

5.1 RUN-TIME ENVIRONMENTS

A translation needs to relate the static source text of a program to the dynamic actions that must occur at runtime to implement the program. The program consists of names for procedures, identifiers etc., that require mapping with the actual memory location at runtime.

Runtime environment is a state of the target machine, which may include software libraries, environment variables, etc., to provide services to the processes running in the system.

5.1.1 SOURCE LANGUAGE ISSUES

Procedure

A *procedure* definition is a declaration that associates an identifier with a statement. The identifier is *procedure* name, and statement is the *procedure* body.

For example, the following definition of procedure named *readarray*

```
procedure readarray;  
var i : integer;  
  
begin  
    for i := 1 to 9 do read(a[i])  
end;
```

When a procedure name appears with in an executable statement, the procedure is said to be *called* at that point.

Activation Tree

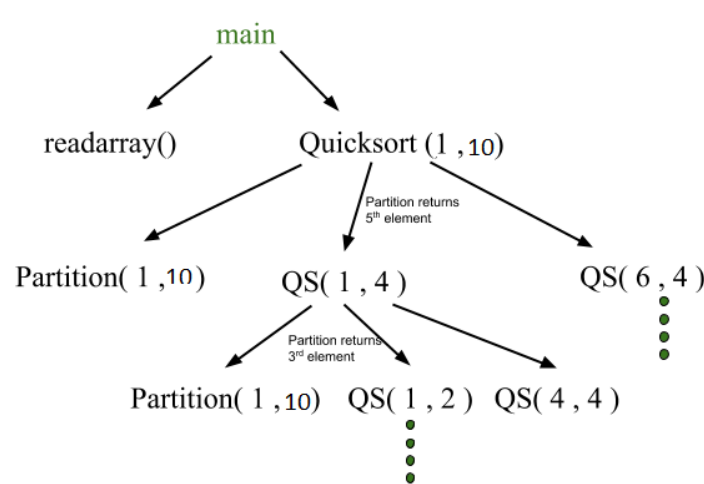
- ✚ Each execution of procedure is referred to as an activation of the procedure. Lifetime of an activation is the sequence of steps present in the execution of the procedure.
- ✚ If 'a' and 'b' be two procedures, then their activations will be non-overlapping (when one is called after other) or nested (nested procedures).
- ✚ A procedure is recursive if a new activation begins before an earlier activation of the same procedure has ended. An activation tree shows the way control enters and leaves, activations.
- ✚ Properties of activation trees are :-
 - ❖ Each node represents an activation of a procedure.
 - ❖ The root shows the activation of the main function.
 - ❖ The node for procedure 'x' is the parent of node for procedure 'y' if and only if the control flows from procedure x to procedure y.

EXAMPLE

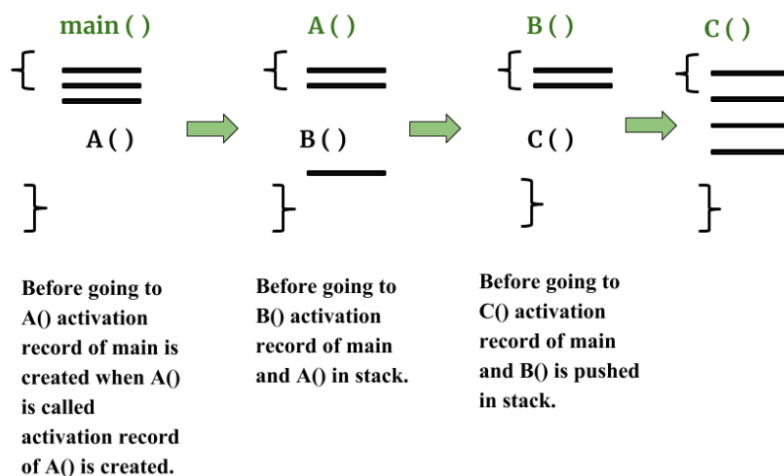
Consider the following program of quicksort

```
main()
{
    readarray();
    quicksort(1,10);
}

quicksort(int m, int n)
{
    int i= partition(m,n);
    quicksort(m,i-1);
    quicksort(i+1,n);
}
```



- First main function as root then main calls readarray and quicksort.
- Quicksort in turn calls partition and quicksort again. The flow of control in a program corresponds to the depth first traversal of activation tree which starts at the root.



Control Stack

- ✚ Control stack or runtime stack is used to keep track of the live procedure activations i.e the procedures whose execution have not been completed.
- ✚ A procedure name is pushed on to the stack when it is called (activation begins) and it is popped when it returns (activation ends).
- ✚ Information needed by a single execution of a procedure is managed using an activation record.
- ✚ When a procedure is called, an activation record is pushed into the stack and as soon as the control returns to the caller function the activation record is popped on as the control turns to the caller function the activation record is popped.
- ✚ Then the contents of the control stack are related to paths to the root of the activation tree. When node n is at the top of the control stack, the stack contains the nodes along the path from n to the root.
- ✚ Consider the above activation tree, when quicksort(4,4) gets executed, the contents of control stack were main() quicksort(1,10) quicksort(1,4), quicksort(4,4)

quicksort(4,4)
Quicksort(1,4)
Quicksort(1.10)
Main()

The Scope Of Declaration

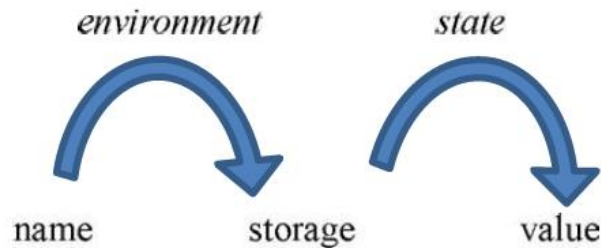
- ✚ A declaration is a syntactic construct that associates information with a name. Declaration may be explicit such as

var i : integer;

or may be explicit. The portion of program to which a declaration applies is called the **scope** of that declaration.

Binding Of Names

- ✚ Even if each name is declared once in a program, the same name may denote different data object at run time. "Data objects" corresponds to a storage location that hold values.
- ✚ The term *environment* refers to a function that maps a name to a storage location.
- ✚ The term *state* refers to a function that maps a storage location to the value held there.

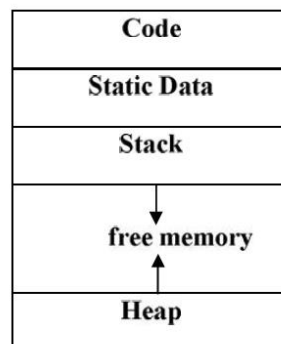


- When an environment associates storage location s with a name x , we say that x is bounds to s . This association is referred to as a binding of x .

5.1.2 STORAGE ORGANIZATION

- 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 address into physical addresses, which are usually spread through memory.

Typical subdivision of run time memory.



- Code area:** used to store the generated executable instructions, memory locations for the code are determined at compile time
- Static Data Area:** Is the locations of data that can be determined at compile time
- Stack Area:** Used to store the data object allocated at runtime. eg. Activation records
- Heap:** Used to store other dynamically allocated data objects at runtime (for ex: malloc)
- This runtime storage can be subdivided to hold the different components of an existing system
 - Generated executable code
 - Static data objects
 - Dynamic data objects-heap
 - Automatic data objects-stack

Activation Records

- ✚ It is LIFO structure used to hold information about each instantiation.
 - ✚ Procedure calls and returns are usually managed by a run time stack called control stack.
 - ✚ Each live activation has an activation record on control stack, with the root of the activation tree at the bottom, the latter activation has its record at the top of the stack
 - ✚ The contents of the activation record vary with the language being implemented.
 - ✚ The diagram below shows the contents of an activation record.
 - ✚ The purpose of the fields of an activation record is as follows, starting from the field for temporaries.
1. Temporary values, such as those arising in the evaluation of expressions, are stored in the field for temporaries.
 2. The field for local data holds data that is local to an execution of a procedure.
 3. The field for saved machine status holds information about the state of the machine just before the procedure is called. This information includes the values of the program counter and machine registers that have to be restored when control returns from the procedure.
 4. The optional access link is used to refer to nonlocal data held in other activation records.
 5. The optional control link points to the activation record of the caller
 6. The field for actual parameters is used by the calling procedure to supply parameters to the called procedure.
 7. The field for the returned value is used by the called procedure to return a value to the calling procedure, Again, in practice this value is often returned in a register for greater efficiency.

Returned value
Actual parameters
Optional control link
Optional access link
Saved machine status
Local data
temporaries

General Activation Record

5.1.3 STORAGE ALLOCATION STRATEGIES

- ✚ The different storage allocation strategies are:

Static allocation - lays out storage for all data objects at compile time

Stack allocation - manages the run-time storage as a stack.

Heap allocation - allocates and deallocates storage as needed at run time from a data area known as heap.

Static Allocation

- ✚ In static allocation, names 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 runtime, every time a procedure activated, its run-time, names bounded to the same storage location.
- ✚ Therefore, values of local names retained across activations of a procedure. That is when control returns to a procedure the value of the local are the same as they were when control left the last time.
- ✚ From the type of a name, the compiler decides amount of storage for the name and decides where the activation records go. At compile time, we can fill in the address at which the target code can find the data it operates on.

Stack Allocation

- ✚ All compilers for languages that use procedures, functions or methods as units of user functions define actions manage at least part of their runtime memory as a stack run-time stack.
- ✚ Each time a procedure called, space for its local variables is pushed onto a stack, and when the procedure terminates, space popped off from the stack

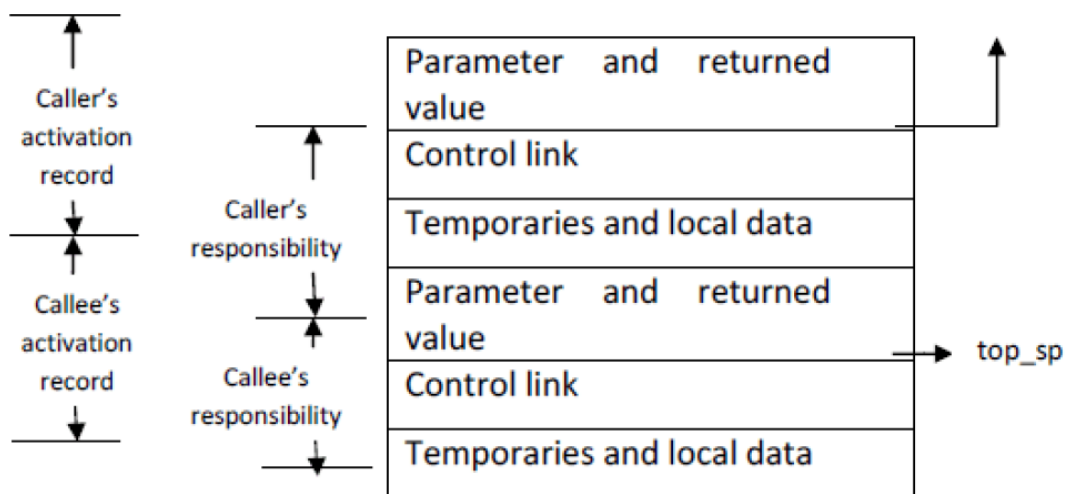
Calling Sequences

- ✚ Procedures called implemented in what is called as calling sequence, which consists of code that allocates an activation record on the stack and enters information into its fields.
- ✚ A return sequence is similar to code to restore the state of a machine so the calling procedure can continue its execution after the call.
- ✚ The code is calling sequence of often divided between the calling procedure (caller) and a procedure is calls (callee)(callee).
- ✚ When designing calling sequences and the layout of activation record, the following principles are helpful:

1. Value communicated between caller and callee generally placed at the caller beginning of the callee's activation record, so they as close as possible to the caller's activation record.
2. Fixed length items generally placed in the middle. Such items typically include the control link, the access link, and the machine status field.
3. Items whose size may not be known early enough placed at the end of the activation record.
4. We must locate the top of the stack pointer judiciously. A common approach is to have it point to the end of fixed length fields in the activation is to have it point to fix the end of fixed length fields in the activation record. Fixed length data can then be accessed by fixed offsets, known to the **intermediate code generator**, relative to the top of the stack pointer.

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

1. The caller evaluates the actual parameters.
2. 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.
3. The callee-saves the register values and other status information.
4. The callee initializes its local data and begins execution.

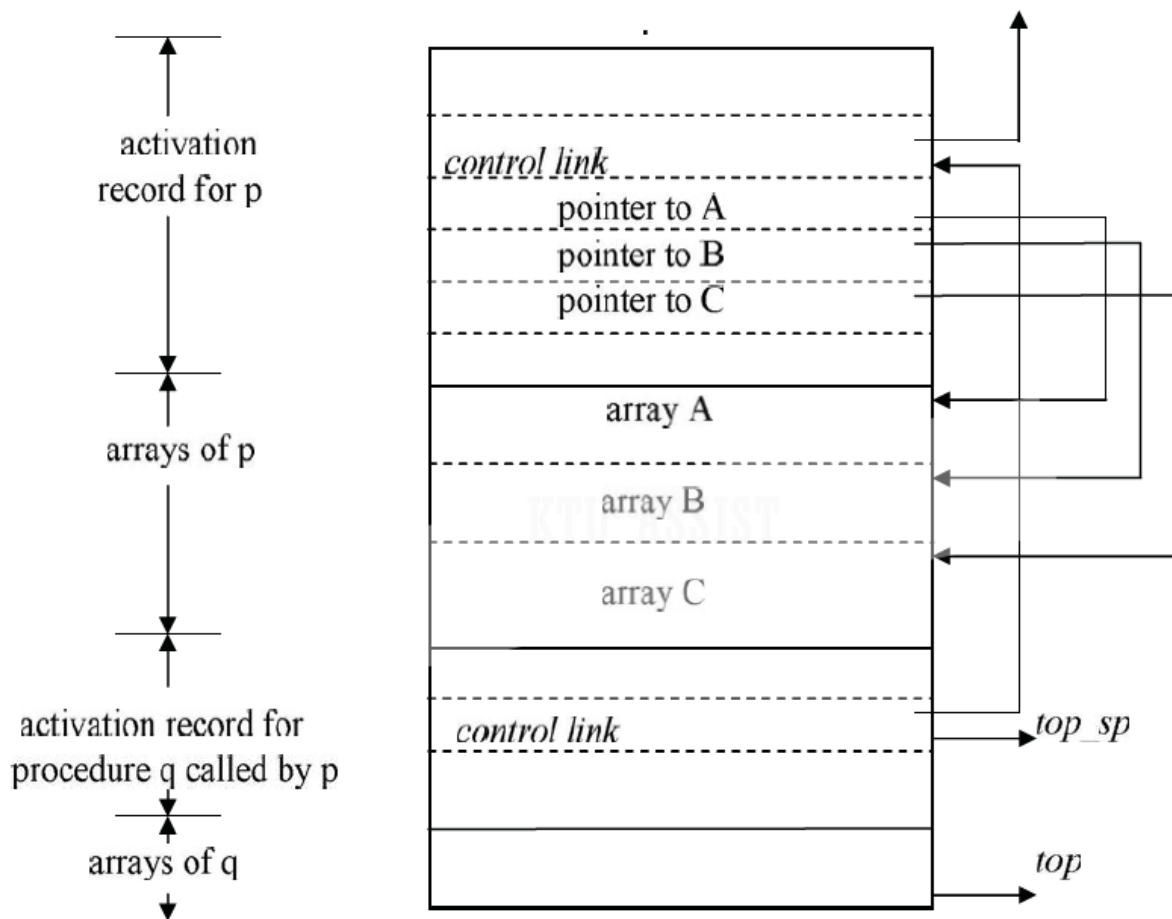


 A suitable, corresponding return sequence is:

1. The callee places the return value next to the parameters.
2. 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.
3. 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.

Variable length data on the stack

- ✚ The run-time memory-management system must deal frequently with the allocation of space for **objects the sizes of which are not known** at compile time, but which are local to a procedure and thus **may be allocated on the stack**.
- ✚ In modern languages, objects whose **size cannot be determined** at **compile time** are **allocated space in the heap**.
- ✚ However, it is also **possible to allocate objects, arrays, or other structures of unknown size on the stack**.
- ✚ We **avoid the expense of garbage collecting** their space. Note that the stack can be used only for an object if it is local to a procedure and **becomes inaccessible when the procedure returns**.
- ✚ A common strategy for allocating variable-length arrays is shown in following figure



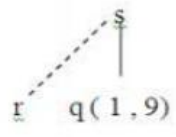
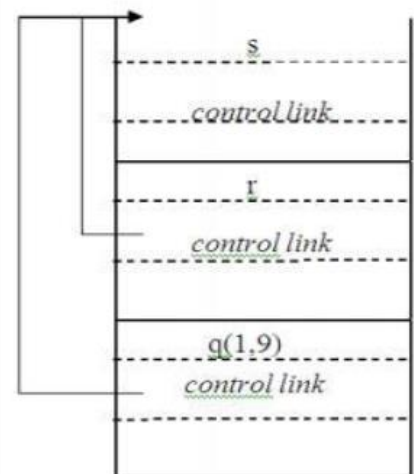
Access to dynamically allocated arrays

Dangling Reference (Storage allocation strategies)

- ✚ Whenever storage allocated, the problem of dangling reference arises. The dangling reference occurs when there is a reference to storage that has been allocated.
- ✚ It is a logical error to u dangling reference, since, the value of de use de-allocated storage is undefined according to the semantics of most languages.
- ✚ Whenever storage allocated, the problem of dangling reference arises. The dangling reference occurs when there is a reference to storage that has been allocated.

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 allocation parcels out pieces of contiguous storage, as needed for activation records or other objects.
- ✚ Pieces may be deallocated in any order, so over the time the heap will consist of alternate areas that are free and in use.

Position in the activation tree	Activation records in the heap	Remarks
		Retained activation record for r

Records for live activations need not be adjacent in heap

- ✚ The record for an activation of procedure r is retained when the activation ends.
- ✚ Therefore, the record for the new activation q(1, 9) cannot follow that for s physically.
- ✚ If the retained activation record for r is deallocated, there will be free space in the heap between the activation records for s and q.