

Design and Pedagogy of The Introductory Programming Course
Prof. Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology – Bombay

Lecture - 15
Pedagogy.4: Tour – 2

Hello and welcome back. We have been talking about the pedagogy of the introductory programming course and we were discussing how we should teach the introductory few lectures. So I want to talk about what should be we telling the students about computers as an introduction. So let me begin by saying, what I think we should tell students about how computer solve problems.


(Refer Slide Time: 00:49)

Topic 1 b: How computers solve problems

- ▶ We would like students to believe that computers are
- ▶ Some interesting computations should be explained at
Candidate examples: Image processing, Text processing, weather prediction.

Key ideas to be explained: Chapter 2, [Ran14]

- ▶ "Everything must be represented as numbers"!
Numeric codes for text, discretization and intensity values
- ▶ "Next apply laws of physics, set up equations, solve..."
- ▶ (Manual) Algorithm = Precise description of what calculations are needed.
"You already know algorithms: primary school arithmetic
- ▶ Assurance that a computer can perform all the steps that are used in manual algorithms.
Arithmetic, conditional execution, looping



So we want our students to believe that computers are very useful. So what we should do if we should show them some interesting computations and we should explain to them how computers work on those computations at a very high level sort of like a popular science level. So some examples that we could to, an Image processing, Text processing may be Weather prediction these are my favourite examples. You could have your own and you could develop your own as well.

But these examples are also developed in the book that we have been mentioning. Okay, so what should we be explaining about these example, okay. So first of all we should say that when a

computer solves the problem everything has to be represented as a number. So whatever questions asked it should phrase in terms of numbers, okay.

So for example, if you are talking about text then we are going to represent text by numbers or in other word for every letter we are going to have a numerical code which as we know is the ASCII code. If you want to do image processing again we should first convert our image into numbers. How do we do that? Well, we divide the image into small squares or pixels if you will, and for each pixel we associate a number which is equal to the intensity of light at that pixel.

If it is a coloured image then whose we will have to talk about intensities, the red blue and green intensities. But the point is that information about colours and parts of images can be represented in terms of number and that is what you need to do if you want to process numbers and pictures. Now about, Weather prediction, okay we can say something like the following, that again here we are going to discretize.

So we are going to take the map of the world and we are going to put points on it and we are going to say let us put variable for different points, okay. So once you express everything as a number we can apply Laws of Physics or somehow get the relationship between numbers and ask questions about the numbers, or we can say here are the questions that these numbers at better satisfy and so we solve them, okay.

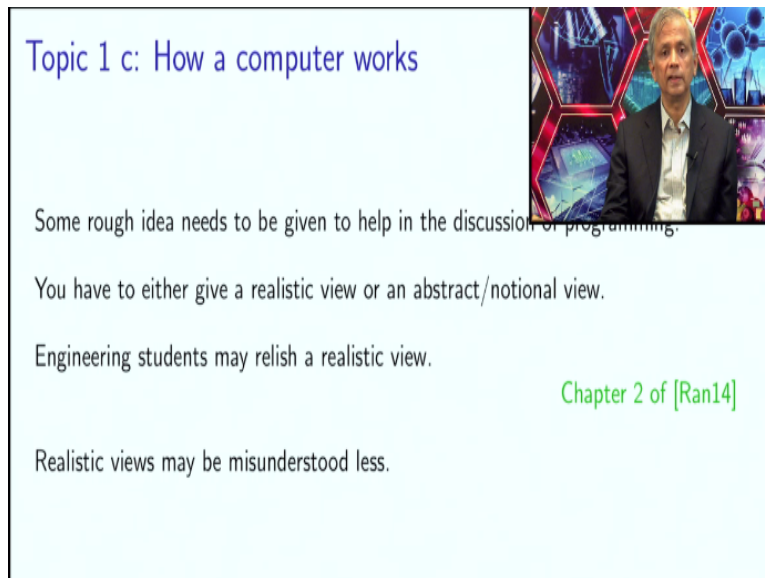
Now, what is an algorithm? Well, an algorithm is a precise description of what calculations are needed. And right now I am talking about a manual algorithm. And this will be say English language descriptions of what do we do with the numbers that we just constructed. Okay, so maybe we are setting up equations we are solving them whatever it is, but that is what we have to say.

Here we could say that look, we should give an introduction to algorithms and what the term algorithm means as well and not only that; we should say that look you really need you really know algorithms. So right now write then in the very first or second lecture I suppose we should

tell the students that this is what an algorithm is and that you already know algorithms. Primary School arithmetic is nothing but an algorithm.

And then we need to assure students that a computer can perform all the steps that are used in manual logarithms, and so whatever manual algorithm you have we can somehow feed to a computer as well. So if in your manual algorithm you are doing some arithmetic your computer will be able to do. If in your manual algorithm you say that okay, if this is 0 then do this; if it is 0 do not do this. We will be able to say all that in the instructions that we give to the computer, okay.

(Refer Slide Time: 05:18)



Topic 1 c: How a computer works

Some rough idea needs to be given to help in the discussion of programming.

You have to either give a realistic view or an abstract/notional view.

Engineering students may relish a realistic view.

Chapter 2 of [Ran14]

Realistic views may be misunderstood less.

Alright so continuing along that we also need to tell a little bit about how a computer works, okay to the extent that it helps in programming, okay. So by this any insights of computer, so we have to say something like a computer contains memory; it contains some logic hardware again at a fairly high level. Now if you could say, you could use anthropomorphic term that a computer is like a dumb human being or something like that.

But—or you could say you can try to give a few which is more realistic, okay. Engineering students, I think will like a realistic view if they can understand it. And in fact that is what is done and that is what I recommended and it is done in this in the Chapter 2 of this book. If you give a realistic view the chance of misunderstanding, it is less.

(Refer Slide Time: 06:29)

Realistic and notional views of how computers work

Notional view: "A computer circuit understands C++ program"
Anthropomorphic view suggests computer will understand C++ program

Realistic view:


- ▶ C++ program is translated into a machine language program by a compiler.
- ▶ Machine language program is a sequence of numbers representing actions that the computer must perform.

Example: 70 30 31 32 might mean "Add the contents of locations 30 and 31 and place the result in location 32".

Example: 80 30 31 32 might mean "Multiply the contents of locations 30 and 31 and place the result in location 32".

Students like knowing this.
Also, it reinforces the idea that a computer will only do what it is told.

No harm in saying in addition that "your program must contain every action you want performed, in exactly the order you want it performed."



Okay, so how do computers work? So a notional view, okay it could be that a computer circuit understand C++ program. The difficulty with this is that now students tend to think of computers as human beings, and they associate human attributes with computers. So human beings have common sense, they start thinking that computers have common sense. So you have to be very careful with notional views.

You have to somehow give the feeling that they are computers are like human being but they are also not like human beings, and this is a little difficult. On the other hand, here is what a realistic view might be, and it is discussed again in the Chapter. You could say that a C++ program is translated into a machine language program by a compiler. After all you are going to tell them what compilation is, so you might as well tell them that look the output of the compilation is machine language.

When you could say that a machine language program is a sequence of numbers representing actions that the computer must perform. So you might say, you might ask me; "Look, do not we talk about machine language in the Computer Architecture course or Computer Organisation course which comes later." Of course you do, of course you talk at that point. But here it is just going to be something in principal at a very high level.

And the reason for doing it is to have some clarity-- the details are not important; the reason for that is to have some clarity about how a computer executes. So this will say this will clearly tell the student that a computer is not like a human being okay. So it has to have this very peculiar input which consists of numbers. So you could say for example that 70 30 31 32 could mean "Add the contents of 30 and 31 and place the result in 32". So 70 somehow means addition.

Another example, 80 30 31 32 might mean multiply, so instead of 70 we have 80 and that changes, the add to the multiply. So this is something that students actually enjoy, okay. So note that these are Engineering and Science students that we are talking about primarily, but even other students will enjoy this. They like course so they will understand and they will have fun with this.

But the more important part—of course, is important but there is also a more important idea, and that Idea is that a computer needs to be told in a very specific kind of a manager what we expected to do. Okay. So the moment you start with these kinds of course then students understand that look the course are important the course are -- I mean they are not going to come out of thin air, so somehow you have to do something which will cause these course to go.

Now in addition of course you should say that your program must contain every action that you want performed in exactly the order in which you want to perform, okay. But the moment you say that instructions are of these numerical types you are sort of telling the students that look, the computer need to get very precise information told to it. Okay. So let us continue this discussion with between the two ways of describing.

(Refer Slide Time: 10:35)

Realistic and notional view of a variable

Realistic view: When you write "int xyz;" you typically get 32 capacitors in your computer memory for storing the value of xyz.

Notional view: "A variable is like a box into which you place the value." This can give rise to many misconceptions. [dB89, Koh17]

- ▶ A variable can contain several values simultaneously. Several slips of paper.
- ▶ When you execute $x = y$; the value *moves* from y to x. The paper moves..
- ▶ A variable can contain a formula or an unevaluated expression.

"The number of children is the number of girls plus the number of boys. How many children are there if there are seven girls and five boys?" [Sim11]

```
int children, boys, girls;
children = boys + girls;    // children "box" gets formula
boys = 5; girls = 7;
```

Misconceptions less likely with capacitors..

So if I want to talk about a variable there are there is a realistic view which says that, when you write `int xyz` what happens is that you get 32 capacitors in your computer memory for storing the value of xyz, okay. So note down I am not saying you always get 32 capacitors but you typically get 32 capacitors. These days saying you always get 32 you do capacitors is not wrong because actually your computer memory does consist of capacitors almost entirely.

So-- but you can play it safe and you can say that no, no it could be a little bit slightly different once in a while, but this is actually what happens that xyz becomes the name that you give to a portion of your memory and int is typically 32 bits long and so you get 32 capacitors. What is the notional view? You could say something like, a variable is like a box into which you place the value. Now this is a fine description but it is (()) (11:41) to misunderstanding, okay.

And this is being studied by many researchers, so the references are at the end of the slides. And-- for example, a variable can be thought of as containing several values simultaneously, why? Because if it is a box—well, I can put several slips of paper in it.

So if I write $A=1$ and then I write $A=2$, well maybe the second slip of paper got in, okay. When you execute $x=y$ the imagery that perhaps this conjures up is that values moves from y to x. Actually, that is not what happens. Right? Nothing goes out of wide-- the value in wide is

copied. So if it is capacitors then the notion that the values sort of moves is harden to imagine, because you will say that why should the capacitors relating to why; why should they change?

Okay, so that would not be the natural way of thinking about it. So here you could say that “Oh the paper moves.” And this is another interesting misconception that students can create, which is that a variable can contain not just the value but it can contain a formula or an unevaluated expression, after all it is a box into which you are going to place a paper on which a value is return. Why no place a formula?

And here is a more detailed example that one of these authors discusses. So suppose you give this information to students, the number of children is the number of girls + the number of boys. How many children are there if there 7 girls or 5 boys? Okay, actually it is a different author. So the way you represent this information is you would say n children boys/girls so okay. So the number of children is an integer; the number of boys is an integer; the number of girls is an integer.

The number of children is the number of girls + number of boys, okay. So children=boys + girls, okay. Now, this is not a good statement. The boys and girls have to give values first and only then can this statement make sense. But students to write this statement because it appears first in the order in the English language description, and here they think that “Oh, children, the box corresponding to the children they will get the formula boys + girls.”

And later on we can write boys=5 and girls=7. And now that box, that the formula in the box will be modified, okay. Now all these misconceptions are very unlikely or less likely I should say with capacitor. So clearly, if there are capacitors and if you are told your students at each capacitor stores are 0 or 1 certainly formula will not going there, okay. So basically the point is that we have we have students who have studied some amount of science to understand what a capacitor is.

And to them you should tell that a memory is made up of capacitors; this will satisfy their curiosity but it will also reduce the likelihood of misunderstanding.

(Refer Slide Time: 15:18)

Remarks


Homeworks:

- ▶ "Describe in words how you multiply n digit numbers"
Student must say "in i th phase"
- ▶ What happens if you execute instruction 70 30 30 30?
Square calculated
- ▶ What happens if you execute program 70 30 30 30 70 30 30 30?
Fourth power calculated

Don't expect students to remember what 70, 80 means.
The homework should reinforce idea: computers work "mechanically"

But what do you do in programming labs?
Drawing programs can be asked!

Just using repeat, left, right it is possible to draw very interesting intricate pictures.



Okay, so what kind of homeworks can you give on this topic in the introductory topic in which you describe how computers work. Well you could say, "Describe in words how you multiply n digit numbers." So you have assured in first part of this, that I you have told student that multiplication is an algorithm, so now you ask them to that out, how exactly do you multiply. So hear you should tell the student that you should expect to think about how many phases there are what happens in the i th phase, are there nested phases, okay and you should write that term.

Why? Because that description is going to help them in writing a program to multiply very large numbers; you can tell them that if you have very large numbers computers will typically multiply them exactly using the high school method. You could ask, what happens if you execute the instruction 70 30 30 30, okay. Here of course this should come immediately after you tell them what 70 30 30 30 means. So it is not a test of memory, okay.

But here, what is going? Here a square is being calculated. So you could ask, what if you execute this two instructions one after another. So you square twice-- well, so what happens? The fourth power gets calculated. So what you are trying to do in this is that you are saying that, what I have told you are elementary ideas and those elementary ideas can be composed and from those from the composition something bigger can be constructed.

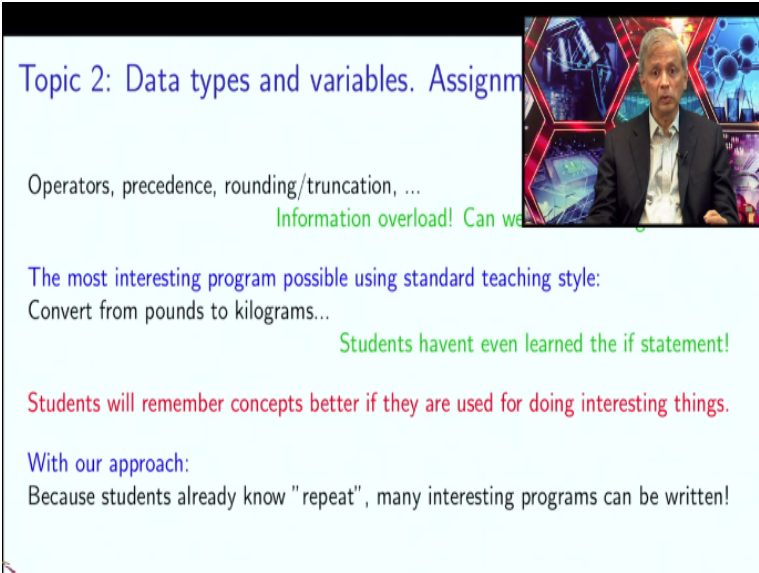
So that sort of a big message that you are sending but a smaller message that you are sending is you that you are testing; whether they understand, whether they are with you. It is not that they need to remember what 70 means, but if you just told them what 70%; if you told them this structure then they should get this; they should start thinking in this manner.

So, basically this homework should be reinforcing the idea that computers work mechanical, they do not have common sense, they have to be told everything. Now the way most of our courses work is that there is something that happens in the classroom and what I have said these homeworks could happen at home or maybe these are assignments that you give to students to check their understanding.

But then you also have programming labs, what you do then? Now, this is the time were you can ask them to do more drawing programs. You have just done turtle geometry; you have shown them what the turtle can do and now you can get them to write 200 homeworks or more complicated drawing programs; and the book certainly describes gives many ideas. But I am sure you will have many ideas.

So just using repeat left, right and repeat q over here; you can draw very interesting an intricate picture and we have seen some of those. And I am sure you will think of more such pictures.

(Refer Slide Time: 18:36)



Topic 2: Data types and variables. Assignment

Operators, precedence, rounding/truncation, ...
Information overload! Can we

The most interesting program possible using standard teaching style:
Convert from pounds to kilograms...
Students havent even learned the if statement!

Students will remember concepts better if they are used for doing interesting things.

With our approach:
Because students already know "repeat", many interesting programs can be written!

The second topic that typically get start is about Data types and variables and Assignment statements. Operator, precedence, rounding, truncation all of these things have to be taught. In some sense, there is an Information overload! So a natural question is, can we make this learning active. By active learning I means, students have to think; students have to participate somehow.

If you just keep feeding information, then you know what happens. Students may have their eyes open but they might actually be sleeping for all practical purposes, we do not want that to happen. If at this disappoint you ask students to write programs, there is not much you can do. If use the standard teaching style what is the most interesting program you can I ask. Well, you can say i way 150 pounds whatever, so what it might in kilograms?

So they reading the number they store it into a variable; they divide it by 2.2 and then they print answer, so all these things they can do because they had just learned, but nothing much more. Remember, that they do not even know the 'if' statement. However, you have to keep in mind that students will remember concepts better if they are used for doing interesting things.

If you do interesting things; if you tell good Panchatantra stories, then students will know moral principles better. If you just tell the moral principles no story nobody remembers. So you have taught them assignment statement; you have taught them very interesting ideas but they have to be used with some interesting stuff if you want them to-- if you want the students to remember them.

Now with our approach students already no repeat and therefore many interesting programs can be written. So what can you write?

(Refer Slide Time: 20:53)

Exciting things you can do with repeat + assignment



Draw more interesting pictures: Spiral

```
int i=10;
repeat(10){forward(i); right(90); i = i + 10;}
```

Calculate averages. Chapter 3, [Ran14]

```
int n; cin >> n; double sum = 0;
repeat(n){
    int x; cin >> x; sum = sum + x;
}
cout << sum/n << endl;
```

So we can draw more interesting pictures. For example, here is the code to draw spiral. So we start with $i=10$ and then 10 times `forward(i) right 90`; this is almost like drawing the square. But after each right 90 we are going to change i . So what will happen? So first time we will go forward 10 pixels then we will turn and after that we will go 20 pixels; after that we will go 30 pixels; after that we will go 40 pixels and so on.

So will not draw a square but it will draw a spiral. Now here is an important point $i=i+10$ is a very interesting statement; it is a very confusing statement as well because students have learnt math in which $=$ means one thing and $=$ means something by different over here. So if I show $i=10$ to somebody who does not know programming they will say, well, cancel i from both sides and we get $10 = 0$. So this is nonsense.

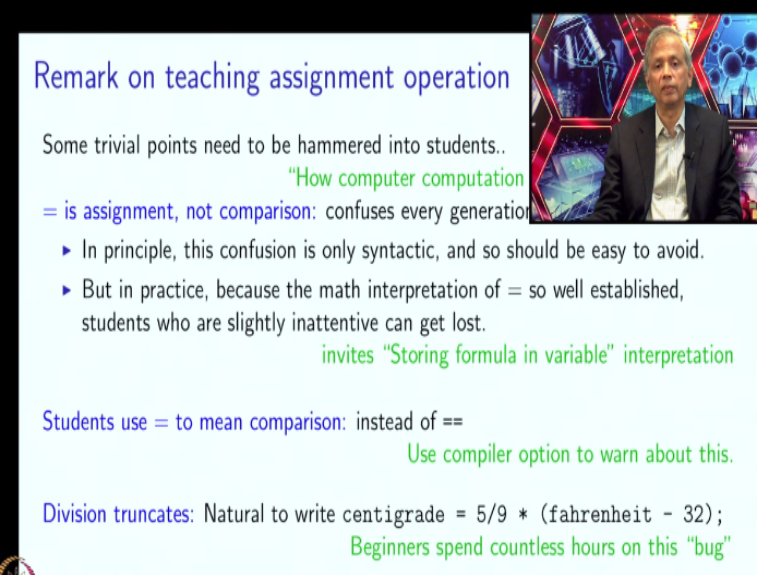
But by writing this statement we are reinforcing the distinction; we are saying that look in this figure if we want i to change in this manner we have to write the statement and what does statement does is it takes the old value of i , adds 10 to it and places it back in i , okay. So this is a very important picture to drive home the fact of self assignment or reassignment but especially dependent on the old value.

Calculate average is another example which is also been discussed in the Chapter 3 where assignment statements are discussed. So what happens over here is that we are going to read in

numbers; we are going to read in n which tells you how many numbers you are going to read. We are going to set sum to 0, and in each iteration of the repeat we are going to read a number and add it to sum and finally we are going to print out sum/n .

So course this assumes that the number of numbers that we read is greater than 0 but baring such points, this is pretty good code. And notice that it is probably the simplest code. If you use a standard looking construct you will need an additional variable to count how many iterations have happened. Over here repeat n sort of heights that, and because the code is simple I think the students have an easier time understanding this piece of core.

(Refer Slide Time: 23:49)



Remark on teaching assignment operation

Some trivial points need to be hammered into students..

"How computer computation
= is assignment, not comparison: confuses every generation"

- ▶ In principle, this confusion is only syntactic, and so should be easy to avoid.
- ▶ But in practice, because the math interpretation of = so well established, students who are slightly inattentive can get lost.
invites "Storing formula in variable" interpretation

Students use = to mean comparison: instead of ==
Use compiler option to warn about this.

Division truncates: Natural to write $centigrade = 5/9 * (fahrenheit - 32)$;
Beginners spend countless hours on this "bug"

Now as far as the assignment operations is concerned I said already that it can confuse students with the equality of mathematics. So several trivial points need to be hammered into students. Okay. And this is another example of how computer computation is different from human computation. So this is being this was one theme of – in our second lecture sequence I should say, of that computers computes in a different manner in some sense from humans.

Although and analogous, analogously different nevertheless. But here is sort of a more low level manifestation of the same principal. Specifically, equality when we talk about it might mean equality, mathematics equality okay. And when you say equality, it might mean some kind of

comparison whereas in a computer equality typically means assignment. And this confuses every student who learns programming.

Or maybe does not confuse a few lucky students but most students will get confused. Most students will write equality somewhere in their program to mean checking for equality rather than assignment and that will make their program make a mistake. So in principle you think that this is this confusion is very superficial; it is only syntactic and so should be easy to avoid. But in practice that is not so.

Because the mathematics interpretation of the equality is so well established that students who might be slightly inattentive in your lecture will get lost. So this is a point that you should say a few times. Look, I mean we have to worry about students who are slightly inattentive, right I mean-- we are not attend to you all the time let us confess it and our students are not going to be attentive all the time and so we have to ourselves pay attention and see are they attentive or not attentive and sort of compensate for inattentiveness, okay.

And because equality in mathematics equal operator can mean equality rather than assignment. If I say the number of boys + the number of girls = the number of children, okay. On one hand it is a statement of mathematical equality, but people may write it and people do write it as children = boys + girls meaning a formula and show boys + girls the formula gets stored in the box for the variable children, and so these kinds of interpretations are invited also by this equality symbol.

So again as I said earlier a students might use might not just get confused by equality but they might also use equality to mean comparison instead of using `==`. There is a compiler option which will warn you, okay. So if you write `if A=B` instead of `A==B` and use this compiler option and `S++` is configured to use is compiler option you will get a warning, okay. So use that. Another point about operators is that Division truncates.

So I might write the following statement `centigrade = 5/9 times Fahrenheit - 32`. So the value of centigrade will be set to this. Unfortunately, `5/9` will be evaluated first and that division is truncating. So many, many students are going to make this mistake unfortunately, okay. This

really I think is a language problem. The language should do something about this. In fact, Python has done this. In Python, if you write $5/9$ it does not mean 0, it means 1.8.

So the division operator will have the result as double, okay. So if you want an integer then you will have to additionally change it to an integer. And I think the way programming languages evolve; I will not be surprised if C++ also changes this at some point or another. Okay. So we are going to take a break here and when they come back we will talk about program design. Thank you.