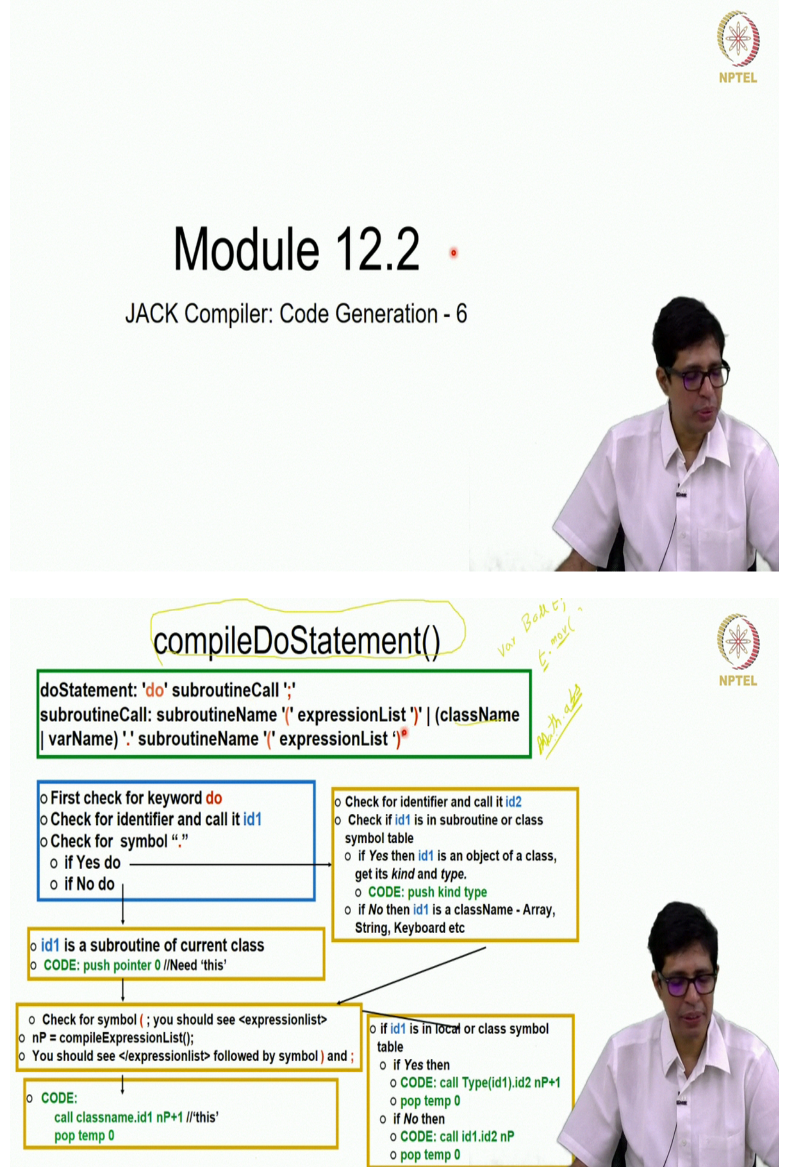
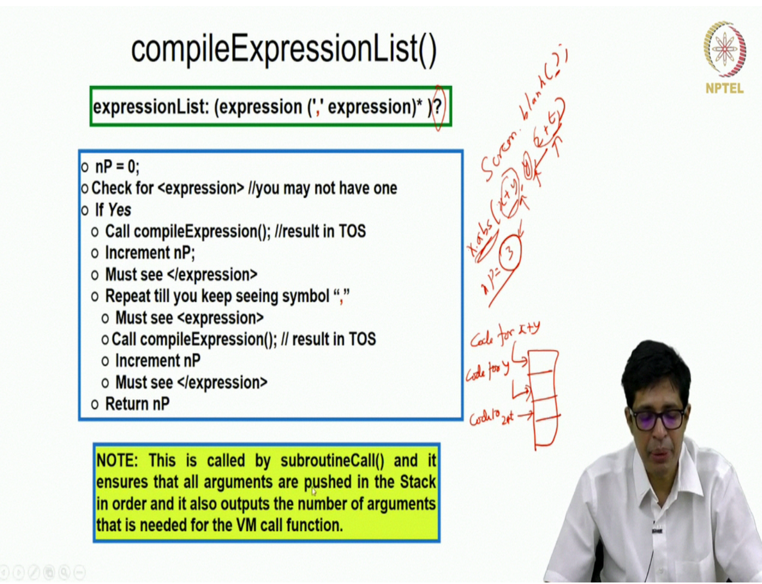
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module: 12.2
Jack Compiler: Code Generation - 6

(Refer Slide Time: 0:18)





Welcome to module 12.2, in this module we will look at compile expression list as we saw in the module 12.1, there was an expression list as a part of your compile do statement, so how the expression list look like, it will be expression followed by there will be no, nothing in this like I could have subroutines which basically says, you know, it is something like screen.blank, so there is nothing here, so that is how we got this question mark here, nothing could be that as a expression, but I could have say some abs of you know x.abs of X+Y, Y, Z+T, each one of these are expressions and this compile expression list need to generate code for evaluating this expressions and this order and then also output the value in this case 3 because there are 3 expressions.

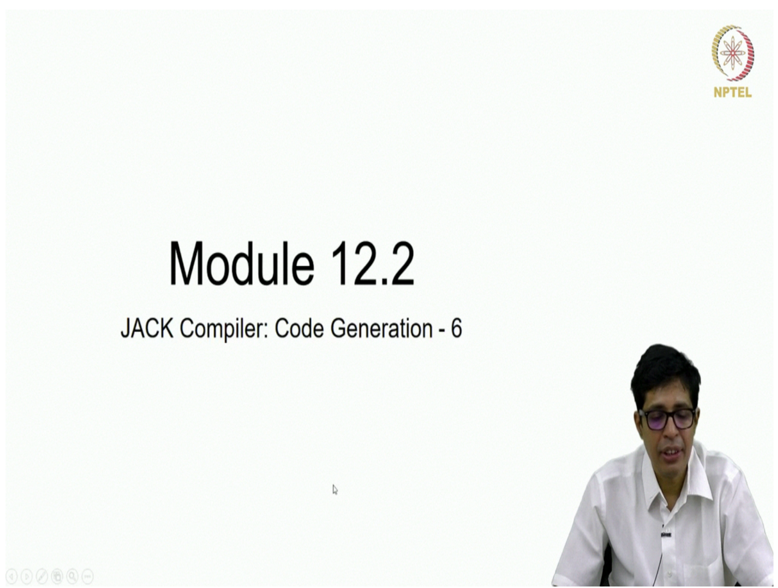
So this is a number of expressions here, so nP will be 3 at the end of the and this compile expression list will evaluate code for X+Y code for Y, code for Z+T, so that will be code for X+Y which will put on top of the stack the result of this, then there will be code for say Y which will put the result of this in the top of the stack after this sorry, after this and this will be the result of X+Y, the result of Y and then there will be code for Z+T which will put on top of the stack for us. Okay, Z+T right.

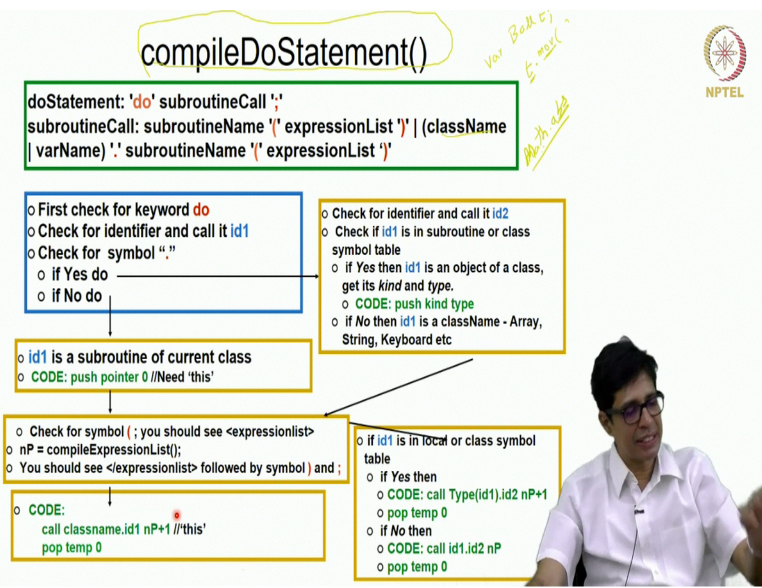
So at the end of this all the arguments that are necessary for say is extra .abs will be there on the stack one after one, so that is the role of compile expression list and in addition it will give you the value 3, it will output an integer which is 3, so this is how compile expression list is formed, so how do you do nP the number of parameters to start with 0, I check for at least one expression should be there, if no expressions is there, then you go out, if there is one expression you may not even have this as I told you that can be question mark, you check for this expression, if yes there is an expression call compile expression.

So if I have one expression say X+Y call compile X+Y, the regional of that will be on the top of stack, now incremented nP because I have seen one expression, so nP, you may see/expression, now as long as I going to see a, once you see, then again, you must see expression call compile expression that reason will be on top of stack, incremented nP again, you must see/expression again, you go and check if you are getting a, and keep on repeating this, that is what is this, expression star means 0 or many repetitions of this and after that you return nP, that is all.

So this does not dump any code at all, there is no green line here right, so green means it is dumping a code, but this will basically return a number of parameters, so what happens in this case x.abs when you do an expression list this will dump the code for X+Y which will ensure that the result of X+Y is on the top of the stack, it will dump the code for Y which will receive, which will assure you that the top of the stack will have Y now, then this will also generate the code for Z+T and it will ensure that the top of the stack has Z+T at end of this, so all the 3 arguments that are necessary for x.abs will be on top of the stack right, so that is what we have mentioned here.

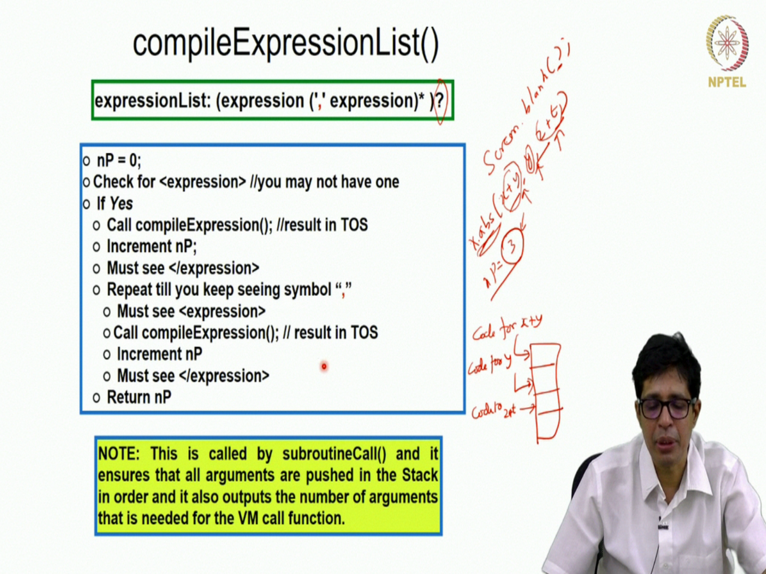
(Refer Slide Time: 4:16)

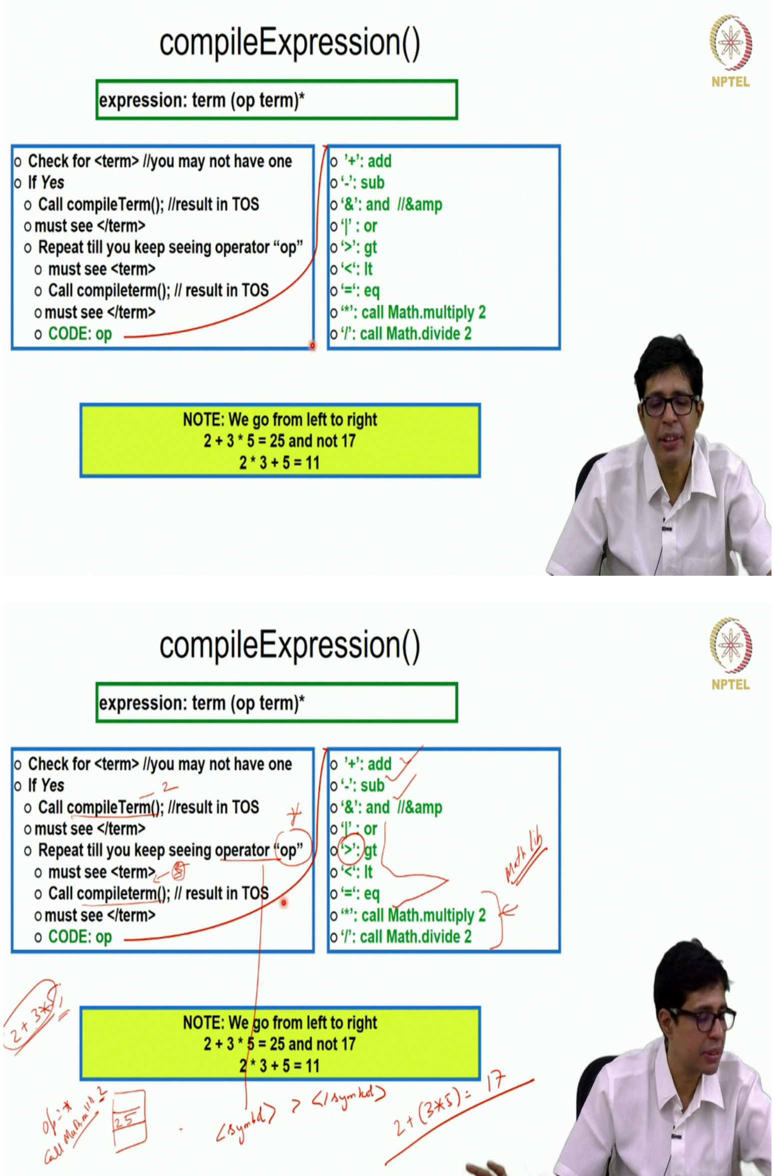




So in the compile do statement that we saw in the previous module, so I have subroutines name or class name, var name.subroutine name of expression list, so when I say compile expression list here as you see here, when I say compile expression list here. This will ensure that all the argument for this or there are already in the stack, then after this only your calling, make a call here you go here and make a call here, so before you actually call the subroutine as per our earlier understanding all the arguments should be in the top of the stack, it should be on the stack in that order and this compile expression list essentially ensures that as a part of this.

(Refer Slide Time: 5:06)





Right, so this is about compile expression list, which neatly works out like this, now next one is compile expression, so in compile expression basically we are looking at you know, what you saw as left to right presidents, for example 2+3 into 5 is 25 first we add 2 and 3 any multiplied by 5, 25 but in traditional terms star has a higher prisoners presidents then plus, so now 2+3 into 5 is normally 17 but in this case, since I left or right presidents, so this will be 25.

Similarly 2 into 3 +5 will be 11 okay, so first 2 and 3 gets multiplied and then +5 is 11 right, so this is how, so if I want actually 17 then basically I can go and say 2+3 into 5 then this will give certainly 17 okay, so this is the thing, so basically we go from left to right, so when I see are term followed by an op terms star, so nothing could be there or it can be term op term or it can be term op term op term like that, so this can be one or more, 0 or more repetition of op

term after this term, so how does, so this is basically how does compile expression works, first it will check for term if yes, it will call compile term, the result of this will be it will generate, compile term will generate a code, which will be the result of that will be on the top of the stack, at to that it must see/term and depicted you keep seeing operator op.

So what will operator op, it can be one of these 9 things plus, minus, you not see ample sign there, you will see & amp because in XML we want, if you want to display & I should & amp, then or, greater than, less than, equal to, star,/, so whenever, so we will see symbol plus/, symbol minus/, symbol & amp/symbol will see that, so as longer as you are going to see of these ops, so what you mean by seeing these ops to repeat this, we will be seeing this as say symbol, say greater than/symbol which will be this. Okay, so like that. Okay.

So as long as you are keeping this, once you see an op you must see are term again you call a compile term the result of this will be in top of the stack, again you will see/term after this, again you go and after you see/term immediately you have to put this code op, what is code op? If you see a plus, if you had seen this op as plus than you should put add, then sub or & amp or greater than like this. Okay.

Suppose I have 2+3 into 5 what this will do? First it will compile term, this compile term will do, it will first see 2, so 2 it will make 2 as top of the stack, then this will see a plus, then it will see term 3 so it will put 3, so this first it compile term will realise 2, the second compile term will realise 3, meanwhile you do have to seen an plus here, so this plus would be add okay.

So I will have, no, I will have add, now what will add do in practice, add will add this 2 and 3 and 5 okay, so the moment I add this to will go and I will have 5 here, now again you will come back and see, so this has seen 3 again you will come back and you will see a star here, so you are op will now become star, what is star? Star is call math.multiply 2, so the op is now star right, then this compile term will now see 5, it has seen 3 now star then this compile term will see 5, it will push 5 on the stack now.

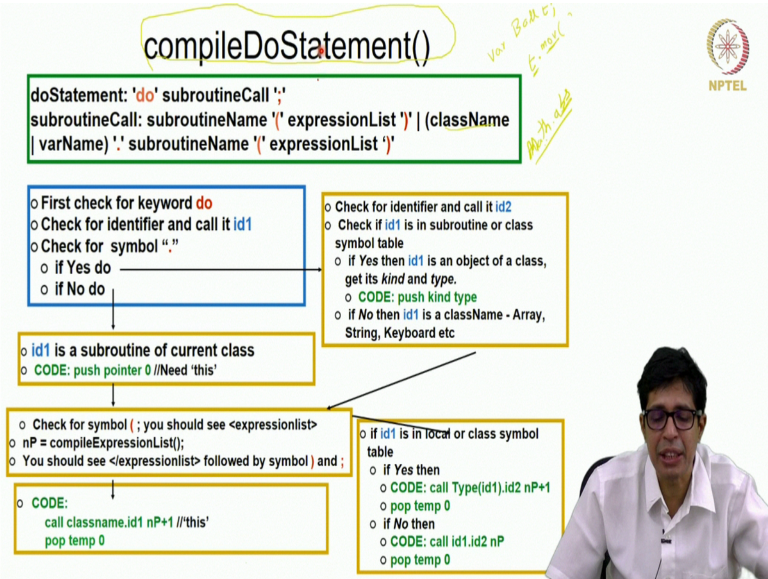
Now I should see/term and then you will see op, now op is star, so I will call math.multiply 2, 2 is the number of arguments and what will math.multiply 2 will do it will take these 2 arguments 5 and 5 and it will make it 25, so on top of the stack now I will see 25 okay, then again you will go, you will not see any op, so this ends here, this; so this ends here and so this expression gets core okay, so this is how is compile expression will work right and so and this

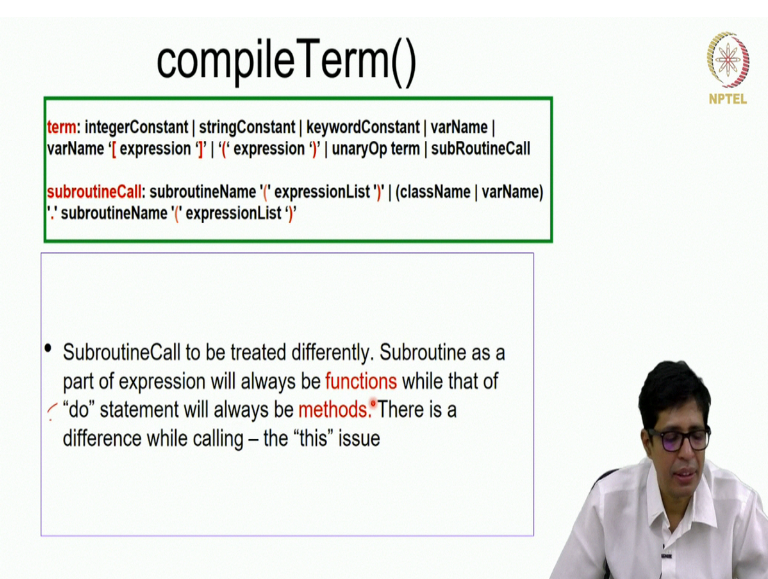
does left-to-right presidents, so 2+3 into 5 will be 25 and not 17, 2+3 into 5 will be 11 because first will go from left to right in that order, whatever you will see and this is how.

This are the code for op, so if I plus I should put add, add is a VM instruction, sub is a VM instruction and the VM instruction all this things you are seeing is a VM instruction and these 2 are from the maths library provided by the operating system right, so this is the code you should see.

So when I say compile expression you check for term, if you do not have term at all just leave it return, if you see at term than call compile term, then you check for a/term, then you see an op save it again you see another term, you should see another term all compile term, again you see/term and then you save that operate and if that op is plus minus like this, then you put is corresponding code to their, whatever we have put in this screen here on the right inside, put that code there, then again go and check if there is another op, save that op, again you should see a term, then call compile term, again/term then put for this whatever op is that and then keep repeating it till you do not see an op right, so this is about compile expression.

(Refer Slide Time: 11:52)





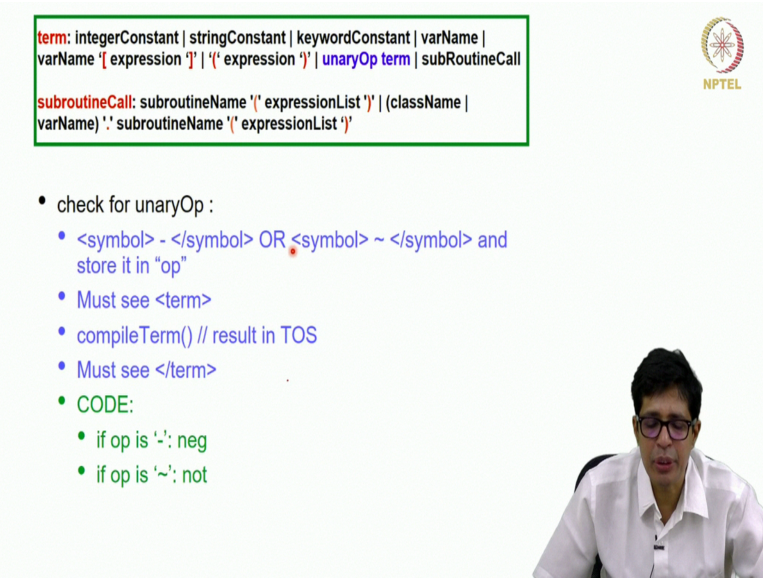
Now we will go and see compile term because everything depends on the term and compile term is the most trickiest of this whole thing, so let us see how we are going to address compile term, so term can be just an integer constant, a string constant, it can be a keyword constant, a var name, just a var name or a var name with an array, var name which is pointing to an array with an square brackets index and that can be an expression inside that and it can be just an expression with a, in a (, it can be a unary op term or it can be a subroutine call.

And as such you have seen for subroutine call, a subroutine call will be a subroutine name followed by an expression list, which is of the same class or I can say a class name or var name .subroutine name followed by a (expression list), so this is all that we need to do for compiling term, now the good for subroutine call, say, we have actually handle subroutine call as a part of your do statement right.

Now, why should we have, can we have we use the code, typically we do not want to use that code because this subroutine call will be part of an term if normally going to be a function, while the subroutine call that we saw as a part of do will be a method okay and there is a difference between handling a function and handling a method because in the case of matter when I am calling a method the first argument, the 0th argument should be this of that object right, so but in practice we do not see a method in the term right, so it is better as we have to go and check for every subroutine call whether that is subroutine name is a function or method, instead of doing it the very simple thing that we can do is that this subroutine call that is part of the term let us handle it separately.

So this is not going to be a method, these are all going to be functions, so there is no necessity for me to push that this there. Okay, while on the other thing, all the things that you see here would be basically, in the do statement will be mostly methods okay, so this is the basic difference here, of course I could have some var name .subroutine name in which case you know, I may need to put that this here, so let us see how does this lexically workout okay, so we are not using the code that we have done for the, you know do statement where I add a subroutine call for this, for the simple reason okay.

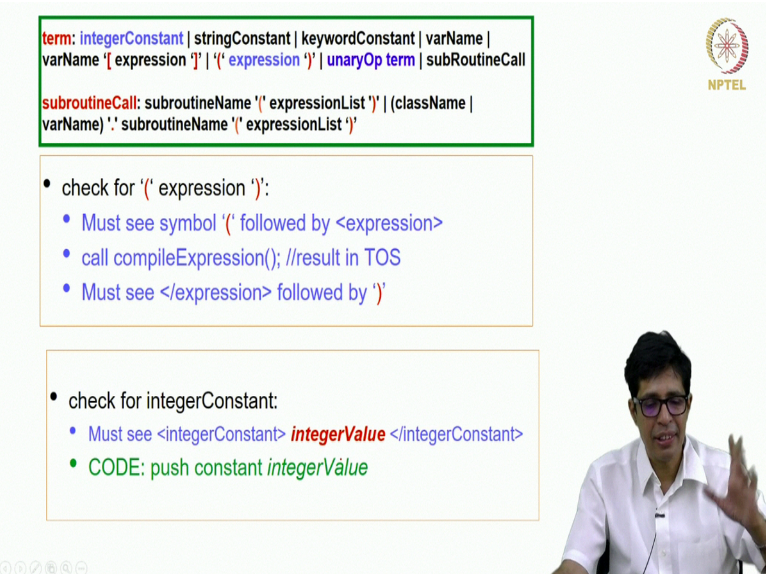
(Refer Slide Time: 14:58)



So let us start handling one by one, a term can have a unary op term, so first we need to check, so when we start entering the subroutine, first you just check it if there is a unary op, what is a unary of? It will be the symbol minus/symbol or symbol ~/symbol and so if you see the first thing as a symbol minus or symbol ~then you store it in op.

The moment you see that you must see a term, you call compile term, the result will be on the top of the stack, now you must see a/term, then you put this code, if you have seen an op is minus then you put neg, if you have put ~then put it is not right, so this is something that we need to do, this is for the unary op, so once you enter compile term, first check if you have symbol minus symbol or symbol ~/symbol and if it is then than do this and return, that is all, over, so that is one part, so every entry here in this rule we need to get something like this similar.

(Refer Slide Time: 16:05)



The next thing is suppose I see first symbol (than it should be followed by expression, then you call compile expression, the result will be the top of the stack and then you must see/expression and then followed by this symbol right, you must is less expression followed by this symbol, so that is, so this is done, so nothing, so there is no code dump as part of this because compile expression will anyway put the code on top of the stack, so this is for this/expression.

Otherwise you will see, the first one is integer constant, so you will see something like integer constant, integer values/integer constant, the moment you see this, so you have integer value here, just copy it here, so the code would be push constant whatever value, you will see integer constant/integer constant, now you put push constant here, that is all.

So what happens is when I enter compile term I repeat here, I see either symbol minus symbol or symbol ~symbol then you do all this things, the first thing that you see if it is there, the first thing you see is symbol (, the first thing you see than do this, the first thing you see is integer constant, integer value/integer constant then do this right, so then we will see the remaining things how we are going to handle as a part.

So compile term is going to be the longest routine, of course you will get more number of good as you are proceeding, so we have handle 3 such cases, now we have another 6 more cases which we will see in the next module. Thank you.