**Unit 4: Stack**



## 4.1 Introduction

- Definition:

A stack is an ordered collection of items into which new items may be inserted and from which items may be deleted at one end, called the TOP of the stack. The insertion and deletion in the stack is done from the TOP of the stack.

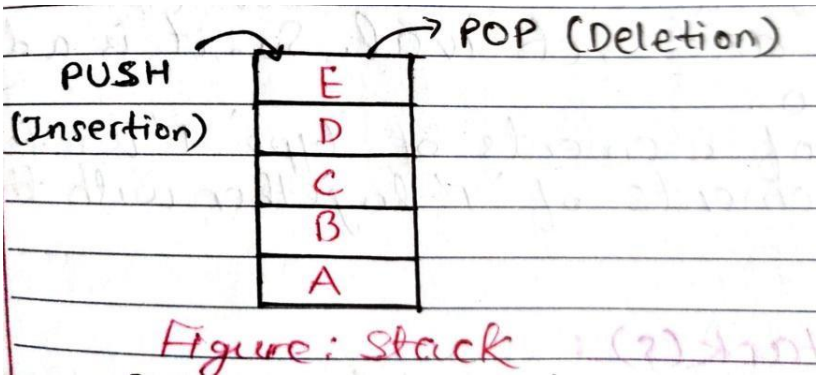



Figure 4.1

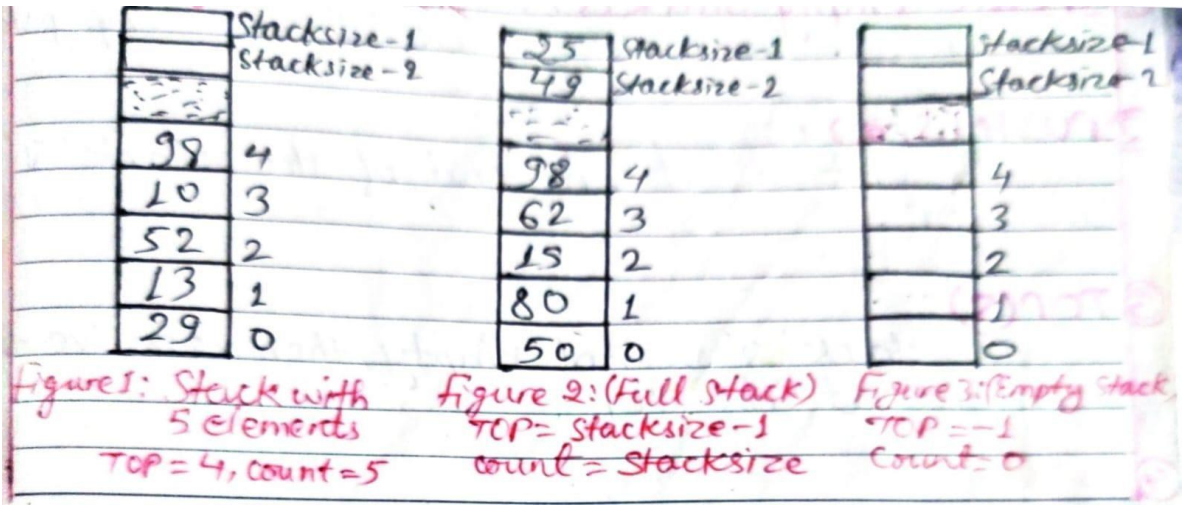



Figure 4.2

- **Stack as an ADT**

A data type can be considered abstract where it is defined in terms of operations in it and its implementation is hidden. A stack is ADT because it hides how it is implemented like using array or linked list. However, it organizes data for efficient management and retrieval. So, it is a data structure also.

A stack of elements of type 'T' is a finite sequence of elements of 'T' together with the following operations.

**1. Create Empty Stack (S)**

Create or make stack 'S' as an empty stack.

**2. PUSH (S, x):**

Insert 'x' at one end of the stack 'S' i.e. TOP.

**3. TOP (S):**

If stack 'S' is not empty, then retrieve the element at its TOP.

**4. POP(S):**

If Stack 'S' is not empty, then delete the element at its TOP.

**5. Is Full (S):**

Determine if 'S' is full or not.
Return TRUE if 'S' is full.
Return FALSE otherwise.

**6. Is Empty (S):**

Determine if 'S' is empty or not. Return TRUE if 'S' is empty.
Return FALSE otherwise.

## 4.2 Array Implementation of Stack (Static Implementation)

Implementation of stack can be done either using array or one way list. The main disadvantage of array implementation of stack is that the size of the array has to be declared ahead. So, the stack size is fixed and dynamically you cannot change the size of stack. Each stack maintains an array STACK, a pointer variable TOP which contains the location of top element and a variable MAXSTK which gives the maximum number of elements that can be held by the stack.

The figure below shows the array representation of stack. Since TOP = 3, the stack has three elements and since MAXSTK =7, more 4 elements can be added to stack

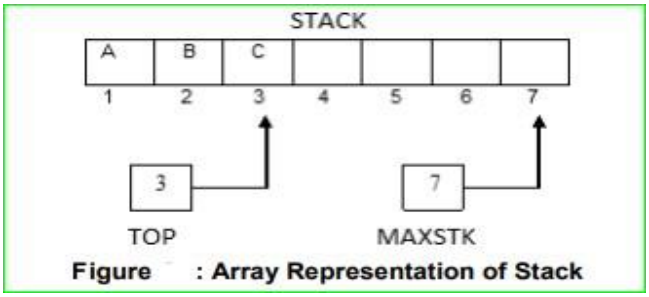



Figure 4.3

The following are the operations to add an item into stack and remove an item from a stack.

1. Push operation:

The push operation is used to add an item into a stack. Before executing push operation one must check for the OVERFLOW condition, i.e. check whether there is room for the new item in the stack. The following procedure performs the PUSH operation.

Procedure: PUSH (STACK, TOP, MAXSTK, ITEM)
1. [Stack already full?]
If TOP = MAXSTK, then: print: OVERFLOW, and Return.
2. Set TOP := TOP + 1. [Increases TOP by 1]
3. Set STACK [TOP]:= ITEM. [Insert ITEM in new TOP position]
4. Return

As per the procedure first it checks for the OVERFLOW condition. Since the stack is not full the TOP pointer is incremented by 1, TOP= TOP + 1. So, now TOP points to a new location and then the ITEM is inserted into that position.

2. Pop operation:

The pop operation is used to remove an item from a stack. Before executing the pop operation one must check for the UNDERFLOW condition, i.e. check whether stack has an item to be removed. The following procedure performs the POP operation

Procedure: POP (STACK, TOP, ITEM)

1. [Stack is empty?]

   If TOP = 0, then: Print: UNDERFLOW, and Return.

2. Set ITEM: = STACK [TOP] . [Assign Top element to ITEM] 3. Set TOP: =

TOP – 1. [Decreases Top by 1]

4. Return.

As per the procedure, first it checks for the underflow condition. Since the stack is not empty the top element in the stack is assigned to ITEM, ITEM: = STACK [TOP]. Then the top is decremented by 1.

**4.3 Linked List Implementation of Stack (Dynamic Implementation)**

Linked list implementation of stack uses singly linked list or one- way list where the DATA field contains the ITEM to be stored in stack and the link field contains the pointer to the next element in the stack. Here TOP points to the first node of linked list which contains the last entered element and null pointer of the last node indicates the bottom of stack.

The figure below shows the linked list representation of stack.

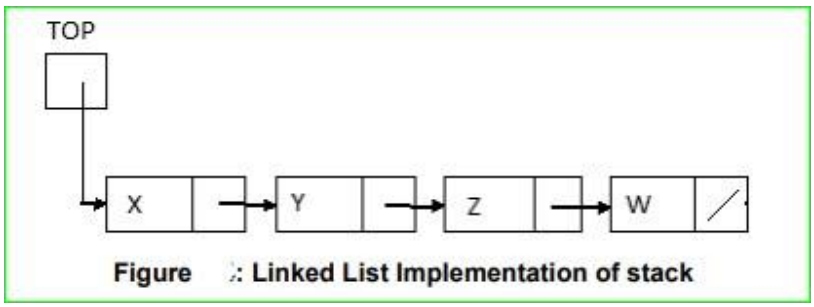



**Figure 4.4**

The following are the operations to add an item into stack and remove an item from a stack.

1. Push operation:

Push operation is performed by inserting a node into the front or start of the list. Even though in this we don't want to declare the size of linked list ahead we have to check for the OVERFLOW condition of the free-storage list.

The following procedure performs the push operation.

Procedure: PUSH (DATA, LINK, TOP, AVAIL, ITEM)

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 DATA [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.

First check the available space in the free storage list. If free space is available then make the pointer variable NEW to point the first node of AVAIL list, NEW: = AVAIL. Get the data for the new node, DATA [NEW]:= ITEM and make the link filed of new node to point the actual top node or first node in the list, LINK [NEW]:= TOP. Now new node becomes the first node in the list and the TOP pointer point to the NEW node.

The figure below shows the PUSH operation in linked list implementation of stack.

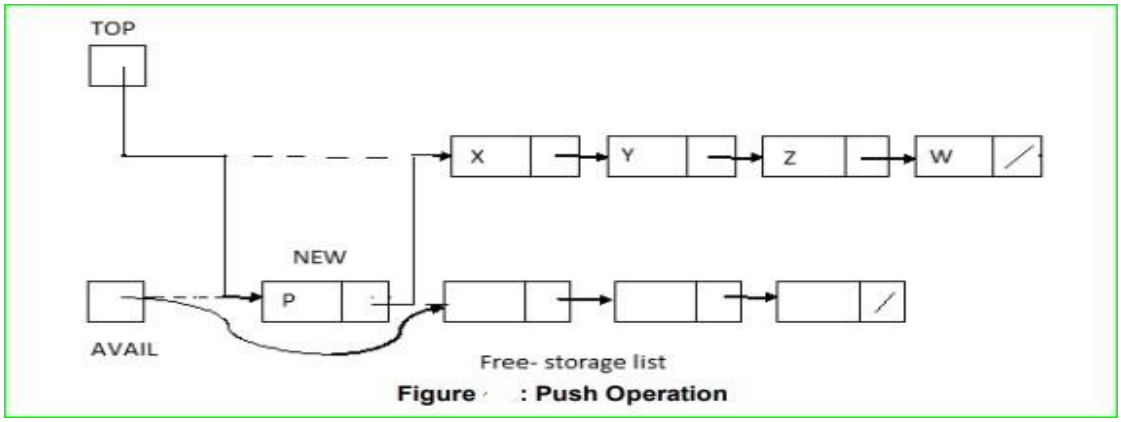



Figure 4.5

2. Pop operation:

Pop operation is performed by removing the first or the start node of a linked list. Before executing the pop operation one must check for the UNDERFLOW condition, i.e. check whether stack has an item to be removed.

The following procedure performs the pop operation.

Procedure: POP (DATA, LINK, TOP, AVAIL, ITEM)

1. [Stack is empty?]
   If TOP = NULL then Write: UNDERFLOW and Exit.
2. Set ITEM: = DATA [TOP] [copies the top element of stack into ITEM]
3. Set TEMP: = TOP and TOP = LINK [TOP]

[Remember 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 AVAL list] Set LINK [TEMP] = AVAIL and AVAIL = TEMP.
5. Exit.

First check the UNDERFLOW condition. If stack is not empty then the element in the top node is copied to the ITEM, ITEM: = DATA [TOP]. A temporary variable TEMP is made to point the top node, TEMP: = TOP and the top pointer now points to the next node in the list, Top = LINK [TOP]. Now the node pointed by TEMP is moved to the AVAIL list, LINK [TEMP] = AVAIL and AVAIL = TEMP.

The figure below shows the POP operation in linked list implementation of stack.

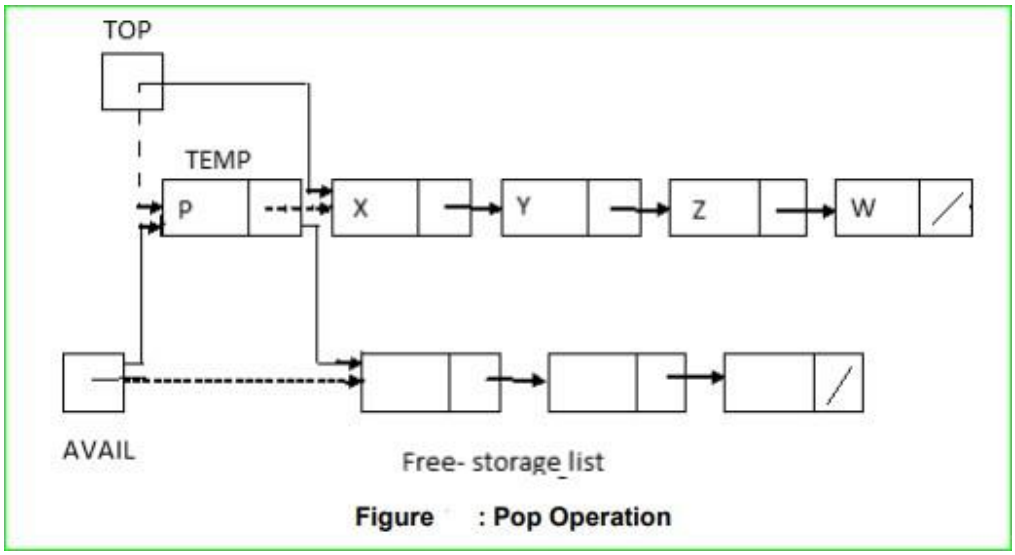



**Figure 4.6**

## 4.4. Applications of Stack

### 4.4.1 Conversion from infix to postfix expression

Generally the expression we use for algebraic notations are in infix form, now we are going to discuss how to convert this infix to postfix, the following algorithm helps us to proceed with. Infix is the input string and the output what we get after the process is the postfix string.

**Algorithm for conversion of infix to postfix**

1. Initialize a STACK as empty in the beginning.
2. Inspect the infix string from left to right. While reading character from a string we may encounter Operand - add with the postfix string. Operator – if the stack is empty push the operator into stack, if any operator available with the stack compares the current operator precedence with the top Stack if it has higher precedence over the current one pop the stack and add with the post string else push the current operator to the stack. Repeat this process until the stack is empty.

   Left Parenthesis: Push in to the STACK.

   Right Parenthesis: Pop everything until you get the left parenthesis or end of STACK.
3. Repeat Process with the infix string until all the characters are read.
4. Check for stack status if it is not empty add top Stack to postfix string and repeat this process until the STACK is empty
5. Return the postfix string.
6. Exit.

Note: Infix precedence

- Parenthesis ()
- Exponentiation ^
- Multiplication *, Division /
- Addition +, Subtraction

**Example for infix to postfix expression**

a + b * c - d

| Infix String | Stack status | Postfix string |
|---|---|---|
| a + b * c - d | Null | Empty |
| + b * c – d | Null | a |
| b * c - d | + | a |
| * c - d | + | ab |
| c - d | * + | ab |
| - d | * + | abc |
| d | - | abc*+ |
| | - | abc*+d |
| | | abc*+d- |

4.4.2 Evaluation of a postfix expression

Suppose we have an arithmetic expression written in postfix notation. By using STACK we are going to evaluate the expression. The following algorithm, which uses a stack to hold operands, evaluates the expression.

**Algorithm:** Evaluation of Postfix Expression.

1. Add a right parenthesis ")" at the end of X.
   [This acts as a sentinel]
2. Scan X from left to right and repeat Steps 3 and 4 for each element of X until the sentinel ")" is encountered.
3. If an operand is encountered, put it on STACK.
4. If an operator is encountered, then:
   (a) Remove the two top element 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.

Let see an example for this.

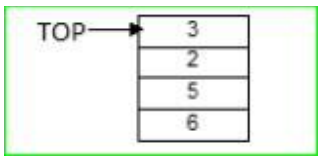
Consider X to be an arithmetic expression written in postfix notation.

X = 6 5 2 3 + 8 * + 3 + *.

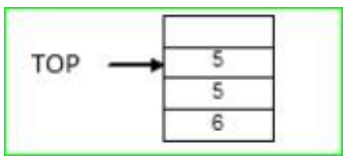
As per the algorithm, first we need to add the right parenthesis ")" at the end of X.

X= 6 5 2 3 + 8 * + 3 + *)

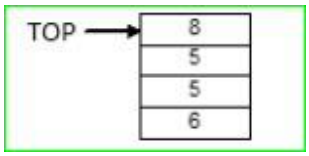
Then we have start the scan from left to right until we get the sentinel ")". So the first four symbols are operands so they are place on the stack.



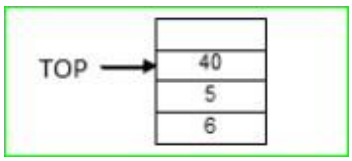

Next to that we have an operator '+', so we pop the top 2 elements i.e. 3 and 2 and their sum, 5 is pushed back into stack.



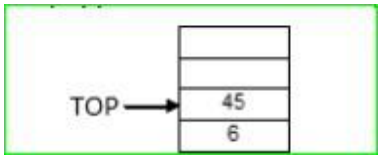

Next 8 is pushed into stack.



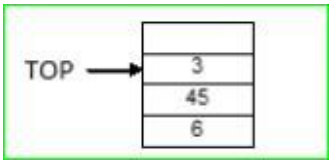

Next is '*', so 8 and 5 are popped and 5*8 = 40 is pushed.



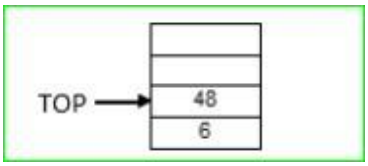

Next is '+', so 40 and 5 are popped and 5+40 = 45 is pushed.




Next 3 is pushed into stack.




Next is '+', so 3 and 45 are popped and 45+3 = 48 is pushed.




Next is '*', so 48 and 6 are popped and 6*48 = 288 is pushed.


Next is ')', so it terminates the loop and the VALUE= 288