Week 2 Module 5 Branching - Part 1

Computing the Absolute Value 1.1

Hi there. Today we're going to talk about the branching statements. Let's start with a problem that we're going to try to solve. So, let's write a program that reads from the user an integer and then prints its absolute value. So, for example, the program can prompt the user to enter an integer, the user can enter let's say negative seven, and then the problem would respond by saying that absolute value of negative seven equals seven.

Let's see how we can implement this behavior. So, it actually depends, in case the user enters a positive integer then the absolute value would remain the same as the input. But in case the user enters a negative number then we need to calculate the additive inverse of that negative number and that would be the absolute value. Mathematically it is kind of easy to calculate the additive inverse of an integer: you just multiply it by negative one so negative seven times negative one that would turn out to be seven, the absolute value.

Let's take a closer look on how we can implement this kind of a behavior. So up to now we have seen some features of a C++. We've talked about a lot of data types and how they are represented; we've talked about several kinds of expressions, such as arithmetic expressions, boolean expressions. In terms of control flow, the only flow we are used to working with is a sequential flow, which is the default by which programs are executed, basically meaning that instructions are executed one after the other in a sequential order.

Let's see if we can use all that in order to implement the absolute value or calculate the absolute value of an integer and then if we take a closer look here it seems that we need some more power in order to do that. Because it seems like that in some cases we want to keep the input as it is in case the input is positive in other cases we want to multiply it by negative one in case the input was negative and a sequential flow, that all instructions are kind of executed one after the other, won't be good enough. Because sometimes you want to do one thing in one and sometimes want to do another things. For that, we will introduce a more complex kind of a control flow and that would be branching flow. That would allow us to execute some instructions on one execution and to execute other set of instructions in another depending on the input or depending on some stuff. Let's show you a few kind of branching statements. The first one would be an IF and let's take a closer look on an IF statement.

Syntax and Semantics 2.1

So, let's take a look at some IF statement. So, actually there are few kinds of IF statements, there would be: a one-way if statement, a two-way if statement, and also a multi-way IF statement. Let's start with a one-way if statement. For each kind of control flow I'll show you two things: first, I'll talk about the syntax, how we can create a legal kind of statement in C++ of that form, and then we'll talk about the semantics, what does it mean or how the compiler or the C.P.U. would execute this kind of statement.

Let's start with the syntax: the syntactic rules of creating such statements. So, we have the program that goes up to the IF keyword. Then, we have the if keyword followed by a condition and closed in parenthesis. Now, it's very important that we put the condition inside parenthesis, the compiler won't allow us to write it differently. It is a very strict rule here. The condition should be a Boolean expression, basically evaluating to True or false.

After we have the IF and the condition, next line comes the body of the IF statement. It is taken one tab to the right. And after that the program just continues. So we have our program before the IF statement then the word if followed by a condition in parenthesis and then pushed one tab to the right we have the body followed by the remaining code of our program. One thing syntactically I want us to note is each expression we know in C++ ends with a semi-colon. So the expressions before the IF statement obviously end with a semi-colon, the expressions after the IF statement obviously end with the semi-colon. Also the IF body, the statement inside the IF, also ends with a semi-colon. But the first line where we have if and the condition in that line we don't have a semi-colon and it's very important that we don't have this semi-colon over there. So that's basically the syntax rules how we can create a legal one-way IF statement.

Let's talk about what happens when we write this kind of a statement: how the compiler is going to understand and interpret this kind of a statement. So the program would be executed up to the IF statement. When we reach the IF statement, the first thing that happens is the condition would be evaluated. Then since the condition is a Boolean expression, there are two cases: in case one case is where the condition evaluates to true, the other is when the condition evaluates to false.

Let's see what happens in each of these cases. So, in the case of the condition is evaluated to true, the body would be executed and then the program just continues. In case the condition evaluates to false, the body would not be executed and the program just continues with its original execution. So, basically the body of the IF here is conditioned to be executed by the value of the Boolean expression that comes in these parenthesis. So, if the condition is true the body is executed, if the condition is not true the body is not executed. So, it is not a sequential flow were each of the expressions are anyway evaluated, the body of the IF is conditioned to be evaluated depending on the value of the Boolean expression.

Computing the Absolute Value 2.2

Now that we have the one way if statement, let's try to implement this problem here that reads from the user an integer and prints its absolute value. Let’s go and implement it.

Computing the Absolute Value Implementation 2.3

Okay, so let's implement this program. Let's first prompt the user to enter an integer so C-out: “please enter an integer.” And let's also break the line here. Then, let's read this integer from the user, so let's have an integer variable. Let's name it, I don't know, maybe “user input,” because this is what the user inputs. So, let's just c-in into this user input variable. And now we need to calculate the absolute value of a user input.

So, as we said there are two cases: one is when the user input is positive, in this case the absolute value just remains whatever the input was, and in case the user input is negative, in this case we need to multiply it by a negative one. So, let's do something like IF and then that's have a Boolean expression here. “User input” is less than zero then; in this case we need to multiply “user input” by negative one. So let's do “user input” times negative one, and let's set it back into the “user input” variable. So, in this case we're kind of multiplying “user input” by negative one and then setting the result back to the variable “user input.”

So, “user input” basically changes its value from its original value to its additive inverse. So this case we have “user input” equals “user input times negative one”. And now we can just C- out, in this case, the user input value. Let's try to execute it. Please enter an integer: so, if we have negative seven, it would a print seven. By the way, it printed the program ended with exit code zero right here after the seven because we didn't break the line. So, let's break the line also, it would make it appear better. And let's also test a positive value, in this case seven and then we get the value seven back.

Okay, so it seems to be working fine. I want to improve it in two ways: the first one is not a great improvement it's just a syntactic shortcut here. Where instead of having user input equals user input times negative one, we can make it shorter and just have the user input times equals negative one. It's a syntactic shortcut of obviating variable value by applying some operation on its previous value, in this case user input would be multiplied by negative one and set back into the same variable, so the times equals operator does that. You can use this shortcut of an operator; there is also plus-equal, minus-equals, div-equals and so on. Sometimes it is easier to write it in this way. So that's one thing I wanted us to use.

The second thing is the output I was expecting out of this program was not just to print the absolute value of the input, to a print something more of the form like the absolute value of whatever original input was equals the absolute value. Not just a seven, but something like the absolute value of seven equals seven For that, we should print something like the absolute value of something and then another pipe here equals and whatever the absolute value is. So this is kind of the format I'm expecting here, like pipe symbol and then the original input and then another pipe and then equals and then the value of the absolute value of the original input. The problem here is that we kind of change, in some cases, the original input. So if you do if it was positive it would remain the same, but if it was negative originally then we changed its value and in this point in our program; we don't have its original value anymore. In order to resolve this we can use two variables: one for the original input and one for the absolute value.

So, in this case let's declare another variable. So we would have user input and, for short, "Abs Value" and then the user input would remain with the original input the user entered and the absolute value would contain the absolute value of the original input. So, to do that I would first initialize absolute value with input value or the user input. And then, in case the user input is negative then I would set the absolute value to be the user input times negative one. If we take a closer look here, we'll see that originally I set absolute value to be the user input. Let's add a semicolon here. And if it is a negative, it would change its sign, if it is not negative it would remain with the original user input. Now at this point, we can use user input, original input, it didn't change, and absolute value down here.

You can see that the names we chose for the variables represent exactly what they're holding. So, the user input actually does hold whatever the user entered that's why it is a good name for it; it was read from the user, input did not change or its value did not change. And then we're printing the absolute value of the user input and absolute value, actually contains the absolute value of this user input; it is better than what I've have done in my previous version, where the user input changed its a value to be the additive inverse of its original value and then it does not contain the user input anymore.

Let's try to execute it now, make sure it works. So let's enter an integer, let's say negative seven and then it just prints the absolute value of negative seven equals seven. I see I'm missing the space here. Let's test it with a positive value. Let's say seven, works fine, let's test it with a zero; though I'm not expecting any issues. Yeah, seems to be working fine.

One-Way if Statements 2.4

So, we have the syntax of a one way IF statement, I want to show you an extension of the syntax. So originally or formally, we can condition the execution of a single expression. You can see that the body here contains a single expression. In case we want to condition the execution of several extensions, we use this syntax here: where we have a compound expression that is basically a set of instructions that are enclosed in curly braces. So, the syntax would go IF parenthesis and the condition in it, and then open the curly brace, we have a set of instructions, again pushed one tab to the right and close curly brace and then the program just continues. So, we kind of group together a set of instructions into a single compound expression using these curly braces syntax and this way we can condition the execution of a set of instructions. A lot of times we use this kind of syntax.

Determining Parity 3.1

Let's try to write another program now. Let's try to write a program that reads from the user a positive integer and determines its parity: whether it's even or odd. So, for example the program can prompt for the user to enter a positive integer, the user would I know say seven, and then the program would respond by saying seven is odd. Let's see how we can, how we can implement this program. So, now that we have if statements it seems much more reasonable we can use the one way if statement in order to implement this program. But it would be better to use a two-way if statement in this case. I'll talk about the if else statement which is a two-way if. We'll implement this program and afterwards with a compare the if else implementation verses of a one-way if implementation of the same program. But let me first introduce you to the two-way if statement, the if else statement.

Syntax and Semantics 3.2

Okay, so in order to use the if else statement, the two-way IF statement, the syntax would go this way; once again, I'll first talk about the syntax, then show you the semantics for how the compiler understands this kind of statements. So, the syntax is very simple it goes like that; we have our program, then comes the if keyword followed by a Boolean expression and enclosed in parenthesis, and then we have the if body, again pushed one tab to the right. After that we have the ELSE keyword and else's body pushed one tab to the right, after the if else statement the program just continues in its original alignment. That's basically the syntax.

The semantics is very straightforward here as well, so when we get to the if else statement, the first thing is the condition is evaluated. As we said the condition is a Boolean expression, so it has two values possible either its value is true or false. In case the condition is true, the if body would be executed and then the program would continue in its original execution. But if the condition would be false then the else's body would be executed followed by the execution of the rest of the program.

So basically based on the value of the Boolean expression, we either evaluate the IF body or the else body. That's why it's called the two way if statement; we either go one way with the IF body or the other way with the Else body, based on the value of the Boolean expression, based on the value of this condition. Once again this is the simplified version of the if else; if we want to use the more compound version, where we want to execute a few of instructions, a set of instructions in the true case, and the set of instructions in the false case, then we need to compound or to group together a set of instructions using curly braces. So, after the if in the condition we can have a set of instructions enclosed in curly braces as after the else we can have a set of instructions enclosed in curly braces.

Determining the Parity 3.3

So, now that we have the if-else statement let's try to implement our program here. Again, we'll read a positive integer and figure out whether it's even or odd. Let's go and implement it.

Determining the Parity Implementation 3.4

So let's first prompt the user to enter a positive integer: let's print "please enter a positive integer." Let's read the input, so let's have an integer user input variable and let's c-in into this user input. And now we need to figure out whether the input is even or odd and create the corresponding output message in order to determine whether a number is even or odd. Basically, we observe the remainder when we divide this number by two; if the remainder is zero it means it's even. if the remainder is one means it means it's odd. So, let's do something like if and then take user input and mod it by two, and figure out whether it's zero.

Basically if user input mod two equals zero, it means the remainder when we divide user input by two is zero that means it is even. So, in this case we need C- out something like user input is even and then else we would need to c- out that user input is odd.
So based on the value of this Boolean expression, once again it is either true or false. So if user input mod two equals zero, it would be true basically we'll print even, but if user input mod two is not zero, otherwise it should be one, then we would just print odd.

Let's go ahead and test it. So let's enter a positive integer here: let's say seven and then it would say that seven is odd. Let's also test it with an even value: let's say sixteen and then it would say sixteen is even.

Sequence of if vs. if-else 3.5

In our previous implementation, when we wanted to determine whether the input was an even or an odd number, we basically read the input and then divided it by two, looked to the remainder and checked whether it's zero or non-zero and printed even or odd respectively. We used the if else statement and it makes a lot of sense because if the remainder equals to zero, it means it's even and if it's not equal to zero, if it's not even, than it is odd. I think I said at the beginning that we can, we don't really have to use the if-else, the two-way if statement we can use of a one way if as well.

So, take a look at this implementation here. So, once again we read the user input and then we kind of check if user input mod two equals zero, we print even after that we check if the user input mod two equals one, we print odd. So, instead of having an if else statement, we have a sequence of two one way if statement; If user input mod two equals zero, print even, if user input mod two equals one print or C- out is odd.

So, first I would say that both of these versions are good, they would both work, but then when we have to choose either to implement the first or the second, I strongly for the first version of the if else. And let me try to say few things why that IF ELSE version is better than the sequence of two if. So in an if else statement, we can assure that exactly one of the bodies would be executed; if the Boolean condition is true the first body would be executed, if the Boolean expression is false then the second body would be executed. So exactly one of the bodies would be executed in an IF-else statement. We cannot say the same when we have a sequence of ifs, now in this case it would work and there is not any issue in exactly one of the bodies would be executed too, but formally or generally if we have a sequence of two IF than maybe none of the Boolean expressions would be true. In that case, nobody would be executed zero bodies would be executed. If both Boolean expressions in a sequence of IF are true then both bodies would be executed, so two bodies would be executed. And obviously if one of the Boolean expressions is true then a single body would be executed. So if we have a sequence of ifs, there could be cases where not exactly one body would be executed. That's not the same in an IF-else statement. Now if we want to express a computation, a process where we want to assure that one thing would happen, an if-else statement would be a better choice. That's why I would choose an if-else statement here in order to implement this program.

Boolean Interpretation 3.6

Before we move on to another control flow, I want to give you another note regarding the syntax of an IF statement in C ++. Let's take a look at this few lines of code here. So we have our main, we have an integer variable named val assigned or initialized to zero and then we check if val equals zero we say that val is zero, otherwise we say that val is not zero. Obviously, when we execute this code I would expect it to print that value zero, because val was initialized to zero and then it would say that value zero. But you can see that we didn't use the relation operator of double equals in this case value equals equals zero so.

I'm kind of wondering what would happen whether it would be a compilation error, using an expression that is formally an arithmetic expression, it's not a Boolean expression its value is not true or false, it's not a double equal operator here, it's a single equals sign so it's basically an assignment. So in case we put an arithmetic expression inside our parenthesis, in the place of a Boolean expression in the place of our condition, I'm kind of wondering what would happen whether; first the compiler would say that this expression or this if expression is illegal, or maybe it would somehow understand and interpret this kind of an expression and would create some kind of a behavior that, I don't know, let's try to think what would happen.

So, the first question is whether to try and figure out whether this thing is valid or invalid, is it legal or illegal in C++. So surprisingly actually I would say that this expression here this if Val equals zero do one thing else do another thing is valid in C++. And it's kind of surprising because when I defined the syntax of an if else statement, we said that in the parenthesis should come a Boolean expression and once again Val equals zero is not a Boolean expression. So, it is surprising that is a valid expression in C++ it is valid. In other programming languages the equivalent syntax would not be considered valid, for example in Java I think it's not valid in other programming languages, it's also more valid but in C++ it is valid.

And now let's try to guess what would be the behavior in this case. So, obviously it's not checking whether val equals zero it assigns val with the value of zero, but then we can wonder whether this assignment expression is considered to be true or false whether we would execute the IF body or would execute the ELSE body. So, if you recall what we talked about the bool data type we said that false is represented by a zero and true is represented by values that are non-zero. In this case, val equals zero is an expression that first assigns zero to val but its value is the value that was assigned in this case zero. So Val equals zero is evaluated to zero, to the integer zero, but the value is zero.

The C++ compiler gives a Boolean interpretation even to expressions that are not Boolean expressions even two arithmetic expressions and the logic is basically the same. If the value is zero it would be considered as false, if the value is non-zero then the Boolean interpretation would be true. In this case, since Val equals zero then the Boolean interpretation of this expression is false. Therefore, the else body would be executed and we would print that val is not zero. Before we end this note I just want to advise you not to use the syntax I showed you it because sometimes we make mistakes and we want to understand the behavior of our program but obviously, it is not a good programming style. In an IF statement we want to have the condition to be a Boolean expression and not an arithmetic expression that is interpreted to a Boolean value.