



IIT Madras

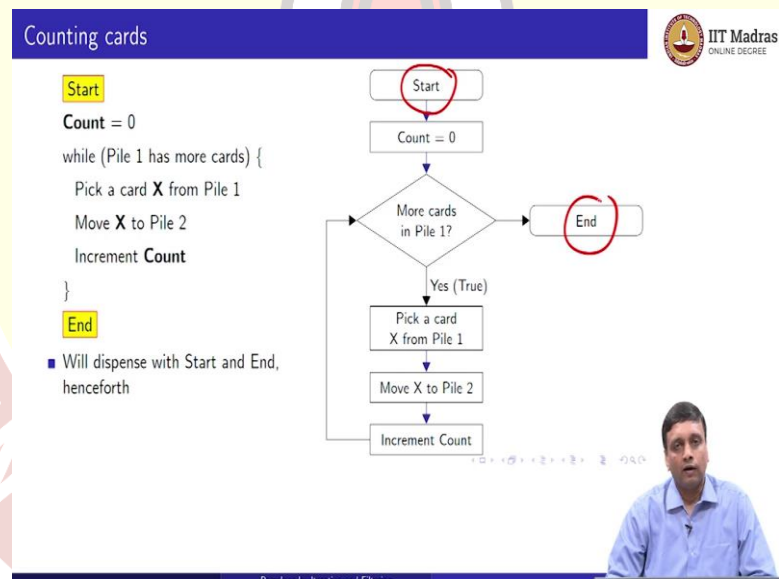
ONLINE DEGREE

Computational Thinking
Prof. Madhavan Mukund
Prof. G. Venkatesh
Department of Computer Science
Chennai Mathematical Institute
Indian Institute of Technology, Madras

Lecture – 2.10
Pseudocode for iteration with filtering

In the previous lecture, we looked at pseudocode as a textual representation of computational algorithms. So, now, let us try to work through some of the examples we did with iteration and filtering, and see how the flow charts that we drew can be translated into pseudocode. And in the process, we will look at some more notations that we will use in pseudocode in order to make things more simplified to read.

(Refer Slide Time: 00:36)



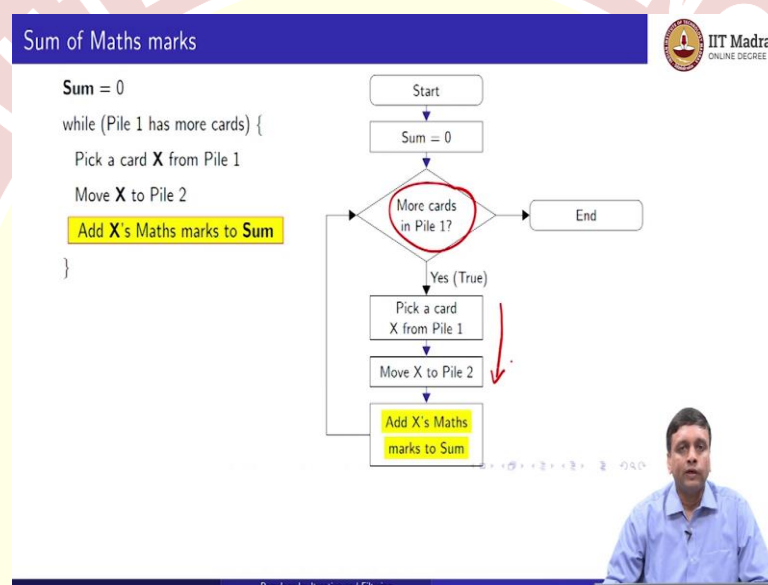
So, this was the first algorithm that we saw in pseudocode in the last lecture. So, this was the one for counting cards. So, on the right, we have the flowchart that we had drawn right at the beginning.

And on the left is the pseudocode that we described in our first lecture to indicate the features such as for instance we have this assignment statement, and then we have this while which is described as a block of things which are to be repeated while a condition holds.

So, one of the things we will do is to dispense with this start and end, because in a piece of pseudocode it is usually clear where you start at the beginning of the text, and you also usually end at the end of the text.

So, for the moment, just to simplify the pseudocode we will drop this what correspond to these terminal node. So, we have these terminal nodes in our flowchart with these rounded boxes. So, we will drop these things from our pseudocode just because there is no harm in doing that without confusing anyone.

(Refer Slide Time: 01:33)



So, let us take this count algorithm and modify it to calculate instead the sum of the maths marks. So, we had done this in the flowchart. So, this was our first modification. So, we take this iteration and we modify the iteration to do something different not just count the cards, but add up the marks of all the students in maths.

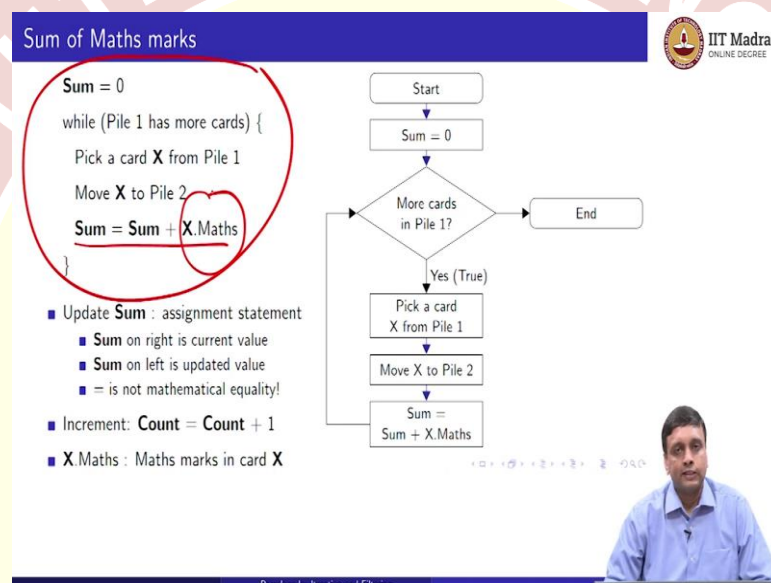
So, the first thing is that count is not an appropriate name for our value that we are keeping track of. So, let us call it Sum. So, we are adding up all the maths marks. So, we start by saying that the total number of maths marks across the class is 0. So, we replace the name count by sum and say sum equal to 0. So, so at this moment the only thing we have done in the flowchart is to change the name of the variable we are keeping track of.

Now, the rest of it is as is similar; so you will again check whether there are more cards pick up a card, move it. And now when we pick up a card instead of just incrementing

the count, what we need to do is increment the sum and not quite increment it, but update it. So, we want to update it by adding the maths marks of the current card to the sum.

So, now, we have made two changes, but otherwise the structure of the flowchart and the structure of the pseudocode is very similar; instead of count we have sum both were initialized to 0. And at the bottom of the loop, at the last statement in our while, instead of incrementing the count what we are doing is we are taking the maths marks of the card we have just seen and adding it to the sum.

(Refer Slide Time: 03:04)



Now, we can write this update to sum just like we did the initialization. So, there is no real difference between initializing a value to a variable that is assigning it a constant value like 0, or 5, or minus 1, and updating it by an expression like 4 plus 2, or 7 plus 8. Now, here we have written something which might look a little strange to some of you. We have said that sum is equal to sum plus X dot Maths. So, this looks like we are saying that a is equal to a plus b, but this is not an equality.

So, what sum stands for is the value that it holds. And in this assignment statement, what we are saying is that take the current value of sum which is on the right hand side, add to it the current value of the maths marks of the card X that we have read and once we have added it replace it into sum.

So, the sum on the right is the current value, the sum on the left is the updated value. So, equality is assignment, equality is not the mathematical equality we will come to that in a later algorithm.

Now, in this notation what we had written earlier as increment count is also an assignment. We say take the current value of count add 1 to it, so Count plus 1, take this expression and update Count with this value. So, Count equal to Count plus 1. This is not saying that a number is equal to 1 plus itself, but saying replace the value of Count by the current value of Count plus 1.

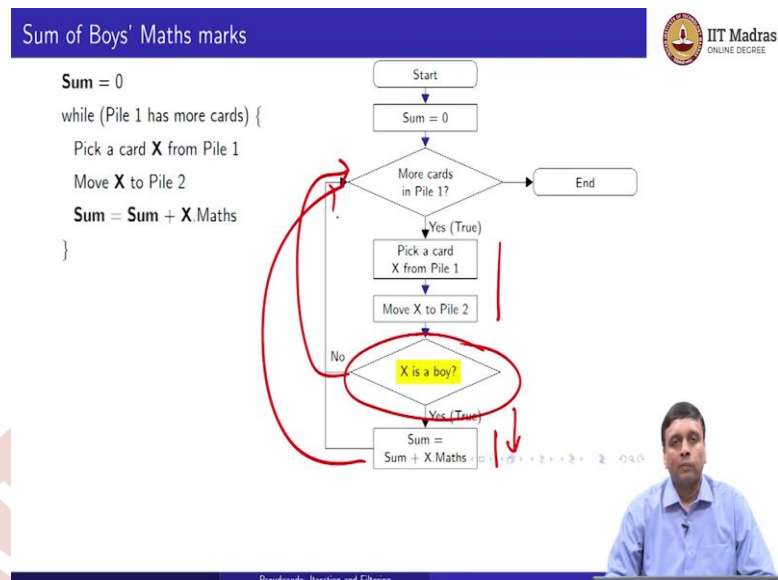
So, this assignment statement is a flexible way both to assign a value initially or a constant value at any time. But also to update a value, and the update could involve the current value of the, of the variable itself. And finally, we have used this kind of notation to indicate a part of a structured data type. So, we have a, we have this card which has many values on it. So, a card itself is called X, it has maths marks, physics marks, chemistry marks, it has a date of birth, it has a gender, and so on.

So, if we want to refer to the specific value of one of these fields inside the card, then we are using this notation X dot Maths. So, X dot Maths is a short form for the maths marks on the current card X. So, this gives us a degree of precision and conciseness we do not have to keep writing this long English sentences. So, now, we have this procedure here which is now in pseudocode right, the variation of our counting cards in order to add up the maths marks.

So, here just to remind you, we have done a couple of things. One is that we have introduced this more general assignment statement which allows us to take a value and update it including the possibility that we are using the old value of the same variable as part of the assignment.

And the second thing is that we have used this notation which tells us that we are looking at the maths field within a card X. So, when we have a complex piece of data with lots of sub parts, we can identify a sub part by using this dot notation.

(Refer Slide Time: 05:57)



So, the next thing that we did when we did flowcharts was to filter this iteration. So, far we have just been iterating, we have gone going through the entire stack of cards and accumulating some value, either the count of the cards or the sum of the boy of the maths marks.

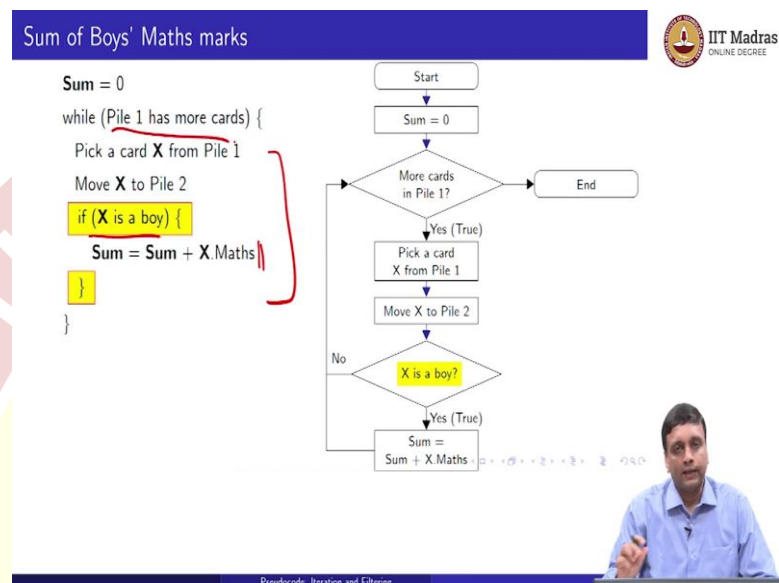
But now supposing we do not want to accumulate all the values, but only some of the values. So, this is what we called filtering. We want to go through the cards and depending on whether a condition holds or not in the card, we want to make an update.

So, here the case is that we want to update only the values of the boys math marks. So, here is our original algorithm which adds up all the marks. So, clearly before we make an update, we need to decide whether this card belongs to a boy or not. So, we have to add an extra condition. So, between picking up the card and adding the marks, we check whether this particular card corresponds to a boy; if it is not a boy, then we directly go back to the next card.

If it is a boy then we go down, add the mark, and then come up right. So, this is how we modify our flow chart by adding one extra decision box between the point where we pick up the card and move it, and then the point where we directly update the sum, because we do not want to update the sum unless it is a boy.

So, we need to transfer this notation to our pseudocode. So, earlier we have seen that while represents a condition which happens again and again until the condition fails to be true, but in this case for each card, we want to do this once. So, we use a different word again a word which is common from English called if.

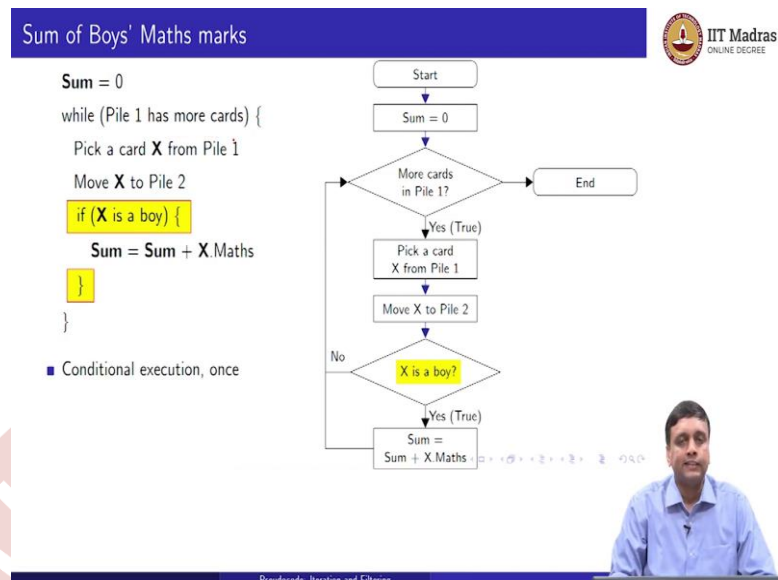
(Refer Slide Time: 07:30)



So, we say that if X is a boy, then we add Sum plus X dot Maths to the value of sum. So, if X is not a boy, then we will skip this. So, this part is done only if the condition is true right.

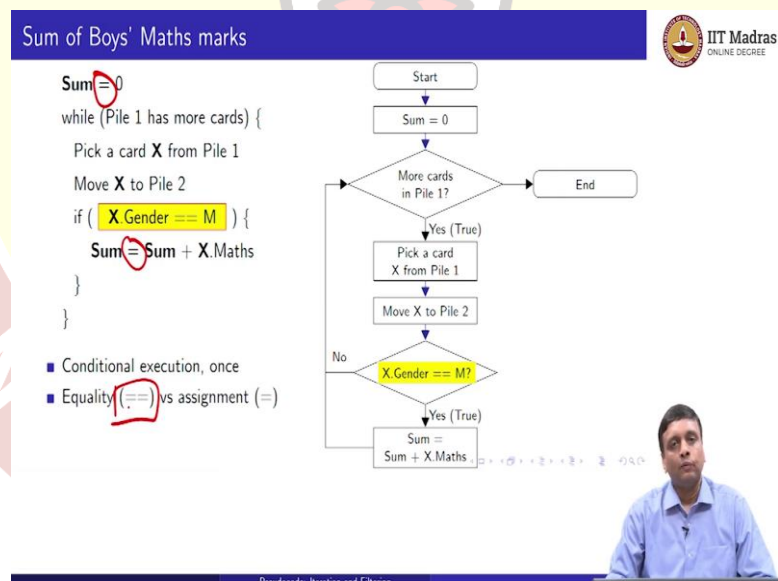
So, just like these steps are done, this entire step is done only if the pile has more cards right. So, while says do this entire block once until you come back and test the condition again; if says do the block once, but do not come back. So, if, it is a one stop; so it is a conditional execution which is done once right.

(Refer Slide Time: 07:59)



So, here is the other thing that we can do.

(Refer Slide Time: 08:03)



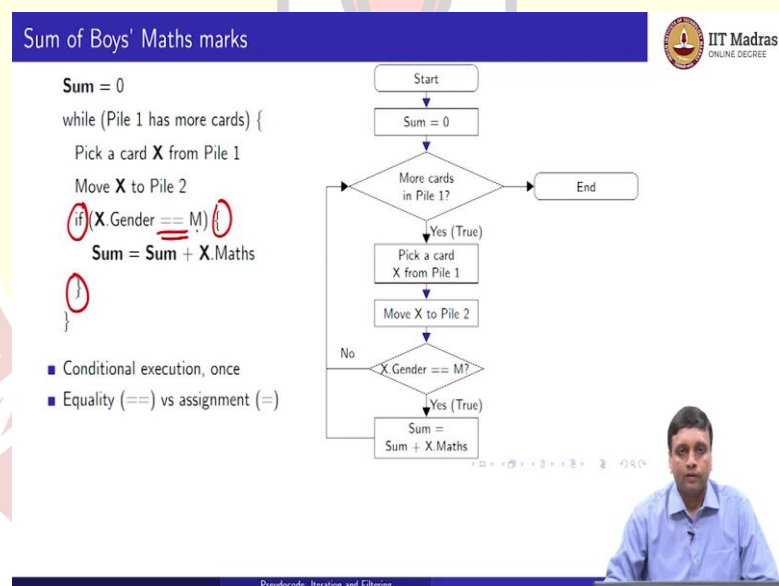
Remember that we said that we would like to make things as precise as possible and not have to write English text. So, what does it mean for X to be a boy? Well, we know in this case that the fact whether X is a boy or not is recorded on the card, it is one of the fields just like the maths marks and the physics marks, we have a gender. So, remember that the gender field of card X, we will write as X dot gender. So, we want to check whether X dot Gender is M.

Now, we have a problem because we want to check equality. We want to check whether X dot Gender is equal to M. But we have already said that equality is been reserved for this assignment right. So, assignment is equality.

So, we have this equality. And if we have one more version of equality, it can be confusing. So, in many programming languages and in the pseudocode that we are informally developing as we go along, we are going to use a different symbol which is a double equal to.


So, a double equal to – equal to equal to – twice, indicates equality in the sense that we know it from mathematics two things are equal if they are the same quantity. So, here we are asking whether the gender field on card X is the same as the letter N. And if it is so, then we add it which is exactly the same as checking whether X is a boy or not. So, this now is our final code right.

(Refer Slide Time: 09:21)



So, we have now looked at two new things in this. One is that we have this if which is a replacement for the while right which looks at not a replacement, but an alternative to the while. So, if, if does a condition once; whereas, while checks a condition repeatedly. And the other thing we have seen is that we can use this double equality in order to check whether a variable equals a value.

(Refer Slide Time: 09:43)

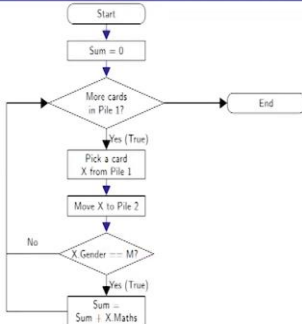


IIT Madras
ONLINE DEGREE

Sum of Boys' and Girls' Maths marks


```


Sum = 0
while (Pile 1 has more cards) {
    Pick a card X from Pile 1
    Move X to Pile 2
    if (X.Gender == M) {
        Sum = Sum + X.Maths
    }
}
    
```



```

graph TD
    Start([Start]) --> Sum0[Sum = 0]
    Sum0 --> MoreCards{More cards in Pile 1?}
    MoreCards -- Yes (True) --> PickCard[Pick a card X from Pile 1]
    PickCard --> MoveX[Move X to Pile 2]
    MoveX --> GenderCheck{X.Gender == M?}
    GenderCheck -- Yes (True) --> SumAdd[Sum = Sum + X.Maths]
    SumAdd --> MoreCards
    GenderCheck -- No --> MoreCards
    MoreCards --> End([End])
    
```





Fundamentals, Reasoning and Algorithms

So, moving along, we saw how to total the boys marks by filtering on the gender of the card, but what if we also want to simultaneously keep track of the girls marks. So, the first thing is that now we have to keep track of two quantities.

It is no longer just the sum of the marks; we have two different sums one corresponding to boys, and one corresponding to girls. So, we will need two different values. So, to begin with let us keep track of the boys sum using a better name.

(Refer Slide Time: 10:13)

IIT Madras
ONLINE DEGREE

Sum of Boys' and Girls' Maths marks

```

BoySum = 0
GirlSum = 0
while (Pile 1 has more cards) {
    Pick a card X from Pile 1
    Move X to Pile 2
    if (X.Gender == M) {
        BoySum = BoySum + X.Maths
    }
}
        
```

```

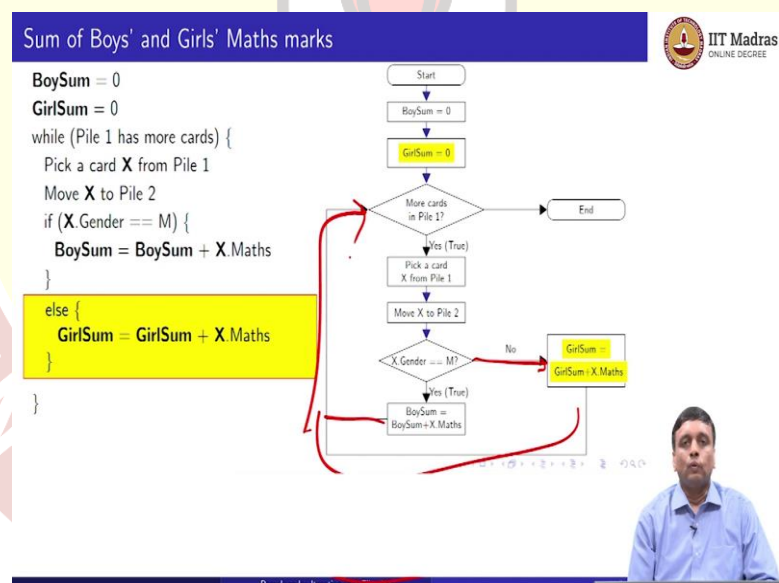
graph TD
    Start([Start]) --> Init1[BoySum = 0]
    Init1 --> Init2[GirlSum = 0]
    Init2 --> Cond1{More cards in Pile 1?}
    Cond1 -- No --> End([End])
    Cond1 -- Yes True --> Pick1[Pick a card X from Pile 1]
    Pick1 --> Move1[Move X to Pile 2]
    Move1 --> Cond2{X.Gender == M?}
    Cond2 -- Yes True --> Calc1[BoySum = BoySum + X.Maths]
    Calc1 --> Cond1
    Cond2 -- No --> Cond1
        
```

So, instead of just Sum, we will call it BoySum. So, BoySum is now the same as the previous algorithm except that we have replaced the word Sum by BoySum. So, everywhere where we use Sum before we are using BoySum. So, we are keeping track of the sum of the boys marks in exactly the same way.

Now, as we said we will need a GirlSum. So, how do we start? Well right at the top where we initialize the BoySum to 0; we will also initialize the GirlSum to 0. So, we add another variable there, and we add another initialization saying GirlSum is 0 at the beginning of the algorithm.

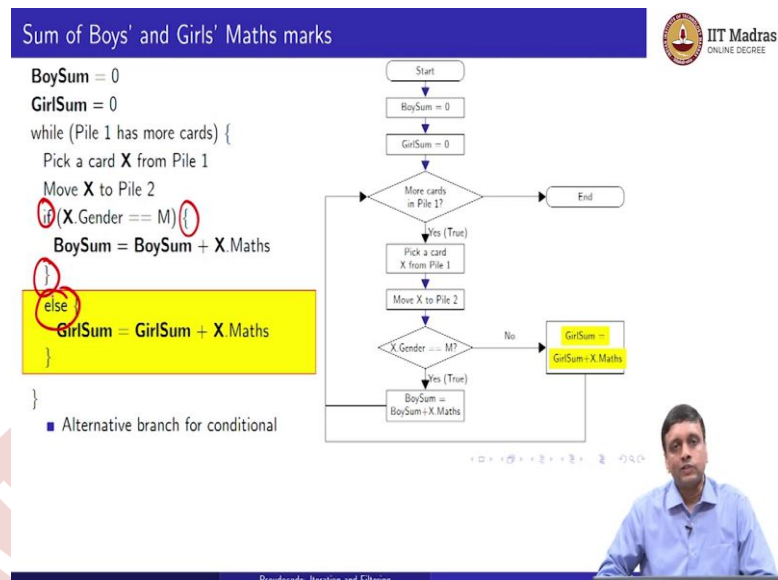
Now, clearly GirlSum has to be updated. So, when do we update GirlSum, well when the card that we pick up corresponds to a girl which means it is not a boy, so that means, we have to take this branch which says no and does nothing currently and instead make it do something. So, when it says no, we need to increment the GirlSum.

(Refer Slide Time: 11:03)



So, we get a new flowchart in which the no branch now takes us to a new process box which says GirlSum is GirlSum plus X dot Maths. And now whether we update the BoySum or the GirlSum both updates take us back to the next card.

(Refer Slide Time: 11:22)

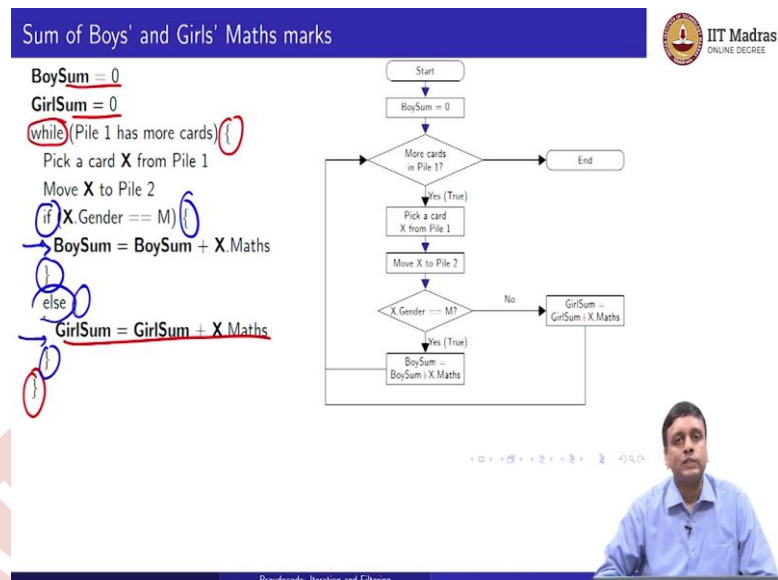


So, this is again a new feature that we have described because we had one if, so earlier we had an if, and what to do, if the if is true. Now, what we say is that if this is not true, we also want to do something. So, it is really a choice between doing two different things. The previous version of if just said do something; otherwise skip it.

Now, we are saying not do something otherwise keep it. We are saying if X is if the X gender is M do something that is update the BoySum; otherwise do something else which is update the GirlSum right.

So, again like in the other words that we have seen while and if, we have a natural English word called else. So, if something holds – do something; else do something else. So, with this we have more or less covered all the things that we need to know about our pseudocode right.

(Refer Slide Time: 12:09)

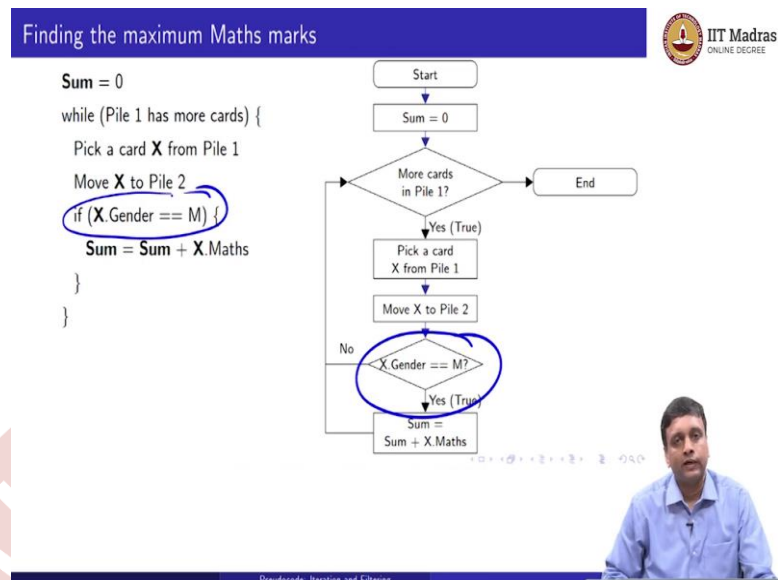


So, we have come up. So, first we have these assignment statements right which could be like that or it could be a complex update, then we have these special conditional things. So, we have this repeated condition while, each one comes with a pair of braces. So, these are the wild braces.

Then this is an if, and these are the if braces and with the if, we have an else and these are the else braces. So, we have to keep track of these braces and make sure that these braces are properly nested inside each other and all that, but otherwise we have used some other notation which is not strictly required for instance we have indented these things right.

So, whenever we have something inside braces, we have pushed it in to indicate that it belongs to the brace. So, this is some kind of formatting which makes the text easier to read, but it is not essential in order to understand it. The pseudocode would be the same whether we indented it or not, but indented pseudocode is better for us to read.

(Refer Slide Time: 13:07)

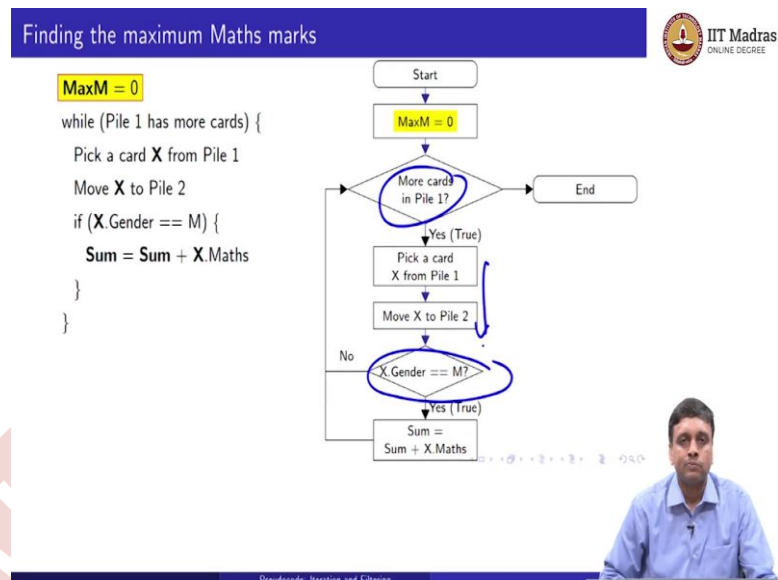


So, let us look at two more examples that we did in the flowchart world. So, one of them was to find not the sum of the marks, but the maximum maths marks. So, this is the flowchart for the sum of the marks; the one that we saw a little while back not the BoySum. So, there is no filtering in this at the moment.

So, there should not be a filtering. So, this is actually. So, for our final example, we will move from taking the sum to finding the maximum. So, now, instead of adding up the maths marks across the cards, what we want to do is find the maximum maths marks.

So, here for instance we have the code that we wrote to find the sum of the boys maths mark. So, this was a filtered iteration. If you remember we went through all the cards and then we had this filtering condition here, we checked whether the gender was M or not which is this if condition here. So, instead of the sum we want to keep of the max maths maximum maths mark.

(Refer Slide Time: 14:03)

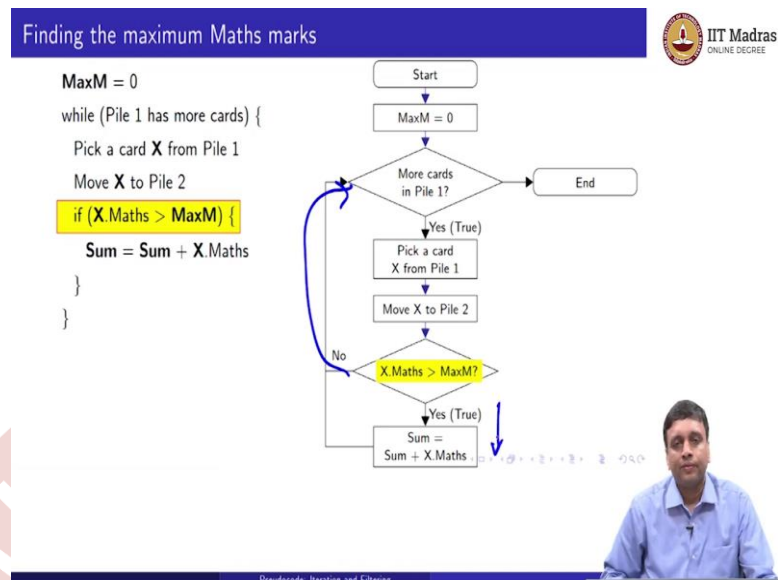


So, first of all let us use a different name for the variable. So, we take instead of Sum the word MaxM. Now, when we are finding the maximum or the minimum, we need to be careful about the initialization, because we need to find a value such that we are guaranteed supposing, we take MaxM to be 100 for example right to initially, then we may not find anybody in this class who has 100 marks in maths, and then the end it will spuriously tell us that we found the maximum maths to be 100. So, in this case we know that the max marks go between 0 and 100.

So, it is safe to set the maximum to be the smallest value that you expect to see in this case 0. So, that whenever you find a real value it is going to be at least as big as this, and it will update the maximum. So, the first step is to initialize this MaxM to be 0. So, MaxM is going to represent the maximum maths marks in the entire stack. Now, we come about the thing.

So, we check if there are more cards we pick up the card and so on, and now of course, we do not want to check the gender rather we want to check whether the marks that we are seeing on the current card exceeds the maximum marks that we have seen so far or not. So, at any point in this iteration MaxM is the maximum maths marks we have seen so far.

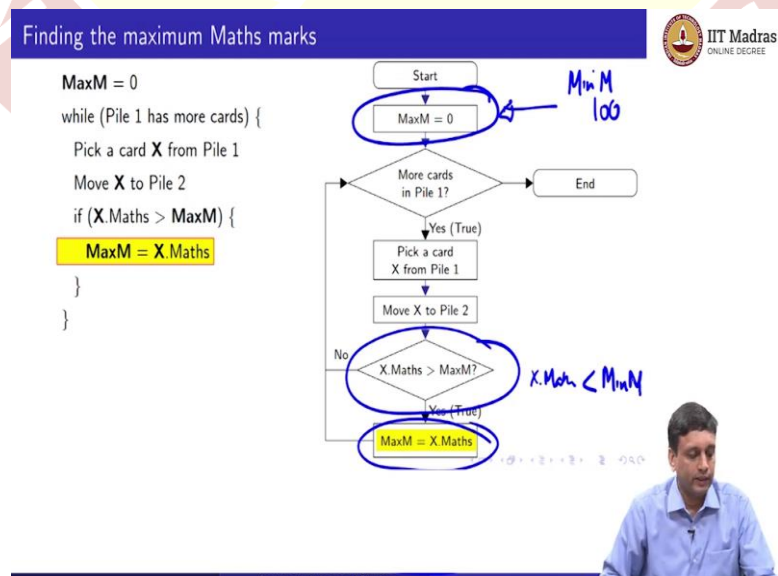
(Refer Slide Time: 15:15)



So, the next step is to change that condition and check whether the maths marks of X, X dot Maths is bigger than MaxM the current maximum. Now, if it is bigger than the current maximum, then we have to do something; if it is not, we just go back right.

If we have seen a card supposing our current maximum is 82, and the current card says that the person has got 76, then there is nothing to do because 76 is smaller. So, we don't need to update the maximum we have seen so far. However, if it is bigger, then we need to do something.

(Refer Slide Time: 15:48)



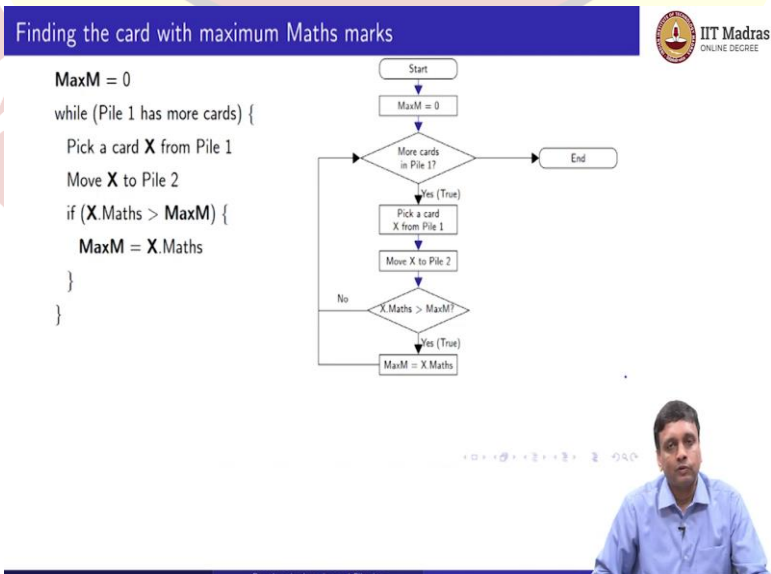
So, now we replace this update right instead of updating the sum, what we do is we reset the maximum to be the current card. So, supposing we were at 82, and now we see a card with 87, then we will replace 82 by 87 in the value MaxM right. So, this is our new filtered iteration. So, it is very similar to the sum of the boys marks. So, the structure is the same. We have an initialization right.

We have a condition that we check and we have an update, except that the condition and the update depend on what we are looking for what we are keeping track of instead of accumulating the sum we are looking for the maximum. And also the initialization also depends on what we are doing. So, here the initialization is to 0, because we know that the range is 0 to 100.

Now, if we were doing minimum for example, then what would we do, we would set this to 100. And then we would check X dot maths is less than of course we would also call it MinM probably.

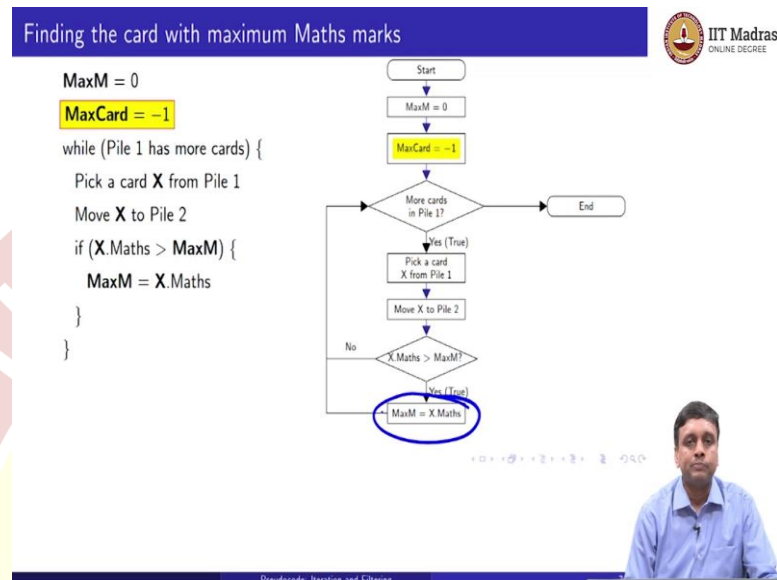
And we would check that the current maths marks is less than the minimum; and if so, we would update it. So, remember that depending on what you are looking for the initialization will change. Finally, what we wanted to do was to take this procedure to find the maximum maths marks, and also find out which student had this.

(Refer Slide Time: 17:04)



So, which card in our stack had this maximum maths marks. So, in order to do this, now we again we need to keep track of two quantities - the marks and the card with those marks. So, we will need something which will represent the card number.

(Refer Slide Time: 17:18)



So, let us call it MaxCard. Now, the MaxCards, we do not really have a clear idea of how big the class is or what these card numbers are, but we do know that they are positive numbers. So, here our safe initialization is minus 1, because there may be a card number 0, we do not know. So, we do not want to spuriously claim that card number 0 is the maximum.

So, we initialize it to a fictitious value minus 1 which will surely be updated whenever we see a card where the maths marks is more than 0. Having initialize this, now we only have to do one more thing which is to update this whenever we find a new card. So, whenever we do this update where we reset the maximum marks, we should also reset the maximum card right.

(Refer Slide Time: 18:02)

Finding the card with maximum Maths marks

```
MaxM = 0
MaxCard = -1
while (Pile 1 has more cards) {
    Pick a card X from Pile 1
    Move X to Pile 2
    if (X.Maths > MaxM) {
        MaxM = X.Maths
        MaxCard = X.Id
    }
}
```

```
graph TD
    Start([Start]) --> Init1[MaxM = 0]
    Init1 --> Init2[MaxCard = -1]
    Init2 --> Cond1{More cards in Pile 1?}
    Cond1 -- Yes (True) --> Pick[Pick a card X from Pile 1]
    Pick --> Move[Move X to Pile 2]
    Move --> Cond2{X.Maths > MaxM?}
    Cond2 -- Yes (True) --> Update1[MaxM = X.Maths]
    Update1 --> Update2[MaxCard = X.Id]
    Update2 --> Cond1
    Cond2 -- No --> Cond1
    Cond1 -- No --> End([End])
```

Pseudocode: Iteration and Filtering

So, we add one more step inside that section which says not only update MaxM to X dot card X dot Maths, but also replace MaxCard by X dot id that is the id of the current card. So, by adding this one extra variable initialized to minus 1 and updated every time we replace the max maximum marks by the current card marks, we can keep track of not just the maximum marks, but also which card corresponds to the maximum marks.

(Refer Slide Time: 18:30)

Summary

- Assignment statement
 - Count = 0
 - Sum = Sum + X.Maths
- Conditional execution
 - Once
 - if (condition) { ... }
 - if (condition) { ... } else { ... }
 - Repeatedly
 - while (condition) { ... }
- Equality (==) vs assignment (=)

Pseudocode: Iteration and Filtering

So, to summarize, let us see what all we have in our pseudocode now. We have this assignment statement. So, the basic form of the assignment statement takes a variable

and assigns it a fixed value. This is typically used for initialization like setting count equal to 0. But the more complex form takes a variable and assigns it to an expression, and this expression could use the same variable that is being assigned.

In this case, we are saying Sum is equal to Sum plus X dot Maths. What this means is take the old value of sum add the current value of X dot Maths from the new card, and replace it back therefore, updating sum. So, this is not an equality, but an assignment.

We also saw how to describe conditional execution. So, if we want to do one conditional step or block of steps, we use an if statement. So, we write if, put a conditional statement, and if this condition evaluates to true then the steps which are between the braces will be executed. We can also give an alternative branch.

So, if we say if the condition holds – do something; else do something else and if we want to repeat this till the condition becomes false, then instead of if we use a while. So, we say while a condition is true, do a sequence of steps, come back, test the condition again; if it is true, do the whole block again and so on until the condition becomes false.

And finally, because we are using this equality symbol for assignment to assign a value to a variable, it is useful to have a different symbol when we actually talk about equality for comparison when we check whether a variable is equal to a value. So, we have decided to use double equal to which is quite common across many programming languages.

So, with this notation now we are well set to describe all the procedures that we will see in pseudocode. If we come across new features which are difficult to capture in this, we will add to our language of pseudocode as we go along.