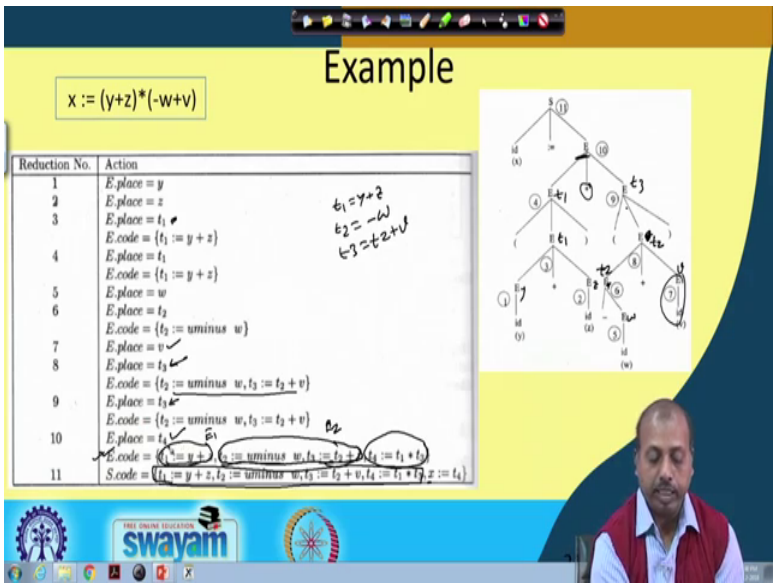
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 53
Intermediate Code Generation (Contd.)

(Refer Slide Time: 00:19)



So, in our last class we are discussing with this particular example like x equal y plus z into minus w plus v with these example and we have seen upto reduction number 6. Now, what about reduction number 7? So, reduction number 7 is again E producing i d and then the action is this E dot place equal to i d dot place that is v and E dot code is null.

Now, coming to this reduction number 8, so E plus E. So, E plus E it says that you should get a new temporary variable first. So, this E dot place equal to t 3 that is a new temporary and then E dot code it will have E 1 dot code, E 2 dot code and then this generation of a new code that t 3 equal to something.

So, t1 equal so, t 2 equal to unary minus w that is E 1 dot code. So, this one this is our E 1. So, that is E 1 dot code, then for this thing for E 2 dot code it was null. So, there is nothing, then it generates this particular line that new E dot place that is t 3 equal to this E 1 dot plus that is t 2 plus E 2 dot plus that is v. So, that way this particular line this code is generated. So, now, I have got a code that t 2 equal to unary minus w. So, the

situation I have got is t 1 equal to y plus z. So, that is there and t 2 equal to unary minus w and then t 3 equal to t 2 plus v. So, that much has been done.

Now, we come to reduction number 9. So, reduction number 9 is nothing, but within bracket E. So, this so, nothing is generated so, this place we remembered right as t 3 only ok, so, this place is t 3. So, you can write down the corresponding places. So, this is this place is y, this place is written as z, this place is w, this place is v, this is also this is now this E plus E. So, reduction number 3. So, it was generated a new temporary. So, this place is t 1, then this place is t 2 so, this minus of E so, this place is t 2. So, this is within bracket so, that is equal to t 2.

Now so, this is this place is t 3 and then when I am doing this so, this place is also t 1 ok. Now, it will be doing t 2 into t 3. So, t1 into t 3 that has to be done. So, then a line number 10, it has got E dot place equal to t 4 line number reduction number 10. So, reduction number 10 is this one. So, E plus E, so, that way it will get a new temporary variable. So, that is what t 4 and then code that is generated is E 1 code E 2 code followed by this code for this multiplication.

So, E 1 code is t 1 plus E 1 plus z E 2 code is t equal to unary minus w and then t 3 this extra code that is generated. So, this line is also there. So, this is the code of E 1. So, the upto this much is the code of E 2 upto this much is the code of E 2. And then this is the extra thing that is added for doing the multiplication that t 4 equal to t1 into t 3.

And finally, this S dot code it says that you have to have this first this E 1 dot. So, this E dot code. So, E dot code is this whole thing upto this much is E dot code as it is pointed out here. So, it will be E dot code and then it will have this i d dot place assigned as E dot place. So, i d dot place is x and then this E dot place is t 4. So, the so, this line number reduction number 10 has got E dot place in t 4. So, this x assigned as t 4.

So, ultimately we have got these as the final piece of code. So, S dot code points out. So, in this style you can generate code for this arithmetic expressions, but the only problem is that we are keeping the code as a pointer. So, that for every non terminal that, I have in my reduction tree. So, there is a pointer to a corresponding code.

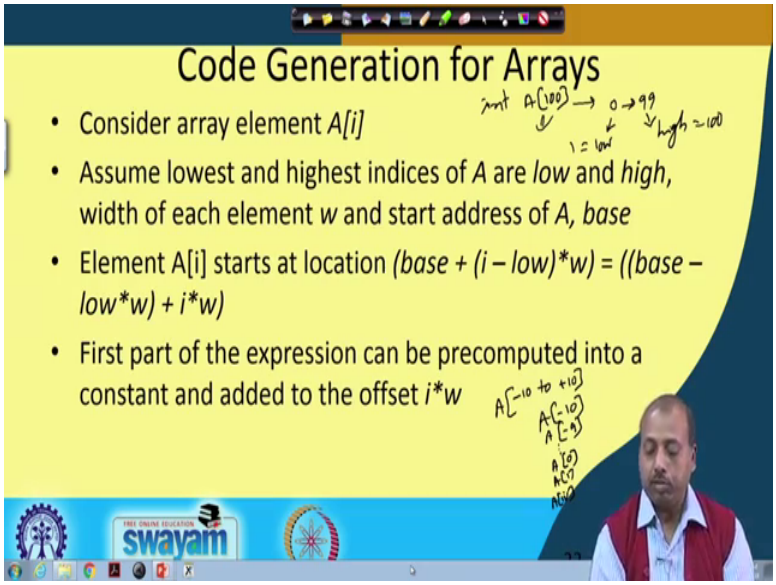
And so, their may be the same code that repeated at several place like, see this particular line say this t 2 equal to unary minus w. So, we have writing it here as well as here. So,

all these places this particular line is going to be repeated. So, the type of code generation function that we have used is that gen. So, that gen function has got this issue that is it will generate the code and while generating codes it will try to concatenate and all these things.

So, a better strategy may be that we just write the code on to a file as an when it is generated ok. So, as an when a line is generated by this the in this syntax directed translation mechanism. So, that is immediately written on to the file. And then so, that has got some difficulty also because we need to rectify some part of the code later many time what happens is that the jump targets are not known.

So, we cannot fill up those issue those points previously, but if we have got this type of organization where you have got this gen based codes. So, the code entire code is available as pointer to these individual non terminals. So, in that case that correction may be easy, but in case of while you are writing to file that correction may be slightly difficult. So, will see how are you going to do all those things?

(Refer Slide Time: 06:37)



Now, so, this is next we look into the code generation for arrays. Like, array this is a special thing, because array is a special data structure that we have. And almost all the programming language is they will have an array type of structure in them. So, generating code for array is very important.

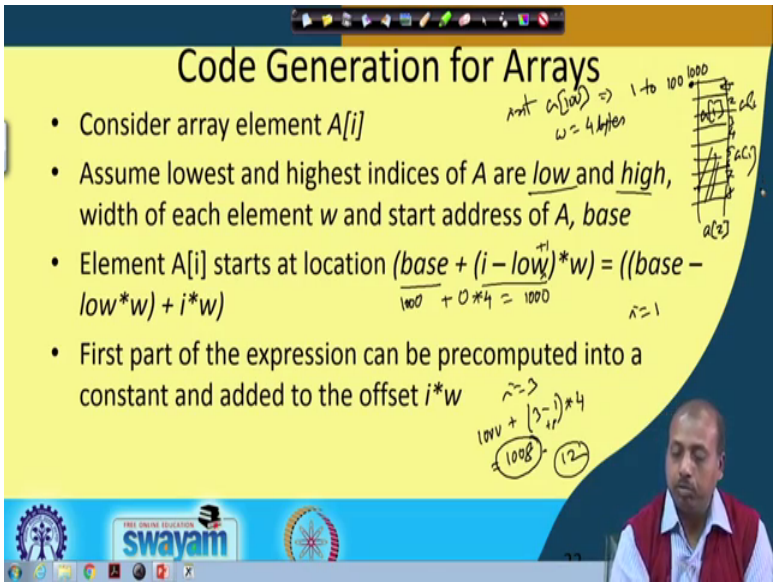
So, let us consider an array element A i. And we will try to see how this A i will be translated into this 3 address code. So, if we assume that the lowest and highest indices of A are low and high. So, in so, in general so, this array when you are declaring. So, say for example, in language c we are writing a 100 with the implicit assumption that the index will run from 0 to 99. So, this 0 is the low index and 99 is the high index.

But, it did not be. So, like if you look into different programming languages you will find different standards like, somebody may think that this array this 100 in that case low is equal to1 and high is equal to 100. Somebody may say that I can I should be able to define some array and I should be able to tell its index range very specifically. For example, somebody may say it is minus 10 to plus 10.

So, the array indices are A of minus 10, A of minus 9 in this so, it goes up to A of 0 and then A of 1, going upto A of 10. So, that can happen. So, the so, all these styles are available. So, to make it very generic what has been done is that we assume that there are low and high indices are available, that will tell us what is the lowest and what is the highest possible index for the array.

And there are width of each element is w and the start the array you starts at the base address A. Now, because when this array will be loaded into the main memory as part of the program then array may not be starting at address 0. So, depending on that so, there will be a base address, but that base address is known when the program is going into execution, because at that time I have got; I have got the array has already been loaded into memory, so, it start address is known. So, where will this element a I will start ok. So, this will be starting at.

(Refer Slide Time: 09:27)



So, let us take an example suppose I am defining an integer a 100 as the array and we are assuming that the indices they are running from1 to 100 ok. So, this is not a c style of declaration. So, it is a hypothetical language where E I into a 100 if we give. So, the indices will run from1 to 100. And if we say that the size of every element in this array is say 4 bytes; that means, in the array organization the locations so,1 2 3 and 4, so, they are deserved for a1.

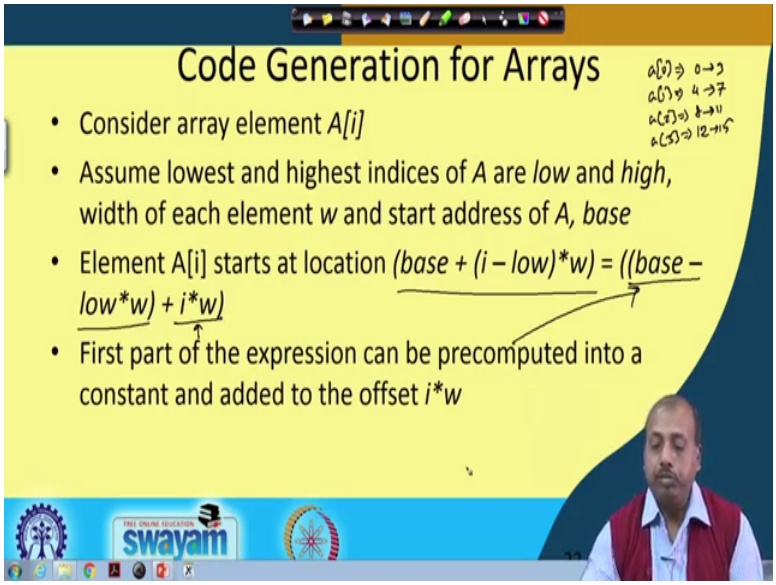
Similarly, 5 from 5 onwards a 2 will start 5 6 7 and 8, 5 6 7 and 8 they will have the element a 2; they will have the element a 2. So, it will go like this. Now, how to get a corresponding address for i? So, the address for i is given by this base plus i minus low into w.

So, because this base starts at some point suppose it start at address 1000. So, if I give i equal to say; i equal to say 1, then what happens. So, 100 1000 plus i minus low so, low is also equal to 1. So, this is 0 into size of this integer is 4. So, that is equal to 1000 and you see that indeed the index 1 starts at location 1000.

So if we try out some other value of i, say i equal to say 3, then it is 1000 plus 3 minus 1 into 4. So, that is 1000 plus 8 so, 1008. So, you see that from 1008 only so, 1 2 3 4 is so,1 2 3 4 is the first element, then so, then this 5 6 7 8. So, they are the second element so, but third element start at 1008.

So, 1008 if it start. So, that is going to be infact here it is assumed that this i minus low, so, this low starts at 0. So, that is why it is the array index starts with 0. So, that is why it is happening like this. So, if it is some other values. So, I have to do a plus here to come to the correct 1 ok. So, this i minus low so, that is 3 plus 1, 3 minus1, 2 plus 3, so, that is equal to 3. So, 3 into 4 it is starting at location 12. So, this is a element this is the third element. So, this is the element a 0, then I have got a1 here then from 8 9 10 11 I will have a 2.

(Refer Slide Time: 13:01)



So, a 0 holds the addresses 0 to 0 0 1 2 3, then a1 it will occupy the addresses from 4 to 4 5 6 7. And then a 2 will occupy the address 8 9 10 11 and then a 3 will occupy the address from this 12 to 15.

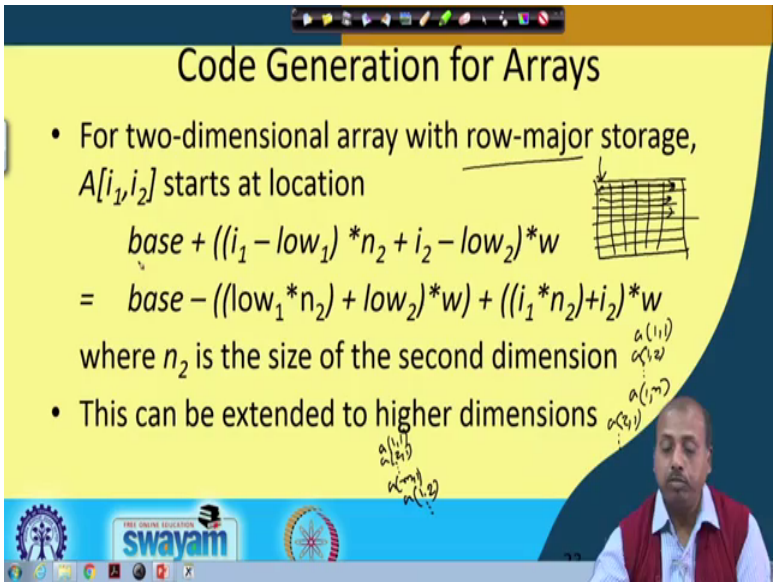
So, in this particular case it is assumed that the array index starts with 0. So, if you take that the array index starts with 1, then you have to; you have to at 1, at another 1 here ok, in this expression then it will be correct. So, first part of the expression so, it can be rewritten like this base plus i minus low into w, so if you just recomputed this expression. Then, what happens is that this base minus low into w so, that you can club in 1 group and this i into w in the other group.

So, this i minus so, first part. So, base minus low into w. So, this part is constant, because base is known and low w all those values are known. So, base minus low into w. So, that becomes a constant part and that i into w is the variable part.

So, that is that will be called the offset part. So, this first part so, this is p computed and it is stored it is remembered with the array. So, whenever this array is defined. So, you see you can compute this base low w all these fields are known. So, you can immediately compute this constant part and store it as a component in the symbol table. So, that later on when this array is referred to. So, from the symbol table you can just retrieve that this part the value of this base minus low into w.

So, that way you get the base address and then this i into w so, that you can compute getting the value of i. So, that way it can be done. So, first part of the expression can be precomputed into a constant and added to the offset i into w.

(Refer Slide Time: 15:23)



So, if that is so, then how can we do this translation like? Say two dimensional array two dimensional array so, it has got array with row major storage. So, as you know that two dimensional array. So, there may be row major storage or column major storage. So, in a row major storage, so, we store the elements row by row. So, if say this is a 2 dash two dimensional array ok, where this individual rows and columns are running, these are the rows and these are the columns.

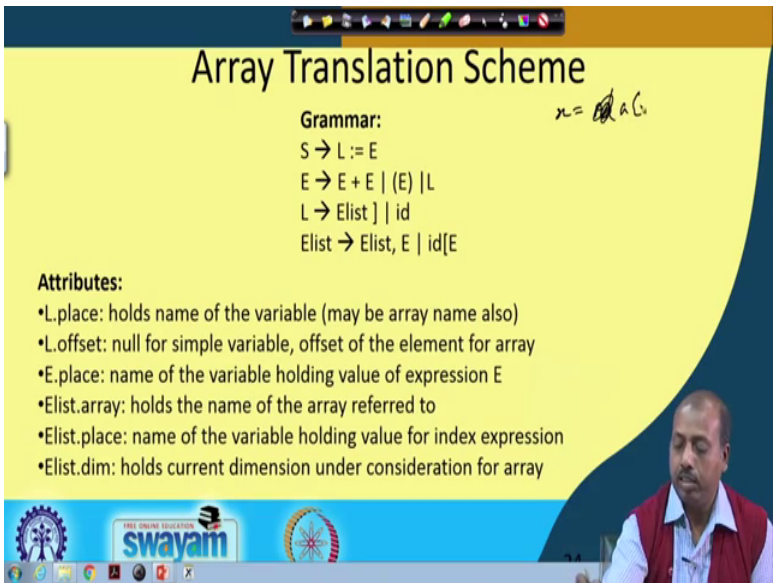
Now, in a row major organization means that ultimately memory is going to be single dimensional. So, it is not multi-dimensional. So, this first element so, we can store in these order. So, you can start store it like a 1 1, then a 1 2, then a 1 n, then a 2 1, a 2 2, so, like that. That is what I am doing is I am storing the elements in this order. First this, then

this, then this, like this, so, I am doing it like this. So, you can so, that organization is called row major organization, I can also store it in a column wise fashion like first I store a 1 1, then I store a 2 1. So, a m 1, then I store a 1 2, that way also I can do, so, if I do that. So, that will give us a column major organization.

So, here formula the for calculating this a i 1 i 2 in a row major organization is given by base plus i 1 minus low 1 into n 2 plus I 2 into low 2 into w. So, this is this you can also verify from any book on data structures. Now, after simplification so, you may like to take the constant parts together. So, that they can be precomputed and stored with the symbol table and this dynamic part may be computed separately. So, this base minus low1 into n 2 plus low 2 whole thing multiplied by w so, this whole thing is the constant part ok. So, this is the constant part for the array and then this part is the variable part this part is the offset part.

So, while n 2 is the size of the second dimension. And so you can go for higher dimension also like, you can accordingly this formula will have one more stage. So, again you will get a constant part and a offset part, for this code generation for this array a array element access. Now, how to generate the actual code? So, that we will try to see in the successive slides.

(Refer Slide Time: 18:15)

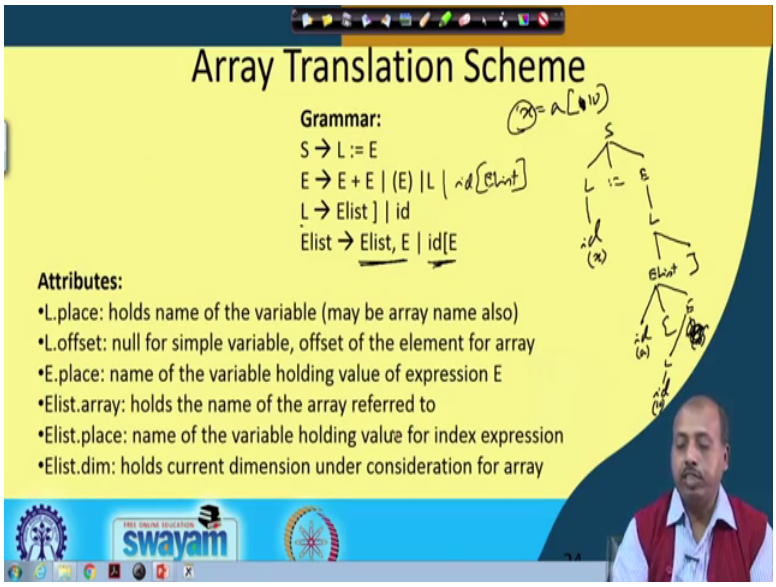


So, what is the grammar? So, that is the first thing. So, our grammar is like this. So, it is slightly twisted to make this code generation process simpler. So, we do it like this that S

producing, L assigned as E. Where, E can produce E plus E within bracket E or L and this L can be Elist bracket close i d, where Elist can be Elist comma E or i d within bracket E.

So, you see that a very simple type of assignment array assignment may be like this say x equal to a say x equal to a sorry, I should follow this particular format say 1 say a 10 very simple one dimensional array access x equal to a 10.

(Refer Slide Time: 19:07)



Now, when I am doing this? So, what is done? So, it is broke up into statement like this S this an assignment statement. So, that this is the assignment L assigned as E. And then this one from the left hand side I do not have any array. And to come to that part I have to use this rule L producing i d so, L producing i d, which is x. Now, for the E part so, I have got an array ok. So, that to come to array I have to come to L actually, from L I can goto this Elist etcetera so, I have to come to L and from this L I have to generate this a 10 this access.

So, I will do it like this, I will make it like Elist and bracket close. And then Elist so, since they have the first part of this right hand side. So, that is for multidimensional array ok. So, there is a comma here, if this is multidimensional array it will be used. So, in our case in this particular case I have got a single dimensional array ok. So, I will follow this particular structure.

So, I will be doing like i d open square bracket and E. This i d is a now this E is an expression. So, expression can give me say number or i d. So, this is a number and the number is 10. So, here I have not written explicitly the rule for that this one can be this E producing number can be a separately one or this I have not written it explicitly if I want to do it in a proper fashion as per this grammar then I have to do it like this.

This E producing L, L producing i d and i d is nothing, but this 10 do it like this ok. So, this grammar is slightly twisted otherwise I should have like this array name should be. So, this expression should be another rule should be i d within bracket E some Elist should be something like this, but for the sake of code generation, so, it has been twisted a bit ok.

Now, this particular grammar for the different non terminals we assume the presence of different attributes. Now, the first attribute the L dot place ok. So, L dot place it holds the name of the variable may be an array name also ok. For this for the for holding the name of the variable. So, this is L dot place will hold the name of the variable corresponding to L.

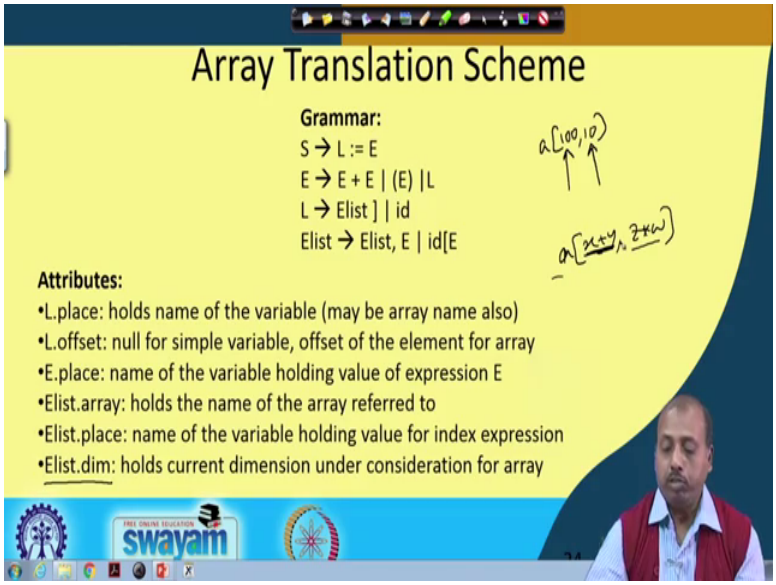
And L dot offset so, L dot offset identifies whether this is an array access or a variable access. So, if it is a simple variable then L dot offset is null. And if it is a array access then this L dot offset will hold the offset of the element for the array. So, previously we have seen that there can be this E dot w. So, we have seen previously that this type of offsets can be calculated ok. So, those offsets are calculated in this those offset will be calculated and that will come to this L dot offset, then E dot place. So, this will hold the name of the variable holding the value of expression E. So, that is E dot place.

Now, E list dot array is that is so, these are about the attributes of the L and E, for the Elist non terminal. So, we will have a number of attributes. So, Elist dot array it holds the name of the array that we are referring to, then Elist dot place it holds the name of the variable that holds the value for the index expression. So, if it a multidimensional array, then this I will get a complex expression, if it is single dimensional array then will this expression will be simple.

So, corresponding to the index whatever expression comes. So, that will be kept in Elist dot place and Elist dot dim. So, this holds the current dimension under consideration for

the array, because for each dimension the maximum size is different. So, like if I have got an array say a if I have got an array a say 100, 10.

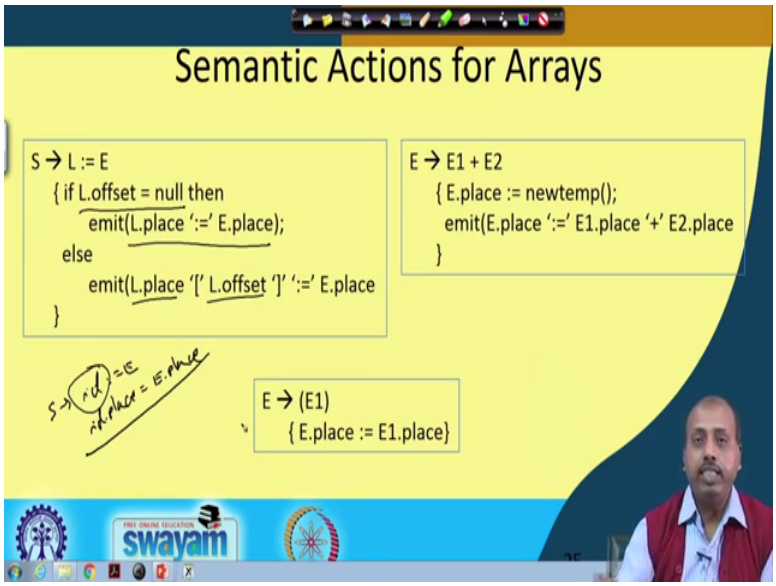
(Refer Slide Time: 24:13)



So, for the first dimension it is the dimension is 100. So, for the second1 it is 10. So, that way that way this Elist dot dimension. So, it will hold the current dimension under consideration. So, when we are doing something like it may be I am so, I am doing it like this. So, this is a x plus y comma z into w. So, this may be the array element that I am trying to access. So, this x plus y so, this is for the first dimensions.

So, this is so, this x plus y 1. So, it corresponds to the first dimension and this z into w. So, this corresponds to the second dimension. So, we will see this array translation scheme. So, it will be using this attributes very cleverly to generate the corresponding code.

(Refer Slide Time: 25:05)



Let us see, what are you going to do? So, initially I will look into this assignment statement, which is L assigned as E. So, this L assigned as E. So, if L dot offset is null; if L dot offset is null then I know that I do not have got so, there is no array reference here. So, I have to generate a code which was previously the simple assignment statement where it was S producing i d assigned as E.

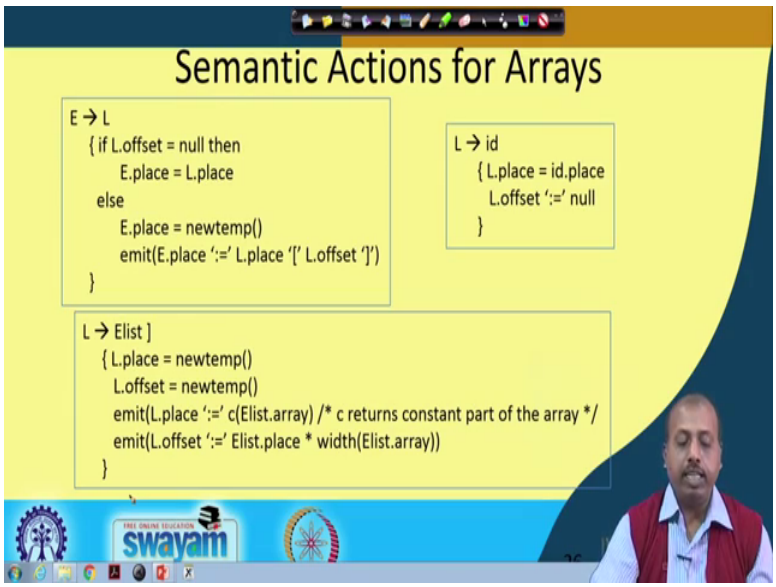
So, this was the situation and there we are said that the code that is generated is i d dot place equal to E dot place. So, this was the code that is generated. So, here also it is similar to that. So, only thing is that if this L dot offset is null; that means, it is a simple identifier like this. So, in that case I can generate this code that L dot place assigned as E dot place. Otherwise, this is an array access, but by this time this L dot place holds the name of the array and this L dot offset. So, this holds the temporary into which the index expression has been calculated.

So, you can just; you can just look into the previous slide. So, this L dot offset is offset of the element of the array. So, this so, L dot place so, this is the holds the name of the variable, and L dot offset is that contains the offset of the element for the array. So, this will be done. So, this L dot offset has got the offset part calculated into a temporary. So, that temporary variable will be used as the index. And that will be assigned the value of E dot place.

So, E dot place already have the temporary that has got the value of E that will be done. So, this is so, this is the new rule that we have and that is taken care of in this fashion, then this rule is the old one so, E 1 plus E 2. So, as we know that here I have to get a new temporary variable for this E.

So, we get a new temp here and then we emit this particular code, that E dot place equal to E 1 dot place plus E 2 dot place, so, they are added. And so, E 1 dot place and E 2 dot place there are in some temporaries. So, they will be so the code says that those values are to be added and that code should be this E dot place should be assigned to that particular temporary. Similarly, E producing within the kit E 1 so, E dot place should be equal to E 1 dot place. So, there is no change at this point. So, this way for this simple rules so, we can do this assignment for more complex ones, so, the situation will be a bit different like say this one.

(Refer Slide Time: 28:03)



Say this E producing L. Now, again the same thing that is this expression that I am talking about so, this is an expression now this L it may be a simple variable or it may be an array expression.

So, what is done is that, if it is a simple variable then this L dot offset will be equal to null. In, that case E dot place equal to L dot place. Otherwise, this I so, are to get a new temporary variable for E dot place. So, that is obtained by this newtemp. And then it generates this piece of code that E dot place assigned as L dot place, then this square

bracket L dot offset then this bracket close. So, this extra code will be generated ok. So, that way it will be generating the code for this E producing L.

So, this way, now, at the end of this so, this E will have this place attribute. So, E dot place which will be holding; which will be holding the value of this computed expression of L. And this L so and it will also have a piece of line of code in the code that is generated that E dot place is assigned as the array element access. So, we continue with this in the next class for this array access.