МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №3
по дисциплине «Алгоритмы и структуры данных»
Тема: Стек и очередь

Студент гр. 8382 _____ Терехов А.Е.

Преподаватель _____ Балтрашевич В.Э.

Санкт-Петербург
2019

Цель работы.

Изучить структуру стека и очереди и методы работы с ними.

Задание.

За один просмотр заданного файла F, имеющего следующую структуру: a1 b1 a2 b2 a3 b3 … an bn

Здесь a1, a2, …, an состоят из символов A, B, C, D, …, Z, a b1, b2, …, bn состоят из символов a, b, c, d …, z.

Без использования дополнительных файлов вывести элементы файла F порядке: a1` b1` a2` b2` a3` b3` … an` bn`, где b1`, строка обратная для строки b1, bn`, строка обратная для строки bn, a a1`, строка обратная для строки a1, an`, строка обратная для строки an.

В решении задачи использовать очередь и/или стек.

Ход работы.

В ходе лабораторной работы был реализован стек на базе однонаправленного списка. Элементы списка имеют тип `char`. Базовые функции для работы с ним: `Stack()` – конструктор, `~Stack()` – деструктор, `push(node*)` – положить на стек, `top()` – показать вершину стека, функция необходима, так как вершина стека приватна, `pop()` – снять элемент с вершины.

Также была написана очередь на базе однонаправленного списка. Элементы списка имеют тип `char`. Базовые функции для работы с ним: `Queue()` – конструктор, `~Queue()` – деструктор, `push(node*)` – вставить в конец очереди, `front()` – показать первый элемент, `pop()` – извлечь первый элемент из очереди.

Задание выполняется по следующему алгоритму. Сначала в процессе чтения файла заполняется очередь. Затем очередь перекладывается в стек пока не встретится пробел, как только встречается пробел, стек очищается в конец выходной строки. Последние действия повторяются до тех пор, пока очередь не опустеет.

Тестирование.

input	***output***
фыва ЙЦУК ячсм	авыф КУЦЙ мсчя
фыва ЙЦУК яВчсм	Некорректная строка
ф Ы а Ф	ф Ы а Ф
PSGHPJSAANBTWYST jzzjnxxtixswpfxxoe RWSGRGDMEDTNNBEC zbatudjdmdphunbv DEZSGNBJ bhcpaymdxjse VWGMSRMHZJVNTVPQO rmkzywukqlnl JXPI patrwdwg VMHXRFLBSHHFDPCEUMQ aewu ADXICSBFMU eugmtmnkay OEEF tkyiwy ODFGBIJDDOHYH r	TSYWTBNAASJPHGSP eoxxfpwsxitxxnjzzj CEBNNTDEMDGRGSWR vbnuhpdmdjdutabz JBNGSZED esjxdmyapchb OQPVTNVJZHMRSMGWV lnlqkuwyzkmr IPXJ gwdwrtap QMUECPDFHHSBLFRXHMV uwea UMFBSCIXDA yaknmtmgue FEEO ywiykt HYHODDJIBGFDO r

Выводы.

Была успешно реализована программа с графическим интерфейсом, выполняющая разворот всех слов в файле, записанных по определенному правилу. Были изучены структуры стека и очереди. И также были закреплены навыки в написании функций для работы с ними и графических приложений в целом. В работе был использован фреймворк Qt.

ПРИЛОЖЕНИЕ А.

stack.h:

```
typedef QChar base;
class Stack
{
private:
    struct Node
    {
        base d;
        Node *next;
    };
    Node *tp;

public:
    Stack();
    ~Stack();
    base top();
    void push(base);
    bool pop();
    bool isEmpty();
};
```

stack.cpp:

```
#include "stack.h"

Stack::Stack() : tp(0) {}

bool Stack::isEmpty()
{
    return tp ? false : true;
}

Stack::~Stack()
{
    while (pop())
        ;
}

base Stack::top()
{
```

```
    return isEmpty() ? 0 : tp->d;
}

void Stack::push(base s)
{
    Node* n = new Node;
    n->d = s;
    n->next = tp;
    tp = n;
}

bool Stack::pop()
{
    if (isEmpty())
        return false;
    Node *n = tp;
    tp = tp->next;
    delete n;
    return true;
}
```

queue.h:

```
typedef QChar base_queue;
class Queue
{
private:
    struct Node
    {
        base_queue d;
        Node *next;
    };
    Node *first;
    Node *last;

public:
    Queue();
    ~Queue();
    base_queue front();
    void push(base_queue);
    bool pop();
    bool isEmpty();};
```

queue.cpp:

```
#include "queue.h"

Queue::Queue() : first(nullptr) {}

void Queue::push(base_queue x)
{
    Node *n = new Node;
    n->d = x;
    n->next = nullptr;
    if (first == nullptr)
    {
        first = n;
    }
    else
    {
        Node *temp = first;
        while (temp->next != nullptr)
            temp = temp->next;
        temp->next = n;
    }
}

bool Queue::pop()
{
    if (isEmpty()){
        first = nullptr;
        return false;
    }
    Node *n = first;
    first = first->next;
    delete n;
    return true;
}

base_queue Queue::front()
{
    if (isEmpty())
        return '\0';
    return first->d;
}
```

```
bool Queue::isEmpty(){
    return first ? false : true;
}


Queue::~Queue()
{
    while(pop());
}
```