Design and Pedagogy of the Introductory Programming Course

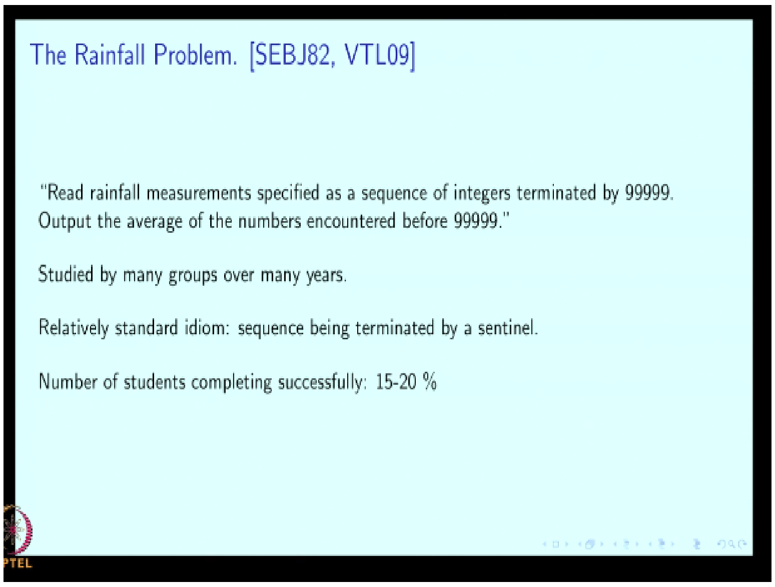
Prof. Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture - 03
Introduction and Survey.1: Experience with the Standard Approach

Hello welcome back again. In the last segment I talked about the standard approach. By standard approach we mean the approach in which you use a language like C maybe Java or Pascal or maybe Python or C++ and essentially teach in a standard what might be called an imperative style program, a programming model. So we discussed the standard approach and we looked at some learning objectives at various universities.

Now I want to talk about the experience with this approach and this is based on worth at researchers have done and reported in the educational literature. So one of the most famous pieces of works considers this so called rainfall problem. This problem has been studied for quite some time.

(Refer Slide Time: 01:24)



The Rainfall Problem. [SEBJ82, VTL09]

"Read rainfall measurements specified as a sequence of integers terminated by 99999. Output the average of the numbers encountered before 99999."

Studied by many groups over many years.

Relatively standard idiom: sequence being terminated by a sentinel.

Number of students completing successfully: 15-20 %

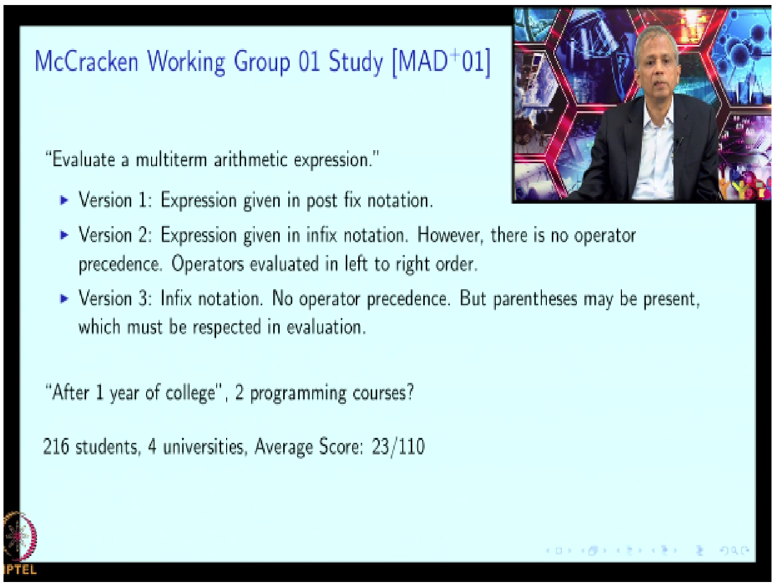
Soloway and his colleagues devised this problem as long ago as 1982 and I have seen the references and studies using the same problem in 2009 as well. So what is the problem, read rainfall measurements specified as a sequence of integers terminated by 99999. Output the average of the numbers encountered before 99999. Seems like a really simple problem, seems like there is just a loop with the loop termination condition being slightly tricky.

Actually it is not too tricky because this is what is called a sentinel. The 99999 is what is called a sentinel in many books and in the literature. The experience with it is fascinating and so it has been studied by many groups over this long period. So typically in all the studies that have been done with this problem it is really shocking that only 15-20% of the students successfully write the program.

Look it is just a single loop and it is a simple sentinel and maybe the students are confused because it is described in terms of rainfall measurements, but whatever it is, it seems really strange and noteworthy that our students are not able to write this program. I will encourage you to give this program to your students and see how many of them complete. Actually, I want to make one more slide point.

So I think the problem over here is related probably just to simple loops. How do loops work on a machine? When is that condition tested, how does control flow through a loop those are the issues which it seems that students are not really getting and being able to use. Next, I want to come to a study done by a group of people led by McCracken and this was done around the turn of the century.

(Refer Slide Time: 03:46)



McCracken Working Group 01 Study [MAD⁺01]

"Evaluate a multiterm arithmetic expression."

- Version 1: Expression given in post fix notation.
- Version 2: Expression given in infix notation. However, there is no operator precedence. Operators evaluated in left to right order.
- Version 3: Infix notation. No operator precedence. But parentheses may be present, which must be respected in evaluation.

"After 1 year of college", 2 programming courses?

216 students, 4 universities, Average Score: 23/110

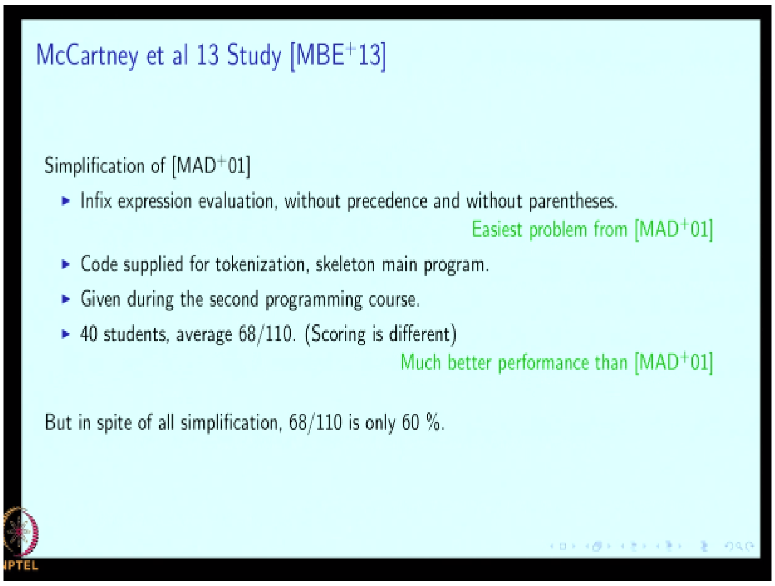
So there were many people some 7-8 co-authors were there. So the basic problem they gave to students was to evaluate a multi-term arithmetic expression. So an expression was given to you say 1 + 2 + 3 times 5 raise to 7 - 8 whatever and you were supposed to evaluate it. In the first version the expression is given in a post fix notation. So this post fix notation is explained or this problem is given to students who have done the first course or maybe who have done one extra course.

So they may have had some familiarity with post fix. The second version is that the expression is given infix notation, okay so the (()) (04:40) normally right; however, there is no operator precedence. Operators are evaluated in left to right order. So this is a much simpler form of the problem. So you just read left to right and keep on doing the operation that is given to you by the last operator.

The third version uses infix notation but there is no operator precedence. So students do not have to worry about operator precedence, we have to use infix notation, but there are parentheses. So if there are parentheses then they must be respected. So whatever is inside the parenthesis has to be evaluated first and only then the result of the parenthesis can be used to be operated upon with the adjacent numbers and operators.

So this was done after 1 year of college or maybe so could be 2 programming courses but maybe a slightly more mature student than just the introductory programming course. It was reasonably last study involving 216 students, 4 universities and the average score was 23/110. So this problem is harder than the rainfall problem surely, but for somebody who has done 1 to 2 course in programing, it would seem that this should not be that difficult.

(Refer Slide Time: 06:21)



McCartney et al 13 Study [MBE+13]

Simplification of [MAD+01]

- Infix expression evaluation, without precedence and without parentheses.
  Easiest problem from [MAD+01]
- Code supplied for tokenization, skeleton main program.
- Given during the second programming course.
- 40 students, average 68/110. (Scoring is different)
  Much better performance than [MAD+01]

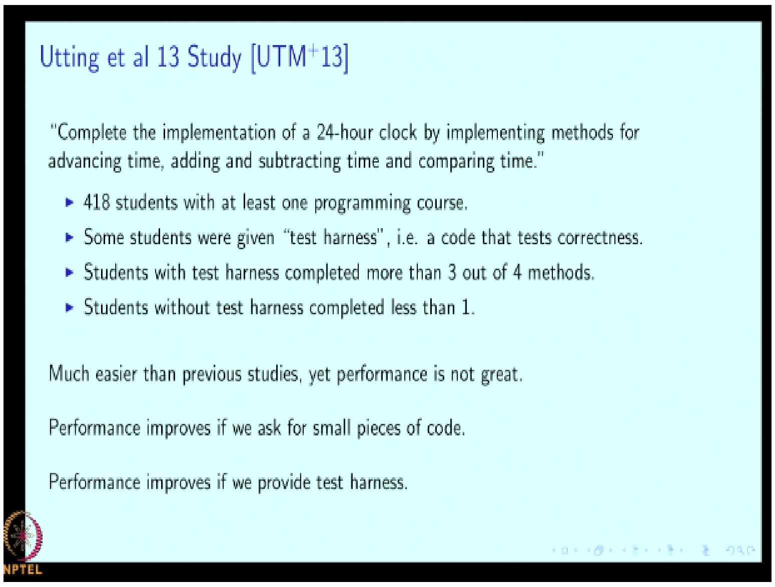
But in spite of all simplification, 68/110 is only 60 %.

McCartney and co-authors studied something like this and it was actually a simplification. So they only used infix evaluation without precedence and without parentheses. So this was the easiest problem from amongst the 3 problems that were used in the previous study. Furthermore, they supplied code for tokenization. So the code contains, the code was given which would say okay now extract the next token and check whether the next token is an operator or a number.

In addition, a skeleton main program was also give. This was however, given during the second programming course. So this was given to students who had taken one programming course. So the students were slightly less mature than possibly the students that were involved in the previous study. This was a smaller study, but they found that on the average the marks were better. So 68/110 and instead of 23/110 and the scoring in the 2 cases is slightly different, they gave marks for getting the structure right.

They may give some marks for style, so those were the differences, but here in any case there is some improvement. So instead of the marks which were something like 23/110, now the average is about 68. So much better than the previous, but in spite of all simplifications it is only 60%. You would think that after all these simplifications maybe more maybe students would obtain more.

(Refer Slide Time: 08:14)



Utting et al 13 Study [UTM+13]

"Complete the implementation of a 24-hour clock by implementing methods for advancing time, adding and subtracting time and comparing time."

- 418 students with at least one programming course.
- Some students were given "test harness", i.e. a code that tests correctness.
- Students with test harness completed more than 3 out of 4 methods.
- Students without test harness completed less than 1.

Much easier than previous studies, yet performance is not great.

Performance improves if we ask for small pieces of code.

Performance improves if we provide test harness.

Another study was done by Utting et al, another group and this was of a different kind. Their problem was as follows. Complete the implementation of a 24-hour clock by implementing methods for advancing time, adding and subtracting time and comparing time, 418 students were surveyed and all of them had done at least one programing course. Some students were given a test harness.

What is this? Well, this is code that you can use to check whether your answer is right. So you came up with a program and then the code would give it some input and check whether your program works correctly on the input. So in other words you get the benefit of somebody telling you that okay on this input, your output is right. On this input your output is wrong and now you can go back and change your program.

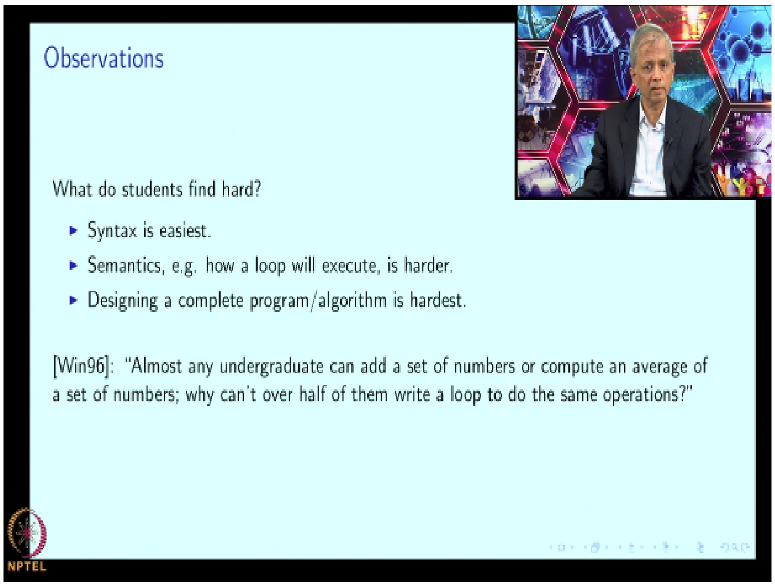
So students with test harness completed on the average more than 3 out of the 4 methods that were to be done. So that is good news okay. Students without test harness; however, completed less than 1. So this is not really very encouraging. So it presumably means that the students have difficulty in understanding the statement of the problem. Once they are given a few examples and once somebody looks over there back and looks at their work and say this is right, this is wrong then they can go ahead and finish whatever is given to them.

Note that this problem is much easier than the previous studies. It is much easier in the sense that you are asked to write 4 small pieces of code and the code is doing something very, very specific. There is hardly any possibility of looping even. So it is doing a minimal testing. It is actually, the test is more about whether you see that these methods that you are asked are part of global of a bigger picture.

You are not supposed to be able to solve a complete problem, but you are supposed to sort of fit, see what fits in a large problem and I am inclined to think that even though this is much easier than the previous study, somehow the performance is not that great and of course the harness, okay, so it is better than the previous study, so it means that if we ask for small pieces of code performance improves and of course with the harness it improves even more, okay.

So that indicates perhaps some lack of comprehension which the harness is correcting. So some observations. What do students find hard?

(Refer Slide Time: 11:30)


Observations

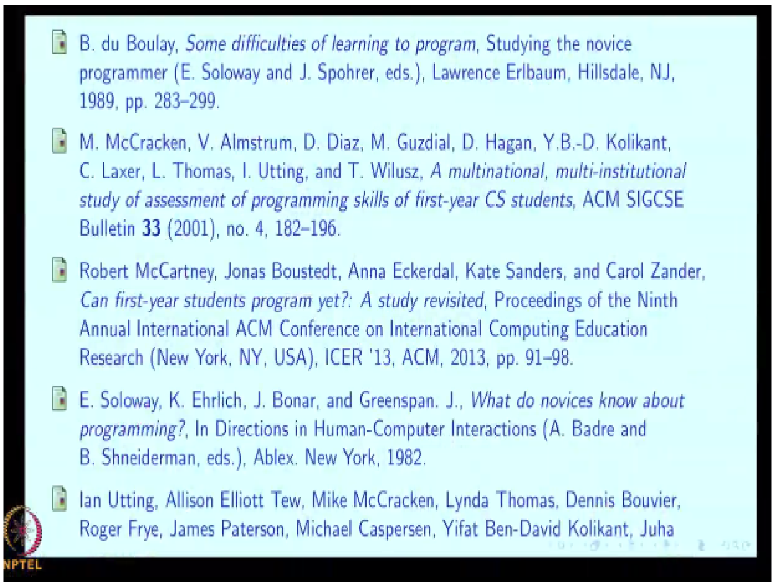
What do students find hard?

- Syntax is easiest.
- Semantics, e.g. how a loop will execute, is harder.
- Designing a complete program/algorithm is hardest.

[Win96]: "Almost any undergraduate can add a set of numbers or compute an average of a set of numbers; why can't over half of them write a loop to do the same operations?"

So summary perhaps is that syntax is relatively easy. Semantics that is how a loop will execute is harder and designing a complete program or an algorithm is hardest. Many students are not really getting this or having great difficulty with this. This sentiment is echoed by Winslow in somewhat old paper and Winslow says almost any undergraduate can add a set of numbers or compute an average of a set of numbers; why cannot over half of them write a loop to do the same operation?

(Refer Slide Time: 12:26)



- B. du Boulay, *Some difficulties of learning to program*, Studying the novice programmer (E. Soloway and J. Spohrer, eds.), Lawrence Erlbaum, Hillsdale, NJ, 1989, pp. 283–299.
- M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y.B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, *A multinational, multi-institutional study of assessment of programming skills of first-year CS students*, ACM SIGCSE Bulletin **33** (2001), no. 4, 182–196.
- Robert McCartney, Jonas Boustedt, Anna Eckerdal, Kate Sanders, and Carol Zander, *Can first-year students program yet?: A study revisited*, Proceedings of the Ninth Annual International ACM Conference on International Computing Education Research (New York, NY, USA), ICER '13, ACM, 2013, pp. 91–98.
- E. Soloway, K. Ehrlich, J. Bonar, and Greenspan. J., *What do novices know about programming?*, In Directions in Human-Computer Interactions (A. Badre and B. Shneiderman, eds.), Ablex. New York, 1982.
- Ian Utting, Allison Elliott Tew, Mike McCracken, Lynda Thomas, Dennis Bouvier, Roger Frye, James Paterson, Michael Caspersen, Yifat Ben-David Kolikant, Juha

Okay, so with that we will conclude this part and again here are the references. I will encourage you to look at the references and again as I said if you cannot get to the references let me know and I will try to put them up. We will stop here.