МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МОЭВМ

ОТЧЁТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: Иерархические списки

Студент гр.8304 _____ Воропаев А.О.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Решить полученную задачу, используя очередь, реализованную на базе списка. Получения навыков работы с нелинейными структурами данных.

Задание.

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).

Описание алгоритма.

Сначала программа принимает числа ***a*** и ***b***. Затем создаются три очереди для трёх разных случаев, описанных в условии задачи. Затем программа принимает строку, из которой вычленяются числа и помещаются в соответствующую очередь согласно условию. Затем последовательно выводятся значения всех очередей.

Описание функций и структур данных программы:

Класс *queue* содержит следующую структуру:

```
typedef struct Element{
    int value;
    struct Element* next;
    Element(int v = 0){
        value = v;
        next = nullptr;
    }
}el;
```

Данная структура содержит описание элемента очереди.

value – поле, содержащее числа, помещённого в очередь.

Next – указатель на следующий элемент очереди

Element(int v = 0) – конструктор.

Конструктор класса:

```
queue::queue(){
    head = nullptr;
```

```
    tail = nullptr;
}
```

Конструктор создаёт пустую очередь. И задаёт полям tail и head значение nullptr.

Методы класса:

```
void push(int v);
```

Метод предназначен для помещения нового элемента в очередь.

V – значение, которое будет помещено в поле value структуры, которая является элементом очереди.

```
el pop();
```

Данный метод возвращает элемент, который содержит голова очереди, и удаляет данный элемент из очереди.

```
int getSize()
```

Метод, возвращающий значение, равное кол-ву элементов очереди.

Выводы.

Для решения данной задачи было логично использовать очереди.

Тестирование:

```
a Value:
228
b Value:
1488
wefrg123faswr4525frg@456gregre544fargrg99343gsregsr344
Digits from the first queue(lesser than A)
123
Digits from the second queue(greater than A and lesser than B)
544 456
Digits from the third queue(greater than B)
99343 4525
```

Входные данные	Выходные данные
228 1488 wefrg123faswr4525frg@456gregre544fargrg99343gsregsr344	Digits from the first queue(lesser than A) 123

	Digits from the second queue(greater than A and lesser than B) 544 456 Digits from the third queue(greater than B) 99343 4525
1488 228 wefrg123faswr4525frg@456gregre544fargrg99343gsregsr344	a-value must be lesser than b-value
228 1488 wefrg123faswr452 5frg@456gregre544fargrg99 343gsregsr344	Digits from the first queue(lesser than A) 123 Digits from the second queue(greater than A and lesser than B) 544 456 Digits from the third queue(greater than B) 99343 4525
234454666557 654 99939dadasd123dawdaw455eththsthre77377	a-value is too large for INT to contain it!!! a-value: 234454666557

Исходный код

Main.cpp

```
#include "queue.h"
#include <iostream>
#include <cstdlib>
#include <fstream>
```

```
int main(int argc, char* argv[]){

    int a,b;

    std::string processed_string;

    queue less_than_a;
    queue between_a_and_b;
    queue greater_than_b;

    std::string val_str;

    std::ifstream input_file_stream(argv[1]);

    if(!input_file_stream){
        std::cout << "File can't be open" << std::endl;
        return 1;
    }

    std::getline(input_file_stream, val_str);

    try {
        a = std::stoi(val_str);
    }
    catch(std::invalid_argument& exception){
        std::cout << "Invalid argument of the a-value:" << val_str <<
std::endl;
        return 1;
    }
    catch(std::out_of_range& exception){
        std::cout << "a-value is too large for INT to contain it!!!\na-value:
" << val_str << std::endl;
        return 1;
    }


    std::cout << "a-value: " << std::endl;
    std::cout << a << ' ' << std::endl;



    std::getline(input_file_stream, val_str);

    try {
        b = std::stoi(val_str);
    }
    catch(std::invalid_argument& exception){
        std::cout << "Invalid argument of the b-value: " << val_str <<
std::endl;
        return 1;
    }
    catch(std::out_of_range& exception){
        std::cout << "b-value is too large for INT to contain it!!!\nb-value:
" << val_str << std::endl;
        return 1;
    }

    std::cout << "b-value: " << std::endl;
    std::cout << b << ' ' << std::endl;



    if(a >= b){
```

```
        std::cout << "a-value must be lesser than b-value" << std::endl;
        return 1;
    }


    std::getline(input_file_stream, processed_string);

    std::cout << "Processed string:\n" << processed_string <<
"\n_____" << std::endl;

    std::string currently_processed_value_str;
    int counter = 0;


    for(auto proc_symbol:processed_string){

        counter++;

        if(isdigit(proc_symbol)){
            currently_processed_value_str += proc_symbol;
            if(counter != processed_string.size())
                continue;
        }

        if(!currently_processed_value_str.empty()) {

            int value;


            try {
                value = std::stoi(currently_processed_value_str);
            }
            catch(std::invalid_argument& exception){
                std::cout << "Invalid argument of the found digit:" <<
currently_processed_value_str << std::endl;
                return 1;
            }
            catch(std::out_of_range& exception){
                std::cout << "Found digit is too large for INT to contain
it!!!\nFound digit: " << currently_processed_value_str << std::endl;
                return 1;
            }


            std::cout << "Current found digit: " << value << std::endl;

            if (value < a)
                less_than_a.push(value);
            else if (value >= a && value <= b)
                between_a_and_b.push(value);
            else if (value > b)
                greater_than_b.push(value);

            currently_processed_value_str = "";
        }

    }
    std::cout << "\n_____\n";


    queue::el popped_value;
```

```
    if (!less_than_a.getSize())
        std::cout << "No numbers in the first queue(lesser than A)" <<
std::endl;

    else if(less_than_a.getSize()){
        std::cout << "Digits from the first queue(lesser than A)" <<
std::endl;

        while (less_than_a.getSize()) {
            popped_value = less_than_a.pop();
            std::cout << popped_value.value << ' ';
        }
    }
    std::cout << "\n_____\n";


    if (!between_a_and_b.getSize())
        std::cout << "No numbers in the second queue(Greater than a and
lesser than b)" << std::endl;

    else {
        std::cout << std::endl << "Digits from the second queue(greater than
A and lesser than B)" << std::endl;

        while (between_a_and_b.getSize()) {
            popped_value = between_a_and_b.pop();
            std::cout << popped_value.value << ' ';
        }
    }
    std::cout << "\n_____\n";


    if (!greater_than_b.getSize())
        std::cout << "No numbers in the third queue(greater than B)" <<
std::endl;

    else {
        std::cout << std::endl << "Digits from the third queue(greater than
B)" << std::endl;

        while (greater_than_b.getSize()) {
            popped_value = greater_than_b.pop();
            std::cout << popped_value.value << ' ';

        }
    }
    std::cout << "\n_____\n";


    return 0;
}
```

queue.cpp

```
#include "queue.h"
#include <iostream>


    queue::queue(){
    tail = nullptr;
    head = nullptr;
```

```
}


void queue::push(int v){

    el* n = new el(v);

    if(tail == nullptr){
        tail = n;
        head = tail;
    }

    else {
        head->next = n;
        head = head->next;
        tail->next = head;
    }

    size++;
}

queue::el queue::pop(){
    if (head == tail){
        el tmp = *tail;

        delete(tail);

        tail = nullptr;
        head = tail;


        size--;

        return tmp;
    }
    else{
        el tmp = *head;
        el* tmp_tail = head;
        head = tail;
        while (head->next != tmp_tail) {
            head = head->next;
        }
        delete(tmp_tail);
        head->next = nullptr;
        size--;
        return tmp;
    }
}

int queue::getSize() {
    return size;
}
```

queue.h

```
class queue {
public:


    typedef struct Element{
        int value;
        struct Element* next;
```

```
        explicit Element(int v = 0){
            value = v;
            next = nullptr;
        }
    }el;

    el* tail;
    el* head;

    queue();

    void push(int v);

    el pop();

    int getSize();

private:
    int size = 0;

};
```