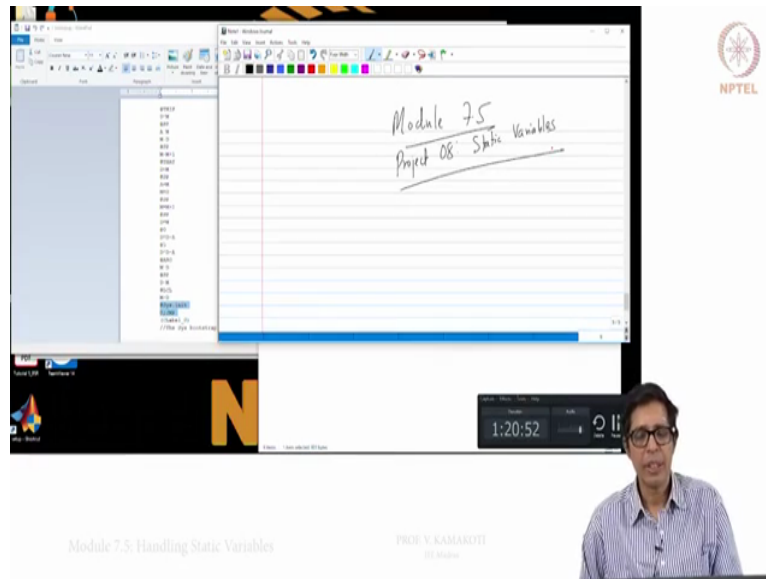


Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 7.5
Handling Static Variables

(Refer Slide Time: 0:16)



The screenshot shows a video lecture interface. On the left, there is a code editor window displaying a list of files. On the right, there is a whiteboard with handwritten text that reads "Module 7.5" and "Project 08: Static Variables". Below the whiteboard, there is a small video window showing a man with glasses and a blue shirt. In the bottom right corner, there is a digital clock showing "1:20:52". The NPTEL logo is visible in the top right corner. At the bottom of the slide, there is text that reads "Module 7.5: Handling Static Variables" and "PROF. V. KAMAKOTI, IIT Madras".

Welcome to module 7.5 and in this we will talk about how to handle static variables. Now what are the static variables? Static variables as we had understood in C and probably in other languages also or those even they can be part of a function, they can also be global variables, after the completion of the function what we saw with respect to the local variables that once you finish execution of the function the entire stack frame actually gets away and so you do not have access to those variables after that it is meaningless.

But the static variables are those whose values are remembered through the entire lifetime of the program, right.

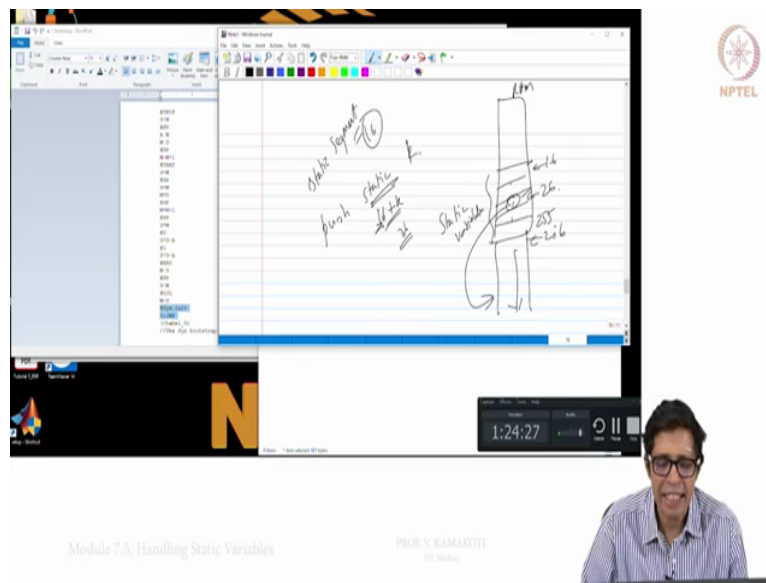
(Refer Slide Time: 1:08)

The screenshot shows a video lecture interface. On the left, there is a list of memory addresses from 0000 to 000F. The main area is a whiteboard with a hand-drawn diagram of memory segments. The diagram shows a vertical stack of memory locations. The top part is labeled 'Stack Segment' and contains the text 'push static k'. Below this, there is a section labeled 'Static Variables' with a diagram showing a stack of memory locations starting from 16 and ending at 255. The diagram also shows a stack of memory locations starting from 256 and going to a large extent. The presenter, Prof. V. Kamargiri, is visible in the bottom right corner of the video frame. The NPTEL logo is in the top right corner. The video title 'Module 7.3: Handling Static Variables' and the presenter's name 'PROF. V. KAMARGIRI' are visible at the bottom of the video frame.

So as far as we are concerned the static variables are stored from the location 16 of your RAM till 255, 16 to 255 are the places where the static variables are stored, after that your stack starts from 256 and goes to a large extent, right. Now the local variables are part of the stack and they get $(())(1:31)$ as the function goes, but the static variables since they have to be there it should be between 16 to 255. So that is how we have defined the segment called a static segment starting from 16, right.

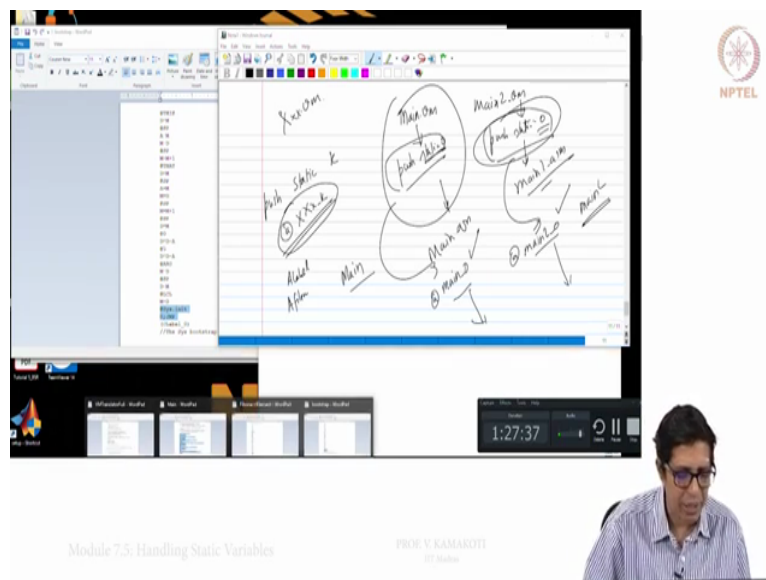
Now what we had done is that as a part of our previous project 7, when we just say push static 0, so we go to the 16th location per static k or whatever index, we go to the 16th location suppose 16th location we go to the 16th location add 16 plus k so let us see this is 10, so now go to the 26th location so somewhere here 26, whatever is content of the 26th location that you push it into the stack, right so this is how we have been handling this.

(Refer Slide Time: 2:40)



Module 7.5: Handling Static Variables

PROF. V. KAMAROTTI
IIT Madras



Module 7.5: Handling Static Variables

PROF. V. KAMAROTTI
IIT Madras

Now we have reached a stage where this will not work, the simple reason is I have so last time we had main dot VM then there will be main 2 dot VM let us say, this has a static variable say this has some push static 0, this has another push static 0, this has a static variable, this has another static variable, both will be since both are written separately they will all assume that they will all start with the numbering 0, right.

Now this static variable belonging to main is different from the static variable belonging to main 2, right. So when we integrate when we compile this it is going to give me main dot ASM, when this will also give me main 2 dot ASM, right so this will have one static variable here and this will have another static variable they are not part of the they are not the same, so this static variable is different from this static variable.

So our original technique of saying that if I have a push static k, then take 16 plus k and then do, so both will now if I follow that technique both will map on to the same static variable same memory location which is wrong. So the simple thing that we need to do in this specifically to solve this problem is to make this, so whenever I say when I say push static k of a function name Xxx dot VM, right so you just create a label.

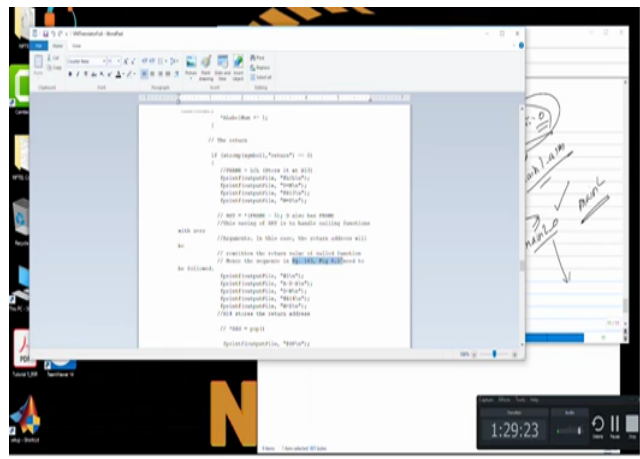
So you just say Xxx underscore k at, right so this will basically this will be a variable symbol. So here you will get for this, you will get at main underscore 0 and for this you will get at main 2 underscore 0. So whenever you see a symbol and it is a static variable the assembler itself will start assigning from 16, 17, etc so it has so a main 0 will be automatically assign by the assembler, main 2 0 will automatically be assign by the assembler in the static segment, right.

So you need not actually do 16 plus 0 just say (())(5:19) 0 or (())(5:20) 2 0 so essentially this will be assigned a different address then this rather all the variables in this main 2 will be assigned a different address from all the variables at main 0 and that will make the life much much simpler, this is one part of the story so this is something that we need to take care in the case of static.

In addition, we have been using lot of labels, right so A label, F label we have been doing lot of labels as a part of our even assigning return address in the case of call, right. Now if we run the same code the labels can get repeated, so wherever we have used label like A label last time we have used A label we should put A some file name label so all the labels in this particular main dot VM will have the word main, all the labels in this particular VM compiling this will have main 2 we will say, so this will distinguish the labels because the label I am using here should be different from the labels I am using there.

So if I execute the same program again and again I may get into some duplication of the labels here and there and that should be avoided. So always (())(6:39) the file name for that label. So labels have to be taken care (())(6:44) and of course the static variables and these two are very very important aspects and modification that we need to do to your assembler translation.

(Refer Slide Time: 7:03)



The screenshot shows a code editor window with the following C code:

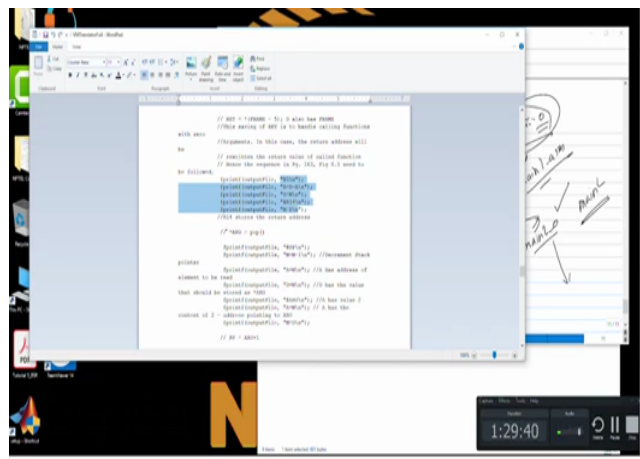

```
int main() {
    // The return
    // If the return type is int
    // Then the return value of the function
    // Should be of type int
    // In this case, the return address will
    // be
    // // Specifies the return value of called function
    // // When the response is 10, the 10 is used as
    // // follows.
    printf("Return value is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    printf("Return address is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    return 10;
}
```

Handwritten notes on the right side of the slide include a diagram with a circle containing '10', an arrow pointing to 'Return', and another arrow pointing to 'Print'.

NPTEL logo is visible in the top right corner.

Module 7.5: Handling Static Variables

PROF. V. KAMAROTTI
IIT Madras



The screenshot shows a code editor window with the following C code:

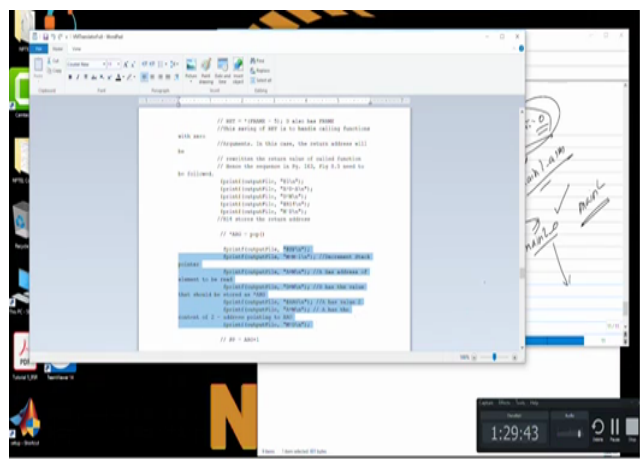

```
int main() {
    // If the return type is int
    // Then the return value of the function
    // Should be of type int
    // In this case, the return address will
    // be
    // // Specifies the return value of called function
    // // When the response is 10, the 10 is used as
    // // follows.
    printf("Return value is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    printf("Return address is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    return 10;
}
```

Handwritten notes on the right side of the slide include a diagram with a circle containing '10', an arrow pointing to 'Return', and another arrow pointing to 'Print'.

NPTEL logo is visible in the top right corner.

Module 7.5: Handling Static Variables

PROF. V. KAMAROTTI
IIT Madras



The screenshot shows a code editor window with the following C code:


```
int main() {
    // If the return type is int
    // Then the return value of the function
    // Should be of type int
    // In this case, the return address will
    // be
    // // Specifies the return value of called function
    // // When the response is 10, the 10 is used as
    // // follows.
    printf("Return value is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    printf("Return address is %d\n", 10);
    // // Prints the return address
    // // when the return address
    // // is 10
    return 10;
}
```

Handwritten notes on the right side of the slide include a diagram with a circle containing '10', an arrow pointing to 'Return', and another arrow pointing to 'Print'.

NPTEL logo is visible in the top right corner.

Module 7.5: Handling Static Variables

PROF. V. KAMAROTTI
IIT Madras



So let me just show you the final assembler that we have done, so there is the so this is assembler, this is the translator VM to the assembly translation we have seen about stack arithmetic etc now let us go back to yeah the function call, the moment I see a function right so I just put the function name because somebody wants to jump here I have to put the function name here as a label and then I push 0 into the stack k times, so that is the thing and then for the written of course I push a return address and so when I am pushing a return address please note that C label and I am also putting a string here the string is nothing but the file name I am appending it here, I am pushing the return address then push LCL, push ARG, push this, push that, ARG, SP and $()$ (8:30) this is for the call.

And for the return this is for the call, this is for the return again frame equal to LCL all that we see here star frame minus $()$ (8:43) so this is all in page 163 figure 8.5 of the book. So we just basically see all your commands being translated into the, so return is equal to star of frame minus 5 so just this is the assembly equivalent of that, star ARG equal to this is the assembly equivalent of that. So like that for every statement we have replaced by that assembly this is what we have done.

(Refer Slide Time: 9:18)

The image shows a video lecture slide. At the top right is the NPTEL logo. The main content is a screenshot of a code editor window displaying assembly code. The code includes comments in C and assembly instructions for pushing constants and setting return values. Below the code editor is a system tray showing the time 1:30:16. In the bottom right corner, there is a small video feed of a man with glasses and a striped shirt. At the bottom of the slide, the text reads "Module 7.5: Handling Static Variables" and "PROF. V. KAMAROTTI, IIT Madras".

The screenshot shows a code editor window with the following C++ code:

```

// This file is part of www.nptel.ac.in
// and is licensed under the Creative Commons Attribution License
// For more information see http://creativecommons.org/licenses/by-nc-sa/4.0/
//
// Shows the applied operations in matrix() and matrix().
//
// Matrix (const int & m)
// print matrix 0
// print matrix 1
// print matrix 2
// print matrix 3
// print matrix 4
// return matrix() + matrix();
// Matrix (const int & m)
// print matrix 0
// print matrix 1
// print matrix 2
// print matrix 3
// print matrix 4
// return

```

The video feed shows the instructor, Prof. V. Kamakoti, at the bottom right. The NPTEL logo is in the top right corner. The slide footer contains the text: "Module 7.5: Handling Static Variables" and "PROF. V. KAMAKOTI IIT Madras". A timer in the bottom right of the video frame shows 1:30:51.

And specifically for this last part of this (9:19) where the static test, in the static test please note that there is a sys and this is it calls class 1 and it also calls class 2 and this class 1 and class 2 are again two other files they have two functions class 1 dot set and class 1 dot get inside them. Similarly, class 2, class 2 dot set and class 2, so there is a static 0 for class 2 this file there is a static 0 and static 1 for this file and class 2 and there is also a static 0 and static 1 for this class 1 and both these static variables are different. When I compile all the three then it may point to the same and that is error we want to reduce, right.

(Refer Slide Time: 10:46)

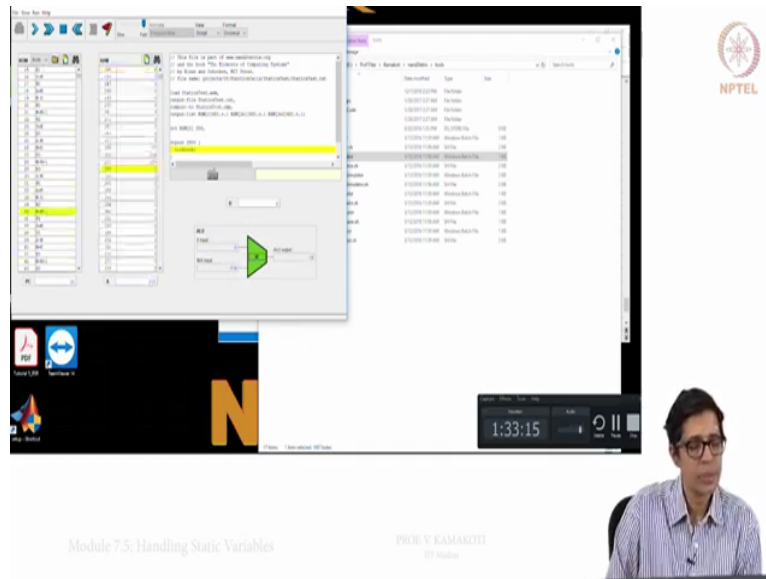
The screenshot shows a code editor window with the following C++ code:

```

// This file is part of www.nptel.ac.in
// and is licensed under the Creative Commons Attribution License
// For more information see http://creativecommons.org/licenses/by-nc-sa/4.0/
//
// Shows the applied operations in matrix() and matrix().
//
// Matrix (const int & m)
// print matrix 0
// print matrix 1
// print matrix 2
// print matrix 3
// print matrix 4
// return matrix() + matrix();
// Matrix (const int & m)
// print matrix 0
// print matrix 1
// print matrix 2
// print matrix 3
// print matrix 4
// return

```

The video feed shows the instructor, Prof. V. Kamakoti, at the bottom right. The NPTEL logo is in the top right corner. The slide footer contains the text: "Module 7.5: Handling Static Variables" and "PROF. V. KAMAKOTI IIT Madras". A timer in the bottom right of the video frame shows 1:32:13.



So what we need to do is now with all the changes that I have been suggested we need to compile this sys dot VM class 1 dot VM and class 2 dot VM put them one after another and also have the bootstrap code at the beginning. So that will form the complete AMS which we call as static test dot ASM (10:44) so this is the ASM so if we just look at this ASM so if we just look at this ASM this is the bootstrap code it starts with the bootstrap code, right and then it has this sys dot init, sys dot VM and then sys dot VM finishes here sys dot VM actually basically has sys dot init and then so this is over and so this is class 1 file, class 1 dot VM similarly you have class 2.

So all the files we have linked together and put it on one single thing and this we can basically run (on your VM) on your CPU emulator now we can load the script and run this and you can make it a fast one also this runs to completion. So these are this is this final full fledge version of the VM emulator is necessary which will basically convert your VM programs into a hack, wherein these static variables and other things are basically taken care of so it has come to an end ultimately the same thing, right. So this is something that will solve the problem of this translation.

So what we have done in this particular till project 8 and so far is that we have taken this virtual machine a stack based virtual machine which had a set of arithmetic operations, it has memory access operations, it has program flow and then these calls the function calls and for each what we do we just take the whole thing and for each statement we go and replace it with a statement by statement we go and replace it with the corresponding assembly code and that is the entire story but the care that we need to take at we need to be we need to really matter you know we need to be extremely careful is that whenever we get some of this labels

and the static variables these are some of the things that we need to handle properly and that will make you know the thing robust and will solve the issue for us.

So take the project 8 now and complete it, I think we have given lot of details which will enable you to work fast on this project and use the VM emulator, use the CPU emulator for all and all the scripts given to verify every time after you make the final version that are so the final version please check all the things like project 7 and project 8, all the codes in project 7 and project 8, test with the final emulator final interpreter that you have written, this is called interpretation because every instruction you take and basically translate it and then put them together and run and see.

So any issues please do login to the website and we will take enough please put on the discussion forum and we will ensure that your questions are answered. So kindly spend lot of time to see that this whole thing is in good shape, all the best.