Министерство цифрового развития, связи и массовых коммуникаций Государственное образовательного учреждение высшего образования

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Лабораторная работа № 4

«Реализация стека/дека»

Выполнил студент

группы БВТ1902

Гинатуллин Айгиз

Москва

2021

**Задание**

Реализовать следующие структуры данных:

●  Стек (stack):  
операции *для стека*: инициализация, проверка на пустоту, добавление нового элемента в начало, извлечение элемента из начала;

●  Дек (двусторонняя очередь, deque):  
операции *для дека*: инициализация, проверка на пустоту, добавление нового элемента в начало, добавление нового элемента в конец, извлечение элемента из начала, извлечение элемента из конца.

Разработать программу обработки данных, содержащихся в заранее подготовленном txt-файле, в соответствии с заданиями, применив указанную в задании структуру данных. Результат работы программы вывести на экран и сохранить в отдельном txt-файле.

Оформить отчет о лабораторной работе в ipynb или pdf-файле.

Задания:

* 1. Отсортировать строки файла, содержащие названия книг, в алфавитном порядке с использованием двух *деков*.
  2. *Дек* содержит последовательность символов для шифровки сообщений. Дан текстовый файл, содержащий зашифрованное сообщение. Пользуясь *деком,* расшифровать текст. Известно, что при шифровке каждый символ сообщения заменялся следующим за ним в *деке* по часовой стрелке через один.
  3. Даны три стержня и *n* дисков различного размера. Диски можно надевать на стержни, образуя из них башни. Перенести *n* дисков со стержня *А* на стержень *С*, сохранив их первоначальный порядок. При переносе дисков необходимо соблюдать следующие правила:

- на каждом шаге со стержня на стержень переносить только один диск;  
- диск нельзя помещать на диск меньшего размера;  
- для промежуточного хранения можно использовать стержень *В*.  
Реализовать алгоритм, используя три *стека* вместо стержней *А*, *В*, *С*. Информация о дисках хранится в исходном файле.

* 1. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя *стек*.
  2. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс квадратных скобок в тексте, используя *дек*.

1. Дан файл из символов. Используя *стек*, за один просмотр файла напечатать сначала все цифры, затем все буквы, и, наконец, все остальные символы, сохраняя исходный порядок в каждой группе символов.
2. Дан файл из целых чисел. Используя *дек*, за один просмотр файла напечатать сначала все отрицательные числа, затем все положительные числа, сохраняя исходный порядок в каждой группе.
3. Дан текстовый файл. Используя *стек*, сформировать новый текстовый файл, содержащий строки исходного файла, записанные в обратном порядке: первая строка становится последней, вторая – предпоследней и т.д.
4. Дан текстовый файл. Используя *стек*, вычислить значение логического выражения, записанного в текстовом файле в следующей форме:  
   < ЛВ > ::= T | F | (N<ЛВ>) | (<ЛВ>A<ЛВ>) | (<ЛВ>X<ЛВ>) | (<ЛВ>O<ЛВ>),  
   где буквами обозначены логические константы и операции:

T – True, F – False, N – Not, A – And, X – Xor, O – Or.

1. Дан текстовый файл. В текстовом файле записана формула следующего вида: <Формула> ::= <Цифра> | M(<Формула>,<Формула>) | N(Формула>,<Формула>) < Цифра > ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  
   где буквами обозначены функции:  
   M – определение максимума, N – определение минимума.  
   Используя *стек*, вычислить значение заданного выражения.

11.Дан текстовый файл. Используя *стек*, проверить, является ли содержимое текстового файла правильной записью формулы вида:  
< Формула > ::= < Терм > | < Терм > + < Формула > | < Терм > - < Формула >  
< Терм > ::= < Имя > | (< Формула >)

< Имя > ::= x | y | z

# Выполнение

Листинг программы:

Lab4

import Deque from "./Classes/Deque.js"  
import Stack from "./Classes/Stack.js";  
import {dequeSort} from "./tasks/task1.js";  
import {bracketFinderStack} from "./tasks/task4.js";  
import {bracketFinderDeque} from "./tasks/task5.js";  
import {regexParse} from "./tasks/task6.js";  
import {numbersParse} from "./tasks/task7.js";  
import {stringRevers} from "./tasks/task8.js";  
import {computeForm, computeLogic1, computeMinMax} from "./tasks/task9-11.js";  
import {han} from "./tasks/task3.js";  
import {cip} from "./tasks/task2.js";  
  
  
  
//1  
console.log(dequeSort([10,11,12,1,123,13414]))  
//2  
let str="12191(45197689)"  
let dec=new Deque  
dec.pushFront("1");  
dec.pushFront("2");  
dec.pushFront("т");  
dec.pushFront("у");  
dec.pushFront("9");  
dec.pushFront("(");  
dec.pushFront(" ");  
dec.pushFront("е");  
dec.pushFront("4");  
dec.pushFront("5");  
dec.pushFront("к");  
dec.pushFront("с");  
dec.pushFront("6");  
dec.pushFront("7");  
dec.pushFront("л");  
dec.pushFront("д");  
dec.pushFront("8");  
dec.pushFront(")");  
dec.pushFront("я");  
dec.pushFront("3");  
  
console.log(cip(str,dec))  
//3  
let kol=3  
han(kol);  
//4  
console.log(bracketFinderStack("(((((dadasd)()adada()(adsad))adad)))".split("")))  
//5  
console.log(bracketFinderDeque("(((((dadasd)()adada()(adsad))adad))())".split("")))  
//6  
regexParse("a123q@#")  
//7  
numbersParse([1, -1,24,0,-123])  
//8  
stringRevers("aaaaa bbbbbbb cccccc")  
//9  
// console.log(computeLogic({  
// type: "compute",  
// value: "+F+(T\*F+(F+T))"  
// }));  
computeLogic1("(F+(T\*F+(F+T)))X(NT)")  
//10  
computeMinMax("N(9,(M(1,N(1,2))))")  
//11  
computeForm("x+(y+z+(z+y))")

Deque

export default class Deque {  
 constructor() {  
 this.front = this.back = undefined;  
 }  
 pushFront(value) {  
 if (!this.front) this.front = this.back = { value };  
 else this.front = this.front.next = { value, prev: this.front };  
 }  
 popFront() {  
 let value = this.peekFront();  
 if (this.front === this.back) this.front = this.back = undefined;  
 else (this.front = this.front.prev).next = undefined;  
 return value;  
 }  
 peekFront() {  
 return this.front && this.front.value;  
 }  
 isEmpty(){  
 return !(!!this.front && !!this.back)  
 }  
 pushBack(value) {  
 if (!this.front) this.front = this.back = { value };  
 else this.back = this.back.prev = { value, next: this.back };  
 }  
 popBack() {  
 let value = this.peekBack();  
 if (this.front === this.back) this.front = this.back = undefined;  
 else (this.back = this.back.next).back = undefined;  
 return value;  
 }  
 peekBack() {  
 return this.back && this.back.value;  
 }  
}

Stack

export default class Stack {  
 constructor() {  
 this.front = undefined;  
 }  
 isEmpty(){  
 return !!!this.front  
 }  
 pushFront(value) {  
 if (!this.front) this.front = this.back = { value };  
 else this.front = this.front.next = { value, prev: this.front };  
 }  
 popFront() {  
 let value = this.peekFront();  
 if (this.front === this.back) this.front = this.back = undefined;  
 else (this.front = this.front.prev).next = undefined;  
 return value;  
 }  
 peekFront() {  
 return this.front && this.front.value;  
 }  
}

task1

class Deque {  
 constructor() {  
 this.front = this.back = undefined;  
 }  
 pushFront(value) {  
 if (!this.front) this.front = this.back = { value };  
 else this.front = this.front.next = { value, prev: this.front };  
 }  
 popFront() {  
 let value = this.peekFront();  
 if (this.front === this.back) this.front = this.back = undefined;  
 else (this.front = this.front.prev).next = undefined;  
 return value;  
 }  
 peekFront() {  
 return this.front && this.front.value;  
 }  
 isEmpty(){  
 return !(!!this.front && !!this.back)  
 }  
 pushBack(value) {  
 if (!this.front) this.front = this.back = { value };  
 else this.back = this.back.prev = { value, next: this.back };  
 }  
 popBack() {  
 let value = this.peekBack();  
 if (this.front === this.back) this.front = this.back = undefined;  
 else (this.back = this.back.next).back = undefined;  
 return value;  
 }  
 peekBack() {  
 return this.back && this.back.value;  
 }  
}  
export const dequeSort = (array) =>{  
 let dequeIncome = new Deque;  
 let dequeSorted = new Deque;  
 let sortedArray = [];  
 array.map(item => dequeIncome.pushFront(item))  
  
 while (!dequeIncome.isEmpty()){  
 if((dequeSorted.peekFront() <= dequeIncome.peekFront()) || dequeSorted.isEmpty()){  
 dequeSorted.pushFront(dequeIncome.popFront())  
 }  
 else{  
 dequeIncome.pushBack(dequeSorted.popFront())  
 }  
 }  
  
 while (!dequeSorted.isEmpty()){  
 sortedArray = [...sortedArray,dequeSorted.popFront()]  
 }  
 return sortedArray;  
}

task2

export const cip = (str, dec) => {  
 let str1 = "";  
 for(let i=0;i<str.length;i++){  
 while (str1.length<i+1){  
 if(dec.peekBack()===str[i]){  
 dec.pushFront(dec.popBack())  
 dec.pushFront(dec.popBack())  
 // dec.pushFront(dec.popBack())  
 // dec.pushFront(dec.popBack())  
 str1+=dec.peekBack()  
 }  
 dec.pushFront(dec.popBack())  
 }  
 }  
 return str1  
}

task3

import Stack from "../Classes/Stack.js";  
  
  
export const transfer\_disk = (a, b) => {  
 if (b.isEmpty() === true) {  
 b.pushFront(a.peekFront());  
 a.popFront();  
 return 1;  
 } else if (a.isEmpty() === true) {  
 a.pushFront(b.peekFront());  
 b.popFront();  
 return 2;  
 } else {  
 if (b.peekFront() > a.peekFront()) {  
 b.pushFront(a.peekFront());  
 a.popFront();  
 return 1;  
 } else {  
 a.pushFront(b.peekFront());  
 b.popFront();  
 return 2;  
 }  
 }  
}  
export const han = (kol) => {  
 let s = new Stack()  
 let a = new Stack()  
 let d = new Stack()  
 let n = kol  
 for (let i = n; i >= 1; i--) {  
 s.pushFront(i);  
 }  
  
 let x = Math.pow(2, n) - 1  
 let i = 1  
  
 if (n % 2 === 0) {  
 while (i <= x) {  
 if (i % 3 === 1) {  
 let y = transfer\_disk(s, a)  
 if (y === 1) {  
 console.log("Переместить диск " + a.peekFront() + " с StackA на StackB")  
 } else  
 console.log("Переместить диск " + s.peekFront() + " с StackB на StackA")  
 } else if (i % 3 === 2) {  
 let y = transfer\_disk(s, d)  
 if (y === 1) {  
 console.log("Переместить диск " + d.peekFront() + " с StackA на StackC")  
 } else  
 console.log("Переместить диск " + s.peekFront() + " с StackC на StackA")  
 } else {  
 let y = transfer\_disk(a, d)  
 if (y === 1) {  
 console.log("Переместить диск " + d.peekFront() + " с StackB на StackC")  
 } else  
 console.log("Переместить диск " + a.peekFront() + " с StackC на StackB")  
 }  
 i++  
 }  
 } else {  
 while (i <= x) {  
 if (i % 3 === 1) {  
 let y = transfer\_disk(s, d);  
 if (y === 1) {  
 console.log("Переместить диск " + d.peekFront() + " с StackA на StackC")  
 } else  
 console.log("Переместить диск " + s.peekFront() + " с StackC на StackA")  
 } else if (i % 3 === 2) {  
 let y = transfer\_disk(s, a);  
 if (y === 1) {  
 console.log("Переместить диск " + a.peekFront() + " с StackA на StackB")  
 } else  
 console.log("Переместить диск " + s.peekFront() + " с StackB на StackA")  
 } else {  
 let y = transfer\_disk(a, d);  
 if (y === 1) {  
 console.log("Переместить диск " + d.peekFront() + " с StackB на StackC")  
 } else  
 console.log("Переместить диск " + a.peekFront() + " с StackC на StackB")  
 }  
 i++;  
 }  
 }  
 // while (d.isEmpty() !== true) {  
 // console.log(d.peekFront());  
 // d.popFront();  
 // }  
 return 0;  
}

task4

import Stack from "../Classes/Stack.js";  
  
export const bracketFinderStack = (array) => {  
 let stack = new Stack()  
 let flag = true;  
 array.map(item => {  
 if (item === '(') {  
 stack.pushFront('(')  
 } else if (item === ')') {  
 if (!stack.isEmpty()) {  
 stack.popFront()  
 } else {  
 flag = false  
 }  
 }  
 })  
 return flag && !!stack.isEmpty()  
}

task5

import Deque from "../Classes/Deque.js";  
  
  
export const bracketFinderDeque = (array) => {  
 let deque = new Deque()  
 let flag = true;  
 array.map(item => {  
 if (item === '(') {  
 deque.pushFront('(')  
 } else if (item === ')') {  
 if (!deque.isEmpty()) {  
 deque.popBack();  
 } else flag = false;  
 }  
 })  
 return flag && !!deque.isEmpty()  
}

task6

import Stack from "../Classes/Stack.js";  
  
  
export const regexParse = (string) => {  
 let array = string.split("")  
 let numbers = new Stack()  
 let letters = new Stack()  
 let other = new Stack()  
  
 array.map(item => {  
 if (item.match(/[0-9]/)){  
 numbers.pushFront(item)  
 }  
 else if (item.match(/[a-zA-Z]/)){  
 letters.pushFront(item)  
 }  
 else{  
 other.pushFront(item)  
 }  
 })  
  
 let numbersRevers = new Stack()  
 let lettersRevers = new Stack()  
 let otherRevers = new Stack()  
  
 while (!numbers.isEmpty()) {  
 numbersRevers.pushFront(numbers.popFront())  
 }  
 while (!letters.isEmpty()) {  
 lettersRevers.pushFront(letters.popFront())  
 }  
 while (!other.isEmpty()) {  
 otherRevers.pushFront(other.popFront())  
 }  
  
 while (!numbersRevers.isEmpty()) {  
 console.log(numbersRevers.popFront());  
 }  
 while (!lettersRevers.isEmpty()) {  
 console.log(lettersRevers.popFront());  
 }  
 while (!otherRevers.isEmpty()) {  
 console.log(otherRevers.popFront());  
 }  
}

task7

import Deque from "../Classes/Deque.js";  
  
  
export const numbersParse = (array) => {  
 let deque = new Deque();  
 array.map(item =>{  
 if(item < 0){  
 deque.pushBack(item)  
 }  
 else{  
 deque.pushFront(item)  
 }  
 })  
 while (!deque.isEmpty()){  
 ***console***.log(deque.popBack())  
 }  
}

task8

import Stack from "../Classes/Stack.js";  
  
export const stringRevers = (string) =>{  
 let array = string.split(' ')  
 let stack = new Stack()  
 array.map(string =>{  
 stack.pushFront(string);  
 })  
 while(!stack.isEmpty()){  
 ***console***.log(stack.popFront());  
 }  
}

task9-11

import Stack from "../Classes/Stack.js";  
  
const typeOfTerm = (term) => {  
 if (term.length === 2) {  
 return {  
 type: "term",  
 value: term  
 }  
 }  
 return {  
 type: "compute",  
 value: "+" + term  
 }  
}  
  
export const computeLogic = (object) => {  
  
 let currentBooleanValue = 0  
 let termsStack = new Stack()  
  
 if (object.type === "compute") {  
 let stringStack = new Stack()  
 let logicalExpressions = object.value.match(/[+\*][FT]|(?<=\().\*(?=\))/gm)  
  
 console.log(logicalExpressions)  
 logicalExpressions.map(term => {  
 stringStack.pushFront(term)  
 })  
 while (!stringStack.isEmpty()) {  
 if (typeOfTerm(stringStack.peekFront()).type === "term") {  
 termsStack.pushFront(typeOfTerm(stringStack.popFront()))  
 } else {  
 termsStack.pushFront(computeLogic(typeOfTerm(stringStack.popFront())))  
 }  
 }  
 }  
 let reverseStack = termsStack  
 while (!reverseStack.isEmpty()) {  
 let currentAction = reverseStack.popFront()  
 if (currentAction.value === "+T") {  
 currentBooleanValue = 1  
 }  
 if (currentAction.value === '\*F') {  
 currentBooleanValue = 0  
 }  
 }  
 let term = currentBooleanValue ? "+T" : "+F"  
 console.log(!!currentBooleanValue ? "+T" : "+F")  
 return {  
 type: "term",  
 value: term  
 }  
}  
  
  
export const computeLogic1 = (Str) => {  
 let str1 = ""  
 let stk = new Stack()  
 for (let i = 0; i < Str.length; i++) {  
 stk.pushFront(Str[i])  
 }  
 for (let i = 0; i < Str.length; i++) {  
 if (stk.peekFront() === "T")  
 str1 = "true " + str1  
 if (stk.peekFront() === "F")  
 str1 = "false " + str1  
 if (stk.peekFront() === "N")  
 str1 = "! " + str1  
 if (stk.peekFront() === "A" || stk.peekFront() === "\*")  
 str1 = "&& " + str1  
 if (stk.peekFront() === "X")  
 str1 = "!= " + str1  
 if (stk.peekFront() === "O" || stk.peekFront() === "+")  
 str1 = "|| " + str1  
 if (stk.peekFront() === "(")  
 str1 = "( " + str1  
 if (stk.peekFront() === ")")  
 str1 = ")" + str1  
 stk.popFront()  
 }  
 console.log(eval(str1))  
}  
  
  
export const computeMinMax = (Str) => {  
 let str1 = ""  
 let stk = new Stack()  
 for (let i = 0; i < Str.length; i++) {  
 stk.pushFront(Str[i])  
 }  
 for (let i = 0; i < Str.length; i++) {  
 if (stk.peekFront() === "0")  
 str1 = "0" + str1  
 if (stk.peekFront() === "1")  
 str1 = "1" + str1  
 if (stk.peekFront() === "2")  
 str1 = "2" + str1  
 if (stk.peekFront() === "3")  
 str1 = "3" + str1  
 if (stk.peekFront() === "4")  
 str1 = "4" + str1  
 if (stk.peekFront() === "5")  
 str1 = "5" + str1  
 if (stk.peekFront() === "6")  
 str1 = "6" + str1  
 if (stk.peekFront() === "7")  
 str1 = "7" + str1  
 if (stk.peekFront() === "8")  
 str1 = "8" + str1  
 if (stk.peekFront() === "9")  
 str1 = "9" + str1  
 if (stk.peekFront() === "M")  
 str1 = "Math.max" + str1  
 if (stk.peekFront() === "N")  
 str1 = "Math.min" + str1  
 if (stk.peekFront() === "," || stk.peekFront() === ".")  
 str1 = "," + str1  
 if (stk.peekFront() === "(")  
 str1 = "(" + str1  
 if (stk.peekFront() === ")")  
 str1 = ")" + str1  
 stk.popFront()  
 }  
 console.log(eval(str1))  
}  
  
  
export const computeForm = (Str) => {  
 let x = 1, y = 1, z = 1  
 let stk = new Stack()  
 let str = ""  
 for (let i = 0; i < Str.length; i++) {  
 stk.pushFront(Str[i])  
 }  
 for (let i = 0; i < Str.length; i++) {  
 str = stk.popFront() + str  
 }  
 try {  
 eval(str)  
 } catch (err) {  
 console.log(false)  
 }  
 console.log(true)  
}

Вывод программы

[ 13414, 123, 12, 11, 10, 1 ]  
тут текст для 3  
Переместить диск 1 с StackA на StackC  
Переместить диск 2 с StackA на StackB  
Переместить диск 1 с StackC на StackB  
Переместить диск 3 с StackA на StackC  
Переместить диск 1 с StackB на StackA  
Переместить диск 2 с StackB на StackC  
Переместить диск 1 с StackA на StackC  
true  
true  
1  
2  
3  
a  
q  
@  
 #  
-123  
-1  
1  
24  
0  
cccccc  
bbbbbbb  
aaaaa  
true  
1  
true

# Вывод

В ходе выполнения лабораторной работы, я реализовал структуры данных стек и дек, при помощи них отсортировала строку в алфавитном порядке, расшифровал сообщение, решил задачу на ханойские башни, проверил соответствие скобок, рассортировал буквы, цифры (отрицательные и положительные), решил логическое выражение, нашёл максимум и минимум и проверил выражение.