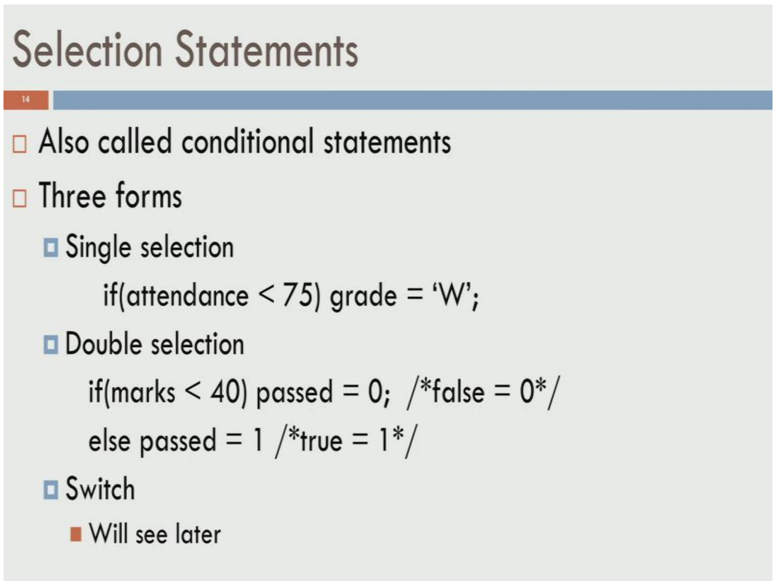
Programming, Data Structures and Algorithms

Prof. Shankar Balachandran
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 06
Conditional Statements
Selection Statements
If statement, If then else, Cascading if
Switch statement; Correct and incorrect usages

We are in lecture 2, we will now look at Conditional Statements. So, as I explained earlier conditional statements are of two kinds, they are of the, if then else kind or the switch case kind.

(Refer Slide Time: 00:26)



Selection Statements

14

- Also called conditional statements
- Three forms
  - Single selection

    ```
    if(attendance < 75) grade = 'W';
    ```
  - Double selection

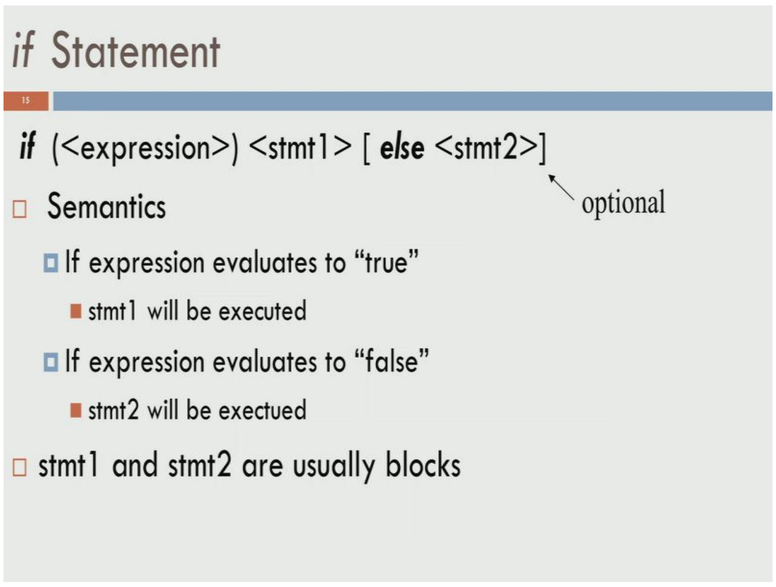
    ```
    if(marks < 40) passed = 0;  /*false = 0*/
    else passed = 1 /*true = 1*/
    ```
  - Switch
    - Will see later

So, will start with the conditional statements, these are also called selection statements for the reason that you actually have an expression, based on the evaluation of the expression, you actually select what should be done. So, for example, there is the single selection statement. So, let us see this example, if attendance is less than 75 grade equals w, so what this is really doing is this, if attendance is less than 75, if the class attendance is less than 75 percentage, grade is withdrawn. That is what this statement is supposed to be.

So, if attendance is less than 75 grade equals w. So, this is does not have any other choices, there is no else here. So, this is a single selection statement, so you choose to assign w to grade or not depending on this expression attendances less that 75. If attendances is actually greater than or equal to 75, this assignment will not be chosen for execution, which means it will not be executed, so this is a single selection statement.

A double selection statement on the other hand has two things that you could do, you could have, if marks less than 40 passed equals 0.. else passed equals 1. So, you have two statements passed equals 0 and passed equals 1 and one of them will be executed depending on the choice of marks. So, if marks is less than 40 passed equals 0 will be executed; otherwise passed equals one will be executed. So, there is semicolon missing here that has to be included. And otherwise it can be a switch statement, where it is multiple selection. So, you have single selection, you have double selection or you have multiple selection. So, we will first see this single selection and double selection and query details before we go into the switch.

(Refer Slide Time: 02:21)



if Statement

15

if (<expression>) <stmt1> [***else*** <stmt2>]

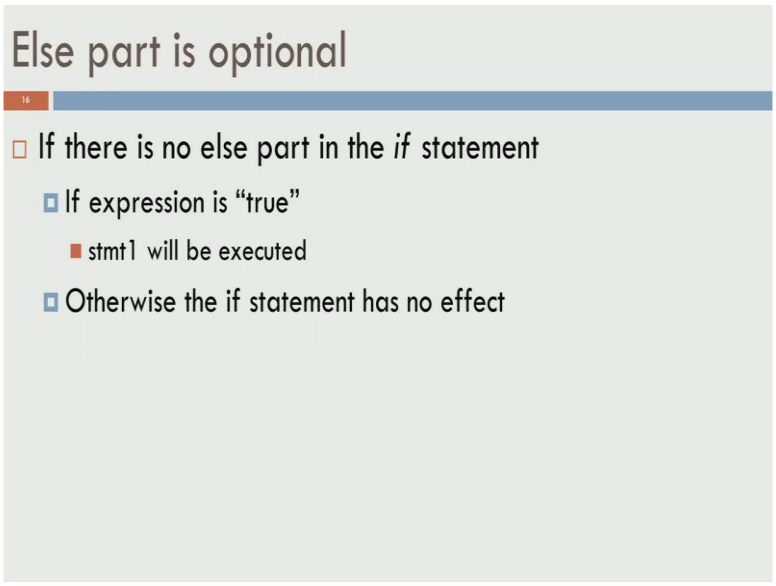
optional

- Semantics
  - If expression evaluates to "true"
    - stmt1 will be executed
  - If expression evaluates to "false"
    - stmt2 will be exectued
- stmt1 and stmt2 are usually blocks

Let us look at the basic structure of a if statement. So, the if statement is start with these keyword called if followed by an expression, followed by a block called statement 1, optionally it can also have the else clause and a block for the else clause called statement 2. So, the meaning of this is, if expression evaluates to true, statement 1 will be executed. So, the statement 1 is, remember it is a compound statement, it is not a single statement stmt 1 is suppose to be a compound statement.

So, compound statement 1 will be executed, if the expression evaluated false, this statement 2 will be executed. So, stmt 1 and stmt 2 are usually blocks of code need not be just single line code, even though single line code is acceptable need not be single statement code.

(Refer Slide Time: 03:14)



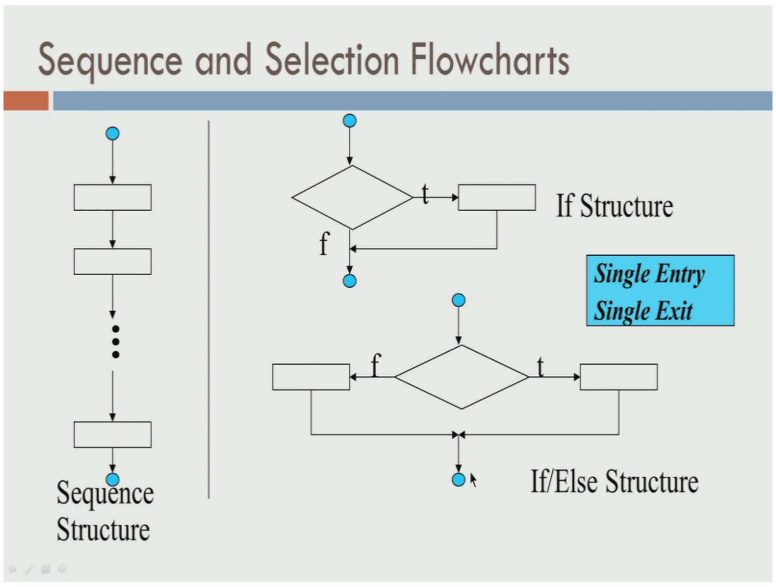
Else part is optional

16

- If there is no else part in the *if* statement
  - If expression is “true”
    - stmt1 will be executed
  - Otherwise the if statement has no effect

So, the else part is completely optional and that is what we have, when we have this square bracket here. Square bracket is not part of the syntax, it is just to tell you that everything that comes with in square bracket is optional here. So, if there is no else part in the if statement, this becomes a single selection statement and if the expression evaluates to true statement 1 will be executed; otherwise, the statement would have no effect.

(Refer Slide Time: 03:43)



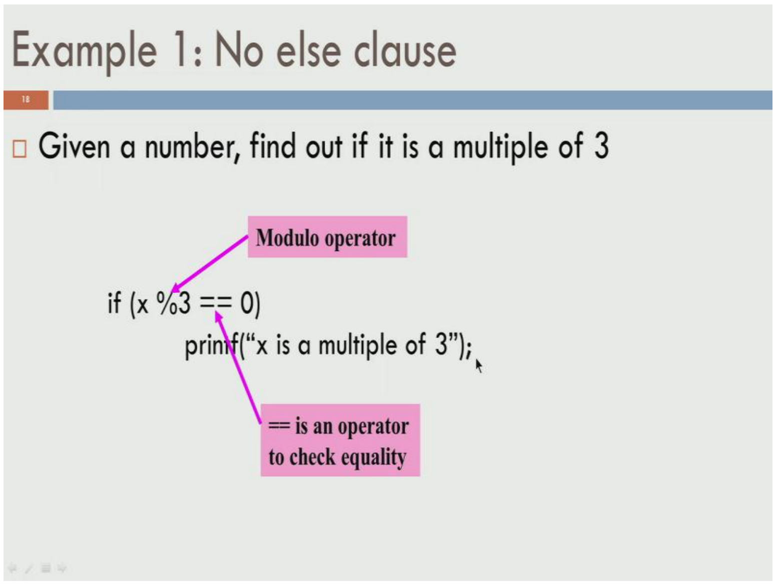
Sequence and Selection Flowcharts

So, what it really does in terms of execution of programmers, let us say I do not have any of these selection statements, you have series of simple statements. If you start from, let us say line 1, you will do line 2, line 3 and so on up till line n, the last line in the program and exit. So, this is a sequential structure whereas, once you have selection statements, it could change the flow of the program.

So, let us say there is some piece of code here in this blue dot here and you evaluate an expression, which is indicated by the diamond if the condition evaluates to true, you execute the block of statement in the true branch, and if the condition evaluates to false, you do not do anything and either way after executing this you come here or if the condition is false you come here. So, this branch that you have here over the two case will not happen if the expression evaluates to false and it will goes to the blue dot.

So, this is the single selection statement, you choose to execute this true block or not depending on the expression here. Then, there is this other type which is double selection statement, you have a body of code here and after that evaluate an expression. So, this body of code could be scanning something from the user or some other piece of program that you have written before, you evaluate an expression. Now, there are two choices either true or false, if the condition evaluation to true you executed the right block, if the condition evaluation to false you executed the left block. And once you execute one of these blocks, you come and execute these blue block of code which is after the if statement. So, this is the general structure of if than else.

(Refer Slide Time: 05:34)



Example 1: No else clause

- Given a number, find out if it is a multiple of 3

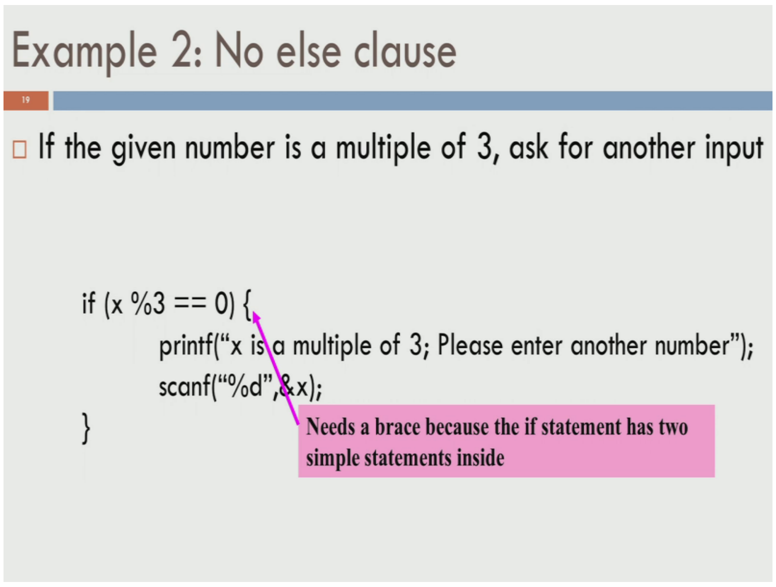
```
if (x %3 == 0)
        printf("x is a multiple of 3");
```

So, let us look at several examples to now. So, starting with the example 1 where there is no else clause. So, let us say the problem that I want you to solve is given a number find out if it is a multiple of 3. So, I give you number x and you have to find out whether it is multiple of 3 if it is multiple of 3, print that it so on the screen. So, the equivalent code segment would be if x percentage 3 equal to equal to 0 printf x is the multiple of 3.

So, this printf is a simple statement here and this if is a compound statement, which contains only one selection. If the condition is true, it will print this on the screen, if the condition is false it will not do anything. So, remember this percentage as the modulo operator that are introduced in lecture 1. This percentage 3 means, we are looking at x mod 3 and this is something that is new here, which is this equal to equal to.

So, this is equal to equal to is essentially an operator in C which checks for equality, a single equal to symbol is useful for expression evaluation. So, we did this in the previous lecture, so p 2 equals something, p 1 something and so on. So, evaluate something on the right side and assign it to the left side, but if you have two equal to symbols right after the other, that is checking for equality. So, the meaning of this expression is, if x modulo 3 is equal to 0, then do this, else there is no else statement here. So, if this condition is true, the statement will be executed.

(Refer Slide Time: 07:22)



Example 2: No else clause

- If the given number is a multiple of 3, ask for another input

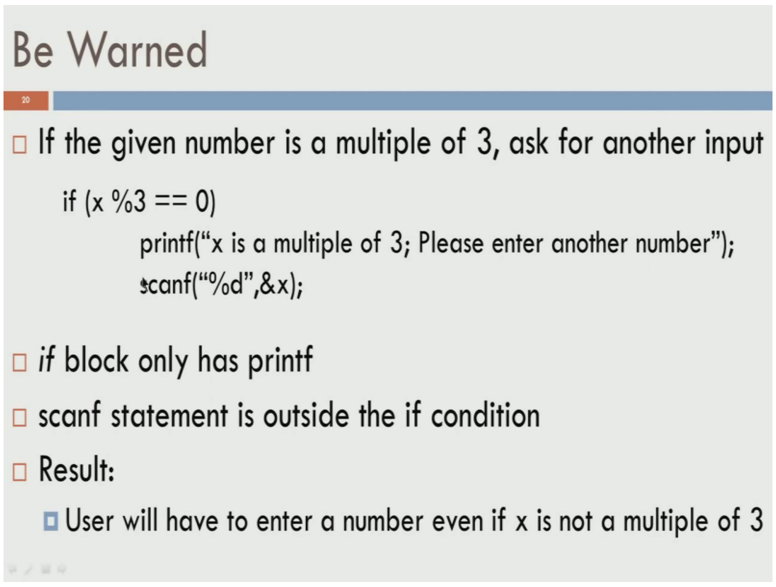
```
if (x %3 == 0) {
        printf("x is a multiple of 3; Please enter another number");
        scanf("%d",&x);
}
```

Needs a brace because the if statement has two simple statements inside

Let us move to another example, where there is no else clause, but it is a compound block. So, if the given number is multiple of 3, let us say you want ask the user for another input. So, if x percentage 3 is 0 you not only want to tell the user that x is the multiple of 3, you also want user to enter another number and you should able to scan it. So, if x percentage 3 equal to equal to 0, so then block is between this and this brace.

So, printf x is the multiple of 3, please enter another number. So, you are prompting the user to enter another number, since the user is expected to enter another number, you have to scan it, scan of percentage d percent x. So, this is the collection of two simple statements that goes into a code block and if this condition is true, you have a logical collection of statements here are a logical sequence, we not only want to print it, we also want to user to enter another input. So, you need a brace because there are two simple statements inside.

(Refer Slide Time: 08:30)



Be Warned

- If the given number is a multiple of 3, ask for another input

```
if (x %3 == 0)
        printf("x is a multiple of 3; Please enter another number");
        scanf("%d",&x);
```

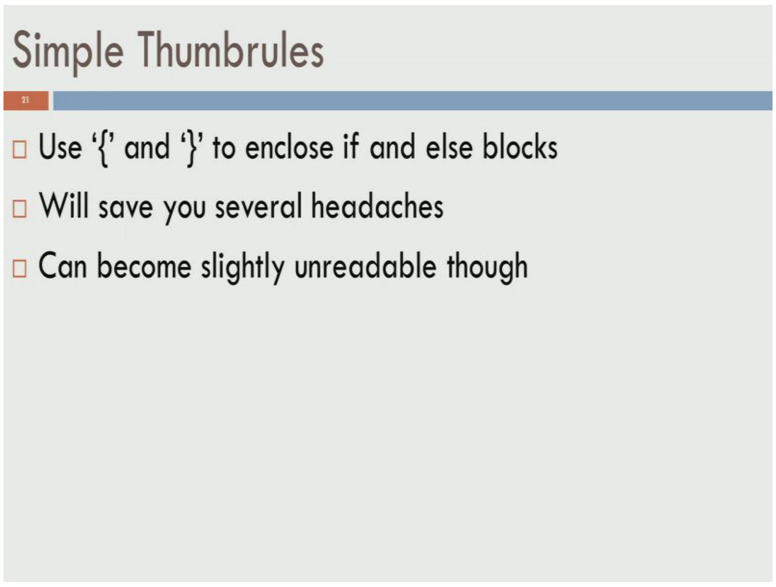
- *if* block only has printf
- scanf statement is outside the if condition
- Result:
  - User will have to enter a number even if x is not a multiple of 3

So, there is a little bit of warning that I want to give here, let us say I want to solve the same problem, but I left out the braces that is what you seeing here, if x percentage 3 is 0 then I have left out the brace here and here. So, from the view of the compiler, the if statement x percentage 3 equals to 0 printf. That means, instead of becoming a compound statement with two lines of simple statements, it instead becomes one compound statement of if only this printf is evaluated if this condition is true and this scanf statement is executed unconditionally.

So, what I mean by that is this printf will happen only if x percentage 3 is actually 0 or if x is a multiple of 3 and this scanf will happen no matter x is modulo 3, x is modulo 3 is 0 or not. So, scanf statement is actually outside the if condition, even though it is formatted in such a way that is printf and scanf seems to be inside the if block, the compiler does not care about the spacing that you given here, this scanf is actually is outside the if condition here.

So, the result would be the user will have to enter a number, even if x is not a multiple of 3 and that is not what we want, we wanted the user to enter a new number only if x is a multiple of 3; otherwise, we want to do something else with that.

(Refer Slide Time: 10:05)



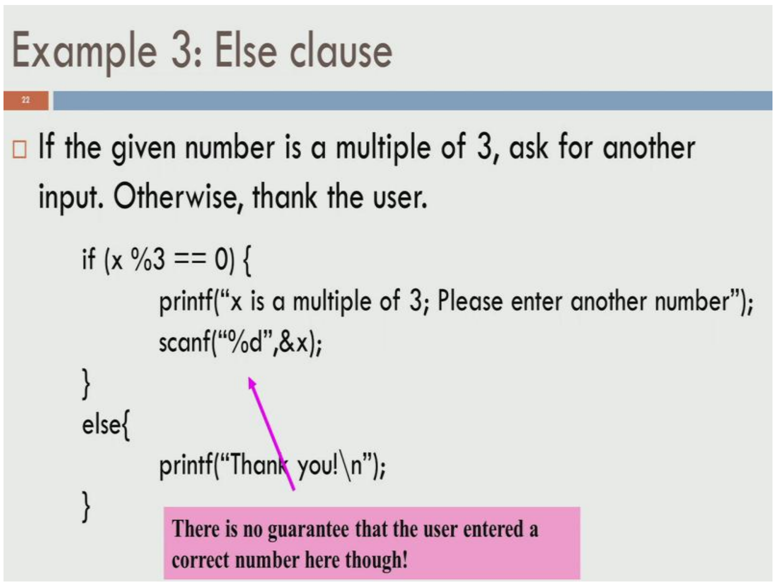
Simple Thumbrules

21

- Use '{' and '}' to enclose if and else blocks
- Will save you several headaches
- Can become slightly unreadable though

So, the simple thumb rules use the left and right brace to enclose if and else blocks. And I typically use it, even if there is only one single statement inside, this will save you several headaches the program can becomes slightly unreadable, because you have two many of this left and right braces. However, it will cause you lesser headaches. So, that is why if you go back to my lecture 1, you will see that is if then else clause that I had, even though there were only single statement inside always enclosed them within braces.

(Refer Slide Time: 10:37)



Example 3: Else clause

22

- If the given number is a multiple of 3, ask for another input. Otherwise, thank the user.

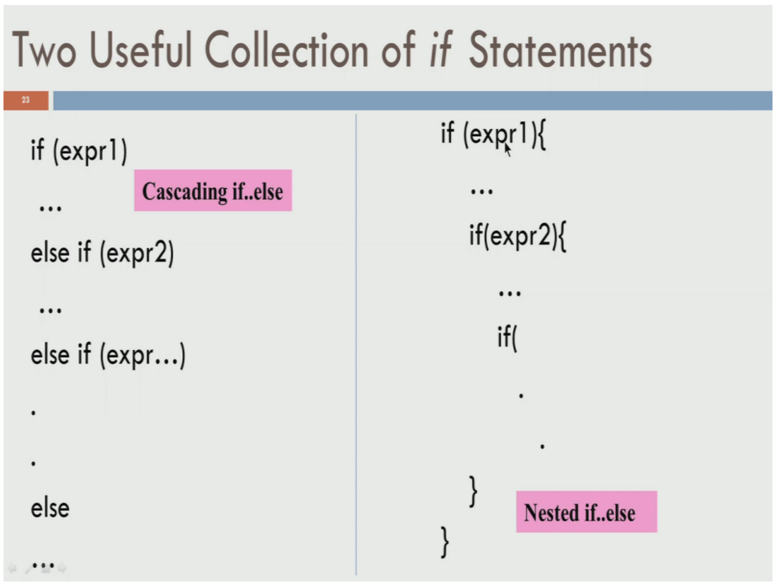
```
if (x %3 == 0) {
        printf("x is a multiple of 3; Please enter another number");
        scanf("%d",&x);
}
else{
        printf("Thank you!\n");
}
```

There is no guarantee that the user entered a correct number here though!

Let us look at another example with an else clause here. So, if the given number is some multiple of 3 ask for another number; otherwise, let us say I want to thank the user. So, if x percentage 3 equal to 0, then you print this on the screen and you scan the another input. So, this is one logical condition, now you have multiple selection though or double selection else. So, if this is not true, you printf thank you, so this is the structure for else and we have seen this before in our lecture 1 also.

So, one word of caution here, so just because you are scanning here it does not mean that the user would enter number, which is not a multiple of 3, the user may still enter a number which is multiple of 3, we are not checking here. So, I want to show this program just show the structure of, if then else. So, at the end here x could still be a multiple of 3. So, you need the mechanism by which you repeatedly scan the input here, till the user input something else, if that is what you want. In this case we are not checking for that, I just want to show you the structure of if then else.

(Refer Slide Time: 11:45)



Two Useful Collection of *if* Statements

```
if (expr1)
 ...
else if (expr2)
 ...
else if (expr...)
.
.
else
 ...
```

Cascading if..else

```
if (expr1){
   ...
   if(expr2){
      ...
      if(
         .
            .
   }
}
```

Nested if..else

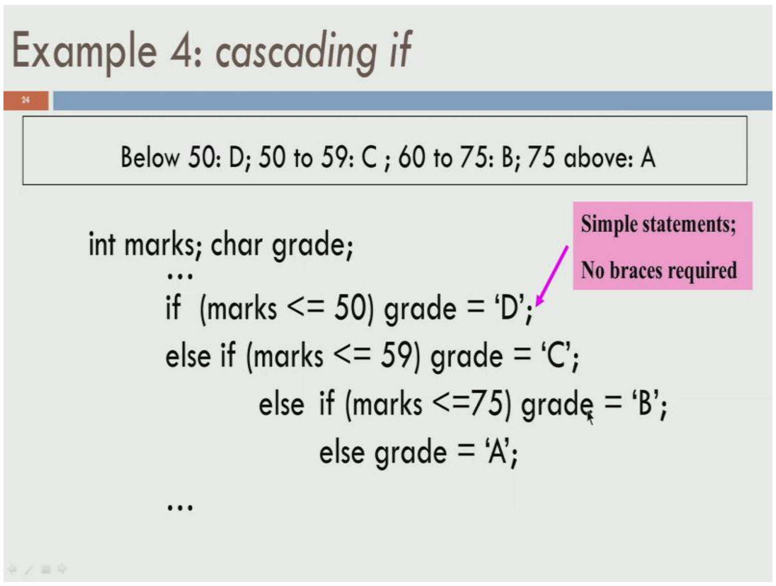
So, there are two other useful collection of if statements. So, one that you see on the left side is what is called a cascading if else and one that you see on the right side is a case of what is called nested if else. So, in the cascading if else case what you have is, if expression 1 evaluates to true, you execute the block of statements. In the else clause you actually evaluate another expression, else if expression 2 evaluate you execute another blob of code, else if expression 3 execute another blob of code and so on.

You can have a series of this else if statements, ending with a final else clause where execute another blob of code. So, this is called cascading if else, we will see a quick example later, the other cases nested if else. So, we have if expression 1 do a blob of code, if expression 2 do a blob of code and so on. So, what happen here is, this blob of code or this blob or this blob only one of them will execute depending on the conditions for expression 1, expression 2 and so on.

So, if expression evaluates to true, this blob of code will execute and none of this will get executed. If expression 1 is false and if expression two is true only this blob will get executed and so on. So, in this cascading if else case, there is only one blob which will really execute not all of them. However, if both expression 1 and expression 2 are true, both this blob and that blob will be executed here.

So, what is really happening is, if you notice if expression 1, there is a curly brace here and this curly brace here, between these two curly braces is iif expression 2, which means this is the compound statement, which is within another compound statement. So, this is completely contained within the evaluation of this expression. So, therefore, this blob of code as well as this statement will be executed if expression 1 is true.

(Refer Slide Time: 13:46)



Example 4: *cascading if*

Below 50: D; 50 to 59: C ; 60 to 75: B; 75 above: A

```
int marks; char grade;
      ...
      if  (marks <= 50) grade = 'D';
      else if (marks <= 59) grade = 'C';
             else  if (marks <=75) grade = 'B';
                 else grade = 'A';
      ...
```

Simple statements;

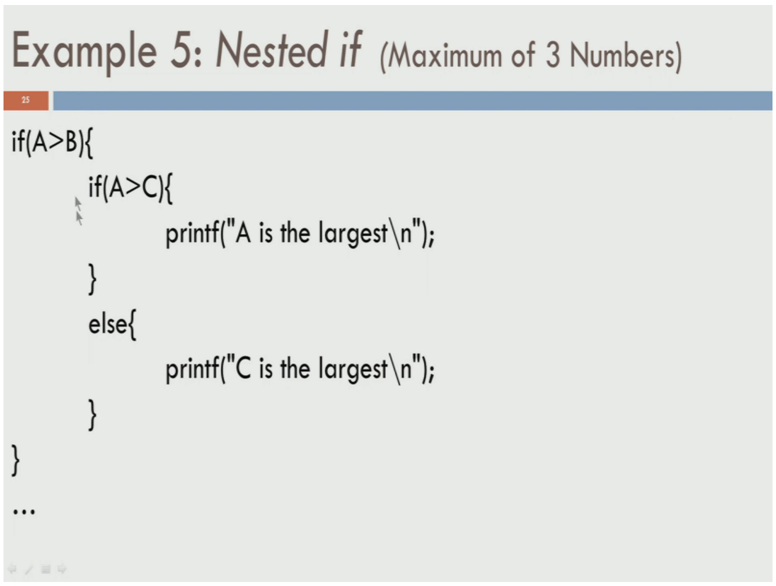
No braces required

So, let us see an example of cascading if, let us say I am grading your course and let us say this is my grading policy, if you get below 50 marks in the exam you get a D grade, if you get between 50 and 59 you get C, if you get 60 to 75 you get B and 75 and above will be A. Let us say this is my grading policy for the course, then I have the integer marks and I have the character grade. So, marks is, something that you are going to enter and grade is something that I am going to assign from the program.

So, you see this structure here, if marks is less than or equal to 50 then grade is D. So, if it is less than or equal to 50 I assign the grade D, else if marks is less than or equal to 59 then grade is C, else if marks is less than or equal to 75 grade is B else grade is A. So, what is happening is, we are seeing only one of these will be true, if you enter the mark here only one of these things will be true and only one blob of code will execute, either you will get grade equal to D or C or B or A, you can never have a case were more than one assignment happens.

So, if my mark is 40 then this if condition is true, then the grade would be D, if the mark is 65 then if this expression will evaluate false, this expression will also evaluate to false, this expression will evaluate to true, you will get grade B. So, this is the structure of cascading if and these are simple statements; therefore, I have not put braces, but putting braces is always recommended. So, there should have been brace here and brace here, there should have been brace here and brace here and so on.

(Refer Slide Time: 15:30)



Example 5: *Nested if* (Maximum of 3 Numbers)

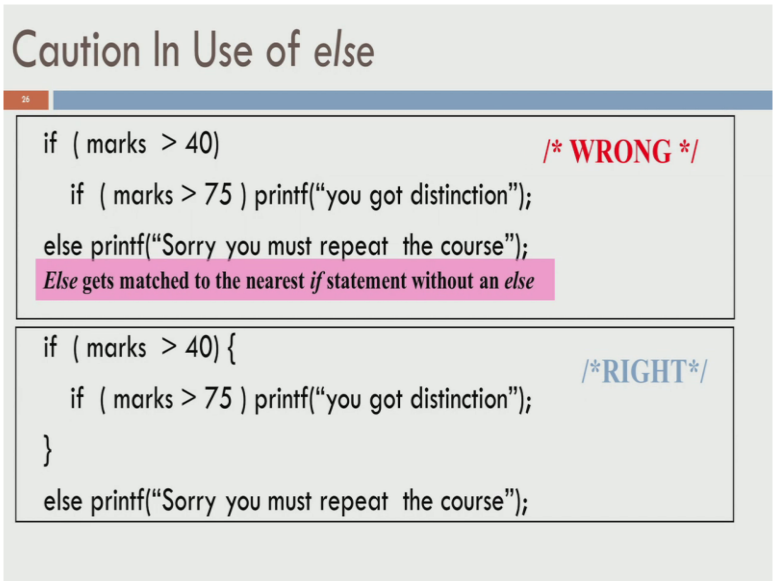
25

```
if(A>B){
        if(A>C){
                printf("A is the largest\n");
        }
        else{
                printf("C is the largest\n");
        }
}
...
```

And nested if, we saw this segment earlier, if A is greater than B and if A is greater than C printf A is the largest else pritntf C is the largest. And there is some other blob of code that comes as else, so I am just showing one segment of the maximum of three numbers.

So, if you are in doubt go back to the lecture 1 and the look at the code, we already used the nested if statement inside the code there.

(Refer Slide Time: 15:56)



Caution In Use of *else*

26

```
if  ( marks  > 40)                                    /* WRONG */
    if  ( marks > 75 ) printf("you got distinction");
else printf("Sorry you must repeat  the course");
```

Else* gets matched to the nearest *if* statement without an *else

```
if  ( marks  > 40) {                                  /*RIGHT*/
    if  ( marks > 75 ) printf("you got distinction");
}
else printf("Sorry you must repeat  the course");
```

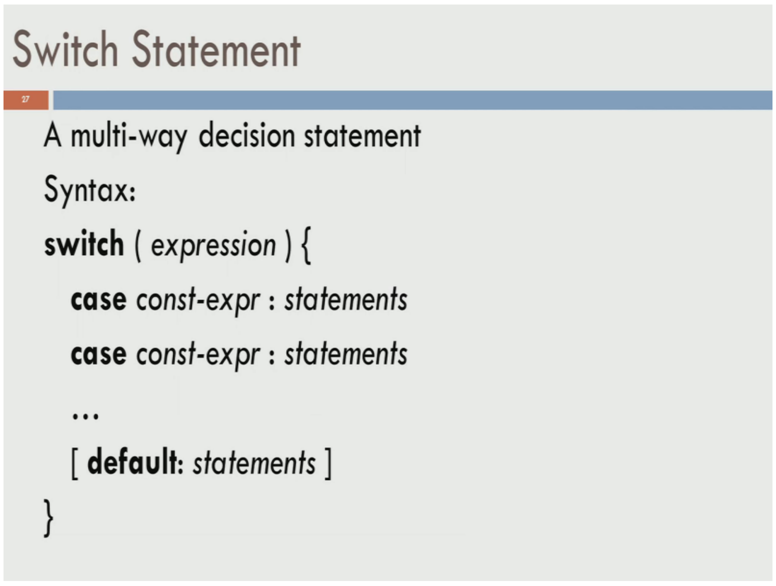
You should be a bit cautious about the use of the else clause. So, let us say I did this, if marks is greater than 40 and within that if true condition, I have if mark is greater than 75 printf you got distinction, else printf sorry you repeat the course, let us say this is what we have. So, what we really want to do is, if somebody got less than 40 we want to tell them that they have to repeat the course. But, let us see what happens here.

So, if we start with this if statement, there is an expression here marks greater than 75 printf you got distinction, if marks is less than 75, even if it is greater than 40 what happens is, this else clause attaches itself with this if clause, and not with this if clause. So, else always attaches itself to the nearest statement, nearest if statement, without the else clause.

So, because of that what happens is, let us say my mark was actually 65 it is not greater than 75. So, I do not get distinction, however I will be asked to repeat the course which is not what we intended. What we really intended is this, if marks is greater than 40 and if the marks is greater than 75, printf you got distinction and the else should have been for this if condition. So, since else always attaches itself the nearest if, not having this brace here even though it is one single statement here, not having this brace here would result in something that you do not desire to have.

So, be cautious about this that is why I said, whenever you have an if condition, if you automatically put braces and for else also if you automatically put braces none of these confusions will occur. So, go back to my basic thumb rule, if true block should have braces, else false block should have ((Refer Time: 17:49)) block should have braces by default and that will help in getting rid of all this headaches.

(Refer Slide Time: 18:01)



Switch Statement

A multi-way decision statement

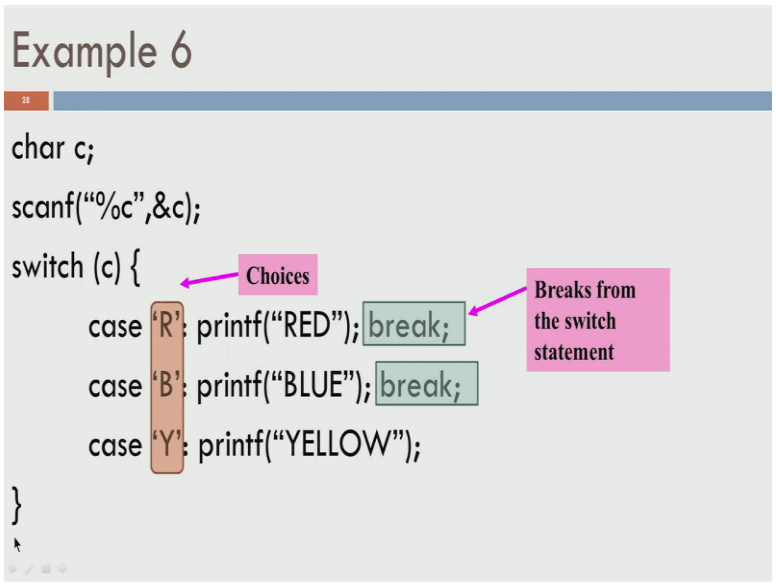
Syntax:

```
switch ( expression ) {
   case const-expr : statements
   case const-expr : statements
   ...
   [ default: statements ]
}
```

So, finally, let us look at the switch statement which is a multi way decision statement. So, in a multi way decision statement it is not a single decision or double decision, it could be more than two decisions. The basic syntax is you have this key word switch followed by an expression, which is within parenthesis and you have braces. So, again this is a block of code. So, switch is a compound statement and it can contain several cases.

So, you have case constant expression statements, case constant expression statements and so on, you can have several cases and you can also have an optional default clause and an optional default. So, essentially what happens is, if we have multiple choices depending on the value of expression, whichever evaluates to true those statements will get executed. If none of these expressions evaluate to true, then the default clause will kick in and this set of statements will get executed. So, let us see examples.

(Refer Slide Time: 18:58)



Example 6

```
char c;
scanf("%c",&c);
switch (c) {
        case 'R': printf("RED"); break;
        case 'B': printf("BLUE"); break;
        case 'Y': printf("YELLOW");
}
```

Choices
Breaks from the switch statement

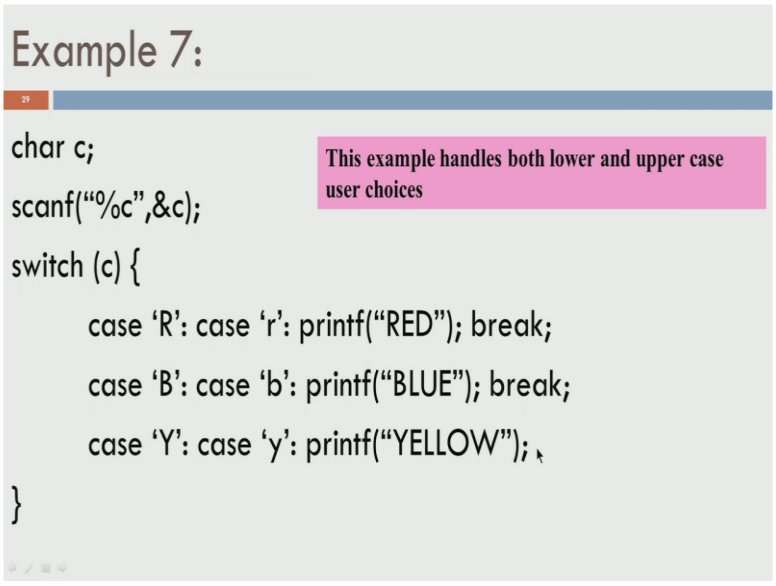
So, let us say I have a character c and I read this character from the user, if the character is one of R, B and Y, I want to print something on this screen. So, if the user input capital R as an input I want to printf RED, if the user inputs B I want to printf BLUE, if the user inputs Y I want to printf YELLOW. So, if we look at this there is a switch c followed by braces followed by one or more case statements, the case then there is a constant expression here R, B or Y followed by a colon and there is a statement here, so that is the structure here.

So, switch followed by expressions and there is something within the curly braces, you have case constant expression colon statements. So, you can see that this program is following that structure. Now, let us see what this constant expression is about, so this R, B and Y are not variables. So, when you put something within single quotes, you are actually looking for the character R or character B or character Y these are not variables R, B, Y.

So, remember variables are those that can change value during the execution of the program, whereas these are constants they cannot change their values during the execution of the program. So, c is a variable, but within quotes R, within quotes B and within quotes Y are all character literals they cannot change their values. So, what you are looking for is, if c the character that you got from the user is one of these, then one of these statements should execute.

The other thing that is new here is, this notion called break. So, what happens with this switch clause is, if R is true then it prints RED and you put the break. So, that you do not want any of this other printfs to execute or even the conditions to be checked. So, if you have printf RED and you print the RED and your now out of the switch clause if you put a break here, so this is something that is new. So, these are the choices that you put in R, B and Y and this break, breaks away from the switch statement to outside the switch statement. So, it starts executing a block of code after the switch statement.

(Refer Slide Time: 21:18)



Example 7:

```
char c;
scanf("%c",&c);
switch (c) {
        case 'R': case 'r': printf("RED"); break;
        case 'B': case 'b': printf("BLUE"); break;
        case 'Y': case 'y': printf("YELLOW");
}
```

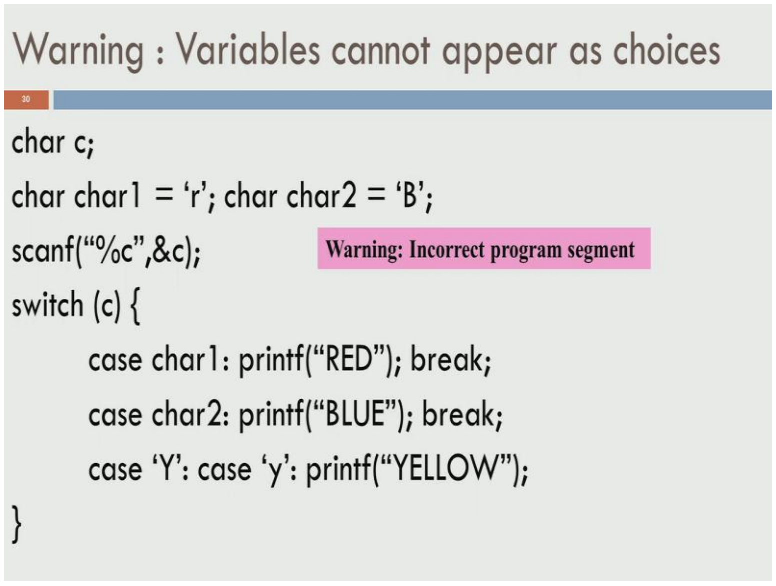
This example handles both lower and upper case user choices

So, let us look at another example where we want to handle both lower and upper case choices, maybe I as a user enter lower case or upper case, I want to handle both of them. So, this is also something that you can do with switch statements, so these two lines are as before, this is also as before the changes in these three lines here to here. So, if you look at case R, R case r the lower case r in both these cases we want to printf RED.

So, I could write this as case capital R printf RED break, case small case R printf RED break and so on. But, we are executing the same set of statement for both these choices small case r and upper case R. So, therefore, if we have a series of choices for which you want to execute the same set of statements, we can just put them back to back and have this set of statement exactly. The same thing we have here for both upper case B and lower case b we want to print blue.

So, therefore, this set of statements is common for both upper and lower case B, similarly this set of statements is true for both upper and lower case Y. So, one thing that you should also notice here is, the last statement here does not have a break. Because, any way it is the last condition you are checking automatically after checking these conditions if both these are true, it will execute this and come out of this switch, if these are false you will any way come out of this switch statement a break is not required in the last choice of case.

(Refer Slide Time: 22:50)



Warning : Variables cannot appear as choices

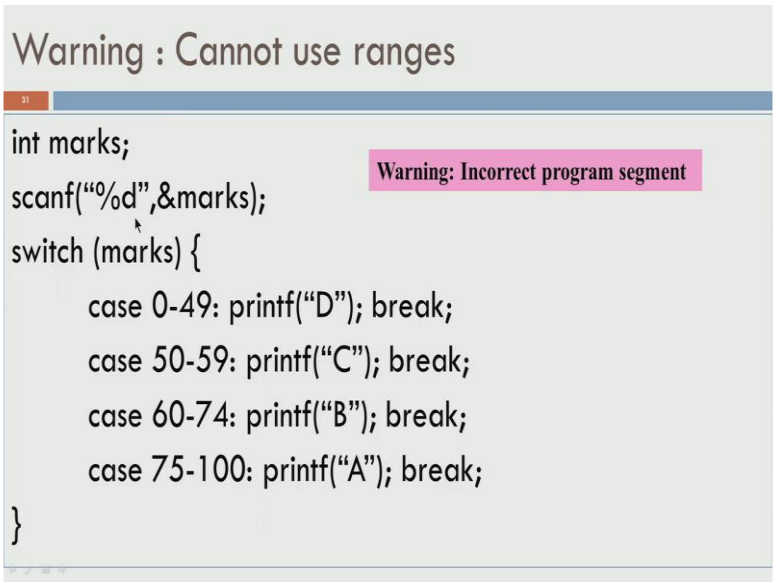
30

```
char c;
char char1 = 'r'; char char2 = 'B';
scanf("%c",&c);
switch (c) {
        case char1: printf("RED"); break;
        case char2: printf("BLUE"); break;
        case 'Y': case 'y': printf("YELLOW");
}
```

Warning: Incorrect program segment

So, there are two things that we have to watch out in the switch cases. So, one warning is that variables cannot appear as choices. So, for example, let us say I put character char 1 is r and character char 2 is B and case char 1, case char 2. So, remember the expression that you have here should be a constant expression whereas, you have use a variable called character 1. Even though character 1 is initialize to a literal r here you are putting a variable character 1 and that is not acceptable. So, c only allows constant expressions to be in the choices, you cannot put variables of any kind here. So, this is an incorrect program segment, so if you want go and try it writing this in a program, you will see that the compiler actually indicates an error.

(Refer Slide Time: 23:38)



Warning : Cannot use ranges

Warning: Incorrect program segment

```
int marks;
scanf("%d",&marks);
switch (marks) {
        case 0-49: printf("D"); break;
        case 50-59: printf("C"); break;
        case 60-74: printf("B"); break;
        case 75-100: printf("A"); break;
}
```

There is another thing that you have to be warned about namely using ranges. So, for example, let us say I want to give these grades and for 0 to 49 I want to give D, for 50 to 59 I want to give C, for 60 to 74 I want to give B and for 75 to 100 I want to give A. Let us say I want that, you cannot do something like this, case 0 dash 49 printf D or 50 dot dot 59 things like what you would do in when you write in paper, you may say 0 dot dot 49 or 50 dash 59 and so on. Those things are not acceptable also you cannot provide a range of values in a constant expression, it has to be a single value for this case clause.

So, whenever you have multiple values you have to have multiple cases. So, that is why we had this here, there were multiple values R and r for which we want to printf RED, we actually have multiple cases also specified explicitly. So, in summary we have a switch statement which has multiple cases and for each of these cases it executes the corresponding block, if there is a break it will break away from the switch statement, if there is no break will go and evaluate the next cases also. So, this is not a valid way to give grades you should use the if then else clauses here, I already showed you this in the earliest slide. So, with this we had the end of module.