Practice Arena

Practice problems aimed to improve your coding skills.

- PRACTICE-02_SCAN-PRINT
- PRACTICE-03_TYPES
- LAB-PRAC-02_SCAN-PRINT
- LAB-PRAC-01
- PRACTICE-04_COND
- BONUS-PRAC-02
- LAB-PRAC-03_TYPES
- PRACTICE-05_COND-LOOPS
- LAB-PRAC-04_COND
- LAB-PRAC-05_CONDLOOPS
- PRACTICE-07_LOOPS-ARR
- LAB-PRAC-06_LOOPS
- LAB-PRAC-07_LOOPS-ARR
- LABEXAM-PRAC-01_MIDSEM
- PRACTICE-09_PTR-MAT
- LAB-PRAC-08_ARR-STR
- PRACTICE-10_MAT-FUN
- LAB-PRAC-09_PTR-MAT
- LAB-PRAC-10_MAT-FUN
  - Stack
  - The Prutor Editor
  - Finding your identity
  - Queue
  - The Prutor Editor Part II
  - Only Ones
  - Graphs
  - How Mr C actually does Math
  - The Hidden Positives and Negatives
  - How Prutor Manages Memory
  - Message in the Matrix
  - The Hidden Key
- PRACTICE-11_FUN-PTR
- LAB-PRAC-11_FUN-PTR
- LAB-PRAC-12_FUN-STRUC
- LABEXAM-PRAC-02_ENDSEM
- LAB-PRAC-13_STRUC-NUM
- LAB-PRAC-14_SORT-MISC

Queue

LAB-PRAC-10_MAT-FUN

Queue [20 marks]

Problem Statement
The queue is a data structure that is incredibly useful in computer science. The Prutor server uses a queue to handle things when several students request compilation or evaluation at the same time. All these requests are put in a queue and the requests that came first are handled first and so on. Queues are also essential to the working of the internet as well as operating systems. However, the basic premise of a queue is extremely simple and all of you encounter queues all the time.

When we are in line to get a movie ticket, the line is also called a *queue*. Note that the person who came first to the ticket counter first is at the front of the queue and when the ticket master issues the next ticket, it is to the person at the front of the queue. People who come later are at the back of the queue and are serviced later. Queues are often called FIFO (first-in-first-out) structures since the first element (e.g. person in a line) that enters the queue is also the first to leave the queue.

The computer science queues work exactly the same way. The queues we will implement in this question will have three operations. The queues will have a limit size of MAX

1. Enqueue x: if there are already MAX elements in the queue, print the error message "FULL" (without quotes), otherwise put the element x at the back of the queue.
2. Dequeue: If the queue is empty, print the error message "EMPTY" (without quotes), otherwise remove the element at the front of the queue and print that element.
3. Check: If the queue is empty, print "EMPTY" (without quotes), else if it is full, print "FULL" (without quotes), else print "NOT EMPTY" (without quotes)

In the first line of the input, you will be give the value of MAX as a strictly positive integer. In the next several lines, you will see instructions on what to do

1. E x: enqueue the integer x into the queue or print the error message. There will be a single space between the character 'E' and the integer x.
2. D: dequeue the element at the front of the queue and print it or else print the error message
3. C: perform the check operation and print the appropriate message
4. X: no more operations to perform

Thus, the list of operations will be terminated by an X. All elements enqueued will be integers.

Caution

1. Print all error messages as well as dequeued elements on separate lines.
2. If you are using getchar() to process input character by character, be careful to read in the newlines at the end of each line as well.
3. We will not penalize you for extra newlines at the end of your output. However, do not have extra newlines in the middle of your output or else have trailing spaces in any line of the output.

HINTS:

1. Use an array with MAX elements to manipulate the queue. You will have to be a bit careful about how you manipulate the array since dequeue operations will leave empty slots at one end of the array. One solution is to shift the array elements one place to the left after every dequeue operation and use a variable to keep track of how many elements are left in the queue.
2. Write functions to perform various queue operations to ensure your code looks neat and clean.

EXAMPLE:
INPUT
2
C
E 5
C
E 4
C
E 3
D
D
D
X

OUTPUT:
EMPTY
NOT EMPTY
FULL
FULL
5
4
EMPTY

Explanation

1. The queue only has a capacity of 2 elements.
2. Command 1: C - queue is empty at the moment so print EMPTY
3. Command 2: E 5 - enqueue 5. Now queue looks like [5]
4. Command 3: C - queue is neither full nor empty so print NOT EMPTY
5. Command 4: E 4 - - enqueue 4. Now stack looks like [5 4]
6. Command 5: C - queue is full so print FULL
7. Command 6: E 3 - queue is already full - cannot enqueue another element so print FULL - queue still looks like [5 4]
8. Command 7: D - dequeue front element 5 and print 5. Now queue looks like [4]
9. Command 8: D - dequeue front element 4 and print 4. Now queue is empty
10. Command 9: D - queue is already empty - cannot dequeue another element so print EMPTY - queue is still empty
11. Command 10: X - No more processing to do.

Grading Scheme:
Total marks: **[20 Points]**

There will be partial grading in this question. There are several lines in your output. Printing each line correctly, in the correct order, carries equal weightage. Each visible test case is worth 2 points and each hidden test case is worth 4 points. There are 2 visible and 4 hidden test cases.

Please remember, however, that when you press Submit/Evaluate, you will get a green bar only if all parts of your answer are correct. Thus, if your answer is only partly correct, Prutor will say that you have not passed that test case completely, but when we do autograding afterwards, you will get partial marks.

Start Solving! (/editor/practice/6200)