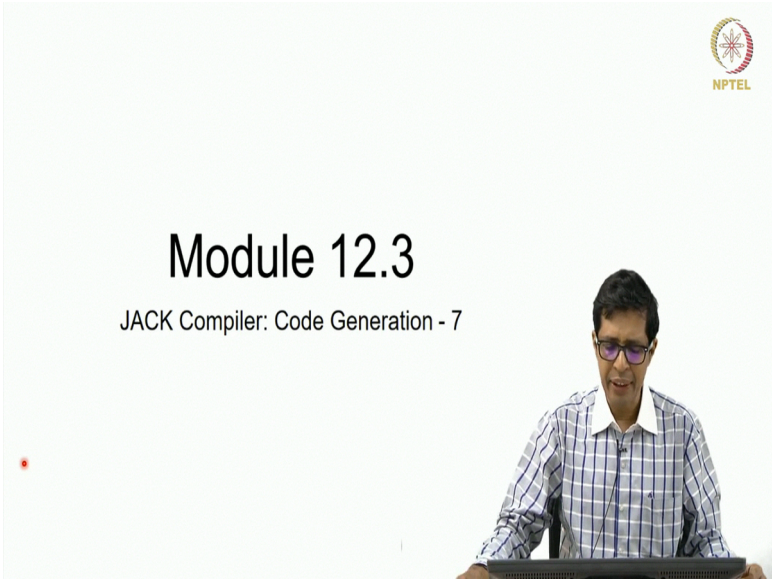
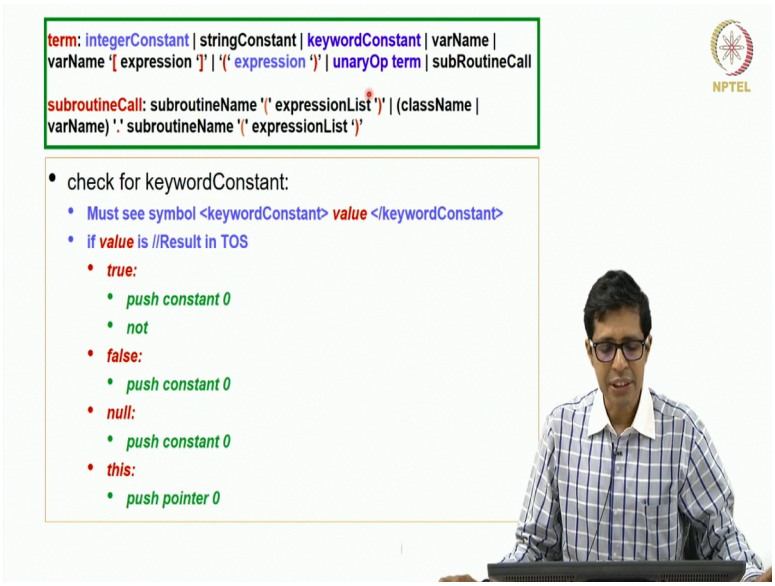
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module: 12.3
Jack Compiler: Code Generation - 7

(Refer Slide Time: 0:17)





Welcome to module 12.3 and in this module we will be going ahead with the code generation, so we are now doing compile terms, in which we have completed integer constant, expression and unary op term, now we see keyword constant, so a term can be just a keyword constant and what are the keyword constants, if you look at there are 4 keyword constants that we have, namely through, false, null, this, so when we are the compile term will look for integer constant, we look for unary op, we look for this starting (and based on those actions would

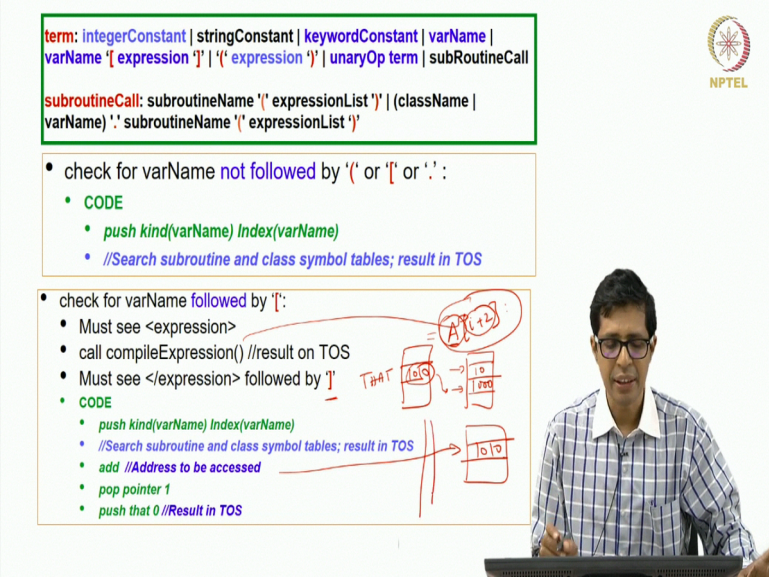
have been taken, if we does not find this 3 than it will look, if there is a symbol keyword constant followed by some value and/keyword constant, if this is then it has to take this particular path.

So the term will be checking for each one of this possible things and if this is one of the, if its keyword constant and it has to do this action, so this value can be one of these for values, namely true, false, null, this, so if you see a keyword constant values/keyword constants than if the values is true, then we actually push constant 0 and 2 not so essentially this becomes 1111 which is true right, so the result, always the result has to be in the top of the stack, so this is how do you do.

If it is a false then you just push constant 0, if it is null again push constant 0, if it is this, if you want this then always that this, the starting address of this will be pointer 0, so I will say push pointer 0, these for values will be there for keyword constant and for each one of these values we need to dump this code there right, so this as a green, so this code will be dump and at the end of this the result of this keyword constant will be on top of the stack.

So what we said compile term will leave the result of, will generate the code, which will leave the result of that particular code on top of stack and that is what we are doing here right, so this is about keyword constant.

(Refer Slide Time: 2:39)



Now we will go and see, if it is a var name, which is not followed by (or [or . That means it just this var name, so it is not is var name with that expression, so just var name that means it just an identifier, so it something like Y like EZ or so one of this identifiers, so what we need

to do is we have to push the value of that EZ or Y on to the top of the stack, so how do we do that?

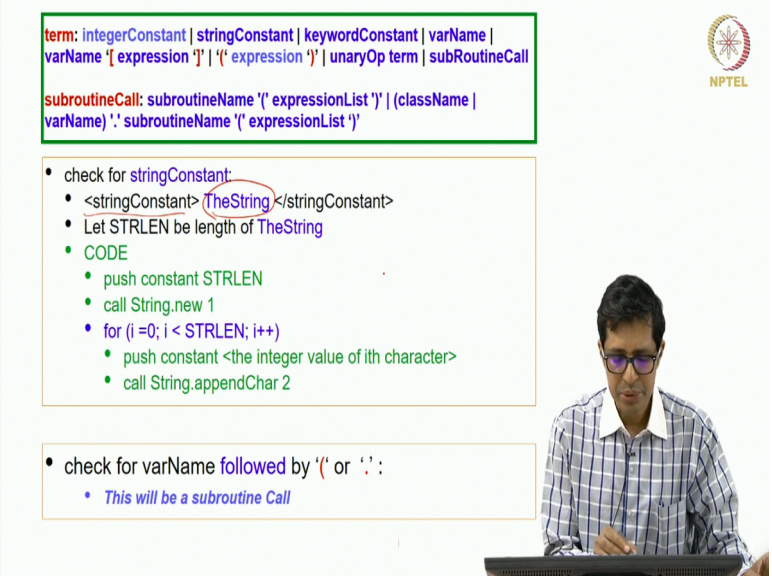
So this is the code, push the kind of that variable name and the indexed of variable name onto the stack, so this push will do that, so it will be push field, 0 or starting, 0 or whatever, so or local, 0 or org 0, so this will push the value of that variable onto the stack and finally the top of the stack does have this value and where do you get this kind of index, search for this variable name first in the subroutine class table and then in the subroutine symbol table and the class symbol table and then from that you will get this kind of index and if you put this code intimately the value of the variable will be on top of the stack and that is what we need.

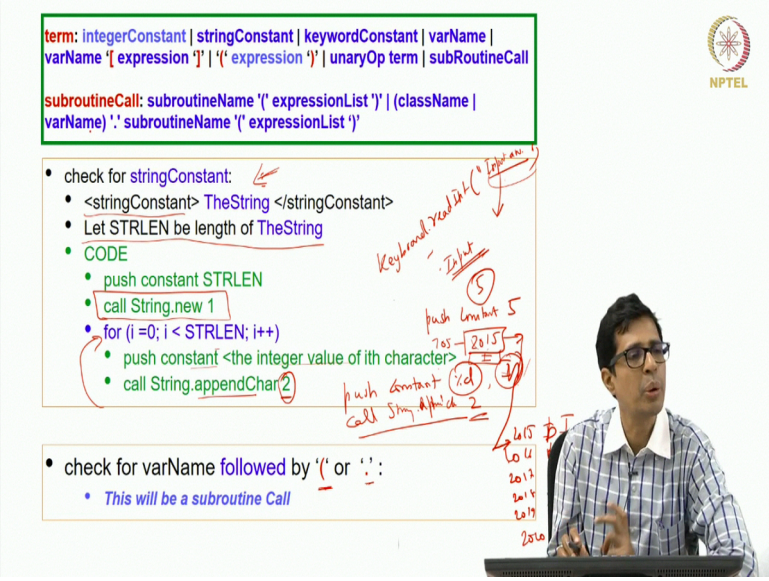
Suppose I have var name, followed by this square parenthesis right, so square brackets than that means it is an array variable, so the moment I see the square parenthesis I must see are expression, so immediately I will call compile expression and the result of that will be on top of stacks, so it is something like you know A of I +2 something like this, so this I +2 is an expression, so this compile expression will basically compile this, write code of I +2 and while you execute that code on the top of the stack the result of I +2 will be there.

So this is at the end of the compile expression you will see this, then you must see a/expression and then the closing square bracket, now you write this code, now you push kind an index of A right, A is the variable name here, so if it is a array variable what will be this push kind and index do, it will give you the starting address of A that we have seen, so here let I +2 be 10, now that starting address of A will get, so A will be say some local variable 2, so if I say push local 2 it will give you starting address of A, let it be someone 1000.

Now I do add, so this becomes 1010, so this is the add, add will give me 1010, I push, I pop this to pointer 1 so that will basically of 1010, now push that 0, so that is 1010 +0 is 1010, so push the content of 1010 onto the stack right and that what 1, so in the end of A of I +2 which is part of an express term, I need the content of A of I +2 on the stack, so I just put push that 0, so the result of this is on the of the stack, so this is the okay, so this is how we handle variable name.

(Refer Slide Time: 6:13)





Now the next thing is string constant right, so now we have to check for, so you will be looking like string constant actually the string, for example string constant enter the number right, so if I have say enter the number, so let me say print int or output int something like, than it will have something like keyboard.input int, let me just take this keyboard.read int some string will be there, now this will be reflected as string constant and whatever string, read your number or I can say input a number, so that input the number will be a string here.

Now what we need to do is we call string length, we actually computer string length, so this input a number will be something like input say let us say input, so input the string length is now 5 right, now we extract this string right, now what we do is, first we push constant string, length, so I will first put push constant 5 and I say call string.new with 1 argument which is 5,

so this will create a new string of length 5 and the starting address of that will be on the top of the stack, that is what string.new does.

So this will create some new strings say starting at 2015, so the new string will be, so 2015 will be on the top of the stack, so the new string will be some wherein 2015, 16, 17, 18, 19 okay, so this is where your string, so this will be input, that is what we need to create now right, so what we do is first 2015, first it will put everything as blank and 2020 will be null, so this is how this will do.

Now what we need to do is, so this was created as the string of size 5, now we see push constant of that character, so I have to push this, so what all this input everything should be stored here as integers right, so there is a one-to-one ask key map here right, so there is a one-to-one ask key map, so essentially if I say, if I want to put input what I do here, for I equal to 0, I less than whatever that 5, I ++ push constant, now I have to push I right, I is percentage D, in C if I put percentage D and I put I can that print of statement, this ask key value of I will be put here right.

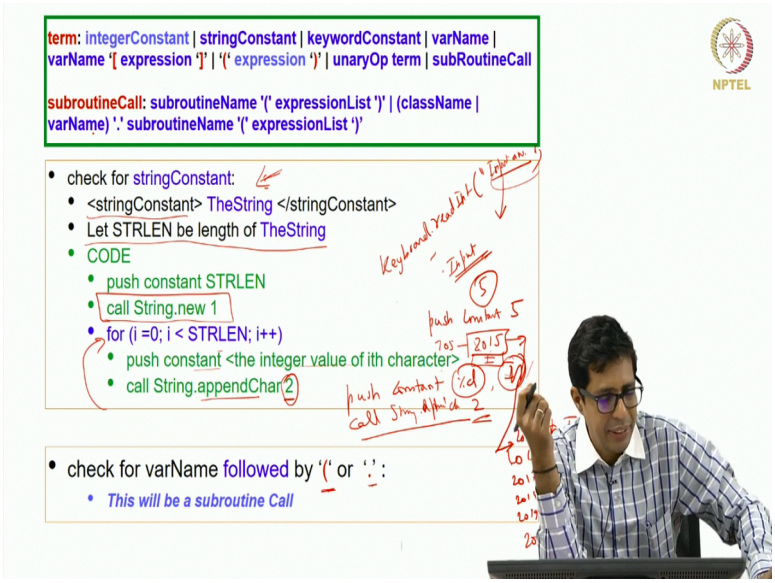
That is how we do this, so I put ask key value of input here, so what I push constant of that ask key value of I and then say call string.appendChar 2, why this 2? This 2 essentially means there are 2 arguments to this function for append char, 1 is the starting address of the string with is already there on the stack 2015, then I am pushing this value of I, the ask key value of I by this statement right and now I am calling string.appendChar which will take the first 2 arguments on top of this.

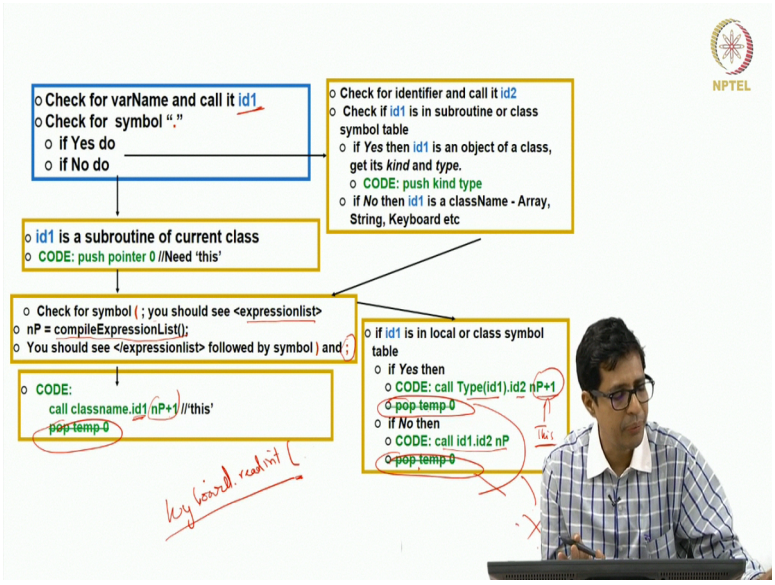
So what this will do? This string.appendChar 2 will do in 2015, string starting at 2015 please append I right, so it will go and append I in 2015 right, then next these goes on loop back, now I push the constant ask key character equivalent to N and again I will say call string append, so what it will do, it will just take this 2015, again 2015 is on the stack, so it will take this N, the ask key equivalent of N and append it to 2015 again I get N right and like that, I will keep on doing, so I will be appending input one after another.

So this for loop will take each character of the string as its ask key value and call string.appendChar to append it back, so and these and please note that there are 2 arguments one is the starting address of the string and another is the actual value that I need to append, so this is what we do with string constant, so at the end of this, if the term is the string constant what will be on the top of the stack? The top of the stack will have pointer the

starting address of the string and if you go than the entire string will be stored, so that is how do, so this is how string constant is handled right. Now, we now go and check for var name which is followed by either dot or this (, then that means this basically leads to a subroutine call.

(Refer Slide Time: 11:53)





Now the difference between the subroutine call that we saw in compile do statement and the subroutine call we are going to see here is that the subroutine automatically will give as a, will result in an output on top of the stack okay, all the subroutines, so in that case, these are all basically functions which will do right, which will return a value on the top of the stack right.

So one of the thing that we need to keep as mentioned earlier that there is a OS function call like keyboard, there is many OS functionalities which you do not define like keyboard, arrays, string, you know sys etc the do subroutine call only methods will be called for those OS routines right, but in this the subroutine call originating from evaluating a term functions will be called right, so for the OS classes like your sys, array, mems, string, math etc.

In the do, the subroutine call originating from do only methods will be called, in the subroutine calls originating from terms of only functions will be called right, so there is a difference between handling a function and handling a method and now we will see how the subroutine call for the term is going to be handled right, so this is very similar to what we have done, but there are some certain changes here and there.

First check for the variable name, we call that variable as ID 1, then check for the symbol . if there is no . then you do this step and this step, if there is . sorry if there is . you do this, that followed by this step and then the step, if there is no . then you do is 3 steps right, so if there is . then you again check for identifier because this will be var name.identifier call it ID 2, check if ID 1 is in the subroutine or symbol class, if yes then ID 1 is object of some class, so we need to get its kind and type, so and you push the kind and type of ID 1 because we need this for the subroutine, if no, then ID 1 is a class name array, so do nothing here at this step.

Then come here, then after that we need to see the symbol this because you have something like keyboard.read int some that starting parenthesis, so you see this starting parenthesis, then you will see expression list, then you call this compile expression list and it will return the number of parameters and it will also ensure that the parameters, that should argument that are need to be passed to this routine or that code for compile expression list will ensure that all the arguments are loaded in the stack in that order and after that you should see/expression list, followed by the symbol and; right.

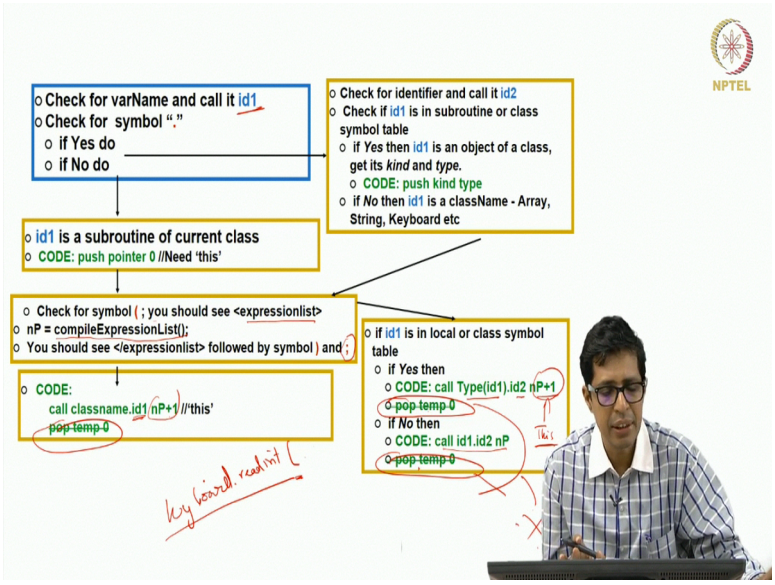
Then you come here if ID one is a local or class symbol then you just say call type ID 1.ID 2, type of ID 1.ID 2 nP+1, if it is ID 1 is of the OS routines than just say call ID 1. ID 2 nP, this nP+ 1 is needed because you are putting this right, so now the difference between do while and this statement is pop temp 0 is not necessary in the while statement we need that, but here we do not need this.

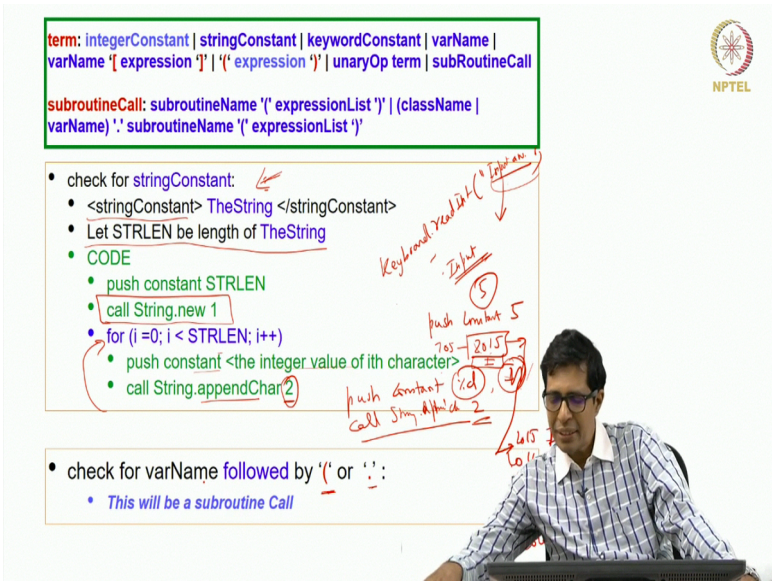
Similarly, if it was just ID 1 without a . then we push pointer 0 because a subroutine of the current class pointer 0, then you again check for this parenthesis this is, this (and expression

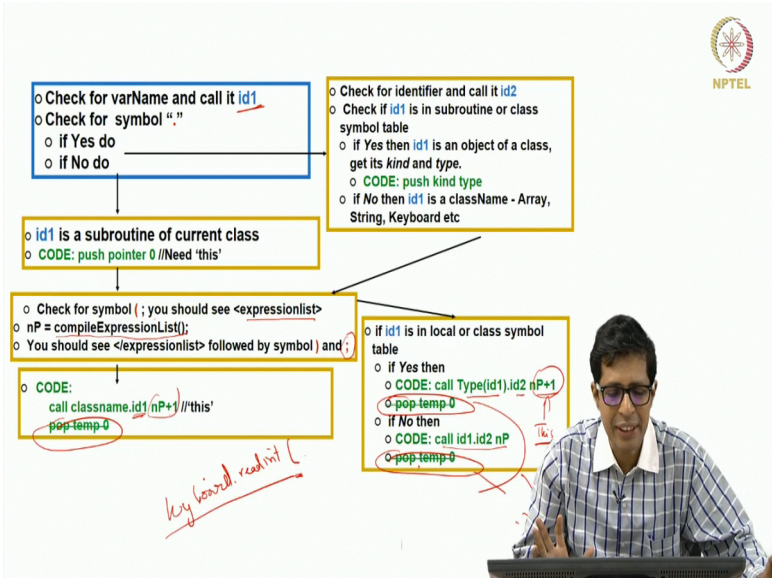
list, then number of parameters will come as part of the compilers expression list and of course this compile expression list will put the code where all the arguments of this are already loaded on the stack in that order, then you should see/expression list and symbol, followed by symbol) and then the symbol;

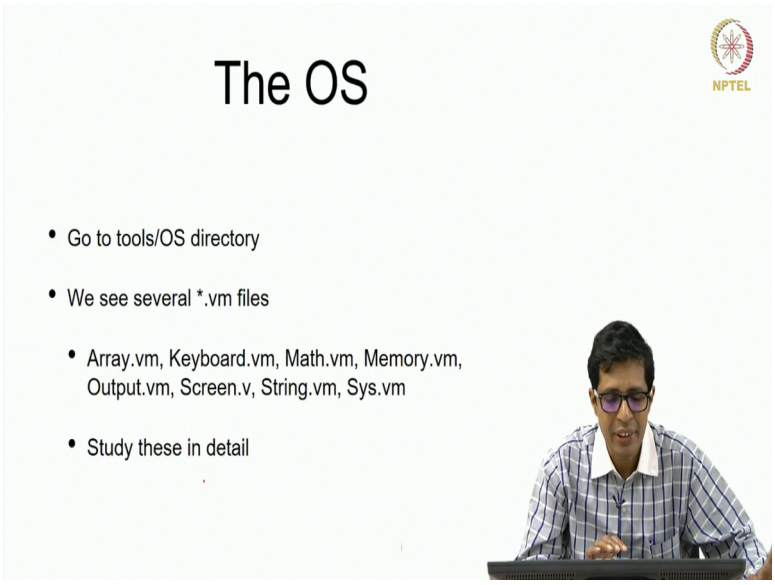
Now the code is call class name.id 1, whatever this ID 1 nP+1 because this is there, so this is what you do if it is not a dotted call, if there is dot call then you do this, this and this, so the difference between do and this is that this pop temp 0 is removed in all this, so that is the difference okay and, so this is how we handle subroutine call.

(Refer Slide Time: 17:21)









So with these we have finish all the 15 subroutines and need to be coded and whatever you see on green you need to have basically will be the code that is dumped out okay and this is how we construct the compiler, so I hope you all followed this and if you have any doubts please do put on the web but do not stop this effort, is going to be slightly you know involved and I have tried to explain as much as I can, of course, always whatevers can improve, I can improve in my explanation and you can also improve in your understanding what the best thing is that you work, you write the code, you get the code up and that will be a very interesting exercise, you will never regret right, and that is very important.

So unless you do this final coding you will never see the josh in this course, I would like you to end this course with that josh okay, so finished this coding right, once you finish this

coding your compiler is ready, now one of the thing that we need to understand next before you even start executing programs, which is understanding the OS interface.