Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**Пермский национальный исследовательский политехнический университет**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Основы алгоритмизации и программирования»

Тема: «Создание списка с помощью »

Семестр 2

*Работу выполнил: Учащийся группы ИВТ - 22-2б: Цыбуцынин Фёдор Александрович*

*Работу проверил: доцент кафедры ИТАС:*

*Полякова Ольга Андреевна*

Г. Пермь – 2023

**Постановка задачи**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом.

1.Создание очереди.

2.Добавление элемента в очередь.

3.Удаление элемента из очереди.

4.Печать очереди.

5.Запись очереди в файл.

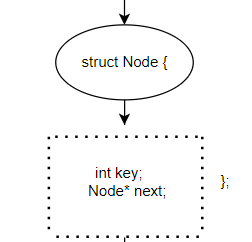
6.Уничтожение очереди.

7. Восстановление очереди из файла.

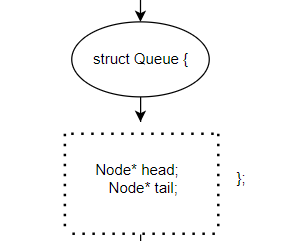
**Алгоритм решения задачи**

1. Создаётся структура struct Node с полями int key (информация, хранящаяся в узле) и Node\* next (указатель на следующий элемент стэка). Данная структура является структурой узла.
2. Создаётся структура struct Queue с полями Node\* head и Node\* tail.
3. Создаётся функция Queue\* createQueue(). Эта функция циклично просит ввести информацию, которая будет записываться в узел очереди, а после создаёт указатель на следующий узел.
4. Создаётся функция void enqueue() для добавления элементов в очередь. Пользователя просят ввести данные нового узла. Затем с помощью цикла находится последний элемент очереди, добавляется новый узел, указатели переносятся соответственно.
5. Создаётся функция void dequeue() с параметрами. С помощью цикла находит последний элемент очереди, затем переносит указатель с предыдущего элемента на nullptr.
6. Создаётся функция void printQueue (). Через цикл она выводит значения ключа, а затем переносит указатель на следующий элемент.
7. Создаётся функция void saveQueueToFile (). С помощью библиотеки fstream открывается файл по указанной ссылке, и через цикл в него записывается информация из очереди. Если файл не удаётся открыть, функция выводит ошибку.
8. Создаётся функция void destroyQueue (). Через цикл она считает количество элементов в очереди, а затем также через цикл удаляет их.
9. Создаётся функция Queue\* loadQueueFromFile () с параметрами. Далее создаётся цикл, в котором вся информация записывается в новые узлы, а так же создаются указатели. Цикл завершается, когда заканчивается информация в файле.
10. Создаётся функция int main(). В ней создаётся список Queue\* q, который пользователя просят заполнить.

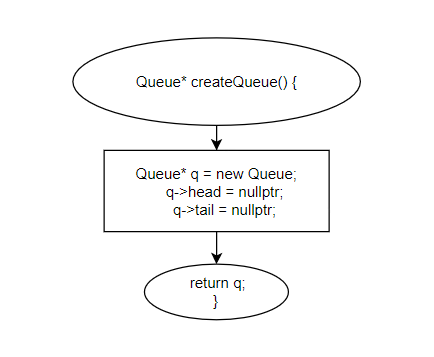
**Блок-схема**

****

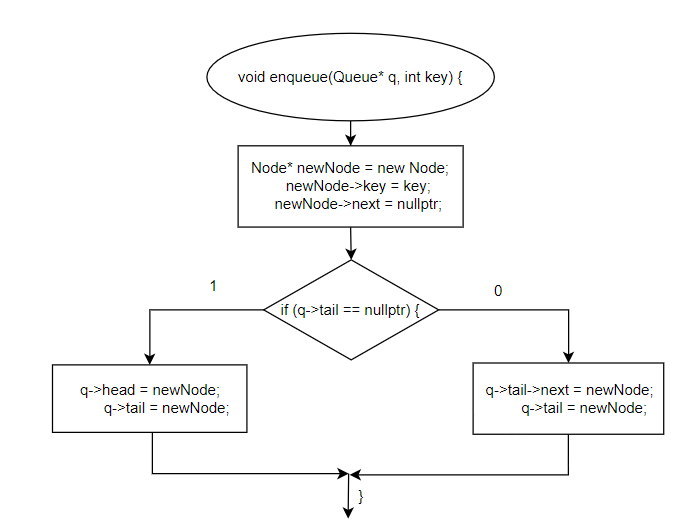
**Рис. 1 – Блок-схема структуры Node**

****

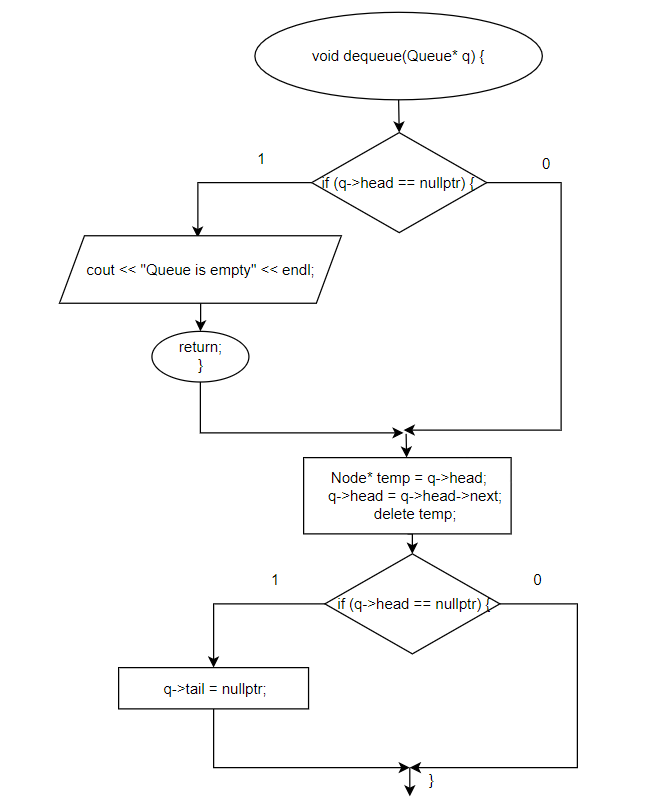
**Рис. 2 – Блок-схема структуры Queue**

****

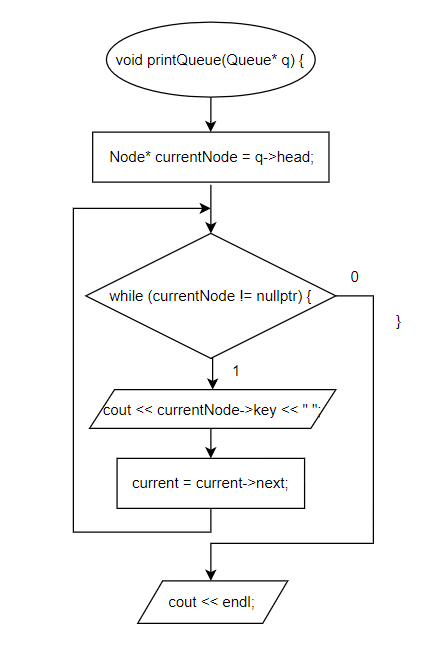
**Рис. 3 – Блок-схема функции Queue\* createQueue()**

****

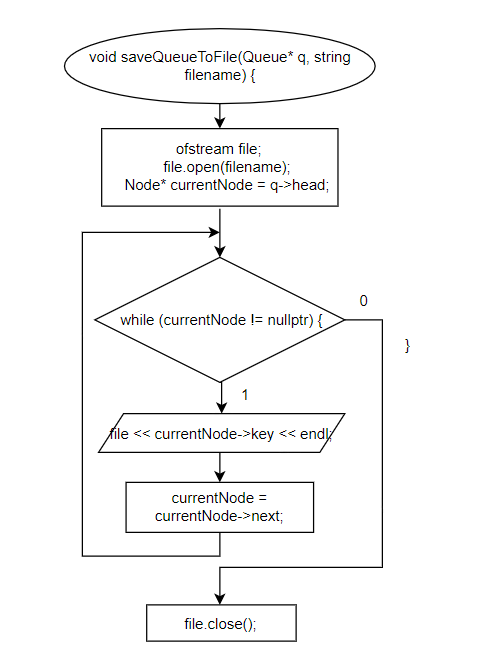
**Рис. 4 – Блок-схема функции void enqueue()**

****

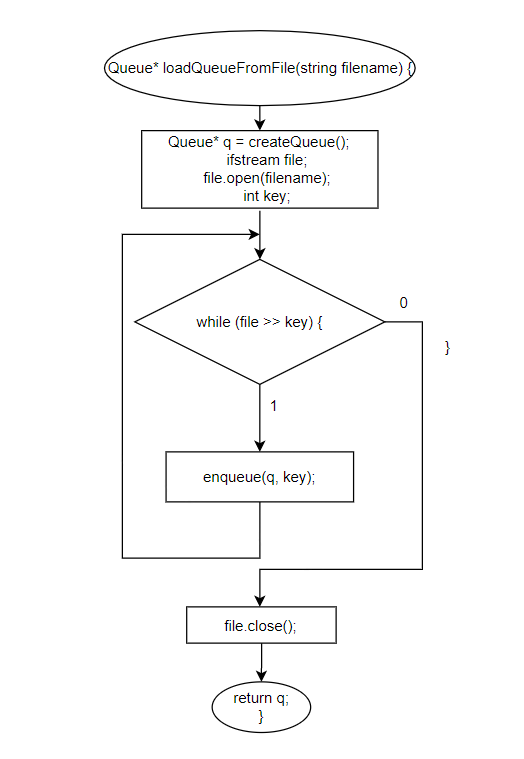
**Рис. 5 – Блок-схема функции void dequeue()**



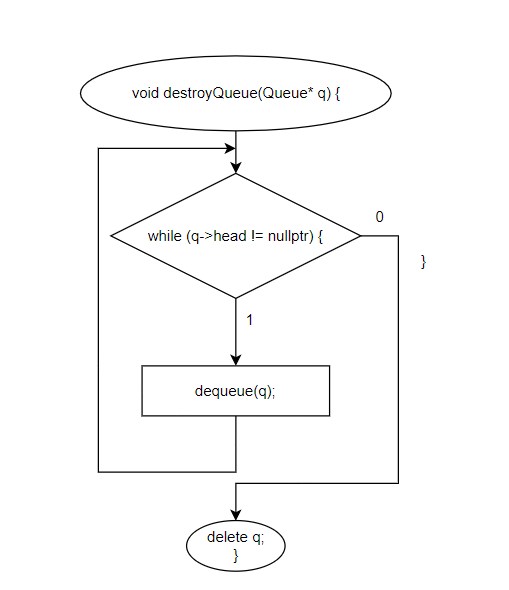
**Рис. 6 – Блок-схема функции void printQueue()**



**Рис. 7 – Блок-схема функции void saveQueueToFile ()**

****

**Рис. 8 – Блок-схема функции Queue\* loadQueueFromFile ()**



**Рис. 9 – Блок-схема функции void destroyQueue()**

**Код программы**

#include <iostream>

#include <fstream>

using namespace std;

struct Node {

int key;

Node\* next;

};

struct Queue {

Node\* head;

Node\* tail;

};

// Создание очереди

Queue\* createQueue() {

Queue\* q = new Queue;

q->head = nullptr;

q->tail = nullptr;

return q;

}

// Добавление элемента в очередь

void enqueue(Queue\* q, int key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

if (q->tail == nullptr) {

q->head = newNode;

q->tail = newNode;

}

else {

q->tail->next = newNode;

q->tail = newNode;

}

}

// Удаление элемента из очереди

void dequeue(Queue\* q) {

if (q->head == nullptr) {

cout << "Queue is empty" << endl;

return;

}

Node\* temp = q->head;

q->head = q->head->next;

delete temp;

if (q->head == nullptr) {

q->tail = nullptr;

}

}

// Печать очереди

void printQueue(Queue\* q) {

Node\* currentNode = q->head;

while (currentNode != nullptr) {

cout << currentNode->key << " ";

currentNode = currentNode->next;

}

cout << endl;

}

// Запись очереди в файл

void saveQueueToFile(Queue\* q, string filename) {

ofstream file;

file.open(filename);

Node\* currentNode = q->head;

while (currentNode != nullptr) {

file << currentNode->key << endl;

currentNode = currentNode->next;

}

file.close();

}

// Восстановление очереди из файла

Queue\* loadQueueFromFile(string filename) {

Queue\* q = createQueue();

ifstream file;

file.open(filename);

int key;

while (file >> key) {

enqueue(q, key);

}

file.close();

return q;

}

// Уничтожение очереди

void destroyQueue(Queue\* q) {

while (q->head != nullptr) {

dequeue(q);

}

delete q;

}

int main() {

Queue\* q = createQueue();

setlocale(0, "rus");

int k;

int val;

cout << "Введите количество элементов: ";

cin >> k;

for (int i = 1; i <= k; i++) {

cout << i << ". ";

cin >> val;

enqueue(q, val);

}

cout << "Очередь: ";

printQueue(q);

dequeue(q);

cout << "Очередь после удаления: ";

printQueue(q);

cout << "Очередь после добавления элемента: ";

enqueue(q, 4);

printQueue(q);

cout << "Очередь успешно записана в файл!" << endl;

cout << "Очередь успешно удалена!" << endl;

saveQueueToFile(q, "queue.txt");

destroyQueue(q);

Queue\* loadedQueue = loadQueueFromFile("queue.txt");

cout << "Очередь после восстановления из файла: ";

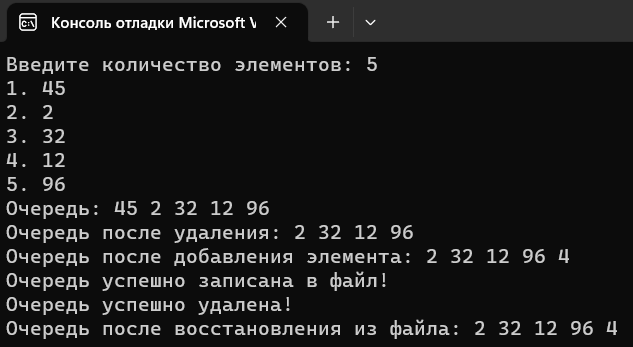
printQueue(loadedQueue);

destroyQueue(loadedQueue);

return 0;

}

**Результаты работы программы**



**Рис. 10 – Вывод консоли решения программы**