МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №3

по дисциплине «Алгоритмы и структуры данных»

Тема: «Стеки и очереди»

Студентка гр. 7381 _____ Алясова А.Н.

Преподаватель _____ Фирсов М.А

Санкт-Петербург

2018

Цель работы.

Познакомиться со структурой и реализацией очереди и использованием ее в практических задачах на языке программирования C++.

Задание.

2 – в

Формулировка задания.

Содержимое заданного текстового файла *F*, разделенного на строки, переписать в текстовый файл *G*, перенося при этом в конец каждой строки все входящие в нее цифры (с сохранением исходного взаимного порядка как среди цифр, так и среди остальных литер строки).

Описание алгоритма.

На вход подается строка, мы ее обрабатываем следующим образом: берем первый элемент, создаем два массива numbers и other. Цифры записываем в массив numbers, другие различные символы в массив other. Затем в выходной файл выписываем все содержимое массива other, потом все содержимое массива numbers.

Описание функций и структур данных.

Переменные, используемые в функции *main*:

• *ifile, ofile* — файловые буфера, участвующие в создании объектов типа *istream* и *ostream* соответственно.

• *ifilename, ofilename* — входной и выходной файлы соответственно.

• *fin, fout* — входной и выходной потоки.

Функция *rewrite()* принимает на вход входной и выходной потоки. Создаются две очереди для цифр и других символов. Пока входной файл не закончится, оттуда считываются строки. Каждая строка обрабатывается посимвольно: цифры добавляются в одну очередь, остальные символы — в другую. Когда строка закончилась, символы из очередей записываются в выходной файл, причем символы из очереди цифр записываются последними.

Переменные, используемые в функции *rewrite*:

- *numbers, other* — очереди цифр и других символов соответственно.
- *input* — обрабатываемая строка.

Создается шаблонный класс *Queue* с шаблонным параметром *T* (тип хранимых элементов), представляет из себя циклическую очередь на базе динамического массива.

Класс содержит следующие поля:

- *vsize* — длина динамического массива (вектора).
- *qsize* — длина очереди.
- *vstart* — указатель на начало массива.
- *qstart* — указатель на начало очереди.
- *qend* — указатель на конец очереди.

Методы класса *Queue*:

1) *Queue(unsigned int new_size) : vsize(size), qsize(0);*

Конструктор, принимающий длину массива. Выделяется память под массив, все указатели устанавливаются на начало массива.

2) void push(T value);

Принимает объект типа *T*, который добавляется в конец очереди по указателю *qend*, *qend* сдвигается вправо с учетом цикличности очереди. Если очередь переполнена, то сначала увеличивается в два раза размер массива, а после добавляется элемент.

3) void resize(insigned int new_size);

Принимает новый размер массива. Создается новый массив и копируется в него содержимое старого массива, после чего память под старый массив высвобождается.

4) bool isEmpty();

Возвращает *true*, если очередь пуста.

5) T pop();

Возвращает первый элемент из очереди по указателю *qstart*, он сдвигается вправо с учетом цикличности очереди.

6) ~Queue();

Деструктор.

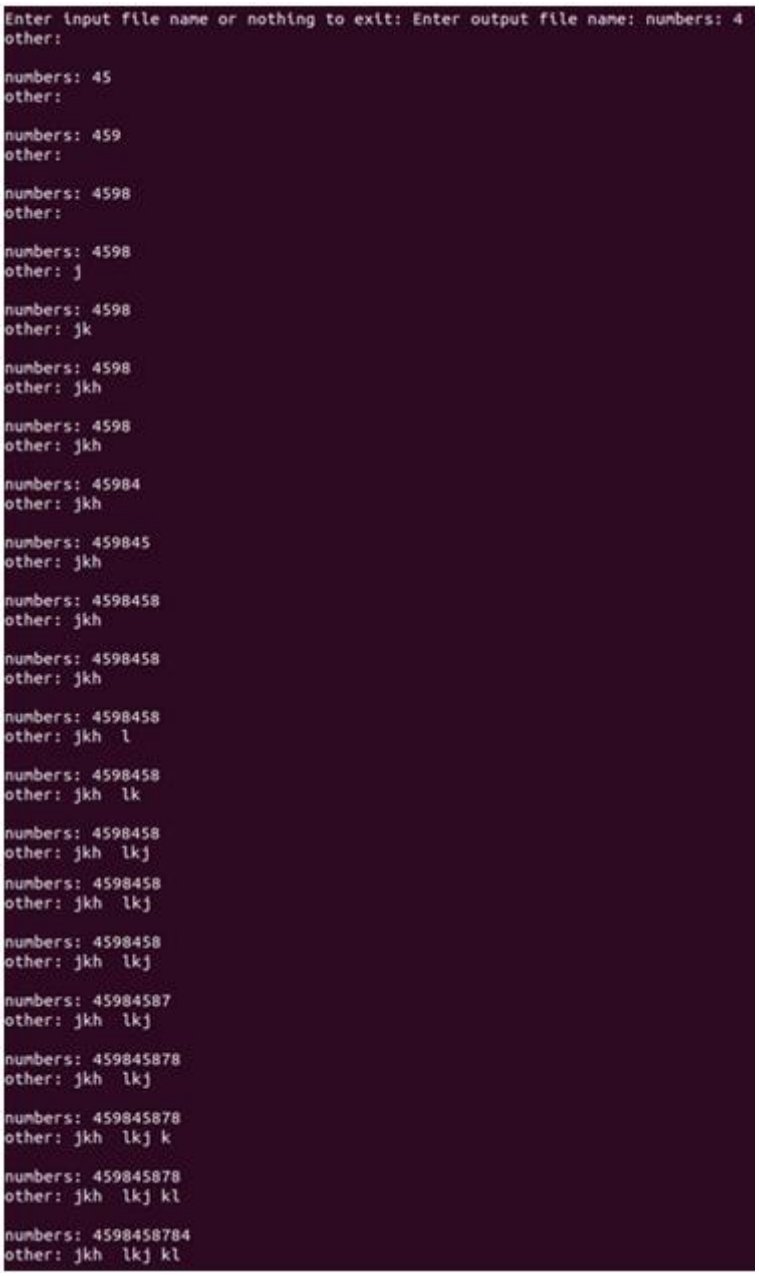
Тестирование.

Для проверки работоспособности программы был создан скрипт, текст которого представлен а Приложении Б для автоматического ввода и вывода тестовых данных.

Таблица 1 - Тестирование

Входные данные	Выходные данные
4598jkh 458 lkj 78kl4	jkh lkj kl4598458784
gshgj 321466gtdvcuysjh63 dhcbj65 5654hdsj 452 bdhjh	gshgj gtdvcuysjh dhcbj hdsj bdhjh32146663655654452
215325 gsuhjk gsahijkn;'d453jk	215325 gsuhjk gsahijkn;'djk453
((8465132)(053)()4578(46))(7_) gj5	(()()()())(_) gj846513205345784675
ASDF4589 BNB782 4578M fghh **1*4	ASDF BNB M fghh ***4589782457814

Подробный пример тестирования:



Выводы.

В процессе выполнения лабораторной работы были получены навыки работы с очередью. Был реализован шаблонный класс, представляющий из себя циклическую очередь на базе массива. Были закреплены навыки работы с системой контроля версий и bash-скриптами. Систематизированы навыки полуавтоматического тестирования программ.

ПРИЛОЖЕНИЕ А

main.cpp

```
#include <iostream>

#include <fstream>

#include <string>

#include "queue.hpp"


void rewrite(std::istream &fin, std::ostream &fout){//функция для вывода процесса работы программы

     Queue<unsigned char> numbers(4), other(10); //устанавливаем размеры элементов

     std::string input;

     while(!fin.eof()){ //пока fin()

          getline(fin, input);

          for(auto i : input){ //Ключевое слово auto указывает компилятору использовать выражение инициализации объявленной переменной, или параметр лямбда-выражения, чтобы вывести ее тип.

               if(i > '9' || i < '0')

                    other.push(i); //push in other

               else

                    numbers.push(i); //push in numbers

               unsigned char c;

               std::cout << "numbers: ";

               for(int j=0; j<numbers.get_size(); ++j){ //вывод содержимого numbers

                    c = numbers.pop();

                    std::cout << c;

                    numbers.push(c);

               }
```

```
                std::cout << std::endl;
                std::cout << "other: ";
                for(int   j=0;   j<other.get_size();   ++j){   //вывод
содержимого other
                     c = other.pop();
                     std::cout << c;
                     other.push(c);
                }
                std::cout << std::endl << std::endl;
           }
           while(!other.isEmpty())
                fout << other.pop();//запись вначале всех символов
           while(!numbers.isEmpty())
                fout << numbers.pop();//запись всех цифр
           fout << std::endl;
     }
}


int main(){
     std::filebuf in_file, out_file; //filebuf - Этот тип является
синонимом класса шаблона basic_filebuf, специализированного для
элементов типа char с признаками символа по умолчанию.
     std::string in_filename, out_filename; //string - это STL’евский
класс основанный на шаблонах(аналогичен char*)
     while(true){
          std::cout << "Enter input file name or nothing to exit: ";
//ввод названия входного файла
          getline(std::cin, in_filename);
          if(in_filename   ==   "")   //если   не   нужно   больше   читать
содержимое файла
```

```
                break;
          if(!in_file.open(in_filename,  std::ios::in)){//если  нельзя открыть файл для чтения; //std::ios::in - открыть файл для чтения
                std::cout << "Incorrect file, try again" << std::endl;
                in_file.close();
                continue;
          }
          std::cout  <<  "Enter  output  file  name:  ";//ввод  имени выходного файла
          getline(std::cin, out_filename);
          if(!out_file.open(out_filename,   std::ios::out)){   //если нельзя открыть файл для записи
                std::cout << "Incorrect file, try again" << std::endl;
                in_file.close();
                out_file.close();
                continue;
          }
          std::istream fin(&in_file);
          std::ostream fout(&out_file);
          rewrite(fin, fout);
          in_file.close();
          out_file.close();
     }
     return 0;
}
```

queue.hpp

```
template <class T>// исп для описания класса queue как шаблона, чтобы хранит в нем данные любого типа; T-параметр шабона класса
class Queue{ //создание класса
```

```
public:
	Queue(unsigned int); //конструктор
	unsigned int get_size();//функция для получения длины очереди
	void push(T); //добавление элемента в очередь
	void resize(unsigned int); //увеличение размера массива(вектора)
	bool isEmpty();//проверка на пустоту очереди
	T pop(); //удаление элемента из очереди
	~Queue(); //деструктор
private:
	unsigned int vsize; //размер вектора
	unsigned int qsize; //размер очереди
	T* vstart; //указатель на начало вектора
	T* qstart; //указатель на начало очереди
	T* qend; // указатель на конец очереди
};


template <class T>
Queue<T>::Queue(unsigned int size) : vsize(size), qsize(0) {
//конструктор
	vstart = new T[size];
	qstart = vstart;
	qend = vstart;
}


template <class T>
	unsigned int Queue<T>::get_size(){ //получение размера очереди
	return qsize;
}
```

```
template <class T>
	void Queue<T>::push(T value) { //value = symbol
	if(qend != qstart || (qsize == 0 && vsize != 0))
		*qend = value;
	else {
		resize(vsize*2);
		*qend = value;
	}
	if(qend < (vstart + vsize - 1))
		++qend;
	else
		qend = vstart;
	++qsize;
}


template <class T>
void Queue<T>::resize(unsigned int new_size) { // увеличение
вектора,создается новый массив и в негокопируется старый массив
	T* new_vect = new T[new_size];
	int i;
	for(i=0; i<(vsize-(qstart-vstart)); ++i)
		new_vect[i] = qstart[i];
	for(int j=0; i < vsize; ++i, ++j)
		new_vect[i] = vstart[j];
	qstart = new_vect;
	qend = new_vect + qsize;
	delete vstart;
```

```
	vstart = new_vect;
	vsize = new_size;
}

template <class T> //проверка на пустоту очереди
bool Queue<T>::isEmpty() {
	return qsize == 0;
}

template <class T>
T Queue<T>::pop() { //функция возвращает первый элемент и сдвигает очередь вправо
	T ret = *qstart;
	if(qstart != (vstart + vsize - 1))
		++qstart;
	else
		qstart = vstart;
	--qsize;
	return ret;
}

template <class T>
Queue<T>::~Queue() { //деструктор
	delete vstart;
}
```

ПРИЛОЖЕНИЕ Б

Exec.sh

```
g++ ./Sourse/main.cpp -o Lab3
echo -e '_______\nTest 1:'
cat ./Tests/Test1.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test1.txt
echo -e ''
echo -e '_______\nTest 2:'
cat ./Tests/Test2.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test2.txt
echo -e ''
echo -e '_______\nTest 3:'
cat ./Tests/Test3.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test3.txt
echo -e ''
echo -e '_______\nTest 4:'
cat ./Tests/Test4.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test4.txt
echo -e ''
echo -e '_______\nTest 5:'
cat ./Tests/Test5.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test5.txt
```

```
echo -e ''
echo -e '_______\nTest 6:'
cat ./Tests/Test6.txt
echo -e '_______\nTesting:\n'
./Lab3 < ./Tests/Test6.txt
```