and thirdly we also saw pruning by branch and bound. These are the three techniques that we have seen in refinement. There are several other techniques in refinement. These techniques of refinement are related to how to improve the initial solution and what is the criteria for improving this initial solution, the major criteria is reduction of complexity. Reduction of complexity means reduction of time, reduction of space and since we are moving into, we cannot do for each and every input asymptotic analysis and try to reduce the space and time asymptotically in the worst case scenario.

So, asymptotic worst case analysis was used as the basis or the yard stick for refinement and complexity, the time complexity came first and then came space complexity and the techniques that we used were balancing. That is the nuclear space that we had in the initial definition, the possibilities that we had in the initial definition were examined and algorithmic, designs, techniques came up. Till now we are not really bothered with the programming language that we have to use to solve the problem because if either it is recursion or there is no recursion, even if there is recursion or no recursion we can still solve the problem. I hope all of us understand but we will come to this issue a little bit later but conceptually we will solve the problems. The third step after the refinement of the initial definition came organization of data elements.

In the whole program you had some data and there were two aspects and on this data you have some operations. For example while solving the problem or may be the problem itself may be such that your data and its operations are clearly identified. You have got a set of integers and you have to insert new integers and you have to pick up the maximum at continuous point of time.

(Refer Slide Time 17:00)



So your operations are insert and delete max. So once you get the solution, you know that these are my data elements and you also know that these are the operations on these data elements. So now you are left with two things, one is how to store the data that is in what exact form I will store the data and so that the complexity of operations on the data is minimized and that is where we came to the very important issue of data structure. Now data structuring relates to one organization or storage of data and second efficient algorithms to operate, here operate means these operations on the data. So to generate an algorithm we require data structuring, for data structuring we require some more algorithms and this loop goes on and on.

So data structuring means how to organize the data and how to store it. Now here came a very important aspects, is the data is statical, static. That is static means what? Data is inserted and your operations do not change that is do not add new data elements and do not remove some data elements. So after some point of time you can say that my data is static or dynamic, dynamic means data elements are new, new data elements are added and data elements are deleted. On a static data set you can modify certain aspects of data but on a dynamic set, even the whole some elements of the data may be totally removed and some may be added.

So in data structuring we had two very important concepts of data structuring, one was static data, second was dynamic. Now storage, how we will store static data and how we will store dynamic data? And in order to store static data, we saw that various structures

are available in dynamic something has to be done. So there are two aspects here. Other than this in data structuring there was a conceptual structuring. For example student information that the name, roll no, etc, subjects student has taken, the credits of that subject, the grade a student has got for that subject. Now all this for a student we would like to keep.

Now you can store them in various ways, you can store all students as an array, you can store all subjects as an array, you can store students and subjects as separate array or you can put all students and subjects together. So the next concept was encapsulation of data. That is when you go to storage of data how you will encapsulate data. In both static and dynamic you had encapsulation or in c language terms structure, these encapsulations were also called structures of data. So how to organize data, which data will be structured with which other data, how will they be linked up, whether they will be static or dynamic all these issues are very important. And among these issues in data structuring was how they will be linked up, so that the functions that operate on them are efficient. And when you come to functions that operate on them, that means data elements are there and they are connected.

For example in a matrix, elements may be connected the rows, the next row, the next column or in a list of elements the next, next, next how will we operate on this. For this we saw several such data structures. What is a data structure? It is a set of data and operations efficient operations so that efficiently they can be operated upon. And what data structures we saw? we saw stacks which was first in first out, it was a linked list, a list of linked list of elements and in this linked list of elements, we have seen first in first out, insert in the beginning, delete from the beginning, insert from the beginning, delete from the end, insert in the end, delete from the beginning. These are all the possibilities that you can have, names you will see in the books, stacks, queues and other things. We also saw other important data structures binary search trees, balancing of balanced search trees why it is important to reduce the time? So data structuring comes again and again and we have seen some of them.

We will see newer problems, newer data problems will come up and in some problems which is not known today which will be defined by its own set of operations even better data structures will come. So data structuring was a very important part of algorithm design and as we have just discussed, in order to get data structuring we need to do good algorithm design. So, algorithm design and data structuring or programming methodologies and data structuring, programming methodology involves algorithm design as well as data structuring. These two together worked in hand in hand to develop a program. So these are all our conceptual ideas. Now we have to translate them into program. So in a program we have to see what programming language offers and based on that programming language, we have to write our program.

Nowadays programming languages offered you recursion, programming languages offered you structures, programming languages offered you dynamic allocation. So all these were not available years ago and at that time, even at that time conceptual decomposition was there but the translation had to be different but today the translation

has to be done, can be done in a more elegant fashion. That is you are now closer to your conceptual programming methodology as with the actual programming methodology as you would do in a language like c. Here you are provided with dynamic allocation, you are provided with recursion, you are provided with structured definitions, you are provided with arrays, you are provided with pointers, you are provided with linked lists and so many other things.

So once you come to translating into a programming language, the conceptual structure has to be adapted to the programming language which is the current scenario. now tomorrow a language can come which is much better than c which allows you to automatically write down more complex structures that you would concede and c plus plus is one language where class definitions with the, were not only data or organized as a structure but data and procedures are encapsulated to form a class.

Therefore these are more general structures and therefore programming, so programming concept conceptual programming methodology has to be understood and that is most important. Once conceptual programming methodology which includes algorithm design and data structuring conceptually is understood better, it is only then when you have to translate into actual language, you come to use in the language in hand but nowadays since recursion is available, since dynamic allocation is available and very soon even higher level structures and queries.

Suppose you are doing it on a database then you would translate it differently but your conceptual structuring would rather remain a lot same. therefore what I would like to stress is that programming methodology and data structures is understood conceptually and it has two phases, one phase is that of conceptual algorithm design which involved as we have seen and as we have studied. initial definition, correctness of the initial definition, understanding the initial flexibility in the initial definition, generating analyzing the solution and trying to generate a refinement of the solution, over a number of this you can analyze and see whether this is the best possible, is there anything better than this, can there be anything better than this.

These are issues of complexity theory which I do not want to go in it at present. And next once you have got your conceptual algorithm and data structure, now you translate it into your programming language and if you don’t have recursion, you will use a stack and you will implement recursion. If you have recursion you will directly use recursion. If you have a structure you will use a structure definition. If you don’t you will organize it in a particular manner, if you have or do not have dynamic allocation you have to approach the problem in a particular way.

I do not want to reiterate all the things that we have done in this whole class but programming has to be understood as to two or three things, first what ever you have written must be correct. That is the first and foremost criteria because if you write something which is incorrect there is no point in trying anything else. after it is correct, it has to be seen that it is efficient and in order to see that it is efficient, you have to do a lot of these analysis and once you have done the conceptual analysis in a conceptual manner

and done a refinement conceptually, translating it into one or more of these known programming languages or the one in which you have to translate it through is not a very difficult task.

Finally you have to remember whether is the best possible that if you can prove then you are done. Lastly but not the least some people are trying and we have not done in this course is given a final solution, you have to verify whether this translation from the initial definition to the final solution is indeed a correct translation. Verification is a very difficult task and these all together makes the art of programming. Programming is a science, it is such a refined science that it tends to become an art when programs are becoming very very good programs. So I would suggest that these philosophies which we are trying to study in this course, we tried again and again till these are imbibed on a person so that they are so natural that you do not have to follow the steps, they will come to you naturally as I expect to come anyone of you. Thank you for participating in this course and I wish you good luck.