Ravi Raushan
Register NO: 20218074

Assignment - 1

1 a) Static Storage Allocation

i) → Allocation is done at compile time.

ii) Binding doesn't change at run time i.e once you bind one variable to some address that variable can never be moved out of that address.
(There is nothing like malloc. & free)

iii) only one activation record per procedure

Disadvantage :-

i) Recursion is not supported.
ii) Size of data objects must be known at compile time.
iii) Data structures can't be created dynamically because there is no support of heap.

b) Stack Storage Allocation

→ When a new activation begins, activation record is pushed on to the stack and whenever activation end, activation record is popped off.

→ Local variables are bound to fresh storage. When you call the function that local variable will be created again & again at different space in the stack.

Advantage:-

→ Support Recursion

Disadvantage:-

→ Local variable can't be retained once activation end.

c) Heap storage Allocation

→ Allocation & deallocation can be done in any order.

→ Allocate the memory to the variables dynamically and when the variables are no more used then claim it back.

→ Supports recursion.

Disadvantage

→ Heap management is overhead (In case of hole generation, memory management is difficult)

2) Symbol table is an important data structure created and maintained by compilers in order to store information about the occurrence of various entities such as variable name, function names, objects, classes, interface etc.

It is used by various phases of compiler as follows :-

	Phase	Usage
1)	Lexical Analysis	Creates new entries for each identifier
2)	Syntax Analysis	Adds information regarding attributes like type, scope, dimension, line of reference and line of use
3)	Semantic Analysis	Uses the available information to check for semantic and is updated
4)	Intermediate code Generation	Information in symbol table helps to add temporary variables information
5)	Code optimization	Uses information present in symbol table for machine dependent optimization by considering address and aliased variable information
6)	Target code Generation	Generate the codes by using the address information of identifiers present in the table

Implementation of Symbol Table

Implementation	Insertion Time	Look up time	Disadvantages
1) Linear list a) Ordered list → Array → Linked list	 O(n) O(n)	 O(logn) O(n)	→ Look up time is directly proportional to table size in case of unorder list
b) unordered list (Array or linked list)	O(1)	O(n)	→ Every insertion operation proceed with lookup operation in case of sorted list.
2) self organized list	O(1)	O(n)	→ poor performance when less frequently items are searched
3) Search Tree	$O(\log_k n)$	$O(\log_k n)$	→ we have to always keep it balance.
4) Hash Table	O(1)	O(1)	→ when there are too many collisions the time complexity increases to O(n)

9) A scope in any programming is a region of the program where a defined variable can have its existence and beyond that variable it can't be accessed.

→ In a source program, every name possess a region of validity, called the scope of the name

scope rules.

Block structured

→ In same program, we can have many blocks.

→ variable with same name can be declared again and its scope is within that block.

The rules in block-structured language :-

i) If a variable is declared within Block B it will be valid only within B.

ii) If B_1 is nested within B_2, then variable valid for B_2 will be valid for B_1 unless the name's identifier is redeclared in B_1.

These scope rules need a more complicated organization of a symbol table than a list of association between names & attributes.

→ Tables are organized into stack :-

→ New block – New table entered into the stack
→ when the declaration is compiled then the table is searched for a name.
→ If name not found, name is inserted.
→ when the name's reference is translated then each table is searched, starting from the each table on the stack.

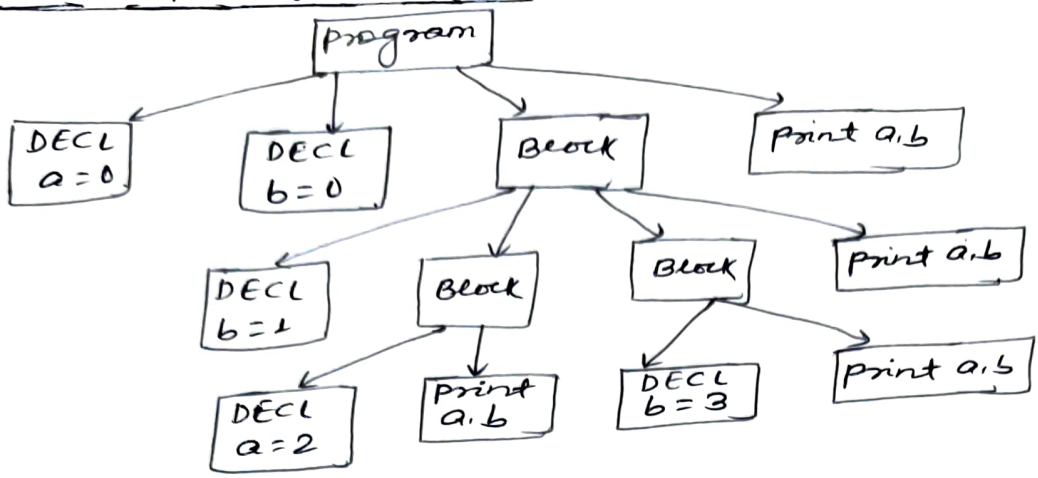
Access to Non-local variables in a Block-Structured lang.

→ Lexical scope (static scoping)
scope is verified by examining the text of the program.
e.g: PASCAL, C, ADA use the static scope rule

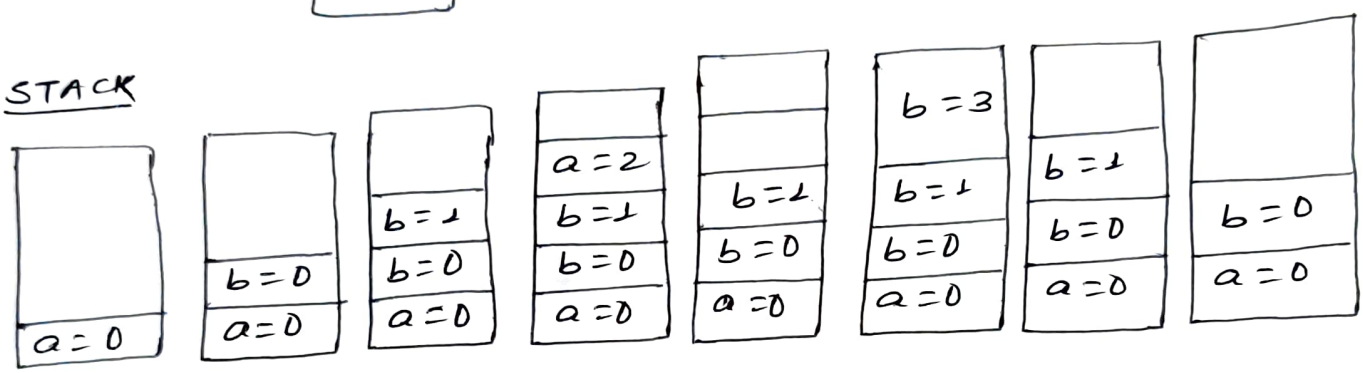
→ Dynamic scope : determines the reference of a variable at run time when the time the reference is made.
To create dynamic scope, compiler can use the address of the calling context.
→ Easier for compiler to implement.

e.g:
```
main
{ int a=0;
  int b=0;
  { int b=1;
    { int a=2;
      printf("%d %d\n", a, b);
    }
    { int b=3;
      printf("%d %d\n", a, b);
    }
    printf("%d %d\n", a, b);
  }
  printf("%d %d\n", a, b);
}
```

Dynamic scope (Symbol Table)



STACK



We implement name access in a block structured language using the control stack mechanism

On this, a block resembles a procedure with no parameter that is called in one place (the spot just before it begin) and returns to one place (the spot just after it ends).