Programming Assignment #1

Lecturer: Prof. Seung-Hwan Baek
Teaching Assistants: Chunghyun Park, Kanghee Lee, Seungjoo Shin, Dongmin You

***** PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT *****

Due date: 11:59PM October 03, 2022

Evaluation policy:

- Late submission penalty
  - 11:59PM October 03 ~ 11:59PM October 04
    - Late submission penalty (30%) will be applied to the total score.
  - After 11:59PM October 04:
    - 100% penalty is applied for that submission.
- Your code will be automatically tested using an evaluation program.
  - Each problem has the maximum score.
  - A score will be assigned based on the behavior of the program.
- We won't accept any submission via email - it will be ignored.
- Do not modify auxiliary files.
  - Such as: utils.h/cpp, evaluate.cpp, and so on.
- Compile your file(s) using repl.it and check your program before the submission.
- Please do not use C++ standard template library.
  - Such as:
    - #include <queue>
    - #include <vector>
    - #include <stack>
  - Any submission using STL library will be disregarded.

File(s) you need to submit:

- pa1.cpp (Do not change the filename!)

Any questions? Please use PLMS - Q&A board.

0. Basic instruction

Please refer to the instruction document, “DataStructure_PA_instructions.pdf”.

```
>> g++ -std=c++11 -o pa1.exe pa1.cpp utils.cpp
```

1. Asymptotic analysis (1 pts)

a. Choose the TIGHT bound of the following **`arraySum`** function.

b. `arraySum`
Input: An integer n >= 1, an array A storing n integers
Output: Sum of the given array A

```
int arraySum(int n, int* A) {
      int currSum = 0;
      for (int i = 0; i < n; i++)
         currSum += A[i];
      return currSum;
}
```

1. *O*(1)
2. *O*(n)
3. *O*(n log (n))
4. *O*(n^2)

c. Example output: If you choose *O*(1), then print 1

```
>> ./pa1.exe 1
[Task 1]
1
```

2. Asymptotic analysis (1 pts)

a. Choose the TIGHT bound of the following **primeSearch** function.

b. primeSearch
   Input: Two integers num1 & num2 where num2>num1
   Output: The total number of prime numbers between num1 and num2

```
int primeSearch(int num1, int num2) {
      int numPrimes = 0;
      int ctr = 0;
      for (int i = num1; i <= num2; i++) {
        for(int j = 2; j <= sqrt(i); j++) {
            if(i % j == 0) {
                  ctr++;
                  break;
            }
        }
        if((ctr == 0) && (i > 1))
            numPrimes++;
        ctr=0;
      }
      return numPrimes;
}
```

1. *O*(log (n))
2. *O*(n log (n))
3. *O*(n)
4. *O*(n^2)

c. Example output: If you choose *O*(log (n)), then print 1

```
>> ./pa1.exe 2
[Task 2]
1
```

3. List (4 pts)

a. Implement a function that can insert or delete an integer into the list. An user can insert or delete an element at the specific index. If the specified index is out of range of the given list, print “error”.

b. Input & Output

Input: Sequence of commands, which is one of the following,

- `(‘insert’, index)`: insert “# of elements in the current list” at the `index` in the list. `index` indicates zero-based index.
- `(‘delete’, index)`: delete an element at the `index` in the list. `index` indicates zero-based index.

Output:

- An array after insertion/deletion in a string separated with the spacebar
- “`error`” if the index is out of range

c. Example input & output

Input	Output
`[(‘insert’,0),(‘insert’,1),(‘insert’,2)]`	`0 1 2`
`[(‘insert’,0),(‘insert’,0),(‘insert’,1)]`	`1 2 0`
`[(‘insert’,0),(‘insert’,1),(‘delete’,0)]`	`1`
`[(‘insert’,0),(‘delete’,1)]`	`error`
`[(‘insert’,1)]`	`error`
`[(‘insert’,0),(‘insert’,1),(‘delete’,0), (‘insert’,1),(‘insert’,0)]`	`2 1 1`

d. Example execution

```
>> ./pa1.exe 3 “[(‘insert’,0),(‘insert’,1),(‘delete’,0),
(‘insert’,1),(‘insert’,0)]”
[Task 3]
2 1 1
```

4. Stack (3 pts)

a. Implement a function that prints the top values of the stack when “top” operation is called after the sequence of “**push**” or “**pop**” operations. If the **“top”** operation is called for an empty stack, print “-1”, If “pop” operation is called for an empty stack, print “**error**”

b. Input & Output
Input: Sequence of commands, which is one of the following,
   - `(‘push’,integer)`: push `integer` into the current stack (`integer` is always positive)
   - `(‘pop’,NULL)`: pop the top value of the current stack (this operation will print nothing)
   - `(‘top’,NULL)`: print the top value of the current stack (print ‘`-1`’ if the current stack is empty)

Output:
   - Expected printed values after processing the whole sequence, in a string separated with the spacebar
   - “`error`” if the pop operation is executed on an empty stack

c. Example Input & Output

Input	Output
`[(‘push’,5),(‘push’,3),(‘top’,NULL)]`	`3`
`[(‘push’,3),(‘top’,NULL),(‘pop’,NULL),(‘push’,5),(‘top’,NULL)]`	`3 5`
`[(‘push’,5),(‘pop’,NULL),(‘top’,NULL)]`	`-1`
`[(‘pop’,NULL)]`	`error`
`[(‘pop’,NULL),(‘push’,5),(‘top’,NULL)]`	`error`

d. Example execution

```
>> ./pa1.exe 4 “[(‘push’,3),(‘top’,NULL),(‘pop’,NULL),
(‘push’,5),(‘top’,NULL)]”
[Task 4]
3 5
```

5. Queue (3 pts)

a. Implement a function that shows the value of a queue after a sequence of arbitrary queue operations. If the queue after the operations is empty, print "**`empty`**". If "dequeue" operates on an empty queue, print "**`error`**".

b. Input & Output

Input: Sequence of commands, which is one of the following,

- `('enqueue',integer)`: enqueue `integer` into the current queue
- `('dequeue',NULL)`: dequeue from the current queue

Output:

- Values in the queue from the front to the rear, in a string separated with the spacebar
- "`empty`" if the queue is empty
- "`error`" if the "dequeue" operation is executed on an empty queue

c. Example Input & Output

Input	Output
`[('enqueue',5),('enqueue',3),('dequeue',NULL)]`	`3`
`[('enqueue',5),('enqueue',3),('dequeue',NULL),('enqueue',5)]`	`3 5`
`[('enqueue',3),('dequeue',NULL)]`	`empty`
`[('enqueue',5),('dequeue',NULL),('dequeue',NULL)]`	`error`

d. Example execution

```
>> ./pa1.exe 5 "[('enqueue',5),('enqueue',3),('dequeue',NULL)]"
[Task 5]
3
```

6. Circular Queue (3 pts)

a. Implement a function that shows the values in a circular queue with a counter. If "enqueue" is called for an already full queue or the "dequeue" operation is called for an empty queue, there should be no changes to the queue. The maximum number of elements (n) in the queue is four.

b. Input & Output
Input: Sequence of commands, which is one of the following,
- `('enqueue',integer)`: enqueue `integer` into the current queue
- `('dequeue',NULL)`: dequeue from the current queue

Output
- Values in the circular queue (mod size n = 4), from the front to the rear. String separated with the spacebar, empty memory shows NULL
- No pointer movement if dequeue applied on an empty queue or enqueue applied on a full queue

c. Example input & output

Input	Output
[('enqueue',5),('enqueue',3),('dequeue',NULL)]	3
[('enqueue',5),('enqueue',3),('dequeue',NULL),('enqueue',6)]	3 6
[('enqueue',3),('dequeue',NULL)]	
[('enqueue',5),('dequeue',NULL),('enqueue',3),('enqueue',6),('enqueue',9),('enqueue',1),('dequeue',NULL),('dequeue',NULL),('enqueue',7),('enqueue',2)]	9 1 7 2

d. Example execution

```
>> ./pa1.exe 6
"[('enqueue',5),('enqueue',3),('dequeue',NULL),('enqueue',6)]"
[Task 6]
3 6
```