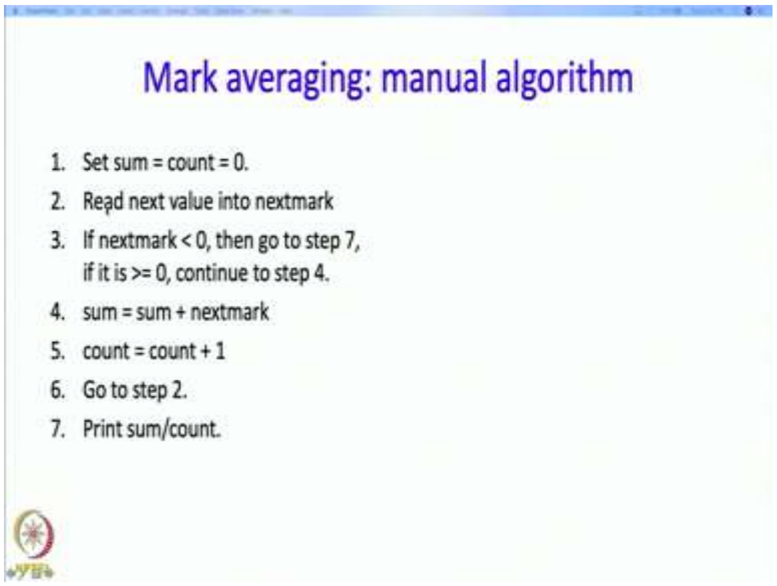
Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 7 Part – 2
Looping Statements
Mark averaging

(Refer Slide Time: 0:25)



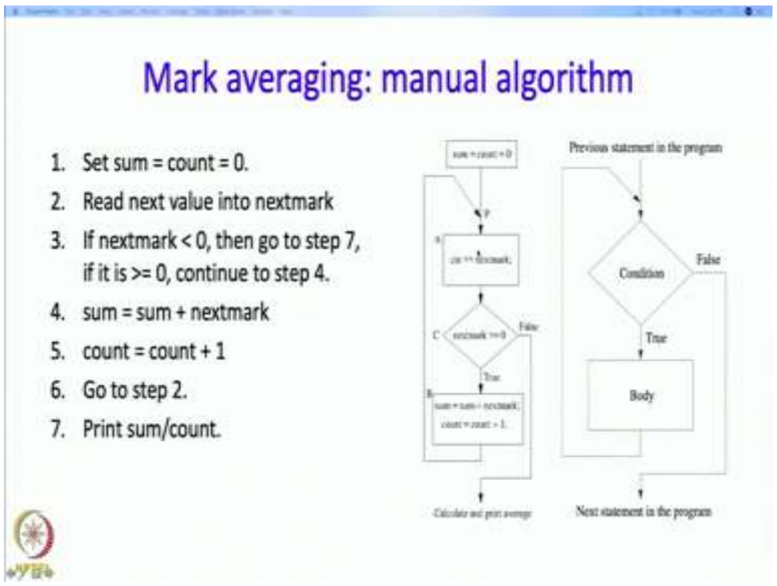
Mark averaging: manual algorithm

1. Set sum = count = 0.
2. Read next value into nextmark
3. If nextmark < 0, then go to step 7,
   if it is >= 0, continue to step 4.
4. sum = sum + nextmark
5. count = count + 1
6. Go to step 2.
7. Print sum/count.

Hello, in the previous segment we discussed the while statement and compared it to the repeat statement. Now, we are going to look at how to do the mark averaging problem. So, let us begin with the manual algorithm for mark averaging. So here it is, so we are going to keep two variables, so one is the variable sum into which we are going to get the sum of all the marks and then there is going to be a variable called count and in this variable we are going to keep track of how many marks we have read, so we are going to start by making these variables 0.

Then we are going to read the next value and the next value is going to be read and let us say we read it into variable called nextmark. Well in a manual algorithm we will not really have variables, of course, but since you now know variables, we might as well bring him in just to make our discussion clearer. Okay, then we are going to check is the value, the mark value that we just read which is in nextmark, is it less than 0, if it is less than 0, we then we know that we need to terminate, so we need to at that point, print the average and stop. So therefore, we are going to go to step 7, and step 7 indeed prints the sum divided by count, which is going to be the average, well, if nextmark is greater than or equal to 0, then what do you do? Well then which is continue, so we go to next step which is step 4 and now whatever mark we just read, which is a valid mark now we know and we added to sum, but we also read one more mark and therefore, we increment count by 1 as well and then we have to repeat the process, so we can go back to step 2, which is the starting point of the loop.

(Refer Slide Time: 2:20)



Mark averaging: manual algorithm

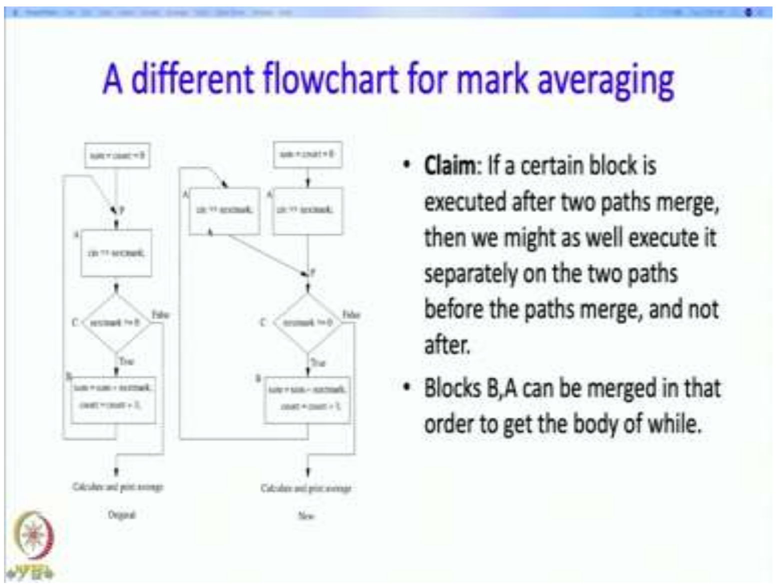
1. Set sum = count = 0.
2. Read next value into nextmark
3. If nextmark < 0, then go to step 7, if it is >= 0, continue to step 4.
4. sum = sum + nextmark
5. count = count + 1
6. Go to step 2.
7. Print sum/count.

Now, let us first draw a flowchart of this. Just to make sure that we understand exactly the control flow, that is what happens after what? So here is the flowchart, so we are going to start off by setting sum and count to 0, then we are going to read in nextmark, then we are going to check whether nextmark is greater than 0. Okay, if it is false or it is less than 0, then we are going to print, calculate and print the average, otherwise we are going to add nextmark to sum, we are going to add 1 to count, and then we are going to repeat again from this statement. Now, our goal is to implement this using the while loop and therefore, it might be instructive to compare the structure or the connections in this flowchart with the connections in a while loop. So here also there is this condition, here also there are some statements which are being executed. Okay, but there is an important difference between these two pictures, what is the difference?

After the body is executed the control goes back to the condition checking, over here if this is the body then after that body is executed the control does not go back to the condition checking, but it goes back to an additional statement before the condition checking happens.

So because of this difference, it is not clear that how we can use the while statement, because if this is the condition and this is the body, what is this? Okay, that is not very clear. Okay, we cannot directly use this. Okay, because the structures don't match. Okay, so we are going to than do something to the flowchart.

(Refer Slide Time: 4:26)



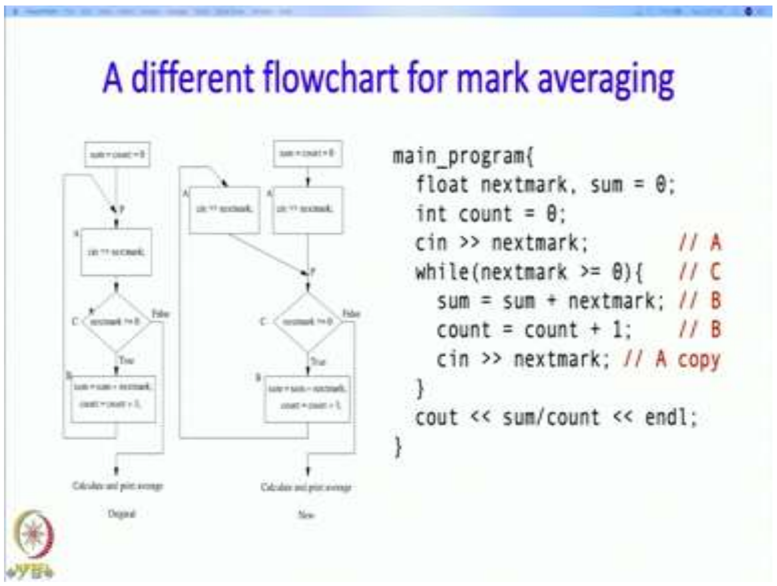
A different flowchart for mark averaging

- **Claim:** If a certain block is executed after two paths merge, then we might as well execute it separately on the two paths before the paths merge, and not after.
- Blocks B,A can be merged in that order to get the body of while.

So we are going to have a different flowchart for mark averaging, in this slide we have given two flowcharts for mark averaging program and I am going to observe that they are really doing the same thing. So what is it that we wanted? We wanted that this path should join not at this point, but it should join directly at the condition check. But we cannot just take this path and join it over here, if you do that, then along this path we would be doing something different. So, if we are going to make any changes, we should make sure that along every path the same operations happen, maybe they happen before the merge in one case, or in the case they happen after the merge. So if we do not like that these block is present between the point of joining and this condition check, what we could do is, we could push this block above on this path, but we cannot just push it on one side, on one of the paths because then if it come down the straight path and if the block is not present, then we would be doing something different around the straight path. So what we can do is that we can take this condition, this statement and we push it back on both paths okay. So what happens now is, that if we come down for the first time, then the operations that we perform are still going to be the same, so we are going to execute this statement, then the statement, then the condition check. Similarly over here, we are going to execute this statement, this statement and the condition check. On a subsequent iteration as well after the condition check we are going to execute this statement and we are going to execute this statement and the condition check again, even here after the condition check we are going to execute this statement exactly the same as before, then we are going to execute this statement exactly the same as before, and then we are going to execute the condition check exactly the same as before. So, the general claim is that if a certain block is executed after two paths merge, say as over here this block is executed, then we might as well execute it separately on the two paths, before they merge and then we do not have to execute it after they merge because we just executed them before. So essentially, what happens over here is exactly the same thing that happens over here.

Now, why is this helpful for us? Well, now we can make this be the condition and we will make this followed by this as the body of the loop, so I can sort of bring this back and just bring it back and sort of blend it into B, so then this will be my condition check and this followed by this will be my body.

(Refer Slide Time: 7:32)



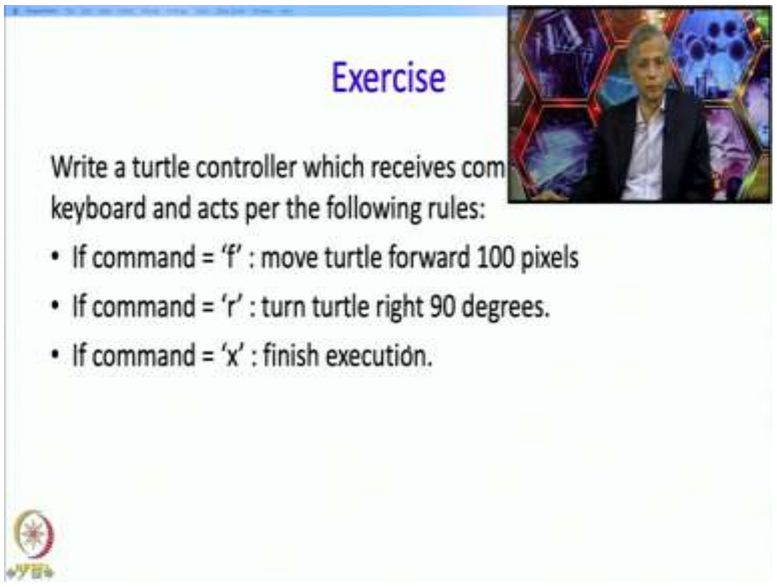
A different flowchart for mark averaging

```
main_program{
  float nextmark, sum = 0;
  int count = 0;
  cin >> nextmark;          // A
  while(nextmark >= 0){     // C
    sum = sum + nextmark;   // B
    count = count + 1;      // B
    cin >> nextmark; // A copy
  }
  cout << sum/count << endl;
}
```

So here is what the program is going to look like, so I am going to declare the variables nextmark and sum and sum is going to be 0, I am also going to declare count and count also going to be 0. Okay, so I am at this point I have executed this statement, and I have executed this statement. Now, I need to execute this statement and this statement is not within any loop or anything, so I have that, so this is this block A which I have marked here, just to indicate the correspondence. Now after this, I come to the condition check of the while, so I am going to have a while and that is going to be my condition check, so this C block over here is going to be this statement. After that, I am going to have the body, so the body first has this statement from B, it then has the second statement from B, but as it executes it has to execute this statement from A again. So that is also going to be a part of the body and this is a copy of

A, so that is what I have indicated over here in the comment. So at this point the body execution ends, so we have this and then again the control will go back and start from this statement which is exactly what happens in the flowchart over here. So if the condition is false, then I want the average to be calculated and printed. Similarly over here, if the condition is false, we will come and execute the statement over here and so we will put the average calculation and printing statement over here and that is what our program is going to be, so this program now uses the while statement and it directly implements this flowchart, but we know that this flowchart is really the same as this flowchart.

(Refer Slide Time: 9:32)



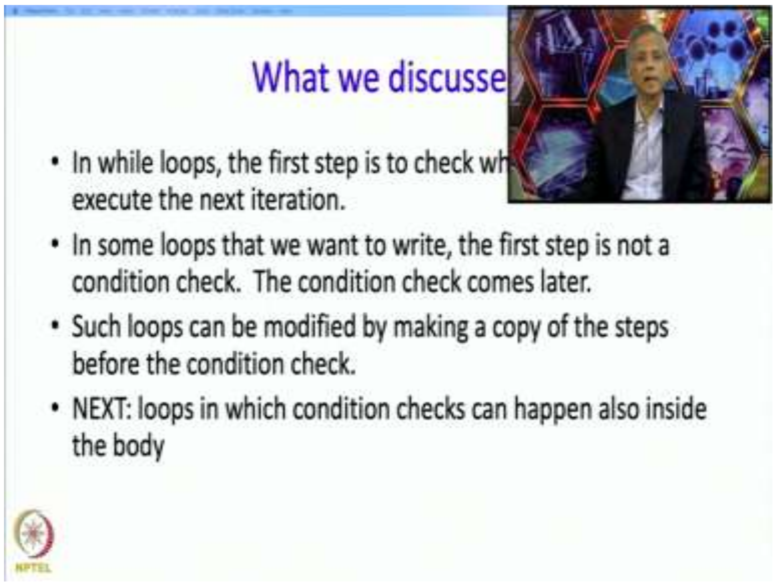
Exercise

Write a turtle controller which receives com
keyboard and acts per the following rules:

- If command = 'f' : move turtle forward 100 pixels
- If command = 'r' : turn turtle right 90 degrees.
- If command = 'x' : finish execution.

So here is an exercise for you, write a turtle controller which we saw last time but which this time has an extra command, so if from keyboard you type 'f', then the turtle should move forward 100 pixels as before, if you type 'r', then the turtle should right 90 degrees. But now if you type 'x' the turtle should finish execution, you will see that you will have to pretty much have the same structure as before, so maybe repeat some part of the code before entering the loop and so on.

(Refer Slide Time: 10:08)



What we discusse

- In while loops, the first step is to check wh
  execute the next iteration.
- In some loops that we want to write, the first step is not a condition check. The condition check comes later.
- Such loops can be modified by making a copy of the steps before the condition check.
- NEXT: loops in which condition checks can happen also inside the body

Alright, so what have we discussed in this segment? So we said that in while loops, the first step is check whether we need to execute the next iteration. In some loops that we want to write, the first step may not be the condition check, we might have have to do something before we check the condition. In this case we can modify the loops or we can modify the flowchart associated with the loops by making a copy of the steps that we need to do before the condition check and they have to be performed before the entry into the loop as well as at the end of the body of the loop. So this is one way of dealing with such slightly tricky loops, but there is another way in which we will see in the next segment, so will take a break here.