GIT Department of Computer Engineering

CSE 222/505 – Spring 2020

Homework #4 Report

Türker Tercan

171044032

1- Convert the infix expressions given below to prefix and postfix, then evaluate them. Show your work step by step.

i) A + ((B – C * D) / E) + F – G / H

ii)! (A && ! ((B < C) | | (C > D))) | | (C < E)

Priority of operators: ‘*’ = ‘/’ > ‘+’ = ‘-‘

Boolean operators: ‘!’ > ‘&&’ = ‘>’ = ‘<’ = ‘>=’ = ‘<=’ > ‘||’

Algorithm for converting infix to postfix:

```
1.  Initialize postfix to an empty StringBuilder
2.  Initialize the operator stack to an empty stack
3.  while there are more tokens in the infix string
4.      Get the next token
5.      if the next token is an operand
6.          Append it to postfix
7.      else if the next token is an operator
8.          if operator stack is empty
9.              Push the current operator onto the stack
10.         else
11.             if precedence of current operator is greater than the topmost
    operator of the stack
12.                 Push current operator onto the stack
13.             else if precedence of current operator is less than or equal to the
    topmost operator of the stack
14.                 Pop the operators from stack until find a low precedence
    operator than current operator and never pop out ‘(‘or ‘)’
15.                 Append these operators to postfix
16.             else if current operator is ‘(‘
17.                 Push current operator onto the stack
18.             else if current operator is ‘)’
19.                 Pop out operators from stack until we find an ‘(‘
20.                 Append these operators to postfix
21. Pop remaining operators off the stack and append them to postfix
```

Convert infix to postfix:

1) A + ((B – C * D) / E) + F – G / H

Token	operatorStack	postfix
A		A
+	+	A
(	+(	A
(	+((	A
B	+((	A B
-	+((-	A B
C	+((-	A B C
*	+((-*	A B C
D	+((-*	A B C D
)	+(	A B C D * -
/	+(/	A B C D * -
E	+(/	A B C D * - E
)	+	A B C D * - E /
+	+	A B C D * - E / +
F	+	A B C D * - E / + F
-	-	A B C D * - E / + F +
G	-	A B C D * - E / + F + G
/	-/	A B C D * - E / + F + G
H	-/	A B C D * - E / + F + G H
		A B C D * - E / +F + G H / -

Result is: A B C - D * E / + F + G H / -

ii)! (A && ! ((B < C) | | (C > D))) | | (C < E)

Token	operatorStack	Postfix
!	!	
(	!(	
A	!(	A
&&	!(&&	A
!	!(&&!	A
(	!(&&!(	A
(	!(&&!((	A
B	!(&&!((	A B
<	!(&&!((<	A B

C	!(&&!((<	A B C
)	!(&&!(	A B C <
||	!(&&!(||	A B C <
(	!(&&!(||(	A B C <
C	!(&&!(||(	A B C < C
>	!(&&!(||(>	A B C < C
D	!(&&!(||(>	A B C < C D
)	!(&&!(||	A B C < C D >
)	!(&&!	A B C < C D > ||
)	!	A B C < C D > || ! &&
||	||	A B C < C D > || ! && !
(	||(	A B C < C D > || ! && !
C	||(	A B C < C D > || ! && ! C
<	||(<	A B C < C D > || ! && ! C
E	||(<	A B C < C D > || ! && ! C E
)	||	A B C < C D > || ! && ! C E <
		A B C < C D > || ! && ! C E < ||

Result: A B C < C D > || ! && ! C E < ||

Evaluating Postfix Expressions

i) A B C D * - E / + F + G H / -

Let A = 20, B = 100, C = 2, D = 10, E = 5, F = 15, G = 150, H = 30

Token	Stack	Operator
20	20	
100	20 100	
2	20 100 2	
10	20 100 2 10	
*	20 100 20	2 * 10
-	20 80	100 - 20
5	20 80 5	
/	20 16	80 / 5
+	36	20 + 16
F	36 15	
+	51	36 + 15
150	51 150	
30	51 150 30	

/	51 5	150 / 30
-	46	51 - 5

Result is 46

ii) A B C < C D > | | ! && ! C E < | |

Let A = 1, B = 1, C = 0, D = 0, E = 1

Token	Stack	Operator
1	1	
1	1 1	
0	1 1 0	
<	1 0	1 < 0
0	1 0 0	
0	1 0 0 0	
>	1 0 0	0 > 0
| |	1 0	0 | | 0
!	1 1	!0
&&	1	1 && 1
!	0	!1
0	0 0	
1	0 0 1	
<	0 1	0 < 1
| |	1	0 | | 1

Result is 1

Algorithm for converting infix to prefix:

1. Reverses given infix string. Each ‘(‘ becomes ‘)’ and ‘)’ becomes ‘(‘
2. Does postfix algorithm and gets an string
3. Reverse the postfix string.

i) A + ((B – C * D) / E) + F – G / H

Reversed string: H / G - F + (E / (D * C – B)) + A

Token	OperatorStack	Postfix
H		H
/	/	H
G	/	H G
-	-	H G /
F	-	H G / F
+	+	H G / F -
(	+(	H G / F -
E	+(	H G / F – E
/	+(/	H G / F – E
(	+(/(	H G / F – E
D	+(/(	H G / F – E D
*	+(/(*	H G / F – E D
C	+(/(*	H G / F – E D C
-	+(/(-	H G / F – E D C *
B	+(/(-	H G / F – E D C * B
)	+(/	H G / F – E D C * B -
)	+	H G / F – E D C * B - /
+	+	H G / F – E D C * B - / +
A	+	H G / F – E D C * B - / + A
		H G / F – E D C * B - / + A +

Postfix Result: H G / F – E D C * B - / + A +

Reverse postfix result: + A + / - B * C D E – F / H G

ii) ! (A && ! ((B < C) | | (C > D))) | | (C < E)

Reversed string: (E > C) | | (((D < C) | | (C > B))! && A)!

Token	OperatorStack	PostFix
(	(	
E	(	E
>	(>	E
C	(>	E C
)		E C >
| |	| |	E C >
(	| |(	E C >
(	| |((	E C >
(	| |(((	E C >
D	| |(((	E C > D
<	| |(((<	E C > D
C	| |(((<	E C > D C
)	| |((	E C > D C <
| |	| |((| |	E C > D C <
(	| |((| |(	E C > D C <
C	| |((| |(	E C > D C < C
>	| |((| |(>	E C > D C < C
B	| |((| |(>	E C > D C < C B
)	| |((| |	E C > D C < C B >
)	| |(	E C > D C < C B > | |
!	| |(!	E C > D C < C B > | |
&&	| |(&&	E C > D C < C B > | | !
A	| |(&&	E C > D C < C B > | | ! A
)	| |	E C > D C < C B > | | ! A &&
!	| |!	E C > D C < C B > | | ! A &&
		E C > D C < C B > | | ! A && ! | |

Postfix Result: E B > D C < C B > | | ! A && ! | |

Reverse it to get prefix expression:

Result: | | ! && A ! | | < B C > D C < C E

Evaluating Prefix Expressions

i) + A + / - B * C D E – F / H G

Let A = 20, B = 100, C = 2, D = 10, E = 5, F = 15, G = 150, H = 30

Reverse the expression

H G / F – E D C * B - / + A +

Token	Stack	Operator
150	30	
30	30 150	
/	5	150 / 30
15	5 15	
-	10	15 - 5
5	10 5	
10	10 5 10	
2	10 5 10 2	
*	10 5 20	2 * 10
100	10 5 20 100	
-	10 5 80	100 - 20
/	10 16	80 / 5
+	26	10 + 16
20	26 20	
+	46	26 + 20

Result is 46 and it is equal to previous evaluation ii)| | ! && A ! | | < B C > D C < C E

Let A = 1, B = 1, C = 0, D = 0, E = 1

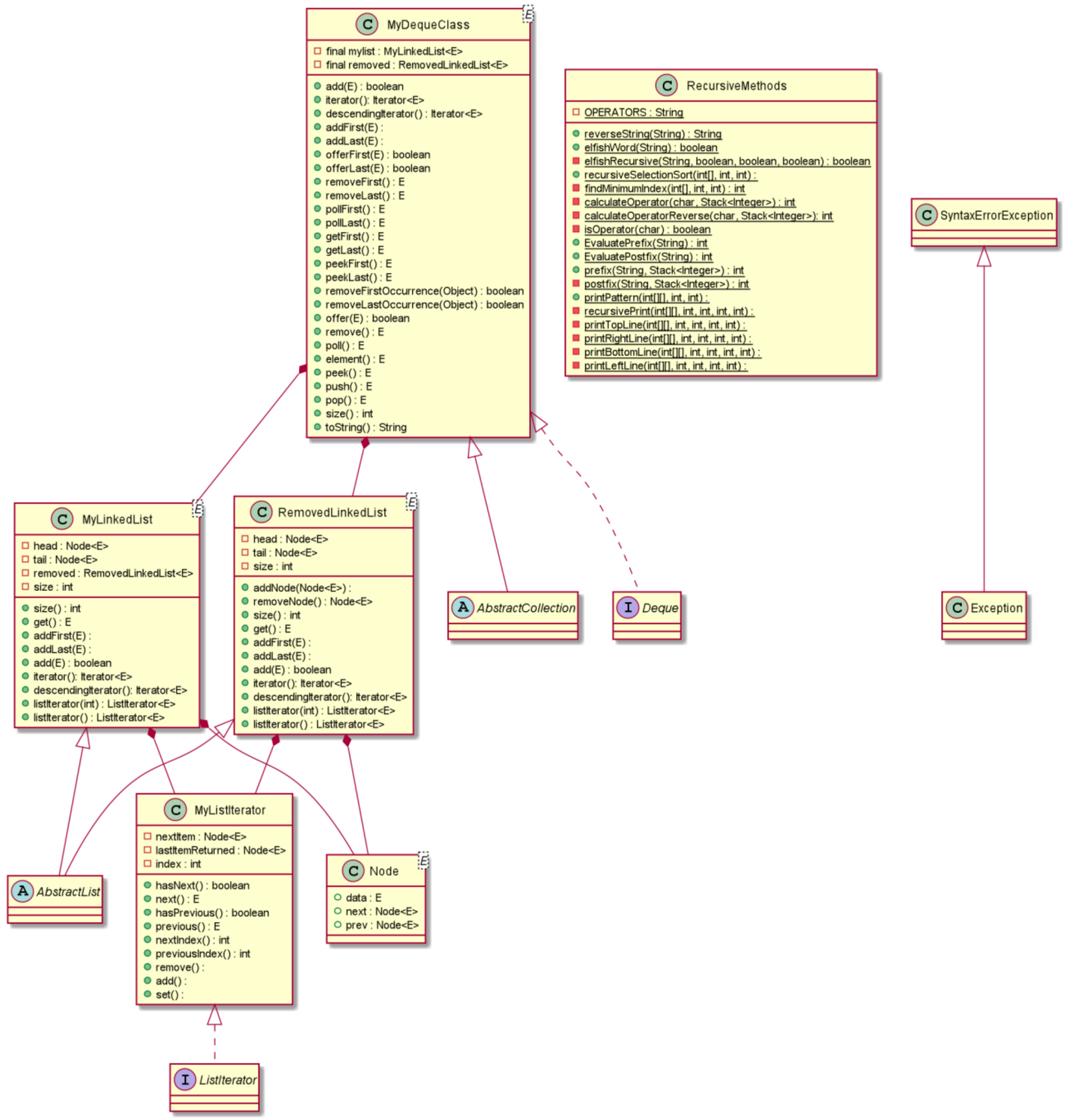
Reverse the expression

E C > C D < C B > | | ! A && ! | |

Token	Stack	Operator
1	1	
0	1 0	
>	1	1 > 0
0	1 0	
0	1 0 0	
<	1 0	0 < 0
0	1 0 0	
1	1 0 0 1	
>	1 0 0	0 > 1
| |	1 0	0 | | 0
!	1 1	! 0
1	1 1 1	
&&	1 1	1 && 1
!	1 0	! 1
| |	1	1 | | 0

Result is 1. It is equal to previous evaluation.

Class Diagrams


MyDequeClass
E
final mylist : MyLinkedList<E>
final removed : RemovedLinkedList<E>
add(E) : boolean
iterator(): Iterator<E>
descendingIterator() : Iterator<E>
addFirst(E) :
addLast(E) :
offerFirst(E) : boolean
offerLast(E) : boolean
removeFirst() : E
removeLast() : E
pollFirst() : E
pollLast() : E
getFirst() : E
getLast() : E
peekFirst() : E
peekLast() : E
removeFirstOccurrence(Object) : boolean
removeLastOccurrence(Object) : boolean
offer(E) : boolean
remove() : E
poll() : E
element() : E
peek() : E
push() : E
pop() : E
size() : int
toString() : String
RecursiveMethods
OPERATORS : String
reverseString(String) : String
elfishWord(String) : boolean
elfishRecursive(String, boolean, boolean, boolean) : boolean
recursiveSelectionSort(int[], int, int) :
findMinimumIndex(int[], int, int) : int
calculateOperator(char, Stack<Integer>) : int
calculateOperatorReverse(char, Stack<Integer>): int
isOperator(char) : boolean
EvaluatePrefix(String) : int
EvaluatePostfix(String) : int
prefix(String, Stack<Integer>) : int
postfix(String, Stack<Integer>) : int
printPattern(int[][], int, int) :
recursivePrint(int[][], int, int, int, int) :
printTopLine(int[][], int, int, int, int) :
printRightLine(int[][], int, int, int, int) :
printBottomLine(int[][], int, int, int, int) :
printLeftLine(int[][], int, int, int, int) :
SyntaxErrorException
MyLinkedList
E
head : Node<E>
tail : Node<E>
removed : RemovedLinkedList<E>
size : int
size() : int
get() : E
addFirst(E) :
addLast(E) :
add(E) : boolean
iterator(): Iterator<E>
descendingIterator(): Iterator<E>
listIterator(int) : ListIterator<E>
listIterator() : ListIterator<E>
RemovedLinkedList
E
head : Node<E>
tail : Node<E>
size : int
addNode(Node<E>) :
removeNode() : Node<E>
size() : int
get() : E
addFirst(E) :
addLast(E) :
add(E) : boolean
iterator(): Iterator<E>
descendingIterator(): Iterator<E>
listIterator(int) : ListIterator<E>
listIterator() : ListIterator<E>
AbstractCollection
Deque
Exception
MyListIterator
nextItem : Node<E>
lastItemReturned : Node<E>
index : int
hasNext() : boolean
next() : E
hasPrevious() : boolean
previous() : E
nextIndex() : int
previousIndex() : int
remove() :
add() :
set() :
Node
E
data : E
next : Node<E>
prev : Node<E>
AbstractList
ListIterator

Problem Solution Approach:

1. Implement a Deque class which extends from AbstractCollection and implements Deque interface
2. This class should keep two linked list
3. One is to keep elements inside of it
4. The other one is keep the removed nodes.
5. When a new node is needed instead of creating new node, use a removed node.
6. I implement my ordinary LinkedList which has all methods and fields necessary.
7. The list that will keep nodes should have a reference to the list that contains removed nodes.
8. And removed node list has all necessary methods and fields and also, it has addNode and removeNode methods to add removed node to the list and remove it when it is needed.
9. When a new element is adding to the list, firstly, it takes a node from removed list if there is any available. If there is not, creates a new node.
10. When removing an element from the list, adds that node to removed list

Recursive Methods:

1. Reverse a String: It must stop when there is any word other than current input. Which means, there is no whitespace character in the input. Method finds whitespace character and divides two from whitespace. Then recalls recursive with remaining substring. When there is no input to divide return input.
2. Elfish Word: It stops when String reaches the end. It goes character by character. It keeps three Boolean values for each letter. When a letter matches it Boolean value becomes true. At the end, returns Boolean values.
3. Selection Sort: It returns when i equals to index. i is the position of the element. Firstly, we have to find minimum value. I used another recursion to find it. It goes to end of the array and returns which element is smaller

one by one. Swaps current element to minimum element and goes next element in array.

4. Evaluating prefix: It takes a string. Firstly, reverse that string and we have postfix notation, but it is like reversed. Create an empty stack to store operands. When a number is encountered pushes it to stack. A operator is encountered pops two elements from stack and does calculation and pushes result to stack. When there is no element to check, pops one more element from stack and returns it.
5. Evaluating postfix: It takes a string. Firstly, Create an empty stack to store operands. When a number is encountered pushes it to stack. A operator is encountered pops two elements from stack and does calculation and pushes result to stack. When there is no element to check, pops one more element from stack and returns it.
6. Printing elements: We have two-dimensional array and want to print it spiral matrix. I thought, we can split it to rectangulars. So it will print all elements topside rectangular then goes inside. When there is no rectangular to print, returns.

Test Cases:

Deque Tests:

Test Scenario: Add methods work properly

Test Data:

```
MyDequeClass<Integer> test = new MyDequeClass<>();
test.add(1);
test.addFirst(2);
test.addLast(3);
test.push(4);
test.offer(5);
test.offerLast(6);
test.offerFirst(7);
test.push(8);
test.push(9);
test.offerFirst(10);
```

```
for (Integer iter : test) {
   System.out.print(iter + " ");
}
System.out.println();
```

Expected: 10 9 8 7 4 2 1 3 5 6

Pass/Fail: Passed

Test Scenario: Iter.remove works

Test Data:

```
Iterator<Integer> iter = test.iterator();
for(int i = 0; i < 2; i++)
   iter.next();
iter.remove();
for (Integer it : test) {
   System.out.print(it + " ");
}
```

Expected: 10 8 7 4 2 1 3 5 6

Pass/Fail: Passed

Test Scenario: checks all get methods and remove methods

Test Data:

```
System.out.println(test.getFirst());
System.out.println(test.getLast());
System.out.println(test.element());
System.out.println(test.peek());
System.out.println(test.peekFirst());
System.out.println(test.peekLast());
System.out.println(test.poll());
System.out.println(test.pollFirst());
System.out.println(test.pollLast());
System.out.println(test.pop());
System.out.println(test.remove());
System.out.println(test.removeFirst());
System.out.println(test.removeLast());
System.out.println(test);
```

Excepted: 10

6

10

10

10

6

10

10

6

10

8

7

6

4 2 1 3 5

Pass/Fail: Passed

Test Scenario: removeFirstOccurence and removeLastOccurence

Test data:

```
System.out.println(test.removeFirstOccurrence(1));
System.out.println(test.removeLastOccurrence(2));
System.out.println(test);
```

Expected:

true

true

4 3 5

Pass/Fail: Passed

Recursive Methods:

Test All Methods:

Test Data:

```
String test = new String("this function writes the sentence in reverse");
System.out.println(RecursiveMethods.reverseString(test));
System.out.println(RecursiveMethods.elfishWord("tasteful"));
int[] arr = {3,1,5,6,31,2,7};
RecursiveMethods.recursiveSelectionSort(arr,7,2);
System.out.println(Arrays.toString(arr));
System.out.println(RecursiveMethods.EvaluatePostfix("5 8 *"));
System.out.println(RecursiveMethods.EvaluatePostfix("20 100 2 10 * - 5 / + 15 + 150 30 / -"));
System.out.println(RecursiveMethods.EvaluatePrefix("* 5 8"));
System.out.println(RecursiveMethods.EvaluatePrefix("+ 20 + / - 100 * 2 10 5 - 15 / 150 30"));
int[][] a = {{1,2,3,4},{5,6,7,8},{9,10,11,12},{13,14,15,16},{17,18,19,20}};
RecursiveMethods.printPattern(a,4,5);
int[][] b= {{1}};
RecursiveMethods.printPattern(b,1,1);
```

Expected:

reverse in sentence the writes function this

true

[3, 1, 2, 5, 6, 7, 31]

40

46

40

46

1 2 3 4 8 12 16 20 19 18 17 13 9 5 6 7 11 15 14 10

1

Pass/Fail: Passed