**Concept Part:**

**Question1:**

Convert infixexpression A + ( B \* C - ( D / ( E + F ) \* G ) ) / H **into its equivalent postfix form using stack.**

**Note: Clearly show the status of stacks at the given checkpoints containing the elements to be pushed and/or popped from the stacks. When you pop any element from the stack, don’t erase it, just delete it as  ~~(~~.**

A + ( B \* C - ( D / ( E + F ) \* G ) ) / H

**After convert to postfix:**

A B C \* D E F + / G \* - H / +

Stack status:

2

3

1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | |  | | **/** | | **\*** | | **+** | |
|  |
|  |
|  |
|  |
|  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| |  | | --- | |  | |  | | **(** | | **/** | | **(** | | **-** | |
|  |
|  |
|  |
|  |
|  |

|  |
| --- |
|  |
|  |
|  |
| **\*** |
| **(** |
| **+** |

**Algorithm Write-up**: **(Queues) (20 Pts)**

Suppose you have two queues Q1 and Q2 (implemented using linked list) containing unordered data. Write an algorithm/pseudocode which merges the data of both of these queues in the third queue Q3 in **ascending order**. After the algorithm is over, queues Q1 and Q2 will be empty.

**Note: Don’t use any sorting method to arrange the elements of queues. Also don’t use recursion. Suppose enque(), deque(), front() etc. methods are already available.**

**Sample Input:**

Front1

5

14

3

null

10

**Q1:**

25

null

Front2

10

30

15

6

**Q2:**

null

Front3

**Q3:**

**Output:**

null

Front1

**Q1:**

null

Front2

**Q2:**

10

null

30

25

15

14

10

6

5

3

Front3

**Q3:**

**Solution:**

**int el1 = dequeueSmallest(q1);**

**int el2 = dequeueSmallest(q2);**

**boolean compare=true;**

**while (compare==true) {**

**if (el1 < el2) {**

**merged.enqueue(el1);**

**if (q1.isEmpty()) {**

**while (!q2.isEmpty()) {**

**merged.enqueue(dequeueSmallest(q2 ) );**

**}**

**compare=false;**

**}**

**// take another element from q1**

**el1 = dequeueSmallest(q1);**

**} else {**

**merged.enqueue(el2);**

**if (q2.isEmpty()) {**

**// add remaining q1 elements**

**while (!q1.isEmpty()) {**

**merged.enqueue(dequeueSmallest(q1));**

**}**

**compare=false;**

**}**

**// take another element from q2**

**el2 = dequeueSmallest(q2);**

**}**

**}**

**Methods:**

**int FindSmallest(Queue q){**

smallest 🡨 q.front().data

int counter=0

while(counter < q.length()){

if(q.front().data < lowest){

smallest 🡨 q.front()

}

q.enqueue(q.dequeue());

counter ++;

}

return smallest;

**}**

**int dequeueSmallest(Queue q){**

int smallest= **FindSmallest(Queue q);**

int counter=0

while( i < q.length()){

if(Q.front().data == smallest){

dequeueElement 🡨 q.dequeue()

counter++;

}

Q.enqueue(q.dequeue());

counter++;

}

return dequeueElement;

**}**