**Programming Assignment #1**

Lecturer: Prof. Jaesik Park  
Teaching Assistants: ByeongHoon So, Hyunmin Lee, Junha Lee

|  |
| --- |
| *\*\*\*\* PLEASE READ THIS GRAY BOX CAREFULLY BEFORE STARTING THE ASSIGNMENT \*\*\*\**  Due date: 11:59PM April 11, 2020  Evaluation policy:   * Late submission penalty   + 11:59PM April 11 ~ 11:59PM April 12     - Late submission penalty (30%) will be applied to the total score   + After 11:59PM April 12:     - 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 * Please do not use C++ standard template library   + Such as:     - #include <queue>     - #include <vector>     - #include <stack>   + Any submission using STL library will be disregarded   Any questions?   * Please use LMS - Q&A board |

1. Basic instruction
   1. Please refer to the attached file named PA\_instructions.pdf
2. Asymptotic analysis (1 pts)  
   1. Choose the TIGHT bound of the following **arrayMax** function
   2. arrayMax  
      Input: An integer n >= 1, an array A storing n integers  
      Output: The maximum element of A

|  |
| --- |
| int arrayMax(int n, int\* A) {  int currMax = A[0];  for (int i = 1; i < n; i++)  if (currMax < A[i])  currMax = A[i];  return currMax; } |

1. *O*(1)
2. *O*(n)
3. *O*(n log (n))
4. *O*(n^2)
   1. Example output: If you choose *O*(1), then print 1

|  |
| --- |
| >> ./pa1.exe 1  [Task 1]  1 |

1. Asymptotic analysis (2 pts)
2. Choose the TIGHT bound of the following **prefixAverages** function
3. prefixAverages   
   Input: An integer n >= 1, an array X storing n real numbers  
   Output: An n-element array A of real numbers such that A[i] is the average of elements X[0], … , X[i]

|  |
| --- |
| double\* prefixAverages(int n, double\* X) {  double \*A = new double[n];  double sum;  for (int i = 0; i < n; i++) {  sum = 0;  for (int j = 0; j <= i; j++)  sum = sum + X[j];  A[i] = sum / (i+1);  }  return A; } |

1. *O*(log (n))
2. *O*(log^2 (n))
3. *O*(n log (n))
4. *O*(n^2)
5. Example output: If you choose *O*(log (n)), then print 1

|  |
| --- |
| >> ./pa1.exe 2  [Task 2]  1 |

1. List (3 pts)
2. Implement a function that can append an integer or insert “0” into the list. A user can specify the position where “0” will be inserted. If the index is out of range of the given list, print “error”
3. Input & Output

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

* (‘append’, integer): append integer at the tail of the list
* (‘insert\_at’, index): insert 0 at the index

Output:

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

1. Example input & output

|  |  |
| --- | --- |
| Input | Output |
| [(‘append’,1),(‘append’,2)] | 1 2 |
| [(‘append’,1),(‘append’,2),(‘insert\_at’,1)] | 1 0 2 |
| [(‘append’,1),(‘insert\_at’,1),(‘append’,2)] | 1 0 2 |
| [(‘append’,1),(‘insert\_at’,2),(‘append’,2)] | error |
| [(‘insert\_at’, 0),(‘append’,1)] | 0 1 |

1. Example execution

|  |
| --- |
| >> ./pa1.exe 3 “[(‘append’,1),(‘append’,2),(‘insert\_at’,1)]”  [Task 3]  1 0 2 |

1. Stack (3 pts)
2. Implement a function which shows the value in the stack from the top
3. Input & Output

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

* (‘push’, integer): push integer into the current stack

Output:

* Values in the stack from the top to the bottom, in a string separated with the spacebar

1. Example input & output

|  |  |
| --- | --- |
| Input | Output |
| [(‘push’,3)] | 3 |
| [(‘push’,5),(‘push’,7)] | 7 5 |
| [(‘push’,5),(‘push’,3),(‘push’,2)] | 2 3 5 |
| [(‘push’,5),(‘push’,5),(‘push’,5),(‘push’,5)] | 5 5 5 5 |

1. Example execution

|  |
| --- |
| >> ./pa1.exe 4 “[(‘push’,5),(‘push’,3),(‘push’,2)]”  [Task 4]  2 3 5 |

1. Stack (3 pts)
2. Implement a function that shows the value in the stack from the top after the sequence of “**push**” or “**pop**” operations. If the stack is empty then print “**empty**”, If “pop” operation from the empty stack then print “**error**”
3. Input & Output

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

* (‘push’, integer): push integer into the current stack
* (‘pop’, NULL): pop the top value of the current stack

Output:

* Values in the stack from the top to the bottom, in a string separated with the spacebar
* “empty” if the resulting stack is empty
* “error” if the pop operation is executed on an empty stack

1. Example Input & Output

|  |  |
| --- | --- |
| Input | Output |
| [(‘push’,5),(‘push’,3),(‘push’,2)] | 2 3 5 |
| [(‘push’,5),(‘pop’,NULL),(‘push’,3)] | 3 |
| [(‘push’,5),(‘pop’,NULL)] | empty |
| [(‘pop’,NULL)] | error |

1. Example execution

|  |
| --- |
| >> ./pa1.exe 5 “[(‘push’,5), (‘push’,3),(‘push’,2)]”  [Task 5]  2 3 5 |

1. Queue (3 pts)  
   1. Implement a function that shows the value of a queue after the sequence of arbitrary queue operations. If the queue after the operations is empty, print “**empty**”. If “dequeue” operates on an empty queue, print “**error**”.
   2. 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 head to the tail, in a string separated with the spacebar
* “empty” if the queue is empty
* “error” if the “dequeue” operation is executed on an empty queue
  1. 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 |

* 1. Example execution

|  |
| --- |
| >> ./pa1.exe 6 “[(‘enqueue’,5),(‘enqueue’,3),(‘dequeue’,NULL)]”  [Task 6]  3 |