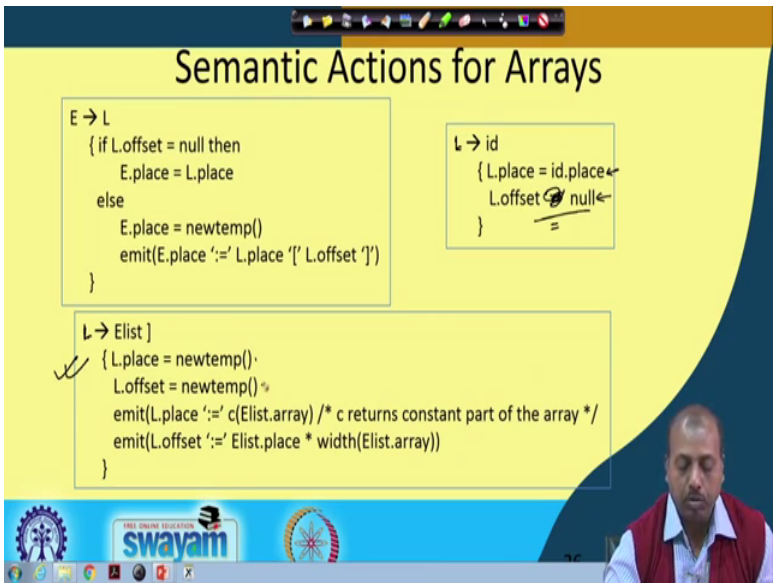
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 54
Intermediate Code Generation (Contd.)

(Refer Slide Time: 00:18)

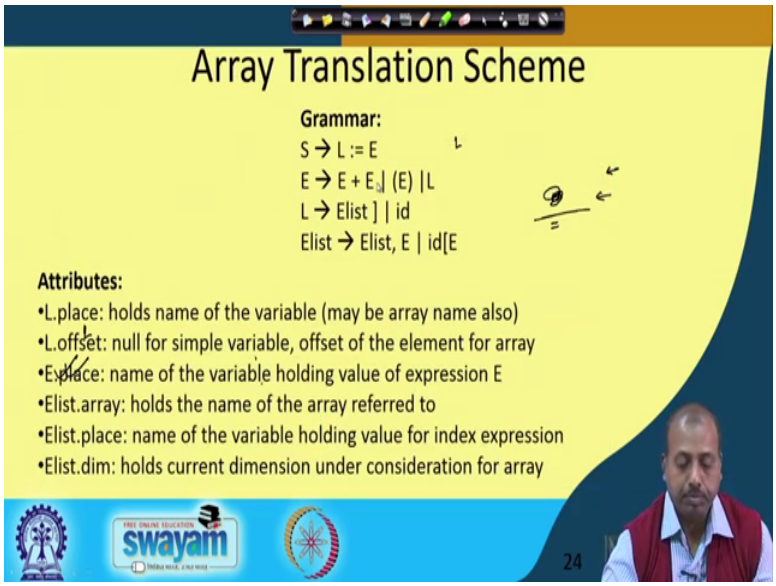


So, the next rule that we have for the array reference is this L producing i d. So, for this L producing i d the situation is very simple. So, this L dot place. So, this means that that L that I am talking about it is a single variable it is not an array.

So, this L dot place equal to i d dot place and L dot offset is null. So, this is this is made null. So, that it is it will mean that I will have so, I will have I will be having simple infact this quote is not necessary. So, it should be like this that L dot offset equal to null. So, that means, it is a simple variable.

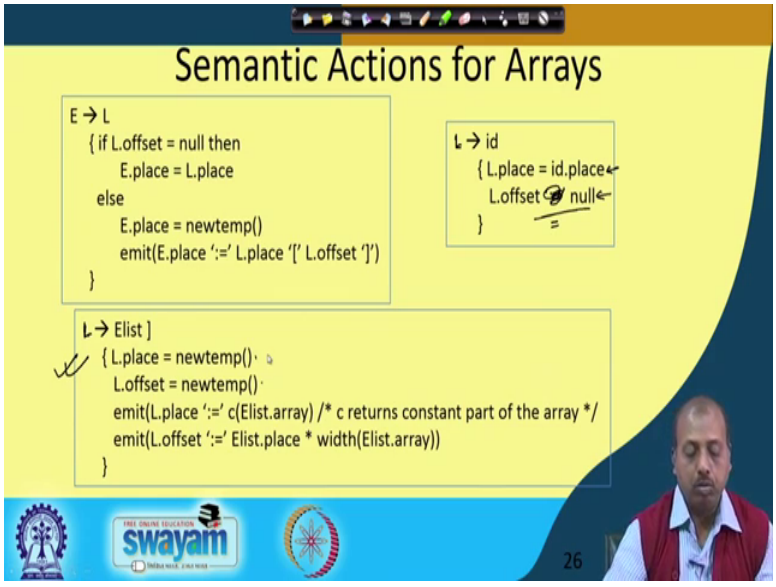
Now, let us look into this particular rule. So, this is array specific. So, this is L producing Elist bracket close. So, how are you going to handle this? L dot place is new temporary; L dot offset is new temporary. So, for this L I need to calculate the I need to generate the place and offset. So, they are generated in 2 new temporaries. And, then L dot place so, L dot place that holds the constant part of the array. So, if you look into the rule that we have seen previously.

(Refer Slide Time: 01:36)



So, this L dot place it holds the sorry Elist dot place is the variable holding the value for the index expression, at any point this whenever it is done so, L producing Eist. So, L should so, L dot place should hold the name of the variable, which may be an array and L dot offset will be the offset within the within the array.

(Refer Slide Time: 02:06)



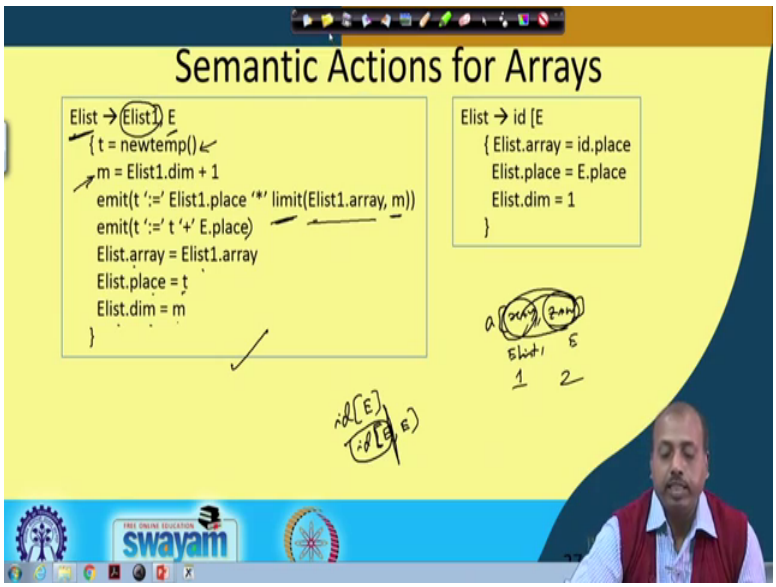
So, you see that this L dot place is newtemp L dot offset is newtemp. Now, after that so, this I have to generate this piece of code that is this newtemp equal to the constant part of the array. So, that is so, that will have this base plus etcetera etcetera so, that constant

part of the array that is available in the symbol table. So, Elist dot array holds the Elist dot array holds the name of the array.

And, then this so, for that the constant part will be retrieved from the symbol table and that will be assigned to L dot place. And, L dot offset will be equal to this Elist dot place multiplied by width of Elist dot array. So, this whatever be the Elist dot place. So, it has got the offset part. So, that multiplied by width of this individual array elements for this particular dimension.

So, they will be multiplied and that will be given to L dot offset. So, this way it will over now.

(Refer Slide Time: 03:12)



So, now if I multidimensional array. So, I will have to do it like this that Elist producing Elist 1, comma E. So; however, so, then this t equal to newtemp. So, this gives me a new temporary variable. And, m so, this holds that this is the new dimension that I that I have got. So, Elist 1 dot dimension. So, that was the dimension that we are considering previously so, which was this part.

So, for example, if you have got an array access like say x plus y comma z into w as I was talking about, now at this point this x plus y. So, this is available in Elist 1 part and this is in E ok. Now, from that I have to get this Elist. So, Elist is for this entire thing. So, how to get it?

So, t equal to newtemp so, m. So, dimension that I am talking about is previously whatever was the dimension of this Elist 1 dot dim so, this is Elist 1 dimension was 1. Now, this is the second dimension has been added. So, this becomes equal to 2.

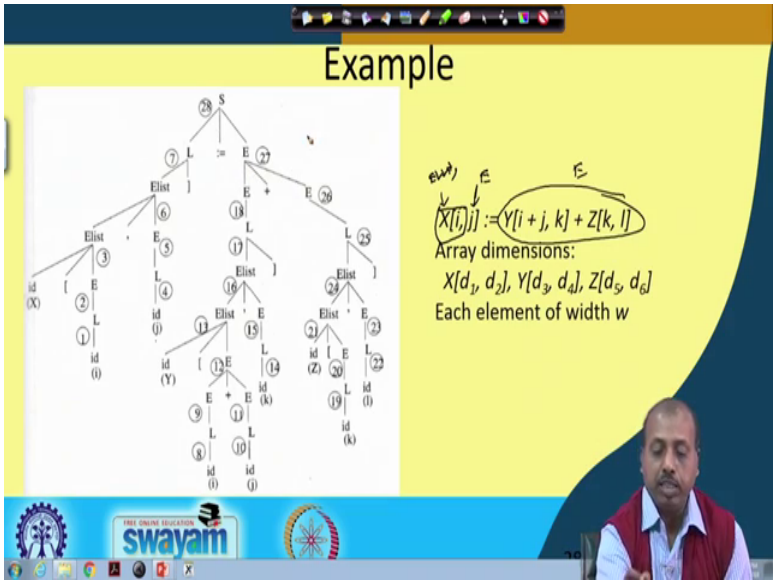
And, then we emit this particular code the t assigned as Elist 1. So, t is a temporary here. So, Elist 1 dot place multiplied by limit of Elist 1 dot array for dimension m. So, we assume that there is a function limit that will search the symbol table and tell us what is the size of the array, what is the dimension of the array in the mth order. So, if it is a two dimensional array. So, initially m is equal to 1 after that m equal to 2. So, this limit will tell in each dimension what is the maximum value or maximum index that the array can take. So, that will give us this thing.

Then, we emit this particular code. So, this t assigned as t plus E dot place. So, that is the offset that has been calculated into this E. So, that will be added and then Elist dot array so, for this Elist. So, this is the array of the array name is same as the Elist 1 dot array. So, that is taken and this Elist dot place equal to t and Elist dot dimension equal to m ok.

So, this is done this way, now the next rule that I have to consider is Elist producing i d within bracket E. So, this is basically I have got an array i d within bracket E and either it is bracket close or other situation is i d within bracket E comma the next dimension say is another E. So, like that. So, here it is so what we do we break at this point, we break at this point and we call it Elist.

So, this Elist producing i d open parenthesis the open square bracket E. So, like that. So, in that case Elist dot array is equal to i d dot place, Elist dot place equal to E dot place and Elist dot dimension equal to 1. So, that is the first dimension. So, Elist dot dimension equal to 1. So, this way I can have this semantic action for the arrays. So, bit combustion, but once you get habituated to this then you will be able to understand that it is very simple it can be the translation can be done very efficiently.

(Refer Slide Time: 06:39)



So, here is an example. So, suppose I have got this in this example. So, there are 3 arrays X, Y and Z. And, their dimensions are X is of dimension d 1 by d 2, Y is of d 3 by d 4 and Z is d 5 by d 6 and each element is of width w.

So, width of each element is taken to be same and that has to be because I am doing some arithmetic operation assignment etcetera on this that has to be done. Now, suppose that statement I have to for which I need to generate the code is this one, that X i j equal to Y i j k plus Z k l ok. So, on the left hand side as well as on the right hand side we have got arrays ok. And, the arrays are of two dimension each of them is a two dimensional array.

So, the first assigned so, for the first state that I have is S producing L assigned as E, where the L should generate the left hand side, that is this one and E should generate this right hand side this whole thing. So, let us see how L generates this left hand side. So, we have to start with this production Elist bracket close. So, in fact, the parser so, if you if you give to it the parser so, it will generate the parse tree like that.

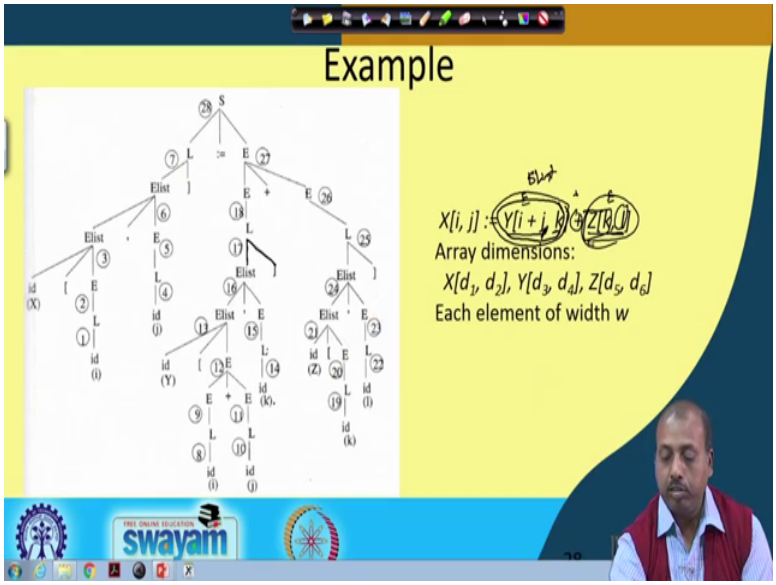
So, for the sake of our practice. So, we have to we see that we will do like this, that L produces Elist bracket close. So, that is taken and then this from this so, this Elist is the last part. So, since it is Elist bracket close so, this is basically this part j bracket close part this is the j bracket close part.

So, I have to generate this part the first part and for doing the first part so, I have to have this Elist comma E. So, that from E I can generate this i sorry so, from this E I can

generate this j part from E I can generate this j part, and from this Elist comma E. So, from Elist comma part so, I can generate this part. So, that is that will be the Elist. So, that should be Elist comma E. So, like that.

Now, from this Elist I have to generate this X bracket start i. So, comma is done. So, I have to generate from this Elist X bracket start i. So, that is i d bracket start E ok. Now, from E i come to L and from L i come to i d. So, that is i fine. Now, for this j part so, that is so, I was stuck at this point Elist comma E and then from E I have to generate L from L to i d that is j. So, that completes the parse tree for the left hand side, for the right hand side. So, this is an this is an E this whole thing is E and this is the sum of 2 expressions. So, the first rule that I should have is E plus E. So, that is done here.

(Refer Slide Time: 10:05)



So, this is first E, this is plus this is second E. Now, how this first E is going to be generated? So, again this is an array reference. So, I have to go by this E producing L and L as I said that whenever there is an array. So, the first rule that I have is by this one Elist bracket close. So, it is Elist bracket close.

So, this whole thing is Elist, this whole thing will be Elist and then the bracket close. And, after that from this Elist I have to generate this part comma and k. So, that is done here Elist comma E ok. Now, from this Elist I have to generate y bracket start i plus j. So, that is i d bracket start E ok. And, this i d will give me Y, now this E it is giving me E plus E.

So, this as this i plus j I have to get i plus j. So, it gives E plus E then this E gives me L this L gives me i d, which is I and this E gives me L which is i d which is j. So, this part is done.

So, i plus j comma so, y i plus j comma up to that much we have seen how the parse tree has been made for the k part. So, this E producing L, L producing i d and that gives the k k part. So, that finishes this one and plus is already done. Now, I have to do for this Z k L. So, Z k L again the same thing from this E if we want to come to an array the first rule is this E producing L, and then L producing Elist bracket close.

So, Elist this whole thing is Elist, then the then the bracket close is separate. Now, this Elist have to break it into Elist comma E, where E will give me this L part and Elist will give me this part and comma will be there. So, it is Elist comma E. Now, this Elist will give me i d. So, this should give me this i d, that is this Z this is the i d part, then this is bracket start and E and E giving me L, L giving me i d which i d is k and this i d is Z.

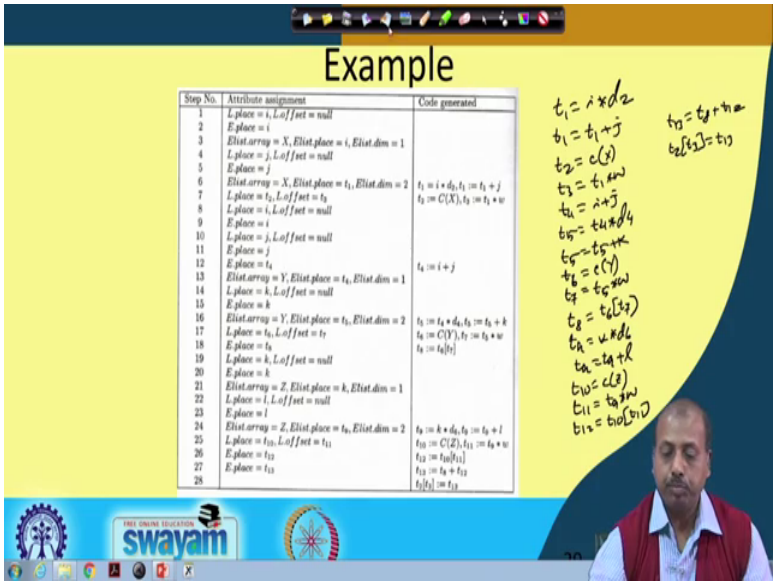
So, this way I can generate this part and then this L part. So, this can be generated by following this E producing L producing i d and this i d gives me L. So, that aims the understanding of the construction of the parse tree. So, one thing is that you see you see this is automatically done by the parser. So, you do not have to do it by hand, but just for our code generation practice and understanding the process of code generation.

So, we are doing it manually by hand ok. Now, how the code is actually generated. So, after that what I have done is that I have marked the reductions in the reverse order. The first reduction that will be done is this one. So, this L producing i d. So, that is the first reduction that the parser can do. So, this is number does 1 then this 1 number 2. So, once 1 and 2 are over. So, I can do this reduction. So, that is 3.

After, that I can do this reduction this is 4, then this one 5, this one 6, this one 7 I cannot proceed further. So, I have to come down to see where is 8. So, this is my 8, once this is done this is 9 so, this is 10, this is 11. Once 9 and 11 are done so, I can do this 12, once 12 is done I can do this 13, once 13 is done I cannot proceed further. So, I have to come down and see at this level that I can do this reduction, L producing i d 14, then after that. So, do this reduction so, 15 and after 15 so, I can do this reduction. So, this is so, this is this reduction is number 16, this reduction is number 17, this is 18.

So, up to that is done now again I have to fall back and see what is the next possible reduction. So, this is this one L producing i d. So, number 19, then E producing L 20, then this reduction 21, then L producing i d 22, E producing L 23, then this is 24, this is 25, 26, 27, 28. So, this way we number them so, that we can understand like how this code will be generated for this individual statements ok.

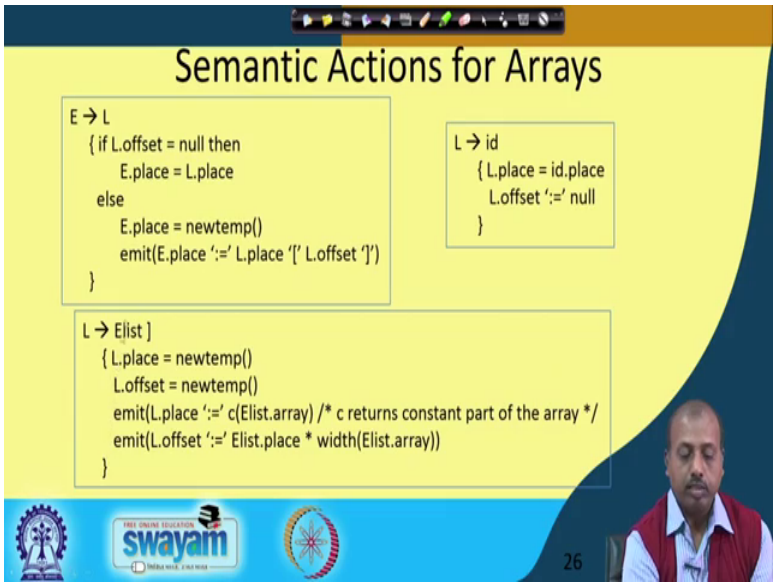
(Refer Slide Time: 14:53)



So, this is the action that goes on as we proceed through this parse tree and try to generate the code. So, at step number 1 so, at step number 1 is this one. So, this L producing i d and the rule for that is L producing i d. So, L dot place is i d dot place L dot offset is null.

So, that is done here. So, L dot offset is i d dot place that is i and L dot offset is null. So, no code is generated. Now, step number 2 E producing L. So, E producing L tells that E dot place will be L dot place.

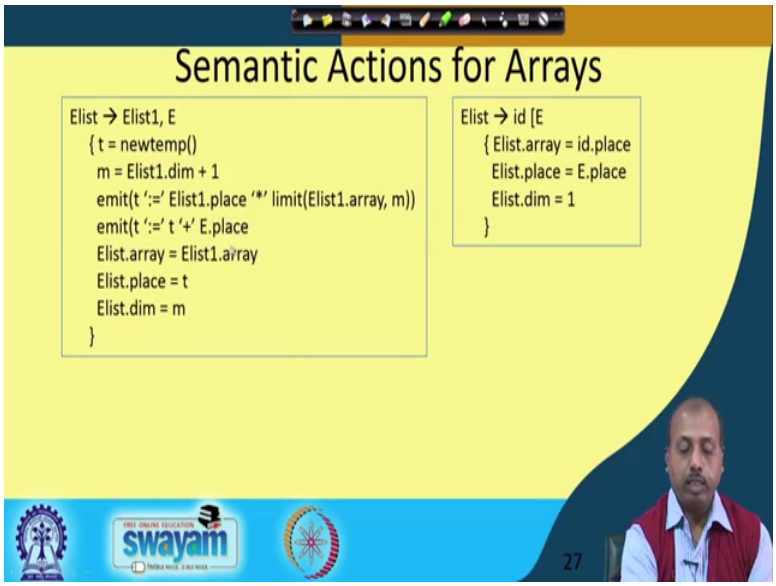
(Refer Slide Time: 15:34)



So, L dot offset is null so, in that so, I have got I will have E dot place equal to L dot place. So, this E dot place equal to i. Now, step number 3. So, step number 3 is this one. So, this is Elist producing i d within bracket E. So, let us see what is the corresponding action. So, this is this is the rule. So, it says that Elist dot array will be i d dot plus Elist dot place will be E dot place and Elist dot dimension will be 1. So, let us see whether that has happened here or not. So, Elist dot array equal to X ok, that i d Elist dot place equal to E dot place that is i and Elist dot dimension equal to 1.

So, that has been done here. Now, comes to step number 4 this L producing i d. So, L producing i d again the same thing that L dot place equal to i d dot place. So, L dot place equal to j L dot offset equal to null. Now, at step 5. So, E dot place equal to L dot place. So, E dot place equal to j. Now, step number 6 is again so, this one Elist comma E. So, that reduction is done.

(Refer Slide Time: 16:53)



So, for that Elist comma E, so, this is the rule. So, I have to get a new temporary variable number or dimensions are to be incremented, then it will generate this type of code. So, what is done here you see at step number 6 is Elist dot array is the name of the array coming from Elist 1 dot array. Now, Elist dot place equal to t 1. So, new temporary and this dimension increases to one more that is 2 and then t 1 equal to i into d 2 so, that is the size of the second dimension.

So, i into d 2, then t 1 equal to t 1 plus j so, that is t 1 plus this offset part E dot place. So, that will be added. So, this code will be generated and, then at reduction number 7 at reduction number 7. So, L producing Elist bracket close this open close parenthesis. So, Elist bracket close. So, this one, this L dot place newtemp L dot offset newtemp, then it will emit this piece of code.

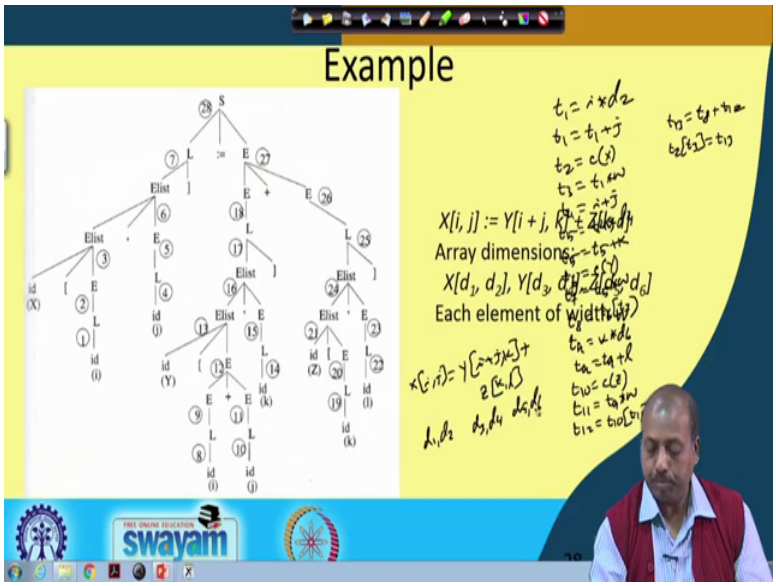
So, this L dot offset L dot place is t 2 L dot offset equal to t 3. Now, it generates the code that t 2 equal to C of X and t 3 equal to t 1 into w. So, t 1 it has calculated the offset. So, that multiplied by the size in the dimension that is t w. So, it is t 1 into w, then, at step number 8 reduction number 8. So, it will be at reduction number 8. So, it is L producing i d. So, it will be doing it this way that L l dot place equal to i and L dot offset equal to null no code is generated.

So, next code is generated somewhere at this point at this reduction, reduction number 12 ok. Now, so, you can trace through this code generation process. So, will do some bigger

exercise later once we come to the end of this chapter, but ultimately we have got this piece of code generated. So, we have got this t 1 equal to i into d 2, t 1 equal to t 1 plus j, then t 2 equal to constant part of x t 3 equal to t 1 into w, then t 4 equal to i plus j, t 5 equal to t 4 into d 4 t 5 equal to t 5 plus k, then t 6 equal to constant part of Y, then t 7 equal to t 5 into w.

Then, t 8 equal to t 6 t 7, then t 9 equal to k into d 6, t 9 equal to t 9 plus L, then t 10 equal to constant part of Z, t 11 equal to t 9 into w, t 12 equal to t 10 t 11, t 13 equal to t 8 plus t 12, and t 2 t 3 equal to t 13. So, let us try to correlate this code with the example that I had that we had taken. So, it is X i j plus Y i j.

(Refer Slide Time: 21:24)



So, we have got this x the example that we have taken is x i j equal to y i plus j k plus z k l. And, the dimensions are x is of dimension d 1 d 2, y is of dimension d 3 d 4, and z is of dimension d 5 d 6 ok. So, let us try to see whether this particular code is being generated x i j equal to y i j k plus y plus j k plus z k that is generated or not.

So, this is i into d 2. So, i into d 2 is for the first dimension. So, it is doing like say i into d 2 and then it is doing t 1 equal to t 1 plus j. So, t 1 equal to so, i t 1 was I into d 2 plus j. So, that is giving me the offset of this i j. And, t 3 t 2 now equal to constant part of x and t 3 equal to t 1 into w. So, that gives me the actual offset. So, this t 2 contains the base part and t 3 contains the t 3 contains the t 2 contains the constant part of the array reference and t 3 contains the base the offset part. And, at the end you will notice that I

am doing like t 2 t 3 assigned as t 13. So, t 2 t 3 so, it is x i j ultimately ok. So, if you look into this so, this X i j equal to something. So, this X i j is turning out to be this t 2 t 3.

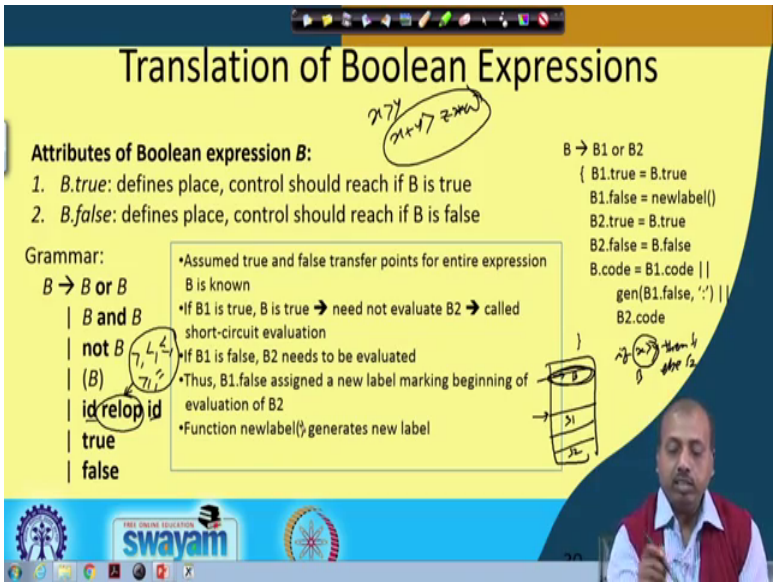
Now, for the next part so, y i plus j k so, y i plus j k for that so, this in t 4 it is computing i plus j. Then, that is multiplied by the dimension of the second array so, that is d 4, in the second in the second index. So, it is d 4, then t 5 equal to t 5 plus k. So, this is the second index that I have so, the it was x y i plus j k. So, i i plus j you have computed the offset. So, with that the offset for k has to be added. So, that is done here. And, t 6 is the constant part of y t 7 is the variable part of this offset part of y axis. So, this is t 7 equal to t 5 into w. So, ultimately I have got t 8 equal to t 6 t 7. So, this t = 8 contains the this part so, y i plus j k part.

Now, this z k L part so, for that t 9 equal to k into d 6 so, d 6 is the dimension of the third array in the in the second index, t 9 is t 9 plus L and then t 10 is the constant part of z t 11 is t 9 into w, then t 12 is t 10 t 11. So, that way I have got this in the variable t 2. So, I have got the array array element copied, then t 13 is t 8 plus t 12. So, t 8 was having this t 6 t 7 and t 13 is having t t 12 is having t 10 t 11. So, they are added. So, we get t 13 equal to t 8 plus t 12. Ultimately, t 2 t 3 assigned as t 1 3. So, this is holding the value of t 1 3.

So, so, t 1 3 is having the t 8 plus t 12. So, this right hand side of the expression. So, this whole thing is available in t 12. So, that is added with t 8 and t 8 was holding the first array access t 67. So, they are added and then that is assigned to the array on the left hand side t 12 t 13 ok.

So, this way we can generate the code for the array elements. So, it is a bit combustion, but so, if with a little bit of practice. So, you will be able to do it by hand so; will also try to do a few some exercises in the class towards the end of the chapter.

(Refer Slide Time: 25:35)



Next, will be looking into translation of Boolean expressions so, Boolean expressions are something very important, because many a time we have got this Boolean variables and this Boolean controls are there like say if then else type of control, while loop type of control. So, like that so, we have got these Boolean expressions. So, they form part of this control statements in a programming language.

So, for doing this translation of the Boolean expression so, will be doing it like this. So, will assume that there is an attribute B dot true for any Boolean expression B, it will have an attribute B dot true. So, that defines the place and control that control should reach if B is true. So, in the program execution, if this is a portion of the program and at this point suppose I am evaluating, this is here I have got some code which is evaluating the Boolean expression B.

Now, as I said that these are these are normally part of if then else or while go to statements. So, if this b condition is true then where to go ok. So, may be if it is if then else statement. So, if some x is greater than y, then s 1 else s 2. So; that means, if this is my B ok. So, if B happens to be true then I should jump to the portion of code where s 1 is being executed.

So, s 1 has been coded. On the other hand if B is false in that case I need to jump to another piece of code which S 2. So, that way with b the Boolean variable B so, we assume that we have got for the Boolean non terminal B. So, we assume that there are

two attributes B dot true and B dot false, while B dot true defines place the control should reach if B is true, and B dot false will define place while control should reach if B is false.

So, the grammar that will be considering is this and or grammar so, and or not these are 3 operators. So, B or B B and B not of B within bracket B, then we also have got this relation relational operators i d relop i d. So, you can have this relop. So, this is the relational operator. So, all the relational operators like greater than less than so, greater or less or equal greater or equal equality. So, all of them we club together and call them the relational operator relop.

Now, this relop operator so, this will so, this. So, we are only allowing this i d relop i d. So, you can only write like x greater than y like that. So, we can we cannot have expression like x plus y greater than z into w. So, that type of expression. So, we are not allowing in this particular grammar ok. So, that can be done, but for the sake of our understanding, because we do not want to bring this arithmetic operators into consideration. So, they are not taken here.

And, then the since it is a Boolean expression so, we have got 2 constant identifiers true and false. So, they are also part of the grammar. So, this is the grammar that we have and then we will see in the next class, how to have this code generation for the Boolean expressions.