Week 2 Module 4 Data Types and Expressions Part 4

The bool Data Type 1.2

Ok so in terms of data types we talked about integers float and doubles character string. The last data type I want to review here is called a bool data type. Bool is short for Boolean. The data it represents is truth value true or false. Actually these are the only two values that can be represented in a bool data type. And if we try to think how this bool would be represented internally actually it seems like a single bit is enough to represent all the range of bool values. Either zero let's say for false or one for true. But then in C++ all the datas come in groups of bytes so even though we don't need 256 different values a bool data would be one byte long. In that one byte a sequence of eight zeroes would be the representation of false. The representation of true is a bit surprising at least for me it was when I first heard of it. Any non-zero value is considered to be true. So actually there is one way to say false all zeroes and all the rest of the ways 255 other ways to say true to represent true. That's how Booleans are represented in the memory. In terms of C++ built in literals there are these two literals there is true or false all lower case. There are C++ key words that are considered to be of type bool. You can use them wherever you need to. In terms of operators in this case the bool data type is not an arithmetic type. It doesn't have a numerical value it is a Boolean type and it has a Boolean value. The operators that are common for bool are called Boolean operators. There are three Boolean operators the not and and or and I will talk about each of them separately. Let's start with the not operator. The not operator is a unary operator basically it has a single operand a single argument p for example. And then you have an expression not p. The meaning or the semantics of the not operator actually matches the meaning we have for the word not in English. So for example if p is true not p would be false. Kind of obvious. If p is false not p would be true. So it basically negates the value of its operand. In C++ the syntax for not is not the word n o t not the sequence of letters n o t. We have exclamation mark for that. For example let's say we have three Boolean values b1 b2 and b3. We can set b1 to the C++ literal true. We can set b2 to be not b1. That would set b2 to be false obviously. We can also set b3 to be not of false that would set b3 to be true. So that's the not operator. Another operator used for Booleans is the end operator. And and is a binary operator meaning it has two operands p and q. And it is defined actually very naturally as we use the word and in English. So in order for p and q to be true both p and q must be true. So if p is true and q is true p and q is true otherwise p and q would be false. If p is true and q is false p and q is not true because not both of them are true. If p is false and q is true it is also considered to be false and obviously if both of them are both p and q is not true. P and q is false in this case. The syntax for an and in C++ is a double ampersand symbols that come together that's the way we write and in C++. And once again if we have three variables b1 b2 and b3 we can see b1 to true we can set b2 to false and then we can have b3 to be b1 and b2. In this case true and false is false. So b3 would be false. We can also set b3 to be b1 and not b2. If b2 is false not b2 is true and then b1 which is true and not b2 which is also true would together with the and operator would be true. In this case b3 would then be true. So we have the not and and operator. The last Boolean operator I want to talk about is the or operator. The or operator is a bit more tricky because in English there are two meaning we have when we use the word or. There is what we call the exclusive or or the inclusive or. Let me demonstrate it. For example when you go to a restaurant and the waiter says with your meal you can have soup or salad. So you can choose either soup or salad. Obviously if you choose soup that's fine if you choose salad that's also fine but it won't be fine to say I want both soup and salad. So an or is true if exactly one of soup or salad is chosen. You can't choose both of them. So exactly one that's the exclusive or. Another meaning for or is demonstrated in the following example. In order to pass the course you need to get A in the midterm or a B in the exam. So if you get an A in the midterm you obviously pass if you get a B in the exam you also obviously pass. If you get both an A on the midterm and a B in the exam obviously you passed. So in this meaning of or if both arguments are true that is also considered to be true. Not like choosing soup and

salad which is not an option. So the way computers or in C++ or in OS define is the inclusive or where like the second example I gave you. So in order for p or q to be true at least one of them has to be true. So if both are true p or q is considered to be true if p is true and q is false it is true. If p is false and q is true it is also true. The only case where p or q is false is where both of them are false. So this is the definition of the or operator we can then use it in C++. The syntax is double pipes for or and it is used something like that. If you have three Boolean variables b1 b2 and b3. If b1 is false and b2 is b1 or not b1 b1 is false not b1 is true false or true is true. So b2 would be true. If we have let's say a more compound expression b2 and b1 or true. B2 as we said is true and b1 which is false or true that would also be true. So true and true is true. So b3 is also true in this case. This is the main syntax of the bool operator. We will use Booleans in a lot of our further programming in the future module you will see how useful this type. Now it seems very standard but then it would be used in order to control the flow of our executions later on.

Boolean Expressions 2.1

Ok so we've talked about a few data types the int float double char string and bool. Now let's talk a bit more about types of expressions. So we've seen arithmetic expressions that are created using the arithmetic types. When we speak about arithmetic expressions we say that arithmetic expressions have numeric values. So when we add integers when we multiply floats when we add an integer to a char they all result with a numeric value. Then they are considered to be arithmetic expressions. When we introduced the bool data type we started using Boolean operators and using Boolean operators we created some form of what we say Boolean expressions. Now Boolean expressions as opposed to arithmetic expressions don't have numeric values they have Boolean values. So arithmetic expressions gave numeric values where Boolean expressions have Boolean values such as true and false. Let's get into some more details about Boolean expressions. So there are atomic Boolean expressions and using atomic Boolean expressions we can create more complex more compound Boolean expressions. For atomic Boolean expressions we've talked about bool C++ literals true and false. Each of them is considered to be an atomic Boolean expressions. And then using atomic Boolean expressions and the atomic Boolean operators not and and or we can create a more compound expressions. For example if we have a Boolean b we can set b to true. True is an atomic expression but then if we set b to true and not b that is a compound expression. Not b is a compound expression true is a Boolean expression and when we combine Boolean expressions with Boolean operator that creates compound expressions. So the second expression here true and not be is considered to be a compound expression. In C++ there is another way to create an atomic Boolean expression in addition to the bool literals. And that is comparing arithmetic expression using relational operators less than greater than less or equal greater or equal something like that. So if we have an integer x assigned to a 3 we can do something like b equals x is less than 5. The expression x is less than 5 is a Boolean expression right. Its value is either true or false and this case since x is 3 x less than 5 is true. So b gets the value true. In this case x less than 5 is a Boolean expression its value is true in this case and it is considered to be an atomic Boolean expression. We can then use the Boolean operators to combine atomic expressions. For example x greater or equal to 0 and x is less than 5. Each of these two operands is considered to be an atomic Boolean expression and in this case x greater or equal to 0 is true because 3 is greater or equal to 0. X less than 5 is also true because 3 is less than 5 and we combine them using the and operator true and true that is also true. So b gets the value of true. The expression x greater or equal to 0 and x is less than 5 is a compound Boolean expression. One last thing I want to note here is that in C++ we cannot change chain sorry relational operator. We can't do x greater or equal to 0 less than 5 in a single statement. We need to separate them and use the and or or operators in order to combine them. So we've talked about less than greater than less or equal greater or equal relational operators. Two additional

relational operators are equal to and not equal to. Surprisingly or actually not really surprisingly we can't use the equal sign in order to check if two values are equal to one another because the equal sign is already used for an assignment expressions. The C++ syntax for checking equality is double equals. So in order to check if two values are equal to one another we say if x equal equal to a value we want to check the equality of. In order to check if two values are not equal to one another we use exclamation mark equal. These two together are not equal. For example we can have b equal to x equals double equal actually to 3 or x double equal to 4. In this case since x is 3 x double equals to 3 is true or x double equals to 4 is false but true or false is true. So b would be true.