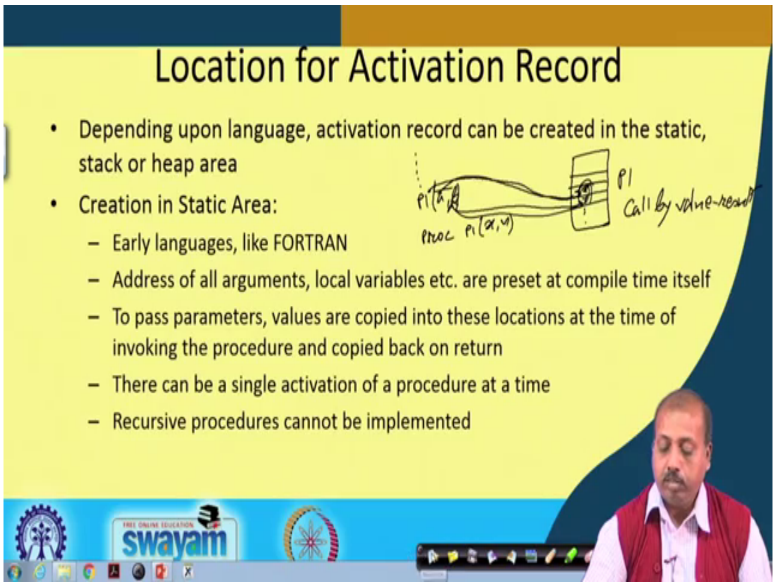
Compiler Design
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 47
Runtime Environment (Contd.)

(Refer Slide Time: 00:17)



Location for Activation Record

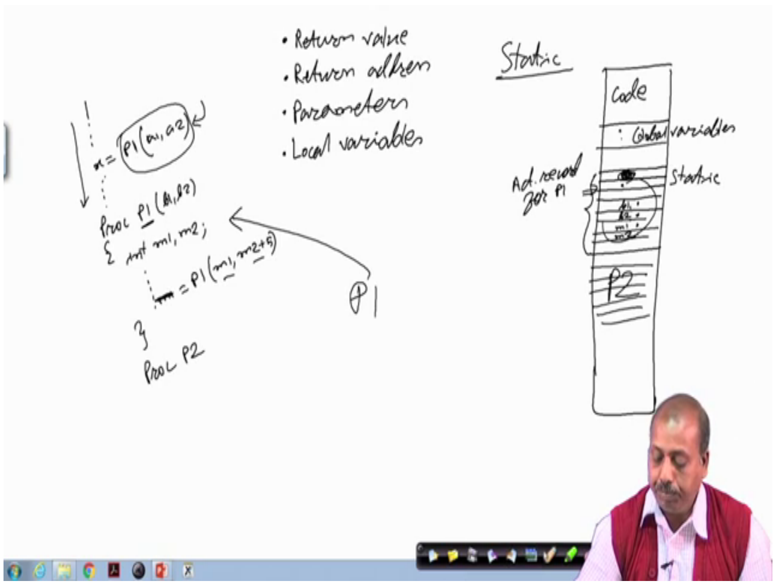
- Depending upon language, activation record can be created in the static, stack or heap area
- Creation in Static Area:
  - Early languages, like FORTRAN
  - Address of all arguments, local variables etc. are preset at compile time itself
  - To pass parameters, values are copied into these locations at the time of invoking the procedure and copied back on return
  - There can be a single activation of a procedure at a time
  - Recursive procedures cannot be implemented

So, location for activation record. So, where do you create this activation record? So, that is a question. Now, depending upon the language that you have, because the activation record can be created in the static stack or heap area. So, these are the three options that we have, where are you going to create this activation record. So, if it is created in the static area that is the global area that we have so, I like you we have seen that a program can have the global variable area.

So, where this, that is called the static area. So, that can apart from the static the global variable. So, it can also have this activation records the area for activation records. Now, this early programming languages like FORTRAN, they used to have this activation record created in the static area. So, so this address of all arguments local variables etcetera or p set at compile time itself and two pass the parameters values are copied into this locations at the time of invoking, the procedure and copied back on return.

So, there can be single activation of a procedure at a time and recursive procedures are cannot be implemented. So, let us try to understand what do you mean by that.

(Refer Slide Time: 01:31)



So, I said that activation record it has got so, it has got the entries like this return value, then return address, then the parameters that you are passing and the local variables local variables.

Now, what this static allocation we will do. So, we are looking to the static allocation what the static allocation we will do. So, it will fix up some some entries ok, like if in a if I got a program where at some point of time I am giving a call two procedure P 1. So, this is say x equal to say P 1 and then I am passing the values like say a 1 and a 2 has two arguments for this. And then this is the procedure P 1 and there I have got the local variable say m 1 and m 2.

Now, so if this is the code that is generated, there is this is the translated version of the object code that is generated. So, with the some part of it is corresponding to the code of the program and say this part of this file the object file is corresponding to the static. So, this is called the static location that is the global variables are allocated here.

So, some part of it will be occupied by the global variables and some part may be allocated for this activation records, like by analyzing this program you know that at this point this procedure P 1 is going to be called so, we your mark a portion within this static area to act as the activation record for P 1.

So, this part is your marked to be the activation record for P 1 that is whenever in my program it is any from any point in the program, if I am going to make a call to P 1 procedure then these entries will be filled up here.

So, the first slot maybe for the return value second slot may be for the return address , then there are two parameters that I am passing so, this is for a so, this has got two values say a 1 a 2 let us call them b 1 b 2. So, this is the space for b 1 this is the space for b 2 and this is the space for m 1 and m 2. So, like that it has got so this part of this memory so, it is your marked two act as the activation record for P 1.

Now, what the compiler we will do at this point when it is generating the code for this procedure call so, it will have the code such that this e 1 is copied into this location a 2 value is copied into this location fine, then the return address is saved on so, the return address is return this space is reserve for return value this space is saved for the return address. So, this way we can do this thing then this m 1 and m 2 so, they are for the local variable. So, when it comes to the local when it comes to this code. So, when it is doing a translation of this m 1 some modification of this m 1 m 2 so, they will be effecting these locations m 1 m 2 like that.

Now on return so, what will happen it knows the compiler knows that the return value is available at this at this particular address. So, it will be coping it to x director. So, for this the for this call the return value will be available at this location for. So, for this x equal to this P 1 etcetera so, it will be copying the content from this location and putting it into x. So, that is how this static things we will work ok.

Now, for every procedure that you have in your program. So, is there maybe another procedure P 2 in the program. So, there will be another such frame created in the static area which will correspond to P 2. So, for every procedure called they have a for every procedure that we have in your system so, there will be a there, there will be an activation record created. And whenever a call is made to that particular procedure this record will be filled up by the calling procedure, and at the manipulations will be done on this variables on this on this activation record only.

And after coming back, this return value is again available in the activation record. So, from there it will be taken. So, this is how the static thing will work, now the problem that we have is that see we cannot have more than one activation of P 1 simultaneously, because in that case if there is an somebody else from some other place is also allowed to call P 1, when this call is active. Then what will happen is that this P 1 will get activation record will get overwritten, as a result the execution will become erroneous.

So, apparently it seems that why they should happen because ultimately we have got a single processor so, as a result. So, there can be only one call at a time, but there can be situation like this P 1 maybe a recursive procedure. So, within this their this P 1 maybe giving a call two P 1 itself with some parameter say m 1 and m 2 plus 5. Suppose it does something like this then what will happen so, then this m 1 and m 2 plus 5 these two values will be copied on to b 1 and b 2. So, previous b 1 b 2 contains will be lost. So, it cannot return to the proper place so, this all calculations will be lost.

So, that is why this static type of organization. So, of this activation record though it is very simple in the sense that, we know the size of this static section ok. So, it is very simple, but it cannot it cannot handle complex situations by like say recursion. So, this is say a recursive is a P 1 being a recursive procedure. So, it cannot handle that situation. So, for the static allocation of activation record. So, it only one active only one procedure call should be active at a time.

So, coming back to the point that we were discussing. So, it was there in the early programming languages like FORTRAN ok. So, this is being FORTRAN you do not have recursion. So, there it was possible so, address of all arguments local variables etcetera are they are computed at the compile time itself and their we fix some locations for them and the code is generated such that, whenever a procedure call is made the code is generated such that the parameters that you are passing. So, they will be copied onto this activation record space.

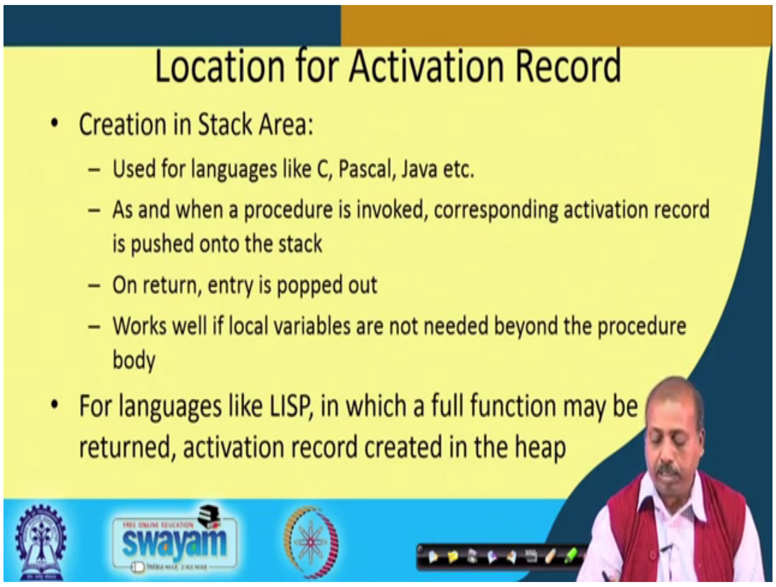
And after it is over so, they will be copied back from this activation record to the actual variables. So, that way if there is a change in the values of those variables within the execution of the procedure. So, that will get reflected. So, what I mean is that, suppose I have got this as the activation record for procedure P 1, now I am giving a call to this procedure P 2 at this point in the main program so, I am giving a call to the procedure P 1 with the values like a and b.

So, whatever arguments we have so, this values will be copied there. So, this value of a will be copied into if this P 1 has the procedure P 1 has got the arguments x and y suppose this is for x and this is for y.

So, at the beginning the value of a will be copied on to x and value of b will be copied into y. And once the procedure is over then this x and y values are now modified due to execution of this P 1 and then this x value will be copied back to a and this y value will be copied back to b so, this is a type of parameter passing which is known as call by value result. This is call by value results at the time of calling the procedure the values are copy to the parameters and at the time of returning the parameters are copied the final values of the parameter so, they are copied back to their original variables.

So, this way so, they be copied back on detail. So, this is call by value result type of parameter passing. And as I discussed previously so, there can be a single activation of a procedure at a time, because multiple activation we will destroy the activation of this activation record content of the previous call and so, and recursive procedures definitely cannot be implemented ok.

(Refer Slide Time: 10:57)



Location for Activation Record

- Creation in Stack Area:
  - Used for languages like C, Pascal, Java etc.
  - As and when a procedure is invoked, corresponding activation record is pushed onto the stack
  - On return, entry is popped out
  - Works well if local variables are not needed beyond the procedure body
- For languages like LISP, in which a full function may be returned, activation record created in the heap

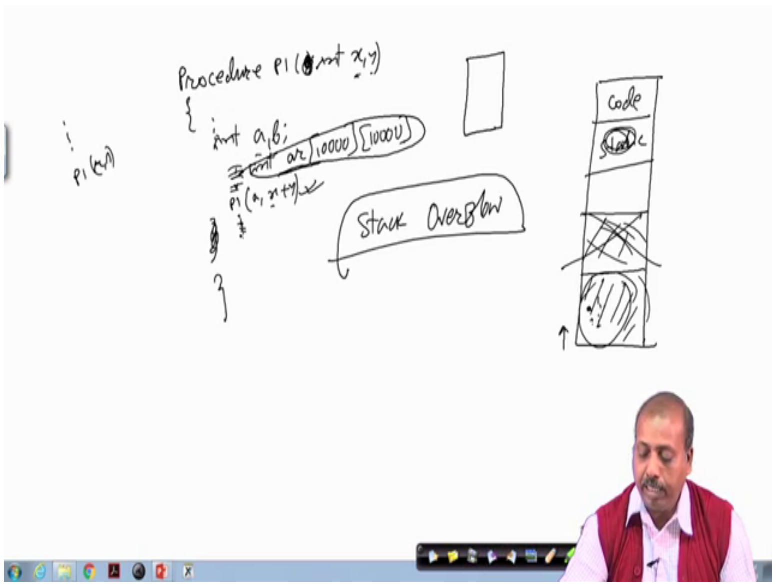
So, next we will be looking into other possible location for this activation record. So, on the stack area so, can we create this activation record on the stack area.

So, it is used in most of the programming languages like C Pascal java etcetera say they do create this activation record in the static area in the stack area. So, as and when a procedure is invoked corresponding activation record is pushed in to the stacks the activation records structure remain same as we have seen previously.

So, it has got the return value return address then parameters and local variables, but they are not created in the static area, they are not created in the along with the global variables. Rather the when this procedure call will be made one such frame will be pushed into the stack. And then this the code that we will have within the procedure when it is referring to those parameters and local variable so, they will be doing it from the stack.

And when it is returning this entry will be popped out. So, that will go out of the stack. So, this works well if local variables are not needed beyond the procedure body.

(Refer Slide Time: 12:15)



So, it is like this what I mean is what I mean is that so, in the previous example. So, we have got this procedure P 1 and it has got say arguments that is say int x and y and then within that I have got some local variables a b etcetera.

So, this activation record structure is fixed as we had previously the return address then x y a b etcetera, but they are not created on the global are or the static area rather. So, as you remember that this program snapshot that we talked about so, it has got the code part, it has got this static part and it has got this heap and stack simultaneously. So, this heap the stack starts from the bottom. So, when a call will be made to the procedure so, this if this is the main routine at some point of time you have given a call to P 1 with say value say r and s, then this activation record will be created onto the stack.

So, this activation record is pushed into the stack and here I have got this r and s copy that x and y then this space for ab etcetera. And this code whatever we have. So, they will be referring to the locations that we have in the stack so, the that way the code will be generated. Now, after sometime when this P 1 is over then what we will do we will just pop out this block pop out this frame from the stack.

So, after this procedure is over this local variables a b or this parameters arguments xy so, they are not visible because they are already deleted, they do not have any existence after this. So, if you do not need them then it is fine. So, you can do it on the then you can do it in this fashion but of course, the what is the biggest advantage of this the biggest advantage is that you can support recursion now. Because now if there is a call to P 1 at this point with the parameter say a and x plus y then you can so, this frame is there in the stack.

So, you can push another frame here which will be corresponding to the activation for this one. And when this call will be over so, this record will be popped out, but this record will remains in the stack. So, that this previous call. So, whatever statements you have after this whatever the statement you have after this. So, they will be executing properly referring to this locations that we have in this part ok. So, that way this recursive procedure calls can be handled very efficiently, if we have got this stack type of realization.

However there are problems we had some places, because one very serious problem that you have is that this local variables that we have so, they are created in the stack. And you really do not know how many times this procedure P 1 will be called recursively.

Now, as I said that is local variable so, they are also created in the stack. Now, suppose I have got this local we have got an array here say that is also an integer array say, but its dimension is very high say 10000 by 10000, then what will happen. So, this array we will get created onto the stack itself. So, this your activation record will now be very large, because it has to accommodate such a big array in it, such a large number of integers in it. So, that will be a very big array. So, this activation record will be very large and the situation will be even worlds. So, if this P 1 happens to be recursive function because now recursive procedure, because now every activation we will have one such large set of integers created as local variables.

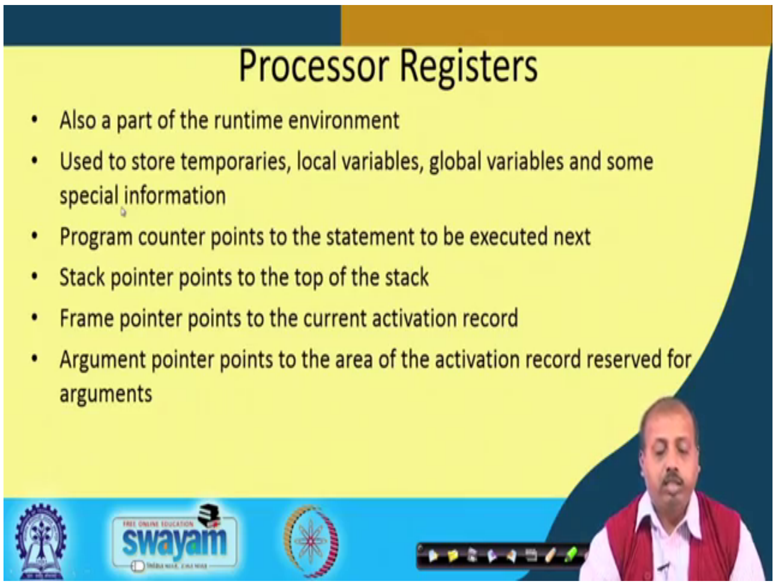
So, as a result what can happen is that this program while executing so, it can be running out of memory ok. So, this there are some errors called stack overflow type of errors. So, that can occur and the program may misbehave ok. So, to what is the solution, solution is that you do not make this arrays local ok. So, you make this array global variable so, that this the array will come to the static area and they will not be created with every activation of this procedure.

So, though it may or may not solve your purpose like, if you need that for every activation of the procedure I should have a new array, then this is this does not provide any solution, but may a time we will just do it carelessly so, we just make large local arrays so, it is advisable that we do not do that because if you do this large local array. So, they will be created on to the stack and as a result they will if there is particularly if it is a recursive function or recursive procedure, then it will be invoked again and again so, there will be large space occupied by this local arrays. So, there may there is a chance of stack over flow.

So, we can we can just try to solve this problem by having this local arrays made global. So, putting them outside and putting them to the static part of the code. So, so, this is good because we can have this recursion part implemented very easily, you can have this recursion part implemented very easily. So, for languages like less when which is a full function may be return so, activation record is created on the heap.

So, this is another situation where. So, the here I was talking about creation in the stack some programming language like list. So, it will create it will return day full function as a return value of one function like that. So, then in the those cases activation record will be created in the heap, because then how many parameters they it has a dynamically allocate variables and all so, that can be done only from the heap. So, the in that case activation record will be created in the heap ok.

(Refer Slide Time: 18:33)



Processor Registers

- Also a part of the runtime environment
- Used to store temporaries, local variables, global variables and some special information
- Program counter points to the statement to be executed next
- Stack pointer points to the top of the stack
- Frame pointer points to the current activation record
- Argument pointer points to the area of the activation record reserved for arguments

Now, processor registers so, they are also part of the runtime environment. So, proceed because when a program is executing so, it has got the, it is a processor registers are holding some of the variable values and all. So, that is also part of the runtime environment.

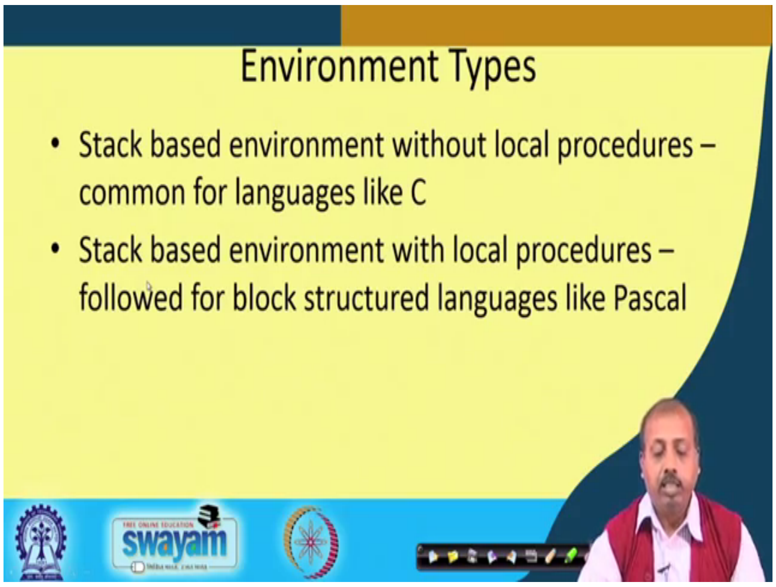
So, they are used to store temporary is local variables global variables and some the special information. So, these are the purpose of this processor registers program counter is another very important register in the processor that is that points to the statement to be executed next. So, that is also a part of the runtime environment then stack pointer.

So, that is also that is also a processor register. So, that is also used to point to the top of the stack. So, that is also part of the runtime environment there is a frame pointer which points to the current activation record. So, some of the processes they will allow this frame pointer to be available. So, you can as a separate register. So, you can use it for pointing to the current activation record.

Some processes will not allow this thing ok. So, in that case you have to dedicate some general purpose register to work as the frame pointer register. And there is a argument pointer that points the area of the activation record reserve for arguments. So, these are the threes supporting pointers that are needed from for supporting this is a dynamic activation records one is the stack pointer, one is frame pointer another is argument pointer. And as I said that we may or may not be possible that we may or may not have all this pointer registers available.

And many a times so, what we have to do is that we have to use some general purpose register to act as the as this registers but; however, they also form part of the runtime environment. So, they are also to be taken care of.

(Refer Slide Time: 20:33)



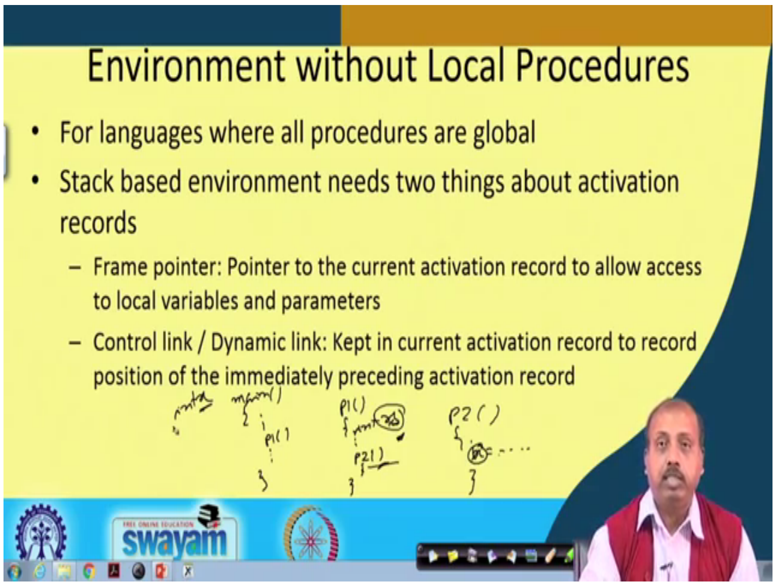
Environment Types

- Stack based environment without local procedures – common for languages like C
- Stack based environment with local procedures – followed for block structured languages like Pascal

Now, if you look into the environment types there can be two type of situation that you can find out, one is tag based environment without local procedures. So, where this local procedures are not supported and we can have stack based environment with local procedures. So, without local procedure as I was telling previously. So, with so we do not have nested procedure definition.

But with local procedure means we have got nested procedure definition. So, they have got so, they have followed in some block structured language like Pascal. So, we will find this nested procedure definition whereas, if we look into languages like C nested callas are allowed, but nested definitions are not allowed. So, we will have to see like how can we handle this situation both of them in grand time environment.

(Refer Slide Time: 21:29)



Environment without Local Procedures

- For languages where all procedures are global
- Stack based environment needs two things about activation records
  - Frame pointer: Pointer to the current activation record to allow access to local variables and parameters
  - Control link / Dynamic link: Kept in current activation record to record position of the immediately preceding activation record

First we look into the simpler version that is where you do not have the local procedures. So, all so all the procedures that we have so, they are globally in nature. So, there is you cannot have a one procedure definition nested within another procedure. So, all procedures are global and stack based environment will need two things one is frame pointer another is control link or dynamic link.

So, in this case what happens is that we have got so, this. So, we have got two things one is the frame point and that point to the current activation record to allow access to the local variables and parameters. And there is a control link or dynamic link that is kept in current activation record to the position of immediately preceding activation record.

So, you will see some example basically what happens is that say you have got say one procedure say from the main program, I have given a call to procedure P 1. Now, within procedure P 1 there is a call to procedure P 2 fine, now while we are executing procedure P 2 now which definitions are we going to use ok.

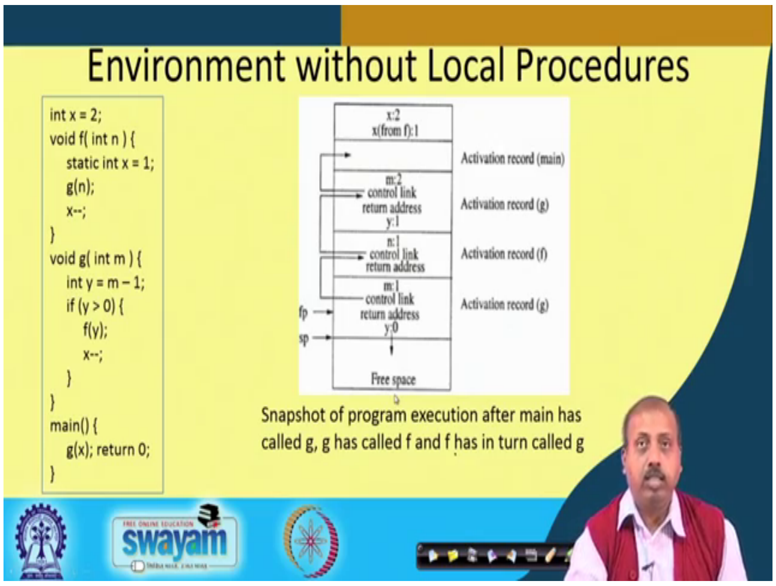
So, if we are if you so if this is the body of P 2, then what are the definitions that we are going to use. Now it so, depending upon the scope rules of the language. So, it may be that it is a static scoping so, that is we are so, P 2 where from which from which portion of the procedure we should take the definition that they are fixed.

So, that is defined by the static nesting nest nesting rules or it may be its a dynamic nesting rule. So, since at this point of time P 2 has come through P 1. So, if there is a variable x so, it may so happened that here it is referring to an x equal to something and I have got a global x and I have got another x which is here in P in P 1.

The when I am referring to this x so, is it referring to this one or this one so, that is the question. So, now if so if it is dynamic situation dynamic scoping rule, then it can tell me which x to follow. So, there I can have in the activation record one control link. So, that can tell what is the preceding activation record so, so that way I will first search in the activation record of P 1 and getting x there so, when I am writing x equal to so, it will refer to this x.

On the other hand if I say that P 2s dynamic link so, it is pointing to the global activation so, this the global the table in that case. So, it will be referring to this x. So, based on this activation record. So, we can with this control link we can find out like which variable definitions are going to be used. So, we will take some example and then try to explain.

(Refer Slide Time: 24:49)



Environment without Local Procedures

```
int x = 2;
void f( int n ) {
    static int x = 1;
    g(n);
    x--;
}
void g( int m ) {
    int y = m – 1;
    if (y > 0) {
        f(y);
        x--;
    }
}
main() {
    g(x); return 0;
}
```

Snapshot of program execution after main has called g, g has called f and f has in turn called g

Say this is the situation we have got this gives the definition like we have got the functions f and g and from the main program there is a call to g. And so, this is so this is the situation so, this is the activation record for main.

So, main does not have any local variable. So, there is nothing here and that the local part so, if nothing we cannot understand much here, but this x is a global variable so this x is coming as the static part. So, this is in the static this is in the static area ok.

Now, this activation record and then there is another x here so, in within function f which is called static int x 1. So, this is this is another x. So, this x and this x they are not same. So, this is a global variable and this is another x coming from the function f so, this x from f. So, that is equal to 1 so that is the situation. And then so this is the snapshot of the program after main has called g, g has called f and f has intern called g.

So, main has called g, g has called g has called f and f has called g from there. So, from here so, we have got this one this m so, we have got from f to g. So, the first call gx y equal to m minus 1 so, this m is getting the value so 2 has been passed. So, y is equal to 2 minus 1 that is 1 so, m is equal to 2 so, that is the parameter that I have passed, then we have got the control link return address etcetera and this is the local variable y so, y equal to 1. So, that is the situation and then there is a call to f of y. So, this function has been called so, this activation record is pushed into the stack now ok.

So, previous to we had add up to this up at the main routine, then it at called g. So, up to this much is created in the activation record. And then now this f has been called so, this is the activation record for f that has been pushed. And in the activation record of f say again we have got this parameter n. So, n value of n is equal to 1. And then because 1 we have passed 1 there so, 1 is passed there and then the this is the static part static int x is the static. So, it has gone to the global part global section and this control link and details I will come to control link later.

Then from here it has given a call to the 0 routine. So, from after this is the activation record for g is created and pushed into the stack. And then this y equal to m minus 1. So, y y becomes equal to 0 so, m was equal to 1 so y becomes equal to 0 so, this is the situation. Now, this control links they are extremely pointing to the previous activation record that we have so, so from main when I was calling to call to g.

So, control link was pointing to main similarly from g when I was considering so, g even from g I given a call to f so, from f the control link is pointing to the 0 g function. And from here from g the control routine is pointing to the previous 1 f.

So, this is the situation now the situation may change so, as we go into further calls and that may complicate the situation. So, here it is very simple. So, here you can say that the control links are not necessary, because I can just pop out the entries from the stack and that will be going absolutely fined this happens here, but in more complex situation that may not be the case ok. So, we will see that in successive examples.