МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Липецкий Государственный Технический Университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Лабораторная работа

по программированию №3

“ Обработка базовых списковых структур данных”

Студент \_\_\_\_\_\_\_\_\_\_\_\_ Станиславчук С. М.

(подпись, дата)

Группа АС-21-1

Руководитель

Доцент, кандидат наук \_\_\_\_\_\_\_\_\_\_\_\_ Муравейко А. Ю.

(подпись, дата)

Липецк 2022 г.

Содержание отчета

1. Титульный лист

2. Цель работы, задание

3. Описание алгоритма

4. Код программы

5. Примеры выполнения

6. Выводы

2. Цель работы: изучение основных структур данных и приобретение навыков их реализации с помощью массивов и указателей.

Вариант 13

Разработать программу, реализующую линейную структуру данных - очередь с использованием структуры данных в памяти:

1) С помощью статического массива,

2) с помощью динамического массива.

Программа должна предусматривать ввод/вывод из/в двоичного файла и с клавиатуры/на экран. Для хранения данных следует использовать заданную списковую структуру данных. Данные представляют собой структуру из элементов data: string, key: double. Ввод-вывод в двоичный файл осуществлять чтением/записью области памяти, занимаемой структурой. Предусмотреть хранение индекса (номера) каждого элемента структуры данных.

Программа должна предоставлять пользователю интерфейс, позволяющий выполнять:

а) выбор одного из двух указанных способов представления структуры данных в памяти;

б) выход из программы;

в) создание структуры данных «с нуля» - вначале структура данных не содержит ни одного элемента;

г) добавление данных из двоичного файла;

д) выгрузку данных в двоичный файл;

е) следующие операции над структурой данных:

1. Добавление элемента (если возможно: в начало, конец, произвольную заданную позицию).

2. Удаление элемента (если возможно: из начала, из конца, из произвольной заданной позиции).

3. Поиск элемента (если возможно).

4. Замена «старого» значения элемента на заданное новое (поиск по значению).

5. Установка значения элемента по заданному индексу (поиск по индексу).

6. Получение значения элемента по заданному индексу.

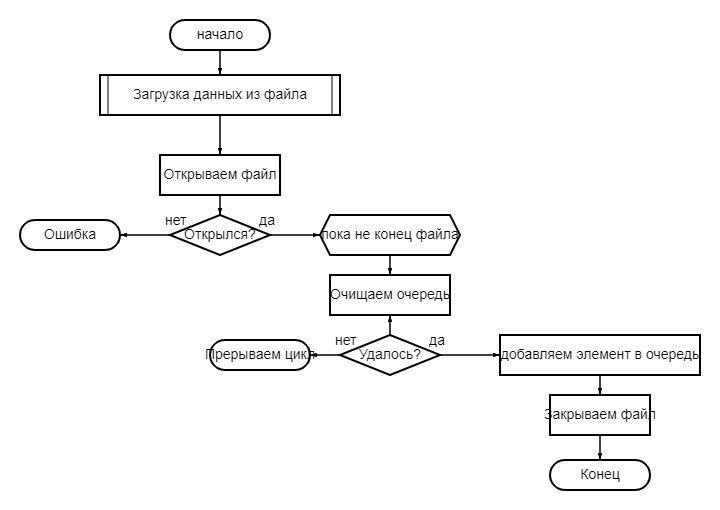
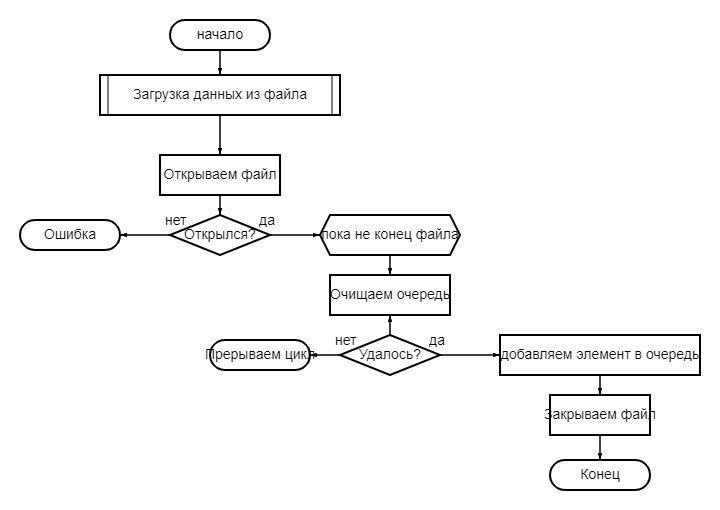
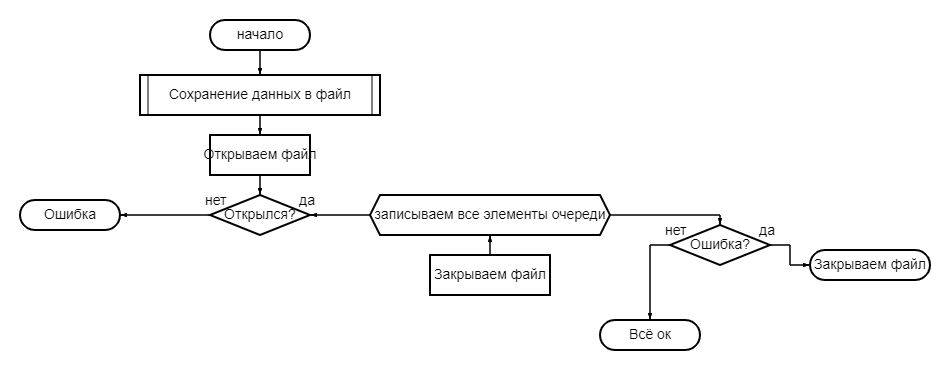
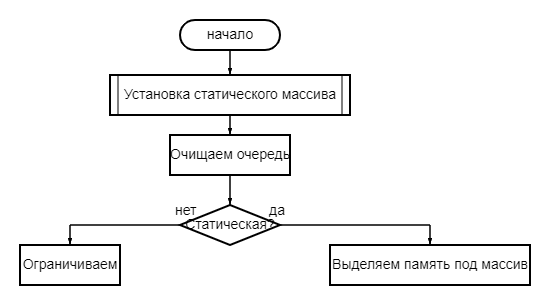
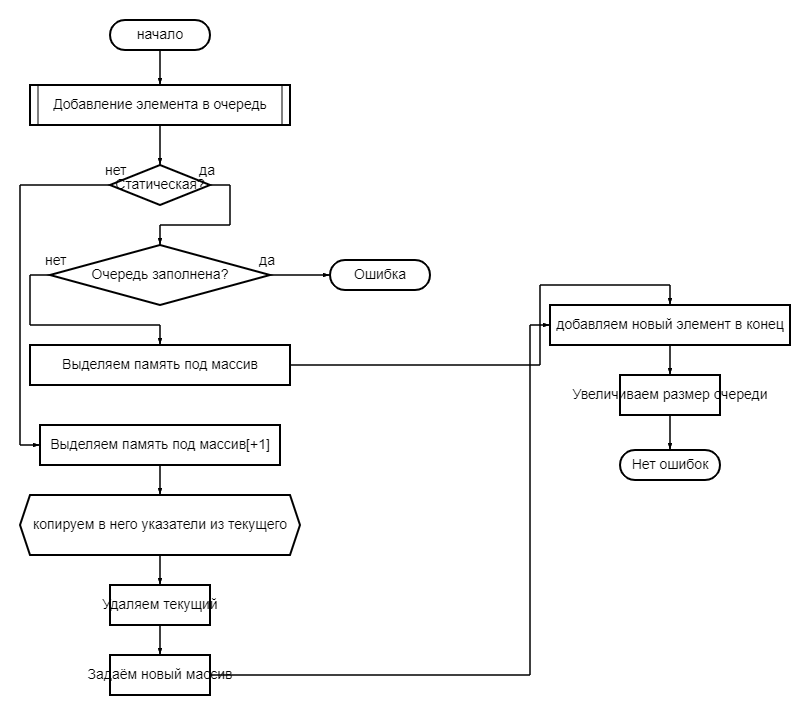
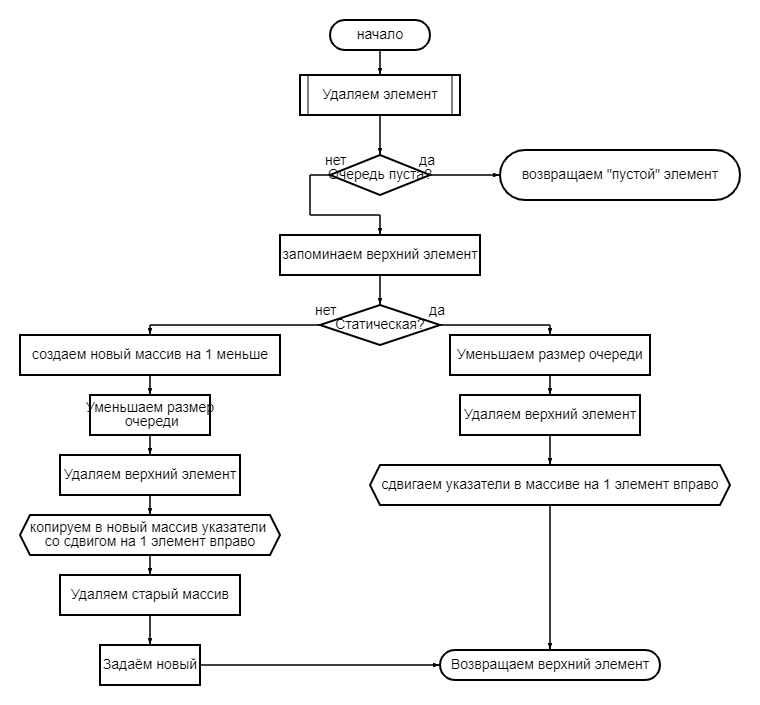
7. Удаление всех элементов.

8. Вывод на экран элементов структуры (для стеков, деков, очередей нужно

использовать дополнительные временные «хранилища», а не реализовывать не присущие данным структурам функции).

9. Структура пуста/не пуста?

К сгенерированной структуре данных должны быть применимы все остальные функции.

3. Описание алгоритма

**4. Код программы**

#include <iostream>

#include <fstream>

using namespace std;

struct queueItem {

double key = 0;

string value;

};

struct queue {

static const size\_t maxSizeUnlimited = 0;

//должен быть первым в структуре, не менять

size\_t maxSize = maxSizeUnlimited;

size\_t size = 0;

queueItem\*\* data = nullptr;

};

//используется статическая очередь (фиксированная) или динамическая

bool queueIsFixed(const queue\* q) {

return q->maxSize != queue::maxSizeUnlimited;

}

//очередь заполнена или нет

bool queueIsFull(const queue\* q) {

return q->size == q->maxSize;

}

//очередь пуста или нет

bool queueIsEmpty(const queue\* q) {

return q->size == 0;

}

//возвращает указатель (только чтение) на верхний (первый) элемент очереди

//не выталкивая его

queueItem\* queueTop(queue\* q) {

return queueIsEmpty(q) ? nullptr : q->data[0];

// <--- здесь не должно быть краха. Будет при q = nullptr или

// на месте вызова при попытке обратиться к top при пустой очереди

// не проверяя возврат на nullptr.

}

//сохраняет элемент в очереди

bool queuePush(queue\* q, queueItem\* item) {

//если статическая очередь

if (queueIsFixed(q)) {

//если полна

if (queueIsFull(q))

//возвращаем признак ошибки

return false;

//если фиксированная и не выделена память - выделить

if (!q->data) q->data = new queueItem \* [q->maxSize];

//если не статическая

}

else {

//выделяем память под массив на один элемент больше

queueItem\*\* tmp = new queueItem \* [q->size + 1];

//копируем в него указатели из текущего

for (size\_t i = 0; i < q->size; i++) {

tmp[i] = q->data[i];

}

//освобождаем текущий

delete[] q->data;

//задаем новый массив

q->data = tmp;

}

//добавляем новый элемент в конец

q->data[q->size] = item;

//увеличиваем размер

q->size++;

//возвращаем, что все в порядке

return true;

}

queueItem queuePop(queue\* q) {

//если очередь пуста

if (queueIsEmpty(q))

//возвращаем "пустой" элемент

return queueItem{};

//иначе запоминаем верхний элемент

queueItem top = \*q->data[0];

//если статическая очередь

if (queueIsFixed(q)) {

//уменьшаем размер

q->size--;

//удаляем из памяти верхний элемент

delete q->data[0];

//сдвигаем указатели в массиве со сдвигом в 1 элемент

for (size\_t i = 0; i < q->size; i++)

q->data[i] = q->data[i + 1];

//иначе если очередь динамическая

}

else {

//создаем новый массив на 1 меньше

queueItem\*\* tmp = new queueItem \* [q->size - 1];

//уменьшаем размер очереди

q->size--;

//освобождаем память от верхнего элемента

delete q->data[0];

//копируем в новый массив указатели со сдвигом в 1 элемент

for (size\_t i = 0; i < q->size; i++) {

tmp[i] = q->data[i + 1];

}

//удаляем старый массив из памяти

delete[] q->data;

//задаем новый массив

q->data = tmp;

}

//возвращаем

return top;

}

//очистка очереди и памяти

void queueClear(queue\* q) {

//пробегаемся по всем элементам очереди

for (size\_t i = 0; i < q->size; i++)

//освобождаем память

delete q->data[i];

//освобождаем память от массива

delete[] q->data;

//обнуляем

q->size = 0;

q->data = nullptr;

}

//установка максимального кол-ва элементов в очереди

//делает очередь cо статическим (фиксированным) массивом

void queueSetMaxSize(queue\* q, const size\_t newSize) {

//очищаем очередь

queueClear(q);

//выделяем память под массив, если используется статическая

if (newSize != queue::maxSizeUnlimited)

q->data = new queueItem \* [newSize];

//задаем макс. размер (или неограниченный при newSize = maxSizeUnlimited)

q->maxSize = newSize;

}

void queueItemPrint(const queueItem\* item)

{

cout << item->key << " " << item->value << endl;

}

void queueItemPrint(const queueItem item)

{

queueItemPrint(&item);

}

//загрузка данных из файла

bool queueLoad(queue\* q, const string name) {

//пытаемся открыть файл

ifstream fin(name);

//если не удалось возвращаем признак ошибки

if (!fin) return false;

//очищаем очередь

queueClear(q);

//пока не конец файла

while (!fin.eof()) {

double key; string value;

fin >> key >> value;

//если чтение не удалось прерываем цикл

if (!fin) break;

//пытаемся добавить элемент в очередь

if (!queuePush(q, new queueItem{ key, value }))

{

//если не удалось

//закрываем файл

fin.close();

//возвращаем признак ошибки

return false;

}

}

//закрываем файл

fin.close();

//возвращаем признак ошибки или все ок в зависимости от состояния файла

return (bool)fin;

}

//выгрузка данных в файл

bool queueSave(queue\* q, const string name) {

//пытаемся открыть файл на запись

ofstream fout(name);

//если ошибка - возвращаем признак ошибки

if (!fout) return false;

//записываем все элементы очереди

for (size\_t i = 0; i < q->size; i++) {

fout << q->data[i]->key << " " << q->data[i]->value << endl;

//если произошла ошибка - прерываем цикл

if (!fout) break;

}

//закрываем файл

fout.close();

//возвращаем признак ошибки или все ок в зависимости от состояния файла

return (bool)fout;

}

void queuePrint(queue\* q) {

cout << "Размер очереди: " << q->size;

if (queueIsFixed(q)) {

cout << " ; Макс. размер: " << q->maxSize <<

" ; Статическая" << "\n";

}

else cout << " ; Динамическая" << "\n";

if (queueIsEmpty(q))

cout << "Очередь пуста\n";

else

for (size\_t i = 0; i < q->size; i++)

queueItemPrint(q->data[i]);

cout << "----" << endl;

}

//<--- в идеале надо возвращать массив указателей, что были найдены,

//а не просто их выводить, но это усложнит код

void queueSearch(queue\* q, double key)

{

//пробегаемся по всем элементам очереди

for (size\_t i = 0; i < q->size; i++)

{

if (key == q->data[i]->key)

queueItemPrint(q->data[i]);

}

}

bool queueReplacement(queue\* q, double oldkey, double key, string value)

{

//признак была-ли замена

bool result = false;

for (size\_t i = 0; i < q->size; i++)

{

if (q->data[i]->key == oldkey)

{

q->data[i]->key = key; q->data[i]->value = value;

result = true; //была ли замена

}

}

return result;

}

bool queueEditByIndex(queue\* q, size\_t index, double key, string value)

{

if (index < q->size)

{

q->data[index]->key = key;

q->data[index]->value = value;

return true;

}

return false;

}

//возвращаем "только на чтение" указатель

const queueItem\* queueSearchByIndex(queue\* q, size\_t index)

{

if (index < q->size)

return q->data[index];

return nullptr;

}

int main()

{

const string fileNameSave = "D:\\QueueSave.txt";

const string fileNameLoad = "D:\\QueueLoad.txt";

setlocale(LC\_ALL, "Rus");

size\_t number; queueItem item, newitem; double key, oldkey; string value;

cout << "Введите 1 для динамического массива и 2 для статического: ";

cin >> number;

queue q;

if (number == 2)

{

cout << "Укажите максимальный размер очереди: ";

cin >> number;

queueSetMaxSize(&q, number);

}

do

{

if (queueIsFixed(&q))

cout << "\n--Статическая очередь";

else

cout << "\n--Динамическая очередь";

cout <<

"--\n"

"1. Добавить элемент\n"

"2. Удалить элемент\n"

"3. Вывести верхний элемент\n"

"4. Узнать размер очереди\n"

"5. Записать данные в файл\n"

"6. Считать данные из файла\n"

"7. Очистить очередь\n"

"8. Поиск элемента по ключу\n"

"9. Установка значения элемента по заданному индексу (поиск по индексу)\n"

"10. Замена «старого» значения элемента на заданное новое (поиск по значению)\n"

"11. Получение значения элемента по заданному индексу\n"

"12. Вывести очередь\n"

"0. Выйти\n\n"

"Номер команды: ";

cin >> number;

switch (number)

{

//-----------------------------------------------

case 1:

cout << "\nВведите ключ и инф. поле: ";

cin >> item.key >> item.value;

if (queuePush(&q, new queueItem{ item.key, item.value }))

cout << "\nЭлемент добавлен\n";

else

cout << "\nОчередь заполнена\n";

break;

//-----------------------------------------------

case 2:

if (!queueIsEmpty(&q))

{

cout << "\nЭлемент удалён: "; queueItemPrint(queuePop(&q));

}

else

cout << "\nОчередь пуста\n";

break;

//-----------------------------------------------

case 3:

cout << "\nСамый верхний элемент: ";

if (!queueIsEmpty(&q))

{

queueItemPrint(queueTop(&q));

cout << "\n";

}

else cout << "\nОчередь пуста\n";

break;

//-----------------------------------------------

case 4:

cout << "\nТекущий размер очереди: ";

if (queueIsFixed(&q))

cout << q.size << "\n" << "Максимальный размер очереди: " << q.maxSize << "\n";

else cout << q.size << "\n";

break;

//-----------------------------------------------

case 5:

if (queueSave(&q, fileNameSave))

cout << "\nФайл записан\n";

else cout << "\nНе удалось записать файл\n";

break;

//-----------------------------------------------

case 6:

if (queueLoad(&q, fileNameLoad))

cout << "\nФайл загружен\n";

//<--- queueLoad возвращает ошибку и с неполной загрузкой файла

else cout << "\nНе удалось полностью загрузить файл\n";

break;

//-----------------------------------------------

case 7:

if (queueIsEmpty(&q)) { cout << "\nОчередь пуста \n"; break; }

queueClear(&q);

cout << "\nОчередь очищена\n";

break;

//-----------------------------------------------

case 8:

if (queueIsEmpty(&q)) { cout << "\nОчередь пуста \n"; break; }

cout << "\nВведите ключ: ";

cin >> key;

queueSearch(&q, key);

break;

//-----------------------------------------------

case 9: {

if (queueIsEmpty(&q)) { cout << "\nОчередь пуста \n"; break; }

size\_t index; cout << "\nВведите индекс: "; cin >> index;

if (index >= q.size) { cout << "\nИндекс выходит за пределы очереди"; break; }

cout << "\nВведите новый key и value: "; cin >> key >> value;

//<--- т.к. индекс мы проверяем раньше => что вернет нам queueEditByIndex не важно

// т.к. всегда будет true

queueEditByIndex(&q, index, key, value);

}

break;

//-----------------------------------------------

case 10:

if (queueIsEmpty(&q)) { cout << "\nОчередь пуста \n"; break; }

cout << "\nВведите ключ, значение которого хотите изменить: "; cin >> oldkey;

cout << "\nВведите новый ключ и его новое значение: "; cin >> key >> value;

//<--- проверка была ли замена

if (queueReplacement(&q, oldkey, key, value))

cout << "замена произведена" << endl;

else

cout << "ключ не найден" << endl;

break;

//-----------------------------------------------

case 11: {

size\_t index; cout << "Введите индекс записи: "; cin >> index;

const queueItem\* item = queueSearchByIndex(&q, index);

if (item)

queueItemPrint(item);

else

cout << "\nИндекс выходит за пределы очереди\n";

}

break;

case 12:

queuePrint(&q);

break;

case 0:

break;

default:

cout << endl << "Команда не определена\n\n";

break;

}

} while (number != 0);

queueClear(&q); //обязательная очистка очереди перед выходом

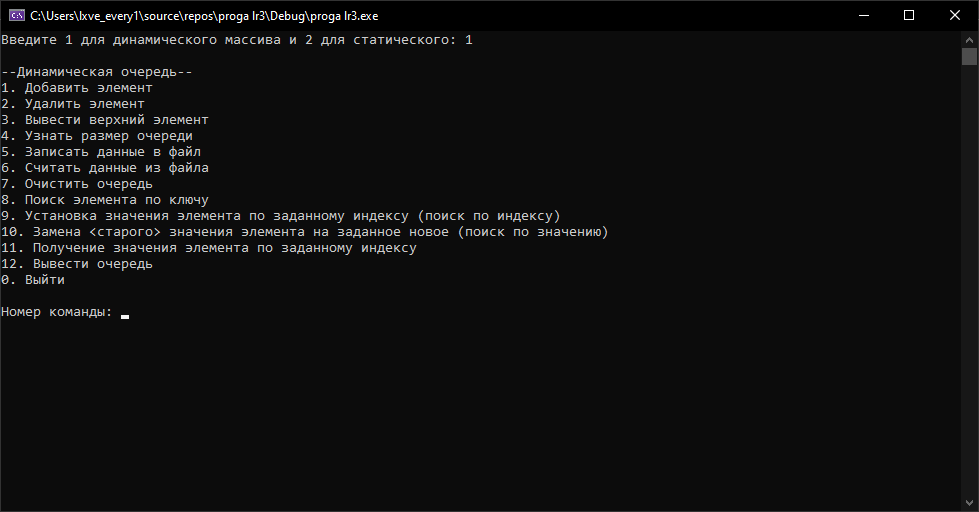
return 0;

}

**5. Примеры выполнения**

1) Меню + функция добавления.

Чтобы выбрать тип массива нажимаем 1 или 2:



Добавим элемент и сразу же проверим его наличие

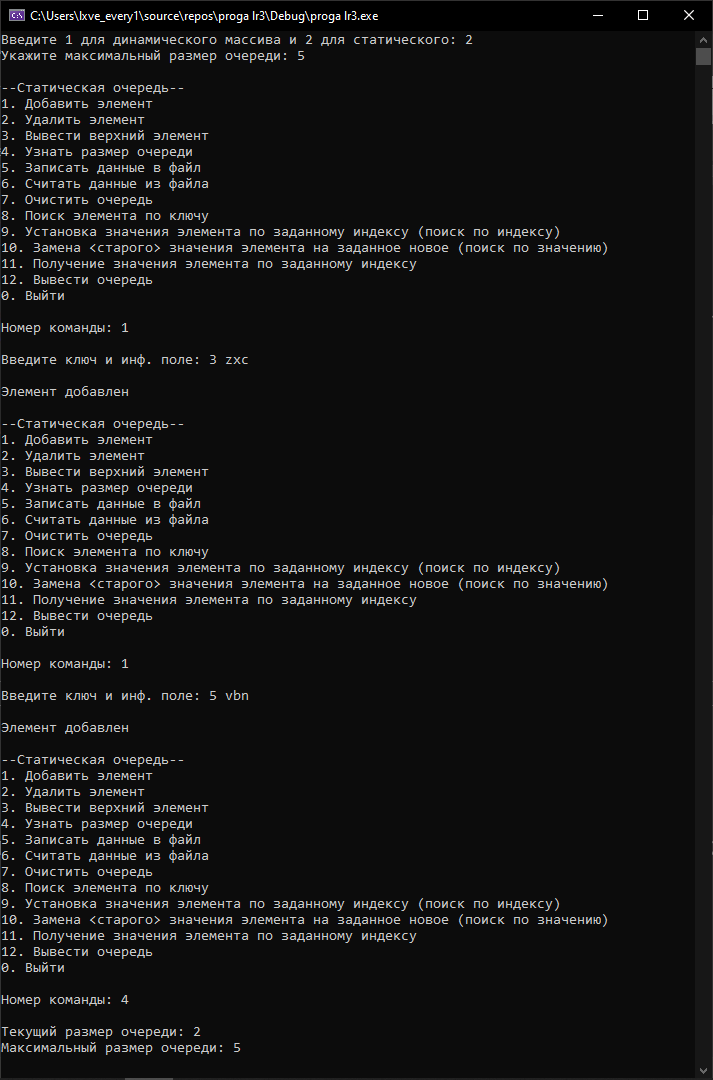


2) Выбираем статический массив, указываем размер = 5.

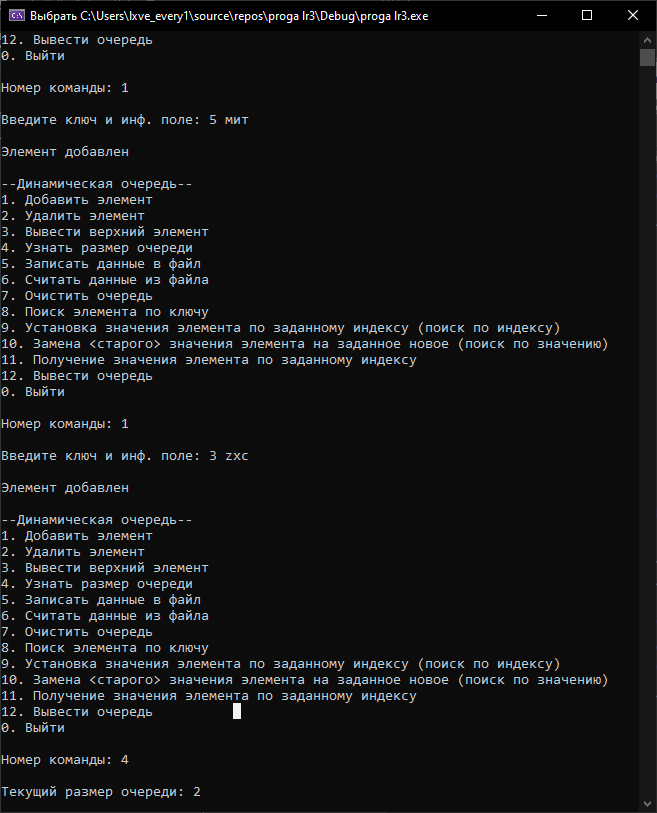
Добавим и удалим элемент



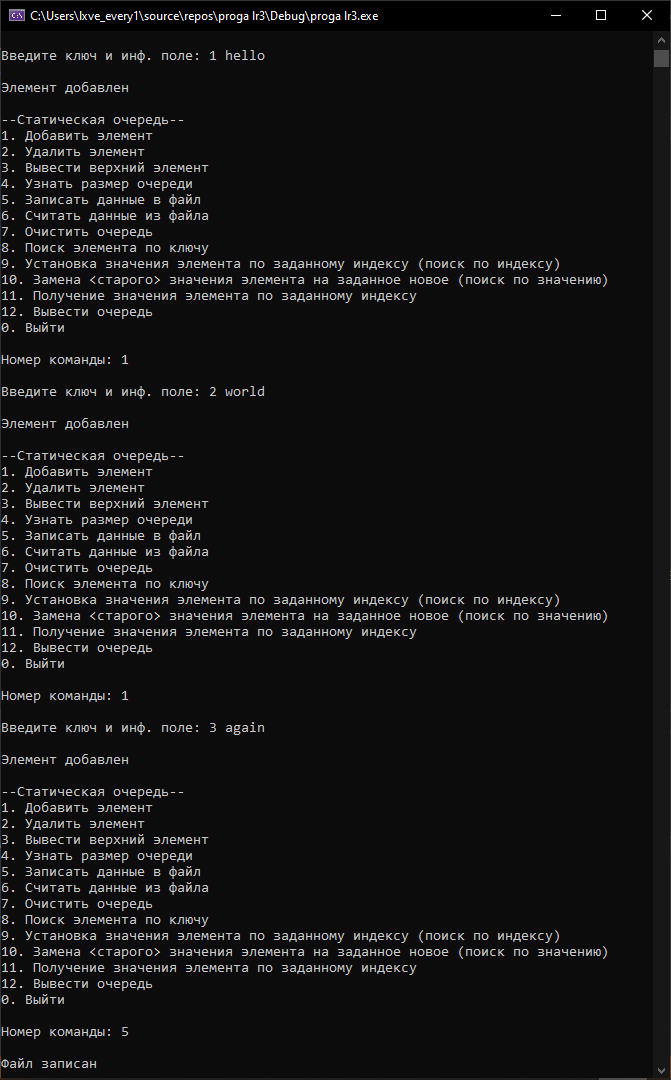
4) В статическом массиве[5] добавим 2 элемента и проверим его вместимость



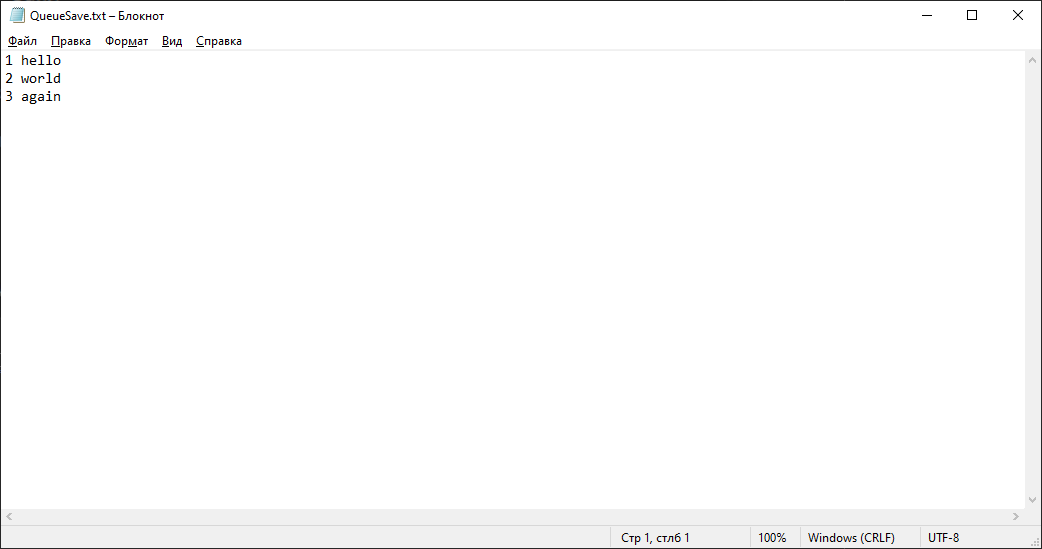
В динамическом массиве ситуация аналогичная, но макс. размер не устанавливается.



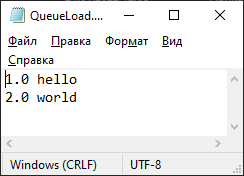
5) Добавим в массив[5] некоторые элементы и запишем их в файл



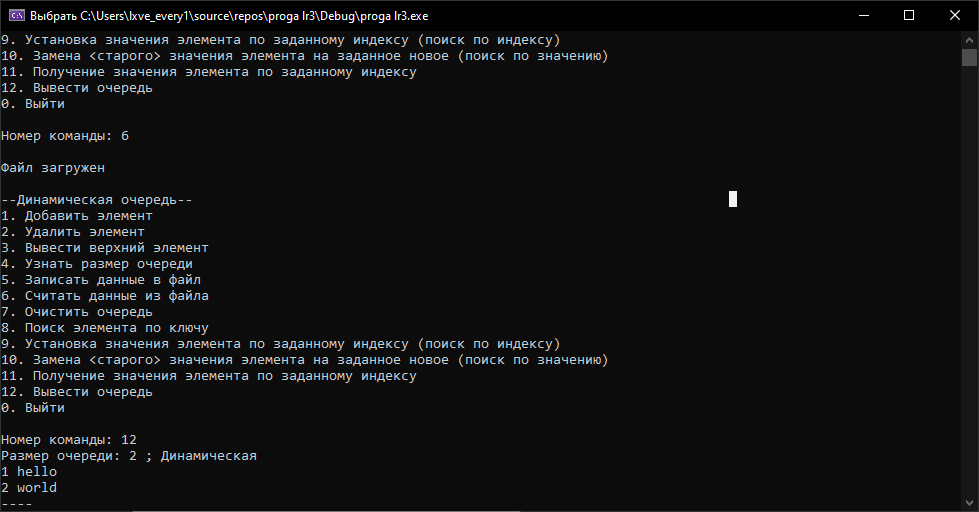
Результат:



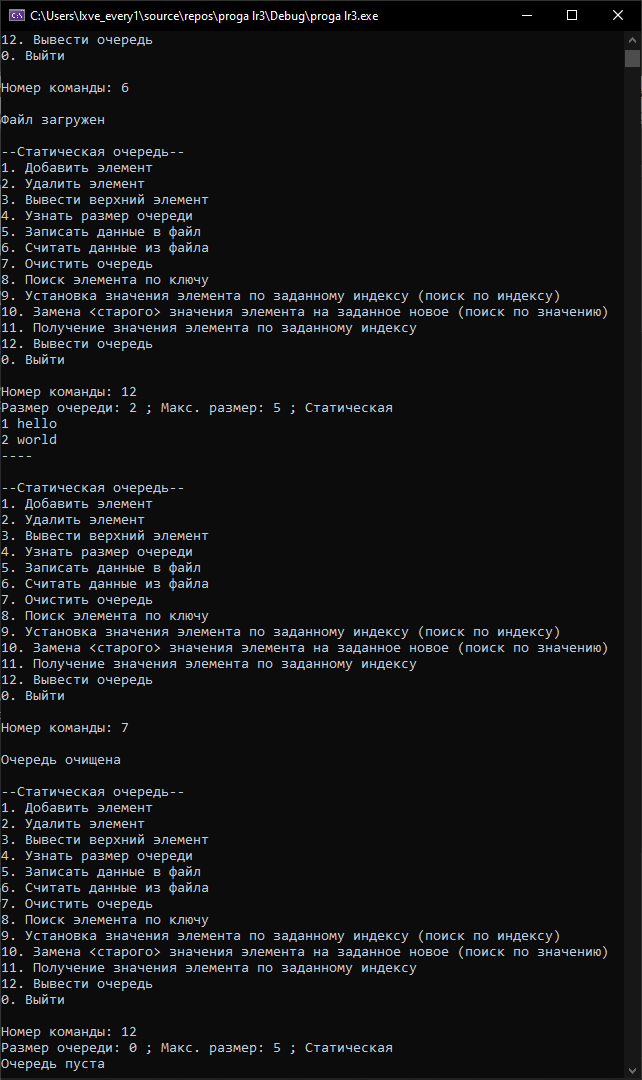
6) Считаем данные из файла в динамический массив



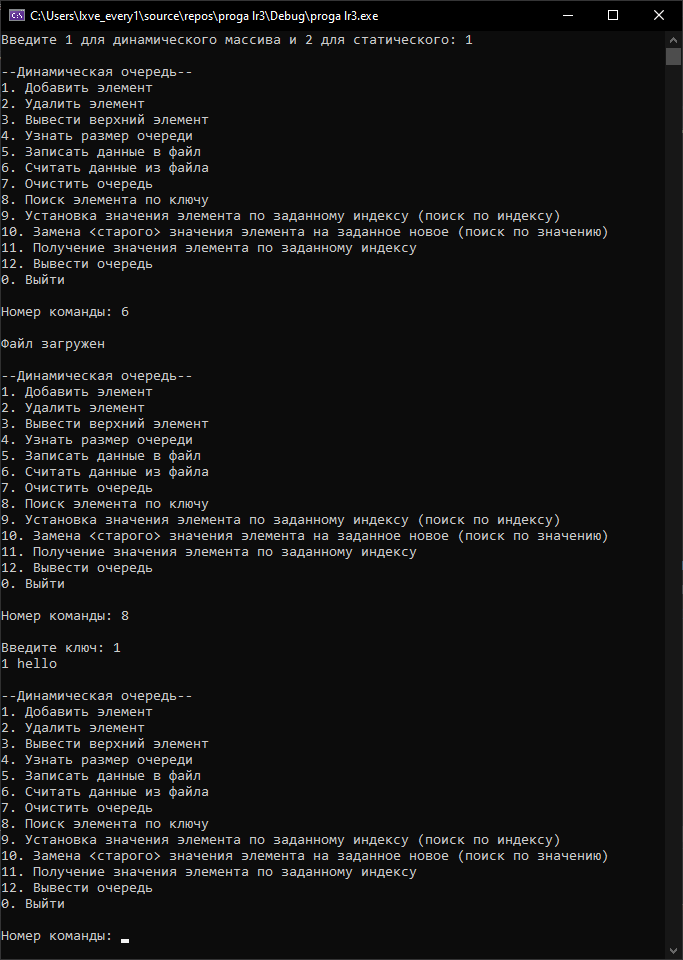
Загрузим и посмотрим на результат (функция 12)



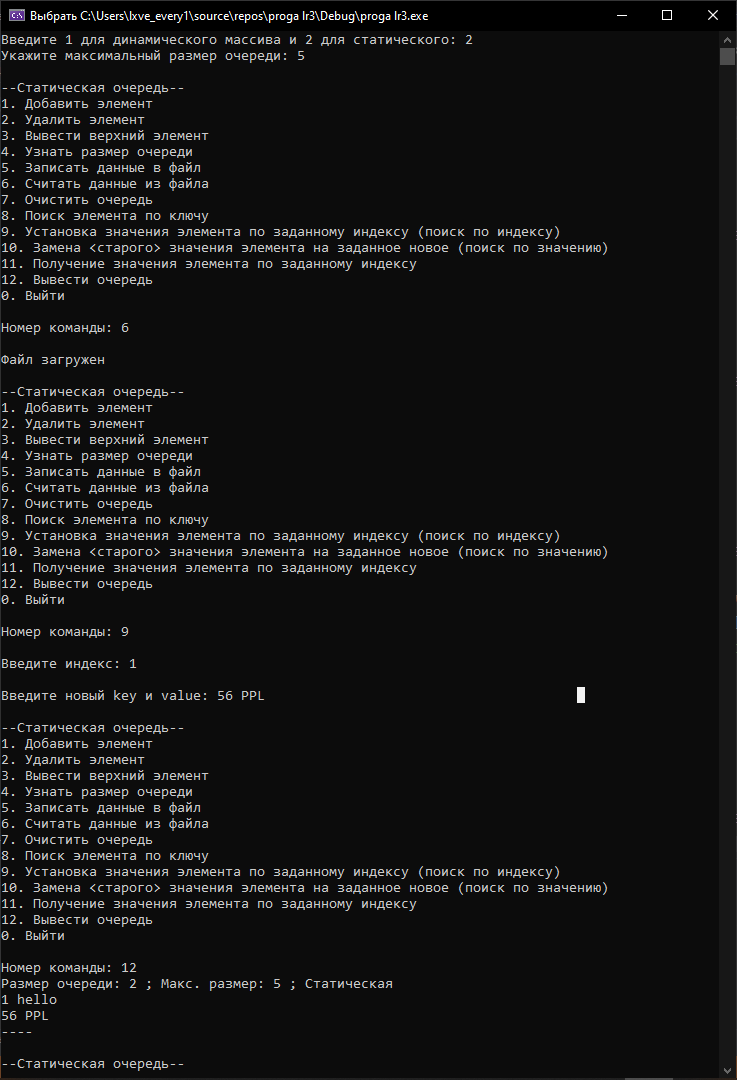
7) Загрузим тот же самый файл, но уже в массив[5] и очистим его



8) Загрузим тот же файл и проверим поиск



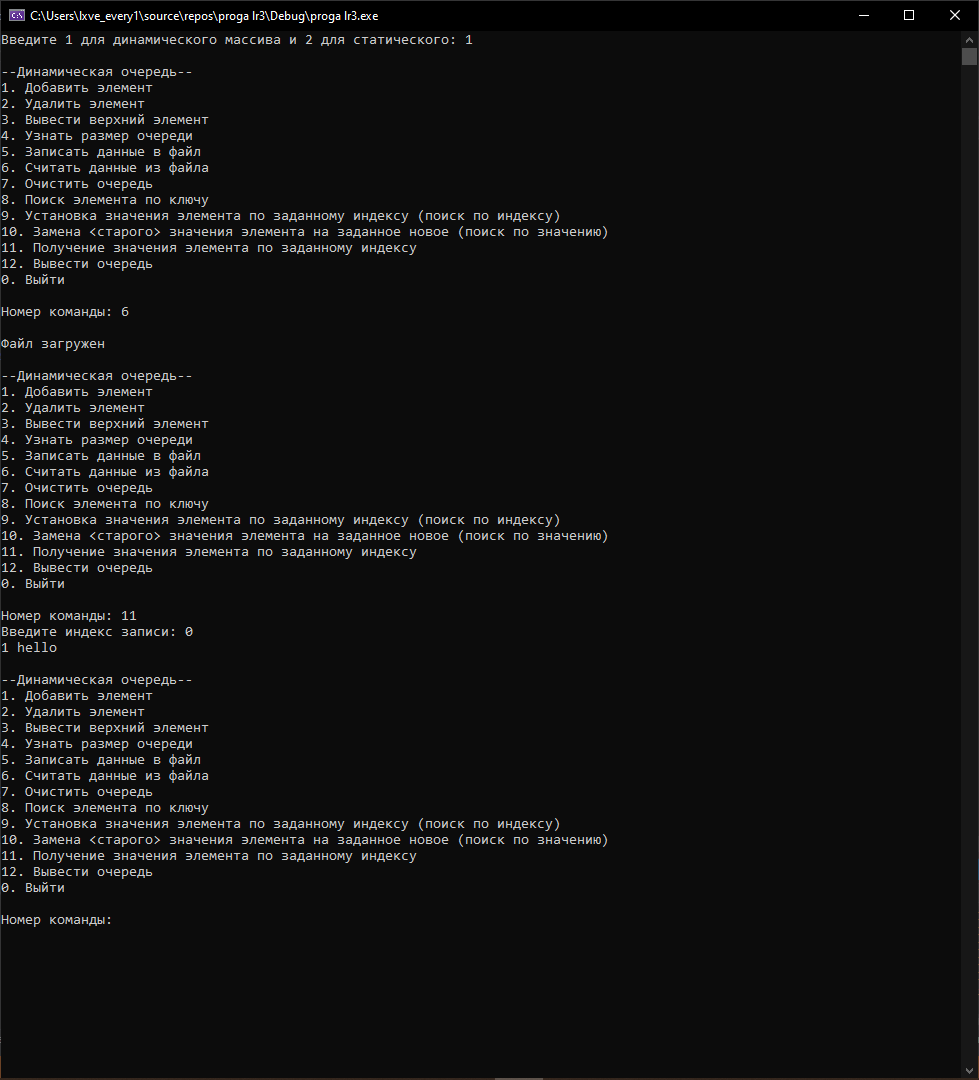
9) Загружу тот же самый файл и попробую замену элемента[1]: world



10) Замена записи с ключом 1



11) Поиск записи по индексу 0:



**6. Вывод:**

Изучил основные структуры данных и приобрёл навыки их реализации с помощью массивов и указателей.