Assignment

1. Explain the source language issues in run time environment

suppose that the source program is made up of procedures. This section distinguishes between the source text of a procedure and its activations at the run time. In this section we deal with

- procedures
- Activation Trees
- control stacks
- The scope of a Declration
- Binding of Names.

procedures

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 a do read (a[i])
end;
```

when a procedure name appears with in executable statement the procedure is said to be called at that point. The identifier appearing in a procedure definition are known as formal parameters of the procedure. Arguments that are passed to a called procedure are known as the actual parameters.

Activation Trees

> Each execution of procedure is referred to as an activation of the procedure. The lifetime of an activation is the sequence of steps present in the execution of that procedure.

> if 'a' and 'b' be two procedures, then their activations will be non-overlapping or nested.

> 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, activation.

> properties of activation trees are:

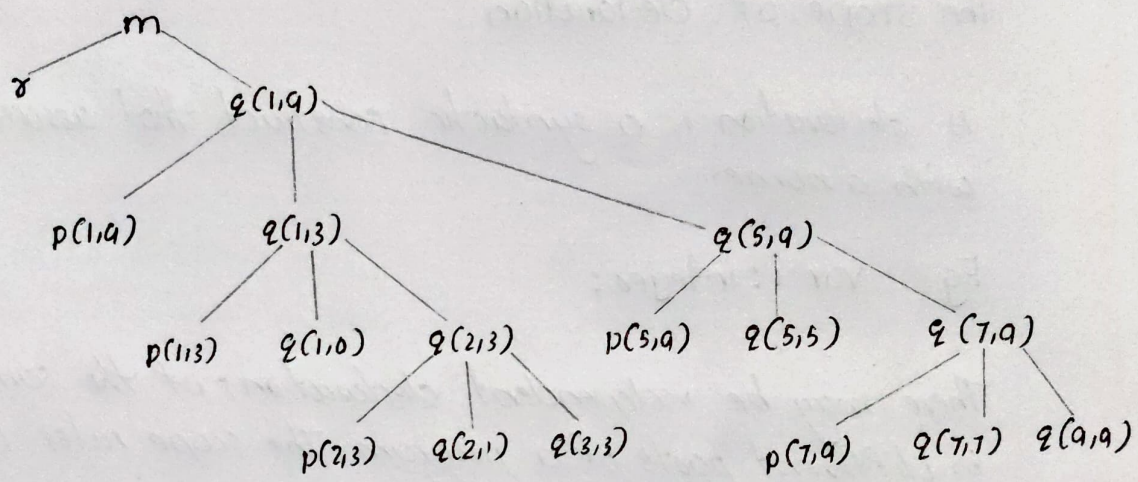
- Each node represents an activation of a procedure.
- The root shows that activation of the main function
- The node for procedure 'x' is the parent of node for procedure 'y' if and only if the control flows from procedure 'x' to procedure 'y'.
- The node for 'x' is to be left of the node for 'y' if and only if the lifetimes of 'x' occurs before the lifetime of 'y'

Example:

```
main()
{
    readarray();
    quicksort(1,9);
}
quicksort(int m, int n)
{
    int i = partition(m,n);
```

```
    quicksort(m, i-1);
    quicksort(i+1, n);
}
```

suppose the above procedure partitioned the array with i=4 during the first call then the activation tree will be as given below.



control stack

- The flow of control in a program follows a depth-first traversal of the activation tree, visiting nodes before their children in a left-to-right order.
- The control stack tracks active procedure execution.
- A procedure name is pushed onto the stack when called and popped when it returns.
- The stack contents represent paths from active nodes to the root of the activation tree.

for example, consider the above activation tree, when ~~quicksort~~ quicksort(2,3) get executed. The contents of the control stack

quicksort(2,3)
quicksort(1,3)
quicksort(1,9)
main()

The scope of declaration

A declaration is a syntactic construct that associates information with a name.

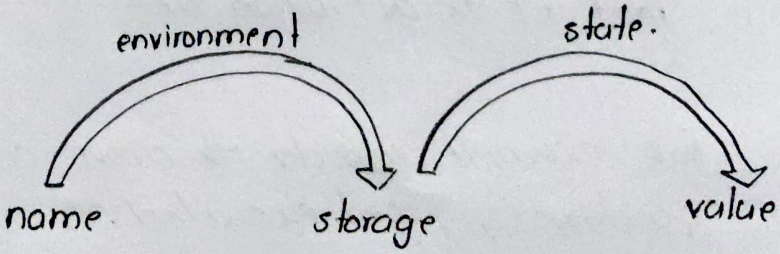
Eg: var i: integer;

There may be independent declarations of the same name in different parts of a program. The scope rules of a language determine which declaration of a name applies when the name appears in the ~~context~~ of a program.

The scope of a declaration is the portion of the program it applies to a name in a procedure is local if declared within in its scope otherwise, it is nonlocal.

Binding of Names

The term environment refers to a function that maps a name to a storage location, and the term state refers to a function that maps a storage location to the value held there. This is, an environment maps a name to an l-value and state maps the l-value to an r-value.



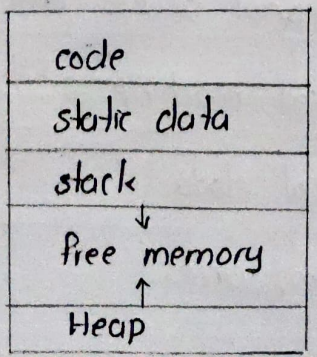
Environment and states differ : Assignments change the state not the enviroment. if pi at address 100 holds 0, after pi = 3.14 it still refers to the same address but with a new value. A name is bound to a storage location, making binding the dynamic counter ~~point~~ post of a declaration.

static Notation	Dynamic Notation
Definition of a procedure	Activation of the procedure
Declaration of a name	Binding of the frame
scope of a declaration	Lifetime of a binding

2. Explain storage organization and storage allocation strategies in run time environment.

Storage Organization

suppose that the compiler obtains a block of storage from the operating system for compiled program to run. This run-time storage might be subdivided to hold : the generated target code, data objects and a counter~~point~~post of the control stack to keep track of procedure activations.



The compiler places fixed-size target code and known-size data objects in a statically determined area since their sizes are known at compile time. This allows efficient memory allocation without the need for dynamic adjustments.

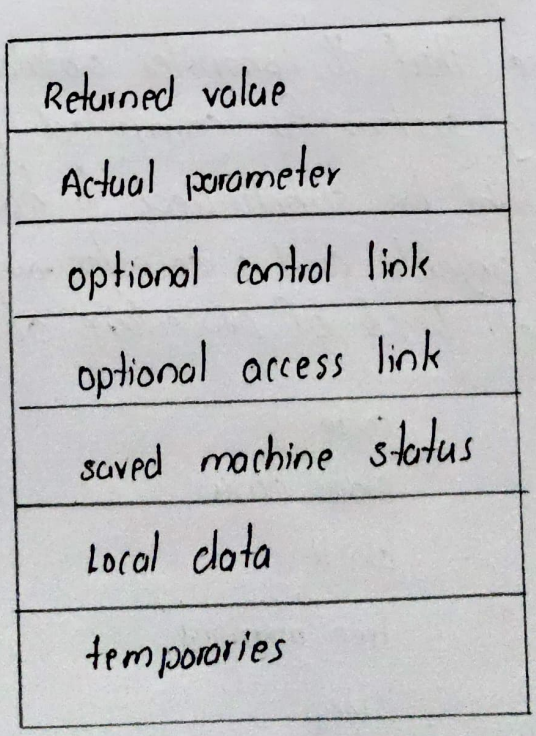
control stack manage procedure activations by saving execution state, such as the program counter and register values, when a function call occures. once the function returns the saved state is restored allowing execution to continue. Data objects with lifetime tied to function activations are also stored on the stack

The heap handles dynamically allocated objects including activation data in some languages.

stack and heap grow dynamically. stack downward and heap upward - to optimize use.

Activation Records

Information needed by a single execution of a procedure is managed using a contiguous block of storage called an activation record or frame, which consists of following fields.



The purpose of the fields of an activates record is a follows:

1. Temporary value, such as those arising in the evaluation of expressions, are stored in the field for tempories.
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 stored when control returns from the procedure.
4. The access link refers to nonlocal data in their activation record
5. The control link points points to the caller's activation record
6. The actual parameters field passes arguments from the caller to the called ~~produ~~ procedure.
7. return value field stores the result returned to the caller.

Storage Allocation Strategies

1. static allocation
2. stack allocation
3. Heap allocation

Static Allocation

In static allocation, names are bound to storage at compile time, eliminating the need for runtime support. since binding remain unchanged, each procedure activation retains the same storage locations, preserving local variables values across

activations. The compiler determines storage requirements and placement, allowing direct access to data.

Limitations:

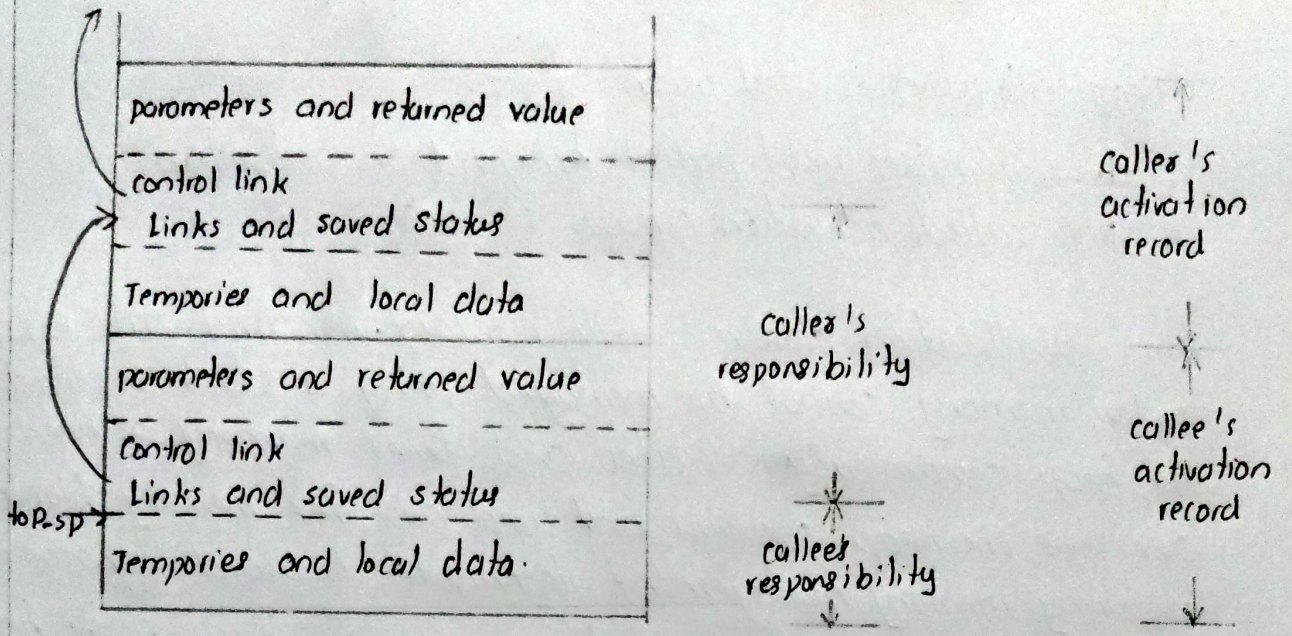
1. Data size and memory position must be known at compile time
2. Recursive procedures are restricted as all activations share the same local bindings.
3. Dynamic data structures cannot be created due to the lack of runtime allocation.

Stack Allocation

Stack allocation uses a control stack where activation records are pushed and popped as procedures are called and return. Each procedure call allocates space for local variables in its activation record, and these values are discarded when the activation ends. The stack grows and shrinks dynamically with each call, using a top register to manage memory.

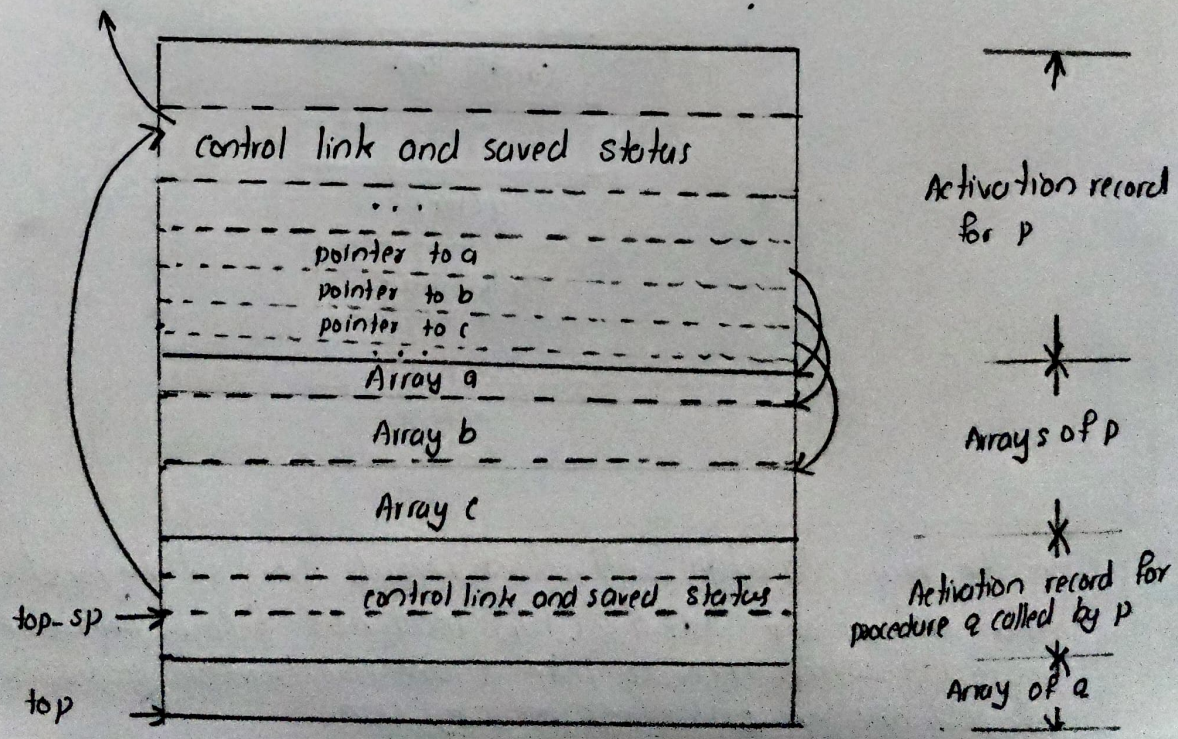
Calling Sequence

- A procedure call involves generating a calling sequence to allocate the activation record and fill in relevent information
- The return sequence restores the machine's state so that the caller can resume execution.
- The calling sequence is split between the caller and callee.
- Fixed size fields like control and access link are placed in the middle, while values shared between caller and callee are placed at the beginning of the callee's activation record.
- variable-length data is placed at the end, and the top-of-stack pointer is carefully located to access fixed-length data.



variable_length Data on the Stack

For arrays or objects whose size ~~operates~~ depends on parameter the activation record contains pointers to the arrays rather than the arrays themselves. These arrays are allocated dynamically on the stack, and access is facilitated through these pointers. When one procedure calls another, the data for the called procedure follows the local data of the caller in the stack.

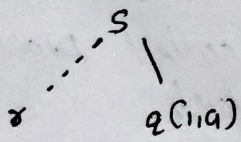
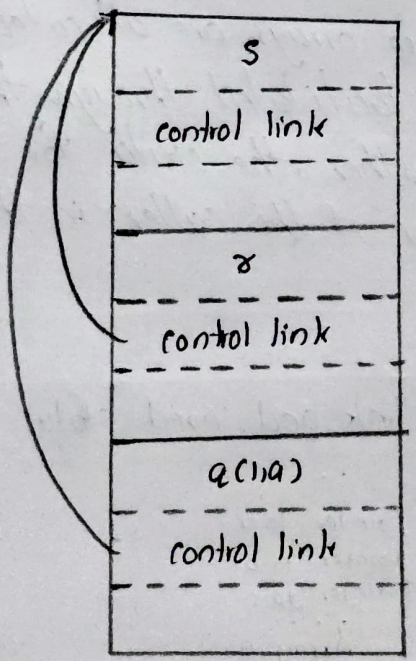


Heap Allocation

Heap allocation is used when:

1. Local values must persist ~~by~~ beyond an activation's end
2. A called activation outlives its caller.

since deallocation doesn't follow a last-in, first-out (FILIFO) order, memory cannot be managed using a stack. Instead, the heap dynamically allocates and frees memory as needed. over time, the heap contains both used and free memory blocks, requiring memory management techniques to avoid fragmentation. Heap ~~soot~~ allocation for activation records is shown in the below figure

position in the Activation tree	Activation Records in the heap	Remarks
		Retained activation record for r

In the heap allocation, activation records persist beyond execution. New activations like q(1,9) may not follow previous ones physically. when records like x are deallocated, gaps form and the heap manager optimizes memory use.