

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

Ордена Трудового Красного Знамени

«Московский технический университет связи и информатики»

Лабораторная работа № 4

по дисциплине «Структура и алгоритмы обработки данных»

Выполнил студент группы БФИ-1901:

Бардюк Д. В.

Москва
2021

Задание

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 A на стержень C , 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 B .Реализовать алгоритм, используя три **стека** вместо стержней A, B, C . Информация о дисках хранится в исходном файле.
4. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс круглых скобок в тексте, используя **стек**.
5. Дан текстовый файл с программой на алгоритмическом языке. За один просмотр файла проверить баланс квадратных скобок в тексте, используя **дек**.
6. 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.
7. 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.
8. 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.
9. Дан текстовый файл. Используя **стек**, вычислить значение логического выражения, записанного в текстовом файле в следующей форме:
 $\langle \text{ЛВ} \rangle ::= \text{T} \mid \text{F} \mid (\text{N}\langle \text{ЛВ} \rangle) \mid (\langle \text{ЛВ} \rangle \text{A} \langle \text{ЛВ} \rangle) \mid (\langle \text{ЛВ} \rangle \text{X} \langle \text{ЛВ} \rangle) \mid (\langle \text{ЛВ} \rangle \text{O} \langle \text{ЛВ} \rangle),$
где буквами обозначены логические константы и операции:
T – True, **F** – False, **N** – Not, **A** – And, **X** – Xor, **O** – Or.
10. Дан текстовый файл. В текстовом файле записана формула следующего вида:
 $\langle \text{Формула} \rangle ::= \langle \text{Цифра} \rangle \mid \text{M}(\langle \text{Формула} \rangle, \langle \text{Формула} \rangle) \mid \text{N}(\langle \text{Формула} \rangle, \langle \text{Формула} \rangle)$
 $\langle \text{Цифра} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
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:
 $\langle \text{Формула} \rangle ::= \langle \text{Терм} \rangle \mid \langle \text{Терм} \rangle + \langle \text{Формула} \rangle \mid \langle \text{Терм} \rangle - \langle \text{Формула} \rangle$
 $\langle \text{Терм} \rangle ::= \langle \text{Имя} \rangle \mid (\langle \text{Формула} \rangle)$
 $\langle \text{Имя} \rangle ::= x \mid y \mid z$

Код программы

Реализация деки

```
function DeQueue(){
    this.head = null;
    this.tail = null;
    this.size = 0;
}

function Node(data){
    this.data = data;
    this.prev = null;
    this.next = null;
}

DeQueue.prototype.pushBack = function(data){
    let node = new Node(data);
    node.prev = null;
    node.next = this.head;
    if (this.head){
        this.head.prev = node;
        this.head = node;
        this.size++;
    }
    else {
        this.head = node;
        this.tail = node;
        this.size++;
    }
};

DeQueue.prototype.pushFront = function(data){
    let node = new Node(data);
    node.prev = this.tail;
    node.next = null;
    if (this.tail){
        this.tail.next = node;
        this.tail = node;
        this.size++;
    }
    else {
        this.head = node;
        this.tail = node;
        this.size++;
    }
};

DeQueue.prototype.popBack = function(){
    if (this.head !== null) {
        let temp = this.head;
```

```

        if(this.head === this.tail){
            this.head = null;
            this.tail = null;
            this.size--;
        }
        else{
            this.head = this.head.next;
            this.head.prev = null;
            this.size--;
        }
        return temp.data;
    }
    return 'It is empty';
};

DeQueue.prototype.popFront = function(){
    if (this.tail !== null) {
        let temp = this.tail;
        if(this.head === this.tail){
            this.head = null;
            this.tail = null;
            this.size --;
        }
        else{
            this.tail = this.tail.prev;
            this.tail.next = null;
            this.size--;
        }
        return temp.data;
    }
    else{ return 'It is empty'}
};

DeQueue.prototype.isEmpty = function(){
    return this.size === 0;
};

DeQueue.prototype.peekBack = function(){
    if(this.head == null){return 'it is empty'}
    return this.head.data;
};

DeQueue.prototype.peekFront = function(){
    if(this.tail == null){return 'it is empty'}
    return this.tail.data;
};

module.exports = {
    DeQueue: DeQueue
};

```

Реализация стека

```
function Stack() {
  this._size = 0;
  this._top = null;
}

function Node(data) {
  this.data = data;
  this.previous = null;
}

Stack.prototype.push = function (data) {
  var node = new Node(data);
  node.previous = this._top;
  this._top = node;
  this._size++;
}

Stack.prototype.peek = function () {
  return this._top.data;
}

Stack.prototype.pop = function () {
  if (this._top !== null) {
    let temp = this._top.data;
    this._top = this._top.previous;
    this._size--;

    return temp;
  }
  return `It is empty`
}

Stack.prototype.isEmpty = function () {
  return this._size == 0;
}

module.exports = {
  Stack: Stack
};
```

Задания

```
const DeQueue = require('./queue');
function task1(array){
  let dequeueInc = new DeQueue.DeQueue();
  let dequeueSort = new DeQueue.DeQueue();
  let sortArr = [];
  array.map(item => dequeueInc.pushFront(item))
```

```

    while(!dequeueInc.isEmpty()){
        console.log(dequeueInc.peekFront())
        if(dequeueInc.peekFront() <= dequeueSort.peekFront() || dequeueSort.isEmpty()){
            dequeueSort.pushFront(dequeueInc.popFront());
        }
        else{
            dequeueInc.pushBack(dequeueSort.popFront());
        }
    }

    while(!dequeueSort.isEmpty()){
        sortArr = [...sortArr, dequeueSort.popFront()];
    }

    return sortArr;
}

console.log(task1([3,2,1,5,6,7,8]))

```

```

const DeQueue = require('./queue');
let dec = new DeQueue.DeQueue()
let str = `abcdef`;
function 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())
                str1+=dec.peekBack();
            }
            dec.pushFront(dec.popBack())
        }
    }
    return str1
}
console.log(cip(str,dec))

```

```

const Stack = require('./stack')
function transferDisk(a, b){
    if (b.isEmpty() === true) {
        b.push(a.peek());
        a.pop();
        return 1;
    } else if (a.isEmpty() === true) {
        a.push(b.peek());
    }
}

```

```

        b.pop();
        return 2;
    } else {
        if (b.peek() > a.peek()) {
            b.push(a.peek());
            a.pop();
            return 1;
        } else {
            a.push(b.peek());
            b.pop();
            return 2;
        }
    }
}
}

function han (kol){
    let s = new Stack.Stack()
    let a = new Stack.Stack()
    let d = new Stack.Stack()
    let n = kol
    for (let i = n; i >= 1; i--) {
        s.push(i);
    }

    let x = Math.pow(2, n) - 1
    let i = 1

    if (n % 2 === 0) {
        while (i <= x) {
            if (i % 3 === 1) {
                let y = transferDisk(s, a)
                if (y === 1) {
                    console.log("Переместить диск " + a.peek() + " с StackA на StackB")
                } else
                    console.log("Переместить диск " + s.peek() + " с StackB на StackA")
            } else if (i % 3 === 2) {
                let y = transferDisk(s, d)
                if (y === 1) {
                    console.log("Переместить диск " + d.peek() + " с StackA на StackC")
                } else
                    console.log("Переместить диск " + s.peek() + " с StackC на StackA")
            } else {
                let y = transferDisk(a, d)
                if (y === 1) {
                    console.log("Переместить диск " + d.peek() + " с StackB на StackC")
                } else
            }
        }
    }
}

```

```

        console.log("Переместить диск " + a.peek() + " с StackC на St
ackB")
    }
    i++
}
} else {
    while (i <= x) {
        if (i % 3 === 1) {
            let y = transferDisk(s, d);
            if (y === 1) {
                console.log("Переместить диск " + d.peek() + " с StackA на St
ackC")
            } else
                console.log("Переместить диск " + s.peek() + " с StackC на St
ackA")
        } else if (i % 3 === 2) {
            let y = transferDisk(s, a);
            if (y === 1) {
                console.log("Переместить диск " + a.peek() + " с StackA на St
ackB")
            } else
                console.log("Переместить диск " + s.peek() + " с StackB на St
ackA")
        } else {
            let y = transferDisk(a, d);
            if (y === 1) {
                console.log("Переместить диск " + d.peek() + " с StackB на St
ackC")
            } else
                console.log("Переместить диск " + a.peek() + " с StackC на St
ackB")
        }
        i++;
    }
}
return 0;
}

console.log(han(3))

```

```

const Stack = require(`./stack`)

function bracketFinderStack (array){
    let stack = new Stack.Stack()
    let flag = true;
    array.map( item =>{
        if(item === '('){
            stack.push('(')
        }
        else if(item === ')'){

```



```

        if(!stack.isEmpty()){
            stack.pop()
        }
        else {
            flag = false
        }
    }
})
return flag && !!stack.isEmpty()
}

console.log(bracketFinderStack ( ['(', '(', '(', ')', ')', ')'] ))

```

```

const DeQueue = require('./queue')
function bracketFinderDeque(array){
    let deque = new DeQueue.DeQueue()
    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();
}

console.log(bracketFinderStack ( ['[', ']', '[', ']', '[', ']' ]))

```

```

const Stack = require('./stack')
function regexParse (string){
    let array = string.split("")
    let numbers = new Stack.Stack()
    let letters = new Stack.Stack()
    let other = new Stack.Stack()

    array.map(item => {
        if (item.match(/[0-9]/)){
            numbers.push(item)
        }
        else if (item.match(/[a-zA-Z]/)){
            letters.push(item)
        }
        else{
            other.push(item)
        }
    })
}

```

```

    }
  })

  let numbersRevers = new Stack.Stack()
  let lettersRevers = new Stack.Stack()
  let otherRevers = new Stack.Stack()

  while (!numbers.isEmpty()) {
    numbersRevers.push(numbers.pop())
  }
  while (!letters.isEmpty()) {
    lettersRevers.push(letters.pop())
  }
  while (!other.isEmpty()) {
    otherRevers.push(other.pop())
  }

  while (!numbersRevers.isEmpty()) {
    console.log(numbersRevers.pop());
  }
  while (!lettersRevers.isEmpty()) {
    console.log(lettersRevers.pop());
  }
  while (!otherRevers.isEmpty()) {
    console.log(otherRevers.pop());
  }
}

console.log(regexParse('ax31ca65s-a*s'))

```

```

const DeQueue = require('./queue')
function numbersParse (array) {
  let deque = new DeQueue.DeQueue();
  array.map(item =>{
    if(item < 0){
      deque.pushBack(item)
    }
    else{
      deque.pushFront(item)
    }
  })
  while (!deque.isEmpty()){
    console.log(deque.popBack())
  }
}

console.log(numbersParse([1,6,5,2,4,0,-3,-1,-2]))

```

```

const Stack = require('./stack')

```

```
function stringRevers (string){
  let array = string.split(' ')
  let stack = new Stack.Stack()
  array.map(string =>{
    stack.push(string);
  })
  while(!stack.isEmpty()){
    console.log(stack.pop());
  }
}

console.log(stringRevers('hello my friend'))
```

```
const Stack = require('./stack')

function computeLogic1 (Str){
  let str1="";
  let stk= new Stack.Stack();
  for(let i=0;i<Str.length;i++){
    stk.push(Str[i])
  }
  for(let i=0;i<Str.length;i++){
    if(stk.peek()=="T")
      str1+="true "
    if(stk.peek()=="F")
      str1+="false "
    if(stk.peek()=="N")
      str1+="! "
    if(stk.peek()=="A" || stk.peek()=="*")
      str1+="&& "
    if(stk.peek()=="X")
      str1+="!= "
    if(stk.peek()=="O" || stk.peek()=="+")
      str1+="|| "
    if(stk.peek()=="(")
      str1+="( "
    if(stk.peek()==")")
      str1+=")"
    stk.pop()
  }

  console.log(eval(str1))
}

console.log(computeLogic1('F+T'))
```

```
const Stack = require('./stack')
function computeMinMax(Str) {
```

```

let str1=""
let stk= new Stack.Stack()
for(let i=0;i<Str.length;i++){
    stk.push(Str[i])
}
for(let i=0;i<Str.length;i++){
    if(stk.peek()=== "0")
        str1="0" +str1
    if(stk.peek()=== "1")
        str1="1" +str1
    if(stk.peek()=== "2")
        str1="2" +str1
    if(stk.peek()=== "3")
        str1="3" +str1
    if(stk.peek()=== "4")
        str1="4" +str1
    if(stk.peek()=== "5")
        str1="5" +str1
    if(stk.peek()=== "6")
        str1="6" +str1
    if(stk.peek()=== "7")
        str1="7" +str1
    if(stk.peek()=== "8")
        str1="8" +str1
    if(stk.peek()=== "9")
        str1="9" +str1
    if(stk.peek()=== "M")
        str1="Math.max" +str1
    if(stk.peek()=== "N")
        str1="Math.min" +str1
    if(stk.peek()=== "," || stk.peek()=== ".")
        str1="," +str1
    if(stk.peek()=== "(")
        str1="(" +str1
    if(stk.peek()=== ")")
        str1=")" +str1
    stk.pop()
}
console.log(eval(str1))
}

```

```

const Stack = require('./stack');

function computeForm (Str)
{
    let stk= new Stack.Stack()
    let str=""
    for(let i=0;i<Str.length;i++){
        stk.push(Str[i])
    }
}

```

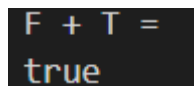
```

for (let i=0;i<Str.length;i++){
    str=stk.pop()+str
}
try{
    eval(str)
}
catch (err){
    console.log(false)
}
console.log(true)
}

```

Вывод

На рисунке 1 представлен результат работы программы задания 9



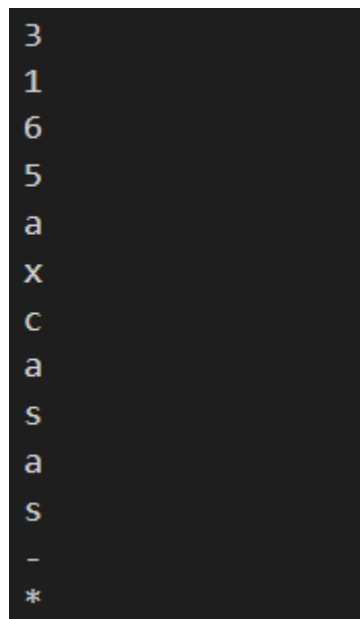
```

F + T =
true

```

Рисунок 1 Результат работы программы проверки выражения на скобки

На рисунке 2 представлен результат работы программы задания 6



```

3165axcsas-*
3165axcsas-*

```

Рисунок 2 Результат работы программы вывода символов строки в определённом порядке

Вывод: в ходе выполнения данной работы мною были получены знания о разнице и структуре типов данных таких как стэк и дэка, реализованы эти структуры, выполнены задачи на работы со структурами.