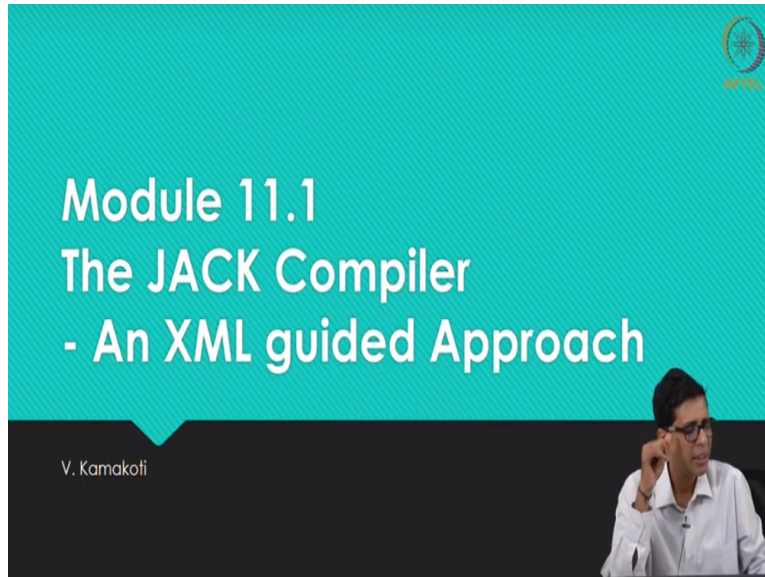


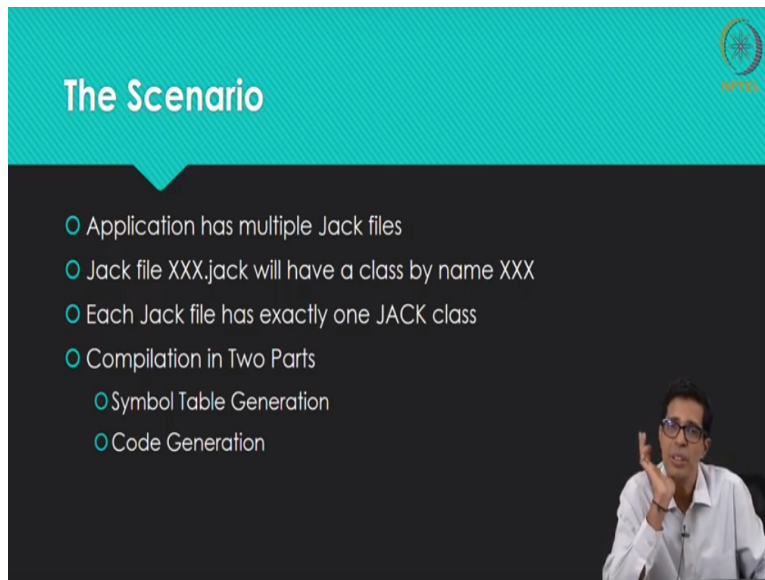
Foundations to Computer Systems design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module11.1

The Jack Compiler Backend: An XML guided approach
(Refer Slide Time: 00:15)



So welcome to module 11.1 and we start the jack compiler and we have already seen the theory behind compiling jack programs and what we now need to see is actually how to generate the code, so the entire jack compiler will be an XML graded approach and we will we have already created an XML file for it to his effect as a part of your project earlier project and now what we will be doing is that we will be using the XML file to actually generate the code.

(Refer Slide Time: 00:56)



The Scenario

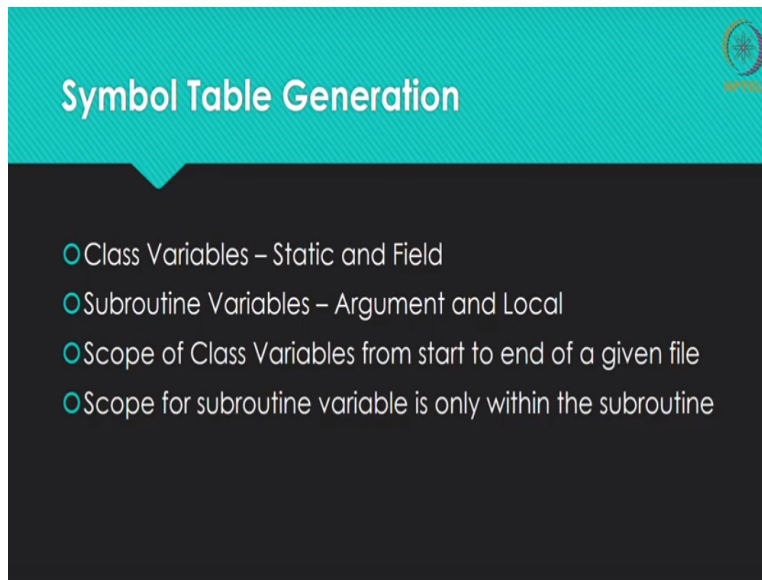
- Application has multiple Jack files
- Jack file XXX.jack will have a class by name XXX
- Each Jack file has exactly one JACK class
- Compilation in Two Parts
 - Symbol Table Generation
 - Code Generation

The slide features a teal header with the title 'The Scenario' and a small logo in the top right corner. The main content is a list of five bullet points on a dark background. A video inset in the bottom right corner shows a man with glasses and a white shirt speaking.

Now the scenario is that the application has any application that you are going to compile will have multiple JAC files each jack file will be named XX dot jack or whatever and that will a class by name X axis, so it i have square dot jack then we have a class by name square and each jack file will have exactly one class and that class will have lass variable and set of subroutine and methods or functions inside that as we had mentioned in the module 10.

The compilation will be in two parts one part is symbol table generation will generating the symbol table and the next part would be code generation the symbol table will be used for compiling the code and actually generating the code and this we have described the theory part of it we have described in the module 10 right.

(Refer Slide Time: 01:54)

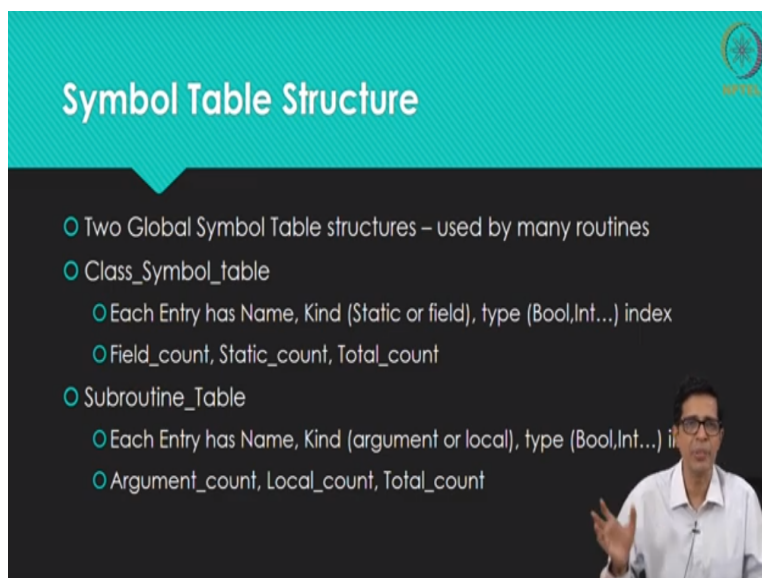


Symbol Table Generation

- Class Variables – Static and Field
- Subroutine Variables – Argument and Local
- Scope of Class Variables from start to end of a given file
- Scope for subroutine variable is only within the subroutine


Now when we look at simple table generation there are basically you know there are two types of variable that we need to handle their class variable which are basically static and filled there are two kinds of class variable and subroutine variable inside every subroutine there will be two class kinds of variable one of the arguments and another are local variables the scope of class variable is from start to end of the given file because each file will have one class so all the class variable are valid from towards the entire compilation process of that file well the scoffer subroutine variable is only limited within that subroutine, so this we have already seen here.

(Refer Slide Time: 02:41)



Symbol Table Structure

- Two Global Symbol Table structures – used by many routines
- Class_Symbol_table
 - Each Entry has Name, Kind (Static or field), type (Bool,Int...) index
 - Field_count, Static_count, Total_count
- Subroutine_Table
 - Each Entry has Name, Kind (argument or local), type (Bool,Int...) i
 - Argument_count, Local_count, Total_count



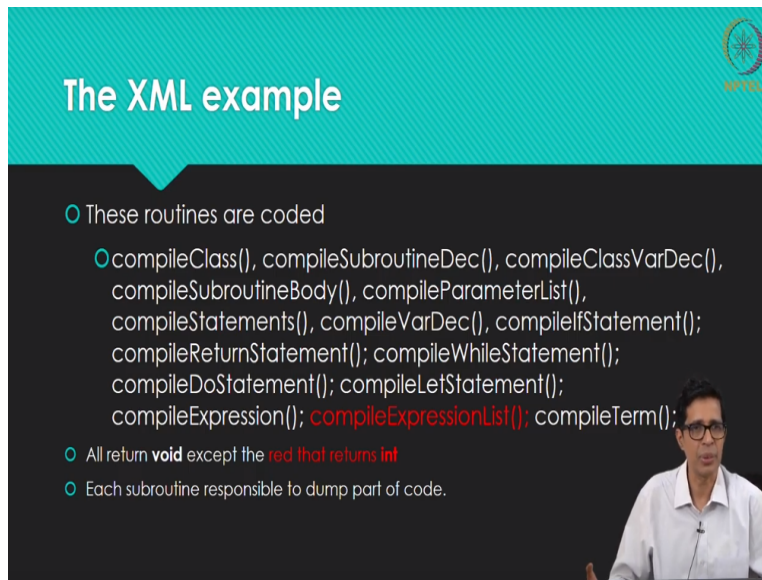
So essentially we have to construct two structures, two symbol table structure and these two symbol table structure are going to be global because they will be used by multiple routine as see so there are so there will be two global symbol table structures and this will be used by many routines one is the class symbol table which will have multiple entries and each entry will have these four things as we had mentioned earlier it will have the name of the variable it will have the kind of the variable in this case.

Since it is class it case be static or field variable then it will have type of the variable we Boolean into care or some class name and then index of the variable similarly then in addition to these entries there will be three other integers that are associated with the class symbol table meaning field account which will tell you the number of field variable that are there static count which will tell you the number of static variable that are there and then total count which is field to could plus static count similarly there are subroutine tables in the subroutine table we will have again there will be multiple entries.

So each entry will have name of the variable then it will have kind which is basically you know the argument are local, so there are two kinds of subroutine variable one will be argument an there can be local and then there is a type of variable so it can be Boolean or int and then there will be an index to that variable which will be you know the index as we had described earlier in our module ten in addition to these entries we will have argument count.

Will have local count argument count is the total number of argument variable that are there local count is a set local variable that are there in the symbol table and total count is the total number of variable this argument countless local count, so this these are two symbol table structure that we still basically create four which will aid in our compilation process okay.

(Refer Slide Time: 05:02)



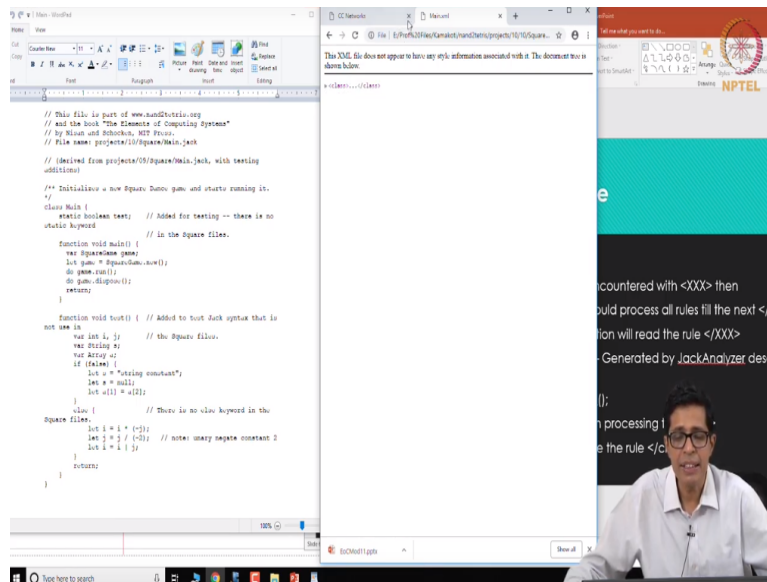
The XML example

- These routines are coded
 - `compileClass()`, `compileSubroutineDec()`, `compileClassVarDec()`, `compileSubroutineBody()`, `compileParameterList()`, `compileStatements()`, `compileVarDec()`, `compileIfStatement()`; `compileReturnStatement()`; `compileWhileStatement()`; `compileDoStatement()`; `compileLetStatement()`; `compileExpression()`; `compileExpressionList()`; `compileTerm()`;
- All return **void** except the **red** that returns **int**
- Each subroutine responsible to dump part of code.

Now as so we will know this is an XML guided compilation, so as a part of our module the previous project we were in the jack file was taken it was tokenized and it was analyzed and an XML file was created now there were thirteen routines there were fifteen routine that were basically written namely compiled class compiled subroutine declaration compiled class where declaration compiled subroutine body compiled parameter list compelled statements compiled a declaration compiled a statement compelled to turn statement compiled while statement compiled do statement compile eight statement compile expression compile expression list and compile them.

Out of these fifteen routine all of them return void except compile expression list which will tell you the number of expression that are there in the list right and each subroutine is responsible to dump part of the entire code that we are generating, so what we will be describing as a part of this module and probably the next is what is the contribution of each of these subroutine towards the code that is complete the VM code that is completely going to be generated from the jack code, so that is what we will basically do.

(Refer Slide Time: 06:32)



Now we will take the XML guidance, so the XML guidance is as follows so I will just show you let us take one example of a file, so we just take this what I have open one of the file here so the entire this is the main dot XML this is coming out of a main dot jack file and i will also show you the jack file here right, so that this is the jack file there is one class this is main dot jack so there is a class called main within this, this has one class variable and then one and two function.

So it is that is one class variable and to four subroutine inside this so this gets compiled as this XML if you open XML on chrome you get this so you this is the complete file so this is the XML file that has been generated in which every entity here has been every every token here has been accounted for and this is what you get so in the file so in the so what the current compiler will be using.

(Refer Slide Time: 08:23)

The XML Guidance

- Start in Main()
- Process rule by rule – when encountered with <XXX> then
 - call compileXXX, which should process all rules till the next </XXX>
 - On return, the calling function will read the rule </XXX>
- Project 10, Square, Main.xml – Generated by JackAnalyzer described in Module 9
 - Main will call compileClass();
 - compileClass() should finish processing till </class>
 - On return Main() should see the rule </class>

The XML Guidance

- Start in Main()
- Process rule by rule – when encountered with <XXX> then
 - call compileXXX, which should process all rules till the next </XXX>
 - On return, the calling function will read the rule </XXX>
- Project 10, Square, Main.xml – Generated by JackAnalyzer described in Module 9
 - Main will call compileClass();
 - compileClass() should finish processing till </class>
 - On return Main() should see the rule </class>

So the current compiler will be using this so we will start in main so the main the moment i see a word right as XXX immediately you will call the routine compile XXX for example when the main starts reading line by line of this file first it will see is class as you see here the moment it sees CLA SS class then immediately it will call compile class now the compile class process all the rule, so the compile class will start from this keyword class and it will process all the rule and when compiled class return then the next thing which main will see in slash class right.

So we start with main the moment i see XXX I immediately call compile XXX and when compile XXX return what I would see is slash XXX right, so main will start main will first to see

class, so immediately it will start compiled class and when compiled class return the next thing that main will see main will see is slash class, so compile class and it is associated and all the other subroutine it is going to call to consume all the rules and it will leave with this slash class alone.

So when main gets back it will only see slash class back, so this is how so this rule is applicable anytime, so as we are processing this XML file one by one whenever i see XXX immediately I will call compile XXX and when compile XXX returns back to the calling function then the cause, so if I am in function A and I see XXX immediately I will call compile XXX and when compile XXX return back to the function A the next token or the rule it will see will be slash XXX, so this is the XML guidance that we are going to get.

So I have taken this square main dot XML in the project 10 square which is generated by the jack analyzer and there you see main will call compiled class and whatever is between this compiled class and slash that entire thing compile clash and slash class that will be consumed by compiled class and when the control return back domain it will see slash class that is what we are seeing here and that is true for everything else so main will call compiled class then compiled should finish processing till slash class so on return main should see the rule slash class I will call every entry line of this XML file that we have created as a rule.

So we will consuming rule the first rule is class and because of that I called compiled class and then the compiled class will consume all the rules and finally when control returns back it will see slash class and this is true for every other thing that we are going to see, now let us go to the next slide, so this is the XML guidance that we will be getting as a part of this entire compilation process, so in the module eleven point two we will now go ahead and tell how a symbol table is populated we have seen the structure of the symbol table so far we will see how the symbol table is populated, thank you.