Q1 Difference between static and dynamic memory allocation.

Ans

static	dynamic
① In this memory is allocated to a variable before the execution of a program begins.	① The memory bindings are established and destroyed during the execution of a program.
② No memory allocation and deallocation action are performed during the execution of a program.	② memory is allocated and deallocation during the execution of program.
③ Variables remain permantly allocated, allocation of variable exists even if the program unit in which it is defined is not active	③ The variable get allocated only if the program unit gets active.
④ eg: FORTRAN	④ Eg. PL/I, Pascal, Ada etc.

Page No. :
Date :

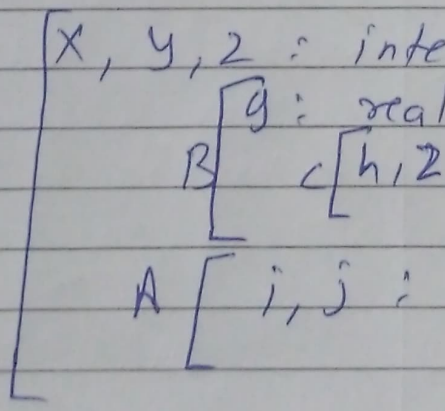
Q2 Explain Scope Rules with example.

Ans. → For data declaration it is possible to use same name in many docks of program.

→ This would establish many bindings of the form (name, var) for different values of a variable.

→ scope rules determines which of these bindings is effective at a specific place in the program.

→ eg.:
Consider the block structured program. The variables accessible within the various block are as follow:-



→ Variable ZA is not accessible inside Block C since C contains a declaration using the name > Thus >A and >C are two distinct variables. This would be true

even if they had identical attributes. i.e. even if 2 of c may declared to be an integer.

Q3) Explain Memory allocation in Recursion.

-> Recursion procedure (or functions) are characterised by the fact that many invocations of a procedure (o exist during the execution of a program.

-> A copy of the local variables of the procedure must be allocated for each invocation.

-> This does not pose any problem when a stack model of memory allocation is used because an activation record is created for every invocation of a procedure or function.

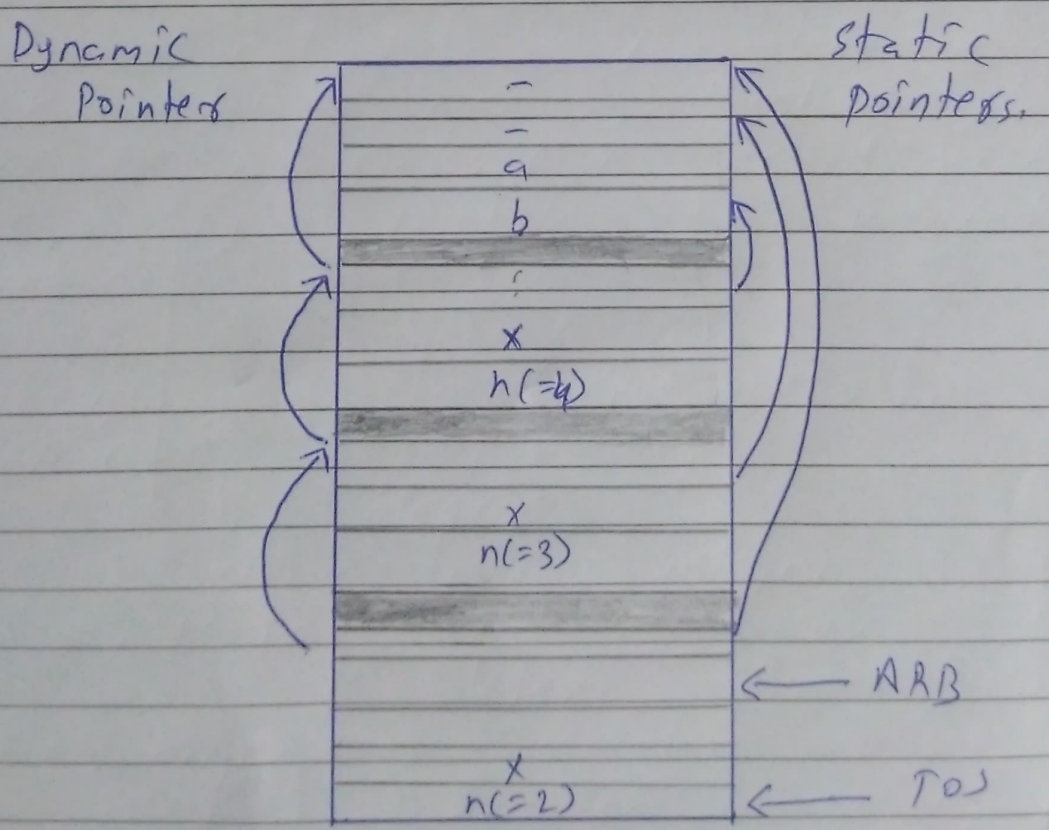
```
Program Sample (input, output);
var
    a, b : integer;
    function fib(n) : integer;
    var
        x : integer
    begin
        if n >2 then
            x := fib(n-1) + fib(n-2);
```

Page No. :
Date :

```
    else
        x = 1;
    return (x);
end fib;
begin
    fib(4);
end.
```

-> For the call feb(4) the function makes recursive calls.

-> Pictorial representation of recursive call in fibonnacci Series Program



Page No. :
Date :

Q4 Define Dope vector with example.

Ans -> If array dimension bounds are not known during compilation the dope vector has to exist during program execution.

-> The number of dimension of an array determines the format and size of its DV.

-> The Dope vector is allocated in the AR of a block and dDV, its displacement in AR. is noted in the symbol table entry of the array.

-> The array is allocated dynamically by repeating step for the array.

-> It start address values of lj, uj and range j are entered in the DV

-> The generated code uses dDV to access the contents of DV to check the validity of subscripts and compute the address of an array element.

Q5. Define with example:-

(i) Operand descripter:-

It has following fields:-

① Attributes :- Contains the subfield type length and miscellaneous information.

Page No. :
Date :

② Addressability :- specifies where the operands is located and how it can be accessed. It has 2 sub fields
① Addressability code.
② Address.

→ An operand descriptor is built for every operand participating in an expression.

→ An operand descriptor is built for an id when the id is reduced during parsing

→ A partial result pri is the result of evaluting same operater opj. A descriptor is built for Pri immediately after code is generated for operator opj.

→ for simplicity, assume that all operand descriptors are stored in an array called operand-descriptor.

eg:-

```
MOVER AREG, A
MULT AREG, B
```

Page No. :
Date :

(ii) Register Descripter :-

⇒ A register descripter has two field.

① status :-
Contains the code free or occupied to indicate register status.

② Operand descripter # :-
If status = occupied this field contains the descripter # for the operand contained in the register

⇒ Register descriptors are stored in an array called Register_descripter.

⇒ one register descriptor exists for each CPU register.

eg :-
The register descripter for AREG after generating code for a+b

| Occupied | #3 |

This indicates that register AREG contains the operand description by descripter #3

Page No. :
Date :

Q6 Write the steps for Assembly lang code Generation from an expression specified in high level language.

→ Expression :- x + y * p + q

steps :-

Step no	Parsing action	code generation
1	<id>x → F^1	Build descriptor #1
2	$F^1 \rightarrow T^1$	–
3.	<id>y → F^2	Build descriptor #2
4.	$T^1 + F^2 \rightarrow T^3$	Generate MOVER AREG
5.	$T^3 \rightarrow E^3$	–
6.	<id>p → F^4	Build descriptor #4
7.	$F^4 \rightarrow T^4$	–
8.	<id>q → F^5	Build descriptor #5

9.	$T^4 * F^5 \rightarrow T^6$	Generate MOVEM AREG TEMP 1
		MOVER AREG, P
		ADD AREG, Q
		Build descriptor #6

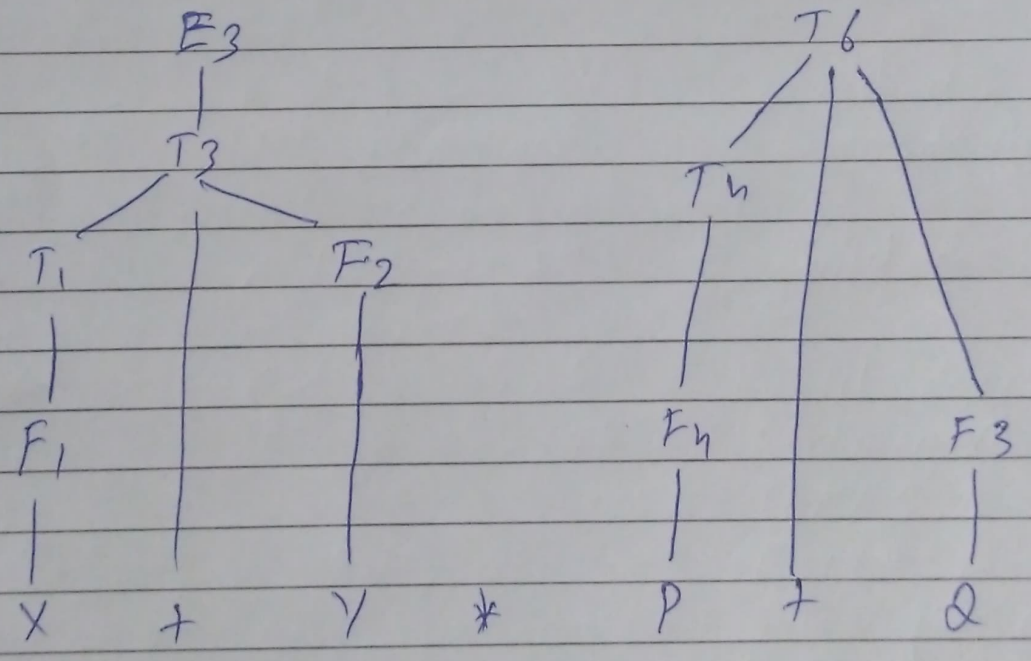
10.	$E^3 + T^6 \rightarrow E^7$	Generate MULT AREG, TEMP 1

Page No. :
Date :

→ Operand descriptors :-

1	(int, 2)	M, addr(x)
2	(int, 2)	M, addr(y)
3	(int, 2)	M, addr(temp[i])
4	(int, 1)	M, addr(p)
5	(int, 1)	M, addr(q)
6	(int, 1)	R, addr(REG)

=>



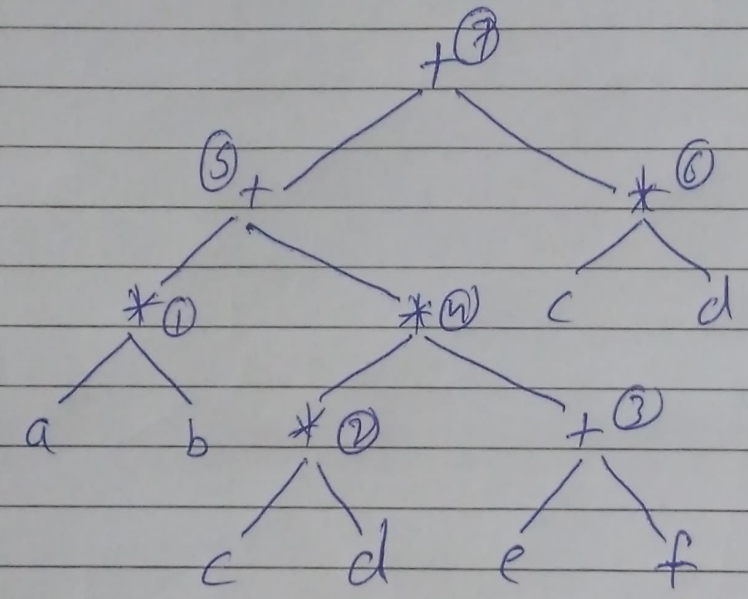
=> Register descriptor

occ	6

Page No. :
Date :

Q7 With an example, Explain the alternative of assembly language code generation for expression

Ans $x * y + P * Q (R + S) + Q * Q$




```
MOVER  AREG, X
MULT   AREG, Y
MOVEM  AREG, TEMP_1
MOVER  AREG, P
MULT   AREG, Q
MOVEM  AREG, TEMP_2
MOVER  AREG, R
MOVER  AREG, S
MULT   AREG, TEMP_2
ADD    AREG, TEMP_3
MOVER  AREG, P,
```

```
MULT AREG Q
ADD AREG, TEMP_3
```

Page No. :
Date :

Q8 How postfix string can be used for generating the intermediate code for expression

Ans → In the postfix notation each operator appears immediately after its last operand.

→ Thus a binary operator OPi appears after its second operand. If any of its operand is itself an expression involving an operator OPj, OPj must appear before OPi.

eg :- +a+b*c+d*e↑f—|
source string

postfix string :- +abc*+def↑*++|

→ The postfix string is a popular intermediate code in non-optimizing compiler due to ease of generation and use.

→ The code generation from the postfix string using a stack of operand descripter.

→ Operand descripter are pushed on the stack as operand appear in string

→ When an operator with arity k appears in

Page No. :
Date :

the string & descriptors are popped off the stack.

→ A descripter for the partial result generated by the operator is now pushed on the stack.

→ Conversion of infix to postfix is modified operands appearing in the source string are copied into the postfix string straight away.

→ The TOS operator is popped into the postfix string of TOS operator ≥ current operator.

29. How triples and quadraples can be used for generating the intermediate code for expression.

Ans → Triples: - A triples is a representation of an elementary operation in the form of a pseudo machine instruction.

Format :-

| Operator | operand 1 | operand 2 |

→ Each operand of a triple is either a variable / constant or the result of some

Page No. :
Date :

evaluation represented by another triple.

→ Conversion of infix string into triples can be achieved by a slight modification

e.g.
infix string :- ↑ a + b * c + d * e ↑ f ↑

⇒ Quadruples :-

→ A Quadruples represents an elementary evaluation in the following format :-

Operator	operands	operand2	Result Name

Q10. List the steps taken by the compiler at function & procedure calls.

Ans While implementing a function call the compiler must the following :-

① Actual parameter are accessible in the called function

② The called function is able to produce side effect according to the rules of the PL.

Page No. :
Date :

(3) Control is transffered to and is returned from the called function.

(4) The function value is returned to the calling Program.

(5) All other aspects of execution of a calling programs are unaffected by the function calls.

- Compiler used a set of feature to implement function
  - (i) Parameter list
  - (ii) calling conventions
  - (iii) Save area.

Q11 Different function calls.

Ans Different types of function calls are :-
(i) Call by Value :- In this mechanism Value of actual Parameter are passed to the called function.

(ii) Call by Value-Result :- In this mechanism the value of formal parameters is copied back into actual Parameter

(iii) Call by reference :- In this the address

Page No. :
Date :

of actual parameter is passed to the called function

(iv) Call by Name :- In this also the same mechanism is follow of the call by reference.

-> Here also change in formal Parameter affect the value of actual parameter.