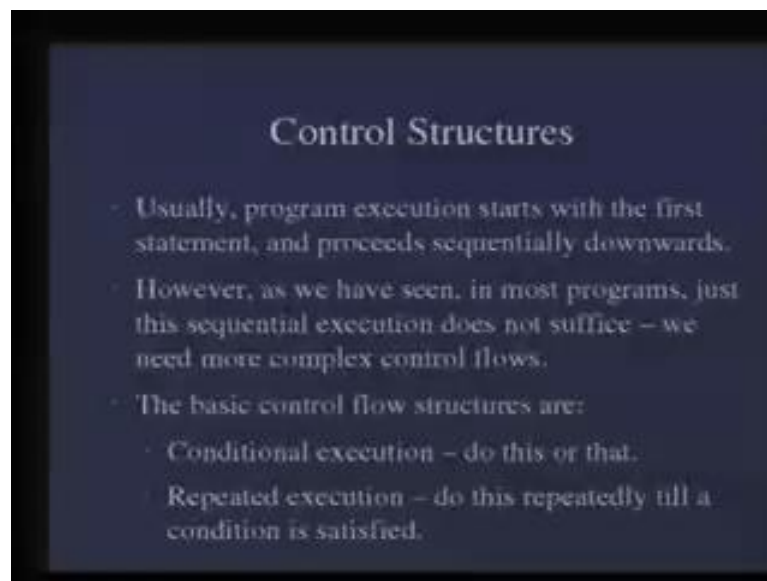


Introduction to Problem Solving and Programming
Prof. Deepak Gupta
Department of Computer Science Engineering
Indian Institute of Technology, Kanpur

Lecture – 10

In the last lecture, we have talked about input and output facility in C and we had looked at the printf, scanf, getchar, and putchar functions. Today we will start looking at another very important topics that is control statements; we already familiar with control statements and we are being using them in most of the programs that we have written, statements like if and the if else and the while statements are what known as the control statements.

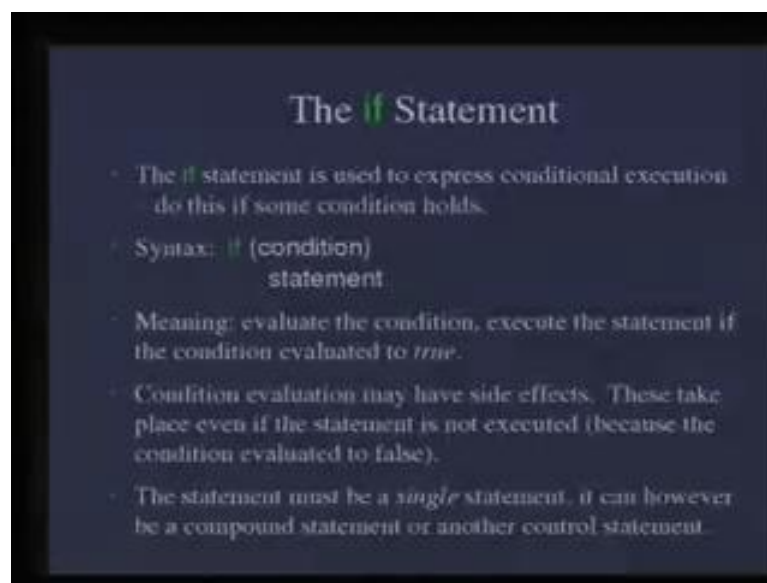
(Refer Slide Time: 00:43)



Usually program execution starts with the first statement and proceeds sequentially downwards; but in any, but most simple program, we this is not sufficient we want be control flow to take more complicated form. Usually for example, we might want to do one thing or the other depending on some condition or you might want to repeatedly execute some block of code for that we need what are known as control statements. We have as I said already seeing this control statements in action and use them in all the programs we have written so far, but we will now in this lecture and the next lecture look at some of these statements in more detail.

There are two basic control flow statement or the control flow structures, which has the first one is for the conditional execution which is either do this or do that. This is implemented using the if and the if else conditional statement that we have already seen. And the other useful kind of control structure is one of the repeated execution where we want to keep doing certain action repeatedly till some condition satisfied. So, this lecture will look at the conditional execution structures in more detail; in particular, we look at the if, if else statements and the switch statements which we have not use in program so far.

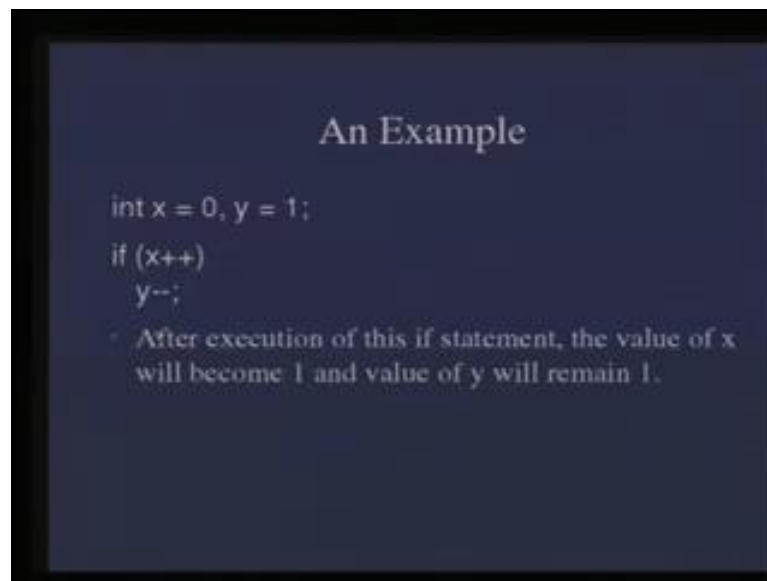
(Refer Slide Time: 02:09)



So, let us look at if statement first. The simplest form are the if statement has the syntax shown here. There is a condition and there is a statement following it, and the meaning of this kind of statement is that if the condition is true then the statement is executed. If the condition evaluates to the false then the statement is not executed. There are two important points about this particular concept; first is that the statement which is shown here must be a single C statement, which means that I cannot be a sequence of several assignment for example. Now of course, in practice we would like this block of codes to be executed when the condition is satisfied to consists of more than one statement, but that is very easy to us. Because as we have already noticed in our earlier lecture a set of statements if they are enclosed in braces then that becomes this single compound statement.

So, this statement has using the statement, but of course, this can be a compound statement meaning that if we put enough sequences of statements within braces that must single compound statement we can use it here. So, the meaning of the statement is first evaluate the condition; note that the condition may have side effect, which we have already talked about. So, those side effects will happen and finally, the result of the condition evaluate to true and remember that in c true is semi value which is non zero then the given statement is executed, otherwise it is not executed.

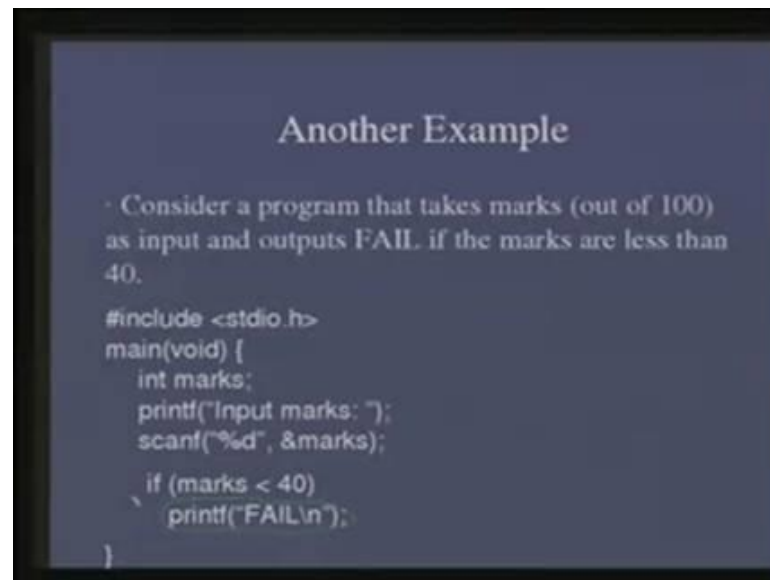
(Refer Slide Time: 04:05)



Here is the simple example to let us check the side effect may takes place during the condition evaluation. In this example, x is initially zero and y is initially one. And the if statement is, if x plus plus then y minus minus. Now plus plus remember the post increment operator and minus minus is the post decrement operator. So, when the condition which is in the case x plus plus is evaluated, the result of the expression is zero which is the old value of the x and the new value of the x becomes one.

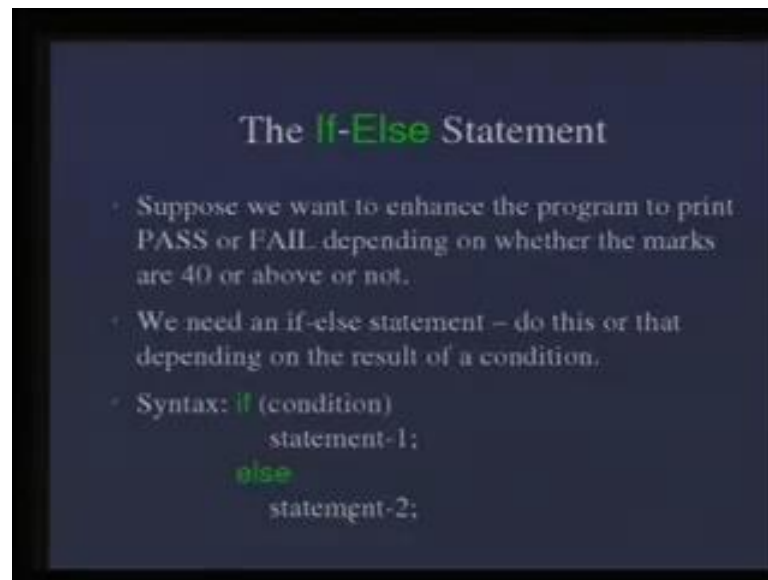
So, since the result of the expression evaluation is zero, which is false in C, therefore the statement y minus minus does not get executed, which means at the end of the execution of the statement the value of x become one, because the side effect in the evaluation of the condition x plus plus. And the value of y remains one because y minus minus does not get executed.

(Refer Slide Time: 04:58)



So, let us take another somewhat more realistic example. Suppose you want to write a simple program that take as input the marks obtained by some students in some exam out of hundred and output FAIL if the marks are less than forty, this is of course a very very simple program. So, the usual things we declare a variable mark and read the value of mark nothing strange here, and here is the main part of the program. This marks less than forty then printf FAIL. Note that printf FAIL is the singular statement and therefore, we do not really need to put braces to make it the body of the if statement. So, when this program executes, we read the value of mark if the value of marks are less than forty then this printf statement execute and FAIL is printed on the output, but if the marks are forty or more then nothing happens. No print statement is executed, therefore there is no output from the program in that case.

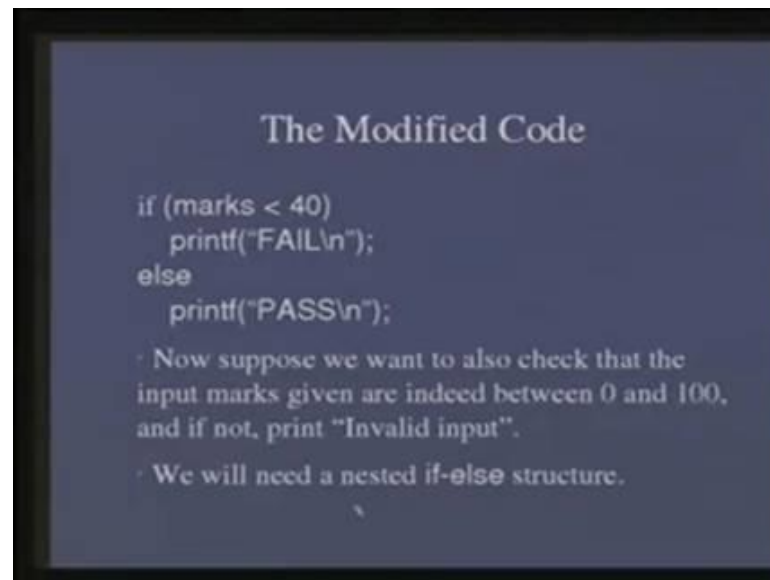
(Refer Slide Time: 06:05)



So, that probably not you wanted suppose we want to have a our program return such a session that it wants get print PASS or FAIL depending on whether the marks are forty or above or not. So, print pass, if marks are forty or more; or print fail, if marks are less than forty. In this case, what we need is an if else statement which essentially said do this or that depending on the result of the condition. So, I want to evaluate a condition in this case whether the marks are more than forty or more or not and if that is true then execute some statement otherwise execute some other statement. This is again the statement, we have already seen in practice in many programs we have written earlier.

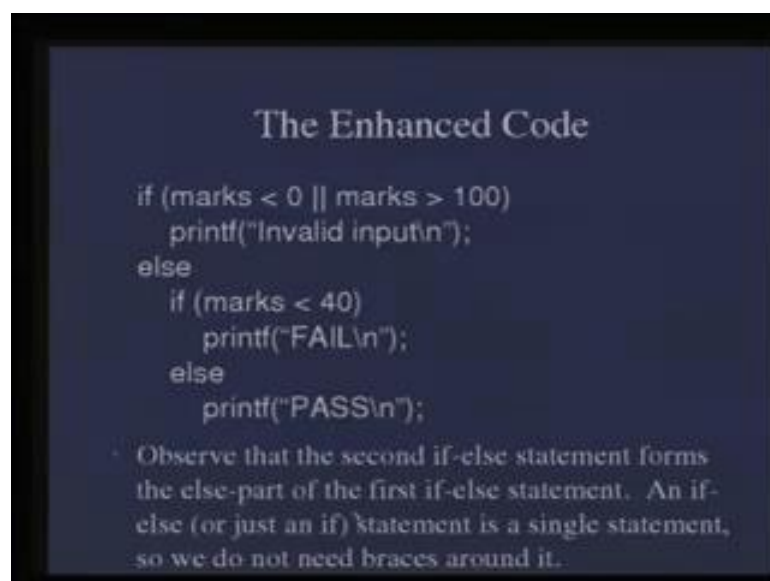
Here is the syntax. Here is again the condition if and else are reserved words of key words; if condition statement one else statement two. Statement one and statement two are again have to be single statement. But of course, they can be compound statement or indeed as we see they can themselves be control statements like the statement one, statement two itself could be another if statement. So, for this particular example, the code looks very simple, if marks less than forty printf FAIL, else printf PASS.

(Refer Slide Time: 07:15)



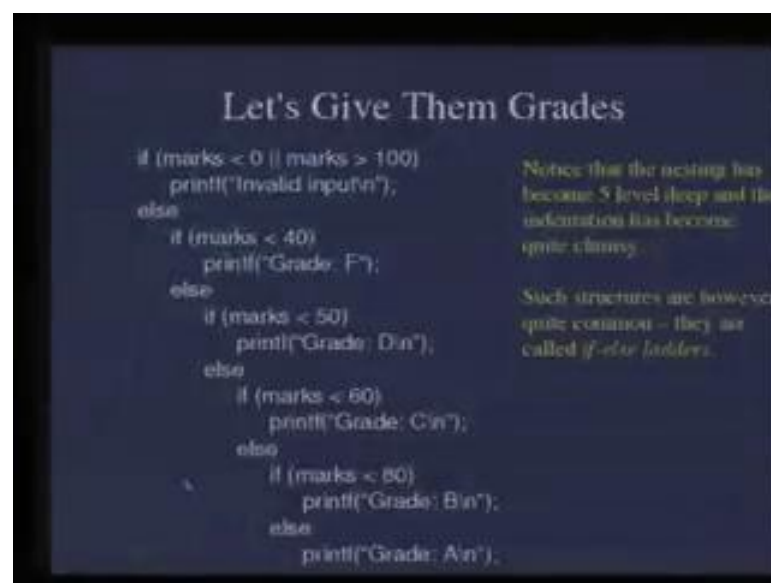
Now, suppose we will also want to check whether the input marks given by the user are indeed between zero and hundred and if not we should print invalid input, remember we talked about program robustness scenario lectures. And we should test all input to make sure that they have sensible values. So, you want to do this then what we need is a nested if else structure because first thing we check whether the marks are between zero and hundred, if not then we print invalid input. Otherwise, we need to check whether the marks are less than forty or not.

(Refer Slide Time: 08:00)



So, here is what the program might look like. If marks are less than zero or marks are greater than hundred then we print invalid input; otherwise, the codes that we have wrote earlier. The entire thing if marks are less than forty printf fail else printf pass. Note that this entire if else statement is a single c statement which we are using as the statement for the else part of the outer if else statement. Now, again since this is the single c statement we do not need to put braces around it to use it as part of another if else statement. Also note that the indentation makes it very clear that this if statement is within the else and this printf is within the 'if', which is of course, within the else.

(Refer Slide Time: 08:56)



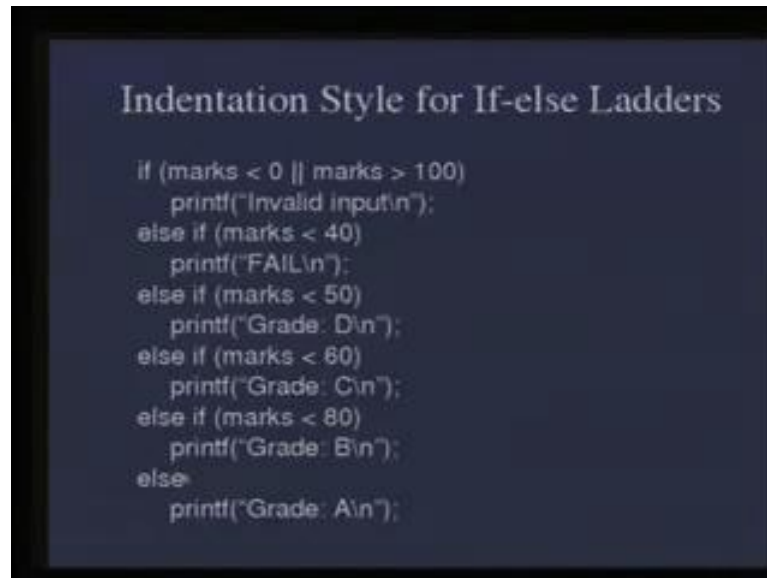
So, now let us take the example little further, and make it little more complicated. Suppose you want to assign grade to student depending upon their marks. If marks are less than forty, let us I want to give grade f if the marks are more than forty, but less than fifty grade D, between fifty and sixty that is fifty or more, but less than sixty then grade C; sixty or more, but less than eighty then the grade B; and eighty or more marks obtained then you want to give the grade A. So, you might use a sequence of nested if else statements. So, if marks are less than zero or marks are greater than hundred, we print invalid input; otherwise, now this entire thing is a single if else statement that is the else part, the else statement of this outer if else statement. So, this state if marks are less than forty print the grade to be F, else if marks are less than fifty then print the grade D, else if the marks are less than sixty grade C, if marks less than eighty grade B, else the grade is A.

Now, you can see that each successive if else statement is the else part of the outer if else statement. So, for example, this if else statement is part, the statement for the else part of this if else statement. And similarly, this if else statement is the else part of the outer one which is this one and similarly this if else statement is part of the entire thing. So, each of these is similar statement and therefore, we do not need braces anywhere. Also note that while execution when we assign the grade D, we have check that the marks are less than fifty, but we have not check whether the marks are more than forty or more or not.

This is because will execute this statement only if this condition marks less than forty is false that because in this else part of the if else statement whose condition is mark less than forty. This means that if we are executing this particular if else statement then the marks are currently to be forty or more. And of course, we are executing any of these statement then we know that the marks are between zero and hundred. And similarly if we executes this if else statement; that means, that the mark must be fifty or more. So, if they are less than sixty then we will give grade c and so on so forth.

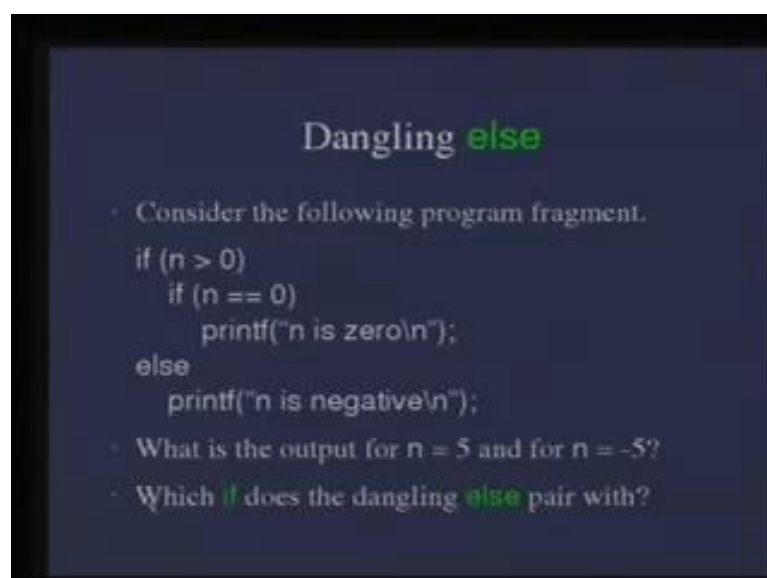
Note that this slide become little cumbersome because the nesting levels of the if else has become little. We have an if else within which we have a second one within which we have a third one within which we have fourth one within which we have the fifth one and correspondingly the indentation is becoming more and more. Now, this make it both make the program both easier harder to write as well as harder to read it has the indentation to some quiet clumsy; however, you will find that in practice such kind of nested if else statements are quiet common and they are called actually if else ladder. So, for these if else ladder we prefer an alternate indentation style, which is more readable. But of course, somewhere exactly the same thing. Let us see that in next slide.

(Refer Slide Time: 12:28)



So, this is the preferred indentation style for if else ladders where essential what has happened is that the if condition the if statement which is the part of the previous else statement is written in the same line the condition is written in the same line, as the key word else at all level. So, what has happened with this indentation style is that regardless. So, what is the nesting level of the if else's the indentation remains only at one level. And also this indentation style makes it very clear for the reader to understand that exactly one of these statements will get executed that is either this statement will execute or this statement is execute or this statement will executed and so on so forth.

(Refer Slide Time: 13:49)



Let us now consider the common problem associated with this if else statement and that is called dangling else problem. Let us consider the following program if n is greater than zero then within that if n equal to zero printf n is zero else printf n is negative the question I want to ask is what will be the output of the program segment for initial values of n is equal to five and for n is equal to minus five. Answer to that will depends upon whether this else is paired with this if or with this if.

So, let us assume that it is the former that is this that this else is associated with this if as indeed this indentation style were just because the this else I have written at the same level as this is. So, the indentation style suggest that this has matches with this if. So, what happens is that is the case. So, suppose the value of n is five we check whether n is greater than zero which it is. So, we have execute this statement because the condition is evaluated to true. So, therefore, we will execute the if part of the if else statement rather than the else part.

So, this is the statement that we execute and this says if n is equal to zero printf n is zero and is not equal to zero and. So, therefore, this is not executed, but suppose this else is associated with this inner if what could happen in that case. Now, suppose if again the value of n is five. So, when we execute when we evaluate this condition that of course, evaluates to true. So, we have to now execute the statement in the if part which is now this entire if else statement.

So, now if n is five n equal to zero evaluates to false and therefore, the else part is print is executed which prints n is negative that is for n is equal to five what the program prints is n is negative which is of course, probably not what I wanted and similarly we will see that for n is equal to minus five if this else is associated with this if.

And the result is not what we expect because if n is minus five then we first execute this n is greater than zero it is not and there is no else for this if. So, nothing happened whereas, if this else is associated with this if then what happen. Is that if n is equal to zero if n equal to if n greater than zero evaluates to false, and since there is no else part the answer to the question will depend upon whether this is considered by the compiler to pair with this if. Or with this if lets considered first for n is equal to five and lets assume that this else if paired with this if. So, if that happens what happens if the condition greater than zero is evaluated which is of course, evaluate to true.

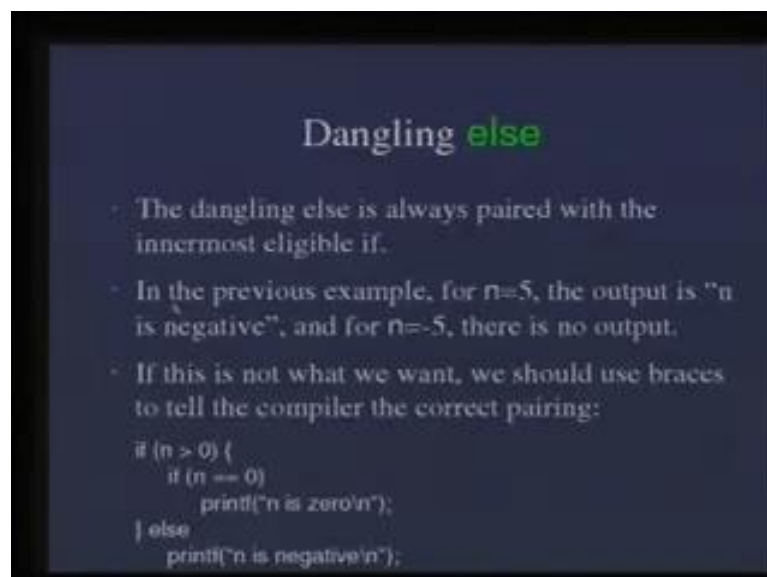
And then this statement get executed and since n is equal to n is not equal to zero nothing is printed, but suppose on the other hand if this is happened to be associated with this if then for the same value of n the result will be different. So, again the condition n greater than zero will be evaluated which evaluates to true again.

And now that is the case this entire if else statement which is within the if part will be executed now this condition again evaluates to false. So, if n is not equal to zero and what is what execute if this printf statement which prints on the screen that n is negative for n is equal to five.

And similarly for n is equal to minus five, we will find that if we trace the program that is this else is associated with the outer if then the output is n is negative, where as if this else is paired with the inner if then there is no output at all. So, there is an ambiguity here because it is not clear whether this as it is associated with the outer if or with the inner if as it turns out to the C language defines that such an ambiguity effort or such an dangling else is paired in the program.

That is then it is always assume to be appear with the inner most eligible if. So, in the example we just saw regardless of how we indented this particular else will be assumed to be appear with inner most if which is this one. So, which means that this program is probably not doing what we wanted to do, because for great for positive values of n it actually prints n is negative and for negative values of n it just does not print anything.

(Refer Slide Time: 18:43)



Dangling **else**

- The dangling else is always paired with the innermost eligible if.
- In the previous example, for $n=5$, the output is "n is negative", and for $n=-5$, there is no output.
- If this is not what we want, we should use braces to tell the compiler the correct pairing:

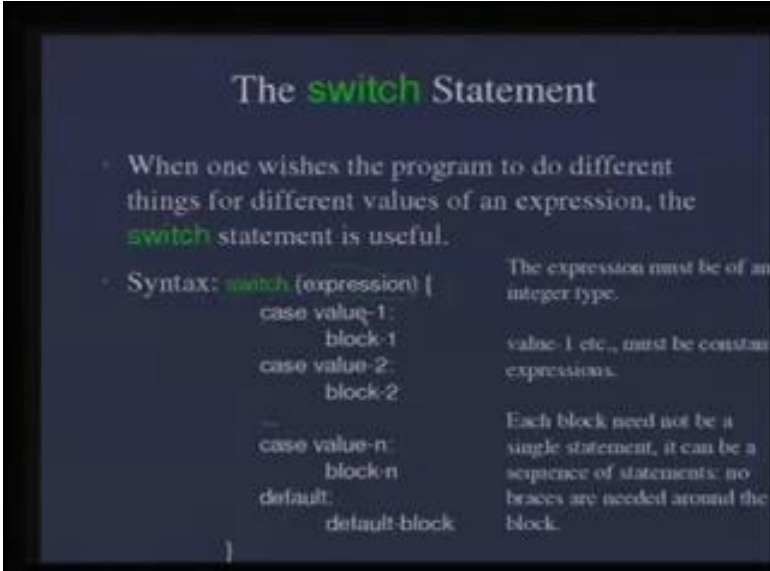
```
if (n > 0) {  
    if (n == 0)  
        printf("n is zero\n");  
} else  
    printf("n is negative\n");
```

So, if this is not what we want as in this example what we can do is this that we can use braces to tell the compiler what the correct pairing is. So, if this is the pairing we want that this else should be paired with the outer if and what we have done is that this if statement does not have a corresponding else we have enclosed it in braces.

And now there is no longer the ambiguity of the dangling else because this else cannot possibly be associated with this if because this if statement is entirely within this set of braces whereas the else is outside the braces.

So, now the program behaves as we expected to behave we will now talk about the next important condition statement which is the switch statement and in many cases it allows us to write a program very elegantly we can of course, do whatever the switch statement can do with if else and if else ladders always. But in many cases the switch statement makes the program much more elegant and similar to read essentially when we have an expression and for different values of the expression that are known in advance we want to perform different action that is where the switch statement is useful.

(Refer Slide Time: 20:05)



The **switch** Statement

- When one wishes the program to do different things for different values of an expression, the **switch** statement is useful.
- Syntax: **switch** (expression) {
 case value-1:
 block-1
 case value-2:
 block-2
 ...
 case value-n:
 block-n
 default:
 default-block
}

The expression must be of an integer type.
value-1 etc., must be constant expressions.
Each block need not be a single statement, it can be a sequence of statements; no braces are needed around the block.

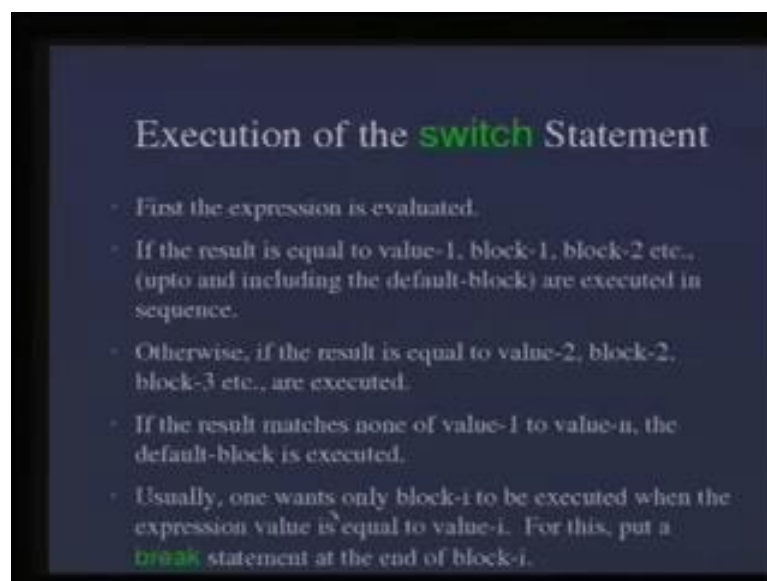
So, this is what the syntax for the switch statement is you have key word switch followed by an in expression bracket followed by within curly braces the curly braces here are essential you have multiple cases.

So, case value one case value two case value n n could be anything and then the default and for each case you could have a block. And now this block is a sequence of statement need not be an single statement and the essential idea is. What this statement capture is that the value of this expression turns out to be equal to that of the if the value of this expression turns out to be value one and you want the set of statements block one to be executed.

If it is equal to value two then we want the set of statement in the block two to be executed and. So, on and if the value of the expression does not in the either of these n values and you want the default block to be executed, but there are there are some facilities in the description that I just mention and we will look at them in a minute.

But before that at the syntactical level this expression must be of an integer types and these values must be constant expression. So, these can be for example, things like x plus y or x or y or both or variable. This has to be evaluate with the value can be evaluated at compile time and as I said each block need not be a single statement unlike if else and if statement it can be sequence of statement and even if there are multiple statement in a block there are no braces needed around the block. So, let us be exactly what the statement does.

(Refer Slide Time: 23:32)



Execution of the **switch** Statement

- First the expression is evaluated.
- If the result is equal to value-1, block-1, block-2 etc., (upto and including the default-block) are executed in sequence.
- Otherwise, if the result is equal to value-2, block-2, block-3 etc., are executed.
- If the result matches none of value-1 to value-n, the default-block is executed.
- Usually, one wants only block-i to be executed when the expression value is equal to value-i. For this, put a **break** statement at the end of block-i.

First expression that we give is going to be evaluated and then it is going to be compared to two values one by one. Of course, the expression evaluates itself, 22:06 had some side effect which will of course, take immediate effect.

So, first the result is compared to value one if it is true if it is equal to value one then the set of statements in block one block two up to block n and followed with the statements in the default block they will all be executed in that sequence now that seems to be 22:27, but that has its uses and we will look at them.

If the value of the expression is not equal to value one then the result is compared to value two, and if the result is equal to value two then block two block three etcetera are executed again up to block n and including n finally, the default block and so on so forth.

If the result is not equal to value two either then we compared with value three and so on. And finally, if the results matches with none of the value value one to value n then the default block alone will get executed. So, essentially that means that the value of the expression is equal to value one then block one block two up to block n and then the default block category to read if it is equal to value two then the block block two block three up to block n the default block is executed, if the value is equal to value n then block n and default block are executed, and if the value of the expression is not equal to any of the then only the default block will be executed.

Of course usually wrong is that we want usually only block I to be executed if the value of the expression matches value I. So, you want that to happen then we need to put a break statement at the end of block I. So, whenever a break statement is encountered in any block within switch then the execution stops right there.

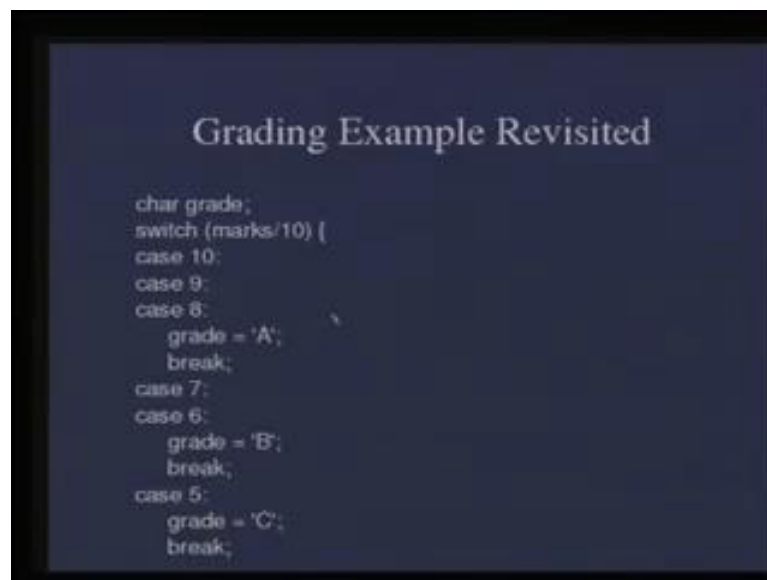
Let us now see an example of the use of the switch statement then we will take the same example as before that of the assigning grade depending on demand. Let us assume that we have already checked the marks are within zero to zero to hundred and you want to assign the grade and you want to do it using the switch statement instead of nested if else now.

Note that we cannot really give Boolean expression here we can only give a six constant value for each case, now you want to do that and use the switch statement of grading problem then one possibility is that.

We could have a case for every possible mark between zero and hundred that is the case for marks equal to zero case for marks equal to one marks equal to zero and so on up to case for marks equal to hundred. And that of course, make the program extremely and widely, but fortunately in this example we will recall that the grades are the grade changes as multiples of ten.

So, the transition for d grade to c grade happens that marks when the marks is more than fifty or fifty. So, we can exploit that. So, if we divide the marks by ten and then look at the various case if we divide the marks by ten then the resulting value will be only be between ten and zero.

(Refer Slide Time: 25:35)



So, you can exploit that as to reduce the number of cases now because that if the marks are eighty or more then you want to assign the A grade if the marks are eighty are more than the marks divide by ten remember that mark is an integer variable. And therefore, marks divide by ten only the quotient says for example, the marks happened to be eighty seven and then marks by ten will be eight it, it will not be eight point seven.

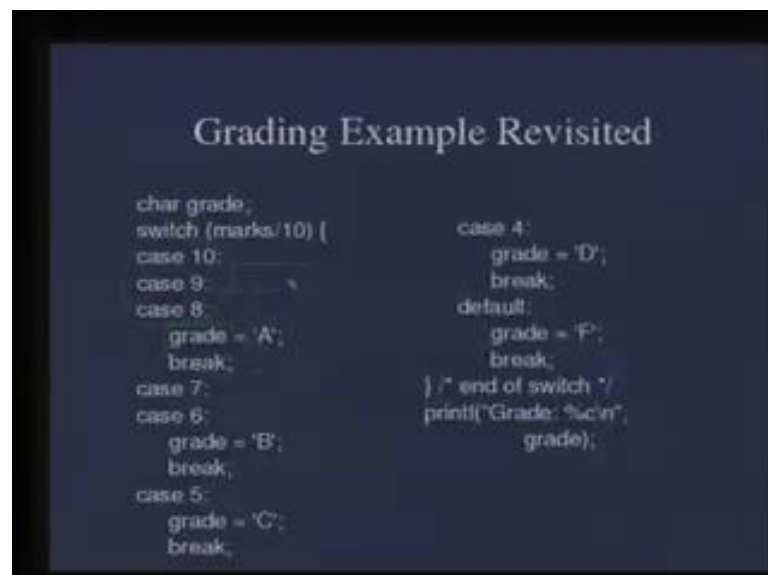
So, in these three cases we want to assign the grade A because in these three cases marks are at least eighty and up to hundred. So, in this case ten what we have done is we have not put any statement in the corresponding block. Similarly in the case nine we have not put any statement in the corresponding block.

And in the case eight we have assigned the variable grade to A and then the break statement what that means, is that suppose the value of marks happen to be 95. So, mark by ten will evaluate to nine we does not match ten, but its matches nine.

So, the statement in the block corresponding to case nine will execute will be executed there is no statement over here and remember that there is no break statement over here either which means that the control false true as it is called. So, the next case without really worrying about whether the value in the next case is equal to the value of the expression or not.

So, even if the value of marks by ten is nine this block corresponding to case eight still get executed because the block for case nine there is no break statement. So, which means that the value of grade becomes A and then there is a break statement, which means that immediately after signing the value of grade to A the execution of the switch statement finishes and the control comes to the next statement following the switch which prints the value of the grade. And similarly if the marks are exactly hundred the candidate got a perfect score then the statement in the block corresponding to case ten would be executed there are no statement here.

(Refer Slide Time: 27:57)



So, nothing happens and there is no break statement the control falls to automatically to the next case which is nine again there are no statements to be executed here. So, nothing happened and there is no break statement which means the control falls to again to the

next case. So, grade is assigned A then the break happens which means switch statement finishes the execution.

And similarly if the value of marks by ten is seven or six. then we assign the grade b which means that from marks from sixty to seventy nine the grade b will be assigned if the value of mark by ten is five, which means the marks are between fifty and fifty nine then the grade is C, if the marks are between forty and fifty nine the grade is D.

And we do not have separate case for zero to three they can all be covered in the default case, because if the value of mark by ten not four five six seven eight nine or ten then it has to be zero one two or three, because as we said we are assuming that we have already checked that the value of marks is at least zero is at least zero and at most 100.

So, in the default case which mean the value of mark by ten is anything other than four to ten which in this case mean that it will be between zero and three the grade is F and this break statement in the last is not really required, because there is no next block after this regardless of whether there is break or not no difference will be there in the execution. But as good programming practices it is a good idea to put the break statement even at the end of the last block, because it later one has some block then we might forget to add the break at that point in time.

So, finally, this is the end of this statement and this switch statement will result in assigning the grade A to F to the variable grade and this printf statement can print the grade. So, this is the end of the lecture. In the next lecture we will look at in more detail the control statements used in implement loops we have already seen the while loop in the programs that we have written we see more kind of loop statements and certain other statement is alter the behavior of the control flow within the loop and so and so forth.