Introduction to Data Structures (430.217, 002)
Seoul National University

Project #1
Due : 04 Nov 2022, 23:59:59

INSTRUCTIONS

- You are not allowed to copy and paste the code of others (e.g., classmates or website). Please try to work on your own.
- It is desirable to check if your codes run on the environment we have introduced in the Linux tutorial. The version of g++ and c++ should be 7+ and 14+, respectively. If it works only on your PC, there may be deduction of the score.
- All kinds of questions are welcome. If you have problem or find out typos in the given skeleton code, please leave the question on the eTL QnA board. It is recommended to make it open to your classmates. However, we are not going to fix your bugs directly.
- Grading policy will be as follows.
  - 100 points for 100 test cases.
  - Late submission : 10% deduction per 12 hours
  - Memory leak or memory error : 30% deduction
  - Only working on your machine : 20% deduction
  - Any other non-compliance with given rules will be also deducted.

1 Queue (30 points)

You are assigned to implement a circular priority queue using an array. All you have to do is fill out the TODOs in "queue.hpp". It is okay to define additional member functions, but not allowed to add member variables of the class/struct. Note that the class is defined with generic data type using template so that it can be reused for data of any type. It is prohibited using C++ Standard Template Library (STL).

The items have the priority itself (The larger the number, the higher the priority). They are enqueued without considering its priority (time complexity of $O(1)$). When trying to dequeue the item or to peek the top, the highest priority is searched from the front (time complexity of $O(n)$). After dequeueing, the array should be aligned. When the array is full, you should allocate a new one with double capacity. The example of execution is depicted as Figure 1 and 2.

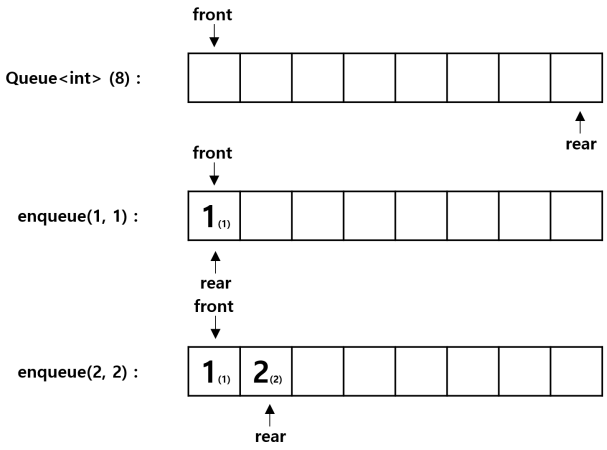

Figure 1: Initializing the queue and enqueueing items

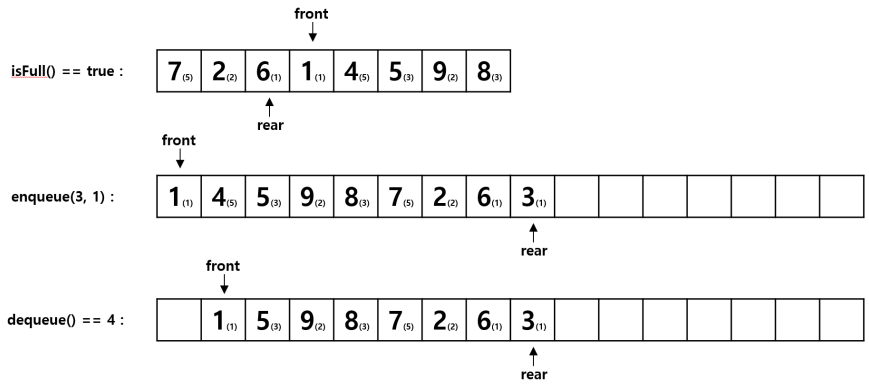

Figure 2: Enqueueing items when the array is full and dequeueing based on the priority

2 Stack (20 + 25 + 25 points)

You are assigned to implement a stack using an array. All you have to do is fill out the TODOs in "stack.hpp" and "utils.cpp". It is okay to define additional member/global functions, but not allowed to add member variables of the class. Note that the class is defined with generic data type using template so that it can be reused for data of any type. It is prohibited using C++ Standard Template Library (STL) except for the parameter of a function "checkParentheses()".

The objectives of the stack are two-fold. First is to check the parentheses matching (25 pts). Any pairs of parenthesis (a single character) such as { , } or #, # can be given to check balanced parentheses. Matching rules should be set according to the given argument. Another objective is to perform a simple calculation including +, -, *, /, (and) (25 pts). The line will be given as string without any parenthesis error, and the answer should be returned as float data type. The example of checking balanced parentheses is depicted as Figure 3 and 4. Note that you should allocate a new array with double capacity as well when the stack is full.

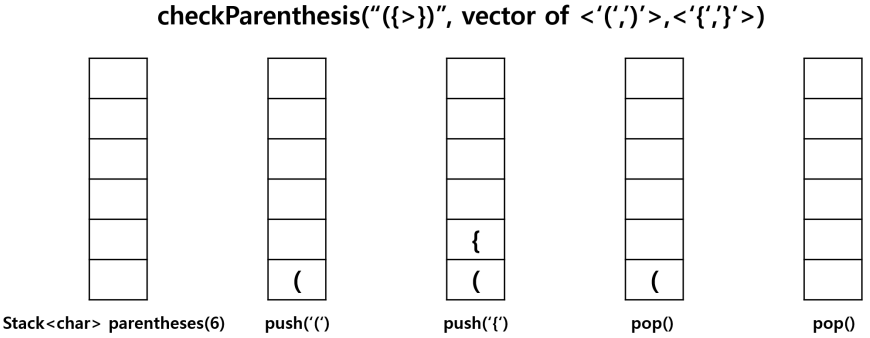

Figure 3: Checking parentheses using a stack (balanced)

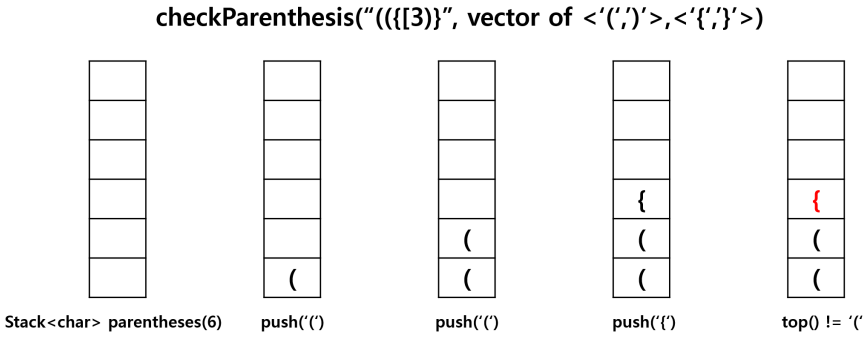

Figure 4: Checking parentheses using a stack (unbalanced)