Seoul National University

M1522.000900 Data Structure

Fall 2017, Kang

Homework 3: List, Stacks, and Queues (Chapter 4)

Due: October 12, 02:00 PM

Reminders

- The points of this homework add up to 100.
- Like all homeworks, this has to be done individually.
- Lead T.A.: Hyunsik Jeon (jeon185@gmail.com)
- Please type your answers in English. Illegible handwriting may get no points, at the discretion of the graders.
- If you have a question about assignments, please upload your question in eTL.
- If you want to use slipdays or consider late submission with penalties, please note that you are allowed one week to submit your assignment after the due date.

Remember that:

- Whenever you are making an assumption, please state it clearly.

Question 1

Answer whether each of the following statements is true or not. Also write the reason for your answer [25 points].

(1) In singly linked list, the time complexity of accessing the previous node is $\theta(1)$ [5 points].

(2) In doubly linked list, the time complexity of accessing the previous node is $\theta(1)$ [5 points].

(3) In general, the time complexity of insertion and deletion for array-based list is $\theta(1)$ [5 points].

(4) Linked list has an overhead of requiring additional memory space compared to array-based list [5 points].

(5) In array-based stack, the time complexity of the "PUSH" is θ(1) regardless of whether the top of the stack points to the leftmost index or the rightmost index [5 points].

Question 2

Assume a list has the following configuration (sorted in ascending order):

<|3, 7, 12, 21>

Answer the following questions. Note that '|' is the current position [20 points].

(1) Write a series of functions in the **List ADT** (**Figure 1 – page7)** to insert the element with value 10 to keep the list sorted in ascending order [10 points].

(2) If the values in the list are always sorted in ascending order, write a series of functions in the **List ADT** to extract the maximum value [5 points].

(3) If the values in the list are always sorted in ascending order, write a series of functions in the **List ADT** to extract the minimum value [5 points].

Question 3

Show the list configuration resulting from each series of list operations using the **List ADT (Figure 1 – page7)**. Assume that lists *L1* and *L2* are empty at the beginning of each series. Show where the current position is in the list. [20 points]

(1) L1.append(1);

L1.append(2);

L1.insert(3);

L1.next();

L1.insert(4); [10 points]

(2) L2.append(1);

L2.insert(2);

L2.insert(3);

L2.moveToEnd();

L2.remove(); [10 points]

Question 4

Show the change in stack *S* resulting from following functions of **Stack ADT** (**Figure 2 – page8)**:

S.push(5);

S.push(2);

S.push(10);

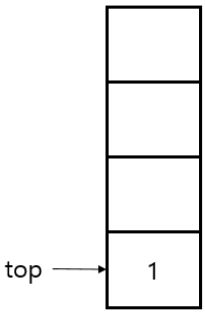
S.pop();

S.pop();

S.push(4);

Assume that stack is empty at the beginning of series, but it is not initialized after each function executed. For each function, you should show the resulting stack status like as below [15 points].

S.push(1)



Question 5

Postfix expression is in the form of A-B-O. A, B are numbers or also postfix expression, and O is an operator like '+', '-', '*', '/'. For example, a commonly used infix expression "1 + 2" is "1 2 +" in the postfix expression. Answer the following questions [20 points].

(1) Transform the following infix expression to postfix expression [5 points].

(1 + 2) * 3

(2) Assume that there is a calculator that takes a postfix expression as an input and calculate the result. When the calculator reads a digit number, it is pushed in the stack. When it reads an operator, it pops two values from the stack and calculate the operation with the two values and push the result to the stack. Show the change in stack resulting from following postfix expression (each element in expression is split by space):

[input expression]

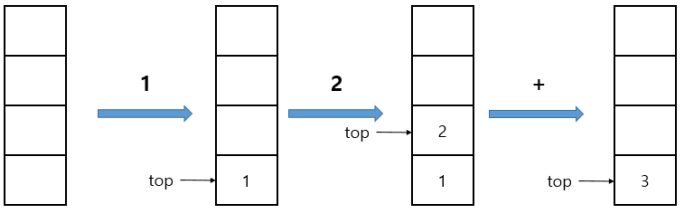
2 10 6 7 + * + 11 /

For each element of expression, you should show the resulting stack status. Below is an example [15 points].

[input expression]

1 2 +

[Your answer]



```
/** List ADT */
public interface List<E> {
  /** Remove all contents from the list, so it is once again
      empty. Client is responsible for reclaiming storage
      used by the list elements. */
  public void clear();

  /** Insert an element at the current location. The client
      must ensure that the list's capacity is not exceeded.
      @param item The element to be inserted. */
  public void insert(E item);

  /** Append an element at the end of the list. The client
      must ensure that the list's capacity is not exceeded.
      @param item The element to be appended. */
  public void append(E item);

  /** Remove and return the current element.
      @return The element that was removed. */
  public E remove();

  /** Set the current position to the start of the list */
  public void moveToStart();

  /** Set the current position to the end of the list */
  public void moveToEnd();

  /** Move the current position one step left. No change
      if already at beginning. */
  public void prev();

  /** Move the current position one step right. No change
      if already at end. */
  public void next();

  /** @return The number of elements in the list. */
  public int length();

  /** @return The position of the current element. */
  public int currPos();

  /** Set current position.
      @param pos The position to make current. */
  public void moveToPos(int pos);

  /** @return The current element. */
  public E getValue();
}
```

Figure 1. The ADT of List.

```
/** Stack ADT */
public interface Stack<E> {

  /** Reinitialize the stack.  The user is responsible for
      reclaiming the storage used by the stack elements. */
  public void clear();

  /** Push an element onto the top of the stack.
      @param it The element being pushed onto the stack. */
  public void push(E it);

  /** Remove and return the element at the top of the stack.
      @return The element at the top of the stack. */
  public E pop();

  /** @return A copy of the top element. */
  public E topValue();

  /** @return The number of elements in the stack. */
  public int length();
};
```

Figure 2. The ADT of Stack.