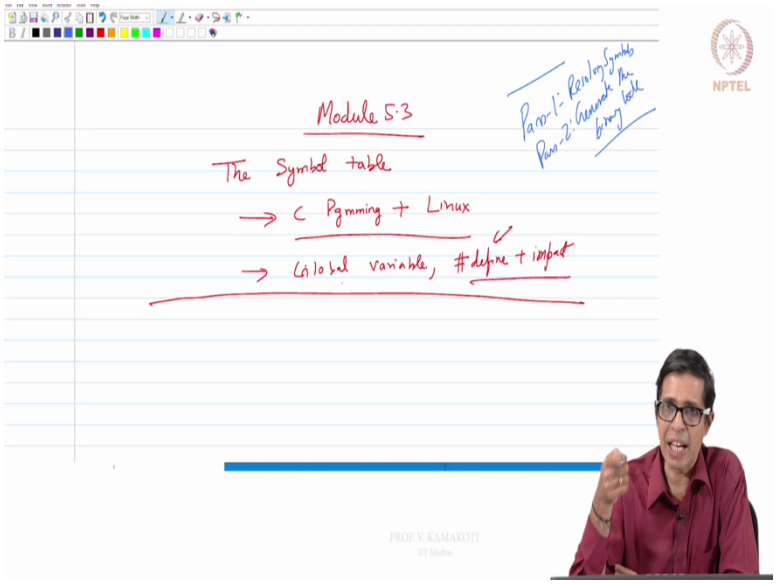
Foundations To Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology Madras
Module 5.3
Assembler: symbol table construction

(Refer Slide Time: 0:20)



So welcome to module 5.3 and in this module we will be talking about implementation of symbol table. So we basically talked about what is symbol table, what is expected out of that symbol table but now we will talk about its implementation essentially we will be using C programming, we can implement the symbol table as mentioned earlier we can implement it using C++ Java or Python or Perl or whatever you want.

But overall you know C is one language that is being thought by many university curriculums and I thought I will do it with C but you can easily modify it to work with other languages too. And importantly from this point of time I would like all of you to implement Linux in your system and then work on the LINUX version of this tool that will make it very very easy and interesting for you to work, right?

So if you want any help on LINUX it's already there quite a lot of websites are there which can help you basically install LINUX if you have you can actually make a partition you can have multiple operating systems with multiple boot in your laptop and basically you can install LINUX to that. Working with LINUX will add a lot more insight into operating systems plus also you will be exposed to large-scale program development, right?

So till now we have worked on hardware now we're going to actually start writing software and that too system software and many a times this is where the challenge actually comes. As I mentioned earlier building and architecture is like building a dam and but we need to fill it with water and the water is the software that we are going to write and it is very very very important that we start writing the software very nicely, right?

So otherwise what would basically happen is if you don't have an sort of understanding of the expectations of software about this layer like what is the expectation of an assembler? What is an expectation of an operating system? The appreciation of the hardware will not be complete, so we need to have this very interest very detail understanding of the expectations of that software.

The system software namely your assembler, your virtual machine, your interpreter and your compiler and how do you understand its requirement actually by writing one, right? If you write this then you basically get lot more insight into what is the expectation of an assembler from an hardware, what is expectation for f interpreter, what is expectation of a compiler et cetera?

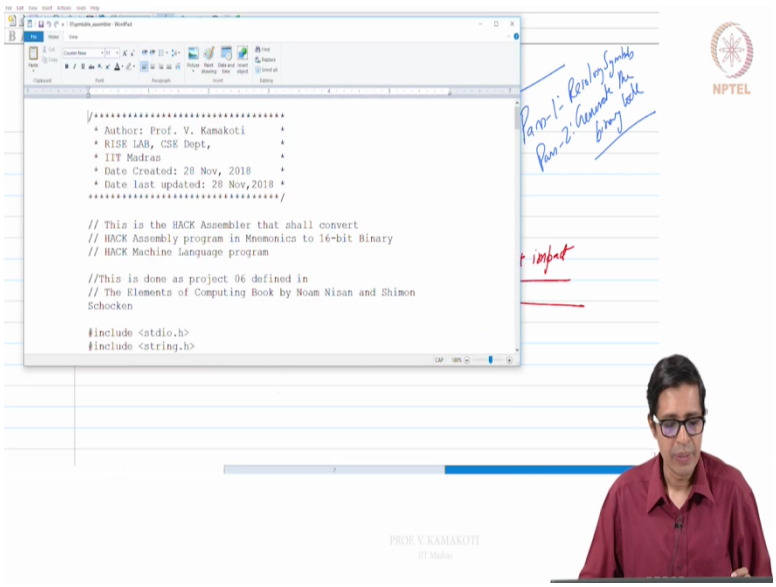
So this is a point where you need to not give up and start working on this and if you actually put some effort you will certainly see this software being generated, so what my request to all the people who are attending this course is that you take this up very seriously and never let down the spirit, you have worked out, you now made an architecture by yourself within just for 2 to 3 weeks and I think this is nice time that you actually write sensible software and see how it is executing on the architecture that you have created.

So with this let me start into the construction of the assembler as I told the assembler is a 2 pass assembler, right? So the pass one is basically used for resolving symbols and the pass 2 is to actually generate the binary code and the most important thing is supposing we are looking at resolving symbols one of the entities that is very much necessary for this is the symbol table.

Now how do we construct the symbol table and what are the things that are expected out of the symbol table, right? And that is what we will see and when we look at the assembler as a program there is a path which will do pass one there is a part that will do pass 2, the pass one and the pass 2 will be basically using the symbol table so this entire symbol table we basically model it as a global variable.

So that now 2 passes can basically share that, right? In general global variable is not a very recommended practice but in a controlled and the environment like this where assembler is going to be used only by your interpreter later, it is a very control and environment we can use this as global variables and see that nothing wrong happens to that, right? So we will now start describing the symbol table and its construction.

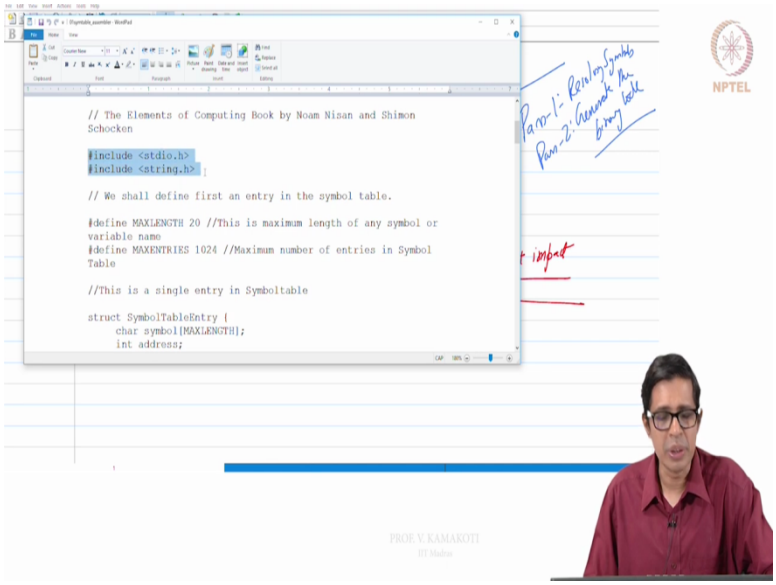
(Refer Slide Time: 5:26)



Now this is a code that I have written and one of the things that I want everyone to see especially undergraduate students is that commenting is very very important. Very rarely you will use the software that you have created; somebody else is going to use that software, so writing comments is very important. So just look at this, so I have just made sort of very specific comments here as I'm moving down.

So I said who is the author, where does the author belong to when he created this file, when he updated it last, so essentially he says he have "done this in a single day", right? And I have told what is this? This is the hack assembler that shall convert hack assembly program in the (())(6:13) etc, right?

(Refer Slide Time: 6:20)



Now these are 2 include files because you're going to use a lot of (())(6:23) and also string, right? Even while creating this symbol table, so you need to do this. Now comes these 2 sorts of what you call as this #define variables that you see here, okay. So we basically have this Max length and Max entries. Max length, so what does a symbol table basically have?

It has the symbol which is a string and then it has the address associated with that symbol, right? Now what you're saying here is the length of the string is restricted to 20, so I could not have a symbol which the character string of more than 20 characters, there are actually 19 characters there is end string also here, right? And then the symbol table cannot have more than 1024 entries.

These are some things that we specify here I can't assume and in finite size symbol table, I can't assume an in finite length symbol like I can't say at a, B, C, D, E, F, G, H, I, J, right? And a long symbol, right? So somewhere I make certain assumptions, so when you develop these system software at lot of places you actually make this type of assumptions, see okay this is the maximum length of the character.

This is the maximum number of entries I could have in the symbol table etc. Now who should know this? The person who is in the software stack who is going to use the assembler, the interpreter of your virtual machine that they're going to see in the next module there is a person who is going to use this particular assembler. And who is going to generate disassembly code?

The interpreter is going to generate the assembly code, so the particular software responsible for generating the assembly code, right? That software or the person who creates that software should be aware that the Max length is 24 for a character and the Max entries are 1024 in the decimal table, so I cannot actually assemble a program which has more than 1024 symbols, right?

That is what this basically says, right? So these types of assumptions have to come out when a system stack is made. This type of assumptions have to come out, so that when the assembler is assembling when the interpreter is generating the assembly program it should know that it cannot have a string of infinite length it cannot have number of entries or symbols of large number of symbols, right?

So these are some very very important important points that we note when we start developing a system software stack, right? Okay, now as we see we need to increase it, so we will be assembling 3 programs the first 4 programs to be as a part of the project which I will be demonstrating in the subsequent modules and the first 3 would be very short programs (()) (9:50) programs.

But the last one the pong is actually a very large program and that when we make Max length as 20 and Max entries as 1024 the assembly fails. It now says I need more number of entries because the number of entries is more than 1500 the number of symbols that are program used is more than 1500 and similarly Max length is something very large, right? There are strings which are more than 20 characters.

So at that point of time the assembler has to tell, okay I am now getting the entries more than 1024 my string length of a symbol is more than 20, the assembler should be in a position to tell, so that people can go and change the assembler and recompile it. So one of the things that is very important when we build a system software is this resilience or diagnostic ability.

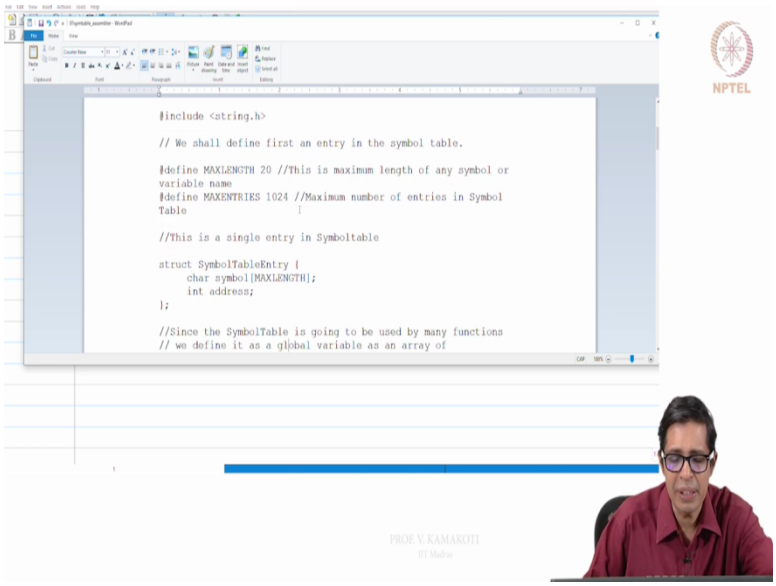
Something is not compiling, something is not assemblable the software the code should come out and tell why it is not assemble able, what is the issue that? So that this should aid in debugging the code and actually fixing the bug. So one of the things that we will see right from this particular module is that when we write system software how are we going to make it.

Give lot of inputs for diagnosing, it is not just diagnosing your thing it also gives you input. Suppose I put are wrong mnemonic, suppose I say some r is equal to f plus t, right? This is

not there in your hack language you have done a lot of assembly now, right? You don't have a statement like r equal to f plus t there is no register call r or f or t. Now the other assembler should say it's an illegal mnemonics, right?

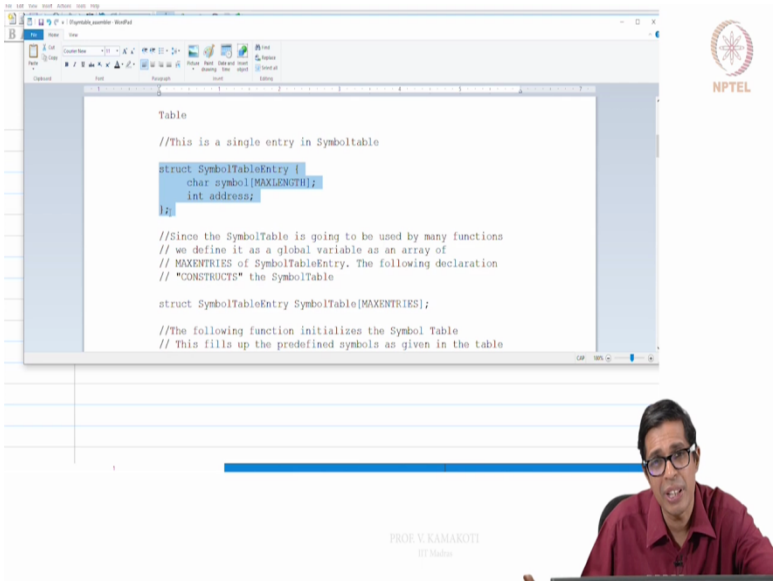
So there and it should also tell you where that thing is right? So what is illegal? Just cannot say illegal mnemonics and keep quite. It should say where is that illegal mnemonics, so that you can go and correct it, alright. So writing code which can help the user diagnose some of the errors that are part of it is also going to be very important and that is also what we are going to see as a part of this development of the system stack.

(Refer Slide Time: 12:22)



Now let us Move on to the next age. Now what is a symbol table? A symbol table is an array of symbol table and each symbol table entry will have symbols of Max length and address which is an integer, right?

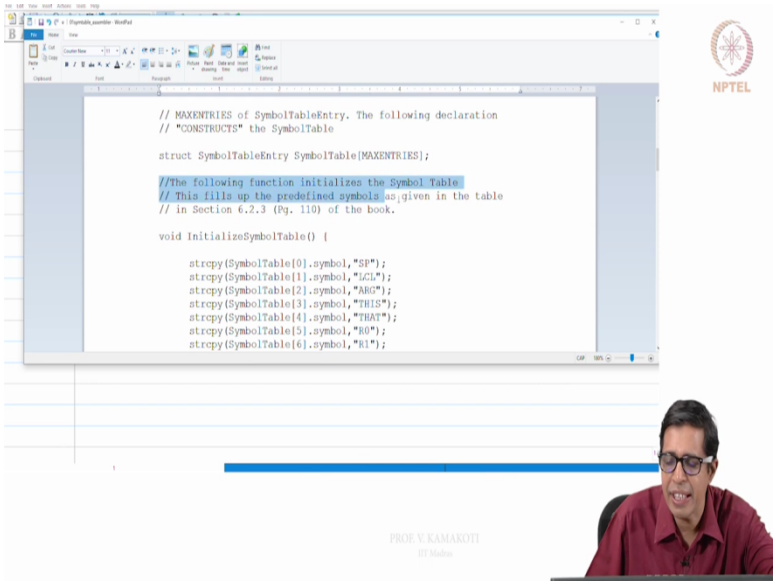
(Refer Slide Time: 12:39)



So I'm defining a Struct, now you can define a class and you can see plus plus Java etc but as far as C is concerned a Struct is there. Now I am creating a symbol table which is nothing but Max entries number of symbol table entry.

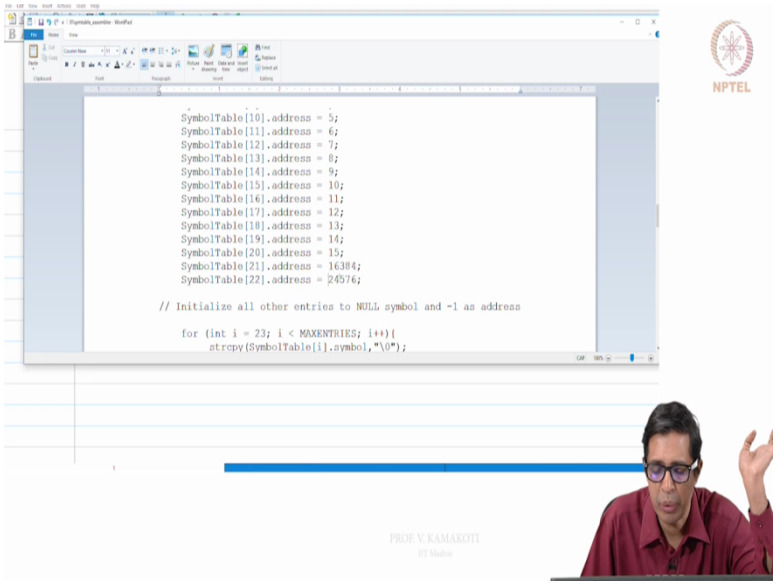
So a symbol table has Max entries number of struck symbol table entry, so now I have created a symbol table of size 1024 as of we are seeing in which each entry can store one symbol of 20 characters in length and one integer at this. So this is what, this particular declaration tells us.

(Refer Slide Time: 13:27)



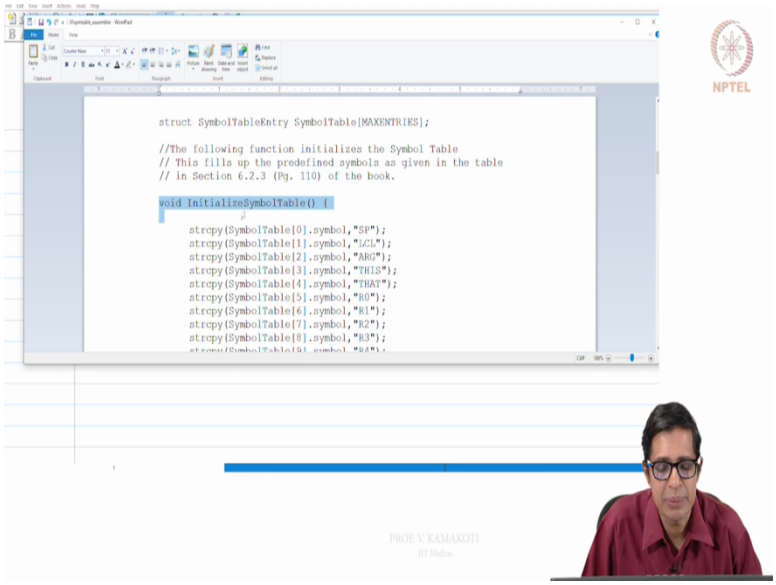
Now the symbol table basically has some predefined symbols put into it, this is available in section 6.2.3 page 110 of the book, right? And what are the things we have already seen? zero stands for the symbol SP, LCL for 1, for 2 it is ARG, for 3 it is THIS, for it is THAT, 5 it is Rnot, 6 is R1, 7 is R2, 8 is R3, 9 till R15 then so that will be 20, so then 21 is SCREEN and 22 is keyboard.

(Refer Slide Time: 14:48)



Now these are all the symbols that I am showing, so there are initially we have something like 23 symbols that is already part of your symbol table. In addition and for this address Sp is at zero, LCL is at 1, ARG is at 2 you know this is 3 that is 4 etc, right? So and then from R0 to R15 it is 0 to 15 and the screen is 16384 while the keyboard is 24576.

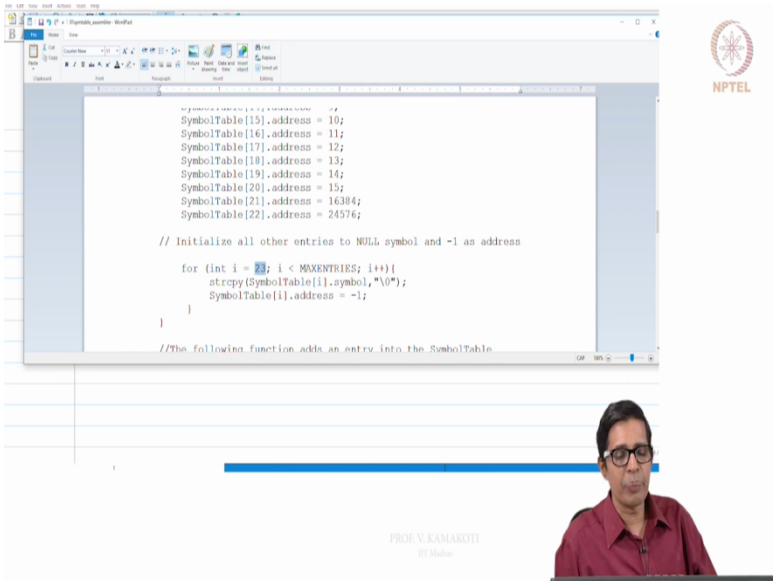
(Refer Slide Time: 14:58)



So this is the initial a session that we do for the table, so this is I have a routine called initial a symbol table, so I have initialize this and while initializing there are some preloaded predefined symbols that are loaded into it. So when we actually initialize ta symbol table for any general programming languages etc, it actually will store all the keywords or received words.

Received words and keywords would be already part of the symbol table, okay. So this is one thing that we need to know. And for all these things we have an associated address here for every symbol we need to get an address and that we have then, so we have finished off till 0 to 22.

(Refer Slide Time: 15:40)

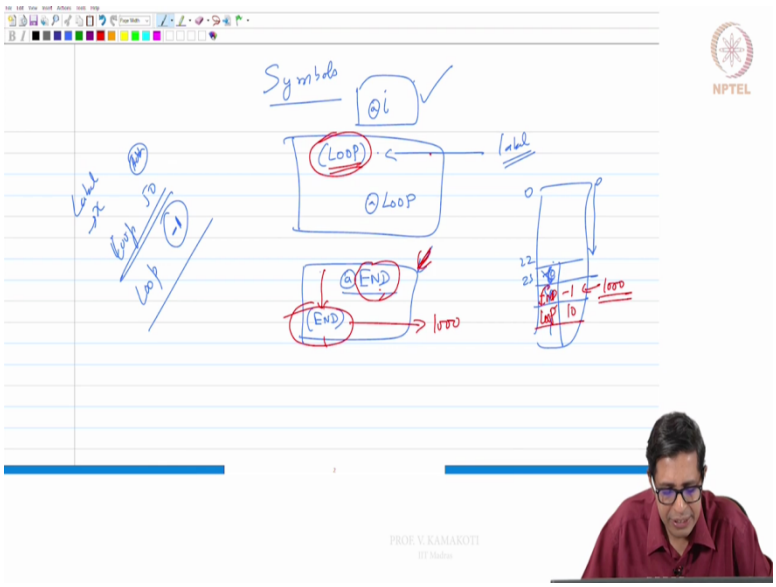


So the remaining entries from 23 to Max entries there is a Max entries, totally there are Max entries 0 to Max entries minus 1 those are the index.

So i equal to 23, i less than Max entries i plus plus we just make every symbol null and the return address as minus 1 these are all initialize but initialize to a non-occupied symbol, so these are not occupied and as I go through the program I will now keep filling up these entries with new variables as we encounter them, okay.

Now the next thing is, so this is all about initialization, so initialization is a very simple thing that I need to initialize all the address and all the entries 23, 0 to 22 is already filled and from 23 onwards you make it null. Now the next thing that we need to address is what it means to add an entry into the symbol table.

(Refer Slide Time: 16:44)



So there are 2 things, i encounter a symbol, when will i encounter a symbol? As a part of my instruction as you see here, as a part of my label instruction, so i encounter a symbol either as part of label or as a part of an act instruction, right? And when we get that label, so there are these scenarios, the first scenario is that label is already entered here, label is entered there with some value, already address is also loaded.

For example so some loop is some 50 is already loaded, there can be a case when i see this loop but the value is not loaded, it is still minus 1. So 2 cases are possible, so when I a symbol that symbol is already there along with a positive address. The symbol is there but with the address minus 1 that means the symbol is not initialized to the address but symbol is already seen at the program.

The symbol is not there itself first time we are seeing it, right? So there are 3 cases when we try to add and entry. So first thing at i normally we would not have seen i, i may not be there, so when entry is not there you search for all the things and you find the first entry that is null. So what you do is the first 22 entries are already filled from the 23 entries you have currently made it null, everything is null.

So what you mean by add entry you first search the first 22 entries 0 to 22 that is 23 entries and see if this symbol actually is there stored. If it is not stored then you go to the 23rd entry and store this symbol, so here i will get into the 23rd entry if it is coming first, in essence what you do is, you start from the 0 and keep on searching till you actually find a null symbol, right?

And if I find a null symbol and till this you have not seen your symbol that means your symbol is first-time to be entered, right? So when I want to add an entry I start from the 0th entry of the symbol table compared every symbol that is stored there till I reach a null symbol and actually filling the thing in increasing order, right? And till I the null symbol I don't find this symbol stored anywhere then I go and say this symbol is not there and so we go and store this symbol.

Along with that symbol when we store, we will also store the address but that address sometimes that address may not be available for example I say at END, let us take this case as I am marking here in red. When I see first I will encounter at END, END will not be there because the first time I am encountering, so I go to the table and find that first entry which is null from the top from zero and I will put END.

But I don't know what address to associate with it, so I will just put minus 1, right? but when I do loop, correct? I am encountering loop first-time as a symbol, as a label here I am encountering END as an argument at instruction but here I am encountering loop as a symbol. In this case I will enter loop along with some say let it be a 10th instruction I will put 10.

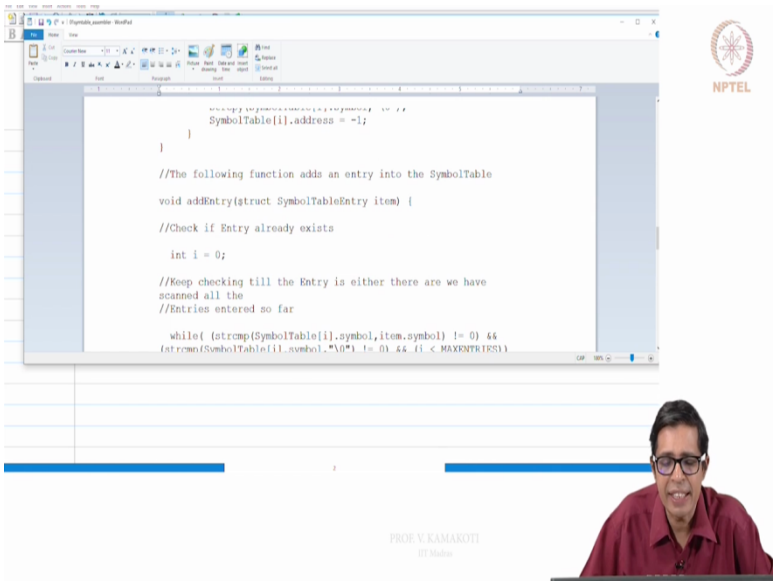
So when a symbol is basically entered its address can be known, it address may not be known. So when a symbol is first standard you can go with a valid address or you can still leave it as minus 1 for somebody else to fill. Who will fill it? In the case of n minus 1 as I go through this program finally we will land up at this END, right? Now we will go and we will find out, so for this END let me say this is 1000.

I know that this is 1000, now it's a searching with END and 1000 I will find END is already there, yes END is there but its value is minus 1, so we will now go and store thousand here, so this value of minus 1 gets updated later. This value of minus 1 gets updated later only when this value of END gets updated later in this case of a l (())(21:33), right? So in this case where at END comes before actually the symbol or the label then we land up with such a scenario.

So there are 3 scenarios when I want to add entry the thing is not there, the thing is there but with minus 1 address and the next time we are putting you are trying to make a nonnegative address, you will update only the address the thing is there with the address again. So then you just have to, again you should not make multiple entries for the same symbol.

Symbol is the key for this table, what is the key? Key is that there is uniqueness, so I can't repeat symbols again and again, so for every symbol there will be only one entry and that's what it happens, okay. So these are some of the things that we need take care when we do this add entry stuff and now let us go back to the code and see how this add entry works.

(Refer Slide Time: 22:33)



So add entry and I'm adding an item here, so first thing is we will check whether it already exists. So this is the while loop, we start from i equal to 0 and we are just checking if the item that I'm adding this item. symbol is equal to the symbol table i.symbol and that symbol table i.symbol is not null and i is less than Max entries, right? So I keep on searching, so when will this while loop stop?

I do i plus plus, I plus every time i increase in the while loop as only one statement which is i plus plus (())(23:27). So when will this while loop stop? When i find that item.symbol is equal to symbol table i.symbol that is I find the symbol there or I find here null that is this condition gets highlighted means I find a null there or I have crossed Max entries.

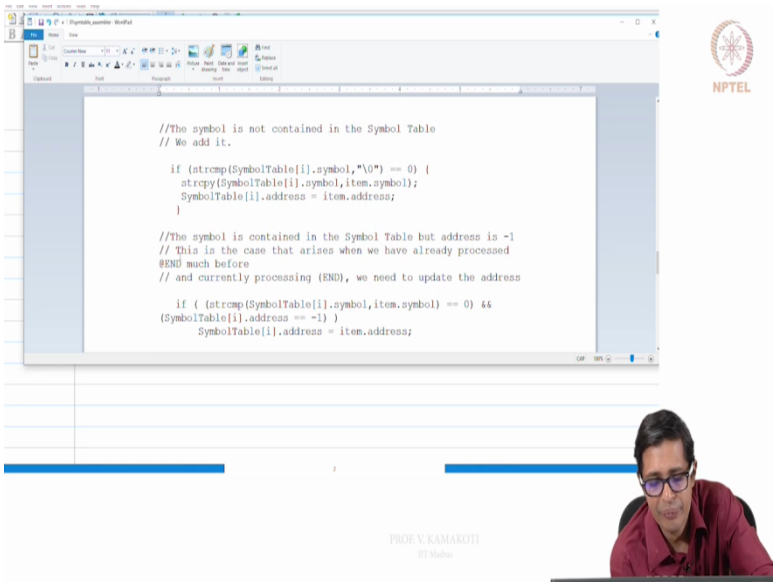
This is very very important this I less then Max entries is very important because if you don't do that then we will land up with a lot of other issues, right? The symbol table does not have enough space but we are trying to push something and we are not even reporting that. So this I less than Max entries is a very very important thing for us to report from the diagnosis point of view, okay.

So this particular while loop will start if I find a symbol or if I find the first null character and since I am filling up one by one, so if I find the first null character, so there is nothing more in

the table which has valid symbols are your I becomes greater than or equal to Max entries that means my table is already full, right?

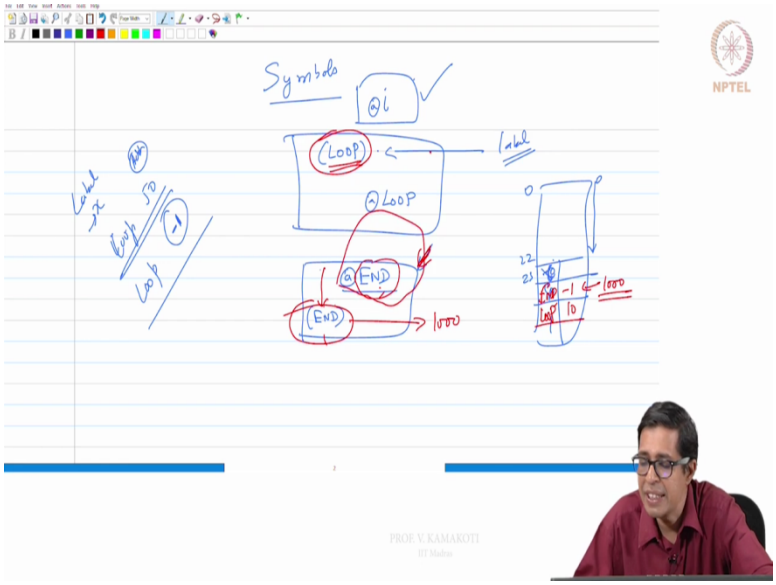
And at this point now 3 cases come if I find the null symbol, right? If it is null then I basically go and add, right? So string copy symbol table i.symbol, item.symbol, so essentially I copy that symbol onto this I also copy the address onto this, this address can also be minus 1 to start with because but we copy whatever address is provided we copy, right?

(Refer Slide Time: 25:29)



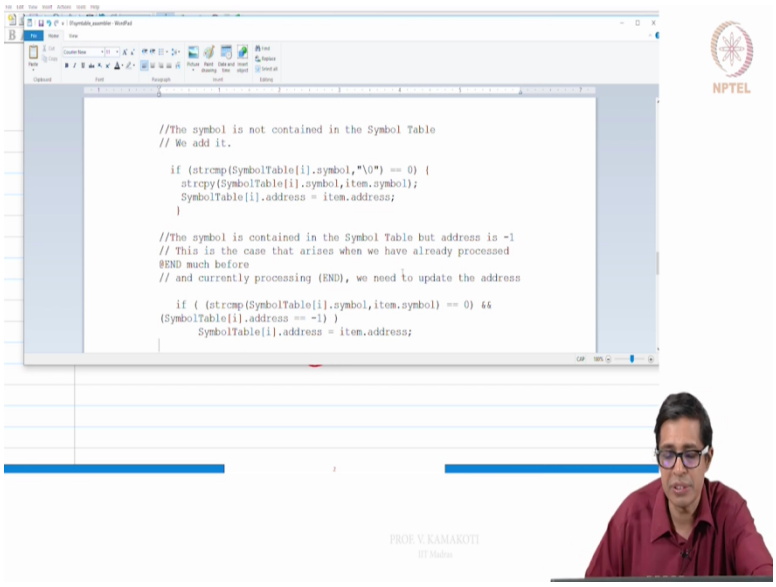
So this is the first case, the next case is you have found a symbol, right? But it's address is minus 1 now you go and update the address alone and that can happen as I have described you that could be at END before and when that END as you see here.

(Refer Slide Time: 25:56)



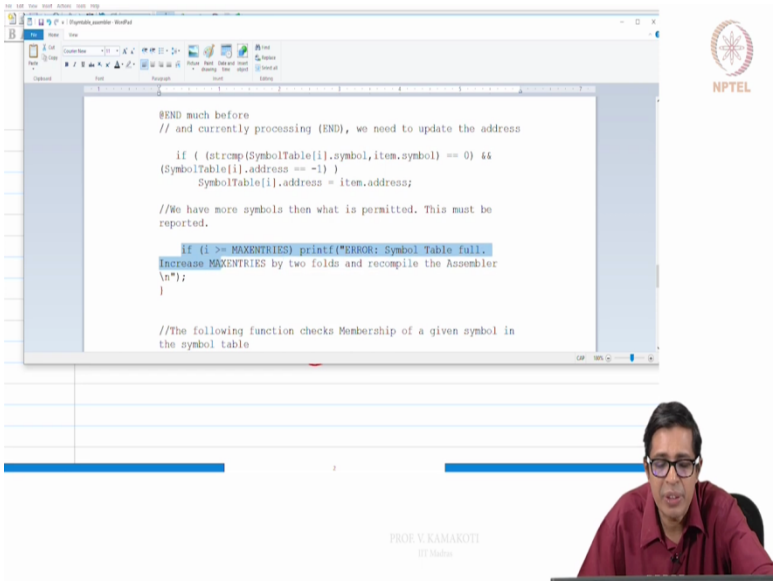
There will be at END before this and then this END is entered it would have been n minus 1 and then we see that at END later and there you go and update it with 1000, okay. Right?

(Refer Slide Time: 26:09)



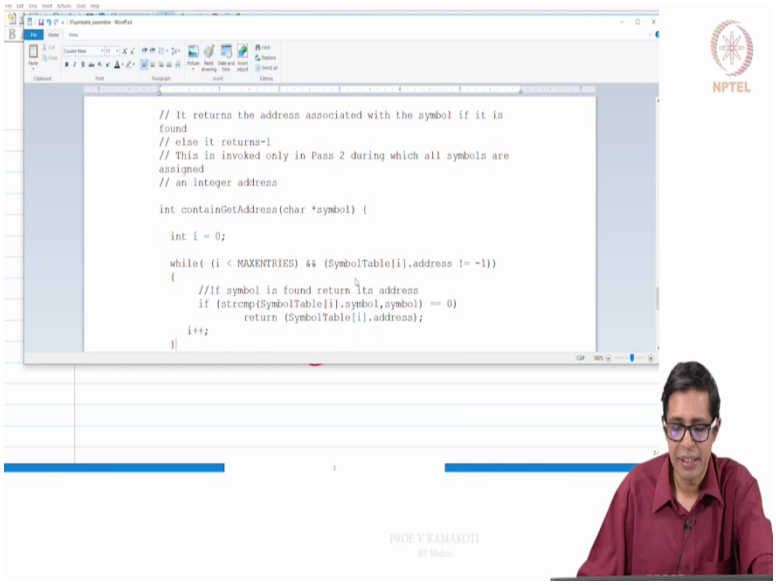
Right, so this is the 2nd thing so I found a symbol but now I find that address is minus 1.

(Refer Slide Time: 26:28)



So you go and put this address but if I is greater than or equal to Max entries then you print an error and this is in my opinion the most important habitat you need to cultivate when you try to do this type of things. So if I is greater than equal to Max entries just print error symbol table is full, you know increase the Max entries by twofold and recompile it (())(26:39) and this is very very important, right?

(Refer Slide Time: 26:50)

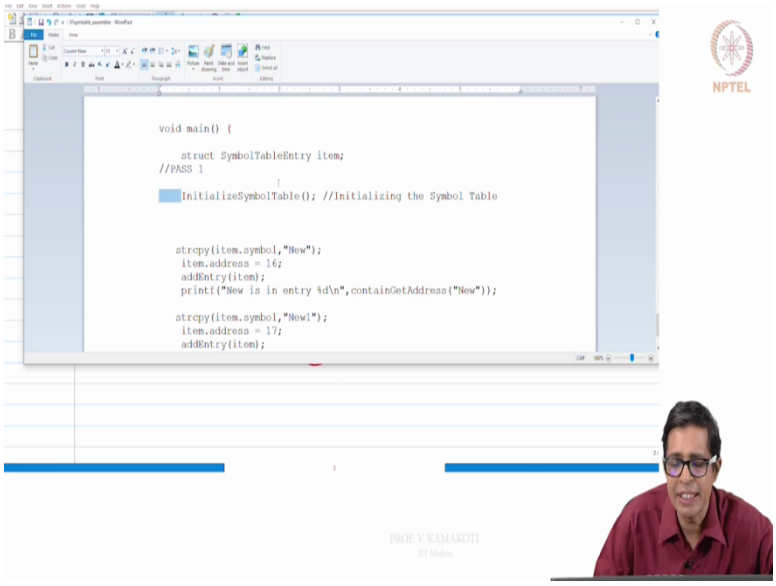


Right, now the next thing is that contain get address. This contain get address is that I'm giving a symbol and I ask what is its address? So what we do? While i is less than Max entries and symbol table i.address is not equal to minus 1, right? And symbol table i.address

is not equal to minus 1 that is if symbol is found and its return address is not with a return address, right?

Right, if string compare symbol table i.symbol, symbol is equal to 0 then you just return address i plus plus. So what you do? You go entry after entry and you find out whether this entry matches and if the address is not minus 1 then you basically return that value, right? Return that integer because I need to get the address, right? So since I put return I will reach this part of the code only if this value is not stored, so I return minus 1in this case and this part will be (())(28:03) only if symbol is not in the symbol table.

(Refer Slide Time: 28:16)



And that's it, so we will just do this simple exercise, so I can initialize symbol table the first routine which will initialize those 23 things and make everything will and minus 1 and this is very important. And the next stage is string copy, so I am just putting a symbol called new with address 16 and I put add entry item, so this item will be added into your symbol table.

Now I put new one with value 17 this will also get added, now I just see contain get address new 1 contain get address new 2 we should get 16 and 17. So to see, check that this is working properly you can also write little more test cases for this. So this is how we basically construct the symbol table and in the next module we will now go into the pass one which basically talks about how to use this symbol table for doing the pass one.

So I hope you understood how to construct the symbol table, it has basically some simple functions like you know contain get address and your add entry initialize of course only 3 routines and I am sure you have understood the concept and if you have any doubts you put it

on the forum we will answer that, we will be too pleased to answer it but the most important thing is you start coding and you see that the code works, thank you. So we will meet in the next module.