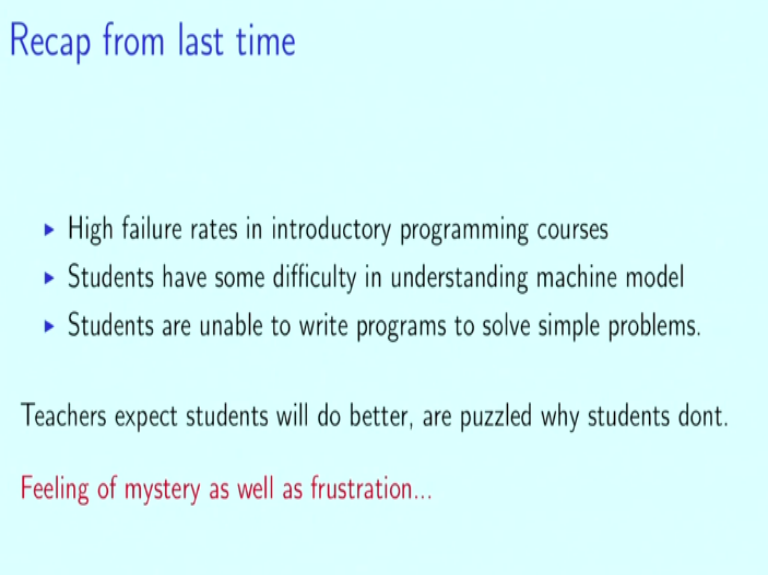
**Design and Pedagogy of The Introductory Programming Course**

**Prof. Abhiram G. Ranade**
**Department of Computer Science and Engineering**
**Indian Institute of Technology – Bombay**

**Lecture – 05**
**Basic Ideas in Our Approach. 0: Introduction**

Hello and welcome again to the course on design and pedagogy of the introductory programming course. In this second set of lectures I will talk about the basic ideas in doing this design and the pedagogy. So a quick recap from last time.

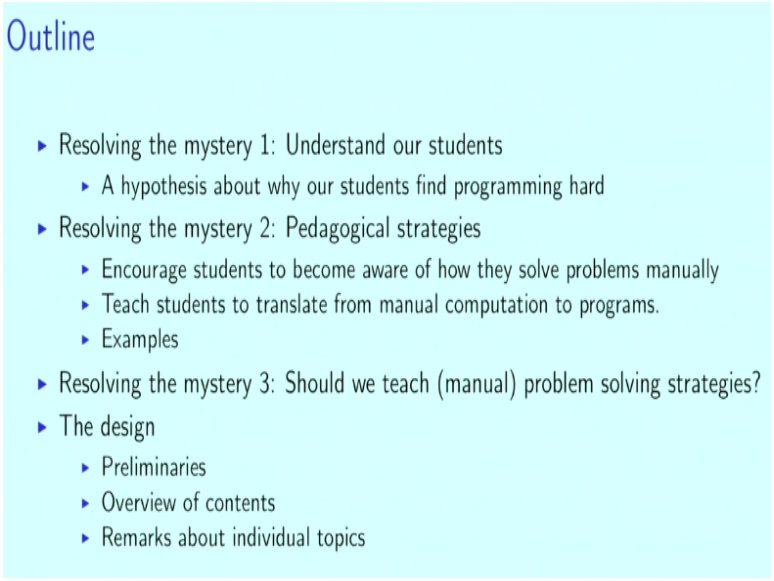
**(Refer Slide Time: 00:35)**




So we said that in programming courses worldwide there is fairly high failure rate and because programming happens to be a fundamental course for computer science and also a very important course for other branches we should take this failure rate very seriously. We said that students have some difficulty in understanding the machine model and more important perhaps is the difficulty that students are unable to solve.

Or, unable to write programs to solve relatively simple problems. Teachers expect that students will do a lot better than how they are doing right now and there is some puzzled as to why students don't do that. So in other words there is a feeling of frustration but there is also a feeling of mystery what is going on here.

**(Refer Slide Time: 01:36)**

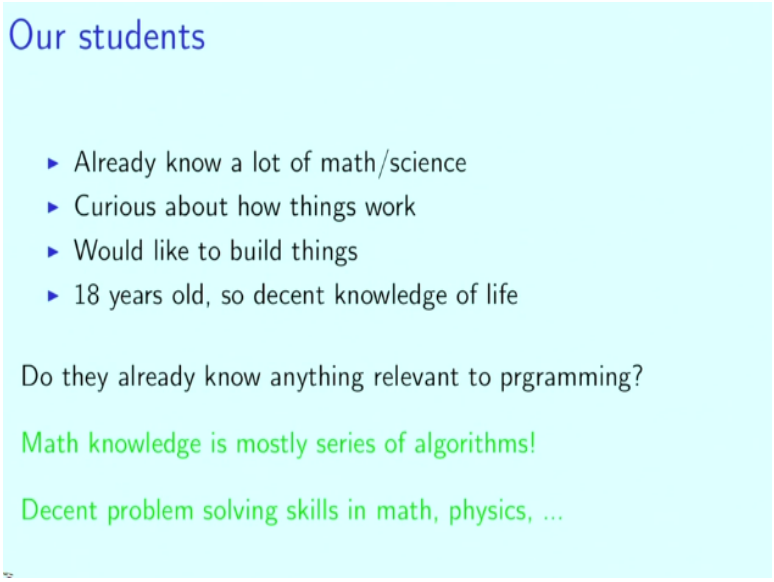



So here is what I am going to do today. I am going to try and resolve this mystery and for doing that we will work towards understanding our students better. And we will present hypothesis about why students find programming hard. Then we will indicate some pedagogical strategies, which are addressed towards the difficulty outline in the hypothesis. Specifically, we will encourage students to become aware of how they solve problems manually and we will teach students how to translate from manual computation to programs.

We will give a number of examples of this and then we will consider the question which we were talking about a little bit last time as well. Which is; should we teach problem solving strategies? So is it the case that students do not understand how to write programs because they do not understand problem solving. So we will dwell on this question a little bit. Then we will talk about the design.

So after a few preliminaries I will talk about the contents that we want to teach or the topics or the outcomes that we desire. Then we will make some remarks about individual topics.

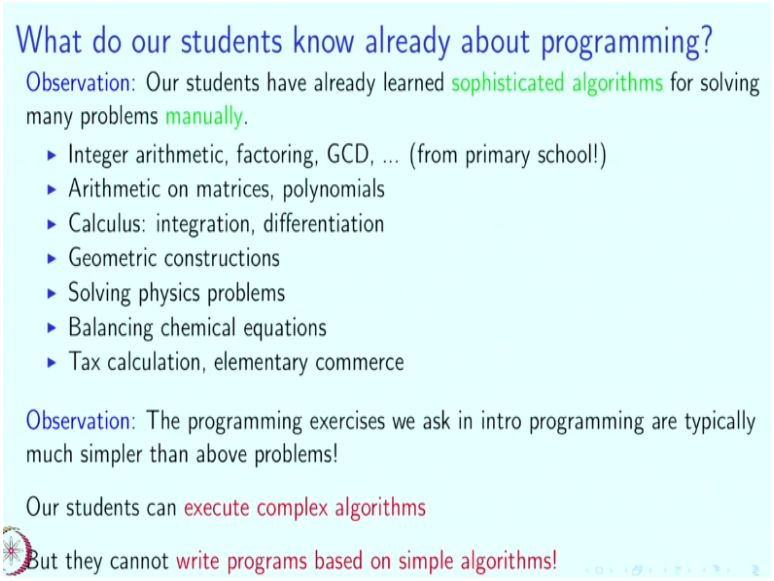
**(Refer Slide Time: 03:01)**




So let us begin in earnest. Who are our students? Well our students when they come to us have already done a lot of math and science in standards 1 through 12. They are curious about how things work and they would like to build things. They are 18 years old or so and they have a decent knowledge of life. What do they know really about programming when they come to us? As it turns out if you think about it the knowledge of mathematics that they have gained in these 18 years of life or whatever 12 years of school life is really mostly a series of algorithms.

It is not that they are starting with a blank slate. They really know algorithms. They also have decent problem solving skills by that I mean they have some information about math and physics and they are able to apply that information to solve day to day life problems. So they understand programming, but they do not understand programming, but they understand algorithms and they understand how to apply knowledge to problems they might be seeing for the first time.

**(Refer Slide Time: 04:25)**




So let us understand this question a little bit better. What do our students really know about programming? So I claim that our students already have learnt fairly sophisticated algorithms for solving many problems, but not on a computer, but solving them manually. In primary school they have learned how to do integer arithmetic, multiply 2 numbers, divide 2 numbers, how to factor a number, how to find the GCD of numbers.

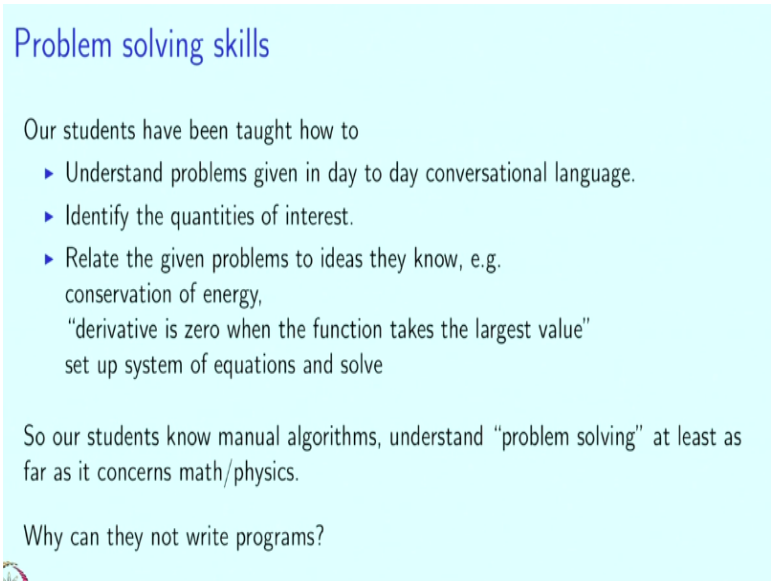
This knowledge is acquired and understood by them in terms of algorithms. They know how to find the GCD. They know how to multiply. So that is basically a bunch of algorithms and if you think about these algorithms are fairly sophisticated they contain reputation, they contain conditional testing and some of them might also be thought of as recursive algorithm. Then they know arithmetic on matrices, polynomials. They have learnt a little bit of calculus.

They know how to integrate or differentiate and these are again algorithms. They know a lot of geometric constructions. So, given a line segment how do I find its midpoint? What is that? That really is an algorithm. It is an algorithm that uses compass and rulers, but nevertheless it has very specific steps, which had to be followed in specific order and that is an algorithm. They may be solving physics problems, for example if there is an inclined plane and there is a block sliding down.

What do you do so they know precisely what is to be done and again that knowledge many people understand as an algorithm balancing chemical equation is again an algorithm may be set up simultaneous equations may be multiply both sides and so on. So in fact often the chemistry teacher will give you a sequence of steps. Tax calculations, elementary commerce they understand these things and again they know how to problems in it. Now, the puzzling observation.

The programming exercises that we ask in introductory programming are typically much simpler than many of these problems. Our students can execute much more complex algorithms, but they cannot write programs based on simple algorithms. Isn't that puzzle. So they can do, they can execute very complex algorithms, but they cannot design for themselves simple algorithm or they cannot write as algorithms things that might be able to do by hand in a very, very easy manner.

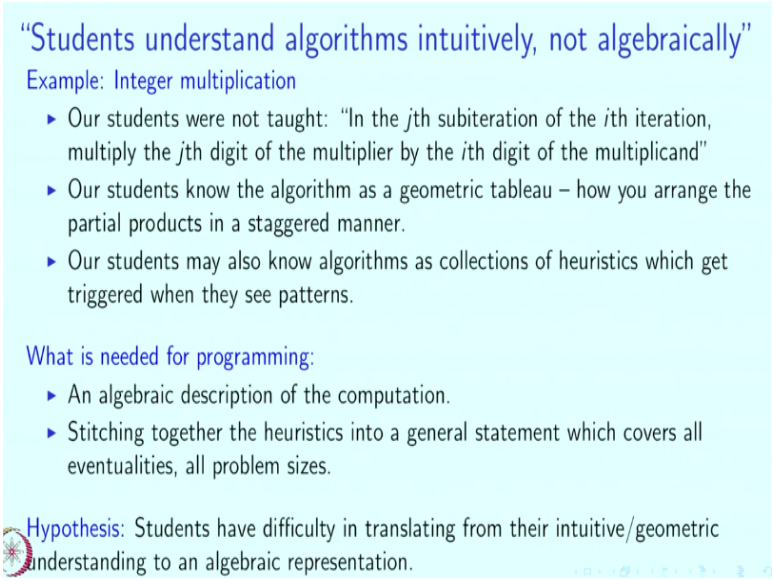
**(Refer Slide Time: 07:39)**




Let us look at their problem solving skills. So our students have been taught how to understand problems given in day to day conversational language, identify the quantities of interest, relate the given problems to ideas that they know. For example, they may have to use conservation of energy. They may have to use the principle that the derivative is 0 when the function takes the maximum value.

So they know how to dig up these principles from their memory from what they have already learned and apply these principles or they may have to set up system of equations and solve it. So they know how to do that as well. So again our students know manual algorithms. They understand problem solving at least as far as it concerns math and physics. Why can they not write programs? Why can they not express this knowledge that they have as a program.

**(Refer Slide Time: 08:39)**




So here is hypothesis. We believe that students understand algorithm intuitively and not algebraically. Let me explain what I mean by that. So let us take integer multiplication which you study in third standard or something like that. So our students who are not taught multiplication in the following manner. Their teacher did not come and say in the jth sub iteration of the ith iteration multiply the jth digit of the multiplier by the ith digit of the multiplicand.

Go back to your time as a third standard student and tell me if your teacher taught this. Of course no. Your teacher said something quite different. What did your teacher say? Well your teacher described the algorithm as a geometric tableau. How you arrange the partial products in a staggered manner. So let us go to paper and I might be multiplying 1234 by 567 so your teacher made a big deal of writing these in a nice tabular manner so 7 * 4 =28 then 2 is carried so you * 7 then the next value you shift and 6 * 4 is 24.

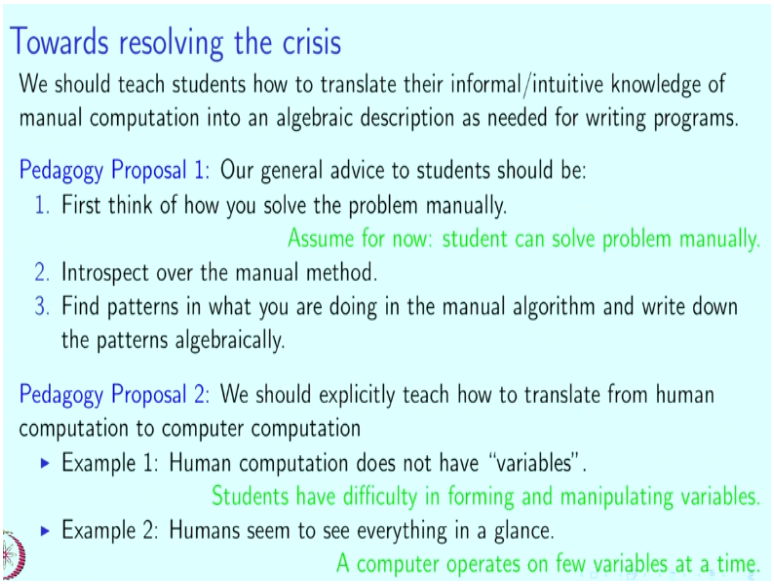
So you either put 0 over here or a cross over here but you align. So this alignment and this geometry is really important. I bet your teacher scolded you if you did not get the alignment right and of course she was correct in doing that. You learned this whole thing as a geometrical algorithm in some sense. Then our students may also have been taught many heuristics. So they might say may be this is later on, but they might say for example that how do you test if a number is divisible by 3.

So there is heuristic for testing numbers for divisibility by 3 may be it is divisibility by 9 and so on. And when needed these heuristics got triggered or you may see that when you have to derive, when you have to take the derivative of a sum then you use this rule. So again if you saw sum the sum pattern you would trigger the rule for taking derivatives or derivatives of sums. Now this is an exactly what is needed for programming.

What is needed well what is needed is an algebraic description. So our students need to know something like what we have stated in the first bullet. They need to know they need to internalize their knowledge as in the jth sub iteration of the ith iteration we need to multiply the jth digit of the multiplier with ith digit of the multiplicand. If they know this, then as you can see they are very close to writing the program.

Then they may know several heuristics, but do those heuristics cover the entire range of problems that they might need to solve. How do you organize those heuristics so that you systematically cover the entire range? This is not really taught and certainly how do you organize that is not really taught. And this is what is needed for programming. So our hypothesis is students know things but they have difficulty in translating from their intuitive or geometric understanding to an algebraic understanding.

**(Refer Slide Time: 12:50)**




So now I want to say something about how should we resolve this problem? So very simply we should teach students, how to translate their information or intuitive knowledge or geometric knowledge of manual computation into an algebraic formal description that is needed for writing programs. So, our first pedagogical proposal. Our general advice to students should be first think of how you solve the problem manually.

Assume for now that students can solve the problem manually. If the student cannot solve the problem manually what should we do? We will deal with that a little bit. That is an important question, but we will deal with that a little bit and we should also note write now that there are many, many, many problems that our students can actually saw so perhaps this is not as important question as it seems to be.

But it is important that students do think about manually solving the problem most of the time they do not how to solve the problem manually. Next step, students should introspect over their manual method. By this I mean they should really think about what they are doing, what am I doing first, what am I doing next. Are there patterns and all of this? So are there patterns am I doing the same step repeatedly and they should write down the patterns in an algebraic sense in an algebraic manner.

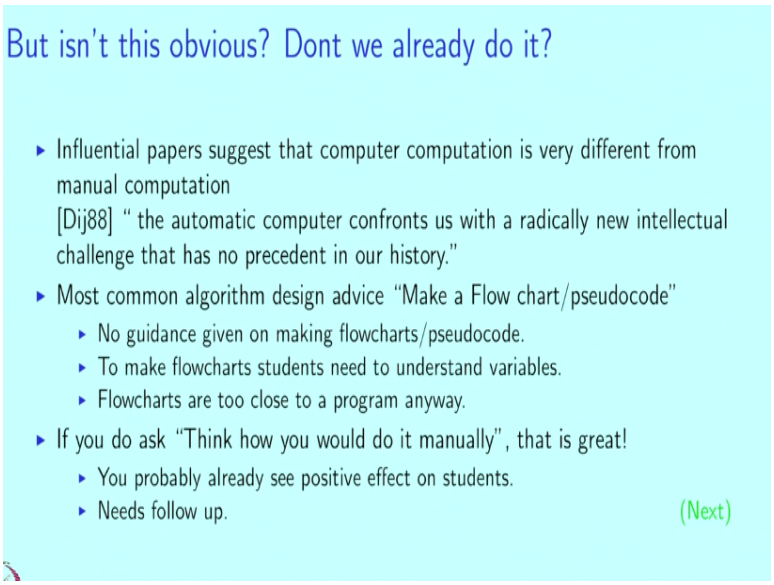
So manual could be I am doing this whole thing and times or I am doing this whole thing as many times as there are digits in the multiplier or in the multiplicand or whatever problem whatever parameter in the problem that you are solving. So that is what they need to write. The second pedagogy proposal is once the students know what exactly they are doing they need to translate from this description to the computer computation description.

So human computation to computer computation and we had to tell the students, we have to teach the students how to do this translation. There are differences between human computation and computer computation here is one example of it. In a computer computation we have a notion of a variable in human computation we do not really have a notion of a variable. We may give names to values, but that is not the same thing as having the variable.

We rarely change the value of a variable. In fact, experience with teaching programming shows that students have difficulty in deciding how to form variables and then how to manipulate variables in a correct manner. Here is another example. Human beings seem to see the whole problem in a glance whereas a computer operates on a few variables at a time.

So these are some of the differences and what we need to do is to go over these differences and tell the student how to deal with these differences. Basically tell the student that if in a manual algorithm you are doing this in a computer algorithm this is the equivalent, these are the equivalent steps that you should be taking.

**(Refer Slide Time: 16:35)**



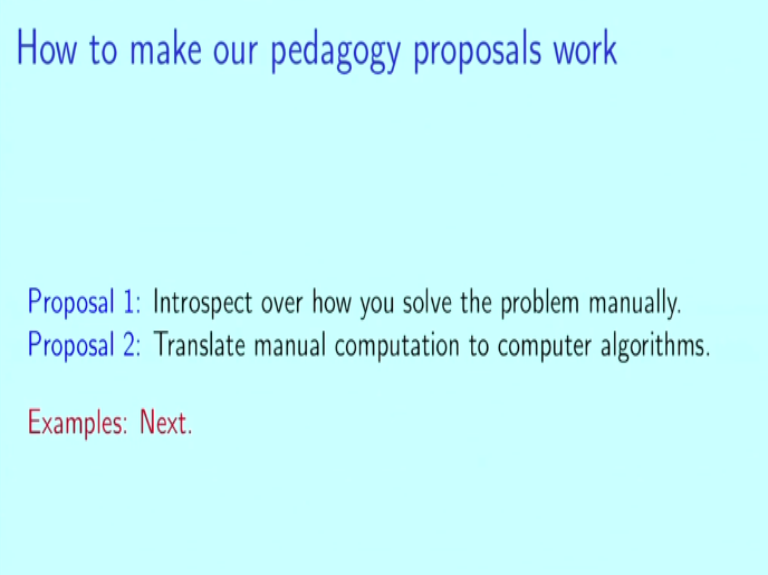

Now you might say look isn't this obvious. Don't students do it anyway? Well they do it and in fact experts often explicitly say that computer computation is very different from manual computation. Dixtra who is considered one of the founding fathers of computer science has said automatic computer confronts us with a radically new intellectual challenge that has no precedent in our history.

So he is saying that programming and computer computation is completely different. So what we are saying over here is that no they are not different and in fact you should be building up on what students already know. Now you also say that most common algorithm design advices make up flow chart or write down pseudo code. However, no guidance given on making flow charts and for example when people make flow charts they already assume that students understand which is very far from the truth.

Students do not understand variables and that is one of the things that they need to know how to translate and we will see more such things and then people talk about making flow chart or pseudo code. They do not really give guidance regarding all of this and in one sense saying that you should be making a flow chart isn't really helping because flow charts are already too close to a program anyway.

So it is like telling the students well you do not know how to write programs, but why do not write something like a program that is not fair advice. However, I am sure there are some teachers who tell their students think about how you do this manually. If you are one of those teachers I congratulate you, I think it is great and I think you should keep it up, but you need to follow it up as well and how do you that follow up I am going to tell you in a minute.

**(Refer Slide Time: 19:03)**




So how do we make our pedagogy proposals work? So our proposal was telling the student introspect over how you solve the problem manually and then help the student to translate manual computation into computer algorithms. So we are going to do this. I am going to tell you how to make these proposals work and I am going to do this through several examples which I am going to come to in a minute.

So in this lecture I have told you what I think are the difficulties that student face in writing computer programs. And I have proposed 2 strategies or 2 pedagogical ideas which will help students and those are the ideas that we should teach. So in the next lecture I am going to give examples of programming problems and how those ideas can be made to work in those programming problems.