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: Линейные структуры данных: стек, очередь, дек

Студент гр. 7383 _____ Александров Р.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург

2018

Цель работы.

Познакомиться с часто используемыми на практике линейными структурами данных, обеспечивающими доступ к элементам последовательности только через её начало и конец, и способами реализации этих структур, освоить на практике использование стека, очереди и дека для решения задач.

Постановка задачи.

За один просмотр заданного файла F (типа file of Real) и без использования дополнительных файлов вывести элементы файла F в следующем порядке: сначала - все числа, меньшие a, затем - все числа на отрезке $[a, b]$ и наконец - все остальные числа, сохраняя исходный взаимный порядок в каждой из этих групп чисел (a и b задаются пользователем, $a < b$).

Реализация задачи.

Для решения поставленной задачи в работе были использованы 3 класса: Main, Queue, Actions.

В классе Main определяется функция void fileRead() для считывания данных из файла.

Пользователю предлагается либо ввести значения a и b и указать текстовый файл, в котором они находятся.

В классе Queue определяются очередь на базе вектора и функции взаимодействия с ней:

- Queue(int newSize) – конструктор, создает новый массив с указанным размером;
- void enqueue(Item item) добавляет элемент в очередь;
- void dequeue() удаляет первый элемент очереди;
- Item getFront() возвращает первый элемент очереди;
- bool isFull() проверяет, не заполнен ли массив;
- bool isEmpty() проверяет, не пуст ли массив;

- Item *resize(Item *prev) создает новый массив, удваивая текущий размер;
- ~Queue() – деструктор.

В классе Actions определяются следующие функции:

- void readFile(string fileName, int a, int b) создает 3 очереди, считывает из файла числа, заполняя очереди в соответствии с условиями;
- void printResult(Queue<Item> &queue) выводит в консоль принятую в качестве аргумента очередь.

Тестирование программы.

Программа собрана и проверена в операционных системах Xubuntu 18.04 с использованием компилятора g++ и Windows с использованием MinGW. В других ОС и компиляторах тестирование не проводилось. Тесты находятся в приложении А.

Вывод.

В ходе лабораторной работы были получены основные навыки программирования линейной структуры – очереди – на языке C++. Результатом стала программа, которая использует очередь на базе массива для вывода чисел из файла по заданному условию.

ПРИЛОЖЕНИЕ А
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ

Таблица 1 – Тестирование программы

Input	Output
A = 110, b = 1500 965 524 123 557 111 125 57 58 63 22 10 111111 12212 5161	57 58 63 22 10 965 524 123 557 111 125 111111 12212 5161
A = 100, b = 200 89 56 20 11 0 4 2 11 211 365 78 966 20 100 -6877 6454 444554 21121 1213 21122 23323 6565 45545 1221 484 5445 212121 484848 656 332 21122 23323 6565 45545 1221 484 5445 212121 484848 656 332 250 21122 23323 6565 45545 1221 484 5445 212121 484848 656 332	89 56 20 11 0 4 2 11 78 20 -6877 100 211 365 966 6454 444554 21121 1213 21122 23323 6565 45545 1221 484 5445 212121 4848 48 656 332 21122 23323 6565 45545 1221 484 5445 212121 484848 656 332 250 21122 23323 6565 45545 1221 484 5445 212121 48 4848 656 332

ПРИЛОЖЕНИЕ Б
КОД ПРОГРАММЫ

main.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include "main.h"

using namespace std;

void Main::menu() {
    cout << "1. Enter the name of the text file" << endl;
    cout << "0. Exit" << endl;
}

void Main::fileRead() {
    string fileName;
    int a;
    int b;
    cout << "Enter a" << endl;
    cin >> a;
    cout << "Enter b" << endl;
    cin >> b;
    if (a >= b) {
        cout << "a must be less then b" << endl;
        return;
    }
    cout << "Enter the name of the text file" << endl;
    cin >> fileName;
    try {
        actions.readFile(fileName, a, b);
    } catch (string s) {
        cout << s << endl;
    }
}

int main() {
    Main main;
    while (true) {
        main.menu();
        cin >> main.choice;
        switch (main.choice) {
            case 1:
                main.fileRead();
                break;
            case 0:
                exit(1);
        }
```

```
    }
}
```

main.h

```
#pragma once
#include "actions.h"

class Main {
private:
    Actions actions;
public:
    Main() {}
    unsigned int choice;

    void fileRead();

    void menu();
};
```

actions.cpp

```
#include <iostream>
#include <string>
#include <fstream>
#include "actions.h"
using namespace std;

void Actions::readFile(string fileName, int a, int b) {
    ifstream inFile;
    inFile.open(fileName);
    int value;
    if (!inFile) {
        throw string("Cannot find this file");
    }
    //
    int beginSize = 5;
    Queue<int> qLessA(beginSize);
    Queue<int> qMoreALessB(beginSize);
    Queue<int> qMoreB(beginSize);
    while (inFile >> value) {
        if (value < a) {
            qLessA.enqueue(value);
        }
        if (value >= a && value <= b) {
            qMoreALessB.enqueue(value);
        }
        if (value > b) {
            qMoreB.enqueue(value);
        }
    }
```

```
    inFile.close();

    cout << "-------- Numbers --------" << endl;
    printResult(qLessA);
    printResult(qMoreALessB);
    printResult(qMoreB);
    cout << endl;
    cout << "--------------" << endl;
}

template<class Item>
void Actions::printResult(Queue<Item> &queue) {
    while (!queue.isEmpty()) {
        cout << queue.getFront() << " ";
    }
}
```

actions.h

```
#pragma once
#include <string>
#include "queue.h"

using namespace std;

class Actions {
public:
    void readFile(string fileName, int a, int b);

private:
    template<class Item>
    void printResult(Queue<Item> &queue);
};
```

queue.h

```
#pragma once

#include <iostream>

using namespace std;

template<class Item>
class Queue {
private:
    Item *q;
    int N;
    int size;
public:
    Queue(int newSize);
```

```
    void enqueue(Item item);

    void dequeue();

    Item getFront();

    bool isFull();

    bool isEmpty();

    Item *resize(Item *prev);

    ~Queue();
};


template<class Item>
Queue<Item>::Queue(int newSize) {
    q = new Item[newSize];
    N = 0;
    size = newSize;
}

template<class Item>
void Queue<Item>::enqueue(Item item) {
    if (isFull()) {
        q = resize(q);
    }
    q[N] = item;
    N++;
}

template<class Item>
void Queue<Item>::dequeue() {
    if (isEmpty()) {
        cout << "Queue is empty" << endl;
        return;
    }
    for (int i = 0; i < size-1; i++) {
        q[i] = q[i+1];
    }
    this->size--;
    this->N--;
}

template<class Item>
Item Queue<Item>::getFront() {
    if (isEmpty()) {
        cout << "Queue is empty" << endl;
        return 0;
    }
```

```
    Item returnEl = q[0];
    dequeue();
    return returnEl;
}

template<class Item>
bool Queue<Item>::isEmpty() {
    return this->N == 0;
}

template<class Item>
Item *Queue<Item>::resize(Item *prev) {
    int newSize = size * 2;
    Item *temp = new Item[newSize];
    for (int i = 0; i < size; i++) {
        temp[i] = prev[i];
    }
    size = newSize;
    return temp;
}

template<class Item>
bool Queue<Item>::isFull() {
    return N == size;
}

template<class Item>
Queue<Item>::~Queue() {
    delete[] q;
}
```

Makefile

```
CXX=g++
RM=rm -f
LDFLAGS=-g -Wall

SRCS=main.cpp actions.cpp
OBJS=$(subst .cpp,.o,$(SRCS))

all: main

main: $(OBJS)
     $(CXX) $(LDFLAGS) -o main $(OBJS)

main.o: main.cpp main.h

actions.o: actions.cpp actions.h

queue.o: queue.h
```

```
clean:
	$(RM) $(OBJS)

distclean: clean
	$(RM) main
```