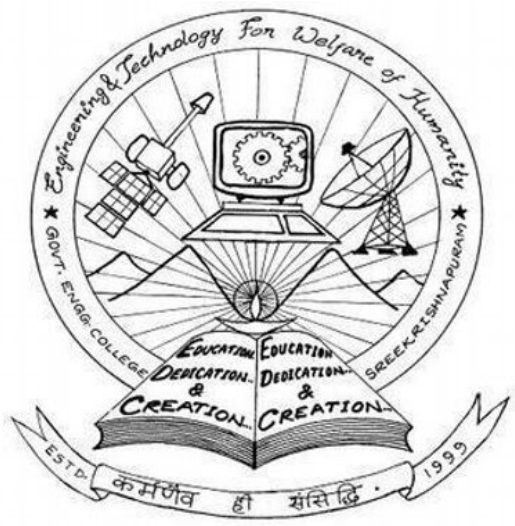
ITT201 Data Structures

Module 3: Stacks and Queues



Anoop S K M
(skmanoop@gmail.com)
Department of Information Technology
Govt. Engg. College, Sreekrishnapuram, Palakkad

Acknowledgements

- All the pictures are taken from the Internet using Google search.
- Wikipedia also referred.

Lecture 13

We Saw & Will See

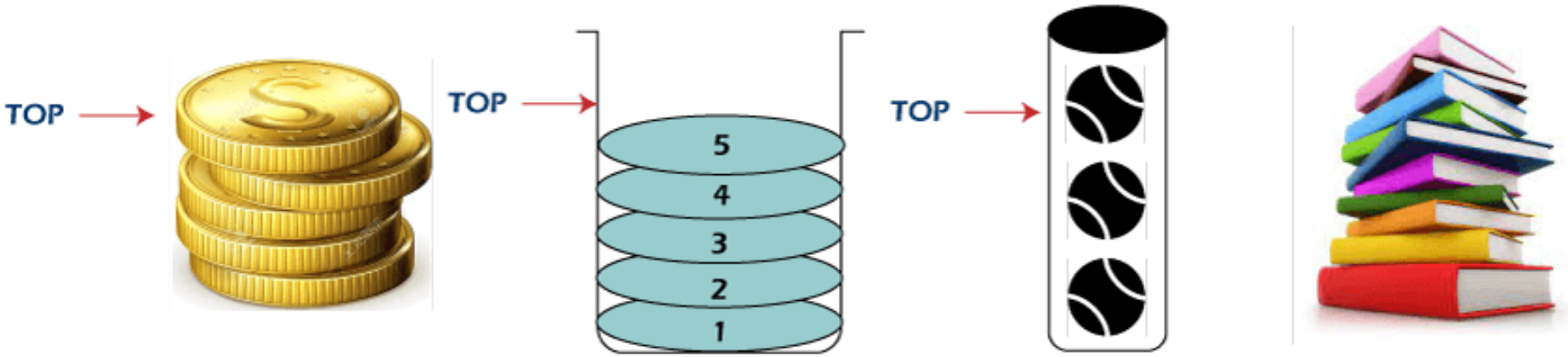
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort

Today We Will See...

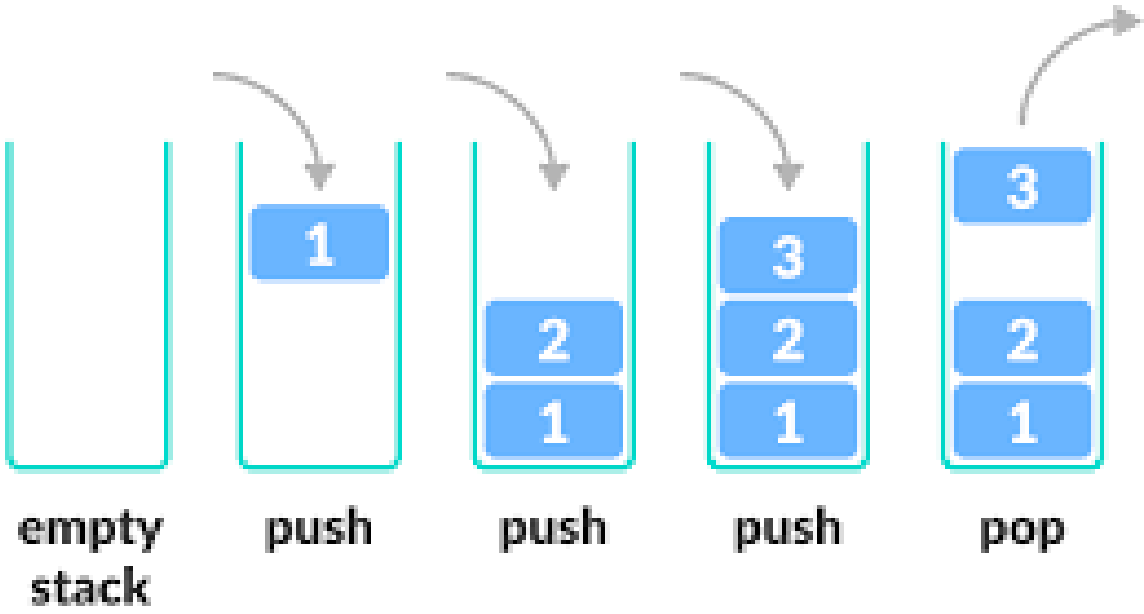
- Module 3 : Stacks and Queues
  - Stack using Array

Stack Examples



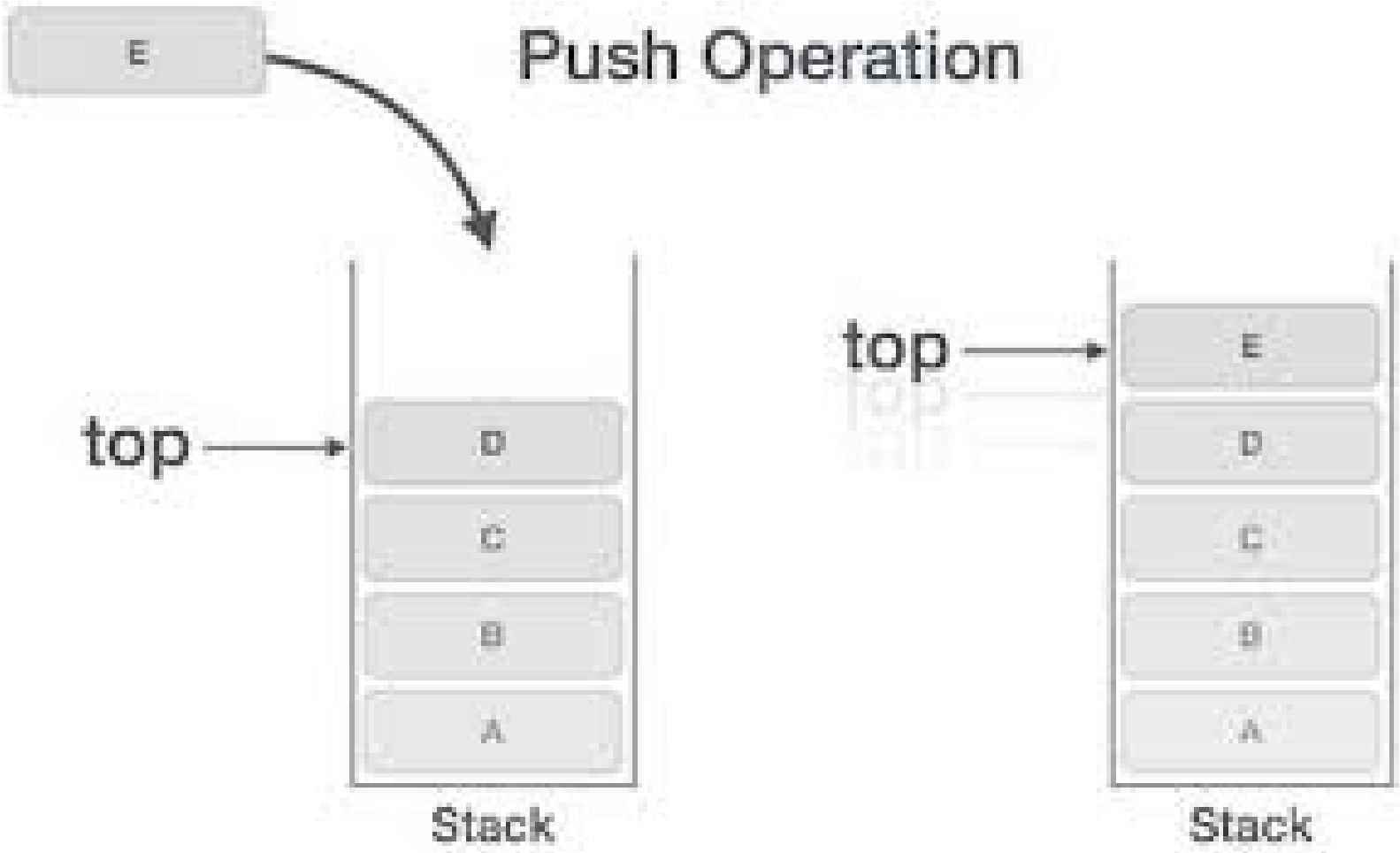
Stack of Coins | Stack of Plates | Can of Tennis Balls | Stack of Books

Stack operations

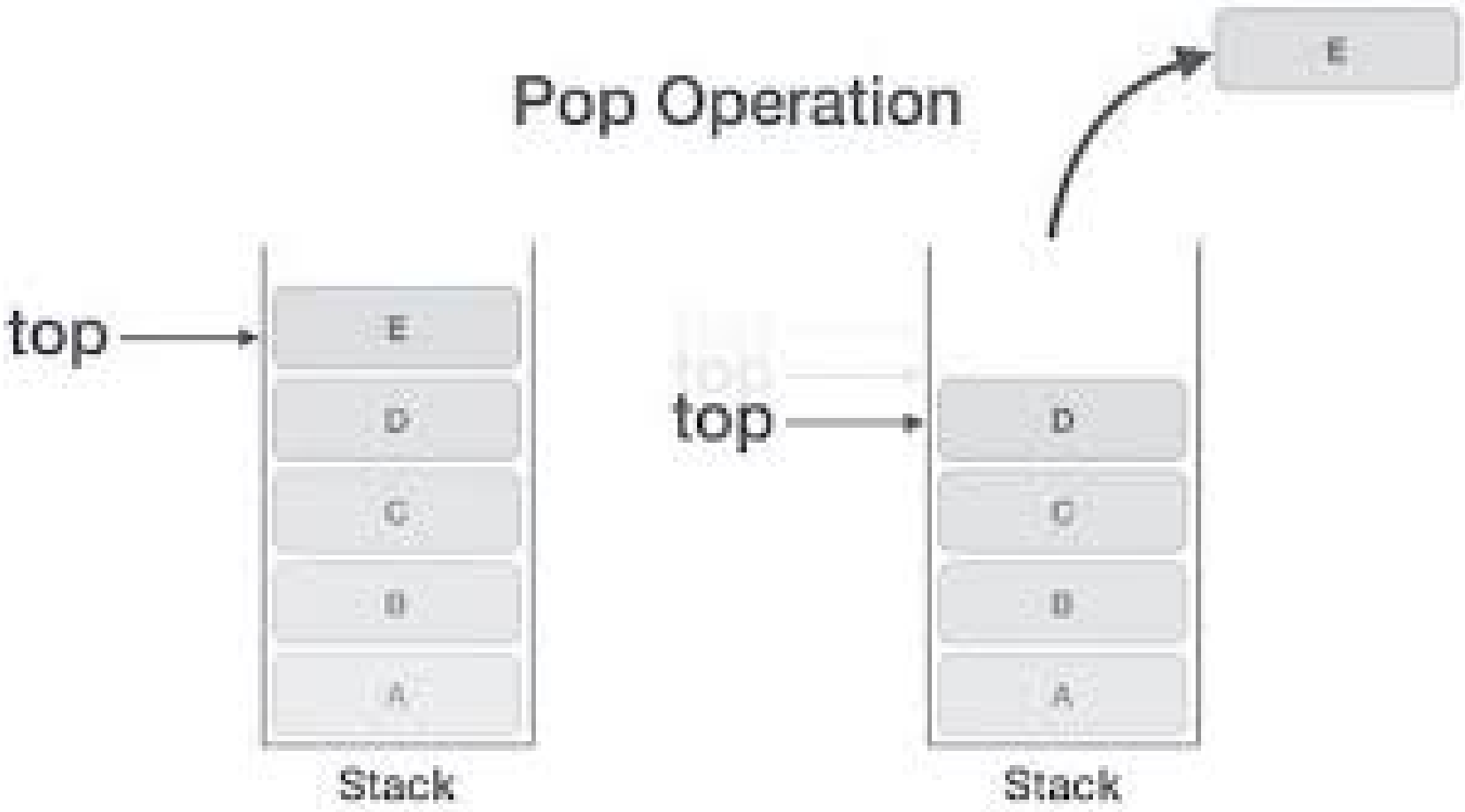


- Stack is a LIFO or FILO
- Insertion in Stack - Push
- Deletion in Stack - Pop

Push

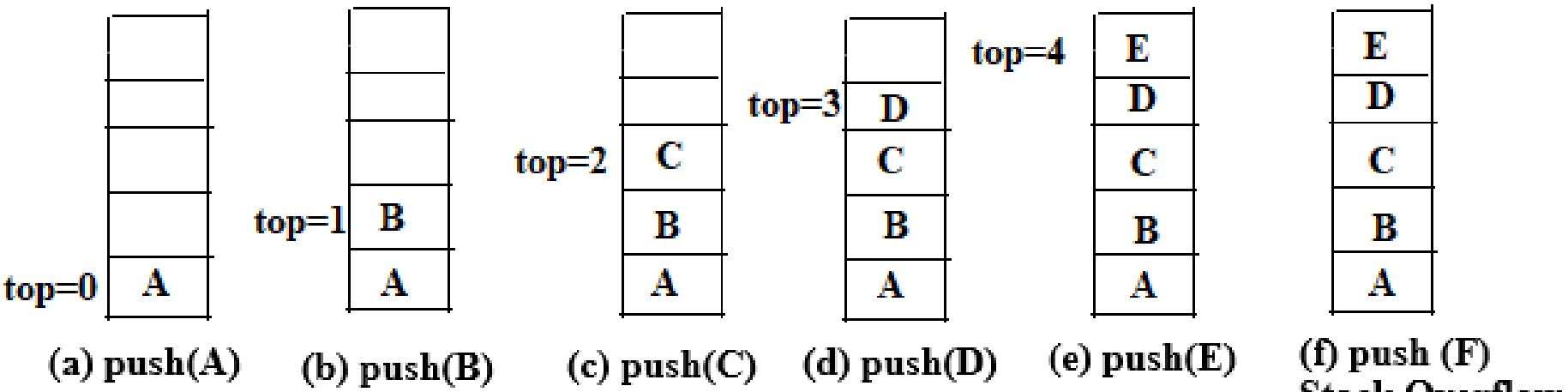


Pop



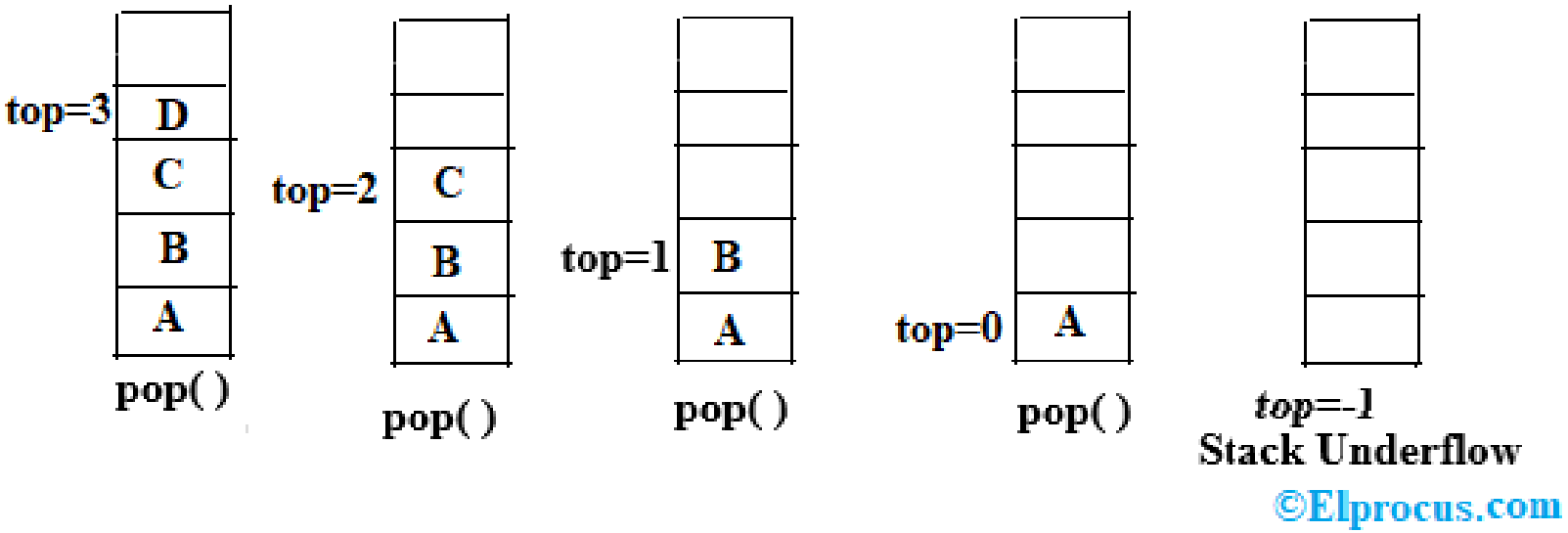
Stack Overflow

- Stack Overflow happens when we try to push an item to a stack that is full.



Stack Underrflow

- Stack Underflow happens when we try to pop an item from a stack that is empty.



isEmpty() & isFull()

- top is used to point to the top most element in the stack
- top $= -1$, indicates that the stack is Empty
- top $= n - 1$, indicates that a stack of size n is Full

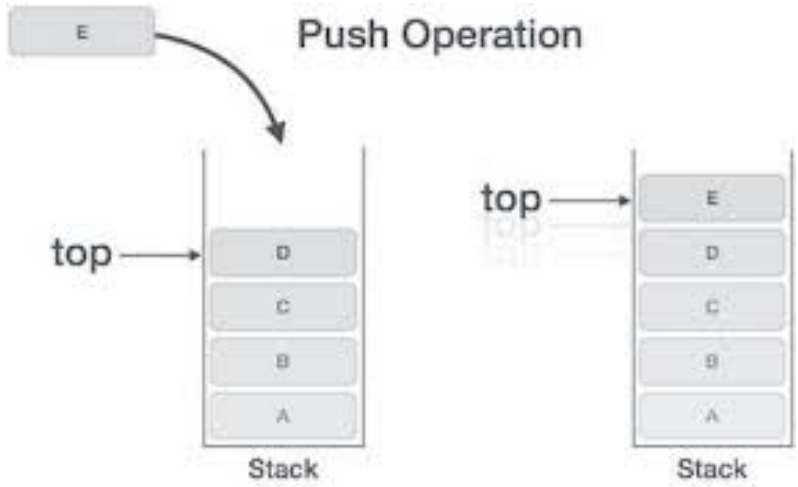
```
int isEmpty( )
{
  if(top == -1)
     return 1;
  else
     return 0;
}
```

```
int isFull()
{
  if(top == n-1)
     return 1;
  else
     return 0;
}
```

Push(int x)

Consider an integer stack S[n] to which an element x is pushed

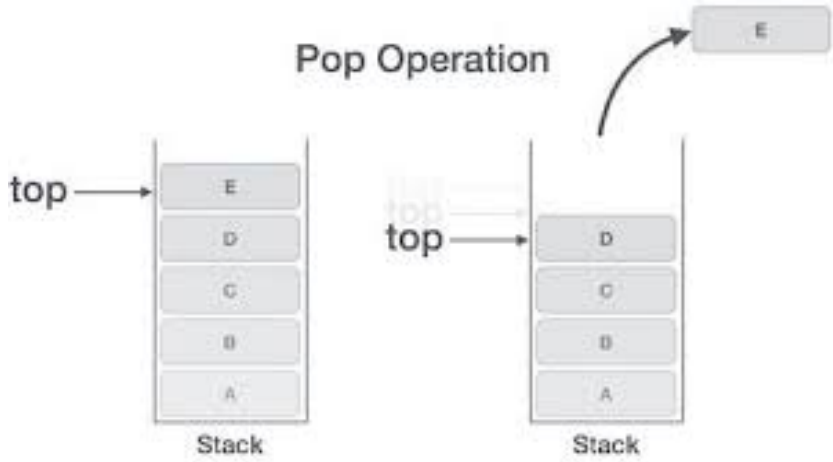
```
void push(int x)
{
  if( !isFull() ){
    top++;
    S[top] = x;
  }
}
```



Pop( )

Consider an integer stack S[n] from which an element is popped

```
int pop( )
{
  if( !isEmpty() ){
    x = S[top];
    top--;
    return x;
  }
 }
```



Lecture 14

We Saw & Will See

Till Now We Saw...

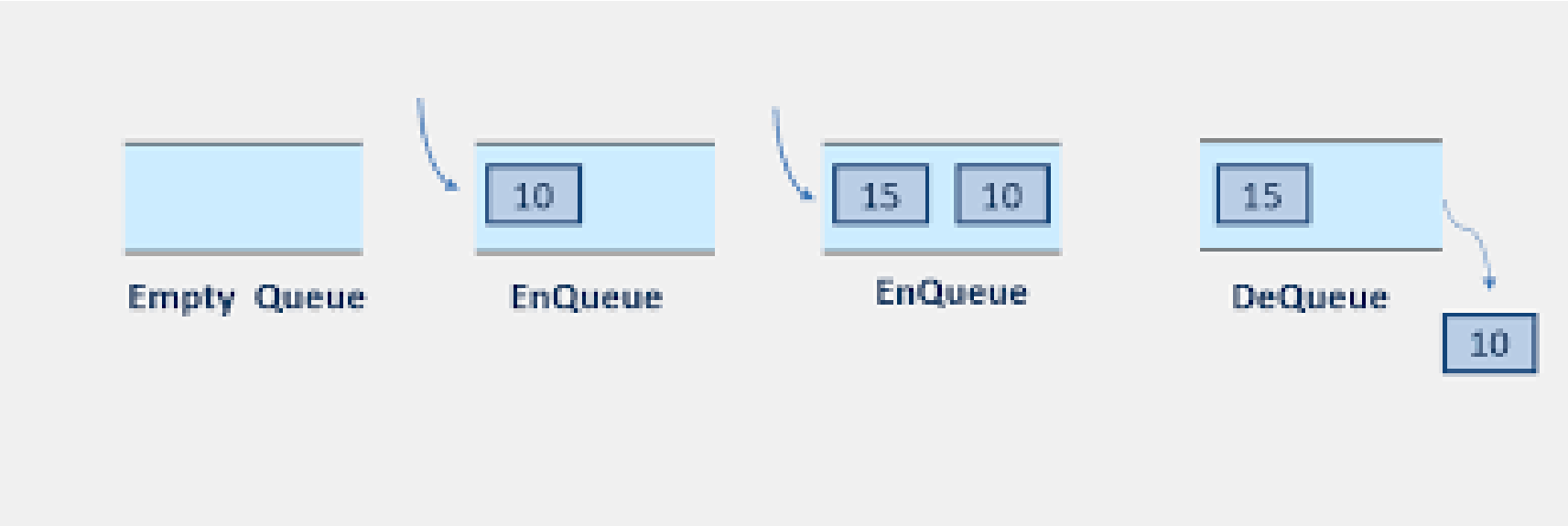
- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack using Array

Today We Will See...

- Queue using Array

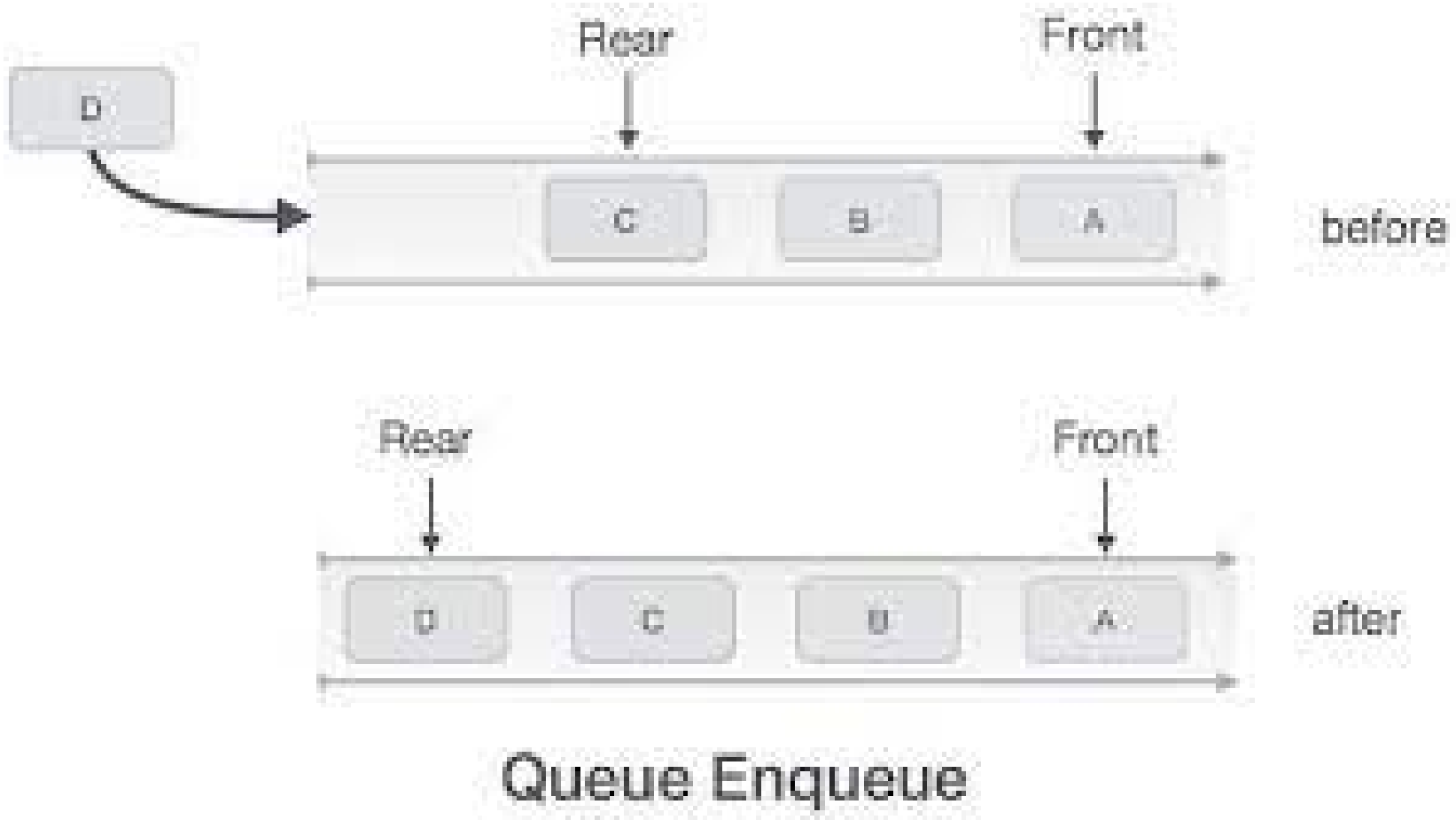
Queue Example

Queue operations



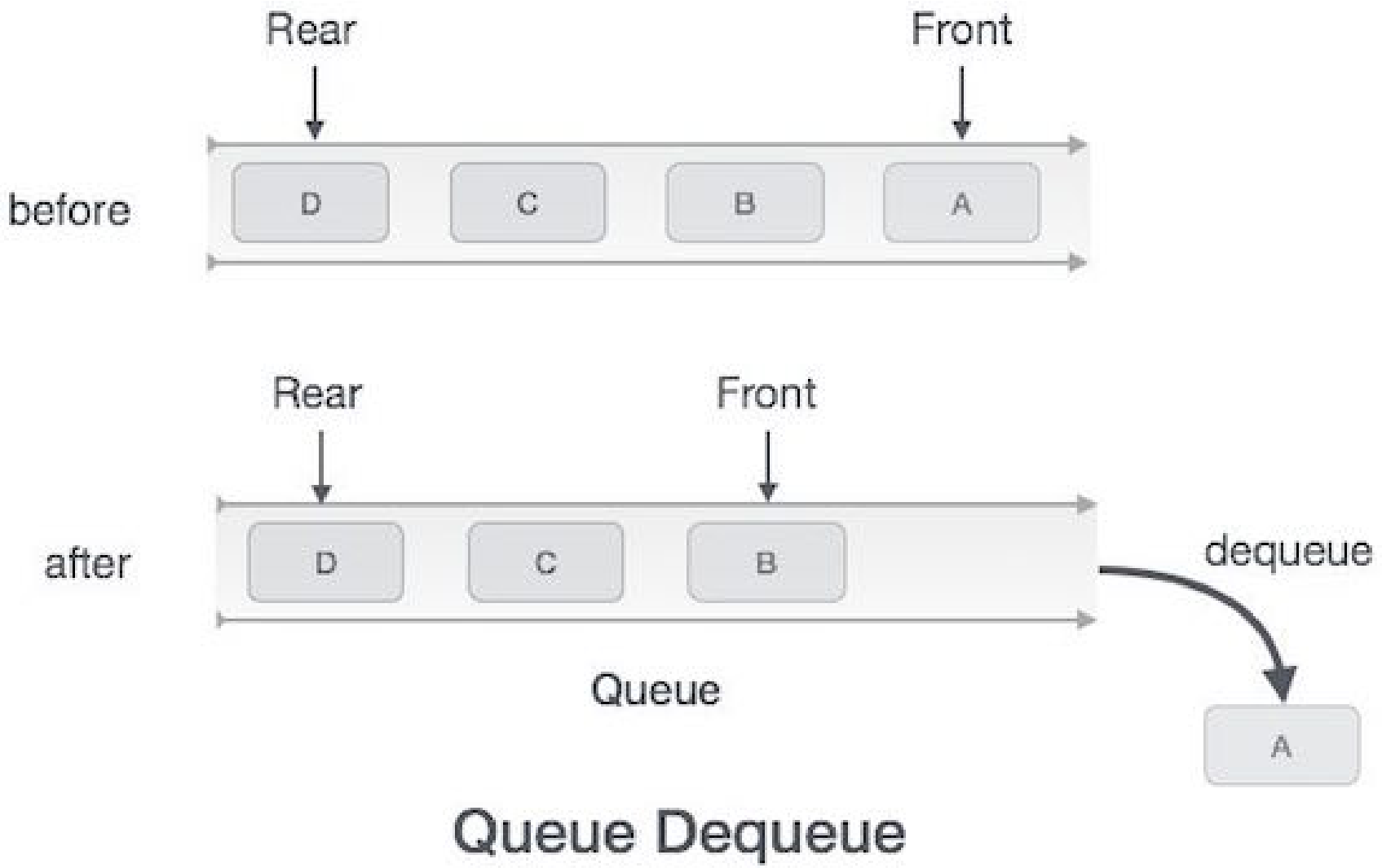
- Queue is a FIFO or LILO
- Insertion in Queue - EnQueue
- Deletion in Queue - DeQueue

EnQueue



Queue Enqueue

DeQueue



Queue Dequeue

Queue Overflow and Underflow

- Queue Overflow happens when we try to EnQueue an item to a Queue that is Full.
- Queue Underflow happens when we try to DeQueue an item from a Queue that is empty.

isQEmpty() & isQFull()

- front is used to point to the item at the start of the Queue
- rear is used to point to the item at the end of the Queue
- front $= -1$ & rear $= -1$, indicates that the Queue is Empty
- rear $= n - 1$, indicates that Queue of size n is Full

```
int isQEmpty( )
{
  if(front == -1)
     return 1;
  else
     return 0;
}
```

```
int isQFull( )
{
  if(rear == n-1)
     return 1;
  else
     return 0;
}
```

EnQueue(int x)

Consider an integer Queue Q[n] to which an element x is enqueued .

```
void EnQueue(int x)
{
  if( !isFull() ){
    rear++;;
    Q[rear] = x;
  }
  if(front == -1)
    front = 0;
}
```

DeQueue( )

Consider an integer Queue Q[n] from which an element is dequeued.
Try Writing this by your own! Submit to me in Whatsapp.

- Think if we need to check for any extra condition similar to the one we did in EnQueue() !

```
int DeQueue( )
{
   if( !isQEmpty() ){
     x = Q[front];
     front++;
     return x;
   }
   if(front > rear)
     front = rear = -1;
}
```

Lecture 15

We Saw & Will See

Till Now We Saw...

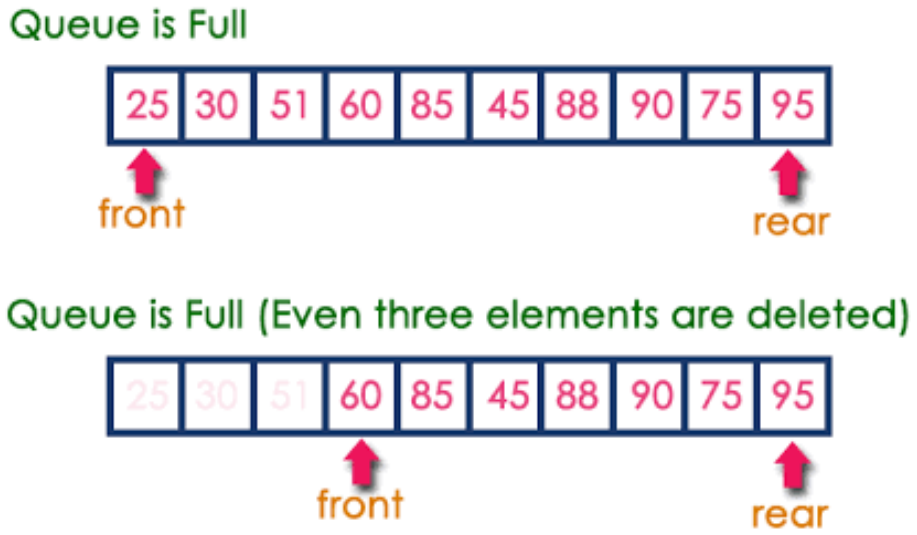
- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack using Array
  - Queue using Array

Today We Will See...

- Circular Queue using Array

Queue Example

Limitation of Queue using Array

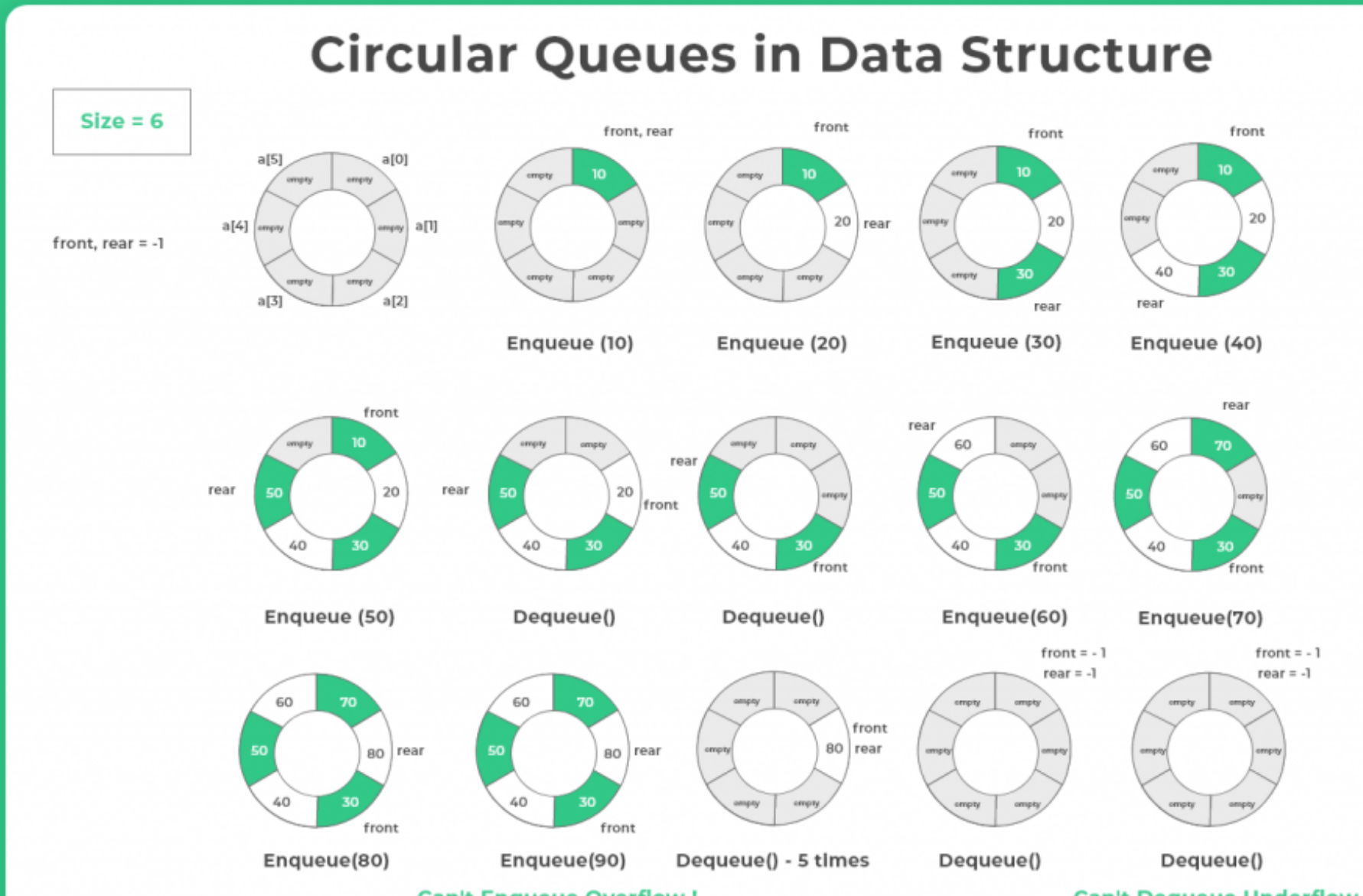


- Queue is Full when rear $= n - 1$, irrespective of where front is.
- There may still be some vacant spaces, in worst case it is $n - 1$

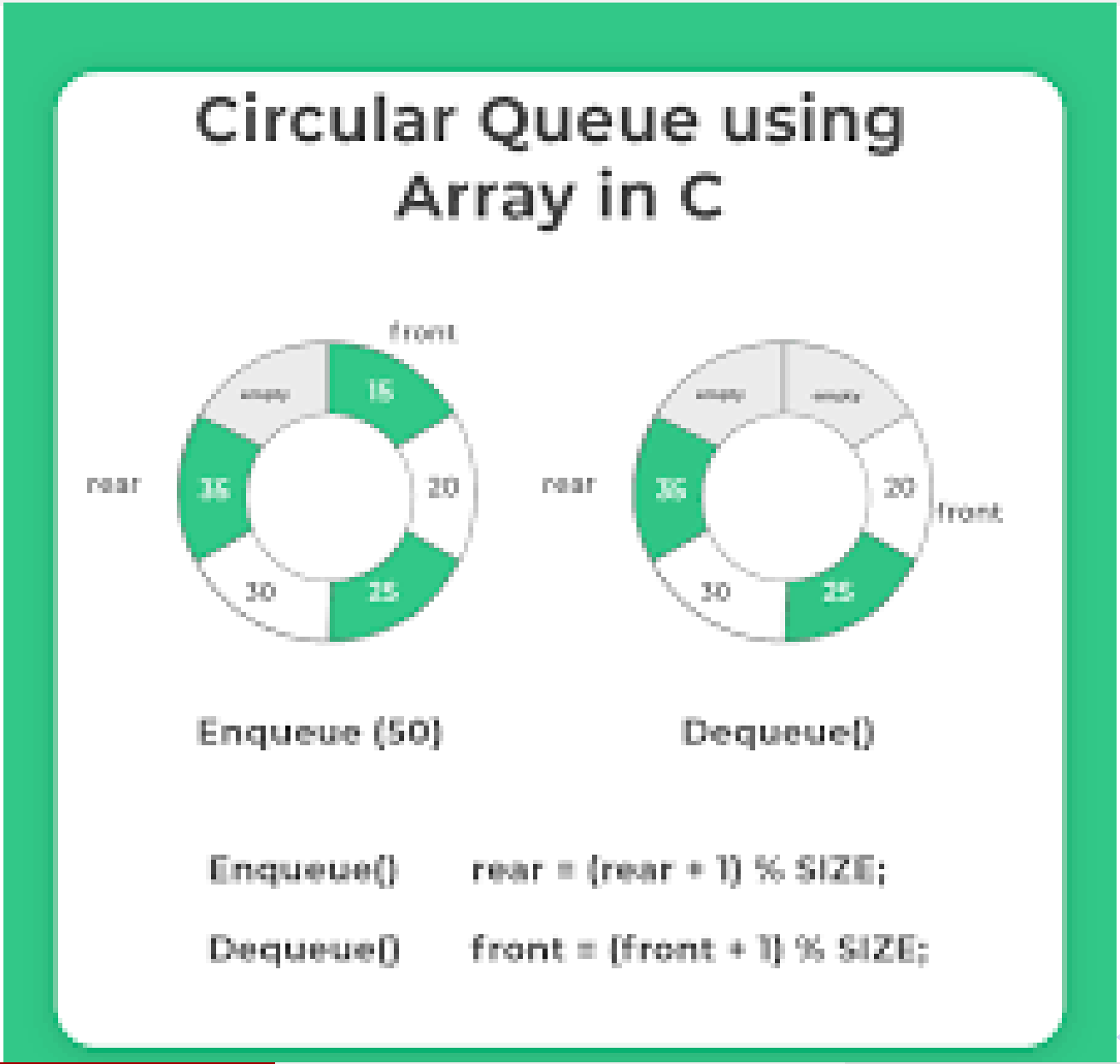
Solution:

- Fix front $= 0$, shift all elements left after a DeQueue (Not Efficient)
- Use Cirucular Queue! ✓

Circular Queue Using Array Example



EnQueue & DeQueue in Circular Queue



Queue Overflow and Underflow

- Circular Queue Overflow happens when we try to EnQueue an item to a Circular Queue that is Full, i.e with n items.
- Circular Queue Underflow happens when we try to DeQueue an item from a Circular Queue that is empty.

isCQEmpty() & isCQFull()

- front is used to point to the item at the start of the Queue
- rear is used to point to the item at the end of the Queue
- Conditions for Circular Queue being Full and Empty ??

```
int isCQFull( )
{
  if((rear+1)%n == front)
     return 1;
  else
     return 0;
}
```

```
int isCQEmpty( )
{
  if(front == -1)
     return 1;
  else
     return 0;
}
```

EnQueue(int x)

Consider an integer Queue Q[n] to which an element x is enqueued .

```
void EnQueue(int x)
{
  if( !isCQFull() ){
    rear= (rear+1)%n;;
    Q[rear] = x;
  }
  if(front == -1)
    front = 0;
}
```

DeQueue( )

Consider an integer Circular Queue Q[n] from which an element is dequeued.

```
int DeQueue( )
{
    if( !isCQEmpty() ){
      x = Q[front];
      front=(front+1)%n;
      return x;
    }
    if((rear+1)%n == front)
      front = rear = -1;
}
```

Lecture 16

We Saw & Will See

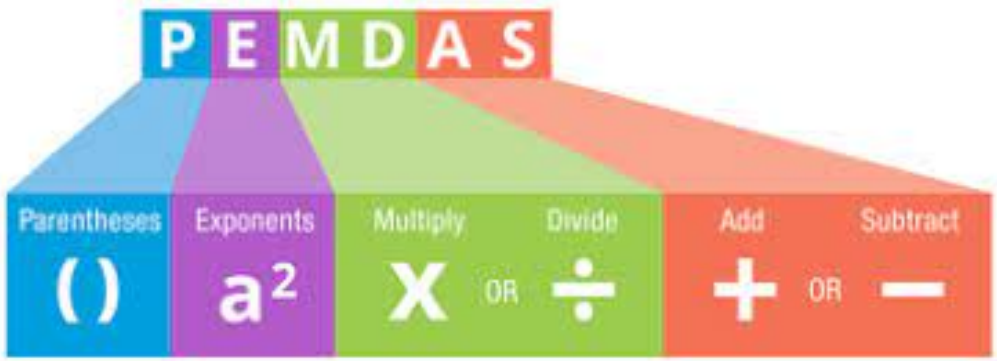
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack using Array
  - Queue, Circular Queue using Array

Today We Will See...

- Infix, Prefix and Postfix Expressions

PEMDAS rule



Order of Operations

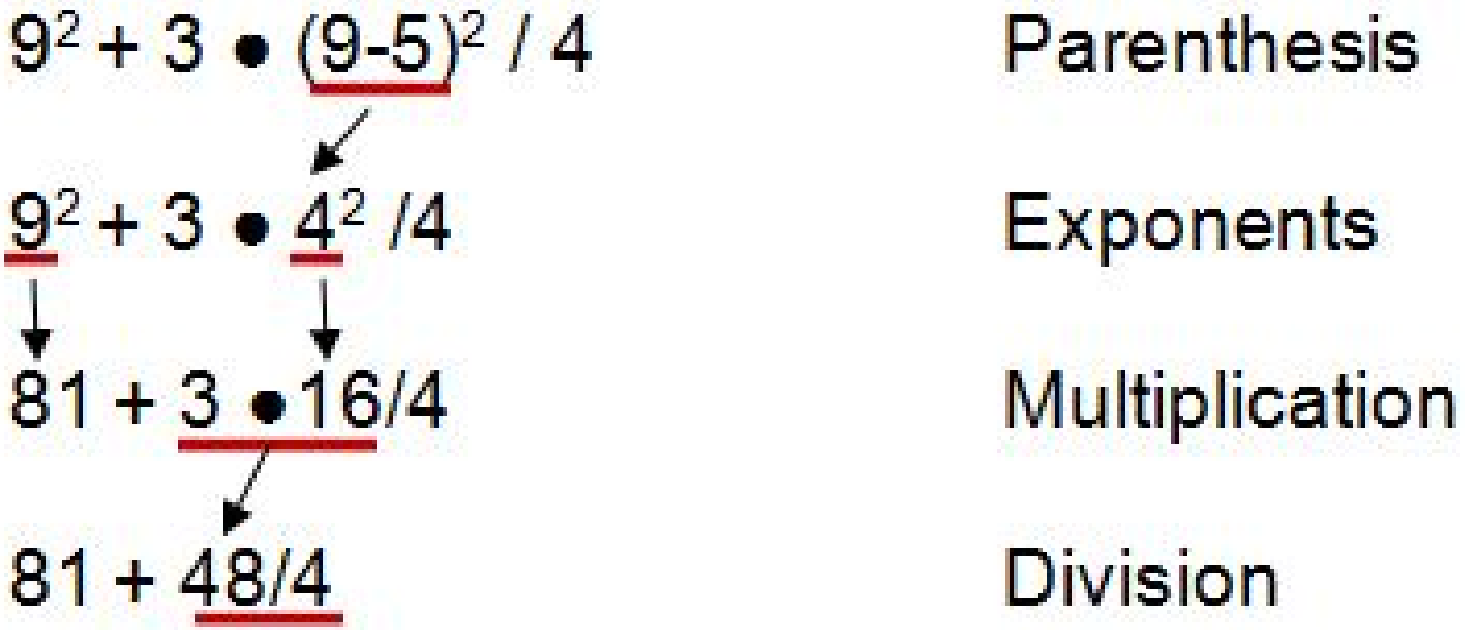
PEMDAS

P	Parenthesis, ()
E	Exponents, a^n
M D	Multiplication or Division (Left to right)
A S	Addition or Subtraction (Left to Right)

Evaluation example

Evaluate the expression:

$$9^2 + 3 \bullet (9-5)^2 / 4$$



Steps to find Prefix/Postfix expression from any given Infix Expression

- Identify the order of expression evaluation in the given Infix expression using PEMDAS rule.
- Put Parenthesis corresponding to each operation in that order. i.e. for each operator identify the two parts in the given expression that is involved and put parenthesis

Prefix Expression

- Replace all open parenthesis with its corresponding operator. Then remove all closing parenthesis as well as operators to get the Prefix expression.

Postfix Expression

- Replace all close parenthesis with its corresponding operator. Then remove all opening parenthesis as well as operators to get the Postfix expression.

Lecture 17

We Saw & Will See

Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack using Array
  - Queue, Circular Queue using Array
  - Infix, Prefix and Postfix Expressions

Today We Will See...

- Evaluation of Postfix Expression
- Infix to Postfix using Stack

Algorithm to Evaluate a Postfix Expression

```
opndstk = the empty stack
/* scan the input string reading one element */
/* at a time into symb */
while (not end of input) {

      symb = next input character;

      if (symb is an operand)
            push(opndstk, symb)

      else {
           /* symb is an operator */
            op2   = pop(opndstk);
            op1   = pop(opndstk);
 value = result of applying symb to op1 & op2
            push(opndstk, value);
      } /* end else */

} /* end while */
return (pop(opndstk));
```

Example:
Postfix Expression: 6 2 3 + - 3 8 2 / + * 2 $ 3 +

symb	opnd1	opnd2	value	opndstk
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

Infix to Postfix Algorithm

- Scan the infix expression from left to right.
- If the scanned character is an operand, output it.
- Else,
  - If the precedence operator is greater than the precedence of the operator in the stacktop(or if stack is empty or contains a '('), then Push it.
  - Else, Pop all the operators from the stack which are greater than or equal to in precedence . After doing that ,Push the scanned operator to the stack. (If you encounter parenthesis while Pop, then stop there and Push the scanned operator in the stack.)
- If the scanned character is an '(', push it to the stack.
- If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
- Repeat the process until the infix expression is fully scanned.
- Pop and output from the stack until it is not empty.

Infix to Postfix using Stack - Example1

	RPN	Stack	Input Expression
1			A+(B*(C-D)/E)
2	A		+(B*(C-D)/E)
3	A	+	(B*(C-D)/E)
4	A	(+	B*(C-D)/E)
5	AB	(+	*(C-D)/E)
6	AB	* (+	(C-D)/E)
7	AB	(* (+	C-D)/E)
8	ABC	(* (+	-D)/E)
9	ABC	(* (+	D)/E)
10	ABCD	(* (+	)/E)
11	ABCD-	* (+	/E)
12	ABCD-*	/ (+	E)
13	ABCD-*E	/ (+	)
14	ABCD-*E/	+	
15	ABCD-*E/+		

Infix to Postfix using Stack - Example2

Suppose we want to convert 2*3/(2-1)+5*3 into Postfix form,

Expression	Stack	Output
2	Empty	2
*	*	2
3	*	23
/	/	23*
(	/(	23*
2	/(	23*2
-	/(-	23*2
1	/(-	23*21
)	/	23*21-
+	+	23*21-/
5	+	23*21-/5
*	+*	23*21-/53
3	+*	23*21-/53
	Empty	23*21-/53*+

So, the Postfix Expression is 23*21-/53*+

Lecture 18

We Saw & Will See

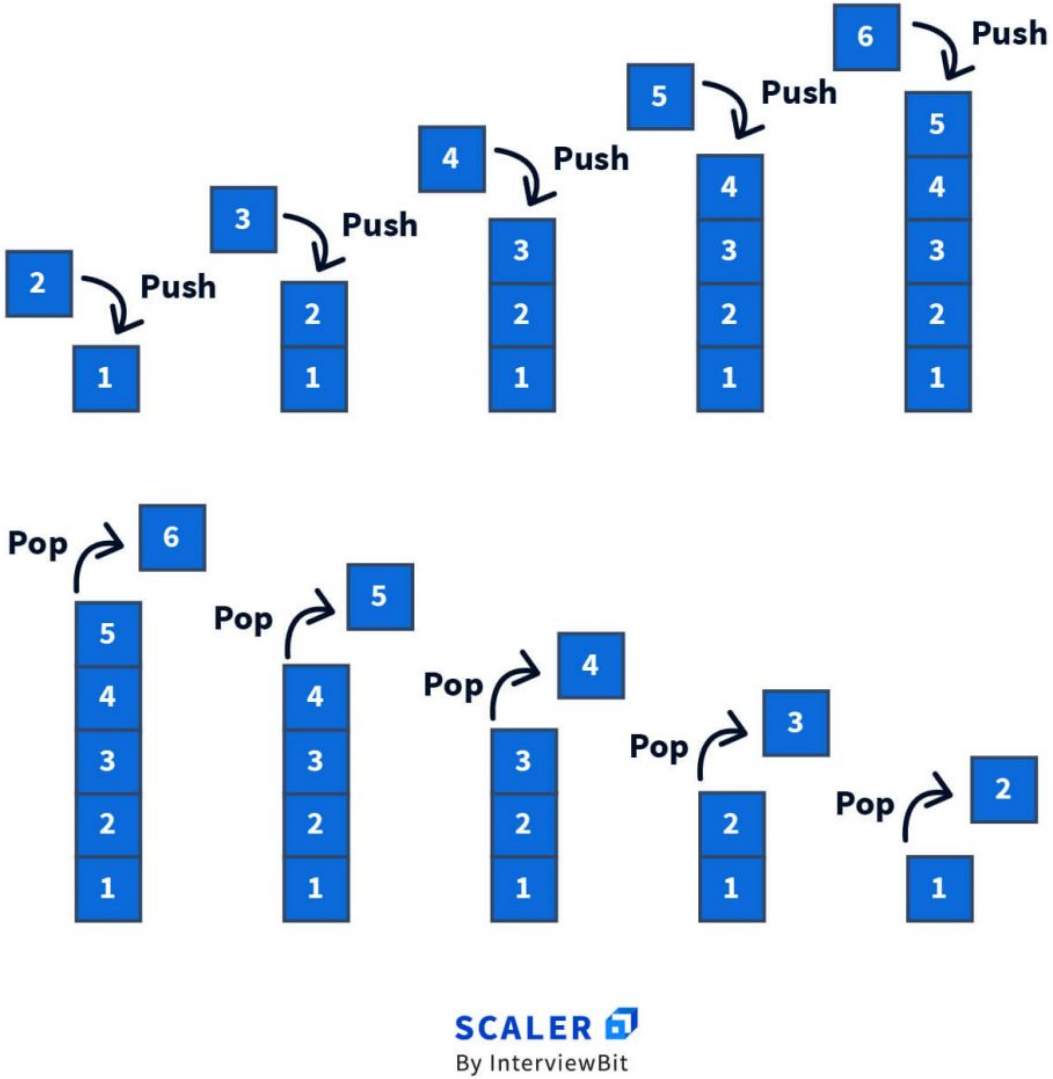
Till Now We Saw...

- Module 1 : Introduction to Data Structures
  - Concept, Classification - ADT, CDT, Linear/Non-linear DS
  - Searching - Linear and Binary Searches
  - $O(n^2)$ Sorting Algorithms - Bubble, Selection, Insertion
  - $O(n \log n)$ Sorting Algorithms- Merge Sort, Quick Sort
- Module 3 : Stacks and Queues
  - Stack using Array
  - Queue, Circular Queue using Array
  - Infix, Prefix and Postfix Expressions
  - Evaluation of Postfix Expression
  - Infix to Postfix using Stack

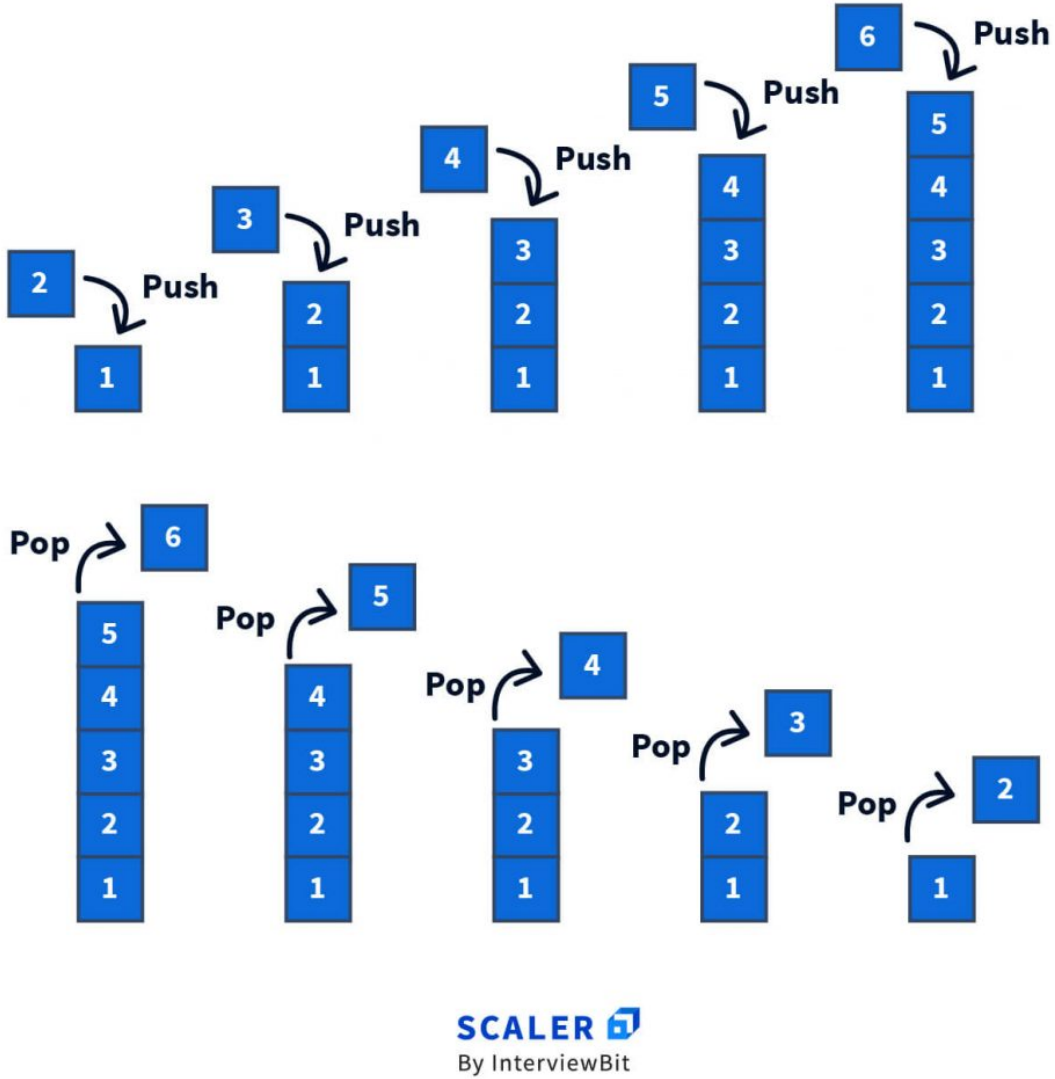
Today We Will See...

- Stack Applications - Reversal, Parenthesis Matching
- More on Queues - Double Ended Queue, Priority Queue

Reversal using Stack

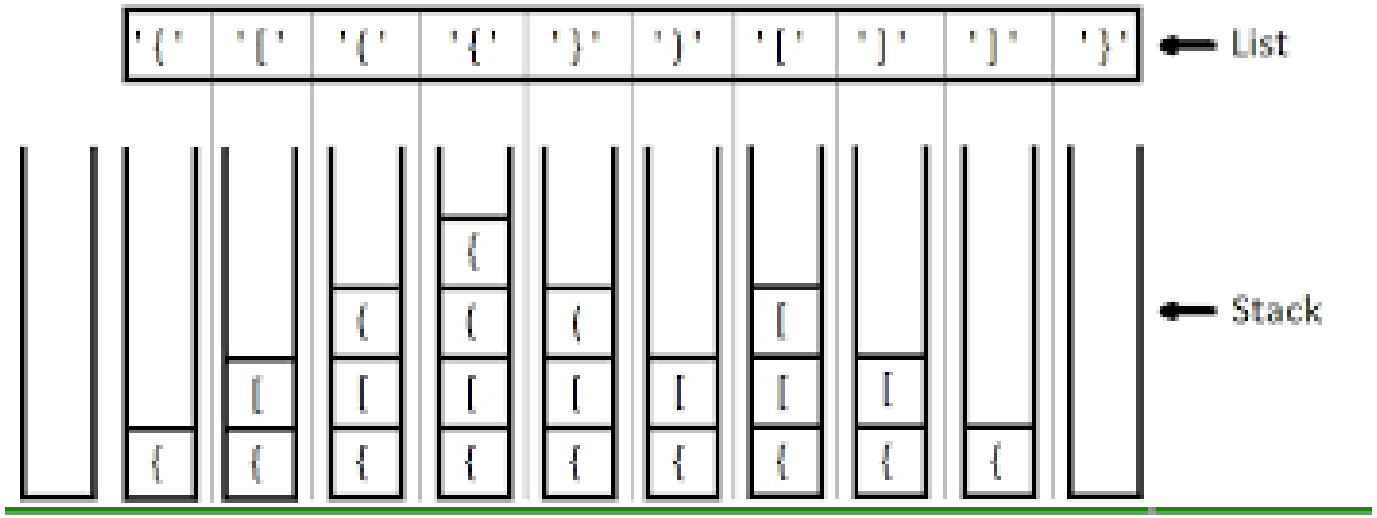


Reversal using Stack

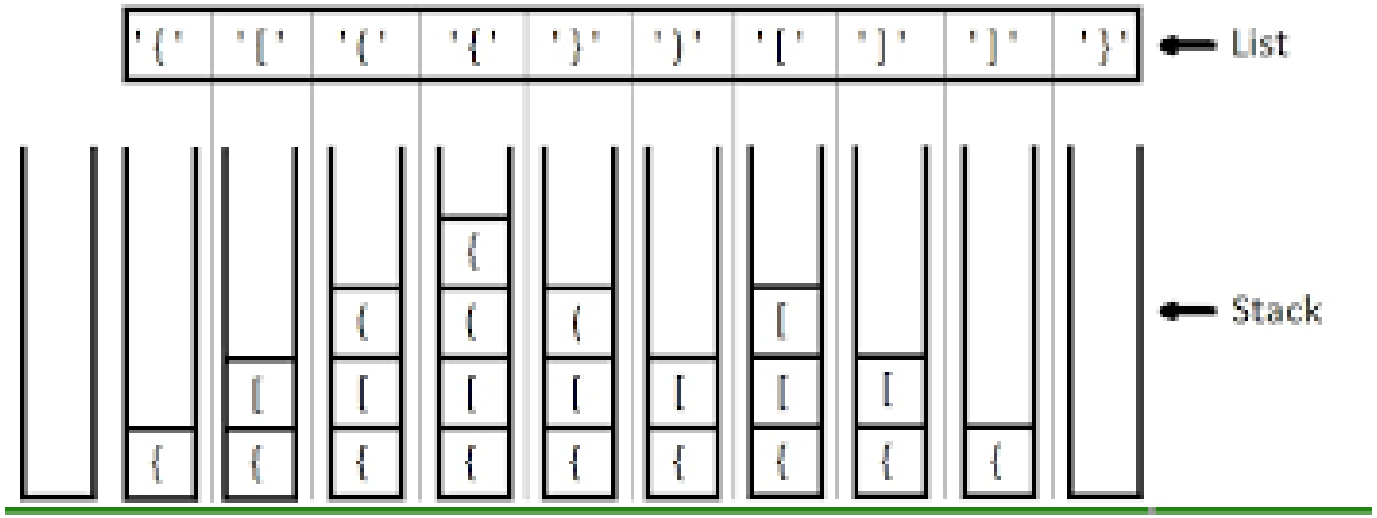


- Push all characters of string from left to right to the Stack
- Pop all the items from the Stack

Parenthesis Matching using Stack

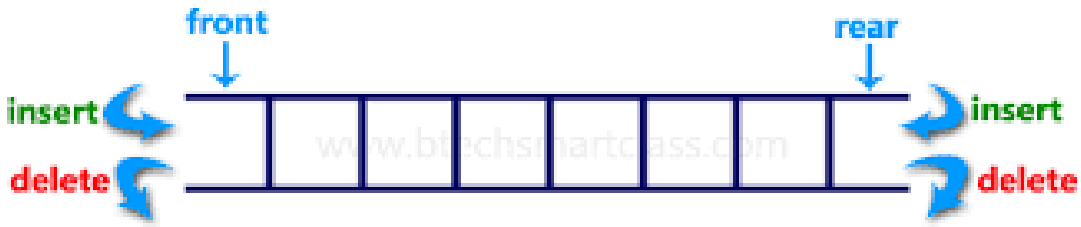


Parenthesis Matching using Stack



- Push all the opening brackets in the stack. Whenever you hit a closing bracket, search if the top of the stack is the opening bracket of the same nature.
- If so, Pop the stack and continue the iteration
- If not, then brackets are not well-ordered
- in the end if the stack is empty, it means all brackets are well-formed

Double Ended Queue



- Both insert and delete at both ends
- Variants : Input Restricted, Output Restricted

Priority Queue

- Every item has a priority associated with it.
- An element with high priority is dequeued before an element with low priority.
- If two elements have the same priority, they are served according to their order in the queue.

Priority queue can be implemented using the following data structures:

- Arrays
- Linked list
- Heap data structure
- Binary search tree

Is it possible to implement Queue using Stack?? Clue: Stacks!