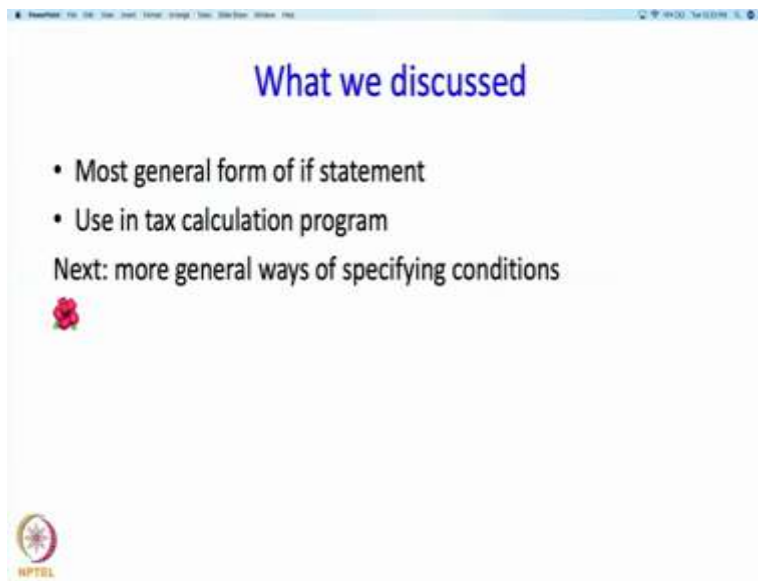


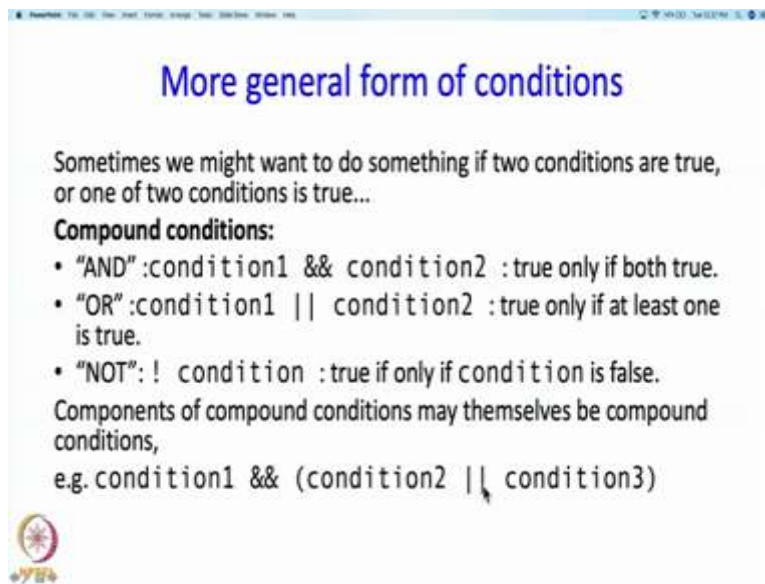
An Introduction to Programming through C++
Professor Abhiram G. Ranade
Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Lecture No. 6 Part – 3
Conditional Execution
Most general form of conditions

(Refer Slide Time: 0:23)



Hello, welcome back. In the previous segment we discussed the most general form of the if statement, in this segment we are going to see a more general way of specifying conditions. Sometimes the simple conditions that we used may not be adequate. So now, we will see a more general way of specifying conditions.

(Refer Slide Time: 00:46)




More general form of conditions

Sometimes we might want to do something if two conditions are true, or one of two conditions is true...

Compound conditions:

- "AND": `condition1 && condition2` : true only if both true.
- "OR": `condition1 || condition2` : true only if at least one is true.
- "NOT": `! condition` : true if only if condition is false.

Components of compound conditions may themselves be compound conditions,
e.g. `condition1 && (condition2 || condition3)`

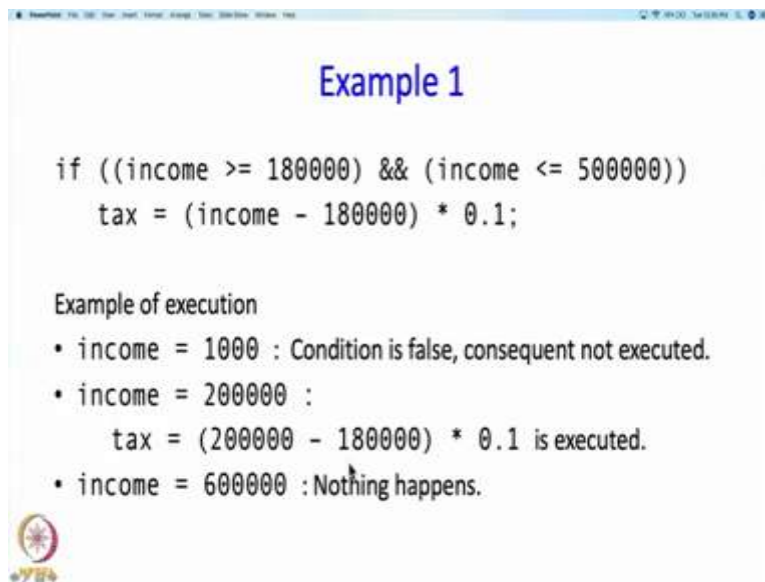


So for example, we might want to say that look do this if some two condition are met rather than just 1 some 2 conditions were true or we might want to say that do this if one of the one of some two condition is true. So for this we need something called compound conditions and this is how we write them. So here is an "AND" compound conditions so the way it is written is condition 1 "&&" condition 2 okay.

So we will see example of this but say if we could we could write this as if $X > 5$ && $Y > 7$ something like that. So when is this condition, is true? Well it is true only if both these conditions are true okay. So if you put this in a "if" statement if say you write "if" this whole thing as your condition as your compound condition then, the consequent is going to be executed if both of these conditions are true okay. I can have an "or" condition as well so an "or" condition looks like condition 1 "||" condition 2 okay and the whole condition this whole compound condition is deemed to be true only if at least if and only if at least 1 of these conditions is true. Both not need to be true if one is true that is good enough. So again if we put this as a condition this entire thing as a condition in an "if" statement then the consequent will be executed if one of these is true and finally we have a "NOT" compound condition this is written as exclamation mark followed by a condition and this is true if and only if this is false. So, sometimes we want that we want to do something if the given condition is false rather then, if it is true. So in that case you can write by putting a "NOT" over here.

Now the components or the conditions included inside compound condition may themselves be compound conditions okay. So for example, you might write something like condition 1 && condition 2 || condition 3 so this is allowed, so how does this work? Well we first evaluate condition 2 || condition 3 or condition 3. So that means we check if one of these conditions is true. So we want one of these conditions to be true and we want condition 1 to be true actually the order in which we do the checking is from left to right so we first check whether condition 1 is true and then we check whether one of these is true. So if this turns out to be false then we do not even have to be worried about checking condition 2 || condition 3 because we know that both need to be true for this entire thing to be true. If one of them is false then the entire thing is false anyway so why bother to check these conditions. So that works here as well. On the other hand if this was an “or” so in this case we would check condition 1 and then if this is true we do not even worry about checking condition 2 || condition 3 because at least one of them has to be true and we have already found one to be true we do not have to worry about checking the second condition okay. Anyway so we can have conditions which are “AND”s and “or”s and we can have even more complex conditions. So compound conditions which are themselves containing compound conditions.

(Refer slide Time: 4:56)




Example 1

```
if ((income >= 180000) && (income <= 500000))
    tax = (income - 180000) * 0.1;
```

Example of execution

- income = 1000 : Condition is false, consequent not executed.
- income = 200000 :
tax = (200000 - 180000) * 0.1 is executed.
- income = 600000 : Nothing happens.

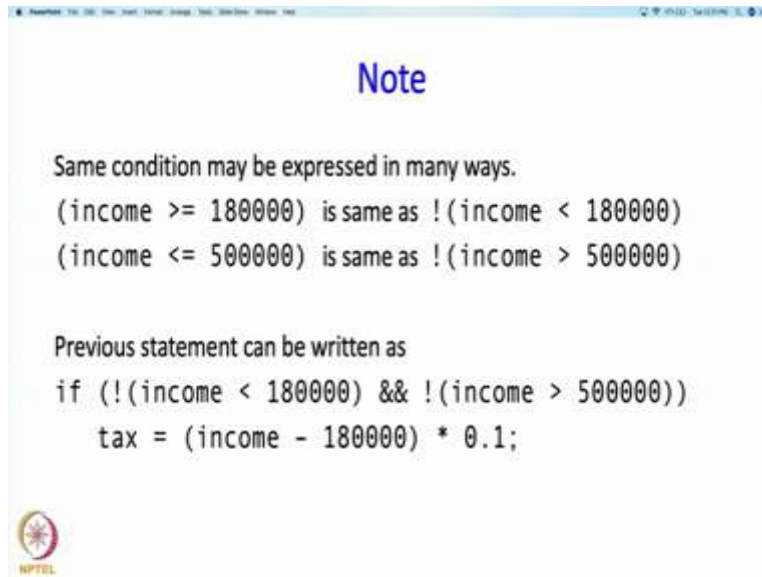


So let us take an example, so I can have a condition if income is greater than 180,000 and income less than 500,000 okay. Do something, so do what? Tax equals to income minus 180,000 excess of income over 180,000, 10 percent of that. So, when will this be executed? So when will be this tax statement be executed, this tax assignment be executed? Well, so suppose income is 1,000 then this condition itself is false okay if this condition is false we do not even worry about evaluating this conditions okay. So because both of these condition need to be true for this entire condition to be true.

So in that case the entire condition is deemed as false if income is 1,000 and so this consequent is not executed or this assignment statement does not get executed. Suppose, income is 200,000 or 200,000 in which case what happens is this condition true 200,000 is indeed bigger than 180,000 so this condition is true. But we do not stop over there because we need to we need to check whether both are true okay. So then we execute this so 200,000 is less than 500,000. So this condition is also true. So as a result a complete compound condition is true so in which case this statement this consequent will get executed. So tax equals, now income is 200,000 so this statement will get executed so we will get this will become 20,000 times 0.1 or the tax will be 2000. Finally, let us say income is equal to 600,000. So in which case what happens is this condition true is 600,000 bigger than 180,000? Yes it is true. So if this is true we should check whether this is also true so in that case 600,000 is not less than or equals to 500,000 so this condition is false. So as a result this compound condition is false and therefore, the consequent is

not executed. So this is not executed so what I mean by nothing happens is that the consequent is not executed.

(Refer slide time: 7:40)




Note

Same condition may be expressed in many ways.

`(income >= 180000)` is same as `!(income < 180000)`
`(income <= 500000)` is same as `!(income > 500000)`

Previous statement can be written as

```
if (!(income < 180000) && !(income > 500000))  
    tax = (income - 180000) * 0.1;
```

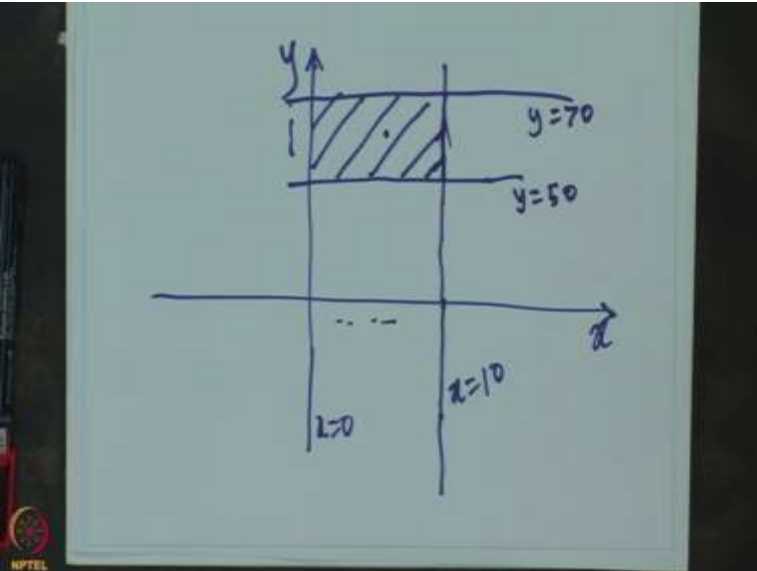


All right, so the same condition may be expressed in many ways. So I can write income greater than or equal to 180,000 or I could have written 180,000 less than or equal to income. But I could also write this as not of income less than 180,000 okay. And similarly, I could write this condition as this and therefore the previous statement can be written by using these forms rather than these okay. So this is just this is just alert you that conditions can be written in the same condition, in the same constrain that we are talking about can be written in, many many ways.

(Refer slide time: 8:29)

Another example

- Consider rectangle lying between lines $x=0$, $x=10$, $y=50$, $y=70$.
- Let (X,Y) denote the coordinates of a point.
- The point is inside the rectangle if $0 \leq X \leq 10$, and $50 \leq Y \leq 70$
- To check this we will write the condition:
 $(0 \leq X \ \&\& \ X \leq 10 \ \&\& \ Y \geq 50 \ \&\& \ Y \leq 70)$
- Do not write $0 \leq X, \leq 10$

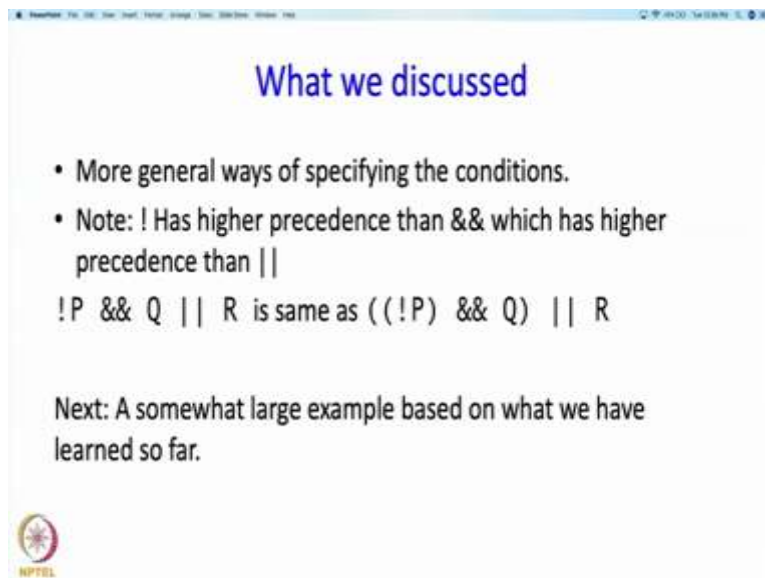


Let me take another example suppose, we have rectangle in X, Y plane so this may, so this will arise in graphics or in geometry. So this is my X, Y plane so this is the X- axis and this is Y- axis in graphics the Y- axis goes down wards but let us not worry about that right now okay. So the rectangle lies between lines X equals to 0 so this is sorry so this is the line X equals to 0 and the line X equals 10 so this is the line say X equals 10 and it also lies between the lines Y equals to 50. So maybe this is the line y equals to 50 and let us say this is the line Y equals to 70. So this is the rectangle that is of interest okay. So so we are going to we are writing program in which somehow say this rectangle is figuring okay. So maybe maybe what is going on is that we have

drawn this rectangle on the screen and then maybe we click and we want to know whether the click point is inside of this rectangle or not okay. So let X, Y denote the coordinates of a point possibly the click point okay, and we want to know whether this point is inside the rectangle so when would a point be inside of the rectangle? So for this, we will require that the X coordinate should be between 0 and 10. So X coordinate must be somewhere in this region, okay and Y coordinate should be between 50 and 70 or between this region or in other words the point should be somewhere inside this. If any of this conditions does not hold, then the point will be outside somewhere. So these are the condition that we need to check. So can we express this condition? So we can write this condition as $(0 \leq X \ \&\& \ X \leq 10) \ \&\& \ (Y > 50 \ \&\& \ Y \leq 70)$ ¹. All right, so if all of this conditions hold then really these conditions are holding and then the point will be inside. So if you want to check whether the point is inside we will have to say something like if this then take whatever action whatever needed for the point if the point is inside or else take action if the point is outside okay. Now, I just want to make one remark okay you will be tempted to write these two conditions jointly in this manner okay, but this is not allowed this is not this is not how you suppose to write these common these 2 condition you have to separate them out and write then in this manner okay.

¹ In fact, there a slight error here. The statement should be $(X \geq 0 \ \&\& \ X \leq 10) \ \&\& \ (Y > 50 \ \&\& \ Y \leq 70)$ and not $(0 \geq X \ \&\& \ X \leq 10) \ \&\& \ (Y > 50 \ \&\& \ Y \leq 70)$

(Refer slide time: 11:46)



The slide is titled "What we discussed" in blue text. It contains two bullet points: "More general ways of specifying the conditions." and "Note: ! Has higher precedence than && which has higher precedence than ||". Below the bullet points is a code example: "!P && Q || R is same as ((!P) && Q) || R". At the bottom of the slide, it says "Next: A somewhat large example based on what we have learned so far." and there is a small circular logo with the text "NPTL" below it.

All right, so what did we discussed? We discussed general ways of specifying conditions and here I should point out that we had these operators “NOT” “AND” and “OR” and the “NOT” operator has higher precedence than “AND” operator which have higher precedence than the “OR” operator so what do I mean by this. So if I have this condition okay now there is some ambiguity over here so the ambiguity is do I mean, should I do P and Q first and then take “NOT” and then take “OR” or should I do it in some other order?

Okay this says that “NOT” has the higher precedence so that means this “NOT” P should be done first. So I have indicated with in this manner by putting parentheses. Then “AND” has higher precedence than “OR” so after that I want “AND” to be evaluated. So that I have indicated in this manner okay and after that sorry “AND” to be evaluated so I have indicated that in this manner and I finally the “OR” should be evaluated. So that happens now. So you can always put parentheses explicitly, but if you do not put parentheses explicitly if you write it in this manner then C++ will use precedence rules and then precedence rules will say look this is what we will, what C++ will execute okay. Now, my preference is not to rely on precedence rules, my preference is just to write the parentheses so that it is very very clear you do not, you do not want to tax anybody’s memory if you are you are yourself reading the program later if you put the parentheses then it is very clear and you do not have to tax your memory.

So in the next segment, we are going to take some what large example what we have, based on what we have learned so far okay full program, but before that we will take a quick break.