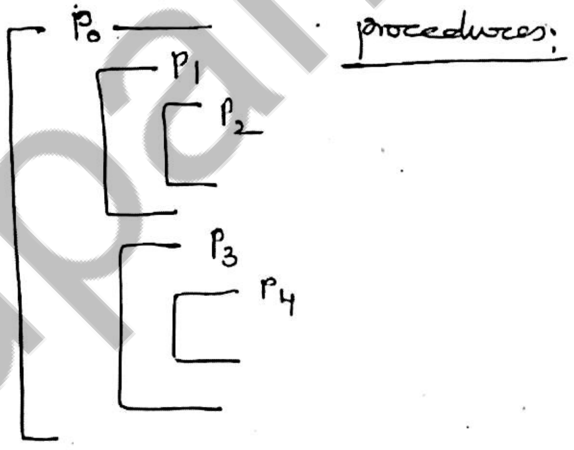
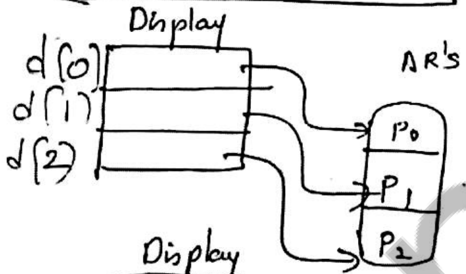
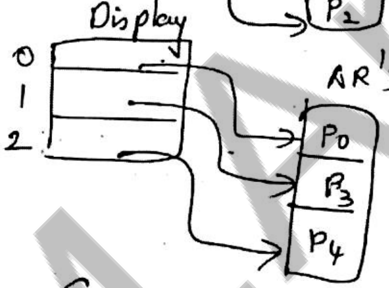
Display: Access to Nonlocal data by traversing down access links every time is expensive when the depth of nesting increases. (as we have to follow long chains of links to reach the data we need) So an array of pointers called display d[i] is* maintained which consists of 1 ptr for each nesting depth.

1. In an display an array of pointers to activation record is maintained. (one pointer for each nesting depth)
2. Array is indexed by nesting level.
3. The pointers point to only accessible activation record.
4. The display changes when a new activation record occurs and it must be reset when control returns from the new activation.



Maintaining a display.





*d[i] is a ptr to the highest activation record on the stack for any procedure at nesting depth i.

Comparision between Access links & displays.

1. Access links take more time to access non local variables especially when non local variables are at many nested levels away.
2. Display is used to generate code efficiently.
3. Display requires more ~~time~~ space at runtime than access links.

Type Conversion, Type Equivalance and Type Coercion

(Explain the terms.)

contd (*) In most languages, even a stack based environment needs some dynamic capabilities in order to handle ptr allocation and deallocation. The data

Heap Allocation: Structure that handles such allocation is called a heap.

If ~~local~~ values of local names must be retained when an activation ends, and a called activation outlives the caller Stack allocation Strategy is not possible. This is because the deallocation of activation records will not occur in a LIFO fashion. So Storage cannot be organized as a Stack. Eg: C++, Java have new to create objects/ptrs which may be passed from procedure to procedure, so they continue to exist after the procedure that created them is gone. So, Heap gives out pieces of storage as needed for activation records or other objects. These pieces can be deallocated at any time and the free space is used by other objects. (*) contd.

Heap allocation retains the activation record even after the activation is complete. Later when it is deallocated the free space will be filled up by heap manager by some other objects, whereas in a Stack the activation record is deallocated after completion of execution and allocates the next activation record which has started execution.
The heap provides two operations allocate & free Eg: new and delete C++, malloc and free C
The size of heap is not fixed. It may grow or shrink interchangeably during the program execution.

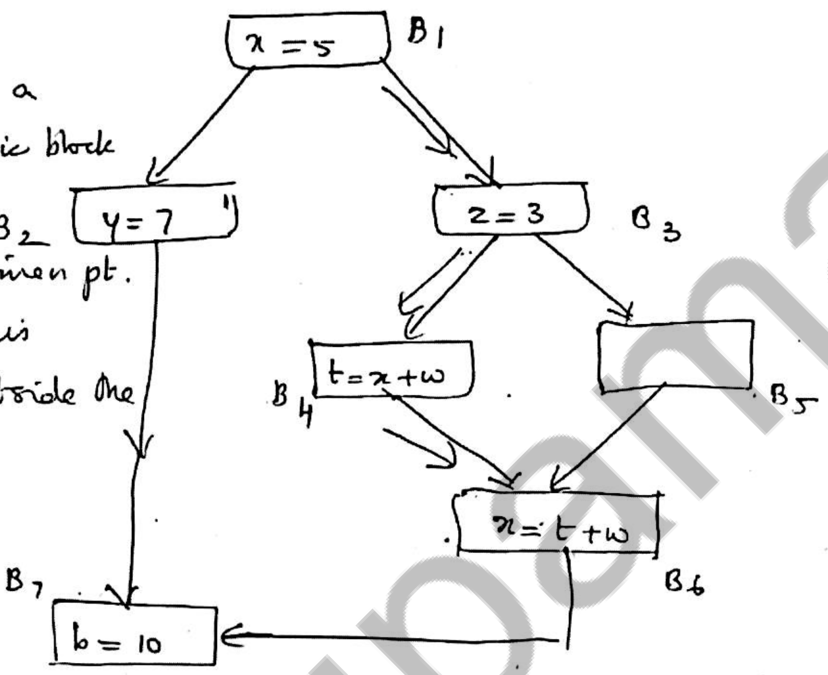
The memory manager: 7.4.1 Aho Ullman.

<u>Garbage Collection</u>: It is the process of finding spaces within the heap that are no longer used by the program, and can therefore be reallocated for other data items. Eg: In Java the garbage collector deallocates memory. Automatic garbage collection deallocates unreachable data.

7.5.1 and 7.5.2 Aho Ullman Design goals for garbage collectors, performance metrics and Reachability.

Live Variables: A variable x is live at some pt. p if there is a path from p to the exit, along which the value of x is used before it is redefined. Otherwise the variable is said to be dead at that point.

In other words a variable ~~name~~ in a basic block is said to be alive at a given pt. if its value is required outside the block.



x is live at block B_1, B_3, B_4 but killed at B_6.

live variables are useful in register allocation.

live variables are useful for dead code elimination.

Access to Non local Data: Variables not local to a procedure are called non local variables.

Handling Non local Data

Used by block structured languages. ← Static Scope or lexical Scope
- Access link
- Display

Used by non-block structured language → Dynamic Scope
- Deep Access
- Shallow Access.

P.T.O ( )

Static Scope Rule: Also called as lexical scope. The scope is determined by examining the program text. Eg: Pascal, C. These languages are called block structured lang.

Dynamic Scope: Non block structured languages Lisp, Snobol.

dynamic scope.

(It determines the scope of declaration of the names at runtime by considering the current activation.)

The use of a non local variable refers to the non local data declared in most recently called and still active procedure. Therefore each time new bindings are set up for local names called procedure. (Symbol tables can be required at run time.

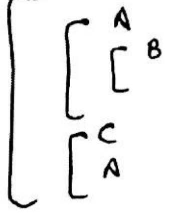
(In dynamic scoping the question of which non local variables to use is determined at run time. There are two ways to implement non local accessing under dynamic scope.

1. Deep Access 2. Shallow Access.)

Static Scope Rule: Block structured languages.

The blocks can be in nested fashion that is B_2 completely inside B_1 (The scope of declaration in a block structured language is given by most closely nested loop or static rule. Block structure storage allocation can be done by stack.)

Nested procedures: Nested procedure is a procedure that can be declared within another procedure.



(Static/lexical scope can be implemented using access link and displays.)