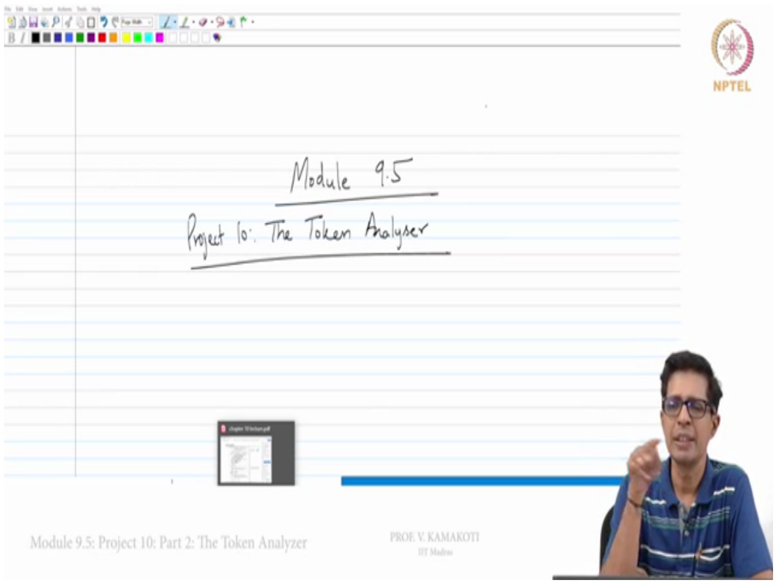
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 9.5
Project 10 Part 2 The Token Analyzer

(Refer Slide Time: 00:16)



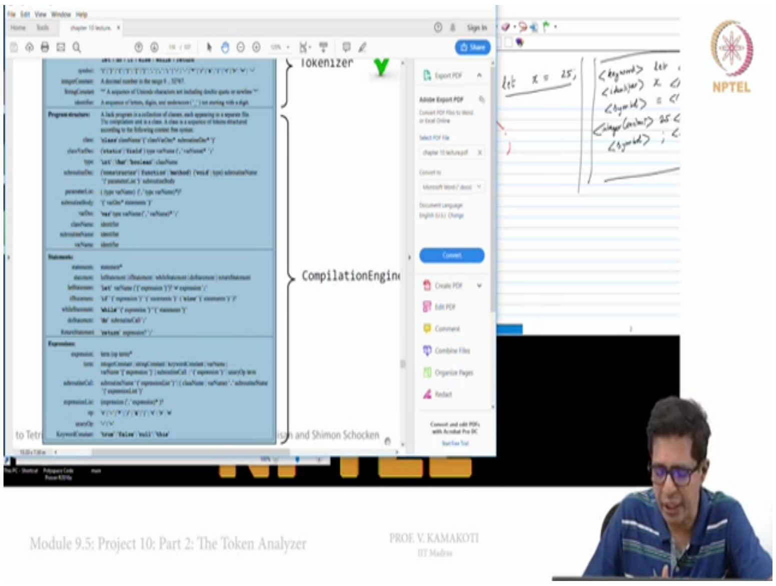
Module 9.5
Project 10: The Token Analyser
Module 9.5: Project 10: Part 2: The Token Analyzer
PROF. V. KAMAKOTI
IIT Madras
NPTEL

So welcome to module 9 point 5, so what we have seen so far is to take a jack program and convert it to a set of tokens this tokens I what we call as lexical elements and we the grammar had five lexical elements namely string constant, integer constant, symbol, terminal and identifier. So now once you have split the entire program into these lexical elements now the next step is to take the set of tokens.

Now you have a file in which there is one token per line and this is stream of tokens now we want to see whether that stream of tokens actually adhere or to the grammar, right that is call parsing and in the act of actually checking whether this adheres to the grammar right we also form another XML file which certain labels inserted so that, that will be useful for our code generation.

So now we have a (())(01:20) stream of tokens from the jack every entity every lexical element got converted a set of tokens, now we will take this stream of tokens one per line and now we will create another XML file which will not only check whether this grammar adheres to the a whether this token adheres to the grammar but also will insert certain more labels so that it becomes easy for code generation, right so that is the second part of your project 10, right.

(Refer Slide Time: 01:50)



Now let us go and see how this is going to work, so here so it so when you see here so this complete things as been used for your this is completely used all this lexical elements have been used by a tokenizer which has which we have done, the remaining three parts of the story namely your these rules is basically has the program structure and then followed so the program structure basically has class, the class rule essentially will content the keyword class followed by the name of the that class which an identifier then open of the braces then it will have zero or more class variable declaration, this class variable declaration star is zero or more class variable declaration followed by zero or more subroutine declaration ending with the end of this parentheses.

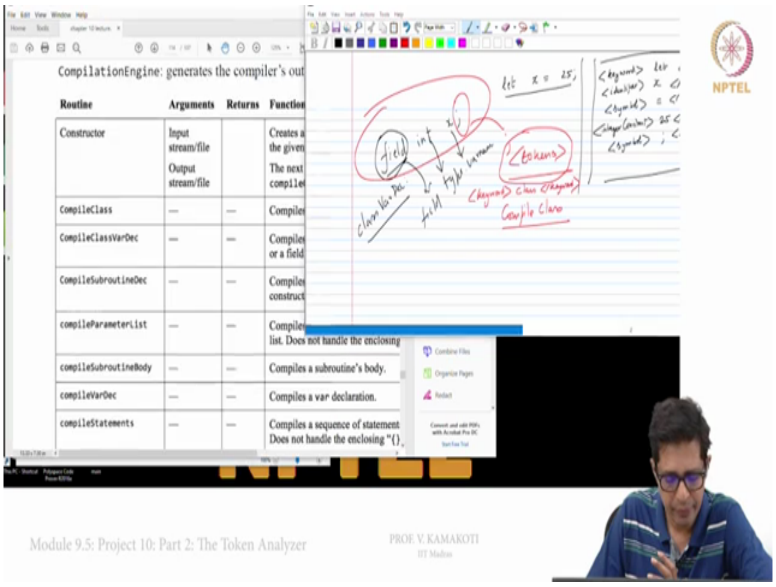
So the class variable declaration in term will have static or field one of this, this pipe actually stands for in this case I am moving the cursor here the pipe actually stands for either static or field then type then var name it will be an identifier followed by comma and zero or more var names ending with a semicolon, your type can be int, char, Boolean, class name etcetera. So if I have a so just to start here with it that is say I have field, int, x this adheres to this so this class variable declaration.

So I there is a the rule class variable declaration will first see field or int as you should see as field then a field or static it sees a field now it goes after this static or field you need to have a type now yes and the type is one of this int, char, Boolean or class name so this adheres to the type then you need a var name, var name this is an identifier this is var name followed by comma var name it would have a zero or more comma var name the star essentially says it could have zero or more comma var name yes I had zero here and then I have the semicolon here, so I have the semicolon here this comes to semicolon.

So this particular statement basically adheres to this root, right now how do we go and so this we have seen in a some understanding we have seen what we call as the what we have this particular how do we interpret these statements we have actually seen in module 9 point 3 sorry module 9 point 4 and now we will now see how do we implement this ok, so for every non terminal that we are seen.

So terminal is something all the lexical elements are call terminals because you do not have anything beyond them, right but if all that you see in the second part of this program structure all that you have seen here this are all non-terminals because after this there is something so what does the class have, class has a keyword call class then class name and something more but a terminal like integer constant as nothing more so that is why it call terminal int, ok.

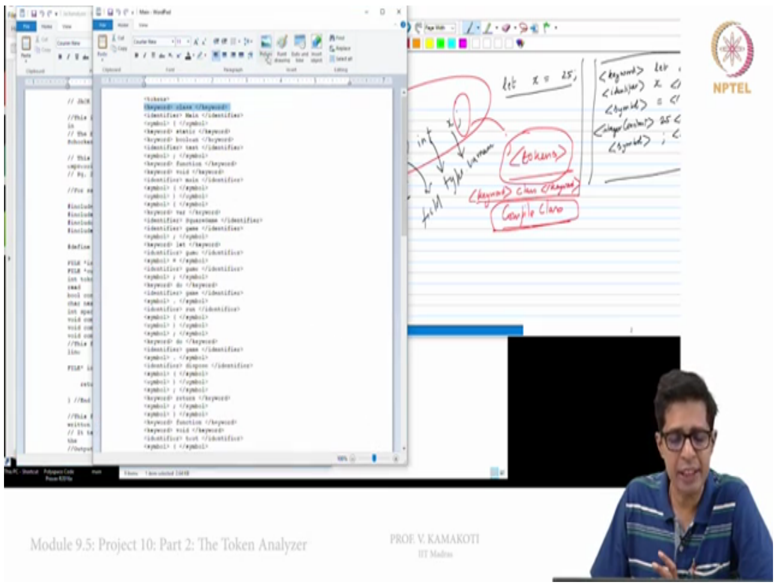
(Refer Slide Time: 05:52)


CompilationEngine: generates the compiler's out

Routine	Arguments	Returns	Function
Constructor	Input stream/file Output stream/file		Creates a the given The next compile
CompileClass	—	—	Compile
CompileClassVarDec	—	—	Compile or a field
CompileSubroutineDec	—	—	Compile construct
compileParameterList	—	—	Compile list. Does not handle the enclosing
compileSubroutineBody	—	—	Compiles a subroutine's body.
compileVarDec	—	—	Compiles a var declaration.
compileStatements	—	—	Compiles a sequence of statement Does not handle the enclosing "{}"

Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras



```
<tokens>
<keyword> class </keyword>
<identifier> Main </identifier>
<symbol> { </symbol>
<keyword> static </keyword>
<keyword> boolean </keyword>
<identifier> test </identifier>
<symbol> ; </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> main </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> var </keyword>
<identifier> SquareGame </identifier>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> let </keyword>
<identifier> game </identifier>
<symbol> = </symbol>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> run </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> dispose </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> return </keyword>
<symbol> ; </symbol>
<symbol> } </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> test </identifier>
<symbol> ( </symbol>
```

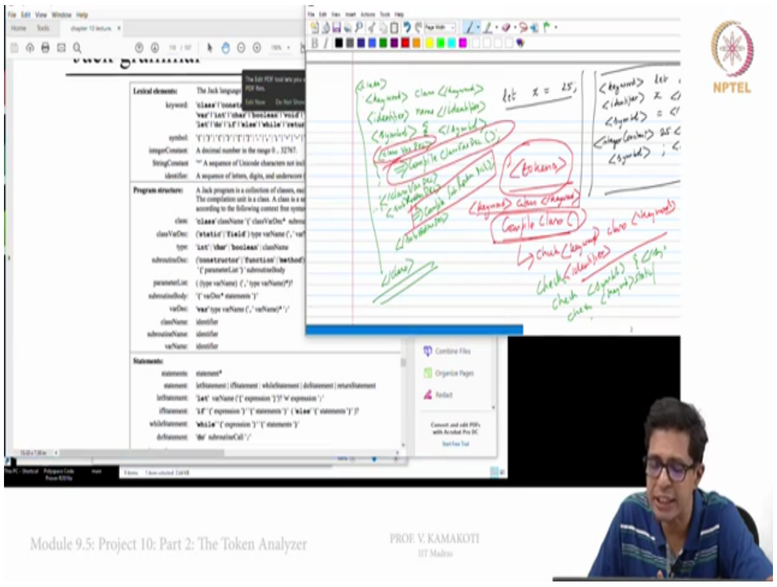
Now what we do here is as follows right, so what we do here is as follows. So we write these routines right, so we write this routines so and this will basically do the following, so I will have something call compile class that will be my first routine because every file starts with every token file start with tokens so in this implementation you write another C program that will read one by one the token and do the following first it will read the file will read tokens, right it need not do anything so it will read tokens and first check that is indeed a token file, ok so this is tokens right.

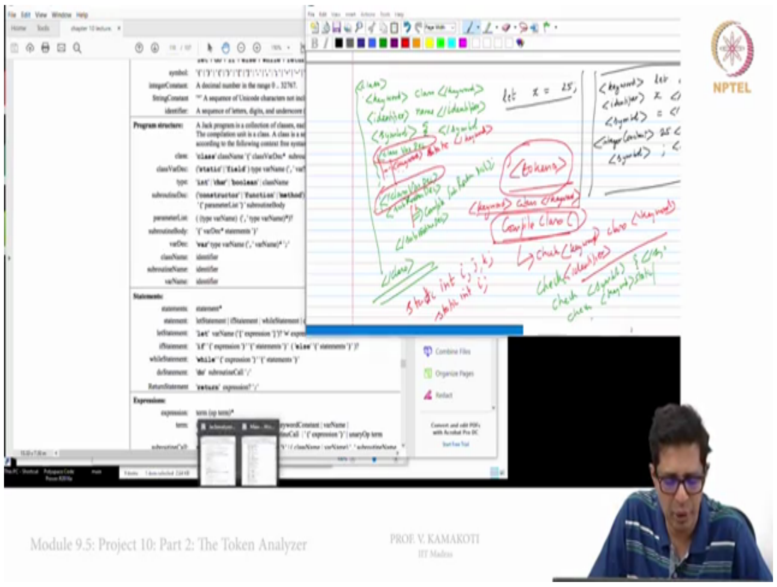
So tokens will be the first entry in your tokenized file, so it will just read is it and there is it then you call a routine call compile class, right now so after this tokens normally you will see a class, right you have to see a class so you should see something called in the token file you will call you will see something called keyword, class slash keyword, right. So let us even see a one example for this, so whatever we have done in the previous screen, so let us go and see so now this tokes, ok.

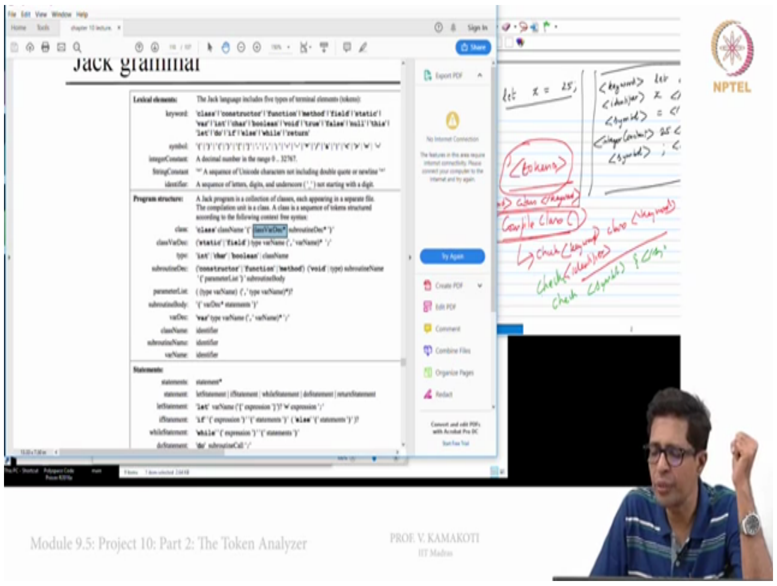
I am opening this, yeah first thing you whatever you see is tokens, the second thing immediately you will see is (keys) class the moment I see class you call this routine call compile class the moment i see class you call this routine call compile class and what we will compile class do now how do we write the code for compile class that is keep this here now let us now we go back to the grammar.

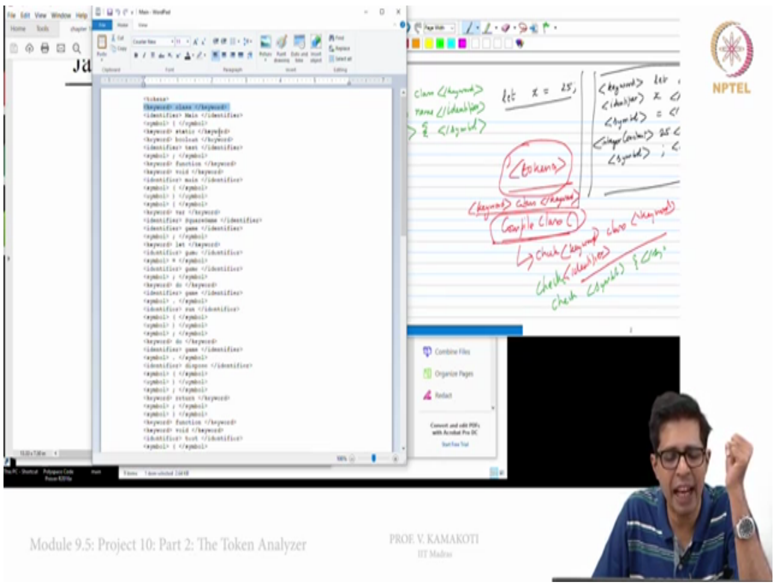
So the compile class is the routine that I am showing here, right now we go back to the grammar very quickly we go back to the grammar.

(Refer Slide Time: 08:30)








let x = 25;
<tokens>
<keyword> class </keyword>
Compile Class ( )
NPTEL
Module 9.5: Project 10: Part 2: The Token Analyzer
PROF. V. KAMAKOTI
IIT Madras

Yeah, so let us look at this right, the compile class should first check, check whether keyword class is there first token is keyword slash keyword that it needs to check, check whether the token is there then read the next token and see if it is going to be an identifier because class name the next thing is class name as you see here the next thing is class name and the class name is an identifier next you go and check whether identifier is there.

So if I find the keyword class immediately go and say yes keyword class is there immediately output that token, so what will this so compile class will do so the moment is see class I will write on the so whatever I am writing on green is the output rear so when I start reading that file I will see first tokens I do not do anything with tokens it is some magic number which basically says yes this is token file.

Now the first thing I need to see is class the moment I see class I will call the routine compile class so before calling that routine compile class so you do this you just write class and then call the routine compile class what will the compile class do it will see first that is given by this rule here it will first see whether the next token is of the type keyword class slash keyword if it is of the type immediately output that token with two spaces here just give two spaces this is a format in which there project wants you to do.

So just we will forward follow that format, so give two spaces here and then two spaces would be blank here then write keyword class slash keyword, right then you read the next token so you read the next token and what will the next token do it will now find out then we need to send whether it is a class name according to this rule the compile class should check whether it is a class name.

So it to check if there is an identifier the class name essentially directly goes to an identifier, so you will check whether there is an identifier, yes there is an identifier so just output that identifier again with the two spaces identifier some name will be there and slash identifier, the same token you just put that token after that you have to check for a symbol of this form slash symbol so you write that symbol also here symbol slash symbol, right.

Then there can be this class var declaration or not ok, so the star essentially means that there can be zero or more class var declaration, right so what we need to do is we go and check so we are finished of three tokens.Let us go back to these token files, right, so the first token was as you see here class second was main and third was symbol, now we need to go and see if there are some class var declaration if there are some class var declaration class variable declaration then the next should be a keyword static or a keyword field, so that we see here, so right.

So if there is going to be any class var declaration at all then the next token should be either static or field is the next token is not static or field then you know that this is gone that star so there is no class variable at all for this particular essay, class cannot need may not have every much then we go and start looking at subroutine declaration, so what we do is now we go and after you do here we go and check whether there is going to be an keyword which is either static or it is going to be keyword static keyword or keyword field keyword,

If that is going to be there then immediately you write with the two spaces here class var dec and here you call this compile class var dec this is another routine. Now currently it was compile class that compile class will now call compile class var dec, right. Now what will compile class var dec to will do later after that the compile class var dec finishes, we will again comeback and output slash class var dec again with the same two spaces here from this, right.

Now if I do not find the next keyword as static or field at this point then there is nothing there is no class var dec, so that is what this class var dec star means again I am repeating star means there can be zero or more class var dec. Now you and go see subroutine dec, now then you go and see whether there is subroutine dec or not what you will see the next keyword you go and see next token you have going to see whether it is constructor function method one of this should be there, if it is there then it is subroutine dec otherwise nothing is there.

So you go and check after you finish this class var dec here and then you go and check the next token whether it is constructor function or method if it is there then again what you do again with the same instruction you just write the subroutine dec, subroutine declaration then you again here you are calling compile subroutine declaration after this fellow finishes you again comeback and write slash subroutine declaration there you see and then comeback and align this and write slash class, ok.

So essentially what we will be here is whatever is output by this compile class var dec that will be part of the statement and what will be the output by compile subroutine will be part of the statement will be part of this output, ok right. So this is how compile class will do, so compile class will check for keyword first it will put class there and then it will check for keyword class if it is there is a keyword class it puts two intention to spaces all the output it is it will put it two till it comes to class var dec.

If there is some class var declaration then only you put these two statements, right class var dec and then this and this right. Now what will compile class var dec two compile class var dec will go and see ok, whether it is states that the current token is static or field it will output that token, so let us see what compile class var dec two does, so it is here so I am just erasing this to tell you what is going to do.

So compile class var dec again you go to the rule here class var dec it will check whether static field if it is there yes it is going to be there because you already checked you now put here keyword let us say without last of generally the static slash keyword but again with two gaps here from this so this is four gaps from the beginning, right then it will go and check if it there is a type again the type can be as you see here the type where is the type you can go here and see your type.

Type can be int, char, boolean or some class name so it will check for the type so the type can be either a symbol, keywords like int, char, Boolean or an identifier like class name, so you can put it here and then it will go and check if there is a variable name there is another identifier which is a variable name then it will check whether there is a then it will check whether there is a comma, if there is a comma then there are more variables there if there is no comma then there is nothing.

So I could have so essentially this rule capture something like static for example int, static int i comma j comma k where int there are three variables here after static int I could also have with the static int i that is all, so then only one variable so that accounts for so could I should this class variable declaration should start with three that will have say minimum three tokens the first token will be of the type keyword static or keyword field, second would be of type is again a keyword int, keyword char keyword Boolean or keyword identifier class name.

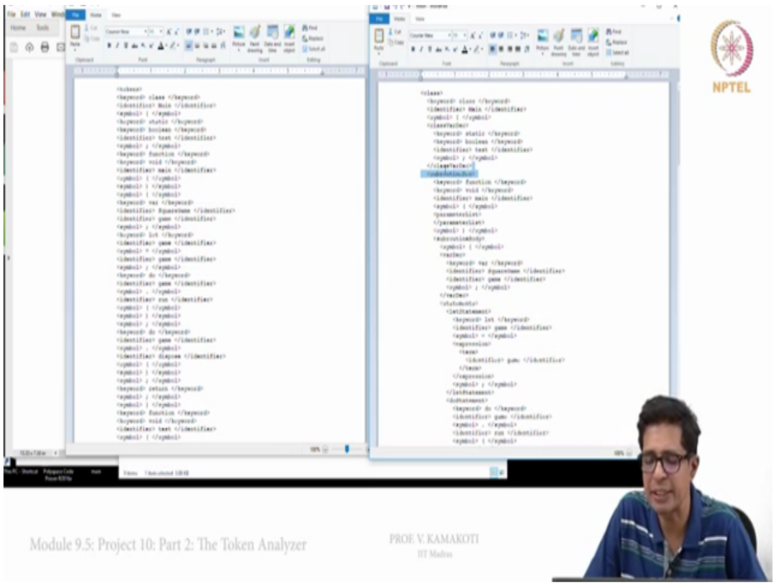
So that will be one of this tokens next and then this var name should actually be an identifier that should followed by an identifier and then I could have comma and more identifiers or not and it should end with a semicolon, so I will check for the semicolon. So all those token when I as an I see here I will also keep dumping it here as a part of the original file with two spaces away, right that is it.

Similarly we will go so now class var dec finishes here, now we will see I have to write first subroutine declaration, so subroutine declaration is start will now go to subroutine body, subroutine body will have a variable declarations which is of this form var type var name comma var name star, so it will call compile so if the compile subroutine dec will calls compile subroutine body, (subri) compile subroutine body will first call compile var dec and then it will call compile statements.

The statements will have zero or more statement the plural and this is a singular and each statement will be let if while do return each less statement will become let var name expression etcetera and it goes away. So whenever I am calling a particular subroutine compile subroutine I write that name write before and with I write that header there, so class var dec for example I am calling sorry I am calling variable declaration class var dec so I put slash class var dec then call compile class var dec and again after a comeback I put this class slash class var dec.

So these are the labels that I am inserting into this entire stuff, so I will just given example so this is how you write the entire grammar one after one very carefully but I will give you some more clues also before we go forwards, so let me take these two files, right.

(Refer Slide Time: 21:51)



```
<tokens>
<keyword> class </keyword>
<identifier> Main </identifier>
<symbol> { </symbol>
<keyword> static </keyword>
<keyword> boolean </keyword>
<identifier> test </identifier>
<symbol> ; </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> main </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> var </keyword>
<identifier> SquareGame </identifier>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> let </keyword>
<identifier> game </identifier>
<symbol> = </symbol>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> run </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> dispose </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> return </keyword>
<symbol> ; </symbol>
<symbol> } </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> test </identifier>
<symbol> ( </symbol>
```

```
<class>
  <keyword> class </keyword>
  <identifier> Main </identifier>
  <symbol> { </symbol>
  <classVarDec>
    <keyword> static </keyword>
    <keyword> boolean </keyword>
    <identifier> test </identifier>
    <symbol> ; </symbol>
  </classVarDec>
<subroutineDec>
    <keyword> function </keyword>
    <keyword> void </keyword>
    <identifier> main </identifier>
    <symbol> ( </symbol>
    <parameterList>
    </parameterList>
    <symbol> ) </symbol>
    <subroutineBody>
      <symbol> { </symbol>
      <varDec>
        <keyword> var </keyword>
        <identifier> SquareGame </identifier>
        <identifier> game </identifier>
        <symbol> ; </symbol>
      </varDec>
      <statements>
        <letStatement>
          <keyword> let </keyword>
          <identifier> game </identifier>
          <symbol> = </symbol>
          <expression>
            <term>
              <identifier> game </identifier>
            </term>
          </expression>
          <symbol> ; </symbol>
        </letStatement>
        <doStatement>
          <keyword> do </keyword>
          <identifier> game </identifier>
          <symbol> . </symbol>
          <identifier> run </identifier>
          <symbol> ( </symbol>
```

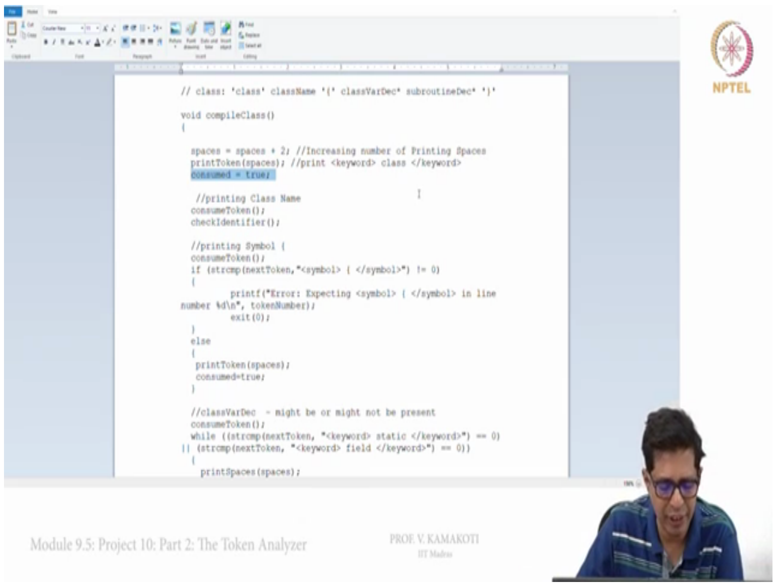
Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras

So I am taking both of these files and let us see it is side by side. So first tokens get ignored then keyword class the moment I see a class I would have outputted class then keyword class slash keyword then I need a identifier that is main, so main sorry I opened the wrong file one minute just, so this is main dot exe, ok yeah so your main identifier main then symbol slash so note that I have given two spaces here, right so there are two spaces that are given here then class var dec because since I see a static here I have now go into class var dec and so that as static keyword then type this boolean is for the type and then identifier is the test and then semicolon, so that is one class var declaration.

Then I am having function void main so that is a subroutine declaration, so I call subroutine declaration again within the subroutine declaration now the subroutine declaration now finds function void main and then parameter (())(23:23) so we go by the grammar has an invent. So whenever I so in the grammar when we are so when we are coding, so this is how we code the entire stuff, so let us go back to this main routine here.

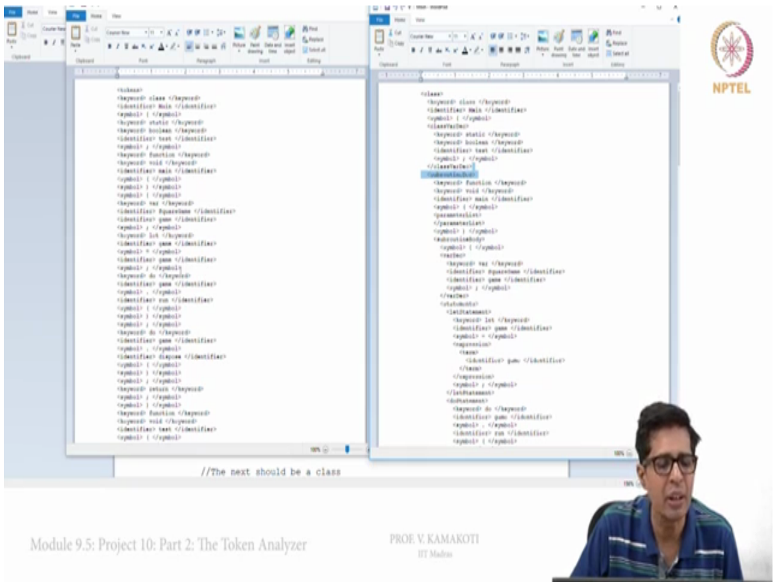
(Refer Slide Time: 23:42)



```
// class: 'class' className '{' classVarDec* subroutineDec* '}'

void compileClass()
{

  spaces = spaces + 2; //Increasing number of Printing Spaces
  printToken(spaces); //print <keyword> class </keyword>
  consumed = true;

   //printing Class Name
  consumeToken();
  checkIdentifier();

  //printing Symbol {
  consumeToken();
  if (strcmp(nextToken,"<symbol> { </symbol>") != 0)
  {
          printf("Error: Expecting <symbol> { </symbol> in line
number %d\n", tokenNumber);
          exit(0);
  }
  else
  {
   printToken(spaces);
   consumed=true;
  }

  //classVarDec  - might be or might not be present
  consumeToken();
  while ((strcmp(nextToken, "<keyword> static </keyword>") == 0)
|| (strcmp(nextToken, "<keyword> field </keyword>") == 0))
  {
    printSpaces(spaces);
```

Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras



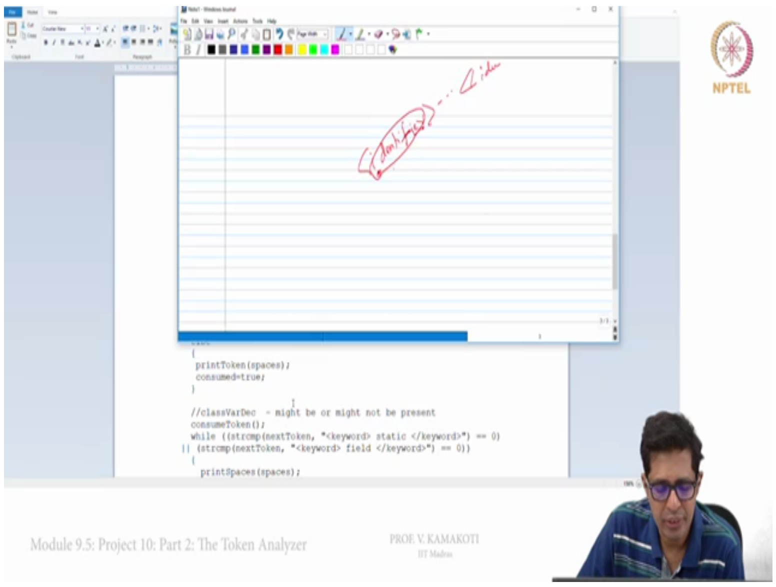
```
<tokens>
<keyword> class </keyword>
<identifier> Main </identifier>
<symbol> { </symbol>
<keyword> static </keyword>
<keyword> boolean </keyword>
<identifier> test </identifier>
<symbol> ; </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> main </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> { </symbol>
<keyword> var </keyword>
<identifier> SquareGame </identifier>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> let </keyword>
<identifier> game </identifier>
<symbol> = </symbol>
<identifier> game </identifier>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> run </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> do </keyword>
<identifier> game </identifier>
<symbol> . </symbol>
<identifier> dispose </identifier>
<symbol> ( </symbol>
<symbol> ) </symbol>
<symbol> ; </symbol>
<keyword> return </keyword>
<symbol> ; </symbol>
<symbol> } </symbol>
<keyword> function </keyword>
<keyword> void </keyword>
<identifier> test </identifier>
<symbol> ( </symbol>
```

```
<class>
  <keyword> class </keyword>
  <identifier> Main </identifier>
  <symbol> { </symbol>
  <classVarDec>
    <keyword> static </keyword>
    <keyword> boolean </keyword>
    <identifier> test </identifier>
    <symbol> ; </symbol>
  </classVarDec>
  <subroutineDec>
    <keyword> function </keyword>
    <keyword> void </keyword>
    <identifier> main </identifier>
    <symbol> ( </symbol>
    <parameterList>
    </parameterList>
    <symbol> ) </symbol>
    <subroutineBody>
      <symbol> { </symbol>
      <varDec>
        <keyword> var </keyword>
        <identifier> SquareGame </identifier>
        <identifier> game </identifier>
        <symbol> ; </symbol>
      </varDec>
      <statements>
        <letStatement>
          <keyword> let </keyword>
          <identifier> game </identifier>
          <symbol> = </symbol>
          <expression>
            <term>
              <identifier> game </identifier>
            </term>
          </expression>
          <symbol> ; </symbol>
        </letStatement>
        <doStatement>
          <keyword> do </keyword>
          <identifier> game </identifier>
          <symbol> . </symbol>
          <identifier> run </identifier>
          <symbol> ( </symbol>
```

```
//The next should be a class
```

Module 9.5: Project 10: Part 2: The Token Analyzer

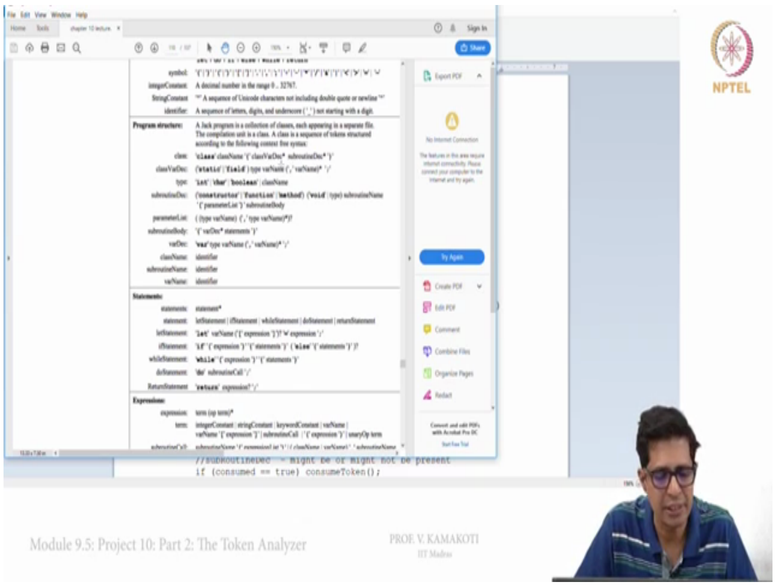
PROF. V. KAMAKOTI
IIT Madras



```
  {
   printToken(spaces);
   consumed=true;
  }

  //classVarDec  - might be or might not be present
  consumeToken();
  while ((strcmp(nextToken, "<keyword> static </keyword>") == 0)
|| (strcmp(nextToken, "<keyword> field </keyword>") == 0))
  {
    printSpaces(spaces);
```

Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras

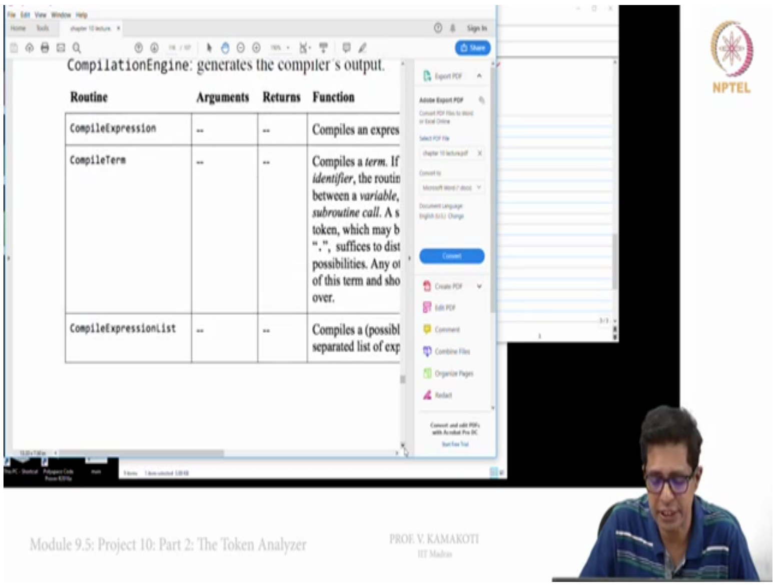


symbol:	'{' | '}' | '(' | ')' | '[' | ']' | '.' | ',' | ';' | '+' | '-' | '*' | '/' | '&' | '|' | '<' | '>' | '=' | '~'
integerConstant:	A decimal number in the range 0 .. 32767.
StringConstant	'"' A sequence of Unicode characters not including double quote or newline '"'
identifier:	A sequence of letters, digits, and underscore ('_') not starting with a digit.
Program structure:	A Jack program is a collection of classes, each appearing in a separate file. The compilation unit is a class. A class is a sequence of tokens structured according to the following context free syntax:
class:	**'class'** className '{' classVarDec* subroutineDec* '}'
classVarDec:	(**'static'** | **'field'**) type varName (',' varName)* ';'
type:	**'int'** | **'char'** | **'boolean'** | className
subroutineDec:	(**'constructor'** | **'function'** | **'method'**) (**'void'** | type) subroutineName '(' parameterList ')' subroutineBody
parameterList:	((type varName) (',' type varName)*)?
subroutineBody:	'{' varDec* statements '}'
varDec:	**'var'** type varName (',' varName)* ';'
className:	identifier
subroutineName:	identifier
varName:	identifier
Statements:	
statements:	statement*
statement:	letStatement | ifStatement | whileStatement | doStatement | returnStatement
letStatement:	**'let'** varName ('[' expression ']')? '=' expression ';'
ifStatement:	**'if'** '(' expression ')' '{' statements '}' (**'else'** '{' statements '}')?
whileStatement:	**'while'** '(' expression ')' '{' statements '}'
doStatement:	**'do'** subroutineCall ';'
ReturnStatement	**'return'** expression? ';'
Expressions:	
expression:	term (op term)*
term:	integerConstant | stringConstant | keywordConstant | varName | varName '[' expression ']' | subroutineCall | '(' expression ')' | unaryOp term
subroutineCall:	subroutineName '(' expressionList ')' | (className | varName) '.' subroutineName

```
//subroutineDec  - might be or might not be present
if (consumed == true) consumeToken();
```

Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras



CompilationEngine: generates the compiler's output.

Routine	Arguments	Returns	Function
CompileExpression	--	--	Compiles an expres
CompileTerm	--	--	Compiles a *term*. If *identifier*, the routin between a *variable*, *subroutine call*. A s token, which may b ".", suffices to dist possibilities. Any ot of this term and sho over.
CompileExpressionList	--	--	Compiles a (possibl separated list of exp

Module 9.5: Project 10: Part 2: The Token Analyzer

PROF. V. KAMAKOTI
IIT Madras

So this is the entire code that we have written here, for our quick understanding so we will just keep it at yeah, so for our quick understanding here so when we say compile class, right we want to before we call compile class so let us go back to the main routine so this is a main function again we have arcc, arcv so it takes a file inside which is a token file and this gives you a analyze file.

The first thing that we do here is that we just ask have you read that so we have something call also what we need to maintain here is one global variable call next token and a global variable call consume, so you keep reading from this file so different subroutine will be reading from this token, so it is better that we have the input file, output file the token everything as global variables, so there any routine can start reading from it.

So every routine so we have two variables one is called next token another is called consumed, consumed is a Boolean variable next token is a string so the next token will you read a token and if there token is not return into the new file, so whenever you read a token finally the token gets return into the file, right so that is what we have seen, so what we have done here just to again go back and tell you.

So all this tokens we have reproduced here, so this is the what you see on the left hand side is the input file, right hand side is the output file all the tokens that we have seen here or again reproduced here but with insertion sub some levels like class, class file back subroutine dec etcetera and this will be useful what you have see on the right hand side will be useful for code generation, so that is the analysis that we have done here, right.

So what we have do here so basically every time you consume a token and you say that consumable token convince you read a token and then you will say that it is still (())(26:05) to be used it is still to be return back and if the you read the token into this global variable call next token and you will find out whether that the first thing that you check is it is tokens or not.

So normally every executable file will read the first part and there will be a magic which tells what short of executable it is, you have for windows operating system we have a different executable for Intel architecture we have different executable for sun we have different executable, so what sort of executable it is? What short of file it is, file type? So that is always there on the top of the file there will be some magic number by which the operating system will identify what is this file, ok.

So here by looking at this token it is it will say it is a tokenized file for jack or for the virtual machine, right then what you do the next thing is you go you consume the next token and you check whether the token is of the type class and if it is of the type class if it is not of the type class you just say there is an error but if it is of the type class you immediately write class then you call this compile class and after that compile class returns you just writes slash class that is all, right.

So this is what we need to do, right at the end we go and check whether there is a slash tokens etcetera, so this is your main file does. Now we will go to compile class so the we will now see what is compile class, right so this is compile class so I have something call spaces,

spaces tells me how many spaces we are moved and spaces also we can emit as a global variable, so I incrementing the spaces weight too.

Now I already seen this class that is why I have come here the keyword class viewer so I print token with spaces these many spaces, right and I now I say I consumed it, so in the moment I have print it so consumed equal to true. Now I have consumed the next token and I check if it is an identifier, right so how do we check whether it is an identifier so I just see if the so there will be a of the one (identi) less than symbol identifier greater than symbol, right.

So how do you check something is an identifier this is the way we go about checking it. So it will be of the form identifier dot, dot slash identifier, so you go and check from the first character to the, so the first character to the 10th character or first to 10 is of the form identifier, right you can check that and if it is identifier, if it is correct then we can say ok, so the next fellow is an identifier.

So I will go and check if there is an identifier and the checked identifier if it is an identifier it will take and print it we need to just print it, then we take the next token and see if it is of the form symbol with this, if it is yes then fine then now we go and see we consumed the next token and see if your next token is of the form static or field, if it is of the form static or field then you say class var dec and you call compile class var dec and after that you print finish to the slash class var dec and I have return a while statement here because it can be class var dec star if you look at the grammar for this it was class var dec star, so it can be zero or many class var declaration, right.

Similarly I go and check whether this is a I go and check if the next is a constructor or function or method, right that means then it is a subroutine declaration and again I put a while statement because we have a subroutine dec star, ok so that is the statement here right. So let us go back and see here, so we have subroutine dec star for the compile class, right and the subroutine whether there will be that or not is going to be given by this constructor.

So the moment I find that the yes there is a subroutine then I will compare I will call the routine call compile subroutine declaration so and that it will take care of it and then just print it. So the same thing so what we have described here I also shown I am also showing you the code of how I am have coded this class, the same thing I follow for subroutine dec the same thing I follow for class var dec.

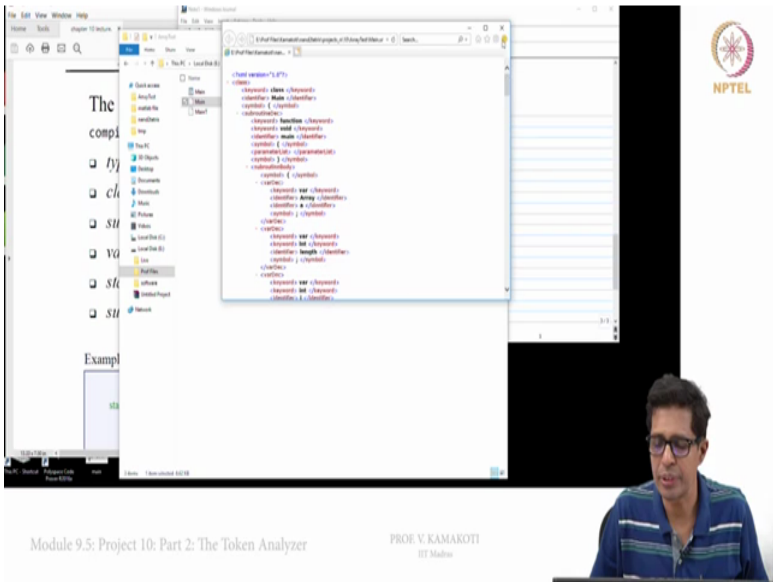
So for all so for some of so these are all the routines that we need to do as a part of this project and so compile class, compile class var dec so these are all part of the slide for chapter 10, chapter in the website you can just refer that compile class var dec, compile subroutine dec, compile parameter list, compile subroutine body sorry and then compile subroutine body, compile var dec, compile statements, compile let, compile if, compile while, compile do and compile return, compile expression term and expression list.

So for the three entities that we saw on the grammar that we saw in the here, yeah so for this entity, this entity, this entity, for these three entities these are the subroutines that we have been writing other rules we just express it directly, so the these are the subroutines for the first second section, the first section was lexical elements, second section these are the subroutines for the third section and these are the subroutines for the fourth section.

So this is how we write the program and every time I call a subroutine before calling the subroutine put the name of that subroutine minus that call print or compile right, that name of the subroutine printed within less than greater than and after you finish executing the subroutine again print slash of that so what we described here and that is the file that we need to basically generate, right.

So once this is generated this code basically we can compile this code, so and so that will give us another XML file of this form, so which I can show you here this is the XML file.

(Refer Slide Time: 33:21)



```
<?xml version="1.0"?>
- <class>
    <keyword> class </keyword>
    <identifier> Main </identifier>
    <symbol> { </symbol>
  - <subroutineDec>
      <keyword> function </keyword>
      <keyword> void </keyword>
      <identifier> main </identifier>
      <symbol> ( </symbol>
      <parameterList> </parameterList>
      <symbol> ) </symbol>
    - <subroutineBody>
        <symbol> { </symbol>
      - <varDec>
          <keyword> var </keyword>
          <identifier> Array </identifier>
          <identifier> a </identifier>
          <symbol> ; </symbol>
        </varDec>
      - <varDec>
          <keyword> var </keyword>
          <keyword> int </keyword>
          <identifier> length </identifier>
          <symbol> ; </symbol>
        </varDec>
      - <varDec>
          <keyword> var </keyword>
          <keyword> int </keyword>
          <identifier> i </identifier>
```

So we can open the this XML file in your browser then it makes actually I am also opening it with internet explorer, right so we can open this, so we kept in the rename this so yeah one of the difficulties with internet explorer you have to I have just named it as analyzed file ans it have to rename it as XML file, so then it only work, so let us go back to this then project so this is the XML file that you will get here, right and this is the output that we need to get out of this course, right.

So we will I will give you the actual implementation details of project 10 in module 9 point 6 but you can start coding with these inputs, thank you.