Week 2 Module 5 Branching – Part 2

Classifying a character 1.1

Let's use all of these statements in order to implement the following problem; let's write a program that reads from the user a character and classifies it to one of the following: either lower case letter, or an uppercase letter, a digit, or a character that is not alphanumeric So, in the execution of the program could behave something like that first the user's prompted to enter a character, the user then says, I don't know says upper-case D. and then the program would respond by saying uppercase D is an uppercase letter. Let's try to implement.

Classifying a character Implementation 1.2

Okay. So, let's first prompt the user to enter a character, so C-out: please enter a character, end-l and then let's read this character so let's have char variable, User-C-H think would be good name here. So User-C-H is the input the user entered. And now, we should figure out whether this character is an uppercase letter, a lower case letter, a digit, or a character that is not alphanumeric. In order to do that, first let's decide in what control flow we're going to use. It seems as if we have to choose one out of four cases here. So a multi-way if seems very reasonable so if else if else something like that should work, so I would definitely use a multi-way if here. So, that's one thing we needed to figure out, the other thing is the how to determine whether our user C.H. or input is an upper case or lower case or a digit.

So if you recall characters are represented using ASCII values; let's take a second look here at our ASCII table. If you remember we said that all the lower case letters are continuous in this table, all the upper case letters are continuous and even all the digits are continuous. We know that we don't need to remember or to memorize the values of the lower case letters in the upper case in the digits. The only thing we need to know is that they are continuous and we can check whether our ASCII value is in the range of the lower case A and lowercase Z, or in the range of the Upper Case A to the upper case Z, or in the range of zero and nine and so on. Let's try to do that so. Let's do something like if and now let's try to check if our user C.H. is in the range of the lower case letters.

So, let's do something like user C-H. is greater or equal to, and now I don't want to use the value of lowercase A I don't want to use ninety seven in this case. I can use the C++ literal for lower case A and that would be something like that. So if user C H. is greater or equal to lowercase A and user C H is less or equal to lowercase C that means that our ASCII value is in the range of the lower case letters. In this case, let's print that the user CH is a lower case letter and let's do the same or something similar, to the upper cases. So else and then I won't break the line here I would continue that if in the same line because I want to use the multi-way if and once again I'll check if user CH is greater or equal to upper case A and user CH is less or equal to upper case Z, then let's c- out that our user CH is an uppercase letter. And now let's also check if it is a digit. So, else if our user CH is greater or equal to zero and the user CH is less or equal to nine then we want to C-out that our user CH value is a digit. And, eventually if it is not either one of these then we should just print that this character is not an alphanumeric character.

Okay, so let's try to test it out. Let's first enter a character, so let's do lowercase D. seems to be working fine, these lowercase letters. Let's try uppercase D also seems to be working fine let's try, I don't know 4, it says four is not alphanumeric character. I was expecting 4 to fall somewhere here and print that four is a digit. Try to think what's wrong here. So here we checked whether user CH is greater equal to zero or and less or equal to nine, User CH is basically the ASCII value of our character but then the ASCII values zero is not zero is forty eight and the ASCII value of nine is not nine, it’s fifty seven.

So in all of these cases here up here this case here in this case here, we compared a character to a character, once again a character to a character. And it basically compared the ASCII value that is represented in here to the ASCII value of a lowercase A and so on here we are comparing a character to an integer. So that's fine with the compiler it would just implicitly cast this character into an integer and it would compare the ASCII value of this character to the integer value zero and the integer value for nine. So, in order for it to work we need to give here the ASCII value of zero and the ASCII value of nine, basically forty eight and fifty something. Again we say we don't want to memorize these values, so I can just use the character literal for zero and the character literal for nine and it would look something like that. Once again I want you to know that the integer zero and the character zero are two different things. The integer zero is a 4 byte 2s compliment representing the value of zero, where the character zero is a single byte representing the digits or the character zero that is represented by the ASCII value. In this case zero is forty eight so the binary representation of forty eight would be stored in this case. So let's try to execute it now. First let's see that this change didn't affect all other cases of lower case works and uppercase also works and now let's see what happens with digit four. Also works. And finally let's check something that is not alphanumeric, let's do the dollar sign, and that also seems to be fine. Okay.

Convert 24-hour to 12-hour 1.3

Let's try to write a program now that reads from the user a time, entered in a twenty four hour format and then the program would print the equivalent time in a twelve hour format. So for example the user would be prompted to enter a time in twenty four hour form, that it would they would say I don't know fifteen colon thirty seven, and for that input the program would respond by saying that fifteen thirty seven is three thirty seven PM. So, basically we're converting time in a twenty-four hour format to twelve-hour format adding the AM PM and stuff like that. I'm sure you're familiar with these two formats, but let's take a second look here how these formats basically work.

So in a twenty four hour format, we have all the hours going from running from zero to twenty three. The first twelve are considered to be A.M., so zero to eleven are considered to be AM. Where twelve to twenty three are obviously considered to be PM, so that's the period value. Regarding the twelve hour format an hour or so in the first half of the day the time basically remains, so one in the twenty four hour format is one AM, two is two am, three is three am, and so on up to eleven which is eleven am. Where in the second half of the day then the time doesn't really stay the same, we kind of shift it twelve hours back so thirteen is considered to be one pm, fourteen is considered to be fourteen minus twelve which is two pm, fifteen is three pm and so on up to twenty three which is eleven pm.

So it seems that the first half of the day the hour basically stays the same and the second half of the day the hour is shifted back twelve hours, twelve numbers before. That's almost true, there are two exceptions here, in the first half of the day, zero is not zero A.M. So, zero doesn't stay the same just as one stayed one A.M. and two stayed two AM. Zero is twelve A.M. So it has different behavior here, same thing regarding twelve or noon, twelve doesn't become zero P.M. is not shifted back twelve hours. Twelve kind of remains twelve P.M., so most of the hours in the first half of the day remain the same besides zero. Most of the hours in the second half of the day are shifted back twelve hours, besides twelve P.M. So, this is basically how the twenty four hour and twelve hour format works. Let's go ahead and implement this program now.

Convert 24-hour to 12 hour Implementation 1.4

So let's start by prompting the user to enter a time in a twenty four hour format so C-out: please enter your time in a twenty four hour format. Let's break the line here, and now let's read the time so the user would print something like fifteen colon thirty seven. So we would need an integer to read the fifteen an integer to read the thirty seven and then we would also need to get rid of the colons here. We're not going to use

it but the user would definitely print it. So we would need two integers. So let's say integer: the fifteen would be an hour in a twenty four hour format so I'll name it hour twenty four, the thirty seven would be minutes so I'll name it minutes twenty four and we'll also need a character, so let's define character; I'll just name it temp because it's not going to be used besides just to be read. Let's C-in this into hour twenty four into temp and into minutes twenty four. So, we have our input hour twenty four and minutes twenty four and now we're basically supposed to convert it into a twelve hour format.

For that we would need the initialize actually three parameters: we would need the new hour, the new minutes, which would basically be the same minutes, but we would also need the AM PM. So, for the hour and the minutes we would have, let's say hour twelve and minutes twelve. For the AM PM, we would store it in a string variable, for that I would include the string library and then I can create a string variable; I'll name it period, so we have the period. So we have basically all the variables to hold the result, the hour twelve the minute twelve and the period. And now we should decide in what kind of a control flow we're going to use to set all these variables.

Let's take another look here at the conversion table. And I'm kind of wondering whether we're going to use multi-way IF, a two-way if, and so on. At first it seems like we have, maybe four different cases the case of zero, the case of one to eleven, the case of twelve, and the case of thirteen to twenty three so maybe a multi-way if would be a good choice here. But if you recall how we kind of thought of it when we introduced this conversion table. We first kind of determined whether it is am and pm so it's kind of a two way choice here, and then for each one of these we thought if we were in this range or in the zero if we're in this range or at the twelve. So it seems more logical to first determine one out of two cases, the AM PM, and then inside it nested in it, to determine what would be the twelve hour format for the hour. So my choice here would be a two way if instead of a multi-way or four if else statement but then a multi-way if would also work and be fine. But I would implement it using two-way IF statement. So for that I would do something like if and then I would ask if the hour twenty four is in the range of greater or equal to zero and hour twenty four is less or equal to eleven so if we are in the first half of the day this is what we're going to do.

It's going to be a lot of stuff so I'm creating a compound expression. Otherwise, we'll do some other stuff. So this would be the basic structure of our program; one thing is that the minute in the twelve hour format is basically the same as the minutes in the twenty four hour format so before even testing and determining in which part of the day we are, I can set the minutes twelve to be the same as many twenty four so that goes anyways. If we're in this part of the day let's set period to be AM, if we're in the second half of the day let's set period to be PM. So, basically up to now we said something like, we read the time in a twenty four hour format and we said the minutes in the twelve hour format to be the same as the minutes we'd just read from the user and then we checked in which half of the day we are setting the period variable accordingly.

Now we just have to set the hour twenty four based on which part of the day we are so let's do something in order to determine what the hour twenty four value is. So let's take a second look here. So in the first half of the day the hour twenty four is the same the hour twelve is the same as the hour twenty four besides the case of hour twenty four equals zero. So let's ask if hour twenty four equals zero then hour twelve is set to twelve, otherwise our hour twelve is the same as hour twenty four right. So in the first half of the day only the case where hour twenty four is zero then hour twelve is twelve, otherwise it's the same as hour twenty four in the second half of the day.
Once again, second of all the day basically hour twelve is shifted back twelve hours before the hour twenty four value. So let's one again once again check if hour twenty four equals twelve then our hour twelve would be set to twelve, otherwise our hour twelve would be hour twenty four minus twelve or shifting it back twelve.

So it seems like we're setting minutes twelve anyway before the IF statement the, if statement determines whether period would be set to AM or PM. And that if the main if statement also determines whether we're going to set our twenty four using this logic or using that logic. Each one of these logics decides whether to set hour twelve one way or another. After we're done doing all of that we are on the ready to create the output message. So we want to say something that I don't know fifteen thirty seven is three thirty seven P.M. So hour twenty four colon minutes twenty four is and then hour twelve colon minutes twelve and let's break the line. Okay, let's try to execute it. So please enter a time in a twenty four hour, let's do fifteen thirty seven, and then it says fifteen thirty seven is three thirty seven.

We forgot to print the period value. So that after printing minutes twelve let's also add space and then print period which would be AM or PM. Let's test it now. So fifteen thirty seven, now it says that fifteen thirty seven is three thirty seven P.M. Let's try to do all four cases. Let's do twelve thirty seven twelve, thirty seven is twelve thirty seven P.M. That's true.

So we've tested a value in this range, we've tested a value here in twelve, now let's do two AMs; one in one to eleven and one with the zero, basically testing all the branches, all the possible branches of this program. So let's do zero thirty seven, it would say that zero thirty seven is twelve thirty seven am. Three thirty seven and then it would say that three thirty seven is three thirty seven AM. Okay, so it seems that all four branches work properly and yes, that's it.

Switch Statements 2.1

We have a few branching statements: we have a one way if, a two way if, multi-way if. I want to show you another kind of a branching statement; it's also a multi-way branching. It's called a switch statement and it goes something like that. Again, let me first talk about the syntax the rules of how to create a valid switch expression, and then let's see how the compiler basically interprets and executes this kind of statements, what's the semantics of this expression.
So let's start with the syntax we have our program up to the switch statement, then we have the switch keyword and then we have a numeric expression basically an arithmetic expression enclosed in parenthesis. Then, we have a compound expression basically a body enclosed in the curly braces.
That has a few case clauses and the default clause; each case clause starts with the keyword case followed by a constant value and colons, and then shifted another tab to the right we have the statements that are condition followed by a break and the semi-colon ending this case. And then another case, another constant colon, body break, and so on. We can have as many cases as we want and then finally would come the word default and that again some statements pushed one tab to the right, with the body and the break.

You can see that here we are kind of conditioning execution of some statements but we're not grouping them in the curly braces, we start their group with colons and a group with a break. This is how we kind of specify the instructions that are conditions. So this is the syntax, the semantics goes something like that. When the execution reaches a switch statement, first the numeric expression is evaluated. A numeric expression is not a Boolean expression; its value is not true or false, it's a numeric value to be seven six point five or whatever. So we have a numeric value, and then this value is compared to the constants that come in the different cases. So first the numeric expression value is compared to constant one, if they're equal if the numeric expression is the value of constant one, then these expressions would be executed. The break would break out of the switch statement and the problem would continue. But if the numeric expression is not equal to constant one, the numeric expression is compared to constant two if it is equal to constant two this statements of the body would be executed break would break us out of the switch statement and so on. So the numeric expression would be compared to the different constants one after the other. If none of the constants are equal to the numeric expression, then the default body would be executed break would break us out of the switch statement. So you can see that we have a muti-choice

here depending on the value of the numeric expression and the values of the constants. One of them would be executed; either the constant one body, or the constant two body, or maybe the default body would be excellent. So it's a multi choice branching statement that we get here out of the switch.

Computing Value of a Simple Expression 2.2

So let's try to use a switch in order to solve of the following problem. Let's write a program that reads from the user a simple mathematical expression. We will allow only an addition difference division or multiplication expression and then we'll print the value. For example, the user would be prompted to enter an expression of the form argument operator argument, and then the user would say I don't know five point two times four and the program would respond by saying twenty point eight which is five point two times four. Let's try to implement this program using switch. Now we don't really need to use switch here we can definitely use the if-else statement or other kind of branching statements, but let's try to use switch anyway. Let's go ahead.

Computing Value of a Simple Expression Implementation 2.3

So let's implement this program first let's prompt the user to enter an expression, so please enter an expression of the form of Arg one, arg 2. Let's break the line and now let's read this expression. So it can be five point two times four whatever. So the format, basically, implies that we need two double variables, arg1 and arg2 and we would also need a comma here obviously. We would also need a character for the operator so let's have a char variable OP and now that we have all of these variables we can c-in arg 1, OP and arg 2. So if the user enters five point two times four ARG one would be five point two, OP would be the star symbol, and ARG two would be four.

Now, we want to calculate the value of this expression. So, basically we want to figure out or depending on the value of the operator, we want to apply the corresponding operator. So we need some sort of a multi- branching statement we said we're going to use a switch for that. So we're switching over the value of the OP variable, so switch OP and then we would have a few cases. So case it is plus. We would do one thing. Case and, just one second we need to take it one tab here, and then case it is minus we would do another thing. And case it is multiplication we would do some other stuff and case it is the dividing, we would do another thing. Default, we would, I don't know let's figure it out later.

Okay, so let's start with the case; the user entered Plus as an operator in this case. Let's have another double variable result, res for short. So let's set res to be ARG 1 plus ARG 2. So in case we get a plus we add these values and then let's print res. Same thing for subtracting and multiplying, so let's do for subtracting. So in this case we want to subtract ARG 1 and ARG 2. In this case we want to multiply ARG one and ARG two. And in this case we would want to divide ARG one and ARG two. Default if it's not one of these operators, we would just want to say that it's an illegal expression so let's write illegal expression and break.

Okay, so let's take a closer look here at what we have. So we're switching over the operator: in case it's plus we're adding, in case it's minus we're subtracting, in case it's a star we're multiplying, in case of the slash we're dividing. Each time we're printing the result, and the body starts here and ends here for each one of these branches. Let's try to execute it to see that we don't have any unpleasant surprises here.

So please enter an expression five point two times four, and we would get twenty point eight. Maybe just one thing I would add here is, maybe this kind of thing. Let's say I want to add five and zero, that would be fine. But if I want to divide five by zero, I don't want to get this; it probably stands for infinite in some other compilers it would create at runtime error. I want to be more cautious here when I divide the ARG one by ARG two, and maybe ask if ARG two is not zero then we want to do all of that, want to divide and print this value.

Otherwise, let's just, print that it’s an illegal expression again. Once again let's take care of the indentation and in the case of dividing, we check if it is legal to divide or not and break after that but see that now five divided by zero would say illegal expressions so five divided by zero is an illegal expression.

Switch Statement – Syntactic Notes 2.4

Before we end this module, I want to say a few additional syntactic notes regarding switch statements. Switch statements are less powerful than the if else if else statement. Everything we can do with the switch statement, we can do with the if else if else statement but not the other way around. There are stuff that we can implement using the multi- way if that are not possible to implement using switch statements. Basically that's because the if else if, or the condition in an IF statement can be a very complex Boolean expression, where in the switch statement we're only comparing a numeric value to some constant so we can only compare equivalence of some values not more than that. But then there are cases when we do want to use switch statements, basically when we want to implement menus. When we can, when you'll get some more experience in programming, you'll see you can get the hang of whether to choose to use a switch statement over multi- way if. But before we do that, let me say some additional notes syntactic notes regarding this switch statement.

The first thing I want to say is that the numeric expression, that comes after the switch keyword, must be of one of these three types: either an int, a char, or a bool. The numeric expression cannot be, let's say, a float or a double, cannot be more complex types that are built in or can be defined later on by programmers, it can only be one of either an int, a char, or a bool. That's a restriction we will have to live with but it's good to know that we have to create a numeric expression of one of these types.

The next thing we want to make explicit here, is that the case labels, these constants with we set case constant one, case constant two, and so on must be constants. It's not a coincidence that we named it constant one constant two and so on there are really constants. So basically means there are either C++ literals of type int, char, or bool or named constants basically constants, we as programmer defines constant X. equals whatever. But if they cannot be expressions, four plus five, we can do case four plus five. It cannot be using variable names, we cannot define int N and then say case N. So the case labels must be constants; that's another restriction you must follow when you use switch statements.

The next thing I want to note here is, when we reach a switch statement we first evaluate the numeric expression and then we start comparing it to the different constants in the case clauses. So if no case label matches the value of the numeric expression, the control branches to the default label and these expressions are basically executed, we've already said that. But what I want to add now is that the default is not, doesn't really have to appear there, if there is no different branch then nothing basically is executed inside the switch statement and the program just continues to move on. So in case the numeric expression doesn't match either one of the constants in the case branches and there is no default branch, then in this case the program just continues on.

The last thing I want to say is that the break keyword also is not mandatory. We don't really need to have a break statement. If we have, let's say case constant one and then some expression we said that the break basically says this is where the execution should stop and breaks out of the switch statement. But if there is no break statement after these few statements then the control basically falls through the next case, even though the constant doesn't match the expressions would be executed till we reach the next break statement. Again try to avoid needing break statements, but just in case you by mistake do that that is how the execution is going to behave.