Stack and Queue

What is a Stack

- Stack of Books

Stacks

- What can we do with a stack?
  - push - place an item on the stack
  - peek - Look at the item on top of the stack, but do not remove it
  - pop - Look at the item on top of the stack and remove it

Stacks

- A stack is a LIFO (Last-In/First-Out) data structure
- A stack is sometimes also called a pushdown store.
- What are some applications of stacks?
  - Program execution
  - Parsing
  - Evaluating postfix expressions
  - Function calls

Operations of Stack

Push
Pop
Peek
Traversal (Display)

Stack

Stack overflow
Stack underflow

Push

- **Step 1 -** Check whether **stack** is **FULL**. (**top == SIZE-1**)
- **Step 2 -** If it is **FULL**, then display **"Stack is FULL!!! Insertion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT FULL**, then increment **top** value by one (**top++**) and set stack[top] to value (**stack[top] = value**).

Stack overflow

Pop

- **Step 1 -** Check whether **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!! Deletion is not possible!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then delete **stack[top]** and decrement **top** value by one (**top--**).

Stack underflow

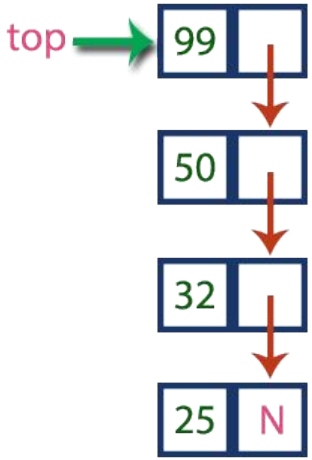
Display

- **Step 1 -** Check whether **stack** is **EMPTY**. (**top == -1**)
- **Step 2 -** If it is **EMPTY**, then display **"Stack is EMPTY!!!"** and terminate the function.
- **Step 3 -** If it is **NOT EMPTY**, then define a variable '**i**' and initialize with top. Display **stack[i]** value and decrement **i** value by one (**i--**).
- **Step 3 -** Repeat above step until **i** value becomes '0'.

Limitations of stack using arrays

- Size of the array should be considered

Stack using linked list



Push

- **Step 1 -** Create a **newNode** with given value.
- **Step 2 -** Check whether stack is **Empty** (**top** == **NULL**)
- **Step 3 -** If it is **Empty**, then set **newNode** → **next** = **NULL**.
- **Step 4 -** If it is **Not Empty**, then set **newNode** → **next** = **top**.
- **Step 5 -** Finally, set **top** = **newNode**.

Pop

- **Step 1 -** Check whether **stack** is **Empty** (**top == NULL**).
- **Step 2 -** If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
- **Step 3 -** If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
- **Step 4 -** Then set '**top = top → next**'.
- **Step 5 -** Finally, delete '**temp**'. (**free(temp)**).

Display

- **Step 1 -** Check whether stack is **Empty** (**top** == **NULL**).
- **Step 2 -** If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.
- **Step 3 -** If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.
- **Step 4 -** Display '**temp → data --->**' and move it to the next node. Repeat the same until **temp** reaches to the first node in the stack. (**temp → next** != **NULL**).
- **Step 5 -** Finally! Display '**temp → data ---> NULL**'.

Evaluation of postfix expression

1) Add `)` to postfix expression.

2) Read postfix expression Left to Right until `)` encountered

3) If operand is encountered, push it onto Stack

[End If]

4) If operator is encountered, Pop two elements

i) `A -> Top` element

ii) `B-> Next to Top` element

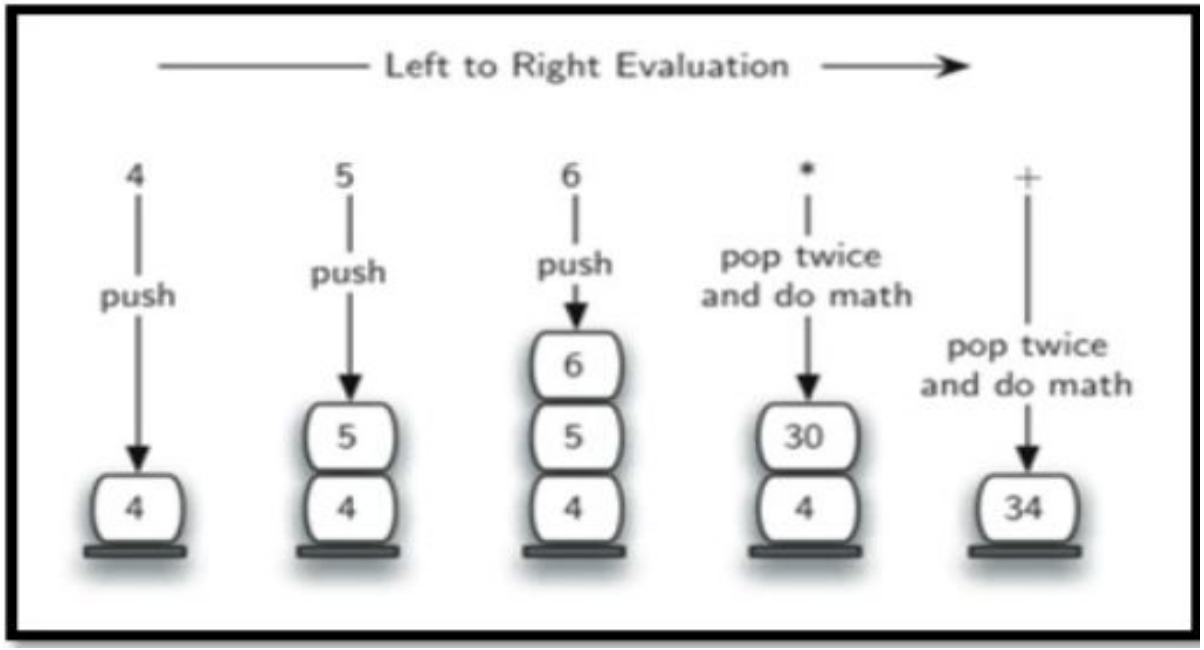
iii) Evaluate `B` operator `A`

push `B` operator `A` onto Stack

5) Set `result = pop`

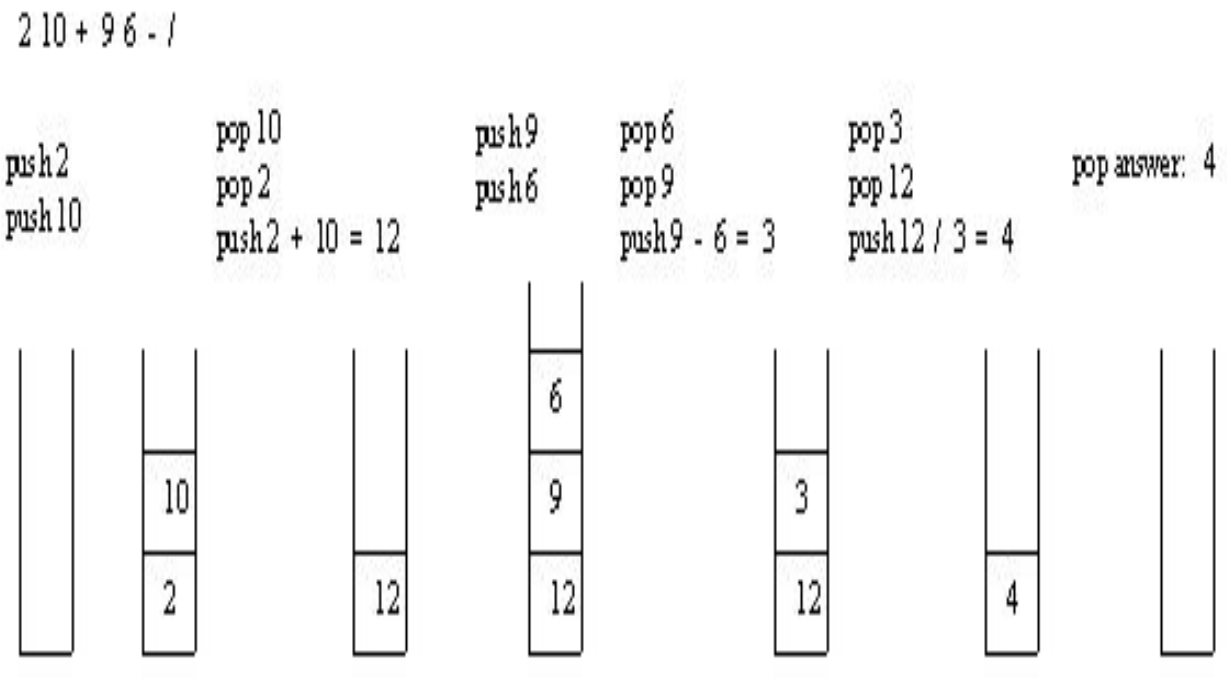
6) END

Expression: 456*+



Step	Input Symbol	Operation	Stack	Calculation
1.	4	Push	4	
2.	5	Push	4,5	
3.	6	Push	4,5,6	
4.	*	Pop(2 elements) & Evaluate	4	5*6=30
5.		Push result(30)	4,30	
6.	+	Pop(2 elements) & Evaluate	Empty	4+30=34
7.		Push result(34)	34	
8.		No-more elements(pop)	Empty	34(Result)

example --2


2 10 + 9 6 - /
push 2
push 10
pop 10
pop 2
push 2 + 10 = 12
push 9
push 6
pop 6
pop 9
push 9 - 6 = 3
pop 3
pop 12
push 12 / 3 = 4
pop answer: 4
10
2
12
6
9
12
3
12
4

try it by yourself

2 3 1 * + 9 -

```
100 200 + 2 / 5 * 7 +
```

46+9*

ANSWERS

1. -4
2. 757
3. 90

Algorithm to convert Infix To Postfix

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the same precedence as or higher precedence than operator.
    2. Add operator to Stack.
    [End of If]
6. If a right parenthesis is encountered ,then:
    1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left parenthesis is encountered.
    2. Remove the left Parenthesis.
    [End of If]
    [End of If]
7. END.

Infix Expression: A+ (B*C-(D/E^F)*G)*H, where ^ is an exponential operator.

Symbol	Scanned	STACK	Postfix Expression	Description
1.		(		Start
2.	A	(	A	
3.	+	(+	A	
4.	(	(+(	A	
5.	B	(+(	AB	
6.	*	(+(*	AB	
7.	C	(+(*	ABC	
8.	-	(+(-	ABC*	'*' is at higher precedence than '-'
9.	(	(+(-(	ABC*	
10.	D	(+(-(	ABC*D	
11.	/	(+(-(/	ABC*D	
12.	E	(+(-(/	ABC*DE	
13.	^	(+(-(/^	ABC*DE	
14.	F	(+(-(/^	ABC*DEF	
15.	)	(+(-	ABC*DEF^/	Pop from top on Stack, that's why '^' Come first
16.	*	(+(-*	ABC*DEF^/	
17.	G	(+(-*	ABC*DEF^/G	
18.	)	(+	ABC*DEF^/G*-	Pop from top on Stack, that's why '^' Come first

Try it by yourself

```
a+b*c
```

```
(a+b)*c+(d-a)
```

```
((4+8)(6-5))/((3-2)(2+2))
```

ANSWERS

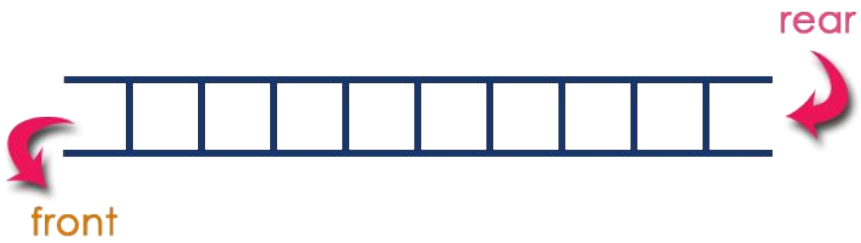
```
a b c * +
```

```
a b + c * d a - +
```

```
4 8 + 6 5 - 3 2 - 2 2 + /
```

QUEUE

- **FIFO (First In First Out)** principle.



Operations on a Queue

1. enQueue(value) - (To insert an element into the queue)
2. deQueue() - (To delete an element from the queue)
3. display() - (To display the elements of the queue)

enqueue

```
void enQueue(int value){
   if(rear == SIZE-1)
      printf("\nQueue is Full!!! Insertion is not possible!!!");
   else{
      if(front == -1&& rear==-1)
          {
          front = 0;
          rear=0;
     }
          else
           rear++;

      queue[rear] = value;
      printf("\nInsertion success!!!");
   }
}
```

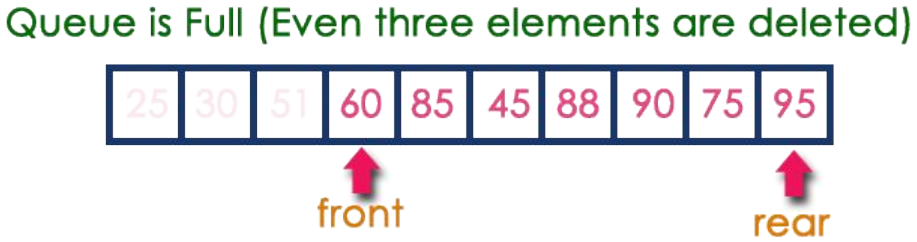
dequeue

```
void deQueue(){
  if(front == -1 && rear==-1)

    printf("\nQueue is Empty!!! Deletion is not possible!!!");

  else if ((front==rear) ||(front>rear))
       {
      printf("\nDeleted : %d", queue[front]);
      front=-1;
      rear=-1;
}
else
    {
        printf("\nDeleted : %d", queue[front]);
    front++;
      }
}
```

Display

```
void display(){
   if(rear == -1)
      printf("\nQueue is Empty!!!");
   else{
      int i;
      printf("\nQueue elements are:\n");
      for(i=front; i<=rear; i++)
         printf("%d\t",queue[i]);
   }
}
```

Drawbacks of queue



Circular queue

- Modulo arithmetic
- Queue full
  - Case 1 full
  - Case 2 empty
  - Few elements

- Queue empty
  - Case 1 empty
  - Case 2 one element
  - Few elements

Priority Queue

Based on value (lowest value / highest value)
Efficient way of realizing priority Q is using Heap
(max /Min heap)