

Foundations To Computer Systems Design
Professor V Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 10.1

The Jack Compiler – Back-end Introduction

So welcome to module 10.1 and we are now starting the construction of the compiler, the back end of the compiler, so what we did in module 9 and also project 10 was that we did the compiler, the front end of the compiler wherein we took a program, we converted the parts of that program, we actually split the part of the program or we bisected the part of the program into tokens and then we did an analysis of this token to basically construct another XML type of file in which we checked whether the tokens adhere to the syntax of the Jack grammar.

So now that we now have an analysis of that token which we can call as an intermediate representation, we will use that analyzed file to basically construct the compiled code which will be the VM file, there is a virtual, which will be in the language of the virtual machine so we had a Jack file, we tokenized it, split it into tokens then we analyzed it, form that XML intermediate representation.

Now what we are going to do the last part is take that XML version and convert it into an virtual machine code, remember earlier we have written an interpreter which will take the virtual machine code and then basically give you the assembly code, we have already written an assembler which will essentially give you the machine language and we have already developed a simulated hardware on which you can execute this machine language statements.

So this will complete the entire stack so we are in the last part of this so the module 10, I will teach you the theory of how to construct this compiler, module 11 and 12 will go deep dive and construct the compiler at the end of module 12 you will have a compiler which you can basically use to compile the code into a virtual machine code and then you use the softwares that you have generated in the past to interpret that virtual machine to an assembly, again use your assembler and make it into that hack language, machine instruction and then that machine instruction you give it to your constructed hardware and run your program and see it is working correctly.

So we are very close to whatever we termed as the objective of this course and I hope you have completed your project 10, if you have not please put that effort and do that project 10 so that

you know project 11 which we are going to do after this particular module, it is very crucial that you have understood project 10, worked on that project 10 and you have the output of the project 10 right.

(Refer Slide Time: 03:16)

From Nand to Tetris
Building a Modern Computer from First Principles

Chapter 11

Compiler II: Code Generation

These slides support chapter 11 of the book
The Elements of Computing Systems
By Noam Nisan and Shimon Schocken
MIT Press

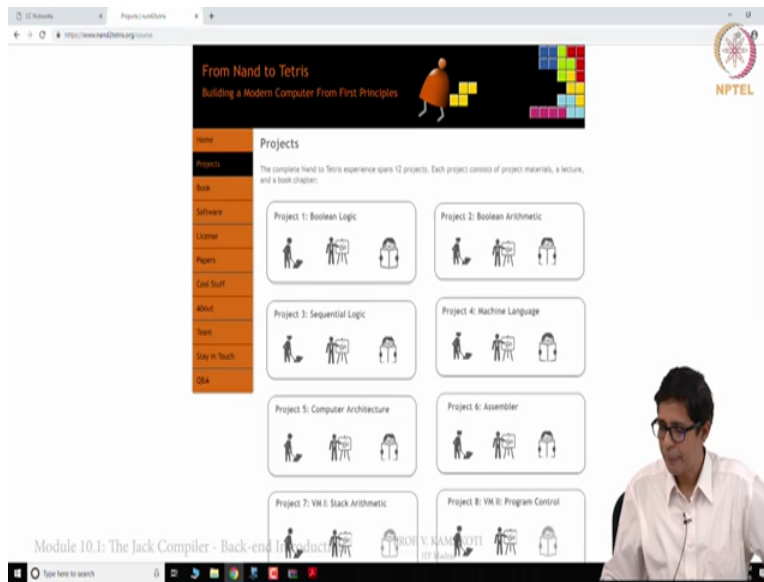
Module 10.1: The Jack Compiler - Back-end Introduction
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTI
2018

NPTEL

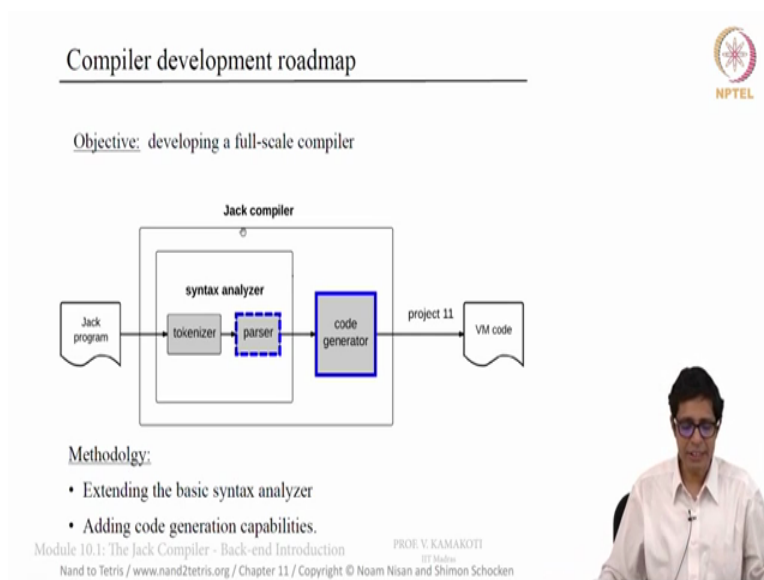
So let us start with project 11, the module 10 right here so we know start with the back end or the code generation, now I am using the slides that are part of the website nand to tetris, so in the nand tetris website, go to projects and there you can download this particular slide, just for your convenience I will also show how to do that.

(Refer Slide Time: 03:47)



You can go to go to projects if you click on projects you get this and go to this project 11 and click on this you know teacher board icon and you will get this chapter 11 like that you could have got for all the other projects also, you can use it this as a guiding material for your projects in general as I mentioned earlier, now let us go and I am going to use this slide so you can also download the slides and keep it for your reference right.

(Refer Slide Time: 04:30)



So yes so we have finished the tokenizer, we have finished the parser as a part of your previous assignment now what we are going to discuss now is a code generator which will take the output

of the parser which is the XML representation and give you a virtual machine code right so this is what we are going to do and once we complete this the token I said plus parser plus code generator together forms a full-scale compiler for the Jack language and so you would have written a complete compiler taking a grammar in mind and you have constructed a compiler and that I believe is going to be a very very interesting value addition to your knowledge reporter.

(Refer Slide Time: 05:18)

The slide is titled "Program compilation" and features the NPTEL logo in the top right corner. It is divided into several sections:

- source code:** A code editor window showing the following code:

```
class Main {
  function void main() {
    var Point p1, p2, p3;
    let p1 = Point.new(1,2);
    let p2 = Point.new(3,4);
    let p3 = p1.plus(p2);
    do p3.print();
    // should print (4,6)
    do Output.println();
    do Output.println(p1.distance(p3));
    // should print 5
    return;
  }
}
```
- Compilation:** A monitor icon displays the output: `(4,6)` followed by `5` on a new line.
- Each class is compiled separately:** A text box with a small circle icon below it.
- Point class code:** A separate code editor window showing:


```
/** Represents a Point. */
class Point {
  field int x, y;
  static int pointCount;
  /** Constructs a new point */
  constructor Point new(int ax,
                       int ay) {
    let x = ax;
    let y = ay;
    let pointCount =
      pointCount + 1;
    return this;
  }
}
```
- Footer:** "Module 10.1: The ~~Jack~~ **more Point methods** Introduction" and "PROF. V. KAMAROTTI" with a small portrait of a man in a white shirt and glasses.

Now let us just quickly go through what is the Jack language, so Jack language any application you are developing on the Jack language will be a collection of Jack files, each Jack files will have a class, the class there is a name for the class for example there is a, this main is the name of this class, similarly point is the name of this class so each Jack file will have exactly one class and the name of that Jack file will be main dot Jack, here it will be point dot Jack.

So you will have several Jack files and you just compile all of them together link them together and you construct an application which you can run right so this is this is basically how an application is there.

(Refer Slide Time: 06:05)

Program compilation



```
class Point {
  field int x, y;
  static int pointCount;

  constructor Point new(int ax, int ay) {}
  method int getX() {}
  method int getY() {}
  function int getPointCount() {}
  method Point plus(Point other) {}
  method int distance(Point other) {
    var int dx, dy;
    let dx = x - other.getX();
    let dy = y - other.getY();
    return Math.sqrt((dx*dx) + (dy*dy));
  }
  method void print() {}
}
```


class declaration

subroutines:

- o constructors
- o methods
- o functions

Module 10.1: The Jack Compiler - Back-end Introduction
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTTI
11/16/2016



Now within every class there are two parts to that class, the first part what we may call as you see here is what we call as the class declaration so what will be there in the class declaration, the name of the class and also the variables that are part of this class, there are two types of variables or two kinds of variables I should say the two kinds are, one are field variable another is called static variable.

The field variable and then each of that field variable will have a type so therefore every variable there is a kind which is field or static, there is a type which is int, in this case it is int, so X is a variable, it is a class variable with its type as int and its kind as field, similarly point count is a class variable which is type as int and its kind as static okay, now we have already seen the difference between a field variable and a static variable so this is a class point right now we instantiate this point in again and again.

So you can say point A, B, C, essentially if I say point A, B, C then I have derived or instantiated three objects of the class point, object A, object B, object C, all the three are points, so there will be a field variable associated with each of these objects, for example there will be A dot X, A dot Y, B dot X, B dot Y, C dot X, C dot Y, but in the case of static there will be only one instance of the static which will be covering all these points right so then there will not be A dot point count, B dot point count, C dot point count, there will be only one point dot point count right.

So there will be only one static variable for all the instantiations of this particular class right so why do we need point count, for example I would like to know how many times my point has been instantiated, so every time I am instantiate a new point like A, B, C, I will keep incrementing this point count so that you know ultimately I will know how many points have been instantiated using this class, so that is an use of this static variable right.

So the static variable is common to all the instantiation or objects of this class while a field variable corresponds to every instantiation there is a field variable that is created, now when you create a class what it means to create a class I need to allocate memory for the field variables in that class right so if I say point A, immediately for A, there will be two memory locations created one to store X and one to store Y.

Similarly if I say point B, for B again two memory locations will be created, allocated by you know you use the memory dot (())(09:18) we will see how to create that but you will create two locations, one for storing the X and Y for B, but the moment I define point, there will be one location that is created in the static segment so please go back, if you go back to the virtual machine there are different segments like you know this, that, argument, local etc. so just revise that, you had one segment called static segment, in the static segment there will be one location created for point count which will be shared by all the three right so that is what.

Then you will have, so this is the class declaration part of your compilation and then the class will have lot of subroutines and the subroutines are of three types constructor, constructor will be constructing an object of a given class, so I will say point A that I should new now A dot new, when I say a dot new then a new object of the type point is created which is now named A right so the construct are basically constructs.

We just told that we need to allocate two memory locations for A right if you define A as a point now these two memory locations are basically created by whom, the constructor right, so the constructor basically constructs an object right and then there are methods and then there are functions okay so the method is basically does not have a return value sorry the method basically is something that it will, it has a return value sorry and it may or may not have a return value and methods basically allows you to change the content of you know, allow you to do some computations within the class.

Functions are those, so the method can basically go and change the variables in the class right, your X and Y can also change here so methods are ways by which I can go and you know change the content of your field variables etc. While a functions are those which will do some computations on your field variables and give you some answers right so these are the difference so there are constructors, methods and functions, so when you look at a class there are two aspects, one is the class declaration which will have the name of the class and the set of field and static variables, then you have subroutines, the subroutines are of three types constructors, methods and functions right.

So when you compile, we need to do something related to class declaration then we need something related to compiling the subroutines, so the compiler essentially is now in, can be viewed into two parts right.

(Refer Slide Time: 12:18)

The slide is titled "Program compilation" and features the NPTEL logo in the top right corner. On the left, a code editor window displays the following Java code for a `Point` class:

```
class Point {
    field int x, y;
    static int pointCount;

    constructor Point new(int ax, int ay) {}

    method int getX() {}
    method int getY() {}

    function int getPointCount() {}

    method Point plus(Point other) {}

    method int distance(Point other) {
        var int dx, dy;
        let dx = x - other.getX();
        let dy = y - other.getY();
        return Math.sqrt((dx*dx) + (dy*dy));
    }

    method void print() {}
}
```

Two blue rectangles are drawn on the code: one around the class declaration (lines 1-3) and another around the `distance` method (lines 11-16). To the right of the code, under the heading "Compilation", there is a bulleted list:

- class-level code
- subroutine-level code
 - constructors
 - methods
 - functions

At the bottom of the slide, there is a small video inset of a man speaking. The footer text reads: "Module 10.1: The Jack Compiler - Back-end Introduction" and "PROF. V. KAMAKOTI IIT Madras".

So this is what, whatever you are marked on the top blue rectangle this code is basically analyzed for the class level compilation and then for every subroutine that you are saying like constructor, method, method for example we will take one method in distance here right, so these individual rectangles for corresponding to each method is taken for every subroutine level compilation right so we have a class level compilation, we have a subroutine level compilation right.

(Refer Slide Time: 12:55)



Compilation challenges

- Handling variables
- Handling expressions
- Handling flow of control
- Handling objects
- Handling arrays

The challenge: expressing the above semantics in the VM language.

Module 10.1: The Jack Compiler - Back-end Introduction
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTI
BY Madhu



Program compilation

```
class Point {
  field int x, y;
  static int pointCount;

  constructor Point new(int ax, int ay) {}

  method int getx() {}
  method int gety() {}

  function int getPointCount() {}
  method Point plus(Point other) {}

  method int distance(Point other) {
    var int dx, dy;
    let dx = x - other.getx();
    let dy = y - other.gety();
    return Math.sqrt((dx*dx) + (dy*dy));
  }



  method void print() {}
}
```

Compilation

- class-level code
- subroutine-level code
 - constructors
 - methods
 - functions

Module 10.1: The Jack Compiler - Back-end Introduction
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTI
BY Madhu



So now what are the challenges now when we start doing the class level compilation we now have variables, we need to handle those variables, we have field variable, static variable etc. The moment I start compiling the subroutines again there are two types of variables in the subroutines, namely the arguments and the local variables inside the subroutine, for example if you take in distance point other, so other in this case other is an argument variable, while your var int dx, dy, these are local variables.

So handling the class variables namely the static and the field variables and handling the variables inside each of those sub routines which can be both arguments and also local variables

come under what we call as the handling variable section then we have to handle expressions, we have to handle the flow of control basically a while statement, do statement etc. then we have to handle objects, creation of objects and disposing of objects and then of course we have arrays right arrays wherein string is actually an array so how do we handle arrays so.

And handling means what, there is when we write a jack statement which has an expression, which has variables or when we derive a Jack object or when we handle a jack array there is a semantics associated with it, there is a meaning associated with it right and what you mean to compile, we need to give another virtual machine language code which essentially has the same meaning okay, so in some sense the challenge here is to express each of these cases, the express the semantics associated with each of these cases in the same VM language right.

So I need to get a VM language equivalent which will express the same semantics and that is going to be the biggest challenge in this whole affair okay so we will now take each one of these challenges and handle them in each one of the modules 10.2 to 10.6, thank you.