Course Code : CS 304

Course Name : Compiler Design

Prepared by : Vineetha K V

Assistant Professor, MEC

Reference : Aho A. Ravi Sethi and D Ullman. Compilers – Principles Techniques and Tools, Addison Wesley, 2006.

MODULE 5

Run-Time Environments:

Source Language issues, Storage organization, Storage-allocation strategies.

Intermediate Code Generation (ICG):

Intermediate languages – Graphical representations, Three-Address code, Quadruples, Triples. Assignment statements, Boolean expressions.

COURSE OUTCOME

At the end students may able to identify different storage allocation strategies and generate intermediate code for programming constructs.

PART I

RUN – TIME ENVIRONMENT

Run – Time Environment

A compiler must implement the abstractions embodied in the source language definition such as names, scopes, bindings, data types, operators, procedures, parameters, and flow-of-control constructs.

The compiler must co- operate with the operating system and other systems software to support these abstractions on the target machine. To do so, the compiler creates and manages a ‘run-time environment’ in which it assumes its target programs are being executed.

Run – Time Environment

Run time environment deals with a variety of issues such as

- the layout and allocation of storage locations for variables used in the source program
- the mechanisms used by the target program to access variables
- the linkages between procedures
- the mechanisms for passing parameters, and
- the interfaces to the operating system, input/output devices, and other programs.

Storage Organizations

- From the perspective of the Compiler the executing target program runs in its own logical address space in which each program value has a location.
- The management and organization of this logical address space is shared between the compiler, operating system, and target machine.
- The operating system maps the logical addresses into physical addresses.
- The layout and allocation of data to memory locations in the run-time environment are the key issues in storage management.

Storage Organizations

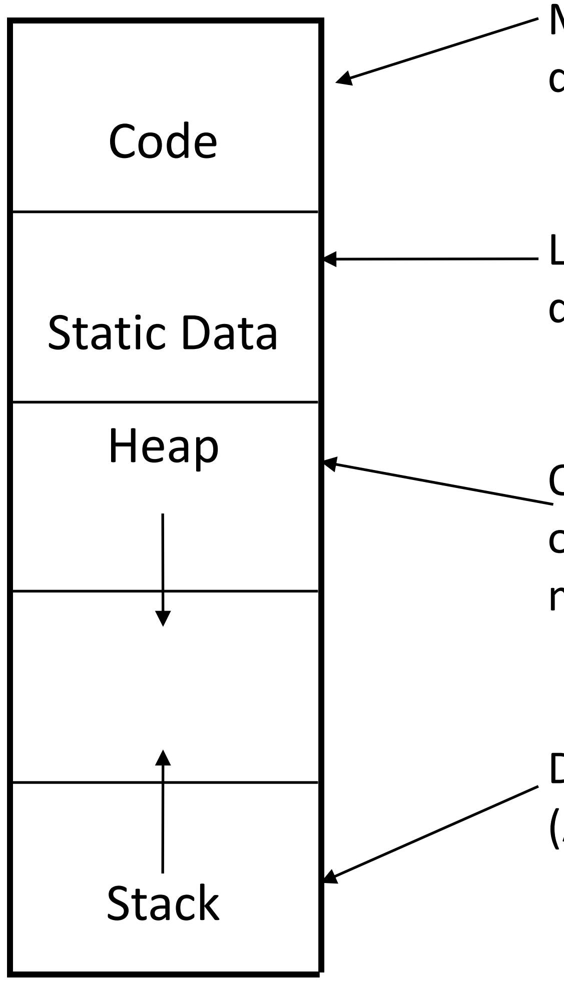
How do we allocate the space for the generated target code and the data object of our source programs?

- The size of the generated code is fixed at compile time, so the compiler can place the executable target code in a statically determined area Code.
- The storage layout for data objects is influenced by the addressing constraints of the target machine.
- A storage-allocation decision is

    Static : if it can be made by the compiler looking only at the text of the program, not at what the program does when it executes.

    Dynamic : if it can be decided only while the program is running

Storage Organizations



Memory locations for code are determined at compile time.

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

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

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

Storage Allocation Strategies

The different storage allocation strategies are :

1. **Static allocation** - lays out storage for all data objects at compile time
2. **Stack allocation** - manages the run-time storage as a stack.
3. **Heap allocation** - allocates and deallocates storage as needed at run time from a data area known as heap.

Static Allocation

- In Static allocation, names are bound to storage as the program is compiled, so there is no need for a run-time support package.
- Since the bindings do not change at run-time, everytime a procedure is activated, its names are bound to the same storage locations. Therefore values of local names are retained across activations of a procedure.
- That is, when control returns to a procedure the values of the locals are the same as they were when control left the last time.
- From the type of a name, the compiler decides the amount of storage for the name and decides where the activation records go.
- At compile time, we can fill in the addresses at which the target code can find the data it operates on.

Limitations of Static allocation.

- The size of a data object and constraints on its position in memory must be known at compile time.
- Recursive procedures are restricted, because all activations of a procedure use the same bindings for local names.
- Data structures cannot be created dynamically, since there is no mechanism for storage allocation at run time.

Stack Allocation

All compilers that use procedures, functions or methods as units of user-defined actions manage at least part of their run-time memory as a stack.

Each time a procedure is called , space for its local variables is pushed onto a stack, and when the procedure terminates, that space is popped off the stack.

Procedure Activations

- An execution of a procedure starts at the beginning of the procedure body.
- When the procedure is completed, it returns the control to the point immediately after the place where that procedure is called.
- Each execution of a procedure is called as its ***activation***.
- *Lifetime* of an activation of a procedure is the sequence of the steps between the first and the last steps in the execution of that procedure (including the other procedures called by that procedure).
- If a and b are procedure activations, then their lifetimes are either non-overlapping or are nested.
- If a procedure is recursive, a new activation can begin before an earlier activation of the same procedure ends.

Activation Tree

- We can represent activations of procedures during running of the entire program by a tree, called an activation tree.
- 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 if the control flows from a to b.
  - The node a is left to the node b if the lifetime of a occurs before the lifetime of b.

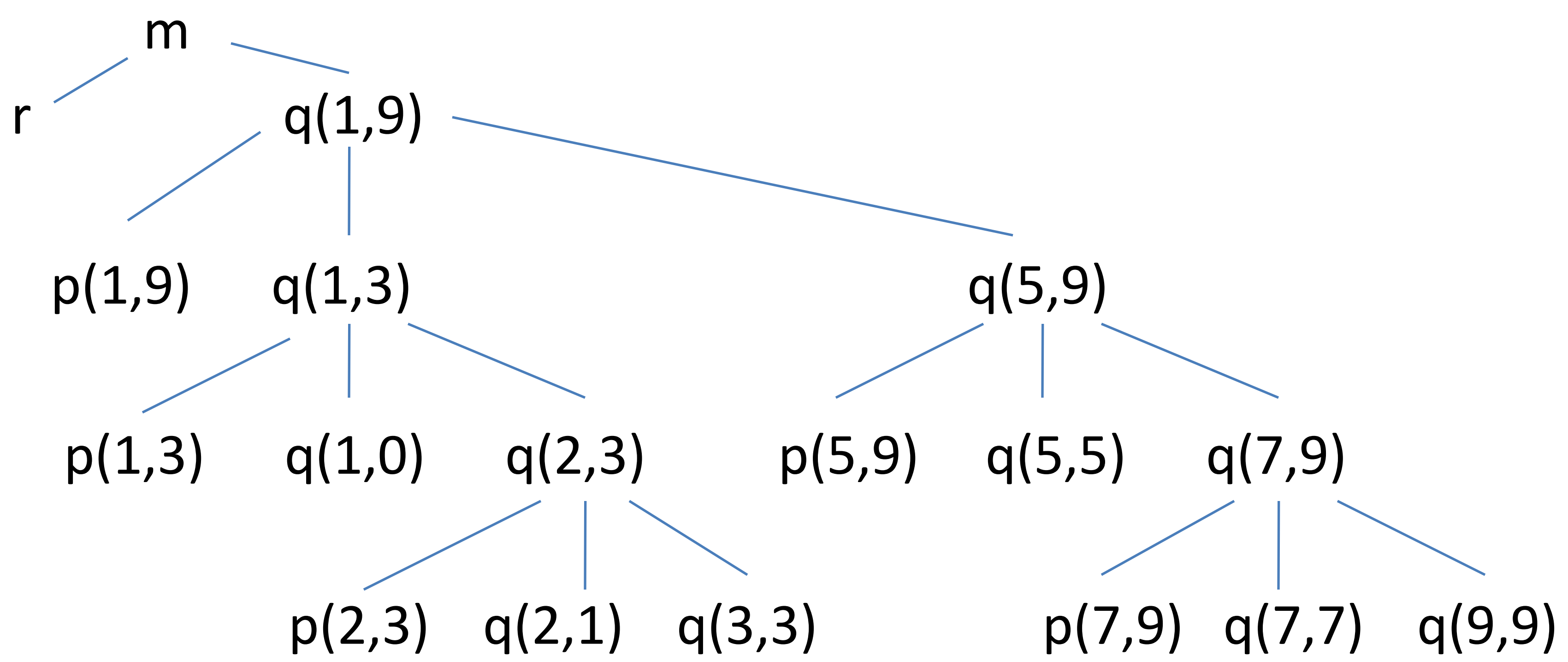
Sketch of quicksort program

```
int a[11] ;
void readArray( )
{ /* Reads array elements*/
 ...... }
 int partition(int m, int n)
{ /* Picks a separator value v, and partitions a[m .. n] so that a[m .. p - 1] are less than v, a[p] = v, and a[p
      + 1 .. n] are equal to or greater than v. Returns p. */
 ....... }
void quicksort(int m, int n)
{ int i;
   if (n > m)
  { i = partition(m, n) ;
     quicksort(m, i-1);
     quicksort(i+1, n) ;
   }
 }
main( )
{ readArray();
  quicksort (1,9) ;
 }
```

Possible activations for the program

```
enter main ( )
    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)
 leave main( )
```

Activation Tree



Activation Tree

- The flow of the control in a program corresponds to a depth-first traversal of the activation tree that:
  - starts at the root,
  - visits a node before its children, and
  - recursively visits children at each node in a left-to-right order.

Control Stack

- Procedure calls and returns are managed by a run-time stack called the control stack.
- Used to keep track of live procedure activations.
- Each live activation has an activation record on the control stack.
- The root of the activation tree is at the bottom.
- The latter activation has its record at the top of the stack.
- An activation record is pushed onto the control stack as the activation starts.
- This activation record is popped when the activation ends.

Variable Scopes

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

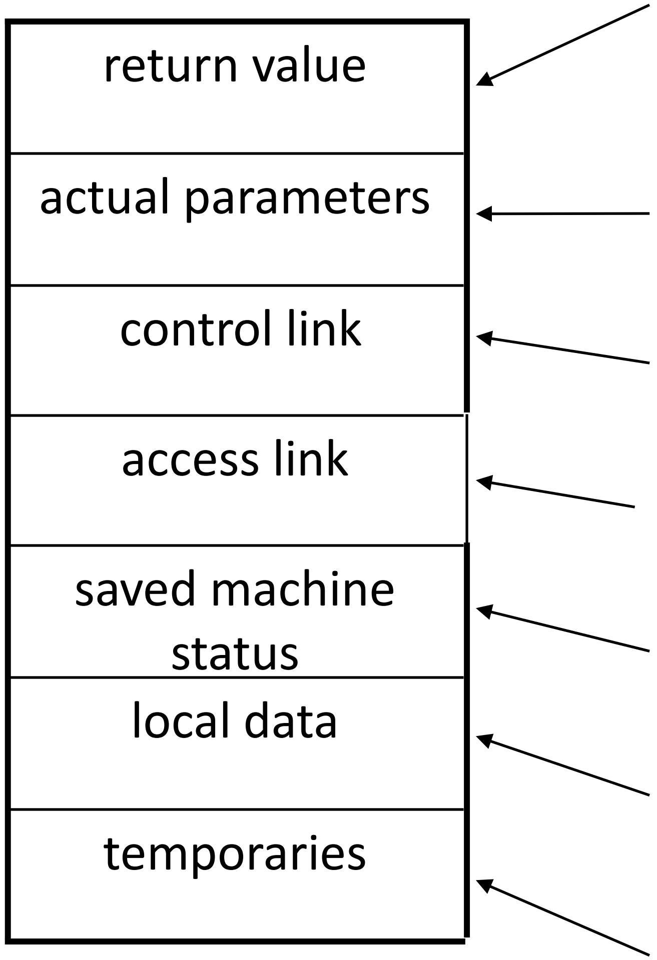
  Example

```
 a()                                    b( )
{ int i;  /* local variable */          { int j;      /* local variable */
………}                                     i = j+1;   /* i is non- local variable */
                                          …….. }
```

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 contents of the activation records vary with the language being implemented.

Activation Records



The returned value of the called procedure . Not all procedures return a value, and if any, we may prefer to place that value in machine register.

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

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

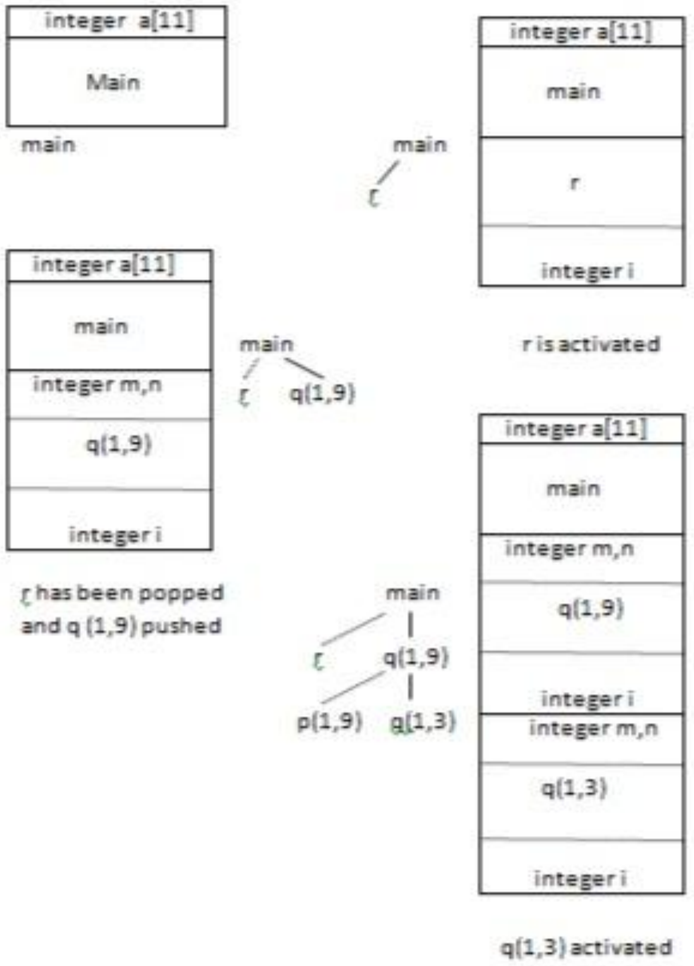
The access link is used to refer to nonlocal data held in other activation records.

Holds information about the state of the machine before the procedure is called. This includes return address(value of program counter) and content of registers.

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

Temporary variables those arising from the evaluation of expressions are stored.

Activation Records



Calling sequences

Procedure calls are implemented by generating calling sequences.

A ***Call sequence*** consists of code that allocates an activation record on the stack and enters information into its fields.

A ***return sequence*** restore the state of machine so the calling procedure can continue its execution.

The code in calling sequence is often divided between the calling procedure (caller) and the procedure it calls (callee).

Calling Sequences

When designing calling sequences and the layout of activation records, the following principles are helpful:

- Values communicated between caller and callee are generally placed at the beginning of the callee's activation record, so they are as close as possible to the caller's activation record.
- Fixed length items are generally placed in the middle. Such as the control link, the access link, and the machine status fields.
- Items whose size may not be known early enough are placed at the end of the activation record.
- Locate top-of-stack pointer point to the end of the fixed-length fields in the activation record.

Calling sequence

The calling sequence and its division between caller and callee are as follows:

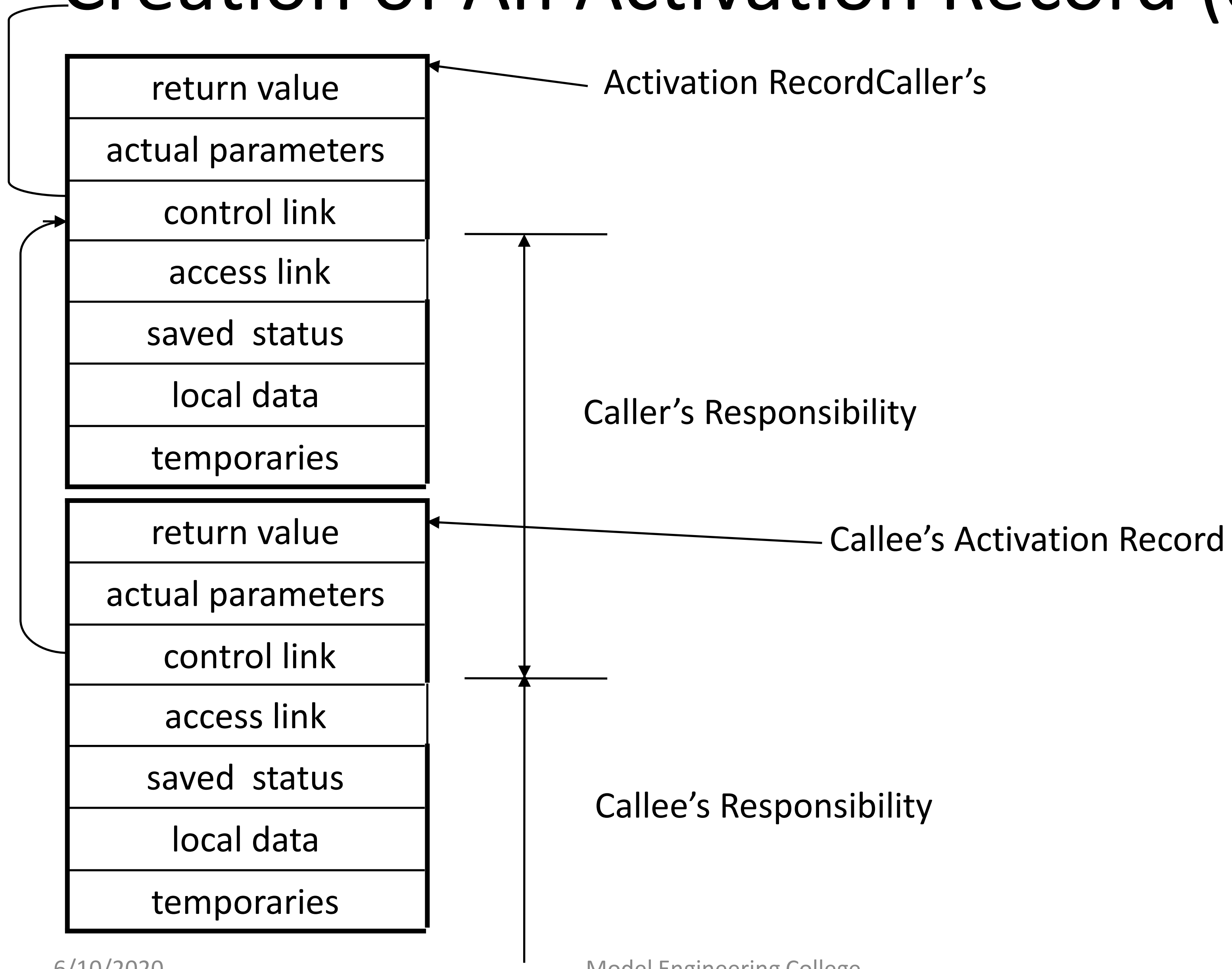
- The caller evaluates the actual parameters.
- The caller stores a return address and the old value of top_sp into the callee’s activation record. The caller then increments the top_sp to the respective positions.
- The callee saves the register values and other status information.
- The callee initializes its local data and begins execution.

Calling sequence

A possible return sequence is:

- The callee places the return value next to the parameters.
- Using the information in the machine-status field, the callee restores top_sp and other registers, and then branches to the return address that the caller placed in the status field.
- Although top_sp has been decremented, the caller knows where the return value is, relative to the current value of top_sp; the caller therefore may use that value.

Creation of An Activation Record (cont.)



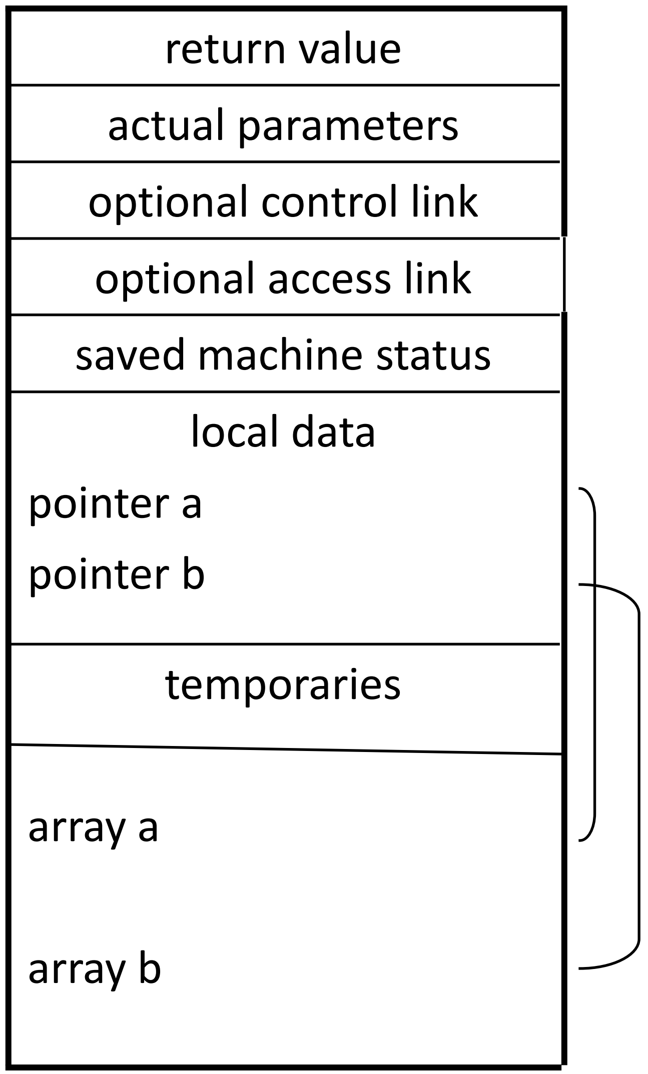
Calling Sequence

- A register *top_sp* points to the end of the machine status field in the current top activation record.
- This position within the callee’s activation record is known to the caller, so the caller can be made responsible for setting *top_sp* before control is passed to the callee.

Variable – Length Data on the Stack

- The run-time memory-management system must deal with the allocation of space for objects of which are not known at compile time, but which are local to a procedure and thus may be allocated on the stack.
- Objects whose size cannot be determined at compile time are normally allocated in the heap
- Placing objects on the stack if possible is that we avoid the expense of garbage collecting their space. The stack can be used only for an object if it is local to a procedure and becomes inaccessible when the procedure returns.

Variable Length Data



Variable length data is allocated after temporaries, and there is a link to from local data to that array.

Access to Nonlocal Name

- How a procedure access their data which is not belong to that procedure?
- A procedure may access to a nonlocal name using:
  - access links in activation records, or
  - displays (an efficient way to access to nonlocal names)

Access Link

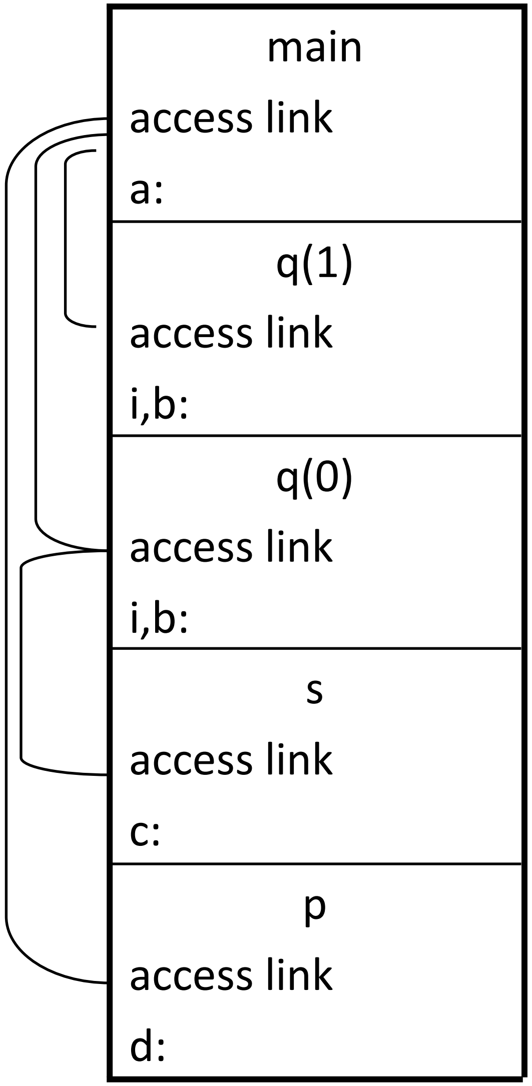
- A direct implementation of the normal static scope rule for nested functions is obtained by adding a pointer called the access link to each activation record.
- If procedure p is nested immediately within procedure q in the source code, then the access link in any activation of p points to the most recent activation of q.

Access Links

```
procedure p;
    var d:int;
    begin a:=1; end;

procedure q(i:int);
    var b:int;
       procedure s;
         var c:int;
         begin p; end;
    begin
      if (i<>0) then q(i-
1)
      else s;
    end;

program main;
  var a:int;
  begin q(1); end;
```



Displays

- An array of pointers to activation records can be used to access activation records.
- This array is called displays.
- For each level, there will be an array entry.

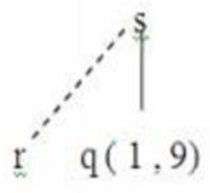
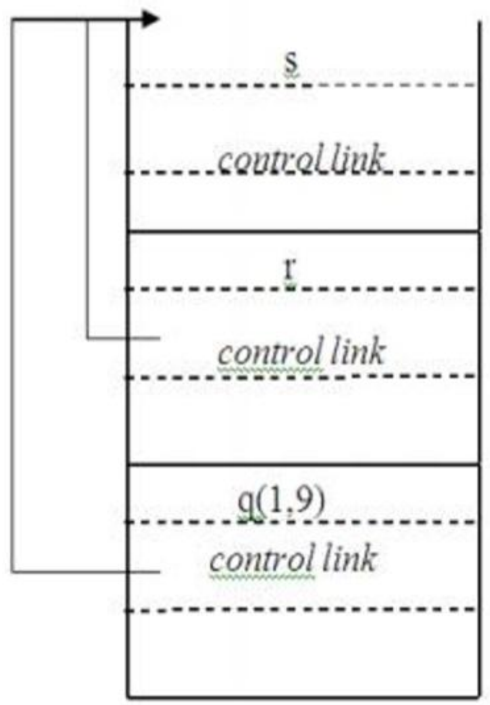
Heap Allocation

Stack allocation strategy cannot be used if either of the following is possible :

1. The values of local names must be retained when an activation ends.
2. A called activation outlives the caller.

- Heap is used for data that can live indefinitely, or until the program deletes it explicitly.
- Memory manager allocates and deallocates spaces within the heap.
- Spaces may be deallocated in any order, so over the time the heap will consist of alternate areas that are free and in use.
- Garbage collector finds apaces within the heap that are no longer in use and can reallocate to other data items.

Heap allocation

Position in the activation tree	Activation records in the heap	Remarks
	 s control link r control link q(1,9) control link	Retained activation record for r

Thank You