Introduction to unit :→ In the last lecture, we studied about array, matrics and multidimensional arrays. In this lecture we'll study about stacks, Queues and Recursion.

Introduction to lecture :→ In linear lists and linear arrays, insertion & deletion takes place at any place. There are certain situations in computer science when one wants to restrict insertions and deletions, so that they can take place only at the beginning or the end of the list. Two of the data structures that are useful in such situations are stacks and queues.

A stack is a linear structure in which items may be added or removed only at one end. Eg. stack of dishes, a stack of folded towels etc.

↳ An item may be added or removed only at one end. i.e. the top of any of the stacks.

↳ Stacks are also called Last-in first-out (LIFO) lists.

Stack of dishes

* 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. This means that elements are removed from a stack in the reverse order of that in which they were inserted into the stack.

↳ Two basic operations associated with stacks :-

(a) "Push" is the term used to insert an element into a stack.

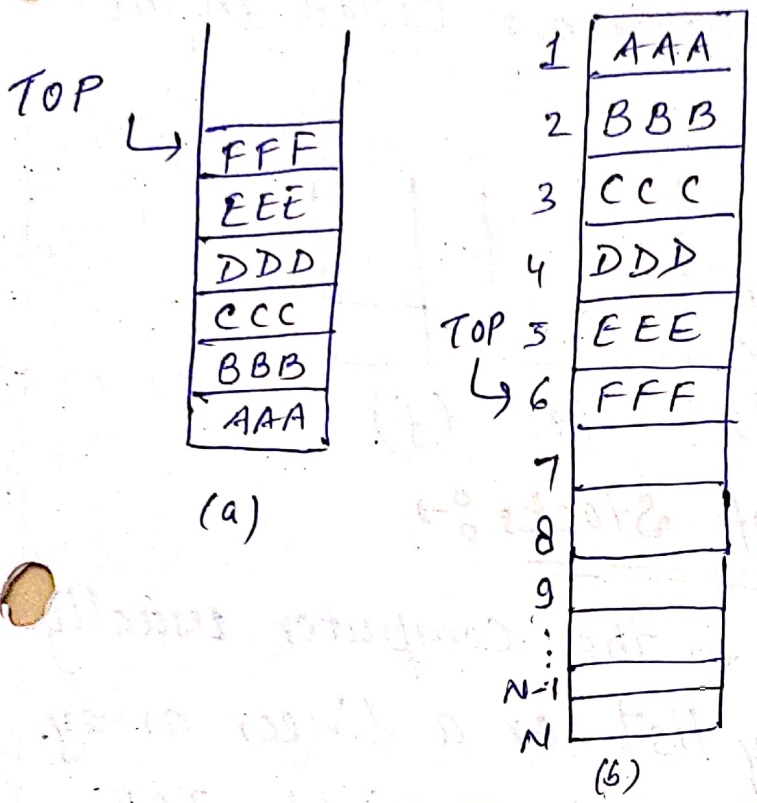
(b) "Pop" is the term used to delete an element from a stack.

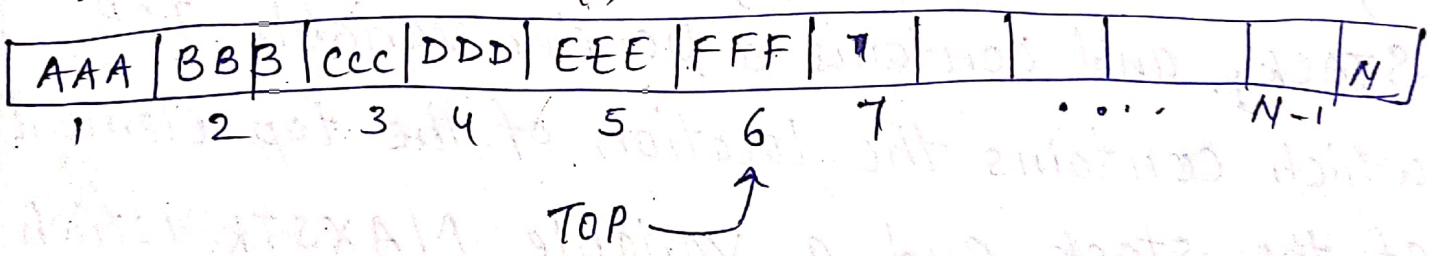
These terms are used only with stacks, not with other data structures.

Eg:- Suppose the following 6 elements are pushed, in order, onto an empty stack:

AAA, BBB, CCC, DDD, EEE, FFF

Three ways of picturing such a stack.



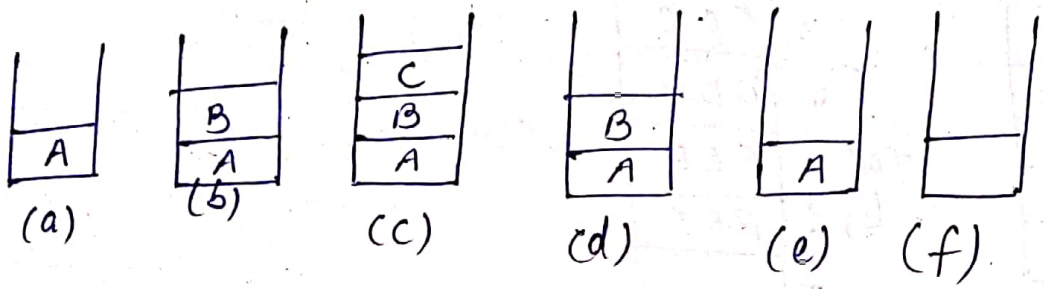


(c)

* EEE cannot be deleted before FFF ~~and~~ is deleted.

* AVAIL list is implemented as a stack.

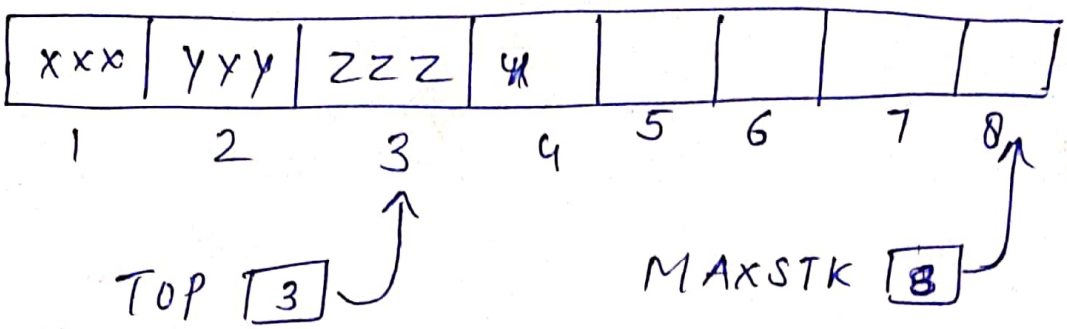
Postponed Decisions :→ Stacks are frequently used to indicate the ~~other~~ order of the processing of data when certain steps of the processing must be postponed until other conditions are fulfilled. An important example of such a processing in computer science is where A is a main program and B, C and D are subprograms called in the order given.



Array Representation of Stacks :→

↳ Stacks is represented in the computer usually by means of a one-way list or a linear array. STACK, and contains a pointer variable TOP, which contains the location of the top element of the stack. and a variable MAXSTK. which gives the maximum no. of elements that can be held by the stack.

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



there is room for 5 more items in the stack.

* Underflow and overflow.

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.

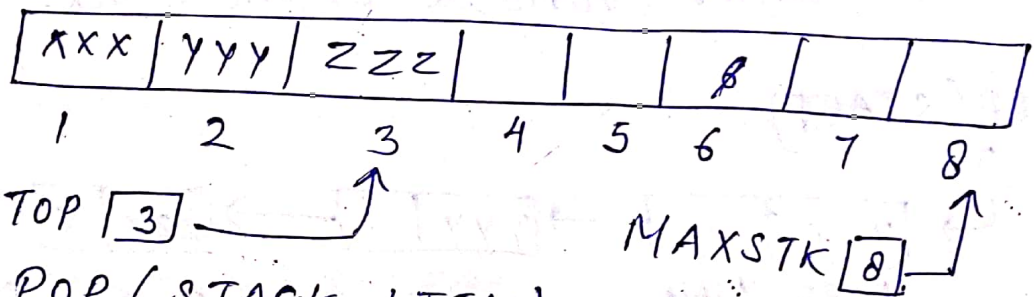
→ On the deletion of an element in the stack, we have to first test whether there is an element in the stack to be deleted; if not then we have the condition known as underflow.

Algorithm:→ 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.

Eg:-



POP (STACK, ITEM):

1. Since TOP = 3, control is transferred to Step 2.

2. ITEM = ZZZ.
3. TOP = 3-1 = 2.
4. Return.

STACK[TOP] = STACK[2] = YYY is now the top element in the stack.

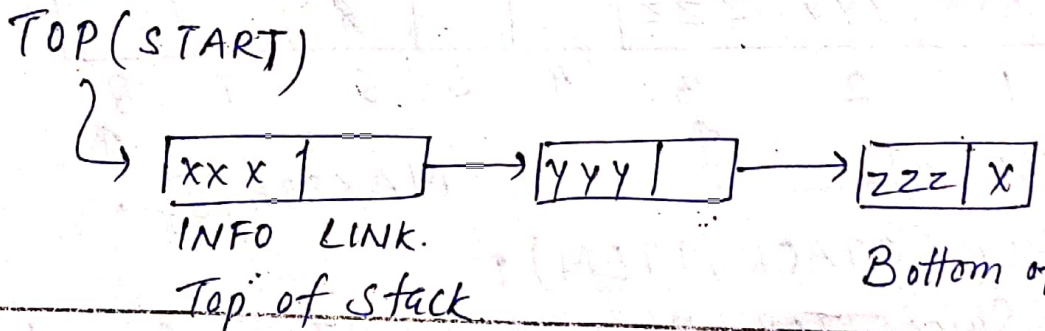
Linked Representation of Stacks :→

Representation of stacks is generally done by using a one way list or singly linked list.

↳ The linked representation of a stack, commonly termed linked ~~list~~ stack is a stack that is implemented using a singly linked list.

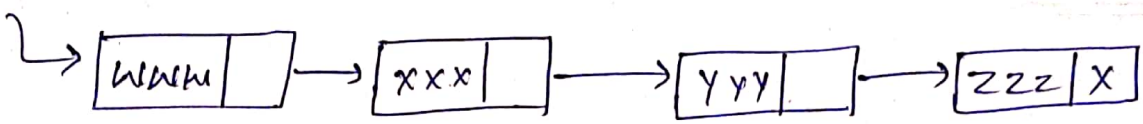
↳ The INFO fields of the nodes hold the elements of the stack and the LINK fields hold pointers to the neighboring element in the stack.

↳ The START pointer of the linked list behaves as the TOP pointer variable of the stack.



↳ A push operation into STACK is accomplished by inserting a node into the front or start of the list and a pop operation is undertaken by deleting the node pointed to by the START pointer.

STACK after Push operation



↳ The linked representation of stacks is free of the overflow condition. There is no limitation on the capacity of the linked stack & hence it can support as many push operations as the free storage list (the AVAIL LIST) can support.

Algorithm :→ 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.

Algorithm:→ POP_LINKSTACK (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 TEMP := TOP and TOP = LINK[TOP]
   [Remember addr the old value of the TOP pointer in TEMP and reset TOP to point to the next element in the stack.]
4. [Return deleted node to the AVAIL list]
   Set LINK[TEMP] = AVAIL & AVAIL = TEMP
5. Exit.

Introduction:- In the last lecture we discussed about post transformation of postfix to infix and viceversa. In this lecture we'll study about recursion & Tower of Hanoi.

Recursion :→ Suppose P is a procedure containing either a Call statement to itself or a Call statement to a second procedure that may result in a Call statement back to the original procedure P. Then P is called a recursive procedure.

So that the program will not continue to run indefinitely, a recursive procedure must have the following two properties :

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

(2) Each time the procedure does call itself, it must be closer to the base criteria.

A recursive procedure with these two properties is said to be well-defined.

Towers of Hanoi :→ In Towers of Hanoi problem, we have three pegs labeled A, B and C and suppose on peg A there are placed a finite no. n of disks with decreasing size. The object of the game is to move the disks from peg A to peg C using peg B as an auxiliary. The rules of the game are as follows :-

(a) Only one disk may be moved at a time.

(b) At no time can a larger disk be placed on a smaller disk.

↳ $X \rightarrow Y$ denotes the instruction "Move top disk from peg X to peg Y" where X and Y may be any of the three pegs.

When $n=3$, we have $2^n - 1 = 2^3 - 1 = 7$ moves.

$A \rightarrow C$
$A \rightarrow B$
$C \rightarrow B$
$A \rightarrow C$
$B \rightarrow A$
$B \rightarrow C$
$A \rightarrow C$

When $n=1$ ∴ we have $A \rightarrow C$

Rather than finding a separate solution for each n, we use the technique of recursion to develop a general soln. First we observe that the solution to the Towers of Hanoi problem for $n>1$ disks may be reduced to the following subproblems:

(1) Move the top $n-1$ disks from peg A to peg B.

(2) Move the top disk from peg A to peg C ∴ $A \rightarrow C$.

(3) Move the top $n-1$ disks from peg B to peg C.

Let us now introduce the general notation.

TOWER (N, BEG, AUX, END)

to denote a procedure which moves the top n disks from the initial peg BEG to the final peg END using the peg AUX as an auxiliary.

↳ When $n=1$, we have

TOWER(1, BEG, AUX, END) consists of a single instruction BEG → END.

But when $N > 1$, we reduced the soln into 3 subproblems:

(1) TOWER(N-1, BEG, END, AUX)

(2) TOWER(1, BEG, AUX, END) or BEG → END

(3) TOWER(N-1, AUX, BEG, END)

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

This procedure gives a recursive solution to the Towers 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.

Stack Applications :→

Major applications of stacks in computer science are in the area of operating systems, compilers etc particularly where LIFO policy is used. Some applications of stack are as follows :-

1. Correct nesting of parenthesis
2. Expression evaluation
3. Conversion of infix expression to postfix & prefix forms.

↳ **Parenthesis Checking or Nesting of Parenthesis :→**

One of the most important application of stack is checking the proper opening & closing of parenthesis in an expression. We can use stack to check the validity of an expression which uses nested parentheses. An expression is valid if it satisfies these two conditions :-

(1) The no. of left parentheses equal to the no. of right parentheses.

(2) Every right parentheses is preceded by a matching left parentheses.