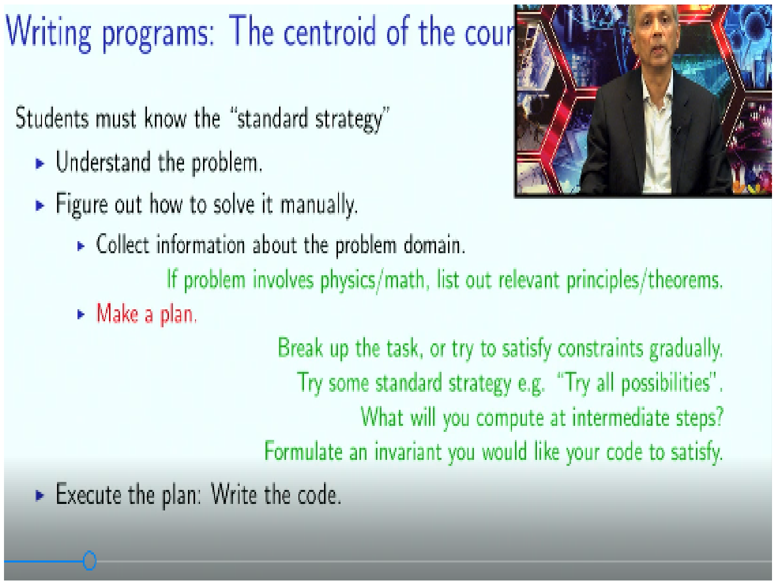
Design and Pedagogy of The Introductory Programming Course

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

Lecture – 11
Basic Ideas In Our Approach.6: Remarks on individual topics -2, Conclusion

Welcome back we have been talking about the design of the introductory programming codes. We discussed it at the level of outcomes and topics and now you had filling in some details. So, now were going to look at exactly what we expect to be talked when we talk about programming or writing programs.

(Refer Slide Time: 00:41)



Writing programs: The centroid of the cour

Students must know the "standard strategy"

- Understand the problem.
- Figure out how to solve it manually.
  - Collect information about the problem domain.
    If problem involves physics/math, list out relevant principles/theorems.
  - Make a plan.
    Break up the task, or try to satisfy constraints gradually.
    Try some standard strategy e.g. "Try all possibilities".
    What will you compute at intermediate steps?
    Formulate an invariant you would like your code to satisfy.
- Execute the plan: Write the code.

So, writing programs is really the focus of the centroid of the course and so what exactly do we want here. So, students must somehow or the other not explicitly as such but in any time you talk about programs you should talk about. the standard strategy, what is the standard strategy understand the problem. Okay understand it understand the problem we said means device thing input input instances and corresponding values.

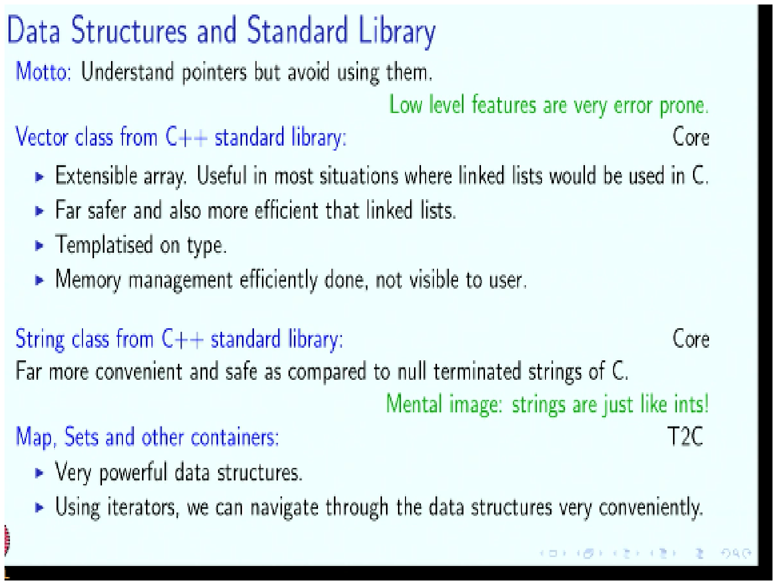
So, what value is needed for what input so at this level the students must understand the problem they must be able to figure this out on their own. Then they should figure out how to solve it manually. Do not even get to a computer unless you know how to solve the problem on your own so forth of this they may need to collect some information about the problem. So, the problem about numbers about digits of a number about physics.

Whatever it is the student has to maybe shift gears a little in hand and go into a mode and when he or she remember that what he or she has learned about these topics. So then the student has to make a plan now here in general it might mean something like break up the task or try to satisfy constraints gradually. But more often it might mean that the students already know how to solve the problem manually or it is a relatively simple thing in this course.

We have not been expecting students to invent very very sophisticated algorithms. So, the student will probably do the step fairly fast okay maybe trying some standard strategy such as try all the possibilities. Okay but what the student has to see is what are you going to compute at intermediate steps. So, just as for the problem of calculating the value of e we say back in the iteration were going to calculate them in the I iteration we are going to calculate 1 over i factor.

So, when i say make a plan and here i really mean this describe what you are going to do in your iterations and write it down. Okay so formulate invariant that you would like your code to satisfy and only then try to write the code okay. So, execute the plan which means write the code review the program and see if it can be improved yes okay. so that is what i want to state about writing programs to bill have a lot more discussion of this when we discuss pedagogy.

(Refer Slide Time: 03:43)



Data Structures and Standard Library

Motto: Understand pointers but avoid using them.

Low level features are very error prone.

Vector class from C++ standard library: Core

- Extensible array. Useful in most situations where linked lists would be used in C.
- Far safer and also more efficient that linked lists.
- Templatised on type.
- Memory management efficiently done, not visible to user.

String class from C++ standard library: Core

Far more convenient and safe as compared to null terminated strings of C.

Mental image: strings are just like ints!

Map, Sets and other containers: T2C

- Very powerful data structures.
- Using iterators, we can navigate through the data structures very conveniently.

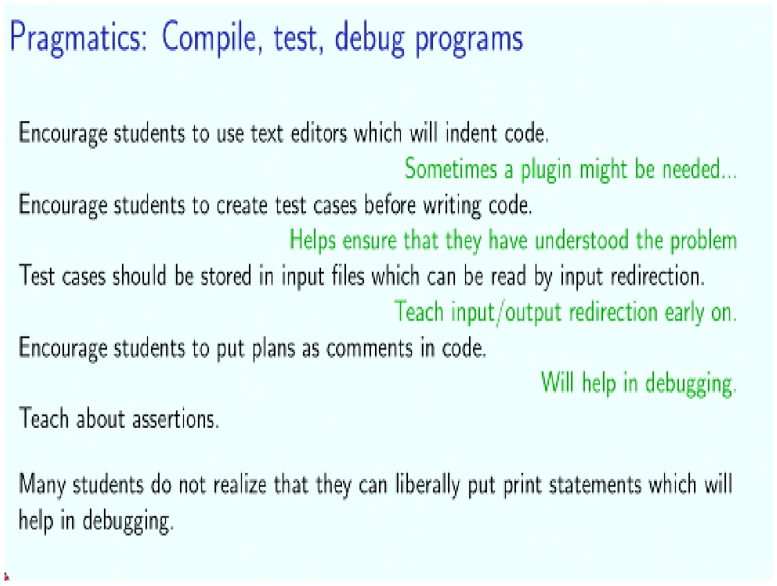
Data structures and standard library so the motto here is understanding pointers but avoid using them, low level features are very error prone and hence this warning. So, vector class from C++ standard library should be taught in my opinion. So, it is a class which is almost core i would say there are three in the core but of course you can do without it. C programmers do without it but it is such a great class that such an easy class to use as well.

That you really should construct consider it in core. Extensible array useful in most situations not only where arrays are needed but even linked lists may be used in C and you get it for free essentially. You do not need to write any core to implement a vector it is far safer and much more efficient linked lists. It is templatized on type so which means it works for all types. It does memory management on its own its not visible to you.

So, you get something really clean and elegant. String class from C++ library is similar okay. It is far convenient and safe as compared to null terminated strings of C and allow this to have this mental image that strings just like ints they may be long but they are there. They are there in where I mean it does not really hurt if you think of the strings as residing in your activation frame, they actually do not reside in the activation frame.

But you can think of them as a redesigning activation frame just as you can think of ints as residing in your activation frames. Bap tests and containers are other powerful container structures from the standard library but those are two core and they had actually very powerful they have this notion of iterators which enable you to navigate through data structures very conveniently and they are discussed all of these are discussed in chapter 22 of our textbook.

(Refer Slide Time: 06:02)



Pragmatics: Compile, test, debug programs

Encourage students to use text editors which will indent code.
Sometimes a plugin might be needed...
Encourage students to create test cases before writing code.
Helps ensure that they have understood the problem
Test cases should be stored in input files which can be read by input redirection.
Teach input/output redirection early on.
Encourage students to put plans as comments in code.
Will help in debugging.
Teach about assertions.

Many students do not realize that they can liberally put print statements which will help in debugging.

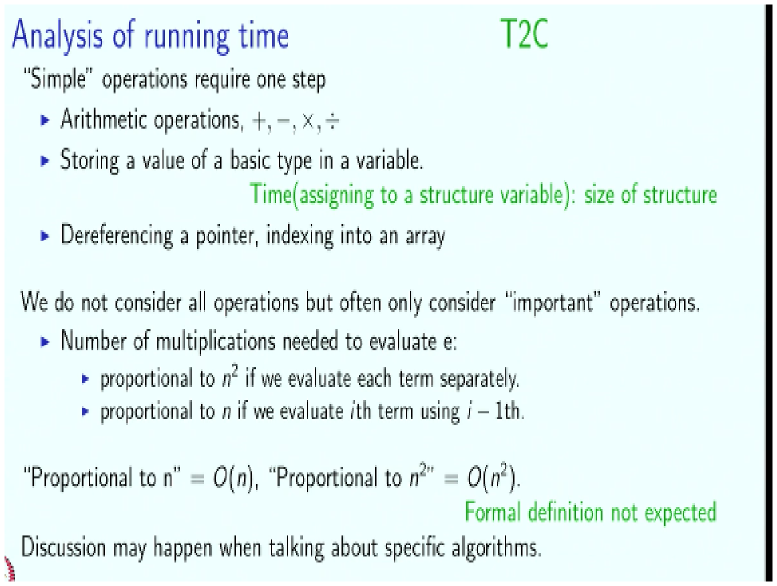
Compiling testing and debugging programs is an important skill expressed rating and then for very important to assure your students that it that it is something that everybody does and everybody does comfortably well with practice. Okay first of all encourage students to use text editors which will indent code. If your code is not indented, it is very hard to spot mistakes you may think that a block is ending at a certain place whereas it ends at a different place.

And therefore no wonder your code is doing something wrong okay and your editor may not immediately have the indentation feature plug in might be needed. So, what you should do as a teacher is to get information about this and tell your students that look if you are using this editor this plug in is there and use it. hit as you did this blog instead and use it. You should encourage students to create test cases before writing code.

Part of the reason is that students should understand the specifications first and that they create this case this is proof that they have understood specifications. It is shocking that even experienced programmers might start writing code before they fully appreciate all the intricacies of the test of the specification that are given to them okay and test cases should be stored in input files which can be read by input redirection.

So, you should teach input output redirection somewhat early on. Encourage students to put plans as comments in code because they would have been debugging okay and of course it would mean that they first of all have a plan in place. Teach about assertions now many students do not realize they can liberally put print statements which will help in debugging. It is obvious but they do not realize. So, tell them.

(Refer Slide Time: 08:17)



Analysis of running time T2C

"Simple" operations require one step

- Arithmetic operations, $+, -, \times, \div$
- Storing a value of a basic type in a variable.

Time(assigning to a structure variable): size of structure

- Dereferencing a pointer, indexing into an array

We do not consider all operations but often only consider "important" operations.

- Number of multiplications needed to evaluate e:
  - proportional to n^2 if we evaluate each term separately.
  - proportional to n if we evaluate ith term using $i - 1$th.

"Proportional to n" $= O(n)$, "Proportional to n^2" $= O(n^2)$.

Formal definition not expected

Discussion may happen when talking about specific algorithms.

Analysis of running time. in my opinion TA2 core as we say is it is core so let us talk about it a little. So, a simple operation is required one step that is basically the guiding principle arithmetic operations require one step but they do not all require the same amount of time not one step but for practical purposes of this course it is convenient to think of them as one step similarly storing a variable storing a value into a basic tape variable int, double float takes one step ahead.

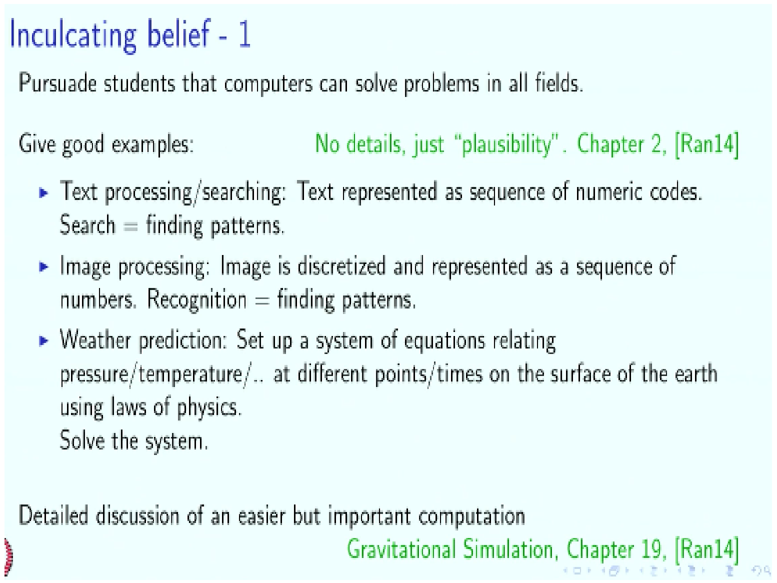
So, chorally of this is time assigning to a structure variable is proportional to the size of the structure. Dereferencing a pointer takes constant time indexing into and it takes constant time one step think of it as one step. So. the second idea in analyzing running time is often not to worry about every operations operation but worry about important operations. So, if you are evaluating a series you may be asking look how many multiplications am i doing.

Because those are the dominant operations or multiplications or divisions whatever. So, you might see that the number of operations is proportional to a square if you evaluate each term separately or it might be that they are proportionate to n if we evaluate the ith term using the i-1th. So, i would recommend that when you teach how to evaluate e you do a quick calculation you tell students how many divisions they did or multiplication of or whatever.

And do not really dwell on this too carefully you should say that proportional to n square can be written as a short form as n but again o often has a formal meaning it is too early to talk about the formula. Okay so proportional to n can be written as O of n and proportional to n square can be written as O of n square do not talk about the formula. So, this is basically the extent to which running time should be analyzed in this course.

And the discussion may happen while talking about specific algorithms you probably do not need to. devote a lecture to this or anything.

(Refer Slide Time: 10:58)



Inculcating belief - 1

Pursuade students that computers can solve problems in all fields.

Give good examples: No details, just "plausibility". Chapter 2, [Ran14]

- Text processing/searching: Text represented as sequence of numeric codes. Search = finding patterns.
- Image processing: Image is discretized and represented as a sequence of numbers. Recognition = finding patterns.
- Weather prediction: Set up a system of equations relating pressure/temperature/.. at different points/times on the surface of the earth using laws of physics. Solve the system.

Detailed discussion of an easier but important computation
Gravitational Simulation, Chapter 19, [Ran14]

So, finally how to be inculcate belief what exactly what part of you hoping to accomplish. So, we want to persuade students that computers can solve problems in all fields. So, we have to give good examples for it okay we do not give details just have to sound right or we should be that look students will think oh this seems reasonable we have to sound reasonable. So, text processing searching if you want to say that computers can do it.

Then you can say that look test is represented as a sequence of numeric codes. So, now if you want to search we are searching the numbers we are comparing numbers and therefore. Students can believe that oh yes text processing or text searching can be done. So, searches finding patterns image processing the images discretized and represented sequence of numbers again recognizing things like finding patterns.

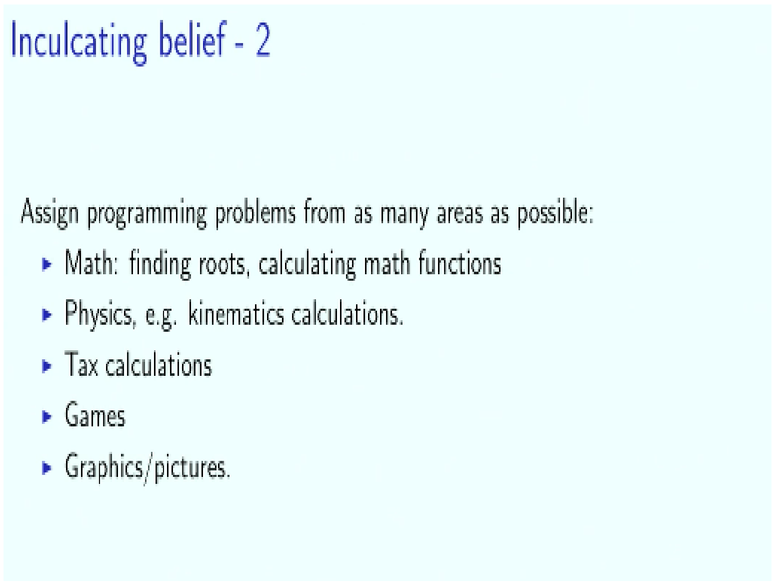
More complicated example is weather prediction how does that happen now this is a subject in itself we cannot explain all of that. But I feel we should explain something here is what we should explain so we should say that to do weather prediction we are going to set up a system of the creations and our unknowns will be pressure temperature humidity whatever different points or times on the surface of the earth and we will relate them using laws of physics.

So, if we know the initial conditions we will be able to calculate the values for subsequent time steps this is about it we cannot say what laws we are going to use because those laws are a science by themselves. But should state at least this much so this is going to persuade the student yes. it could be predicted okay and that is going to improve his respect for our subject. Okay we set up the systems we solve the system.

So, then we should do a detailed discussion of an easier but important computation so we cannot talk about weather it is too complicated but we can and we very we should talk about simpler physical systems. So, for example the gravitational situation. everybody laws newtons law of gravitation everybody knows newtons laws of motions motion and using those we can in fact predict reasonably well how say planets in a solar system move.

And that calculation is one of the classical calculations that people do astronomers do this conclusions calculation all the time and we can describe it we can describe it we can get students to programming this is something that will increase their belief in the utility of computers and this is discussed in chapter 19.

(Refer Slide Time: 14:19)



Inculcating belief - 2

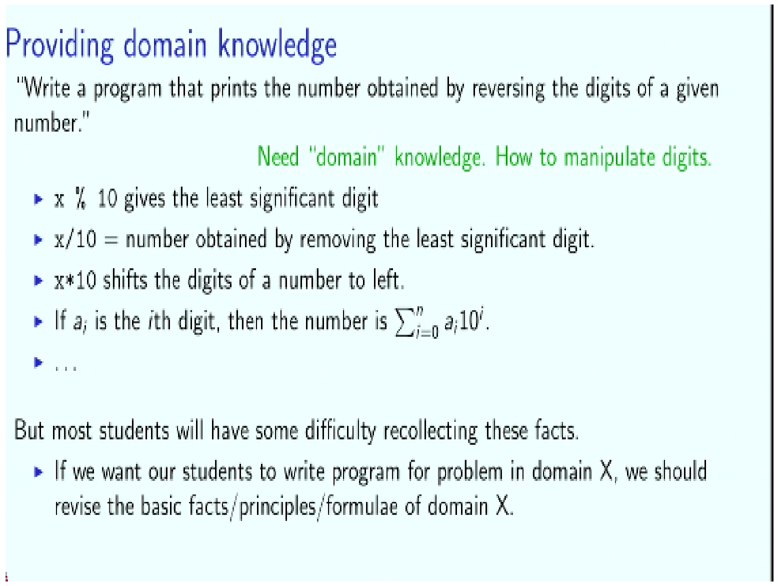
Assign programming problems from as many areas as possible:

- Math: finding roots, calculating math functions
- Physics, e.g. kinematics calculations.
- Tax calculations
- Games
- Graphics/pictures.

To inculcate belief, we should assign programming problems from many areas as possible Math finding roots calculating that functions physics, kinematics calculations that have additional calculations which we just said. Tax calculations games can be programmed graphics and pictures. So, computers are good with numbers that students should be persuaded like computers are also good with pictures and animations.

This will needy some additional code what is called scaffolding code and we should provide it okay. The more areas we can bring in the stronger will be the belief.

(Refer Slide Time: 14:35)



Providing domain knowledge

"Write a program that prints the number obtained by reversing the digits of a given number."

Need "domain" knowledge. How to manipulate digits.

- x % 10 gives the least significant digit
- x/10 = number obtained by removing the least significant digit.
- x*10 shifts the digits of a number to left.
- If a_i is the ith digit, then the number is $\sum_{i=0}^{n} a_i 10^i$.
- ...

But most students will have some difficulty recollecting these facts.

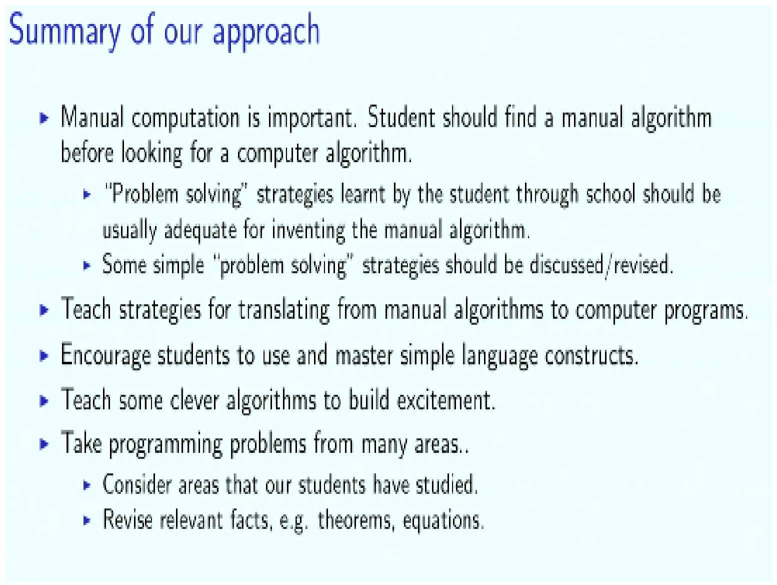
- If we want our students to write program for problem in domain X, we should revise the basic facts/principles/formulae of domain X.

We should provide the domain knowledge whenever it is needed. So, if you say write a program that prints the number of 10 by reversing the digits of a given number. So, we need them domain knowledge how to manipulate digits so for example students need to know that x/10 gives the least significant digit. x/10 gives the number of 10. x time shifts the digits of a number to the left.

And if I want to get the number the value of a number itself I can use this formula to calculate it given the digits Now these facts are expected to be known by students after all they are taught maybe in primary school. But students may have some difficulty so remind them be nice to them and remind them okay in general if you want our students to write programs for solving a problem in domain x.

We should revise the basic facts principles formally of domain x. Very quickly we do not have teach them we just have to remind them okay. Revision could happen in class by solving another problem front from their domain or by a giving a set of printed notes.

(Refer Slide Time: 16:04)



Summary of our approach

- Manual computation is important. Student should find a manual algorithm before looking for a computer algorithm.
  - "Problem solving" strategies learnt by the student through school should be usually adequate for inventing the manual algorithm.
  - Some simple "problem solving" strategies should be discussed/revised.
- Teach strategies for translating from manual algorithms to computer programs.
- Encourage students to use and master simple language constructs.
- Teach some clever algorithms to build excitement.
- Take programming problems from many areas..
  - Consider areas that our students have studied.
  - Revise relevant facts, e.g. theorems, equations.

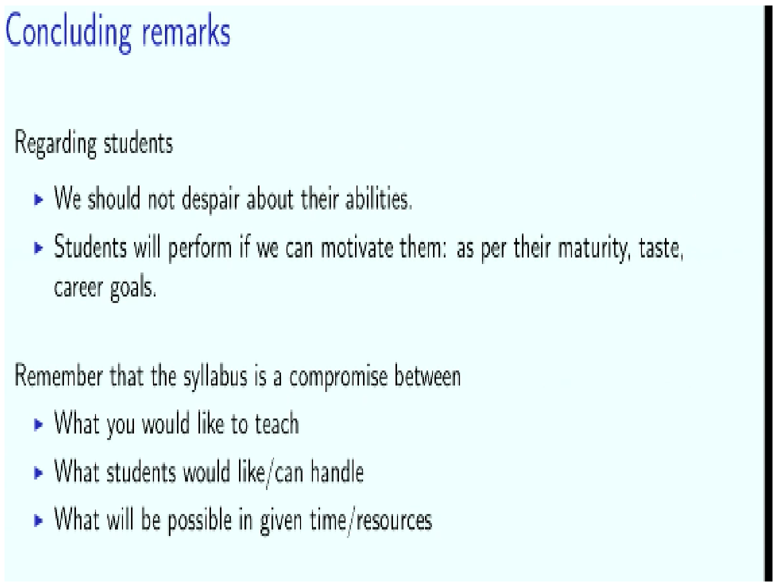
Okay so we now come to the conclusion of the sequence of lectures. So, i want to summarize so what is our approach about so first we are going to stress manual computation. Manual computation is important. Students should first find a manual algorithm before finding a computer algorithm before writing a computer program. Problem solving strategy learned by the student through school should usually be adequate for inventing the manual algorithm.

We are not going to teach problem solving strategies we might just do some fairly simple commonsense things but elaborate problems solving strategies there is no time in this course. Now what we do teach is teach strategies for translating from manual algorithms to computer algorithms to computer programs okay. Then we are going to encourage students to use and master simple language constructs. We will teach some clever algorithms to build excitement.

We will take programming problems from many areas okay we consider areas that our students have studied. and want to study which are consistent with their career goals. We are going to revise the relevant facts. We gave you a detailed list of topics that we are going to cover in our syllabus. We also talked about what exactly we meant and what exactly we intended to teach for those topics,

So, in the next lectures we are going to talk about Pedagogy how exactly we will do the teaching okay. Yes, we also talked about talk about compiling testing debugging this can be frustrating and so we should hand hold as much as possible.

(Refer Slide Time: 18:37)



Concluding remarks

Regarding students

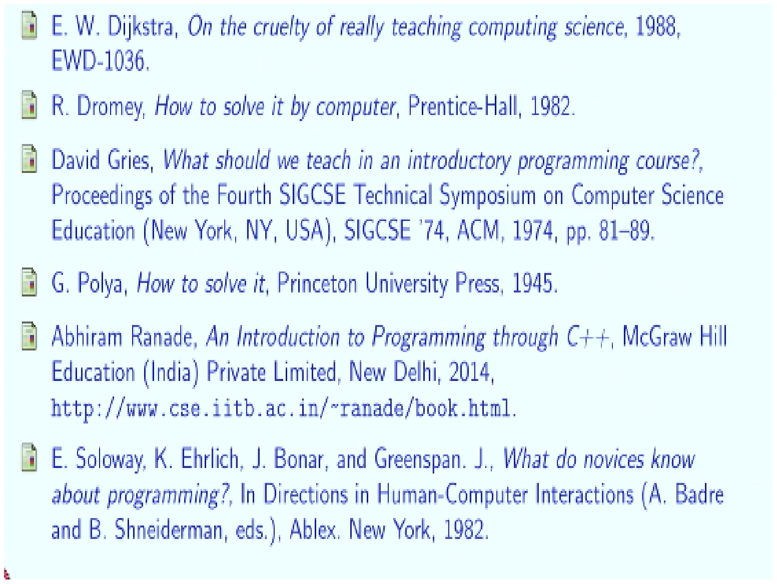
- We should not despair about their abilities.
- Students will perform if we can motivate them: as per their maturity, taste, career goals.

Remember that the syllabus is a compromise between

- What you would like to teach
- What students would like/can handle
- What will be possible in given time/resources

So, final remarks regarding students we should not despair about their capabilities students while they perform if we can motivate them as better than maturity taste and career grows. But you should remember that the syllabus is a compromise between what you would like to teach what students would like or can do and what will be possible in the given time and resource. So, we need to iterate over it.

(Refer Slide Time: 19:05)



- E. W. Dijkstra, *On the cruelty of really teaching computing science*, 1988, EWD-1036.
- R. Dromey, *How to solve it by computer*, Prentice-Hall, 1982.
- David Gries, *What should we teach in an introductory programming course?*, Proceedings of the Fourth SIGCSE Technical Symposium on Computer Science Education (New York, NY, USA), SIGCSE '74, ACM, 1974, pp. 81–89.
- G. Polya, *How to solve it*, Princeton University Press, 1945.
- Abhiram Ranade, *An Introduction to Programming through C++*, McGraw Hill Education (India) Private Limited, New Delhi, 2014, `http://www.cse.iitb.ac.in/~ranade/book.html`.
- 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.

So, these are the references and there is more I will not go through the references but it will stop over here. In the next lecture, we will talk about how to teach all of this or the pedagogy of the course. Thank you.