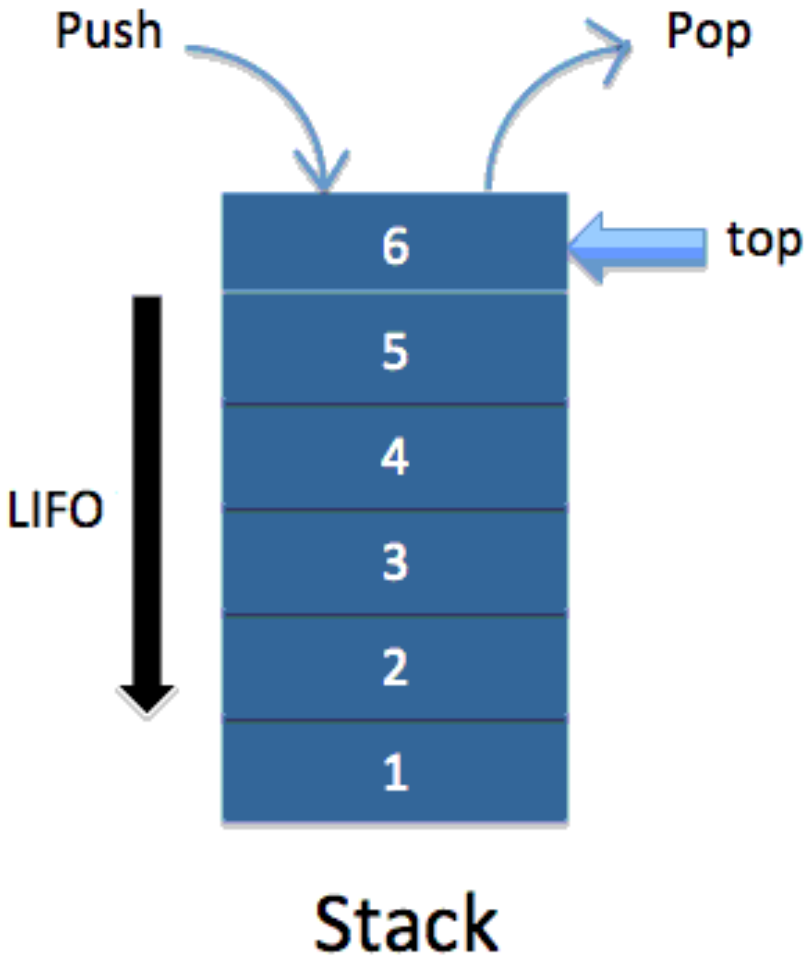
STACK

Concepts & Operations

Prof. Siddharth Shah

Concept

- A linear list which allows insertion and deletion of an element at one end only is called stack.
- The insertion operation is called as PUSH and deletion operation as POP.
- The most and least accessible elements in stack are known as top and bottom of the stack respectively.



Concept

- Since insertion and deletion operations are performed at one end of a stack, the elements can only be removed in the opposite orders from that in which they were added to the stack; such a linear list is referred to as a LIFO (last in first out) list.
- A pointer TOP keeps track of the top element in the stack.
- Initially, when the stack is empty, TOP is not pointing to any element and hence its value is NULL
- Each time a new element is inserted in the stack, the pointer is incremented by “one” before, the element is placed on the stack.
- The pointer is decremented by “one” each time a deletion is made from the stack after the deletion of an element.

Applications (This is not the all)

- Keeping track of function calls
- Recursion
- Evaluation of expressions (Polish Notation – Postfix and Prefix)
- Reversing characters
- Servicing hardware interrupts
- Solving problems using backtracking.
- .....

Operations on Stack

- PUSH
- POP
- PEEP
- CHANGE

PUSH

Algorithm : PUSH (S, TOP, X)

This procedure inserts an element x to the top of a stack which is represented by a vector S containing N elements with a pointer TOP denoting the top element in the stack.

1. [Check for stack overflow]

If TOP ≥ N Then

write ('STACK OVERFLOW')

Return

2. [Increment TOP]

TOP ←TOP + 1

3. [Insert Element]

S[TOP] ←X

4. [Finished]

Return

POP

Algorithm : POP (S, TOP)

This function removes the top element from a stack which is represented by a vector S and returns this element. TOP is a pointer to the top element of the stack.

1. [Check for underflow of stack]

If TOP = 0 Then

Write ('STACK UNDERFLOW ON POP')

/* Take action in response to underflow */

Return

2. [Decrement Pointer]

TOP ← TOP – 1

3. [Return former top element of stack]

Return (S[TOP + 1])

PEEP

Algorithm : PEEP (S, TOP, I)

This function returns the value of the I^{th} element from the TOP of the stack which is represented by a vector S containing N elements. The element is not deleted by this function.

1. [Check for stack Underflow]

If TOP - I +1 ≤ 0 Then

Write ('STACK UNDERFLOW ON PEEP')

/* Take action in response to Underflow */

Exit

2. [Return I^{th} element from top of the stack

Return (S[TOP – I + 1])

CHANGE

Algorithm : CHANGE (S, TOP, X, I)

This procedure changes the value of the I^{th} element from the top of the stack to the value containing in X. Stack is represented by a vector S containing N elements.

1. [Check for stack Underflow]

If TOP – I + 1 ≤ 0 Then

Write ('STACK UNDERFLOW ON CHANGE')

Return

2. [Change I^{th} element from top of the stack]

S[TOP – I + 1] ← X

3. [Finished]

Return

Algorithm: RECOGNIZE

Given an input string named STRING on the alphabet {a, b, c} which contains a blank in its rightmost character position and function NEXTCHAR which returns the next symbol in STRING, this algorithm determines whether the contents of STRING belong to the language **L** shown below.

$L=\{wcw^R \mid w \in \{a,b\}^*\}$, Where w^R is the reverse of w

The vector S represents the stack, and TOP is a pointer to the top element of the stack.

Algorithm: RECOGNIZE

1. [Initialize stack by placing a letter 'c' on the top]

TOP ← 1

S [TOP] ← 'c'

2. [Get and stack symbols till either 'c' or blank is encountered]

NEXT ← NEXTCHAR (STRING)

Repeat while NEXT ≠ 'c'

If NEXT = ' ' Then

Write ('Invalid String')

Exit

Else

Call PUSH (S, TOP, NEXT)

NEXT ← NEXTCHAR (STRING)

3. [Scan characters following 'c'; Compare them to the characters on stack]

Repeat While S [TOP] ≠ 'c'

NEXT ← NEXTCHAR (STRING)

X ← POP (S, TOP)

If NEXT ≠ X Then

Write ('INVALID STRING')

Exit

4. [Next symbol must be blank]

NEXT ← NEXTCHAR (STRING)

If NEXT = ' ' Then

Write ('VALID STRING')

Else

Write ('INVALID STRING')

5. [Finished]

Exit

Example 1 : RECOGNIZE

Input String	Character Scanned	Content of Stack
a b c b a _	None	c
	a	c a
	b	c a b
	c	c a b
	b	c a
	a	c
	_	c
Valid String		

Note: Top element of stack is the rightmost character

Example 2 : RECOGNIZE

Input String	Character Scanned	Content of Stack
a a b c a a b _	None	c
	a	c a
	a	c a a
	b	c a a b
	c	c a a b
	a	c a a
Invalid String: since a ≠ b		

Note: Top element of stack is the rightmost character

Example 3 : RECOGNIZE

Input String	Character Scanned	Content of Stack
a a b c b a a a _	None	c
	a	c a
	a	c a a
	b	c a a b
	c	c a a b
	b	c a a
	a	c a
	a	c
	a	c
Invalid String: since c is top element of stack and NEXT ≠ ' '		

Note: Top element of stack is the rightmost character

Assignment - 1

If following sequence of operations is performed on a stack, what the sequence of values popped out would be?

PUSH (10)

PUSH (20)

POP

PUSH (10)

PUSH (20)

POP

POP

POP

PUSH (20)

POP

Assignment - 2

Following is C like pseudo code of a function that takes a number as an argument, and uses a stack S to do processing. What does the above function do in general?

```
void fun(int n)
{
    Stack S; // Say it creates an empty stack S
    while (n > 0)
    {
        // This line pushes the value of n%2 to stack S
        push(&S, n%2);

        n = n/2;
    }

    // Run while Stack S is not empty
    while (!isEmpty(&S))
    printf("%d ", pop(&S)); // pop an element from S and print it
}
```