Run-Time Environments

Reading: Section 7.1 – 7.3 of chapter 7

How storage organizations used to support the run-time environment of a program

Source Language Issues

How do we allocate the space for the generated target code and the data object of our source programs?

The places of the data objects that can be determined at compile time will be *allocated statically*.

But the places for the some of data objects will be *allocated at run-time*.

The allocation of de-allocation of the data objects is managed by the *run-time support package*.

- run-time support package is loaded together with the generate target code.
- the structure of the run-time support package depends on the semantics of the programming language (especially the semantics of procedures in that language).

Each execution of a procedure is called as *activation of that procedure*.

Procedure Activation and Lifetime

A procedure is *activated* when called

The *lifetime* of an activation of a procedure is the sequence of steps between the first and last steps in the execution of the procedure body

A procedure is *recursive* if a new activation can begin before an earlier activation of the same procedure has ended

Procedure Activations

Example

```
program sort(input, output)
   var a : array [0..10] of integer;
   procedure readarray;
      var i : integer;
      begin
         for i := 1 to 9 do read(a[i])
      end;
   function partition(y, z : integer) :
integer
      var i, j, x, v : integer;
      begin …
      end
   procedure quicksort(m, n : integer);
      var i : integer;
      begin
         if (n > m) then begin
            i := partition(m, n);
            quicksort(m, i - 1);
            quicksort(i + 1, n)
         end
      end;
   begin
      a[0] := -9999; a[10] := 9999;
      readarray;
      quicksort(1, 9)
   end.
```

Activations:

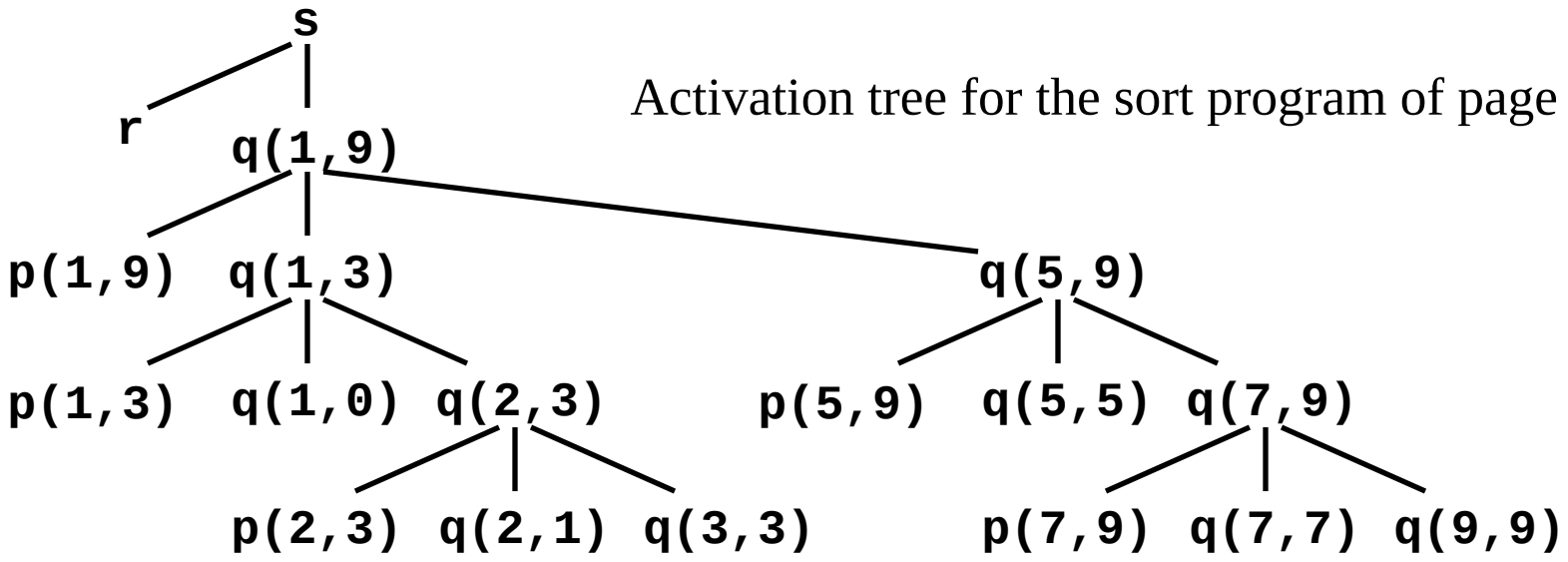
```
begin sort
 enter readarray
 leave readarray
 enter quicksort(1,9)
  enter partition(1,9)
  leave partition(1,9)
  enter quicksort(1,3)
   …
  leave quicksort(1,3)
  enter quicksort(5,9)
  …
  leave quicksort(5,9)
 leave quicksort(1,9)
end sort.
```

Activation Tree

We can use a tree (called **activation tree**) to show the way control enters and leaves activations.

In an activation tree:

- Each node represents an activation of a procedure.
- The root represents the activation of the main program.
- The node a is a parent of the node b iff the control flows from a to b.
- The node a is left to to the node b iff the lifetime of a occurs before the lifetime of b.



Activation tree for the sort program of page 4

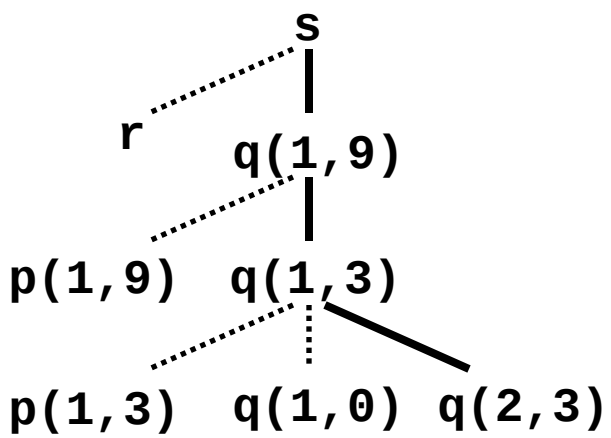
Control Stack

The flow of the control in a program corresponds to a depth-first traversal of the activation tree.

A stack (called **control stack**) can be used to keep track of live procedure activations. An activation record is pushed onto the control stack as the activation starts and popped when that activation ends.

When node n is at the top of the control stack, the stack contains the nodes along the path from n to the root.

Activation tree:



Control stack:

s
q(1,9)
q(1,3)
q(2,3)

Activations:

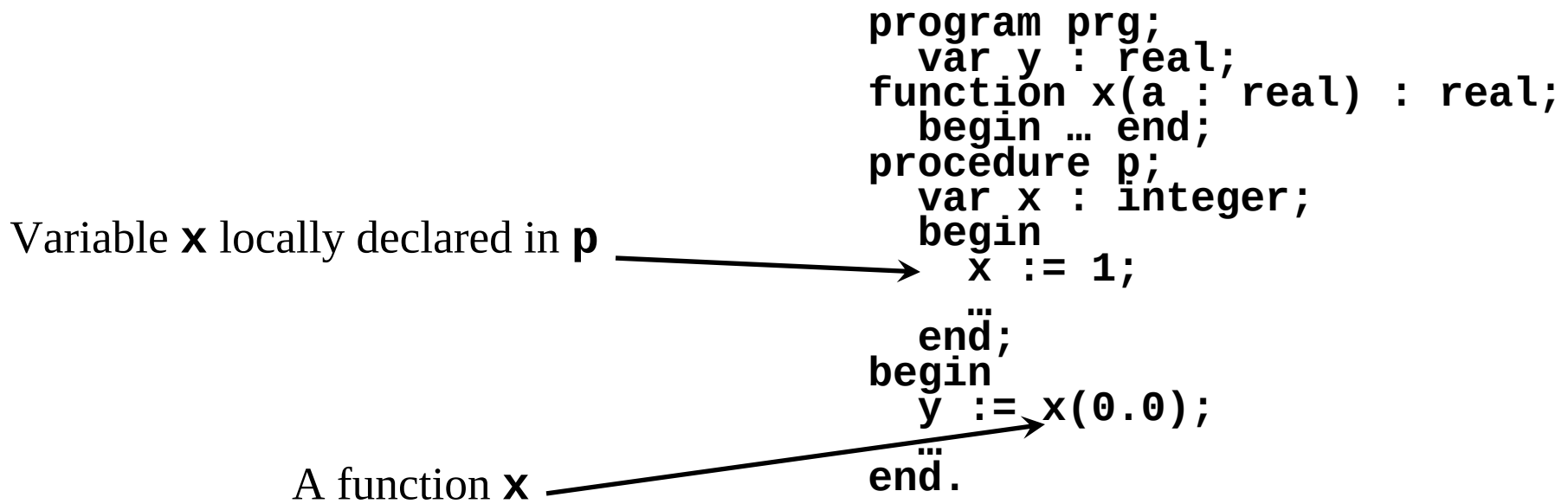
```
begin sort
 enter readarray
 leave readarray
 enter quicksort(1,9)
  enter partition(1,9)
  leave partition(1,9)
  enter quicksort(1,3)
   enter partition(1,3)
   leave partition(1,3)
   enter quicksort(1,0)
   leave quicksort(1,0)
   enter quicksort(2,3)
   …
```

Scope Rules

The same variable name can be used in the different parts of the program. The scope rules of the language determine which declaration of a name applies when the name appears in the program.

An occurrence of a variable (a name) is:

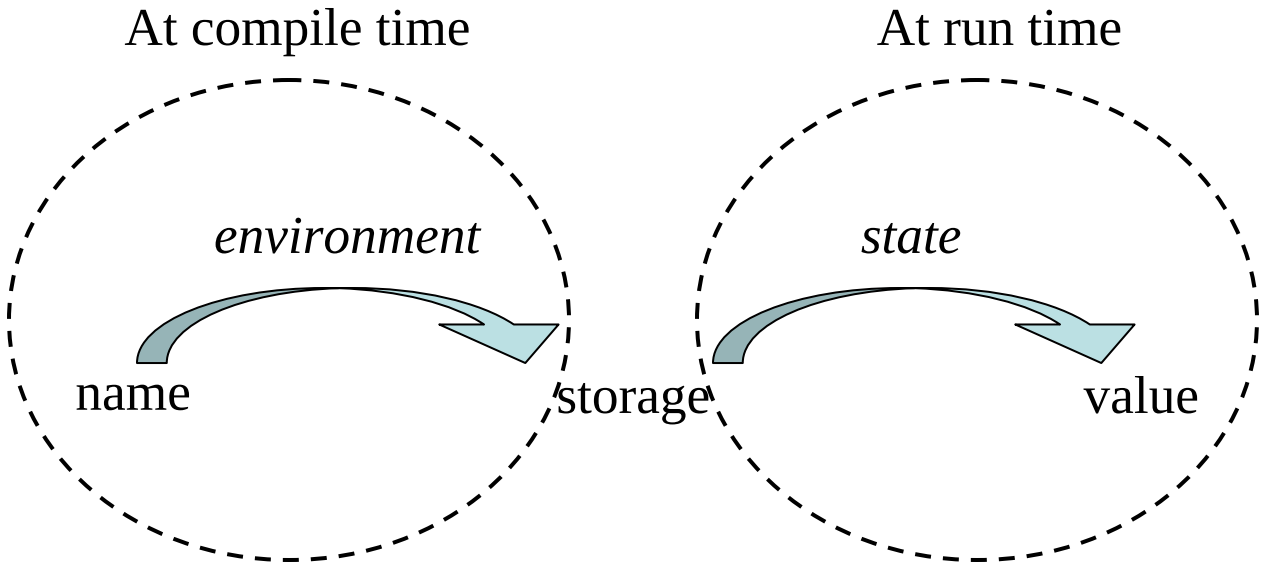
- **local**: If that occurrence is in the same procedure in which that name is declared.
- **non-local**: Otherwise (ie. it is declared outside of that procedure)



Binding of Names

The same name may denote different data objects at run time.

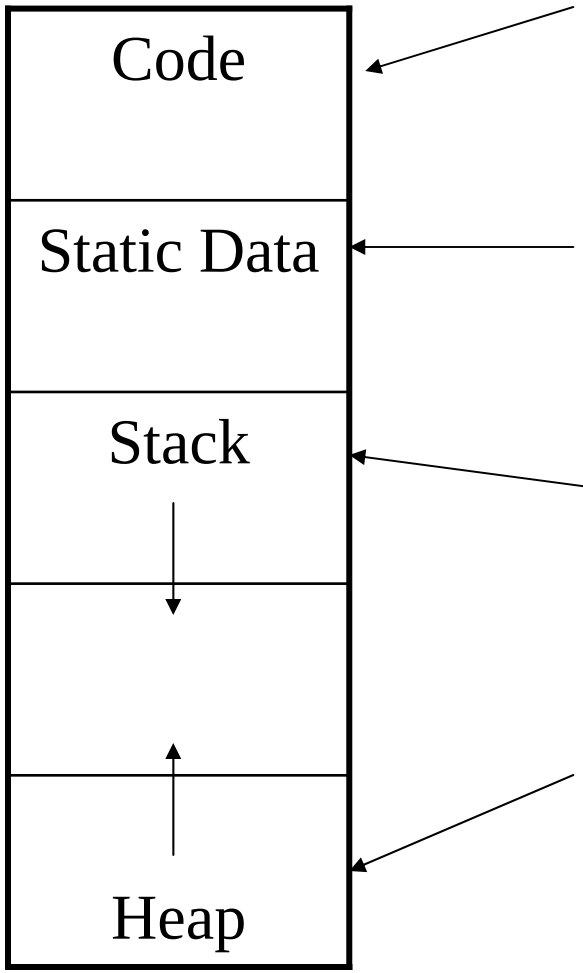
Storage location s with name x, we say that x is bound to s; the association is referred to as a binding of x.




```
var i;
…
i := 0;
…
i := i + 1;
```

A binding is the dynamic counterpart of a declaration

Run-Time Storage Organization



Memory locations for code are determined at compile time.

Locations of static data can also be determined at compile time.

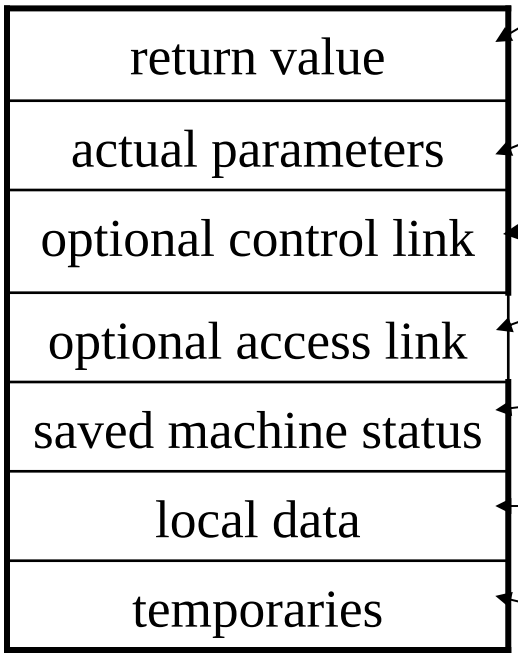
Data objects allocated at run-time. (Activation Records)

Other dynamically allocated data objects at run-time. (For example, malloc area in C).

Activation Records

Information needed by a single execution of a procedure is managed using a contiguous block of storage called **activation record**.

An activation record is allocated when a procedure is entered, and it is de-allocated when that procedure exited.



The returned value of the called procedure is returned in this field to the calling procedure. This value return in machine register for efficiency..

The field for actual parameters is used by the calling procedure to supply parameters to the called procedure.

The control link points to the activation record of the caller.

The access link is used to refer to nonlocal data held in other activation records.

The field for saved machine status holds information about the state of the machine before the procedure is called.

The field of local data holds data that local to an execution of a procedure..

Temporary variables is stored in the field of temporaries.

Creation of An Activation Record

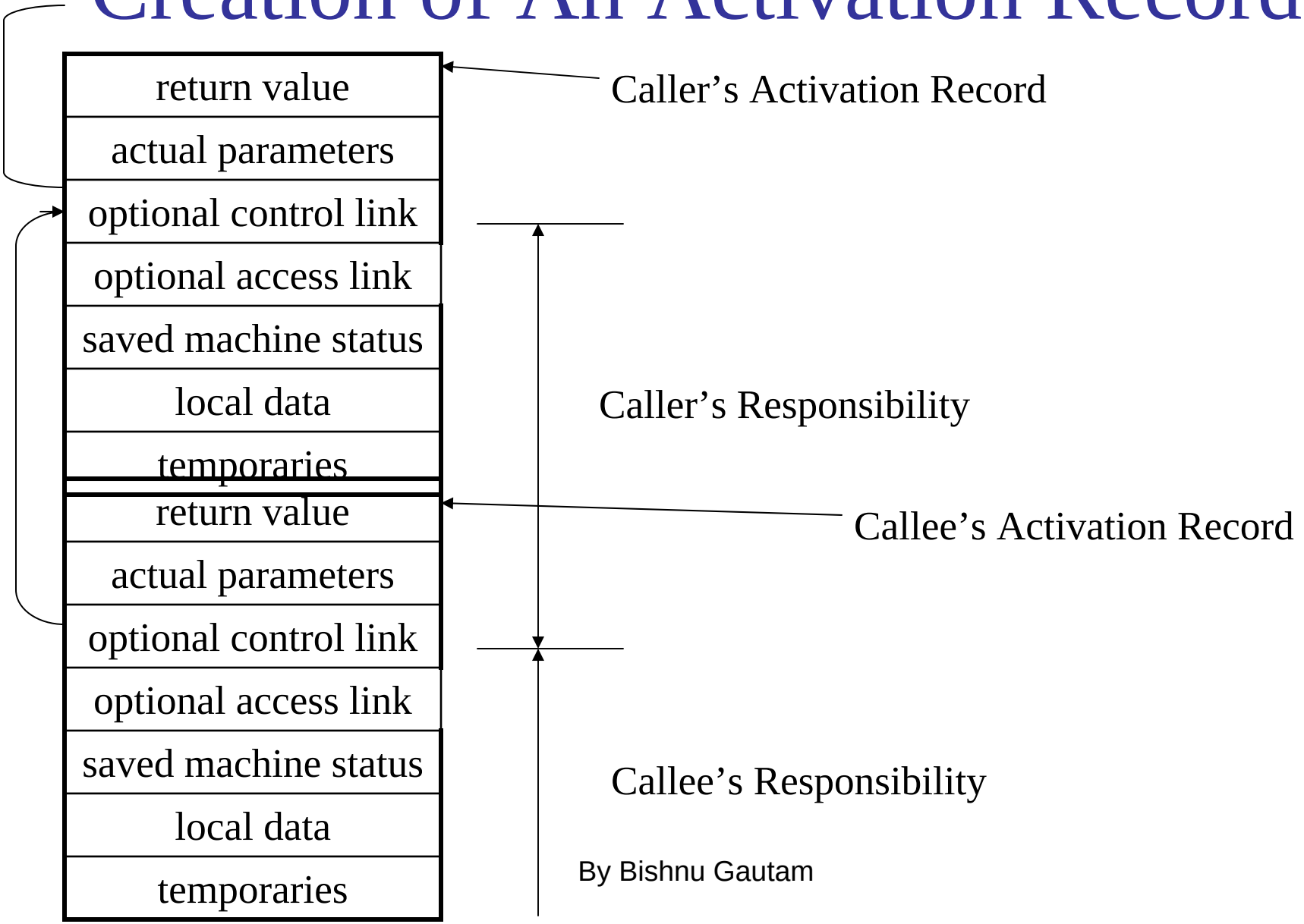
Who allocates an activation record of a procedure?

- Some part of the activation record of a procedure is created by that procedure immediately after that procedure is entered.
- Some part is created by the caller of that procedure before that procedure is entered.

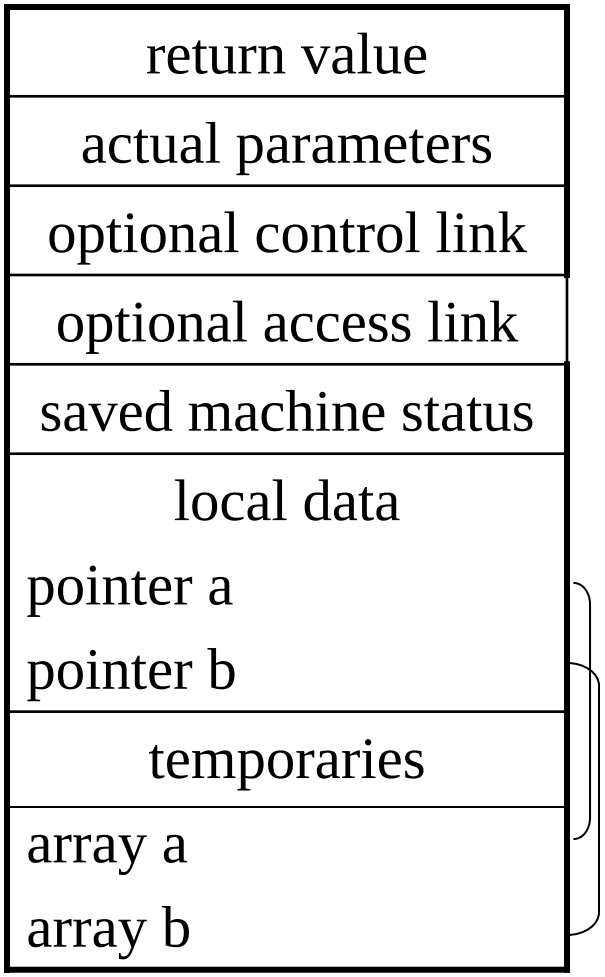
Calling sequences are code statements to create activations records on the stack and enter data in them

- Caller's calling sequence enters actual arguments, control link, access link, and saved machine state
- Callee's calling sequence initializes local data
- Callee's return sequence enters return value
- Caller's return sequence removes activation record

Creation of An Activation Record



Variable Length Data



The storage of variable length is not the part of the activation record, only a pointer to the beginning of each array appears in the activation record.

Exercise

Q.7.1 and Q. 7.2 from text book