# Tutorial 5

Stacks and Queues

# Method Invocation Mechanism : The call stack

https://www.youtube.com/watch?v=Q2sFmqvpBe0

# Balanced Parenthesis : Is the following pseudocode correct?

```
1.  declare a character stack
2.  while ( more input is available)
    {
3.     read a character
4.     if ( the character is a '(' )
5.        push it on the stack
```

```
6.    else if ( the character is a ')' and the stack
is not empty )
7.              pop a character off the stack
8.        else
9.              print "unbalanced" and exit
10.    }
11. print "balanced"
```

https://www.geeksforgeeks.org/data-structure-gq/stack-gq/

Which of these unbalanced sequences does the pseudocode code (given in previous slide) wrongly outputs as balanced?

1. ((())
2. ())(()
3. (()()))
4. (()))()

# Which of these unbalanced sequences does the pseudocode code (given in previous slide) wrongly outputs as balanced?

1. ((()) :
2. ())(()
3. (()()))
4. (()))()

At the end of while loop, we must check whether the stack is empty or not. For input ((()), the stack doesn't remain empty after the loop

# Parenthesis Checking

1. Declare a character stack S.
2. Now traverse the expression string exp.
   a. If the current character is a starting bracket (‘(‘ or ‘{‘ or ‘[‘) then push it to stack.
   b. If the current character is a closing bracket (‘)’ or ‘}’ or ‘]’) then pop from stack and if the popped character is the matching starting bracket then fine else parenthesis are not balanced.
3. After complete traversal, if there is some starting bracket left in stack then “not balanced”

Time complexity : O(n)

https://www.geeksforgeeks.org/check-for-balanced-parentheses-in-an-expression/

# Check for balanced parenthesis using stack

1. [()]{}{[()()]()}

2. [(])

# Check for balanced parenthesis using stack

1. `[()]{}{[()()]()}`: True

2. `[(])`

# Check for balanced parenthesis using stack

1. [ ( ) ] { } { [ ( ) ( ) ] ( ) }

2. [ ( ] ) : False

# Evaluate the following postfix expressions using stack

(a) 1 2 3 + *

(b) 9 3 4 * 8 + 4 / +

(c) 8 2 3 ^ / 2 3 * + 5 1 * - 8 +

(d) 5 2 * 3 3 2 + * +

(e) 12 25 3 * 180 6 2 / / 12 * - 7 * +

# Infix to postfix conversion using stack

1. 1 + ( 2 * 3 - ( 4 / 5 + 6 ) * 7 ) * 8

2. ( 1 * 2 + 3 * 4) + 5

Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

```
void fun(int n)
{
    IntQueue q = new IntQueue();
    q.enqueue(0);
    q.enqueue(1);
    for (int i = 0; i < n; i++)
    {
        int a = q.dequeue();
        int b = q.dequeue();
        q.enqueue(b);
        q.enqueue(a + b);
        ptint(a);
    }
}
```

https://www.geeksforgeeks.org/data-structures-queue-question-11/

Consider the following pseudo code. Assume that IntQueue is an integer queue. What does the function fun do?

```
void fun(int n)
{
    IntQueue q = new IntQueue();
    q.enqueue(0);
    q.enqueue(1);

    for (int i = 0; i < n; i++)
    {
        int a = q.dequeue();
        int b = q.dequeue();
        q.enqueue(b);
        q.enqueue(a + b);
        ptint(a);
    }
}
```

The function prints first n Fibonacci Numbers. Note that 0 and 1 are initially there in q. In every iteration of loop sum of the two queue items is enqueued and the front item is dequeued.

Suppose a circular queue of capacity (n – 1) elements is implemented with an array of n elements. Assume that the insertion and deletion operation are carried out using REAR and FRONT as array index variables, respectively. Initially, REAR = FRONT = 0. The conditions to detect queue full and queue empty are:

(A) Full: (REAR+1) mod n == FRONT, empty: REAR == FRONT
(B) Full: (REAR+1) mod n == FRONT, empty: (FRONT+1) mod n == REAR
(C) Full: REAR == FRONT, empty: (REAR+1) mod n == FRONT
(D) Full: (FRONT+1) mod n == REAR, empty: REAR == FRONT

https://www.geeksforgeeks.org/data-structures-queue-question-7/

# Answer: (A)

Suppose we start filling the queue.

Let the maxQueueSize ( Capacity of the Queue) is 4.
So the size of the array which is used to implement
this circular queue is 5, which is n.

In the begining when the queue is empty, FRONT and REAR
point to 0 index in the array.

REAR represents insertion at the REAR index.
FRONT represents deletion from the FRONT index.

https://www.geeksforgeeks.org/data-structures-queue-question-7/

enqueue("a"); REAR = (REAR+1)%5; ( FRONT = 0, REAR = 1)
enqueue("b"); REAR = (REAR+1)%5; ( FRONT = 0, REAR = 2)
enqueue("c"); REAR = (REAR+1)%5; ( FRONT = 0, REAR = 3)
enqueue("d"); REAR = (REAR+1)%5; ( FRONT = 0, REAR = 4)

Now the queue size is 4 which is equal to the maxQueueSize.
Hence overflow condition is reached.

Now, we can check for the conditions

.When Queue Full :

( REAR+1)%n = (4+1)%5 = 0

FRONT is also 0.

Hence ( REAR + 1 ) %n is equal to FRONT.

https://www.geeksforgeeks.org/data-structures-queue-question-7/

When Queue Empty :

REAR was equal to FRONT when empty ( because in the starting before filling the queue FRONT = REAR = 0 )

Hence Option A is correct.

https://www.geeksforgeeks.org/data-structures-queue-question-7/

# Homework questions for practice

1. Implement stack using queues
2. Implement queue using stacks