

Assignment #3 – LISTS, STACKS, QUEUES & HEAPS

Evaluation on:

Tuesday, 26 February, 2013 (for A Batch)

Thursday, 28 February, 2013 (for B Batch)

Note:

This assignment is based on lists, stacks, queues & heaps. Implement ALL questions. Your program MUST take input from text files, which is formatted as specified with each question.

1. STACK

- a) Implement a stack using an array.
- b) Implement a stack using a linked list.

Your program must support the following functions:

- **push(stk, element)** – puts the data specified by element on top of the stack specified by stk.
- **pop(stk)** – removes and returns the topmost element of the stack specified by stk. Return null (or some special value), if the stack is empty.
- **peek(stk)** – returns the topmost element of the stack specified by stk, without actually removing the element from the stack. Return null (or some special value), if the stack is empty.
- **show(stk)** – displays all the data present in the stack specified by stk.

Input - Output Format

The input consists of multiple lines, each one containing either one or two integers.

The first integer in the line can be 0, 1, 2, 3 or 4, and each one has its own meaning:

- The integer 0 means stop the program.
- The integer 1 means push the next integer from the input on the stack. In this case, the next integer (≥ 0) is given on the same line as the 1, separated by a space.
- The integer 2 means pop and output the topmost element of the stack. Output "EMPTY", if the stack was originally empty.
- The integer 3 means peek and output the topmost element of the stack. Output "EMPTY", if the stack was originally empty.
- The integer 4 means show all elements in the stack. In this case, output all elements of the stack on a single line, separated by space, starting with the top most element. Output "EMPTY", if the stack was originally empty.

<u>Sample Input</u>	<u>Sample Output</u>
1 45	
1 65	
1 74	
1 25	
1 98	
3	98
3	98
2	98
3	25
1 17	
4	17 25 74 65 45
2	17
2	25
2	74
2	65
2	45
2	EMPTY
3	EMPTY
4	EMPTY
0	

Note:

The above input and output is for the linked list implementation of the stack. For the array implementation, the very first line of input contains an integer c , $0 < c < 100$, which is the capacity of the stack. In this case, the push operation must output "OVERFLOW" when an element is being tried to push into an already full stack. Other input and output format remains the same.

2. QUEUE

- a) Implement a queue using an array.
- b) Implement a queue using a linked list.

Your program must support the following functions:

- **enqueue(q , $element$)** – puts the data specified by $element$ at the rear end of the queue specified by q .
- **dequeue(q)** – removes and returns the element at the front of the queue specified by q . Return null (or some special value), if the queue is empty.
- **peek(q)** – returns the element at the front of the queue specified by q , without actually removing the element from the queue. Return null (or some special value), if the queue is empty.
- **show(q)** – displays all the data present in the queue.

Input - Output Format

The input consists of multiple lines, each one containing either one or two integers.

The first integer in the line can be 0, 1, 2, 3 or 4, and each one has its own meaning:

- The integer 0 means stop the program.
- The integer 1 means enqueue the next integer from the input into the queue. In this case, the next integer (≥ 0) is given on the same line as the 1, separated by a space.
- The integer 2 means dequeue and output the element from the front of the queue. Output "EMPTY", if the queue was originally empty.
- The integer 3 means peek and output the element from the front of the queue. Output "EMPTY", if the queue was originally empty.
- The integer 4 means show all elements in the queue. In this case, output all elements of the stack on a single line, separated by space, starting with the element at the front. Output "EMPTY", if the queue was originally empty.

Sample Input

1 45

1 65

1 74

1 25

1 98

3

3

2

3

1 17

4

2

2

2

2

2

2

3

4

0

Sample Output

45

45

45

65

65 74 25 98 17

65

74

25

98

17

EMPTY

EMPTY

EMPTY

Note:

The above input and output is for the linked list implementation of the queue. For the array implementation, the very first line of input contains an integer c , $0 < c < 100$, which is the capacity of the queue. In this case, the enqueue operation must

output “OVERFLOW” when an element is being tried to enqueue into an already full queue. Other input and output format remains the same.

3. EXPRESSION

- Given an infix expression, convert it into its equivalent postfix expression.
- Given an infix expression, convert it into its equivalent prefix expression.

The program must support the following:

➤ Binary operators:

^ – Exponentiation (Highest precedence)

/, * – Division, Multiplication

+, - – Addition, Subtraction (Lowest precedence)

For operators having the same precedence, use left-to-right associativity.

➤ Operands:

The operands are all variables, which is represented by a single lowercase English character. (a – z)

➤ Parentheses

Input – Output Format

The only line of the input contains a valid infix expression. (There will not be any space anywhere in the expression)

Output a single line containing the corresponding postfix (or prefix, depending on the program) notation.

<u>Sample Input</u>	<u>Sample Output (for postfix)</u>	<u>Sample Output (for prefix)</u>
a+b*c	abc*+	+a*bc
i+j-k	ij+k-	-+ijk
i-j+k	ij-k+	+ -ijk
m	m	m
(m)	m	m
((x+(y)))	xy+	+xy
(p+q)^r	pq+r^	^+pqr

4. PRIORITYQ

- Implement a priority queue using a heap.

Your program must support the following functions:

- **insert(pq, element)** – adds the data specified by element into the priority queue specified by pq. The priority of the element will have been already set.
- **remove(pq)** – removes and returns the element with the highest priority from the priority queue specified by pq. Return null (or some special value), if the priority queue is empty.

- **peek(pq)** – returns the element with the highest priority from the priority queue specified by pq, without actually removing the element from the priority queue. Return null (or some special value), if the priority queue is empty.
- **increase_priority(pq, element, newpr)** – change the priority of the data specified by element, in the priority queue specified by pq, by assigning it the new priority, newpr. It is guaranteed that newpr will be higher (in the sense of priority) than the original priority of the data specified by element.

Input - Output Format

The input consists of multiple lines, each one containing either one or three integers.

The first integer in the line can be 0, 1, 2, 3 or 4, and each one has its own meaning:

- The integer 0 means stop the program.
- The integer 1 means insert the next integer from the input into the priority queue. In this case, two more integers follow the 1, each separated by space. The first integer (≥ 0) is the data to be inserted. The second integer (≥ 1) is the priority of this data item (1 being the highest priority). Assign the priority to the data, and then insert this data into the priority queue.
- The integer 2 means remove and output the element with the highest priority from the priority queue (Output the priority of the element in parenthesis, separated by a space). Output “EMPTY”, if the priority queue was originally empty.
- The integer 3 means peek and output the element with the highest priority from the priority queue (Output the priority of the element in parenthesis, separated by a space). Output “EMPTY”, if the priority queue was originally empty.
- The integer 4 means increase the priority of a datum in the priority queue. In this case, two more integers follow the 4, each separated by space. The first integer (≥ 0) is the data (which will be present in the priority queue). The second integer (≥ 1) is the new increased priority. After this operation, the priority of this data should be the one specified in this input.

Sample Input

```
1 45 5
1 65 9
1 74 7
1 25 2
1 98 3
3
4 74 1
```

Sample Output

```
25 (2)
```

3	74 (1)
2	74 (1)
3	25 (2)
1 17 6	
2	25 (2)
2	98 (3)
3	45 (5)
4 65 1	
3	65 (1)
2	65 (1)
2	45 (5)
2	17 (6)
2	EMPTY
3	EMPTY
0	

For more details on priority queues and heaps, refer *Introduction to Algorithms* by Thomas H Cormen *et al.*

5. LIST

There is a text file containing some integers. The first line contains a positive integer n . Then, exactly n lines follow, each containing an integer.

Read this file and create a linked list (singly or doubly, based on part (a) or part (b) of the question) of all these n elements. On this list, do the following:

- Sort the singly linked list in non-decreasing order.
- Sort the doubly linked list in non-decreasing order.

Output the sorted numbers one per line.

You can use any two different sorting algorithms (one each for the two parts of the question) for this problem.

Please note that for sorting, instead of moving the data around, you must only change the links (pointers) to point to the correct element.

--- Best Wishes ---

Appeal to Students – Standard of Conduct

Violations of academic integrity will be severely penalized.

Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work **MUST BE** an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or evaluation and will be reported to the department council for record keeping and for permission to assign F grade in the course. The department policy on academic integrity can be found at: <http://cse.nitc.ac.in/sites/default/files/Academic-Integrity.pdf>.