

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

Lecture – 49
Runtime Environment (Contd.)

(Refer Slide Time: 00:18)

Compiler's Responsibility

- Proper code to access the correct definitions:
 - Find difference d between the lexical nesting level of declaration of the name and the lexical nesting level of the procedure referring to it
 - Generate code for following d access links to reach the right activation record
 - Generate code to access the variable through offset mechanism

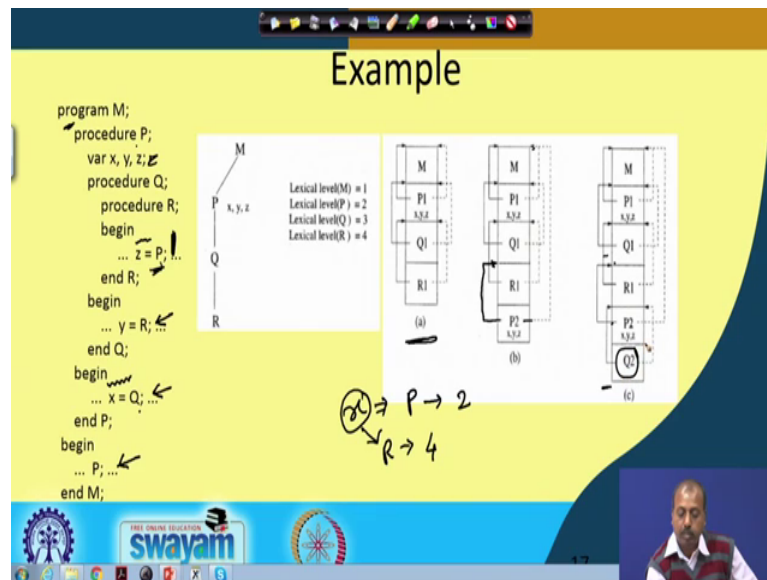
$x \rightarrow P \rightarrow 2$
 $\quad \quad R \rightarrow 4$

The slide features a yellow background with a blue header and footer. The header contains the title 'Compiler's Responsibility'. The footer includes logos for 'swayam' and 'IIT Kharagpur'. A handwritten diagram in the center shows a variable 'x' with an arrow pointing to 'P', which has an arrow pointing to '2'. Below this, 'R' has an arrow pointing to '4'.

So, what is the compiler's responsibility? Like, so this access link is fine. So, we have to go by the access link to come to the corresponding definition of your name of an identifier. But, how the compiler will generate code so that this access will be correct. So, first thing that it has to do is to find difference d between the lexical nesting level of declaration of the name and the lexical nesting level of the procedure referring to it. So, this is basically in our previous case previous example that we had.

So, we were referring to x and then the, so it is defined it is, so this is at this is this was at procedure P and procedure P is nesting level was some say 2 because main was 1 and P was 2. And then this it is now different at the current procedure that we are referring is a procedure R whose nesting level is 4, so it has to go by 4 minus 2. So, it has to go by two access links further to come to the proper definition of x . So, this will generate code for following d access links to reach the, right activation record and then it will generate code to access the variable through offset mechanism.

(Refer Slide Time: 01:37)



So, we will take an example and explain it. So, like in the previous case what we had. So, this was the program that we are talking about. So, here we have got a number of variable, so x, y, z etcetera. Now, suppose we are at a situation where from the main routine the P 1 has been called ok. So, when the P 1 has been called, so in P 1 it has got reference for this x y and z and this P ones. So, this access link points to the made as it corresponds to the parent frame, so that is M. So, it is point P is the access link is pointing to M.

Similarly, so from P 1 when this after from this procedure P we are going to call Q. So, this is the begin of P, so from here we are going to call Q. So, when this call Q is made so this Q is invoked and then it has got its access link is pointing to P because Q is defined within P ok. So, if you call if you have to look for some definitions. So, we have to look into the local variables of P first ok.

So, if it is, like this x; so, x is not defined in Q in the procedure Q ok. So, it is so this is not defined in the procedure because, so this variable y is refer to. So, y is not defined in the procedure Q. So, when it is trying to get the corresponding declaration for a y so it has to look into the activation record the, it has to look into the activation record for the nesting procedure and the nesting procedure happens to be procedure Q. So, this R it points to procedure Q and Q a points to procedure P.

So, that is the situation where this M has given a call to P, P has given a, so this is M has given a call to P at this point P has given a call. So, it has got the main has call given a call to P ok, then P has given a call to Q. So, this is the P, so this has given a call to Q and Q has given a call to R. So, this call has been made and we are in this procedure. So, we are in this procedure. Then that is the situation that is depicted by the first diagram ok. So, it has not yet made this call for procedure P.

So, you can say that as if we are somewhere here, ok. So, at this point the this is the situation of the activation record that in the stack. So, this solid lines are the control links and this dotted lines at the access links. So, this access links, so, they are pointing to the corresponding procedure into which we have to look into.

Now, the situation changes when we have got this thing see this, see this one, see this when from this P when from this R when this P is called again. So, another frame gets created and this P 2 is this frame P 2 is pushed into the stack. And then this frame pointer so this control link points to the parent frame from where it was called.

And the access link for P, so see now it points to this points to M. Why? This procedure P is statically nested within M. So, it is now, so though it has been called from R. So, this call is from R, but R is not the place where it should look for some variable which are which may be undefined in P. So, for all the variables that this P 2 wants to access. So, it has got this local variables fines, but if it has to access in the code for P if there are some access to the some variables which are defined in the form in the main then in that case it will be going into this. So, you see that the control link points to R 1, but the access link points to M ok. So, this is the situation at this point.

Now, after from this if I think that this P call has been made and within from this P call, so it has given a call to Q and Q has given a call to R. So, then there this is the situation at that point. So, this M, P 1, Q 1, R 1, so that was remaining from R 1 a call to P has been made, so this is P 2.

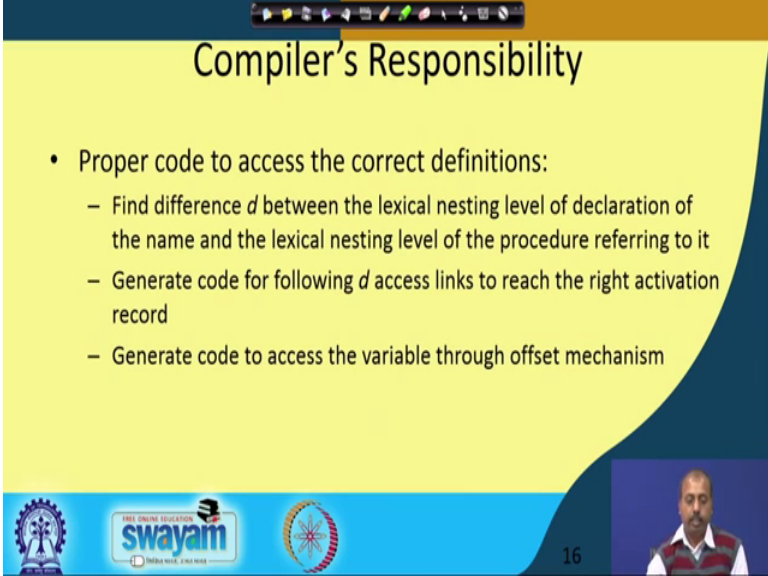
So, access link points to the main routine, control link points to R 1, the access link points to the frame corresponding to main and this control link points to the frame corresponding to R 1. And from P to another call to Q has been made, so this Q frame has been. So, this Q frame has been the activation record for Q has been created and then

this points to the this access link points to the activation record for P because this procedure Q is defined within procedure P.

So, that is statically nested thing. So, Q is within statically nested group of P procedure P. So, that way this access link points to P. And then this control link also points to P because from P this called to Q has been made ok. So, both of them are done like this.

And you note that this Q, it is not pointing to M, the access link is not pointing to M because Q is defined within procedure P. So, for definition it has to look into procedure P only. So, this way this access link and control link, so they can be useful for getting the getting access to local proper variables that we need for a particular program for a particular procedure ok. And for a define depending upon the language the this nesting rules we will come and based on that the compiler generator can design this portion.

(Refer Slide Time: 07:56)



The slide is titled "Compiler's Responsibility" in a large, bold, black font. Below the title, there is a bulleted list of three items:

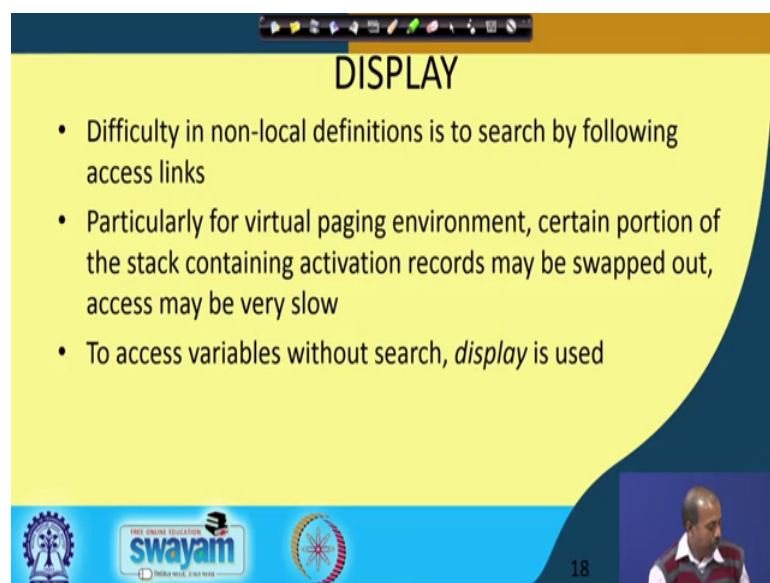
- Proper code to access the correct definitions:
 - Find difference d between the lexical nesting level of declaration of the name and the lexical nesting level of the procedure referring to it
 - Generate code for following d access links to reach the right activation record
 - Generate code to access the variable through offset mechanism

At the bottom of the slide, there are four logos: a gear-like logo on the left, the "swayam" logo in the center, a circular logo with a star on the right, and a small video inset of a man in a red and white striped shirt in the bottom right corner. The number "16" is visible in the bottom right corner of the slide area.

Now, so that is the situation, so what the compiler designer has to do, is to find the difference d between the lexical nesting level of the declaration of the name and the lexical nesting level of the current procedure. And after knowing that so it will be it will generate code, so that at runtime the system we will traverse d such links and it will come to the proper frame. And after coming to the proper; after coming to the proper frame, so it can use the offset mechanism that we have discussed previously to come to the exact location where that particular variable occurs ok.

Now, you see that this is the very combustion thing because what is happening is that as this nesting level increases, so there is a they get tree like this. So, this will be, this will be growing like this and then for if something is defined at level M that for example, then it has to go through all this levels, so it has to traverse through a large number of links to come to M. So, can we do something so that it may be made faster, ok. And for doing it faster so, there is a concept called display. So, this display has got nothing to do with say computer display and all. So, this is a data structure and in the display data structure. So, we store the relevant pointers to the to this activation record stack.

(Refer Slide Time: 09:20)



DISPLAY

- Difficulty in non-local definitions is to search by following access links
- Particularly for virtual paging environment, certain portion of the stack containing activation records may be swapped out, access may be very slow
- To access variables without search, *display* is used

18

So, we will see how this thing works. So, it is the difficulty in non-local definitions is to search by the by following access links. So, that we have understood because if there are nonlocal definition, so you have to go by the access links only to explore the previous procedures and come to the proper definition.

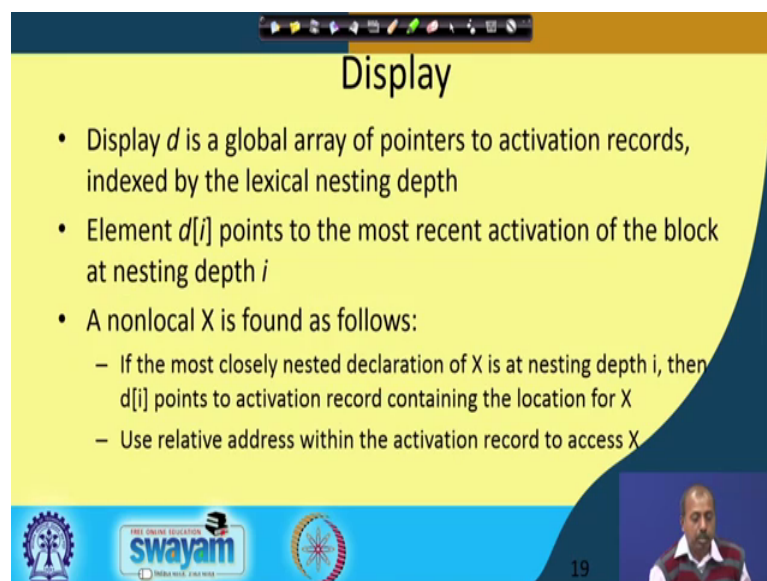
And particularly for virtual paging environment certain portion of the stack containing activation records may be swapped out and access may be very slow. So, what it may. So, virtual paging is a memory management policy where we keep only a few pages of an executing program into the main memory. So, the idea is that we can we do not load the entire program into main memory at any point of time. So, only the relevant pages are loaded. And then after the after sometime if a reference is made to a page which is not present in the main memory then a page fault occurs and the system we will system

will tell the disk save controller to transfer the proper pages from the disk to the main memory and then the execution of that process resumes.

So, it becomes slow because in between the disk controller has to come into play. So, if it happens like this that this stack becomes very large then the system may decide that, I will be a swapping out some portions of the stack for some are more important jobs and then the activation. So, as a result some of some part of the activation record maybe swapped out from the main memory. So, this will make the access very slow.

And how to solve this problem? So, we are trying to look for some solutions which is similar to the cash memory type of organization, where the most important and most relevant portion, so they are kept in some fast accessible memory. So, here also we do something like this. So, we use a data structure called display which will be used to access variables without search ok.

(Refer Slide Time: 11:23)



Display

- Display d is a global array of pointers to activation records, indexed by the lexical nesting depth
- Element $d[i]$ points to the most recent activation of the block at nesting depth i
- A nonlocal X is found as follows:
 - If the most closely nested declaration of X is at nesting depth i , then $d[i]$ points to activation record containing the location for X
 - Use relative address within the activation record to access X

19

So, how are you going to do this? So, this is the situation.

(Refer Slide Time: 11:25)

Example

- Maximum nesting depth 4, so 4 entries in the display
- In Fig (a), M has called P, P has called Q and Q has in turn called R
- Compiler knows that x is in procedure P at lexical level 2
- Code is generated to access second entry of the display to reach the activation record of P directly
- Same is in Fig (b)

The diagram consists of two parts, (a) and (b), showing activation records and a display stack. In part (a), the activation records are M, P1, Q1, and R1, stacked vertically. The display stack is a vertical list with four entries: 1, 2, 3, and 4. Arrows point from each activation record to its corresponding entry in the display: M to 1, P1 to 2, Q1 to 3, and R1 to 4. In part (b), the activation records are M, P1, Q1, R1, and P2, stacked vertically. The display stack is a vertical list with four entries: 1, 2, 3, and 4. Arrows point from M to 1, P1 to 2, Q1 to 3, and R1 to 4. Additionally, an arrow points from P2 to entry 2 in the display stack.

(a) (b)

20

So, we have got this M. So, the this suppose this was the this were the procedures M, P, Q and R. So, at this point of time M has given a call to P, P has given a call to Q and Q has given a call to R. So, that is the situation. So, there are 4 procedures which are active at this point of time. So, 4 activation records are important at this point. So, accordingly this display point display has got 4 levels 1 2 3 and 4, level 1 points to M, level 2 points to P 1, 3 point to Q 1 and R 4 points to R 1.

Now, once the compiler knows that, the difference between this current position and this definition is by certain number of access links. So, it can go up or down by so many access links from here and directly come to this. For example, to go two step backward, so is the current level is 4 it will do 4 minus 2, so it will come to this one and accordingly it will be accessing Q 1. So, it does not need to go through this pointers through this Q 1 to reach P 1. So, from R 1 it can immediately reach P 1.

Similarly, after sometime when P R 1 has given a call to P, so, another activation record P 2 is created. Now, you know that P 2s variable, so they are all defined in M. So, if something is not available within P 2 that becomes non-local for P 2 then it needs to look for M. So, the nesting level of P 2 is 2 and nesting level of M is 1. So, if the compiler finds that it has to go by access links, so the difference will be 2 minus 1 equal to 1. So, d will be equal to 1. So, from this it will be just going by one level and it will be coming to M and get it done, get the things.

So, let us try to explain what are writing here. So, what is display? Display is a global array of pointers to activation records indexed by lexical nesting depth. So, this is an array of pointers to activation record and index is lexical nesting depth. So, this M the main routine will have depth i, then the procedures that the based on this static nesting, so they will be having 1 2 3 4 like that.

Element d_i points to the most recent activation of the block at nesting depth i. So, this is the d_i . So, a non-local x will be found like this that if the most closely nested declaration of x is at nesting depth i, then d_i points to the activation record containing the location for X. So, if the most closely nested declaration is at, so is at X is at nesting depth i like here. So, here the for getting the x y etcetera. So, we know that it is at level 2. So, that way it can, so, the X is at nesting depth 2.

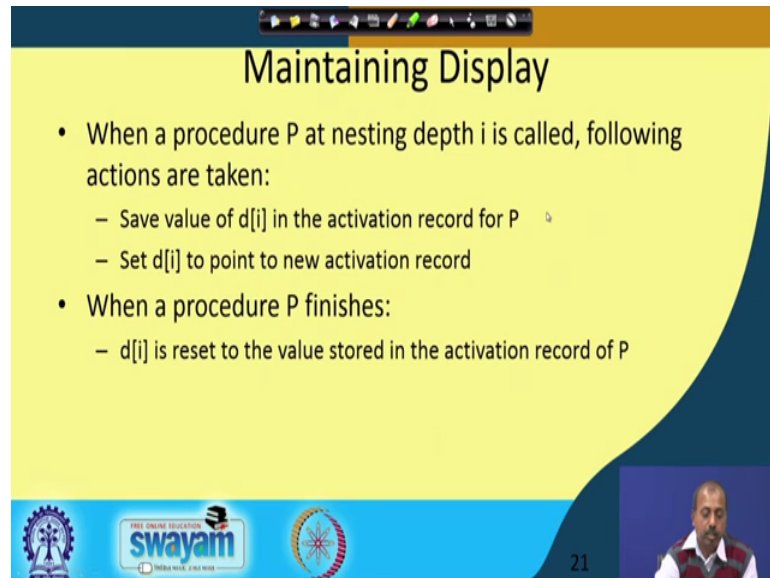
Then d_2 will point to the activation record for the location for X and then we can use relative address within activation record to access X. So, this is basically that frame pointer plus minus for local variables and parameters; so, that is there. So, this says that a to go to a particular depth so you do not need to go traverse by the access links. So, just look into the index of the display and get it come to the access link. So, in this particular example say maximum nesting depth is 4 because there are 4 procedures, so maximum nesting depth is 4. So, we have got 4 entries in the display.

So, in figure a, the first part of the figure M has given a call to P, P has called Q and Q has intern called R. And now if the compiler has if there is if R is referring to x in the code of R if it is referring to x and it has to found out then the compiler knows that the that identifier x is in procedure P at lexical level 2. So, by searching symbol table it knows that x is at level 2. So, it has to go by 2 levels as I was telling. So, this code to generate code is generated to access control entry of the display to reach the activation record of P directly. So, since this is 4 and this is 2, so it will do 4 minus 2, 2, so, it will come here directly.

And here also as I was explaining that this x y z. So, if P is referring to some variable which is not defined in P 2, then this compiler we will know that whether it is defined in M or not if it is so, then it will know that what is the difference between these two levels. So, this is at level 2 and this is at level 1. So, 2 minus 1, 1, so, it will immediately find the index 1. So, that way it can this it can utilize this display, so that we do not need to

traverse by the links we can directly come to the corresponding, we can be directly come to the corresponding activation record.

(Refer Slide Time: 16:52)



The slide is titled "Maintaining Display" and is set against a yellow background with a blue header and footer. The header contains a navigation bar with various icons. The footer includes the Swayam logo, the text "FREE ONLINE EDUCATION", and the number "21". A small video inset of a man is in the bottom right corner.

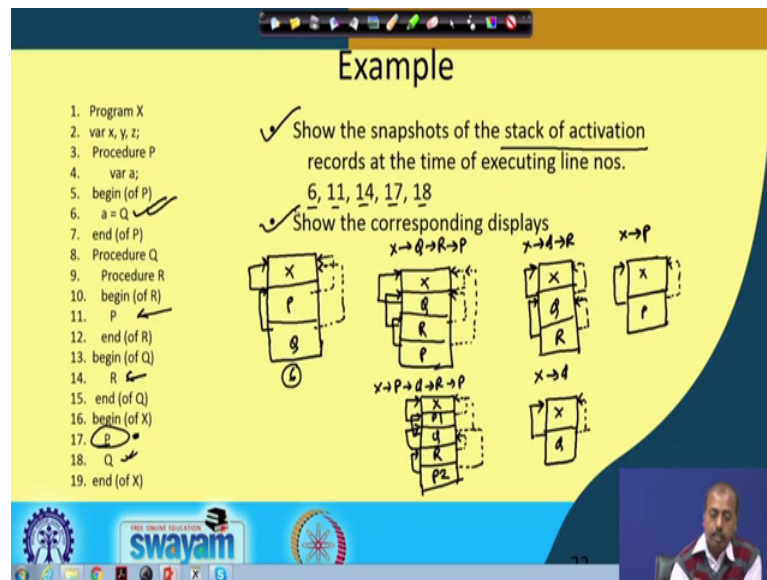
Maintaining Display

- When a procedure P at nesting depth i is called, following actions are taken:
 - Save value of $d[i]$ in the activation record for P
 - Set $d[i]$ to point to new activation record
- When a procedure P finishes:
 - $d[i]$ is reset to the value stored in the activation record of P

But maintaining display is a problem because this compiler has to do something extra for maintaining the displays. At runtime this has to be done so compiler has to generate appropriate code, so that this displays are maintained.

When a procedure P at nesting depth i is called, so, we are doing this actions. Say value of $d[i]$ in the activation record for P and said $d[i]$ to point to new activation record. So, this $d[i]$ will be said in the activation record. So, previous display value will be same. So, that will be because when coming back when we will need to restore the this displaying index value, so that way it is saved there and then $d[i]$ is $d[i]$ points the new activation record. And similarly, when a procedure P finishes $d[i]$ will be reset to the value stored in the activation record for P, so that the display gets restored to its proper value. So, you can just try out these things and try to see like it is really happening like that.

(Refer Slide Time: 17:55)



Next, we will be looking into an example where we have got a set of procedures nested like this, and then we are going to use we are going to see like how this activation records will look like ok. In the first part of the problem, so we will see how this activation records we will look like in this part and then we will be is a going for some display based solution.

So, it says that show the continent the stack of activation records at line number 6, 11, 14, 17 and 18. So, at line number 6 what has happened is, so this P has started and so from the main routine, so from the main routine that is the name of the program is X it has given a call to P and after that it has given a call to Q. So, at this point has given a call to Q, so, what happens at that point ok.

So, a X has given a call to P and P has given a call to Q. So, the activation record it will be like this that I will have this X, P and Q will be the activation records. So, this is the; this is the activation record for X, this will be the activation record for P and this will be the activation record for Q. Now, as per as control links are concerned, so Q will be pointing to P because P has given a called to Q and P will be pointing to X. So, this will be the activation record.

And as far as your access link is concerned this is the control links and the access links you see P and Q. So, they are not, so Q is not nested within P. So, for both of them if you are looking for some variable, so you have to look into X, so, for both of them the access

link will be this X. So, this is the thing and here also this is the situation. So, this is the situation at line number 6.

Now, what about line number 11. So, at line 11, so this one, so what has happened? P has given a call to Q. So, sorry X has given a call to Q, the situation is like this, X has given a call to Q and this Q has given a call to R and R has given a call to P, so, that is the situation. So, X has given a call to Q or. So, this situation may be made more difficult also.

So, let us say suppose this is a situation. So, suppose this P call is somehow over and we are concerned about this Q call, then what will happen, the activation record it will be looking something like this. So, there I will have the portions for X, I will have for Q, I will have for R and P. And for this control link, so they will be like this. So, P has, R has called P. So, it is like this then this R Q has called R and then Q has this X has called Q. So, this is the situation as per as access links are concerned.

And as far as control links are concerned. So, this is the R is nested within Q. So, if you do not find some definition then for R you have to look into Q; so, it is like this. As far as Q is R is pointing to this. As far as Q is concerned, so Q is X, so, Q will be pointing to X and this P will also be pointing to X. P will be also be pointing to X. Only Qs, Rs access link will be pointing to Q ok. So, this is the situation.

Of course, you can make it more complex. So, you can try to create the record for this situation X calling P, P calling Q, Q calling R, R calling P. So, that is also possible. Like if I assume that it has it is from this line it has called P and P has given a call to Q and from Q it has come to R and from R it has come to P. So, the situation will be in that case the situation is something like this.

So, we have got X, then P 1, then Q, then R, then P 2 ok. Now, this the control links are simple control link, so P 2 came from R, R came from Q, Q came from P 1 and P 1 came from X that is the control link. As far as access links are concerned, so this P 1, Q and this P 2 they will point to the X and then R will point to Q. So, that will be the situation for access link.

Now, at line 14, so this point. So, if I assume that the call is like this X to Q to R. X has given a call to Q and Q has given a call to R, then the situation is X, Q and R and now X

is. So, now, the control links will be like this from R to Q and from Q to X, because the call came from X to Q to R, at the access links will be like this because access link of R will be pointing to Q and access link of Q will be pointing to X. So, that is the situation.

At line number 17; at line number 17 so, it has just given a call to P. So, line number 17 the situation is from X, P has been called, so the situation will be simply like this. So, this is for X and this is for P. Now, the control link is like this and access link is also like this ok.

And X to Q, so the line number 18, so it is X to Q. So, if I assume there is X to Q. So, what will happen is that I will have two portions X and Q. So, Q will point to the X for activation for the for the control link and the access link will also be; access link will also be pointing to a X because whatever is defined in X. So, they can be available in Q, so, this way we can do it using control link and this is this access links.

Now, next we will see next we will solve the next part of the problem where it is done using displays. So, how will it look like if we do it using displays?

(Refer Slide Time: 26:06)

Example

1. Program X
2. var x, y, z;
3. Procedure P
4. var a;
5. begin (of P)
6. a = Q;
7. end (of P)
8. Procedure Q
9. Procedure R
10. begin (of R)
11. P
12. end (of R)
13. begin (of Q)
14. R
15. end (of Q)
16. begin (of X)
17. P
18. Q
19. end (of X)

- Show the snapshots of the stack of activation records at the time of executing line nos. 6, 11, 14, 17, 18
- Show the corresponding displays

$x \rightarrow p \rightarrow q$

$x \rightarrow q \rightarrow r \rightarrow p$

$x \rightarrow q \rightarrow r$

The first situation is at line number 6; so, that is X to P to Q. So, X to P to Q, so we have got this activation record X P Q. And now since there are 4 procedure, so the display will have 4 entries in it. So, display we will have 4 entries, in it 1, 2, 3, 4 and then this, so the currently valid once are this X and Q, these two are only valid once. So, it will be

because if it if I do not get something in Q have to look in X. So, rest of the entries are not valid so, it will be like this.

Similarly, at line 11, so if I say that the call is like this that X to Q to P to R. So, if the situation is like this then I will have the situation. The activation record will be like this X.

Student: (Refer Time: 27:14).

Q, P R I am not drawing the control link, so that you can at line 11 you know, so this is sorry this is R and this is P. So, this is R this is P and then I have got this display 1 2 3 4. Here, this 1 points to X and then this 2 it will point to P, because if I am looking if P is the currently valid 1 and if I do not find something in P I have to look in X only.

Then this one, so X to Q to R so if you looking for this situation then the display will be sorry the activation record will have 3 entries X, Q and R, my display has got 4 entries in it 1 2 3 4 then X is there and R is defined within Q. So, this 2 will be pointing to this Q and 3 will be pointing to R ok. So, this way we can draw the you can make the displays, and that shows the how this displays are going to be updated and all.

(Refer Slide Time: 28:52)

Conclusion

- Data structure activation record contains necessary information to control program execution
- Compiler writer must generate appropriate code for operations
- A small array, display, helps in the process
- Display management becomes a part of compiler's responsibility

swayam

Now, so the next we will come to the conclusion part. So, we have seen that this data structure activation record it contains necessary information to control program execution and compiler writer has to generate appropriate code for operations and a

small array called display it helps in this runtime environment management process, and display management becomes a part of compilers responsibility. So, the if the display is introduced then this management code also has to be put into the code that is generated.