CHAPTER → 4

STACKS

:- A stack is a list of elements in which an element may be inserted or deleted only at one end called the top of the stack.

* Operations associated with stacks:-

1) Push used to insert an element into a stack.
2) Pop used to delete an element from stack.

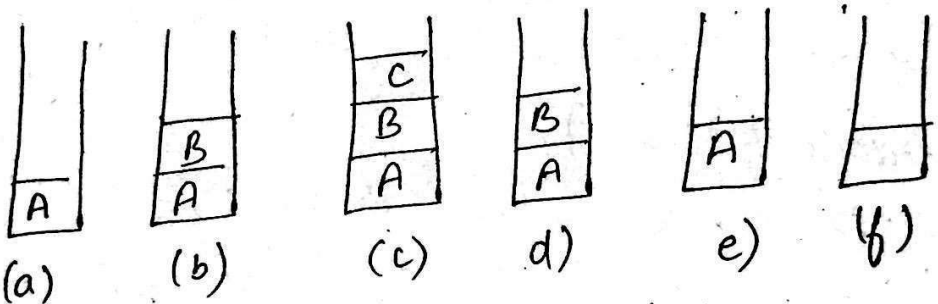


fig (1)

(a) → Insert A at top of stack
(b) → Insert B at top of stack
(c) → Insert C at top of stack
(d) → Delete C from top of stack
(e) → Delete B from top of stack
(f) → Delete A from top of stack.

* Array Representation of Stack

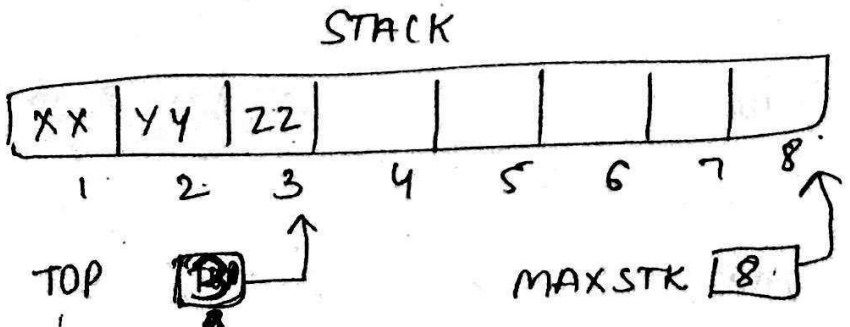


fig (2)

Name of linear array STACK
TOP is a pointer variable contains the

location of the top element of the stack.
TOP = 3 in array STACK.

The condition when TOP = 0 or TOP = NULL indicate that the stack is empty.

MAXSTK is a variable that gives the maximum number of elements that can be held by the stack.

MAXSTK = 8 in array STACK.

Procedure: 1 PUSH (STACK, TOP, MAXSTK, ITEM)

This procedure pushes an ITEM onto a stack.

1. [Stack already filled?]
   If TOP = MAXSTK then Print: OVERFLOW and Return
2. Set TOP := TOP + 1 [Increases TOP by 1]
3. Set STACK [TOP] := ITEM [Inserts ITEM in new TOP position]
4. Return.

Procedure 2: POP (STACK, TOP, ITEM)

This procedure deletes the top element of STACK and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]
   If TOP = 0 then Print: UNDERFLOW and Return
2. Set ITEM := STACK [TOP] [Assigns TOP element to ITEM]
3. Set TOP := TOP - 1 [Decreases TOP by 1]
4. Return.

① PUSH (STACK, WW)

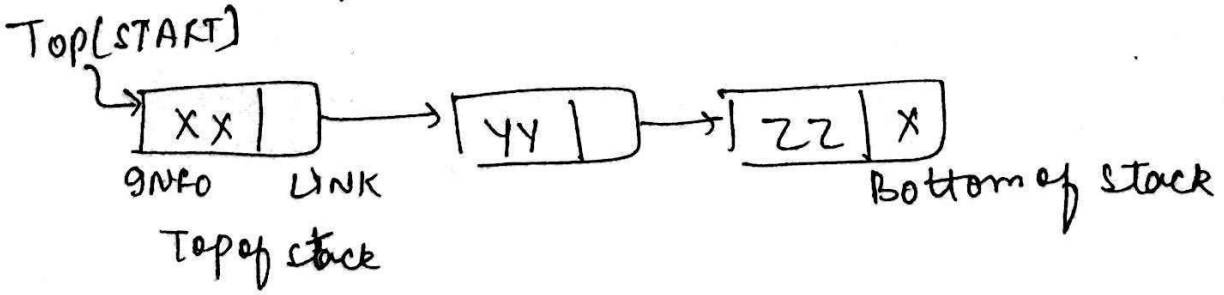
1. Let TOP = 3
2. TOP = 3 + 1 = 4
3. STACK[TOP] = STACK [4] = WW
4. Return

② POP (STACK, ITEM)

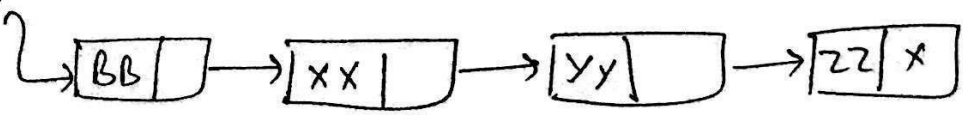
1. let TOP = 3
2. ITEM = ZZ
3. TOP = 3 - 1 = 2
4. Return

* Linked Representation of Stacks :-

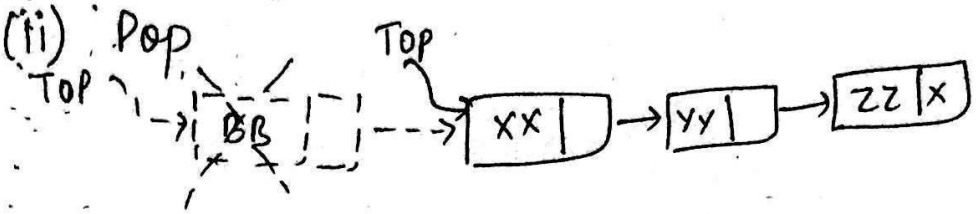
- INFO fields of the nodes hold the elements of the stack and LINK fields hold pointers to the neighbouring element in the stack.
- START pointer of the linked list behaves as the TOP pointer variable of the stack.
- NULL pointer of the last node in the list signals the bottom of stack.



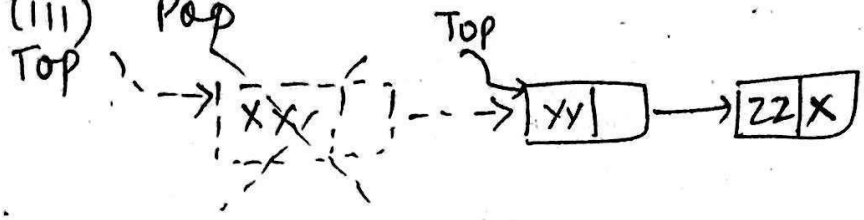
(i) Push BB onto stack.



(ii) Pop



(iii) Pop



Procedure :- PUSH_LINKSTACK (INFO, LINK, TOP, AVAIL, ITEM)

This procedure pushes an ITEM into a linked stack.

1. [Available space ?] If AVAIL = NULL then write : OVERFLOW and Exit.
2. [Remove first node from AVAIL list] Set NEW := AVAIL and AVAIL := LINK[AVAIL].
3. Set INFO[NEW] := ITEM [Copies ITEM into new node].
4. Set LINK[NEW] := TOP [New node points to the original top node in the stack]
5. Set TOP := NEW [Reset TOP to point to the new node at the top of the stack]
6. Exit.

Procedure:- POP_LINK STACK (INFO, LINK, TOP, AVAIL, ITEM)

This procedure deletes the top element of a linked stack and assigns it to the variable ITEM.

1. [Stack has an item to be removed?]
   If TOP = NULL then write: UNDERFLOW and Exit.
2. Set ITEM := INFO [TOP] [Copies the top element of stack into ITEM]
3. Set ITEM := TOP and TOP := LINK [TOP]
   [Remember the old value of the TOP pointer in ITEM and reset TOP to point to the next element in the stack]
4. [Return deleted node to the AVAIL list]
   Set LINK [ITEM] := AVAIL and AVAIL = ITEM
5. Exit.

* Arithmetic Expressions; Polish Notation

Levels of Precedence

Highest :- Exponential (↑)
Next highest :- Multiplication (*) and division (/)
Lowest :- Addition (+) and Subtraction (−)

Ques: Evaluate the following parenthesis-free arithmetic expression:-

$$2 \uparrow 3 + 5 * 2 \uparrow 2 - 12/6$$

Step:1 Evaluate exponentiation

$$8 + 5 * 4 - 12/6$$

Step2: Evaluate multiplication and division

$$8 + 20 - 2$$

Step3: Evaluate addition and subtraction

$$= 26 \quad \text{Ans}$$

* Polish Notation:- Refers to the notation in which the operator symbol is placed before its two operands.

eg +AB , -CD , *EF , /GH

also called Prefix notation.

Infix Notation:- When the operator symbol is placed between its two operands.

eg A+B , C-D , E*F , G/H

* Translation of Infix Expression into Polish notation using brackets [ ] to indicate partial translation

$(A+B) * C = [+AB] * C = *+ABC$

$A + (B * C) = A + [*BC] = +A*BC$

$(A+B) / (C-D) = [+AB] / [-CD] = /+AB-CD$

* Reverse Polish notation:- refers to the notation in which operator symbol is placed after its two operands: AB+ CD- EF* GH/

This notation is also called Postfix notation.

Evaluation of a Postfix Expression:-

Algorithm:- This algorithm finds the VALUE of an arithmetic expression P written in postfix notation.

1. Add a right parenthesis ")" at the end of P.
2. Scan P from left to right and repeat Steps 3 and 4 for each element of P until ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator ⊗ is encountered then:
   (a) Remove the two top elements of STACK, where A is the top element and B is the next-to-top element.
   (b) Evaluate B ⊗ A.
   c) Place the result of (b) back on STACK.
   [End of If structure]
   [End of Step 2 loop]
5. Set VALUE equal to the top element on STACK
6. Exit.

Example:- P:- 5, 6, 2, +, *, 12, 4, /, -

Soln Add ")" right parenthesis at the end of expression.

	Symbol Scanned	STACK
(1)	5	5
(2)	6	5, 6
(3)	2	5, 6, 2
(4)	+	5, 8
(5).	*	40
(6)	12	40, 12
(7)	4	40, 12, 4
(8)	/	40, 3
(9)	−	37
(10)	)	

* Transforming Infix Expressions into Postfix Expressions:-

Algorithm POLISH (Q, P)

Suppose Q is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression P.

1. Push "(" onto STACK and add ")" at the end of Q
2. Scan Q from left to right and repeat steps 3 to 6 for each element of Q until the STACK is empty!
3. If an operand is encountered, add it to P.
4. If a left parenthesis is encountered, push it onto STACK.

⑤ If an operator ⊗ is encountered then:

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) which has the same precedence as or higher precedence than ⊗.

(b) Add ⊗ to STACK.

[End of If structure]

⑥ If a right parenthesis is encountered, then:

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered

(b) Remove the left parenthesis [Do not add the left parenthesis to P.]

[End of If structure]

[End of step 2 loop]

⑦ Exit.

Ques:- Transform the following infix expression Q into postfix expression P.

Q: A + (B * C - (D / E ↑ F) * G) * H

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

add) at the end of Q.

	Symbol scanned	STACK	Expression P
(1)	A	(	A
(2)	+	(+	A
(3)	(	(+(	A
(4)	B	(+(	A B
(5)	*	(+(*	A B
(6)	C	(+(*	A B C
(7)	−	(+(−	A B C *
(8)	(	(+(−(	A B C *
(9)	D	(+(−(	A B C * D
(10)	/	(+(−(/	A B C * D
(11)	E	(+(−(/	A B C * D E
(12)	↑	(+(−(/↑	A B C * D E
(13)	F	(+(−(/↑	A B C * D E F
(14)	)	(+(−	A B C * D E F ↑ /
(15)	*	(+(−*	A B C * D E F ↑ /
(16)	G	(+(−*	A B C * D E F ↑ / G
(17)	)	(+	A B C * D E F ↑ / G * −
(18)	*	(+*	A B C * D E F ↑ / G * −
(19)	H	(+*	A B C * D E F ↑ / G * − H
(20)	)		A B C * D E F ↑ / G * − H * +

It is an application of Stacks.

Quicksort is an algorithm of divide & conquer type.

(44), 33, 11, 55, 77, 90, 40, 60, 99, (22), 88, 66.

Interchange 44 with element smaller than 44. ← Scan from right to left.

22, 33, 11, (55), 77, 90, 40, 60, 99, (44), 88, 66.

→ Scan from left to right.
Compare 44 with each element & interchange 44 with number greater than 44.

22, 33, 11, (44), 77, 90, (40), 60, 99, 55, 88, 66.

22, 33, 11, 40, (77), 90, (44), 60, 99, 55, 88, 66.

22, 33, 11, 40, 44, 90, 77, 60, 99, 55, 88, 66.

First sublist — second sublist

Now 44 is at middle position.
left side elements are smaller than 44 and right side elements are larger than 44.
Thus divide the list into two parts & solve two sublist similarly.

(22) 33, (11), 40 (i) Sublist

11, (33), (22), 40

11, 22, 33, 40 sorted

(ii) Second sublist

(90), 77, 60, 99, 55, 88, (66)

66, 77, 60, (99), 55, 88, (90)

66, 77, 60, (90), 55, (88), 99

66, 77, 60, 88, 55, 90, 99

at mid

(iii) sublist

(66), 77, 60, 88, (55)

55, (77), 60, 88, (66)

55, (66), (60), 88, 77

55, 60, 66, 88, 77

(iv) sublist

77, 88

Sorted sublist 2

55, 60, 66, 77, 88, 90, 99

Combine all sublist

sublist 1 + 44 + Sublist 2

11, 22, 33, 40, 55, 60, 66, 77, 88, 90, 99.

Sorted list

mplexity of Quick sort = $O(n^2)$.

Algorithm from Book pg 6.16, 6.17.

* Recursion:- Suppose P is a procedure containing either a Call statement to itself or a Call statement to a second procedure that may eventually result in a Call Statement back to the original procedure P. Then P is called a recursive procedure. So that program will not continue to run indefinitely Recursive Procedure must have the following two properties :-

(i) There must be certain criteria, called base criteria, for which the Procedure does not call itself.

(ii) Each time the procedure does call itself (directly or indirectly) it must be closer to the base criteria.

(iii) Factorial function :-

FACTORIAL(FACT, N)

This procedure calculates N! and returns the value in the variable FACT.

1. If N = 0, then set FACT := 1 and Return
2. Set FACT := 1 [Initializes FACT for loop]
3. Repeat for K = 1 to N
   Set FACT := K * FACT
   [End of loop]
4. Return.

Example : $4! = 4 \times 3 \times 2 \times 1 = 24$

(2) Fibonacci Sequence :-

FIBONACCI (FIB, N)

This procedure calculates F_N and returns the value in the first parameter FIB.

(1) If N=0 or N=1 then set FIB := N and Return
(2) Call FIBONACCI (FIBA, N-2)
(3) Call FIBONACCI (FIBB, N-1)
(4) set FIB := FIBA + FIBB
(5) Return.

2013

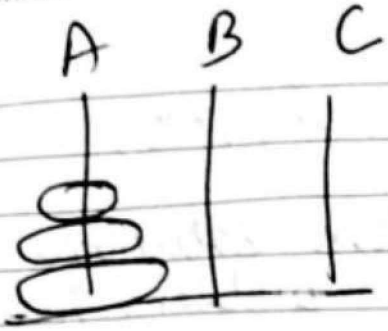
TOWER OF HANOI

MARCH

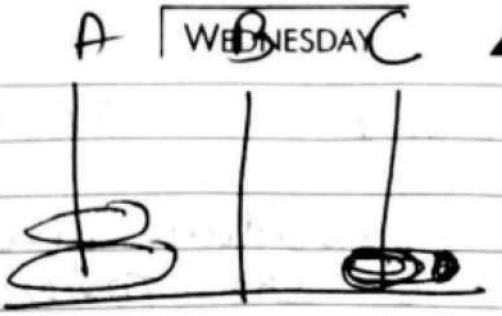
IMPORTANT

WEEK-13 | 086-279

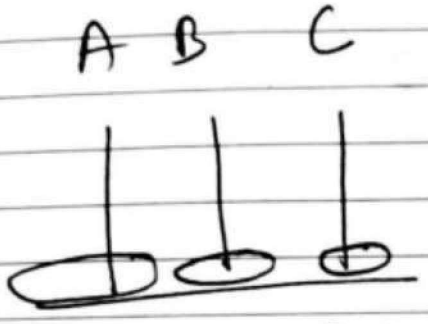
WEDNESDAY 27



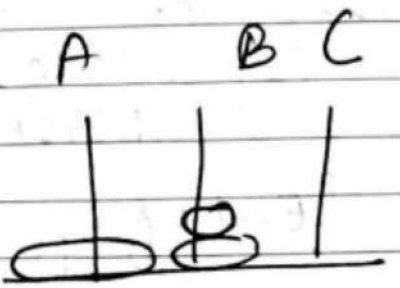
a) initial



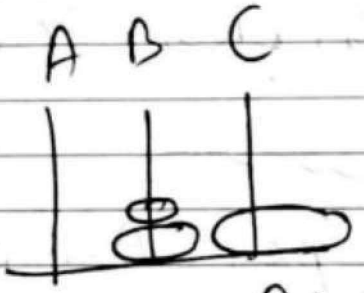
(1) A → C



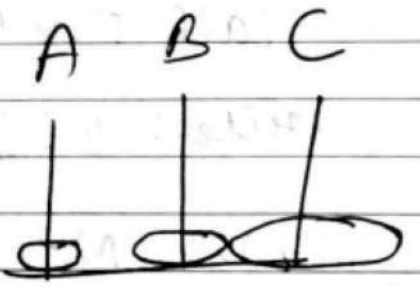
(2) A → B



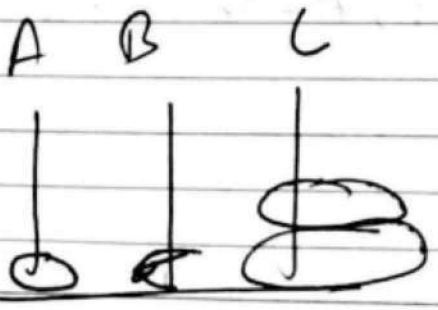
(3) C → B

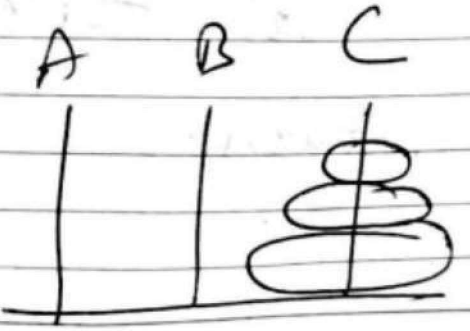


(4) A → C



(5) B → A





(7) A → C

THINGS TO DO

B → C

APRIL '13

M	T	W	T	F	S	S
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30					

MAY

M	T	W	T	F	S	S
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

APR

MAY

JUN

Procedure : TOWER (N, BEG, AUX, END)

This procedure gives a recursive solution to the Tower of Hanoi problem for N disks.

(1) If N=1 then :-
  (a) Write : BEG → END
  (b) RETURN
  [End of If Structure]

(2) [MOVE N-1 disks from peg BEG to peg AUX]
  Call TOWER (N-1, BEG, END, AUX)

(3) Write : BEG → END

(4) [MOVE N-1 disks from peg AUX to peg END]
  Call TOWER (N-1, AUX, BEG, END)

(5) Return.