Data Structure Lab7 : Stack 2022-2023

Topics

1. Create Stack Interface

2. Create Stack Using Array

3. Create Stack Using Linked Lists

4. Implement Basic Methods of Stack

● isEmpty()

● size()

● top()

● push(E e)

● pop()

Homework

1. Implement a method with signature transfer(S, T) that transfers all elements from stack S onto stack T, so that the element that starts at the top of S is the first to be inserted onto T, and the element at the bottom of S ends up at the top of T.

public void transfer(Stack<E> S, Stack<E> T) {

Stack<E> auxStack = new Stack<>(); // Auxiliary stack to reverse order

// Move all elements from S to auxStack

while (!S.isEmpty()) {

auxStack.push(S.pop());

}

// Move all elements from auxStack to T (now they are in correct order)

while (!auxStack.isEmpty()) {

T.push(auxStack.pop());

}

}

2. Give a recursive method for removing all the elements from a stack.

public void clearStack(Stack<E> stack) {

if (!stack.isEmpty()) {

stack.pop(); // Remove the top element

clearStack(stack); // Recursively remove the next element

}

}

3. Postfix notation is an unambiguous way of writing an arithmetic expression without parentheses. It is defined so that if

“(exp1)op(exp2)” is a normal fully parenthesized expression whose operation is op, the postfix version of this is “pexp1 pexp2 op”, where pexp1 is the postfix version of exp1 and pexp2 is the postfix version of exp2. The postfix version of a single number or variable is just that number or variable. So, for example, the postfix version of “((5 + 2) (8 − 3))/4” is “5 2 + 8 3 − 4 /”. Describe a nonrecursive way of evaluating an expression in postfix notation.

import java.util.Stack;

public class PostfixEvaluator {

public double evaluatePostfix(String[] expression) {

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

for (String token : expression) {

if (isNumber(token)) {

stack.push(Double.parseDouble(token));

} else {

double operand2 = stack.pop();

double operand1 = stack.pop();

double result = applyOperator(operand1, operand2, token);

stack.push(result);

}

}

return stack.pop(); // The result is the last element in the stack

}

private boolean isNumber(String token) {

try {

Double.parseDouble(token);

return true;

} catch (NumberFormatException e) {

return false;

}

}

private double applyOperator(double operand1, double operand2, String operator) {

switch (operator) {

case "+": return operand1 + operand2;

case "-": return operand1 - operand2;

case "\*": return operand1 \* operand2;

case "/": return operand1 / operand2;

default: throw new IllegalArgumentException("Unknown operator: " + operator);

}

}

public static void main(String[] args) {

PostfixEvaluator evaluator = new PostfixEvaluator();

String[] expression = {"5", "2", "+", "8", "3", "-", "4", "/", "/"};

System.out.println(evaluator.evaluatePostfix(expression)); // Output: 5.6

}

}

Data Structure Lab7 : Stack 2022-2023

4. Implement the clone( ) method for the ArrayStack class.

public class ArrayStack<E> {

private Object[] data;

private int size;

public ArrayStack() {

data = new Object[10];

size = 0;

}

public void push(E element) {

if (size == data.length) resize(2 \* data.length);

data[size++] = element;

}

public E pop() {

if (size == 0) throw new IllegalStateException("Stack is empty");

E element = (E) data[--size];

data[size] = null;

return element;

}

@Override

public ArrayStack<E> clone() {

ArrayStack<E> cloned = new ArrayStack<>();

cloned.size = this.size;

cloned.data = new Object[this.size];

System.arraycopy(this.data, 0, cloned.data, 0, this.size);

return cloned;

}

private void resize(int capacity) {

Object[] newData = new Object[capacity];

System.arraycopy(data, 0, newData, 0, size);

data = newData;

}

}

5. Implement a program that can input an expression in postfix notation (see Exercise C-6.19) and output its value.

import java.util.Scanner;

import java.util.Stack;

public class PostfixEvaluator {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.println("Enter a postfix expression (space-separated):");

String expression = scanner.nextLine();

try {

double result = evaluatePostfix(expression.split(" "));

System.out.println("Result: " + result);

} catch (Exception e) {

System.out.println("Error evaluating the expression: " + e.getMessage());

}

scanner.close();

}

public static double evaluatePostfix(String[] expression) {

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

for (String token : expression) {

if (isNumber(token)) {

stack.push(Double.parseDouble(token)); // Push number to stack

} else {

double operand2 = stack.pop(); // Pop second operand

double operand1 = stack.pop(); // Pop first operand

double result = applyOperator(operand1, operand2, token); // Apply operator

stack.push(result); // Push result back to stack

}

}

return stack.pop(); // The result is the last element in the stack

}

private static boolean isNumber(String token) {

try {

Double.parseDouble(token);

return true;

} catch (NumberFormatException e) {

return false;

}

}

private static double applyOperator(double operand1, double operand2, String operator) {

switch (operator) {

case "+": return operand1 + operand2;

case "-": return operand1 - operand2;

case "\*": return operand1 \* operand2;

case "/": return operand1 / operand2;

default: throw new IllegalArgumentException("Invalid operator: " + operator);

}

}

}