CS146Section8Lab3 Name: Michael Huang Due Date: February 24th, 2016

Ex-1: Given two sorted lists, L1 and L2, write a procedure to compute intersection of L1 and L2 using only the basic list operations. Test your code!

Code:

import java.util.List;

import java.util.ArrayList;

public class Intersection {

public static <T> ArrayList<T> intersect(List<T> l1, List<T> l2) {

ArrayList<T> list = new ArrayList<T>();

for (T element: l1) {

if (l2.contains(element) && !list.contains(element)) {

list.add(element);

}

}

return list;

}

public static void main(String[] args) {

List<Integer> a = new ArrayList<Integer>();

List<Integer> b = new ArrayList<Integer>();

a.add(1);

a.add(2);

a.add(3);

a.add(4);

b.add(1);

b.add(3);

b.add(4);

b.add(4);

b.add(5);

b.add(6);

System.out.println(a);

System.out.println(b);

System.out.println("Intersection: " + intersect(a, b));

}

}

Console output:



Ex-2: Given two sorted, L1 and L2, write a procedure to compute union of L1 and L2 using only the basic list operations. Test your code!

Code:

import java.util.List;

import java.util.ArrayList;

public class Union {

public static <T> ArrayList<T> union(List<T> l1, List<T> l2) {

ArrayList<T> list = new ArrayList<T>();

for (T element : l1) {

if (!list.contains(element)) {

list.add(element);

}

}

for (T element : l2) {

if (!list.contains(element)) {

list.add(element);

}

}

return list;

}

public static void main(String[] args) {

List<Integer> a = new ArrayList<Integer>();

List<Integer> b = new ArrayList<Integer>();

a.add(1);

a.add(2);

a.add(3);

a.add(4);

b.add(1);

b.add(3);

b.add(4);

b.add(4);

b.add(5);

b.add(6);

System.out.println(a);

System.out.println(b);

System.out.println("Union: " + union(a, b));

}

}

Console output:



Ex-3: Write a program to evaluate a postfix expression. Assume the function evaluates a postfix expression, using + ,–, \* and / ending in =. Test your code!

Code:

import java.util.Stack;

import java.util.Scanner;

public class EvaluatePostfix {

public static int postfix(String input) {

try (Scanner in = new Scanner(input)) {

Stack<Integer> stack = new Stack<Integer>();

while (in.hasNext()) {

String current = in.next();

if (current.equals("=")) {

return stack.pop();

} else if (Character.isDigit(current.charAt(0))) {

stack.push(Integer.parseInt(current));

} else {

int b = stack.pop();

int a = stack.pop();

if (current.equals("+")) {

stack.push(a + b);

} else if (current.equals("-")) {

stack.push(a - b);

} else if (current.equals("\*")) {

stack.push(a \* b);

} else {

stack.push(a / b);

}

}

}

return 0;

}

}

public static void main(String[] args) {

System.out.println("1 2 + 3 \* 6 + 2 3 + / = " + postfix("1 2 + 3 \* 6 + 2 3 + / ="));

System.out.println("3 2 + 10 \* 10 / = " + postfix("3 2 + 10 \* 10 / ="));

}

}

Console output:



Ex-4: Write routines to implement two stacks using only one array. Your stack routines should not declare an overflow unless every slot in the array is used.

Code:

import java.util.\*;

public class DoubleStackArray<T> {

private Object[] stack;

private int stack1Front;

private int stack2Front;

public DoubleStackArray(int size) {

stack = new Object[size];

stack1Front = -1;

stack2Front = size;

}

public void pushStack1(T adding) {

if (isFull()) {

throw new StackOverflowError();

}

stack1Front++;

stack[stack1Front] = adding;

}

public void pushStack2(T adding) {

if (isFull()) {

throw new StackOverflowError();

}

stack2Front--;

stack[stack2Front] = adding;

}

public boolean isFull() {

if (stack2Front - stack1Front < 2) {

return true;

} else {

return false;

}

}

public Object popStack1() {

if (stack1Front < 0) {

throw new EmptyStackException();

}

Object pop = stack[stack1Front];

stack1Front--;

return pop;

}

public Object popStack2() {

if (stack2Front > stack.length - 1) {

throw new EmptyStackException();

}

Object pop = stack[stack2Front];

stack2Front++;

return pop;

}

public Object peekStack1() {

if (stack1Front < 0) {

throw new EmptyStackException();

}

Object peek = stack[stack1Front];

return peek;

}

public Object peekStack2() {

if (stack2Front > stack.length - 1) {

throw new EmptyStackException();

}

Object peek = stack[stack2Front];

return peek;

}

public static void main(String[] args) {

DoubleStackArray<Integer> test = new DoubleStackArray<Integer>(3);

test.pushStack1(69);

test.pushStack1(420);

test.pushStack2(1);

System.out.println("size: 3, stack1: [69, 420], stack2: [1]");

System.out.println("peekStack1: " + test.peekStack1());

System.out.println("popStack1: " + test.popStack1());

System.out.println("peekStack1: " + test.peekStack1());

System.out.println("popStack1: " + test.popStack1());

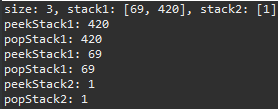
System.out.println("peekStack2: " + test.peekStack2());

System.out.println("popStack2: " + test.popStack2());

}

}

Console output:



Ex-5: You are given a list, L and another list P, containing integers sorted in ascending order. The operation printLots(L, P) will print the elements in L that are in positions specified by P. For instance, if P = 1, 3, 4, 6, the elements in positions 1, 3, 4, and 6 in L are printed. Write the procedure printLots(L, P). You may use only the public Collection API container operations. What is the running time of your procedure?

Code:

import java.util.\*;

public class Printing {

public static <T> void printLots(List<T> l, List<Integer> p) {

for (int i: p) {

if (i < l.size()) {

System.out.println(l.get(i));

}

}

}

public static void main(String[] args) {

List<String> l = new ArrayList<String>();

l.add("zero");

l.add("one");

l.add("two");

l.add("three");

l.add("four");

l.add("five");

l.add("six");

l.add("seven");

l.add("eight");

l.add("nine");

List<Integer> p = new ArrayList<Integer>();

p.add(0);

p.add(1);

p.add(5);

p.add(6);

p.add(8);

printLots(l, p);

}

}

Console output:



List l contains the String of numbers 0 to 9, whereas List p contains the indexes 0, 1, 5, 6, and 8, and would print the corresponding index’s word.

The complexity of the algorithm is O(n). There is one for loop, which loops through the elements in List P. Inside the for loop, there is an if statement, which is O(1), and inside it, a print statement of l.get(i). ArrayList’s get method is O(1), for a total of a max of O(n) + O(1) + O(1) per loop, a total of O(2n), which is just O(n) because 2n <= 2\*n where c = 2 and n0 at any n.

Ex-6: Using the algorithm convertToPostfix given in lecture, convert each of the following infix expressions to postfix expressions. Use the stack technique but check your work using pencil and paper technique.

A: a \* b / (c- d)

|  |  |  |
| --- | --- | --- |
| Reading input | Stack | Postfix |
| a |  | a |
| a \* | \* | a |
| a \* b | \* | a b |
| a \* b / | \* / | a b |
| a \* b / | / | a b \* |
| a \* b / ( | / ( | a b \* |
| a \* b / (c | / ( | a b \* c |
| a \* b / (c - | / ( - | a b \* c |
| a \* b / (c - d | / ( - | a b \* c d |
| a \* b / (c – d) | / ( - ) | a b \* c d |
| a \* b / (c – d) | / ( - | a b \* c d |
| a \* b / (c – d) | / ( | a b \* c d - |
| a \* b / (c – d) | / | a b \* c d - |
| a \* b / (c – d) |  | a b \* c d - / |

B: (a – b \* c) / (d \* e \* f + g)

|  |  |  |
| --- | --- | --- |
| Reading input | Stack | Postfix |
| ( | ( |  |
| (a | ( | a |
| (a - | ( - | a |
| (a - b | ( - | a b |
| (a – b \* | ( - \* | a b |
| (a – b \* c | ( - \* | a b c |
| (a – b \* c) | ( - \* ) | a b c |
| (a – b \* c) | ( - \* | a b c |
| (a – b \* c) | ( - | a b c \* |
| (a – b \* c) | ( | a b c \* - |
| (a – b \* c) |  | a b c \* - |
| (a – b \* c) / | / | a b c \* - |
| (a – b \* c) / ( | / ( | a b c \* - |
| (a – b \* c) / (d | / ( | 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 |
| (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 + | / ( \* + | a b c \* - d e \* f |
| (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) | / ( + ) | a b c \* - d e \* f \* g |
| (a – b \* c) / (d \* e \* f + g) | / ( + | a b c \* - d e \* f \* g |
| (a – b \* c) / (d \* e \* f + g) | / ( | a b c \* - d e \* f \* g + |
| (a – b \* c) / (d \* e \* f + g) | / | a b c \* - d e \* f \* g + |
| (a – b \* c) / (d \* e \* f + g) |  | a b c \* - d e \* f \* g + / |

C: a / b \* (c + (d – e))

|  |  |  |
| --- | --- | --- |
| Reading input | Stack | Postfix |
| a |  | a |
| a / | / | a |
| a / b | / | a b |
| a / b \* | / \* | a b |
| a / b \* | \* | a b / |
| a / b \* ( | \* ( | a b / |
| a / b \* (c | \* ( | a b / c |
| a / b \* (c + | \* ( + | a b / c |
| a / b \* (c + ( | \* ( + ( | a b / c |
| a / b \* (c + (d | \* ( + ( | 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 |
| a / b \* (c + (d – e) | \* ( + ( - | a b / c d e |
| a / b \* (c + (d – e) | \* ( + ( | a b / c d e - |
| a / b \* (c + (d – e) | \* ( + | a b / c d e - |
| a / b \* (c + (d – e)) | \* ( + ) | a b / c d e - |
| a / b \* (c + (d – e)) | \* ( + | a b / c d e - |
| a / b \* (c + (d – e)) | \* ( | a b / c d e - + |
| a / b \* (c + (d – e)) | \* | a b / c d e - + |
| a / b \* (c + (d – e)) |  | a b / c d e - + \* |

I also spent 5 hours writing the code. Is there a chance of extra credit? Haha.

Code:

import java.util.\*;

public class Postfix {

public static String convertToPostfix(String input) {

Stack<String> operators = new Stack<String>();

input = input.replaceAll("\\s+", "");

String postfix = "";

for (int i = 0; i < input.length(); i++) {

String current = input.substring(i, i + 1);

if (operators.isEmpty() || current.equals("(") || operators.peek().equals("(")) {

operators.push(current);

} else if (isOperator(current)) {

if (order(current) <= order(operators.peek())) {

while (operators.size() > 0 && !operators.peek().equals("(")

&& order(current) <= order(operators.peek())) {

postfix += operators.pop() + " ";

}

operators.push(current);

} else {

operators.push(current);

}

} else {

if (current.equals(")")) {

while (operators.size() > 0 && !operators.peek().equals("(")) {

postfix += operators.pop() + " ";

}

operators.pop();

} else {

postfix += current + " ";

}

}

}

while (operators.size() > 0) {

postfix += operators.pop() + " ";

}

return postfix.substring(0, postfix.length() - 1);

}

public static boolean isOperator(String input) {

if (input.equals("+") || input.equals("-") || input.equals("\*") || input.equals("/")) {

return true;

} else {

return false;

}

}

public static int order(String operator) {

if (operator.equals("+") || operator.equals("-")) {

return 1;

} else {

return 2;

}

}

public static void main(String[] args) {

System.out.println(convertToPostfix("a \* b / (c - d)"));

System.out.println("a b \* c d - /");

System.out.println(convertToPostfix("a \* b / (c - d)").equals("a b \* c d - /"));

System.out.println(convertToPostfix("(a - b \* c) / (d \* e \* f + g)"));

System.out.println("a b c \* - d e \* f \* g + /");

System.out.println(convertToPostfix("(a - b \* c) / (d \* e \* f + g)").equals("a b c \* - d e \* f \* g + /"));

System.out.println(convertToPostfix("a / b \* (c + (d - e))"));

System.out.println("a b / c d e - + \*");

System.out.println(convertToPostfix("a / b \* (c + (d - e))").equals("a b / c d e - + \*"));

}

}