National Institute of Technology Calicut
Department of Computer Science and Engineering
Third Semester B. Tech.(CSE)
CS2092D Programming Laboratory
Assignment #6
Submission deadline (on or before): 08/10/2023, 11:55 PM

Naming Conventions for Submission

- Submit a single ZIP (.zip) file (do not submit in any other archived formats like .rar, .tar, .gz). The name of this file must be

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>.zip`

  (Example: *ASSG6_BxxyyyyCS_LAXMAN.zip*). DO NOT add any other files (like temporary files, input files, etc.) except your source code, into the zip archive.

- The source codes must be named as

  `ASSG<NUMBER>_<ROLLNO>_<FIRST-NAME>_<PROGRAM-NUMBER>.c`

  (For example: *ASSG6_BxxyyyyCS_LAXMAN_1.c*). If you do not conform to the above naming conventions, your submission might not be recognized by our automated tools, and hence will lead to a score of 0 marks for the submission. So, make sure that you follow the naming conventions.

Standard of Conduct

- Violation 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_new.pdf.

General Instructions

- Programs should be written in C language.
- Check your programs with sufficiently large values of inputs with in the range as specified in the question.
- Global and/or static variables should not be used in your program.

QUESTIONS

1. Write a menu driven program to implement a STACK S of at most n elements using an array $A[0..n-1]$. STACK is to be declared as a struct with an attribute *top* and an array $A[0..n-1]$ as members. The attribute *top* indexes the most recently inserted element in the stack. The stack consists of elements $A[0..S.top]$ where *A[0]* is the element at the bottom of the stack and *A[S.top]* is the element at the top. Your program must contain the following functions: (In function prototypes, S denotes a stack, and k denotes a character. All operations should run in *O(1)* time.)

   - MAIN() - repeatedly reads an input character from the menu list through the terminal and executes menu-driven operations accordingly. The menu list is ['*i*', '*d*', '*p*' or '*t*']. The program terminates when '*t*' is entered.
   - STACK-EMPTY(S) - checks whether the Stack S is empty or not. It returns -1 when S is empty, else returns 1.

- STACK-FULL(S) - checks whether the Stack S is full or not. It returns -1 when S is full, else returns 1.
- PUSH(S, k) - inserts k to the top of S and check STACK-FULL() inside this function.
- POP(S) - prints and deletes the most recently inserted element from S (check STACK-EMPTY() inside this function).
- PEEK(S) - returns the top element of S.

Input Format:

- The first line of the input contains an integer $n \in [0, 10^5]$, the capacity of stack S.
- Next line inputs a character from the menu list ['*i*', '*d*', '*p*', or '*t*'] followed by either nothing or one upper case alphabet.
- Input '*i*' should be followed by a character `C` $\in [A..Z]$ that is separated by a space. In this operation, the character `C` is inserted to S by calling the function PUSH(S, `C`).
- Input '*d*' performs the `Pop` operation on S by calling the function POP(S).
- Input '*p*' perform the `Peek` operation on S by calling the function PEEK(S).
- Input '*t*' is to 'terminate' the execution of the program.

Output Format:

- The output (if any) for each input should be printed on a separate line.
- For option '*i*': Print `-1` if S is full.
- For option '*p*'and '*d*': Print `-1` if S is empty.

Sample Input :

```
5
i D
i H
d
i A
i D
i Z
i K
i X
d
d
d
p
i Y
d
d
d
p
d
t
```

Sample Output:

```
H
-1
K
Z
D
A
```

Y
A
D
-1
-1

2. Given an array *arr[ ]* of size N (i.e. It has N elements, where $1 \leq N \leq 1000$). For each element in the array, your task is to find the next greater element appearing in the array (The order of the array should not be modified).
For each element in the array, the next greater element is the nearest greater element appearing on the right side. If no next greater is available for an element, then the next greater element for that element is set to -1. Write a program that prints the next greater element for every element in the input array. Implement it using stack.
NOTE: next greater for the last element in the array is always -1.

Input Format:

- First line of your input is the size of the array, *N*.
- The Second line of your input should be the array of N elements.

Output Format:

- Your output should be a new array which stores the corresponding next greater values for each element in the array *arr[ ]*.

Sample Input:
4
1 3 2 4

Sample Output:
3 4 4 -1

3. Write a menu-driven program to implement a QUEUE Q using an array $A[0..n-1]$. Queue is to be declared as a struct with three attributes, namely *head*, *tail* and array $A[0..n-1]$. The attribute *Q.head* indexes its head and *Q.tail* indexes the next location at which a newly arriving element will be inserted into the queue. The elements in the queue reside in locations *Q.head, Q.head+1,...., Q.tail-1*, where the location *0* immediately follows location *n-1* in a circular order.Your program must contain the following functions: (in function prototypes, Q denotes a Queue and S denotes a string of characters. All operations should run in *O(1)* time.)

- MAIN() - repeatedly reads an input character from the menu list through the terminal and execute menu-driven operations accordingly. The menu list is ['*i*', '*d*', '*f*', '*e*' '*t*'] from the terminal and calls the appropriate function until character '*t*'is entered.
- QUEUEFULL(Q) - checks whether the Queue is full or not. It returns -1 when Q is full, else returns 1.
- QUEUEEMPTY(Q) - checks whether the Queue is empty or not. It returns -1 when Q is empty, else returns 1.
- ENQUEUE(Q, S) - inserts the element S to the tail of Q and also check QUEUEFULL() inside this function.
- DEQUEUE(Q) - prints and deletes the element from the head of Q and also check QUEUEEMPTY() inside this function.

Input format:

- The first line of the input contains an integer $n \in [0, 10^5]$, the size of the array A.

- The next lines should begin with an input character from the menu list ['*i*', '*d*', '*f*' , '*e*', '*t*'] followed by either nothing or one string of characters.
- Input '*i*' should be followed by an alphanumeric string $S \in [A - Z, a - z, 0 - 9]$ of at most 20 characters separated by a space. In this operation, string S is inserted to the tail of Q by calling the function ENQUEUE(Q, S).
- Input '*d*' performs the `Dequeue` operation on Q by calling the function DEQUEUE(Q).
- Input '*f*' checks whether the Queue is full or not by calling the function QUEUEFULL(Q).
- Input '*e*' is to check whether the Queue is empty or not by calling the function QUEUEEMPTY(Q).
- Input '*t*' 'terminate' the program.

Output Format:

- The output (if any) of each input should be printed on a separate line.
- For option '*i*'and '*f*': Print -1 if Q is full.
- For option '*d*'and '*e*': Print -1 if Q is empty.
- For option '*f*': Print 1 if Q is not full.
- For option '*e*', Print 1 if Q is not empty.

Sample Input

```
5
i Abc20
i Xyz38
d
d
d
e
i Nit40
e
i IIT35
i IIM42
f
i Fix33
i Calc27
f
d
t
```

Sample Output

```
Abc20
Xyz38
-1
-1
1
1
-1
Nit40
```

4. Given an integer K and a queue of N integers (where $1 \leq K \leq N$). Your task is to reverse the order of the first K elements of the queue, leaving the rest of the (N-K) elements in the queue in the same original order. Design and implement this using a queue.
Your program should be implemented using the following standard operations:

- ENQUEUE(X): add an element X to rear of queue.
- DEQUEUE(): remove an element from front of queue.
- SIZE(): returns number of elements in queue.
- FRONT(): returns the element present in the front of the queue.

Input Format:

- The first line of your input contains 2 integers separated by a space, the first integer inputs the size N of the queue, and the second integer inputs position K.
- The second line inputs the elements of the queue.

Output Format:

- A single array containing the modified queue with the first K elements in reversed order.

Sample Input:

```
5 3
1 2 3 4 5
```

Sample Output:

```
3 2 1 4 5
```