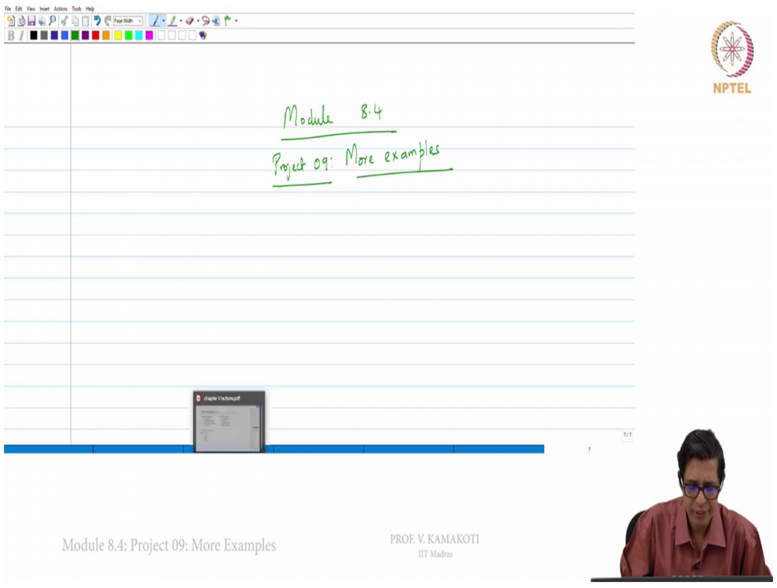
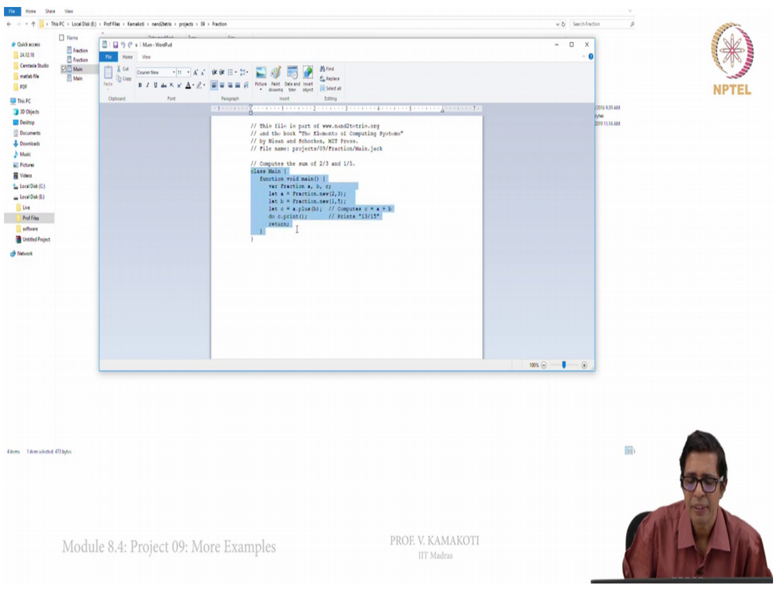
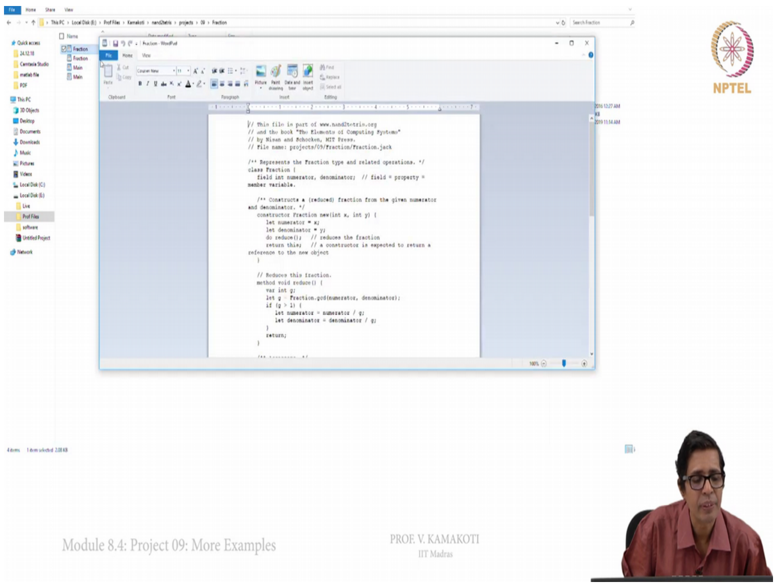
Foundations to Computer Systems Design
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 8.4
Project 09: More Examples

(Refer Slide Time: 0:17)

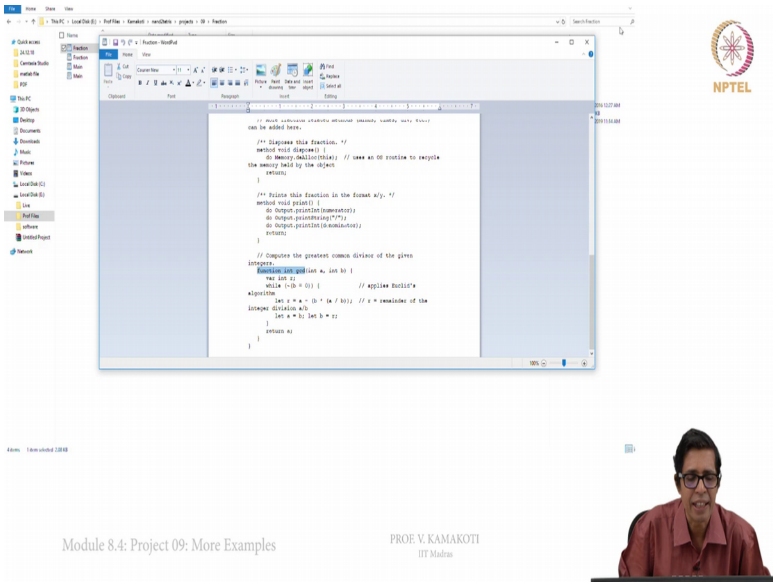

Module 8.4
Project 09: More examples
NPTEL
Module 8.4: Project 09: More Examples
PROF. V. KAMAKOTI
IIT Madras



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Fraction/Main.jack

// Computes the sum of 2/3 and 1/5.
class Main {
   function void main() {
      var Fraction a, b, c;
      let a = Fraction.new(2,3);
      let b = Fraction.new(1,5);
      let c = a.plus(b);  // Computes c = a + b
      do c.print();       // Prints "13/15"
      return;
   }
}
```



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Fraction/Fraction.jack

/** Represents the Fraction type and related operations. */
class Fraction {
   field int numerator, denominator;  // field = property =
member variable.

   /** Constructs a (reduced) fraction from the given numerator
and denominator. */
   constructor Fraction new(int x, int y) {
      let numerator = x;
      let denominator = y;
      do reduce();   // reduces the fraction
      return this;   // a constructor is expected to return a
reference to the new object
   }

   // Reduces this fraction.
   method void reduce() {
      var int g;
      let g = Fraction.gcd(numerator, denominator);
      if (g > 1) {
         let numerator = numerator / g;
         let denominator = denominator / g;
      }
      return;
   }
```

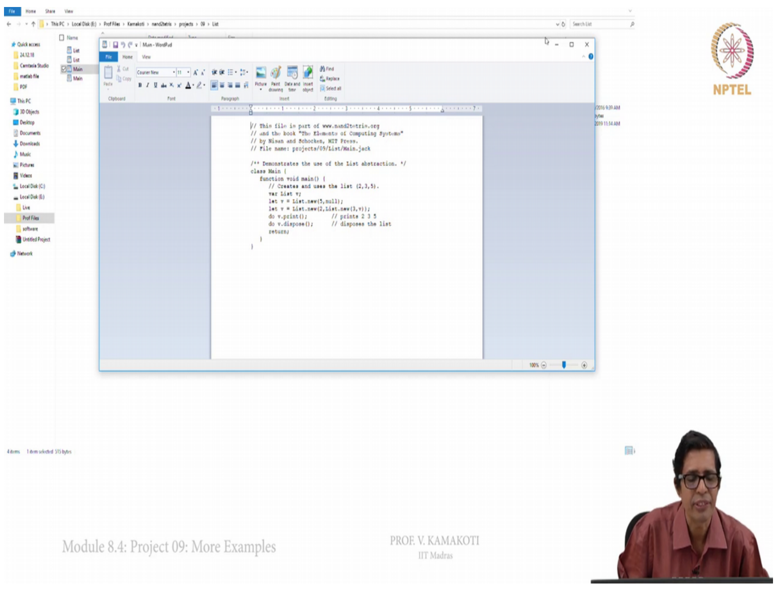


```
can be added here.

   /** Disposes this fraction. */
   method void dispose() {
      do Memory.deAlloc(this);  // uses an OS routine to recycle
the memory held by the object
      return;
   }

   /** Prints this fraction in the format x/y. */
   method void print() {
      do Output.printInt(numerator);
      do Output.printString("/");
      do Output.printInt(denominator);
      return;
   }

   // Computes the greatest common divisor of the given
integers.
   function int gcd(int a, int b) {
      var int r;
      while (~(b = 0)) {             // applies Euclid's
algorithm
         let r = a - (b * (a / b));  // r = remainder of the
integer division a/b
         let a = b; let b = r;
      }
      return a;
   }
}
```

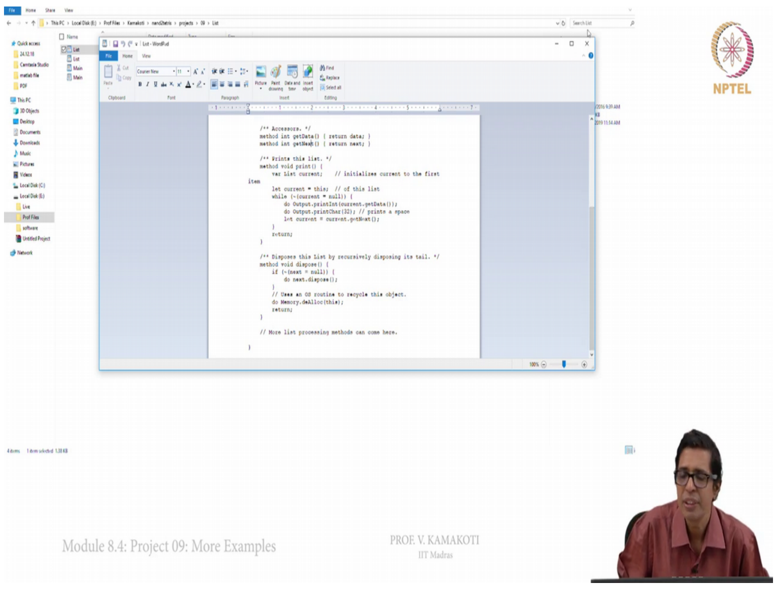
So welcome to module 8.4, in this module we will see project 09 with more examples of project 09, so as we have seen now as we go into this project 09, we have seen the notion of average hello world and fraction, hello world and square, now let us see this fraction code, again as you see there is a main.jack file and main.fraction, so in the main there is a class main which uses fraction right and it has to write and there is a class main which uses fraction.new and fraction and then there is a class fraction which has all the things that has been implemented at we have seen here.

So it has a new, a construction, it has a reduce and then it has get numerator, get denominator a function fraction plus, dispose of this, and then it also has printing the fraction, these are all the methods and of course as I told you that there is a function also internally which will do this GCD, the greatest common devisor computation.

(Refer Slide Time: 1:42)



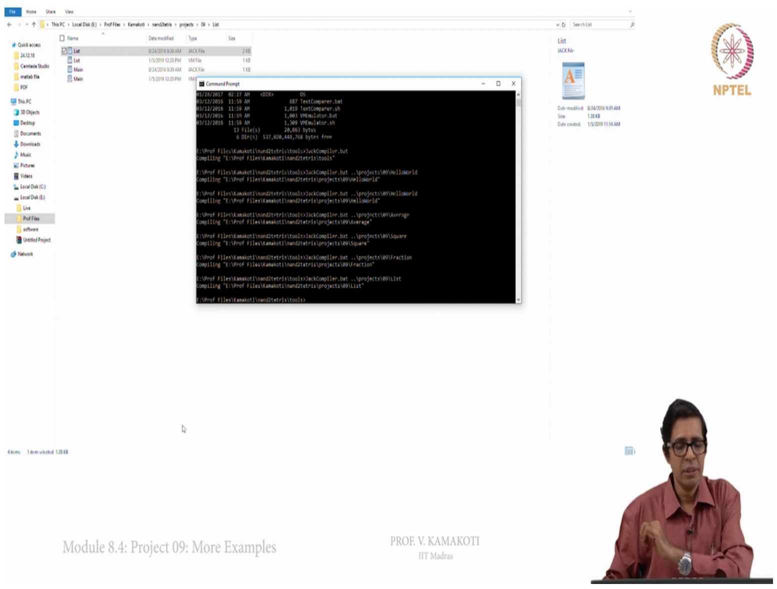
```
// This file in part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/List/Main.jack

/** Demonstrates the use of the List abstraction. */
class Main {
   function void main() {
      // Creates and uses the list (2,3,5).
      var List v;
      let v = List.new(5,null);
      let v = List.new(2,List.new(3,v));
      do v.print();        // prints 2 3 5
      do v.dispose();      // disposes the list
      return;
   }
}
```



```
    /** Accessors. */
    method int getData() { return data; }
    method int getNext() { return next; }

    /** Prints this list. */
    method void print() {
        var List current;    // initializes current to the first
item
        let current = this;  // of this list
        while (~(current = null)) {
            do Output.printInt(current.getData());
            do Output.printChar(32); // prints a space
            let current = current.getNext();
        }
        return;
    }

    /** Disposes this List by recursively disposing its tail. */
    method void dispose() {
        if (~(next = null)) {
            do next.dispose();
        }
        // Uses an OS routine to recycle this object.
        do Memory.deAlloc(this);
        return;
    }

    // More list processing methods can come here.

}
```

So, similarly like the fraction we also have the list, list also has two check files here, the main file which will create the list, 2, 3, 5, it will print the list and also dispose of the list right and then of course there is a list class which has the self-referential structure as you see here, field list next and there is a constructor for that list, there is get data, get next for every element and then printing that list and then disposing the list, all that I have described a part of this.

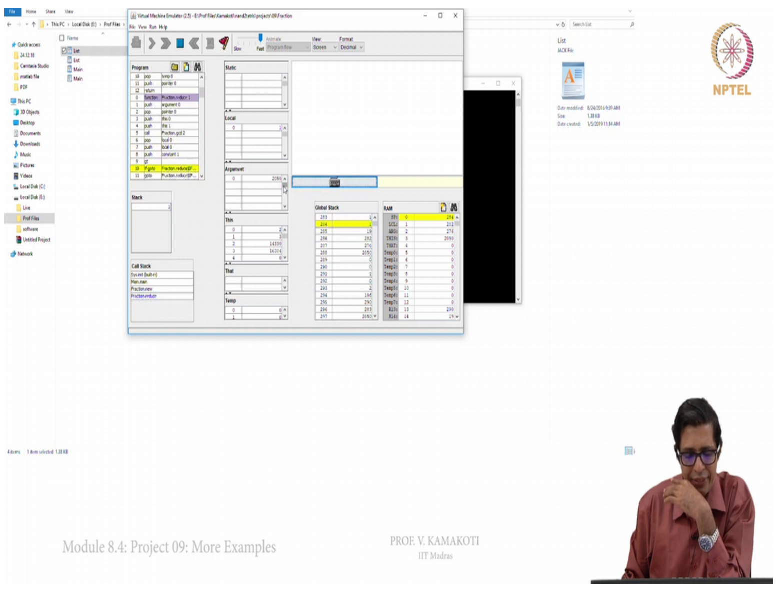
(Refer Slide Time: 2:25)



```
01/28/2017  02:27 AM    <DIR>          OS
03/12/2016  11:59 AM               687 TextComparer.bat
03/12/2016  11:59 AM             1,019 TextComparer.sh
03/12/2016  11:59 AM             1,003 VMEmulator.bat
03/12/2016  11:59 AM             1,309 VMEmulator.sh
              13 File(s)         20,863 bytes
               6 Dir(s)  537,020,448,768 bytes free

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat
Compiling "E:\Prof Files\Kamakoti\nand2tetris\tools"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\HelloWorld
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\HelloWorld"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\HelloWorld
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\HelloWorld"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\Average
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\Average"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\Square
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\Square"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\Fraction
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\Fraction"

E:\Prof Files\Kamakoti\nand2tetris\tools>JackCompiler.bat ..\projects\09\List
Compiling "E:\Prof Files\Kamakoti\nand2tetris\projects\09\List"

E:\Prof Files\Kamakoti\nand2tetris\tools>
```

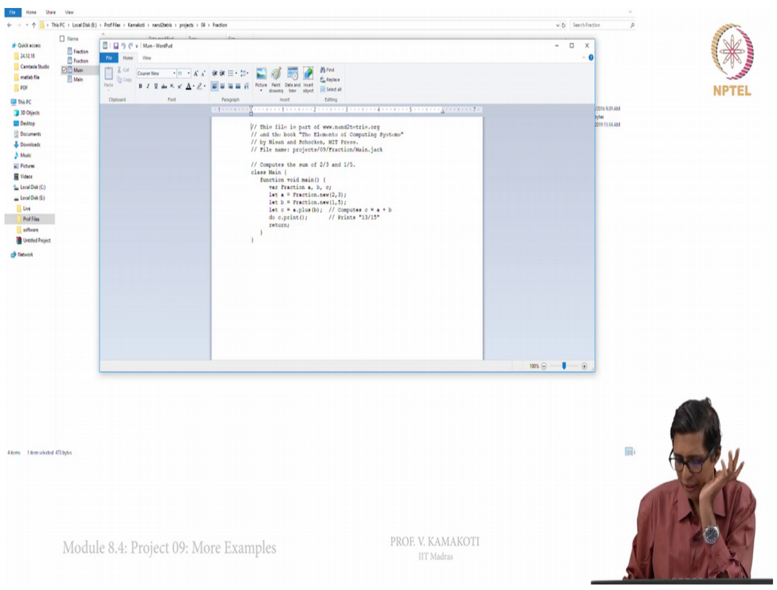
Module 8.4: Project 09: More Examples

PROF. V. KAMAKOTI
IIT Madras



Module 8.4: Project 09: More Examples

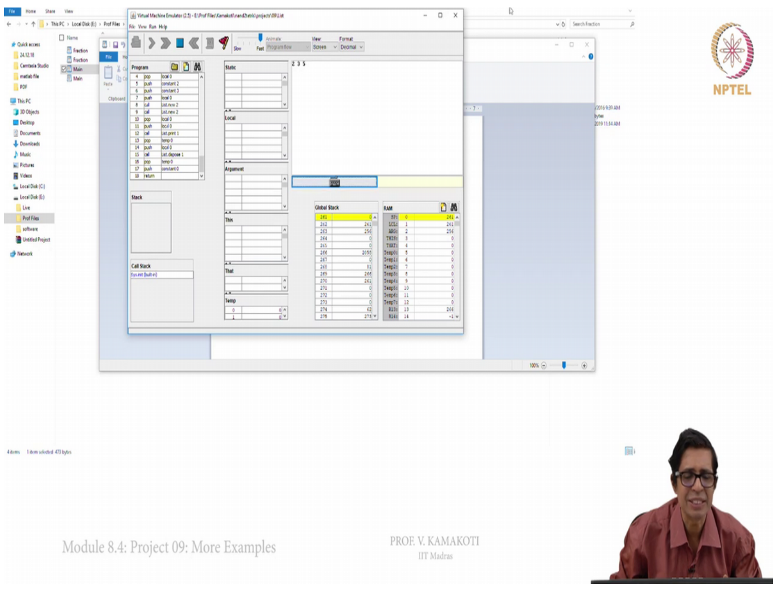
PROF. V. KAMAKOTI
IIT Madras



```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Fraction/Main.jack

// Computes the sum of 2/3 and 1/5.
class Main {
   function void main() {
      var Fraction a, b, c;
      let a = Fraction.new(2,3);
      let b = Fraction.new(1,5);
      let c = a.plus(b);  // Computes c = a + b
      do c.print();       // Prints "13/15"
      return;
   }
}
```

Module 8.4: Project 09: More Examples

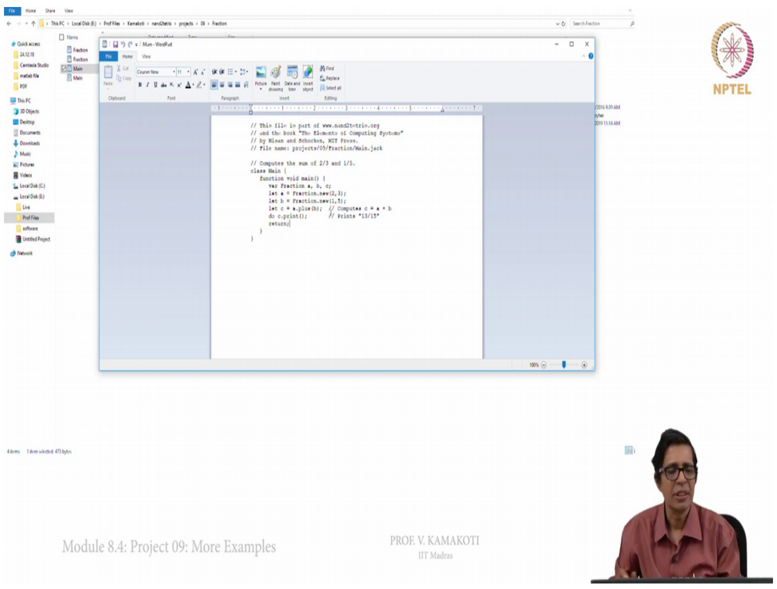
PROF. V. KAMAKOTI
IIT Madras


Virtual Machine Emulator (2.5) - E:\Prof Files\Kamakoti\nand2tetris\projects\09\List

```
4   pop   local 0
5   push  constant 2
6   push  constant 3
7   push  local 0
8   call  List.new 2
9   call  List.new 2
10  pop   local 0
11  push  local 0
12  call  List.print 1
13  pop   temp 0
14  push  local 0
15  call  List.dispose 1
16  pop   temp 0
17  push  constant 0
18  return
```

```
2 3 5
```

Call Stack: Sys.init (built-in)


```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/09/Fraction/Main.jack

// Computes the sum of 2/3 and 1/5.
class Main {
   function void main() {
      var Fraction a, b, c;
      let a = Fraction.new(2,3);
      let b = Fraction.new(1,5);
      let c = a.plus(b);  // Computes c = a + b
      do c.print();       // Prints "13/15"
      return;
   }
}
```

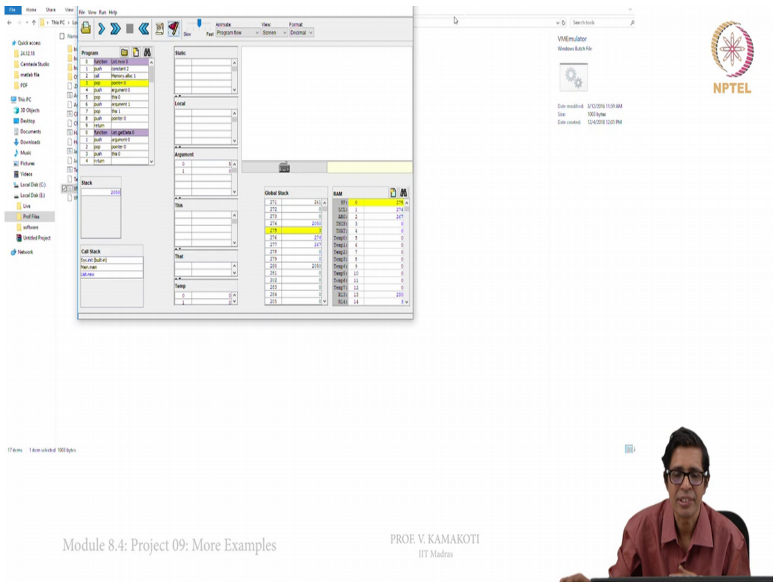
Now let us quickly go and compile and execute this things, so we can just use go back to your jack prompt, project 09/fraction, you can also compile project 09/list, than we can go to the VM emulator, we can go and load the program, so this is still running, you can load the program here, so we can first load the fraction, just go to the directory, top of the directory we need not go inside the directory and say load programs, so all the press S for this because we are using some of the OS functions which we have not implemented.

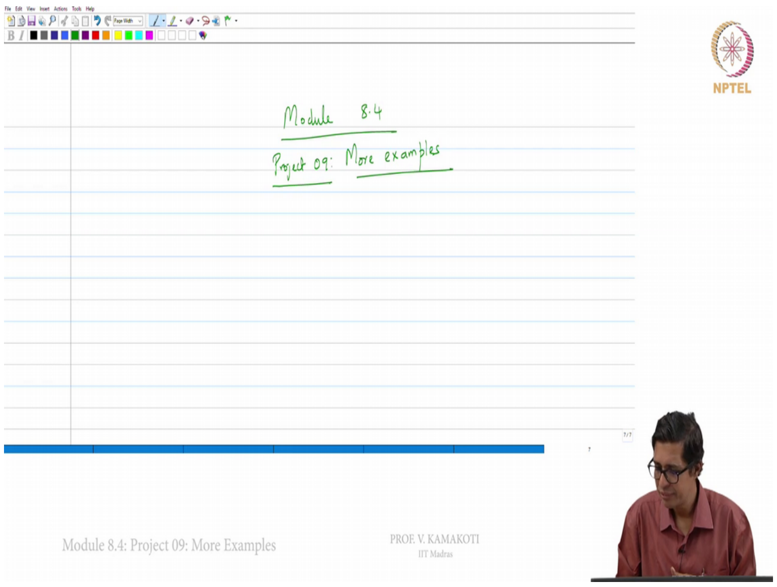
So this is a fraction, so fraction.new, fraction.reduce so many functions, all the functions get loaded right and now we just go and execute this function on the VM later, so the main gets executed, now creating two fractions as you see here, 2, 3 are arguments so that function, so many things, so your GCD is working on this, so as this is working let us just recap what the fraction we are trying to do 2 x 3+ 1 x 5 which would give you 13 x 15, so let us see what it is, still working yes, 13 x 15 got printed here, similarly we can also load the program corresponding to your list, go program and yes and then run it complete.

So in case we start writing more and more programs the way we very whether it is working correctly or not, even this is the, you can compile it using the compiler and execute it on the VM emulator and that will basically tell you, it is a debugging tool for us, the tool is working, the code is working correctly or not right, so this is a complete debugging tool, so see it is printed the list, 2, 3, 5 here and now it is currently disposing it off.

So you can go through this slowly emulation and understand how the thing is working right, so now you understood the concept, so why are we doing this exercise, there is not much that we are programming, we are just executing and seeing on the VM emulation but this will basically tell the interface between the application programming language and the operating system right, that is the most important thing, when I say in let us take this fraction as you are saying here, fraction.new, what happens when fraction.new is executed code?

(Refer Slide Time: 6:05)




Module 8.4

Project 09: More examples

Module 8.4: Project 09: More Examples

PROF. V. KAMAKOTI
IIT Madras

So when you actually go to this VM machine as you see here, when we actually go to this VM machine right and when we load this file say load program, so we go and say list, we load the entire list program here right, we have loaded, now we can see that here we know all the functions list.new, list.get, data list.get next, list.print and then list.dispose, main.main etc, so we know all these programs now, so the moment we start executing right, so we have come to this main.main right, now we say every statement.

So list.new now gots, and now what is happening list.new? So I am creating a new list with you know 5, null or whatever, so what are the arguments going 5 and null are going as arguments and what is happening here? So how is this executing, so then you basically come out and find out that, that there is a memory.alloc that is happening here, so what does memory.dalock actually do right, it is creating something, so you can see the stack, you can see all these things whatever I have described their.

So when you carefully look at this code one by one, if you start looking at this code detail, you will understand the exact interface between the application programming language and the operating system right, so when the application programming language ask for certain facility, what does the operating system to on the where to provide you that facility, like memory allocation, printing, memory deallocation all this things and that is something that we achieve as part of this project 09 that understanding is very crucial right and whenever use that doing your programming later, you know there is a big failed which gives you a lot of you know job opportunities, it will give you a lot of job opportunities in the few years to come, when you are in your prime of the carrier that is information security.

When you really want to write a secure code, you need to understand the interface between the programming language and the operating system, if you do not understand that in clear detail you cannot write a real secure code, this exercise that we do now is a very important foundation for you to do secure coding at some later point of time and that is why we are insisting on this project 09, you are not writing one line of code as part of project 09 but you execute this code on the VM emulator and find out how the operating system and the programming language interfaces right.

So spend some time executing it step-by-step as we see here next step, next step, next step and see what is happening on these stack, understand, one or two programs like your fraction and this list, if you understand this two programs and also there is some of the previous programs I think that itself will give you not of clue of how things work. Okay, so with this, we end up the module 8.4 and we will see or in the module 8.5. Thank you.