Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет»**

Электротехнический факультет

Кафедра **«**Информационные технологии и автоматизированные системы**»**

направление подготовки: 09.03.04 – **«**Программная инженерия»

**Лабораторная работа №11.**

**“Функции и массивы”  
Вариант 5**

Выполнил студент гр. РИС-24-2б

Трофимов Степан Степанович

Проверил:

Доц. каф. ИТАС

Ольга Андреевна Полякова

(оценка) (подпись)

(дата)

г. Пермь, 2025

1. **Постановка задачи:**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом. Для каждого вариант разработать следующие функции:

1. Создание списка.
2. Добавление элемента в список (в соответствии со своим вариантом).
3. Удаление элемента из списка (в соответствии со своим вариантом).
4. Печать списка.
5. Запись списка в файл.
6. Уничтожение списка.
7. Восстановление списка из файла.

Записи в линейном списке содержат ключевое поле типа int. Сформировать однонаправленный список. Удалить из него К элементов, начиная с заданного номера, добавить К элементов, начиная с заданного номера;

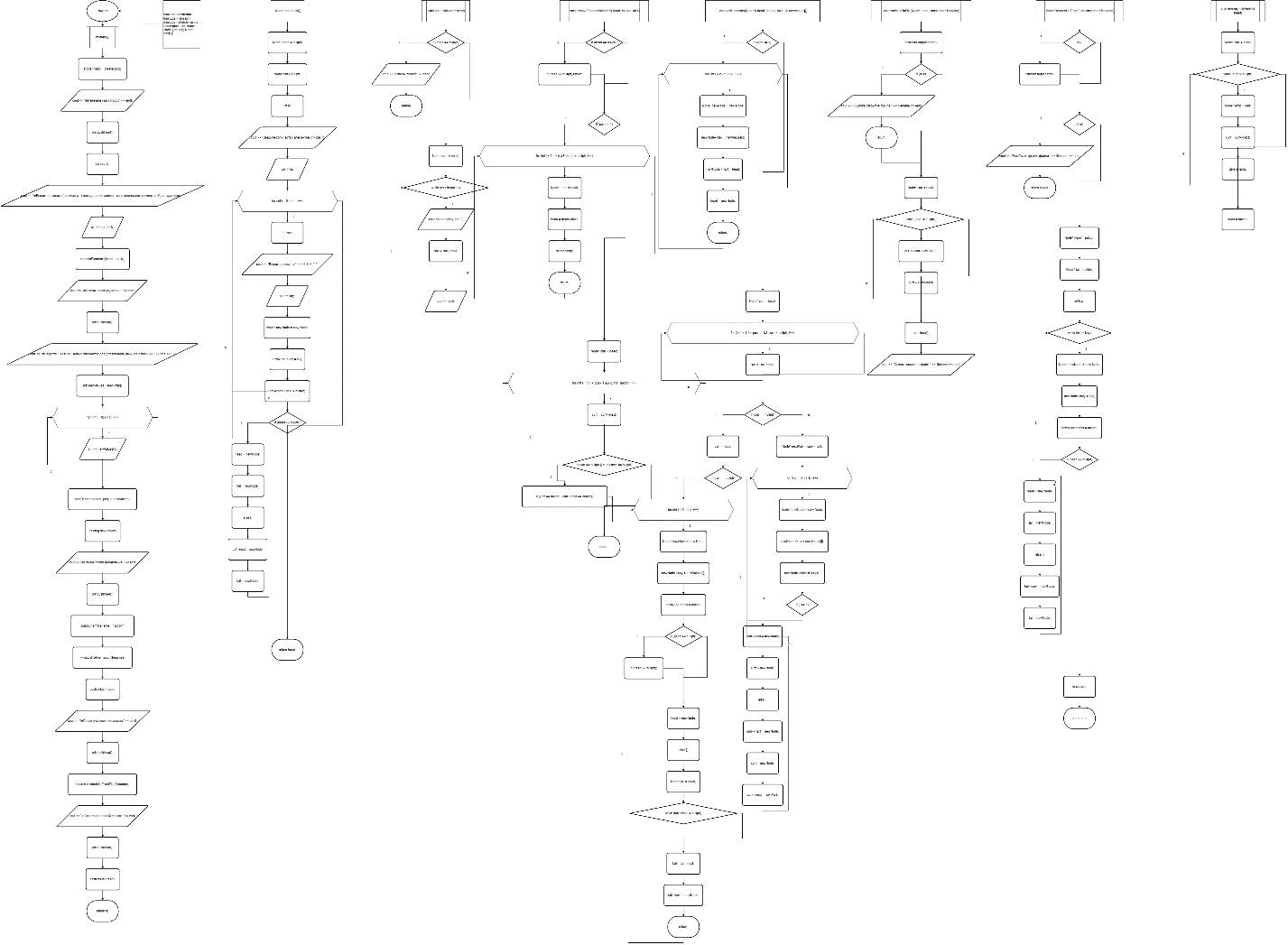
**Односвязные списки**

**Анализ задачи**

Данная задача требует реализации основных операций над однонаправленным списком, что позволяет проанализировать работу с динамической памятью и базовыми операциями со структурами данных в C++.

* **Создание и заполнение списка:**  
  Необходимо создать динамически выделенную память для каждого узла, реализовать механизм добавления новых элементов в конец списка. Это демонстрирует работу с указателями и динамическим выделением памяти.
* **Печать списка:**  
  Функция печати должна корректно обрабатывать пустой список, выводя соответствующее сообщение. Это важно для проверки корректности работы остальных функций.
* **Удаление и добавление элементов:**  
  Реализованы операции удаления и вставки элементов, начиная с указанной позиции. Здесь важно корректное управление указателями, чтобы не потерять связь между узлами, и избегать утечек памяти.
* **Уничтожение списка:**  
  Необходимо корректно освобождать всю выделенную память, что является важным аспектом для предотвращения утечек памяти.

**Блок-схема**

****

**Код**

#include <iostream>

#include <fstream>

#include <stdio.h>

using namespace std;

struct Node {

int key;

Node\* next;

};

Node\* createList() {

Node\* head = nullptr;

Node\* tail = nullptr;

int n;

cout << "Введите количество элементов списка: ";

cin >> n;

for (int i = 0; i < n; i++) {

int key;

cout << "Введите элемент " << i + 1 << ": ";

cin >> key;

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

}

return head;

}

void printList(Node\* head) {

if (head == nullptr) {

cout << "Список пустой" << endl;

return;

}

Node\* curr = head;

while (curr != nullptr) {

cout << curr->key << " ";

curr = curr->next;

}

cout << endl;

}

void removeElements(Node\*& head, int pos, int k) {

if (head == nullptr) return;

if (pos <= 1) {

for (int i = 0; i < k && head != nullptr; i++) {

Node\* temp = head;

head = head->next;

delete temp;

}

return;

}

Node\* curr = head;

for (int i = 1; i < pos - 1 && curr != nullptr; i++) {

curr = curr->next;

}

if (curr == nullptr || curr->next == nullptr)

return;

Node\* temp = curr->next;

for (int i = 0; i < k && temp != nullptr; i++) {

Node\* toDelete = temp;

temp = temp->next;

delete toDelete;

}

curr->next = temp;

}

void addElements(Node\*& head, int pos, int k, int newValues[]) {

if (pos <= 1) {

for (int i = k - 1; i >= 0; i--) {

Node\* newNode = new Node;

newNode->key = newValues[i];

newNode->next = head;

head = newNode;

}

return;

}

Node\* curr = head;

for (int i = 1; i < pos - 1 && curr != nullptr; i++) {

curr = curr->next;

}

if (curr == nullptr) {

curr = head;

if (curr == nullptr) {

for (int i = 0; i < k; i++) {

Node\* newNode = new Node;

newNode->key = newValues[i];

newNode->next = nullptr;

if (head == nullptr)

head = newNode;

else {

Node\* tail = head;

while (tail->next != nullptr)

tail = tail->next;

tail->next = newNode;

}

}

return;

}

else {

while (curr->next != nullptr)

curr = curr->next;

for (int i = 0; i < k; i++) {

Node\* newNode = new Node;

newNode->key = newValues[i];

newNode->next = nullptr;

curr->next = newNode;

curr = newNode;

}

return;

}

}

else {

Node\* nextPart = curr->next;

for (int i = 0; i < k; i++) {

Node\* newNode = new Node;

newNode->key = newValues[i];

newNode->next = nullptr;

if (i == 0) {

curr->next = newNode;

curr = newNode;

}

else {

curr->next = newNode;

curr = newNode;

}

}

curr->next = nextPart;

}

}

void writeListToFile(Node\* head, const char\* filename) {

ofstream out(filename);

if (!out) {

cout << "Ошибка открытия файла " << filename << endl;

return;

}

Node\* curr = head;

while (curr != nullptr) {

out << curr->key << " ";

curr = curr->next;

}

out.close();

cout << "Список записан в файл " << filename << endl;

}

Node\* restoreListFromFile(const char\* filename) {

ifstream in(filename);

if (!in) {

cout << "Ошибка открытия файла " << filename << endl;

return nullptr;

}

Node\* head = nullptr;

Node\* tail = nullptr;

int key;

while (in >> key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

tail = newNode;

}

else {

tail->next = newNode;

tail = newNode;

}

}

in.close();

return head;

}

void destroyList(Node\*& head) {

Node\* curr = head;

while (curr != nullptr) {

Node\* temp = curr;

curr = curr->next;

delete temp;

}

head = nullptr;

}

int main() {

Node\* head = createList();

cout << "\nИзначальный список:" << endl;

printList(head);

int pos, k;

cout << "\nВведите позицию (начиная с 1) для удаления элементов и количество элементов K для удаления: ";

cin >> pos >> k;

removeElements(head, pos, k);

cout << "\nСписок после удаления:" << endl;

printList(head);

cout << "\nВведите " << k << " новых элементов для добавления, начиная с позиции " << pos << ": ";

int\* newValues = new int[k];

for (int i = 0; i < k; i++) {

cin >> newValues[i];

}

addElements(head, pos, k, newValues);

delete[] newValues;

cout << "\nСписок после добавления:" << endl;

printList(head);

const char\* filename = "list.txt";

writeListToFile(head, filename);

destroyList(head);

cout << "\nПосле уничтожения списка:" << endl;

printList(head);

head = restoreListFromFile(filename);

cout << "\nВосстановленный список:" << endl;

printList(head);

destroyList(head);

return 0;

}

**Двусвязные списки**

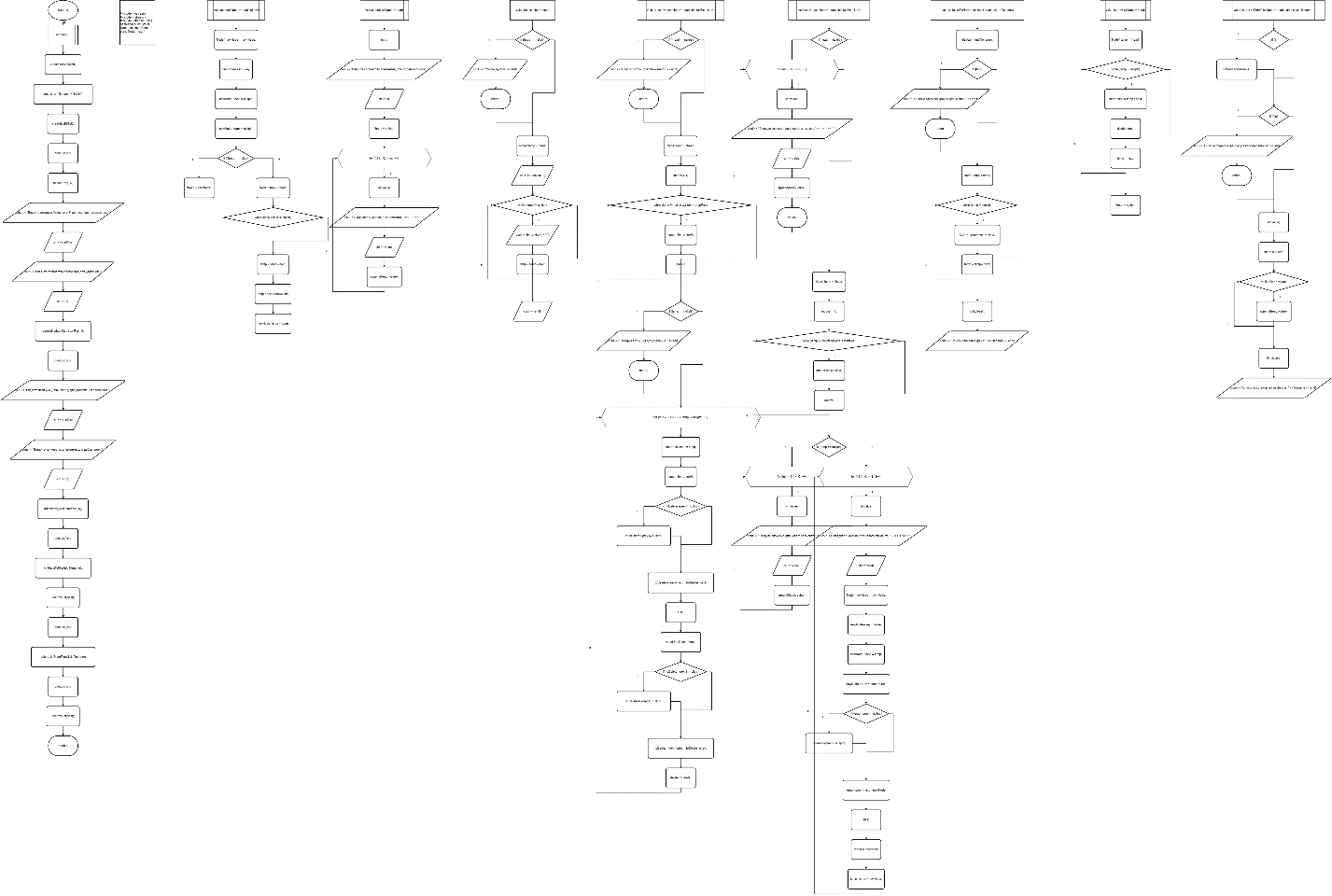
**Анализ задачи**

**Работа с динамической памятью:**  
Решение предполагает создание двусвязного списка, где каждый узел выделяется динамически. Это требует грамотного управления памятью и последующего её освобождения, чтобы избежать утечек.

**Базовые операции со списком:**:

* **Создания списка:** Запрос у пользователя количества и значений элементов.
* **Печати списка:** Вывод элементов или сообщения о пустом списке.
* **Удаления и добавления элементов:** Удаление K узлов, начиная с заданного номера, и вставка K новых узлов в указанное место, что демонстрирует манипуляции указателями.

**Блок-схема**

****

**Код**

#include <iostream>

#include <fstream>

#include <stdio.h>

using namespace std;

struct Node {

int key;

Node\* prev;

Node\* next;

};

void append(Node\*\* head, int key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

newNode->prev = nullptr;

if (\*head == nullptr) {

\*head = newNode;

}

else {

Node\* temp = \*head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

newNode->prev = temp;

}

}

void createList(Node\*\* head) {

int n;

cout << "Введите количество элементов для создания списка: ";

cin >> n;

\*head = nullptr;

for (int i = 0; i < n; i++) {

int value;

cout << "Введите значение для элемента " << i + 1 << ": ";

cin >> value;

append(head, value);

}

}

void printList(Node\* head) {

if (head == nullptr) {

cout << "Список пустой" << endl;

return;

}

Node\* temp = head;

cout << "Список: ";

while (temp != nullptr) {

cout << temp->key << " ";

temp = temp->next;

}

cout << endl;

}

void removeNodes(Node\*\* head, int startPos, int K) {

if (\*head == nullptr) {

cout << "Список пустой, удалять нечего." << endl;

return;

}

Node\* temp = \*head;

int pos = 1;

while (temp != nullptr && pos < startPos) {

temp = temp->next;

pos++;

}

if (temp == nullptr) {

cout << "Неверная позиция для удаления." << endl;

return;

}

for (int i = 0; i < K && temp != nullptr; i++) {

Node\* toDelete = temp;

temp = temp->next;

if (toDelete->prev != nullptr)

toDelete->prev->next = toDelete->next;

else

\*head = toDelete->next;

if (toDelete->next != nullptr)

toDelete->next->prev = toDelete->prev;

delete toDelete;

}

}

void addNodes(Node\*\* head, int startPos, int K) {

if (\*head == nullptr) {

for (int i = 0; i < K; i++) {

int value;

cout << "Введите значение для нового элемента " << i + 1 << ": ";

cin >> value;

append(head, value);

}

return;

}

Node\* temp = \*head;

int pos = 1;

while (temp != nullptr && pos < startPos) {

temp = temp->next;

pos++;

}

if (temp == nullptr) {

for (int i = 0; i < K; i++) {

int value;

cout << "Введите значение для нового элемента " << i + 1 << ": ";

cin >> value;

append(head, value);

}

}

else {

for (int i = 0; i < K; i++) {

int value;

cout << "Введите значение для нового элемента " << i + 1 << ": ";

cin >> value;

Node\* newNode = new Node;

newNode->key = value;

newNode->next = temp;

newNode->prev = temp->prev;

if (temp->prev != nullptr)

temp->prev->next = newNode;

else

\*head = newNode;

temp->prev = newNode;

}

}

}

void writeListToFile(Node\* head, const char\* filename) {

ofstream fout(filename);

if (!fout) {

cout << "Ошибка открытия файла для записи." << endl;

return;

}

Node\* temp = head;

while (temp != nullptr) {

fout << temp->key << endl;

temp = temp->next;

}

fout.close();

cout << "Список записан в файл: " << filename << endl;

}

void destroyList(Node\*\* head) {

Node\* temp = \*head;

while (temp != nullptr) {

Node\* next = temp->next;

delete temp;

temp = next;

}

\*head = nullptr;

}

void restoreListFromFile(Node\*\* head, const char\* filename) {

ifstream fin(filename);

if (!fin) {

cout << "Ошибка открытия файла для восстановления." << endl;

return;

}

int value;

\*head = nullptr;

while (fin >> value) {

append(head, value);

}

fin.close();

cout << "Список восстановлен из файла: " << filename << endl;

}

int main() {

Node\* list = nullptr;

const char\* filename = "list.txt";

createList(&list);

printList(list);

int startPos, K;

cout << "Введите позицию (начиная с 1) для удаления элементов: ";

cin >> startPos;

cout << "Введите количество элементов для удаления: ";

cin >> K;

removeNodes(&list, startPos, K);

printList(list);

cout << "Введите позицию (начиная с 1) для добавления элементов: ";

cin >> startPos;

cout << "Введите количество элементов для добавления: ";

cin >> K;

addNodes(&list, startPos, K);

printList(list);

writeListToFile(list, filename);

destroyList(&list);

printList(list);

restoreListFromFile(&list, filename);

printList(list);

destroyList(&list);

return 0;

}

**Стек**

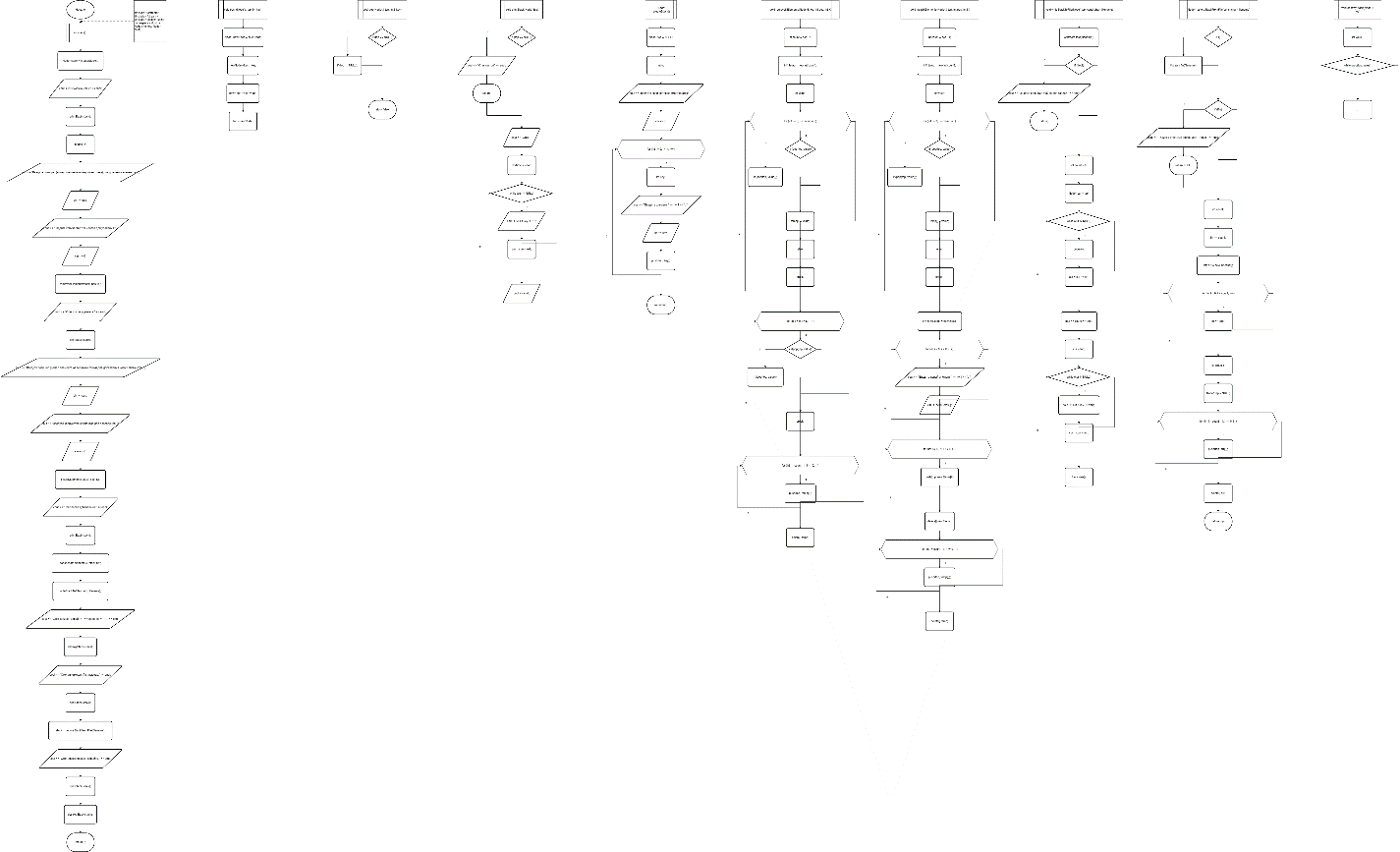
**Анализ задачи**

Задача демонстрирует реализацию основных операций над динамическим стеком, используя структуру связного списка. Краткий анализ:

* **Структура данных:**  
  Стек реализован как связный список, что позволяет эффективно добавлять и удалять элементы (операции push/pop).
* **Модификация стека:**  
  Особое внимание уделено модификации стека — удалению и вставке K элементов с заданной позиции. Для сохранения порядка элементов используются временные массивы, что демонстрирует практический подход к работе с ограниченной функциональностью стека.
* **Работа с файлами:**  
  Реализованы функции записи и восстановления стека в/из файла. Это позволяет сохранить состояние стека между сессиями и правильно восстановить исходное расположение элементов.
* **Управление памятью:**  
  Задача включает корректное выделение и освобождение динамической памяти, что критически важно для предотвращения утечек в C++.
* **Ограничение библиотек:**  
  Решение соответствует заданным ограничениям, используя только библиотеки iostream, fstream и stdio.h, что подчёркивает умение работать с базовыми средствами языка.

В целом, задача объединяет работу с динамическими структурами данных, операциями ввода/вывода и файлами, а также демонстрирует правильное управление памятью в C++.

**Блок-схема**

****

**Код**

#include <iostream>

#include <fstream>

#include <stdio.h>

using namespace std;

struct Node {

int key;

Node\* next;

};

void push(Node\*& top, int key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = top;

top = newNode;

}

bool pop(Node\*& top, int& key) {

if (top == NULL)

return false;

Node\* temp = top;

key = top->key;

top = top->next;

delete temp;

return true;

}

void printStack(Node\* top) {

if (top == NULL) {

cout << "Стек пустой" << endl;

return;

}

cout << "Стек: ";

Node\* cur = top;

while (cur != NULL) {

cout << cur->key << " ";

cur = cur->next;

}

cout << endl;

}

Node\* createStack() {

Node\* top = NULL;

int n;

cout << "Введите количество элементов стека: ";

cin >> n;

for (int i = 0; i < n; i++) {

int key;

cout << "Введите элемент " << i + 1 << ": ";

cin >> key;

push(top, key);

}

return top;

}

void removeKElements(Node\*& top, int pos, int K) {

int count = pos - 1;

int\* temp = new int[count];

int value;

for (int i = 0; i < count; i++) {

if (pop(top, value))

temp[i] = value;

else

break;

}

for (int j = 0; j < K; j++) {

if (!pop(top, value))

break;

}

for (int i = count - 1; i >= 0; i--) {

push(top, temp[i]);

}

delete[] temp;

}

void insertKElements(Node\*& top, int pos, int K) {

int count = pos - 1;

int\* temp = new int[count];

int value;

for (int i = 0; i < count; i++) {

if (pop(top, value))

temp[i] = value;

else

break;

}

int\* newElems = new int[K];

for (int i = 0; i < K; i++) {

cout << "Введите новый элемент " << i + 1 << ": ";

cin >> newElems[i];

}

for (int i = K - 1; i >= 0; i--) {

push(top, newElems[i]);

}

delete[] newElems;

for (int i = count - 1; i >= 0; i--) {

push(top, temp[i]);

}

delete[] temp;

}

void writeStackToFile(Node\* top, const char\* filename) {

ofstream fout(filename);

if (!fout) {

cout << "Ошибка открытия файла для записи." << endl;

return;

}

int count = 0;

Node\* cur = top;

while (cur != NULL) {

count++;

cur = cur->next;

}

fout << count << endl;

cur = top;

while (cur != NULL) {

fout << cur->key << endl;

cur = cur->next;

}

fout.close();

}

Node\* restoreStackFromFile(const char\* filename) {

ifstream fin(filename);

if (!fin) {

cout << "Ошибка открытия файла для чтения." << endl;

return NULL;

}

int count;

fin >> count;

int\* arr = new int[count];

for (int i = 0; i < count; i++) {

fin >> arr[i];

}

fin.close();

Node\* top = NULL;

for (int i = count - 1; i >= 0; i--) {

push(top, arr[i]);

}

delete[] arr;

return top;

}

void destroyStack(Node\*& top) {

int value;

while (pop(top, value))

;

}

int main() {

Node\* stack = createStack();

cout << "Исходный стек:" << endl;

printStack(stack);

int pos, K;

cout << "Введите позицию (номер элемента от вершины стека) для удаления элементов: ";

cin >> pos;

cout << "Введите количество элементов для удаления: ";

cin >> K;

removeKElements(stack, pos, K);

cout << "Стек после удаления:" << endl;

printStack(stack);

cout << "Введите позицию (номер элемента от вершины стека) для добавления новых элементов: ";

cin >> pos;

cout << "Введите количество элементов для добавления: ";

cin >> K;

insertKElements(stack, pos, K);

cout << "Стек после добавления:" << endl;

printStack(stack);

const char\* filename = "stack.txt";

writeStackToFile(stack, filename);

cout << "Стек записан в файл \"" << filename << "\"." << endl;

destroyStack(stack);

cout << "Стек уничтожен. При выводе:" << endl;

printStack(stack);

stack = restoreStackFromFile(filename);

cout << "Стек восстановлен из файла:" << endl;

printStack(stack);

destroyStack(stack);

return 0;

}

**Очередь**

**Анализ задачи**

Ниже приведён пример реализации на C++ с комментариями. В решении используется собственная реализация очереди на базе связного списка. Функции реализуют:

* Создание очереди и добавление в неё элементов (enqueue).
* Вывод очереди с сообщением, если она пуста.
* Удаление KK элементов, начиная с заданной позиции.
* Добавление KK элементов в заданную позицию (сдвиг узлов).
* Запись очереди в файл и восстановление очереди из файла.
* Уничтожение очереди.

Код использует только библиотеки **iostream**, **fstream** и **stdio.h**.

#include <iostream>

#include <fstream>

#include <stdio.h>

using namespace std;

// Структура узла очереди

struct Node {

int key;

Node\* next;

};

// Функция для создания пустой очереди (возвращает указатель на голову)

Node\* createQueue() {

return nullptr;

}

// Функция добавления элемента в конец очереди

void enqueue(Node\*& head, int key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

} else {

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

}

// Функция вывода очереди. Если очередь пуста, выводится сообщение "Очередь пустая"

void printQueue(Node\* head) {

if (head == nullptr) {

cout << "Очередь пустая" << endl;

return;

}

Node\* temp = head;

while (temp != nullptr) {

cout << temp->key << " ";

temp = temp->next;

}

cout << endl;

}

// Функция удаления K элементов, начиная с позиции pos (позиция начинается с 1)

// Если pos выходит за диапазон, никаких изменений не производится

void removeElements(Node\*& head, int pos, int K) {

if (head == nullptr || pos < 1 || K <= 0) return;

// Удаляем элементы начиная с позиции pos.

// Для удобства используем указатель на указатель.

Node \*\*cur = &head;

int currentPos = 1;

// Перемещаемся до позиции pos-1

while (\*cur && currentPos < pos) {

cur = &((\*cur)->next);

currentPos++;

}

// Если достигли конца, выходим

if (\*cur == nullptr) return;

// Удаляем K элементов

for (int i = 0; i < K && \*cur; i++) {

Node\* temp = \*cur;

\*cur = (\*cur)->next;

delete temp;

}

}

// Функция добавления K элементов, начиная с позиции pos (позиция начинается с 1)

// Пользователь вводит ключи для новых элементов.

void addElements(Node\*& head, int pos, int K) {

if (K <= 0) return;

// Если pos == 1, нужно вставлять в начало очереди

if (pos <= 1) {

// Вставляем новые элементы перед head.

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

Node\* newNode = new Node;

newNode->key = key;

newNode->next = head;

head = newNode;

}

} else {

// Находим позицию pos-1

Node\* temp = head;

int currentPos = 1;

while (temp != nullptr && currentPos < pos - 1) {

temp = temp->next;

currentPos++;

}

// Если позиция больше длины очереди, добавляем в конец

if (temp == nullptr) {

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

enqueue(head, key);

}

} else {

// Вставляем K элементов после temp

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

Node\* newNode = new Node;

newNode->key = key;

newNode->next = temp->next;

temp->next = newNode;

// После вставки сдвигаем temp, чтобы следующий новый элемент вставлялся после уже вставленного

temp = newNode;

}

}

}

}

// Функция записи очереди в файл

void writeQueueToFile(Node\* head, const char\* filename) {

ofstream fout(filename);

if (!fout) {

cout << "Ошибка открытия файла для записи" << endl;

return;

}

// Сначала запишем количество элементов

int count = 0;

Node\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

fout << count << endl;

temp = head;

while (temp) {

fout << temp->key << " ";

temp = temp->next;

}

fout.close();

cout << "Очередь записана в файл " << filename << endl;

}

// Функция уничтожения очереди (освобождение памяти)

void destroyQueue(Node\*& head) {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

// Функция восстановления очереди из файла

Node\* readQueueFromFile(const char\* filename) {

ifstream fin(filename);

if (!fin) {

cout << "Ошибка открытия файла для чтения" << endl;

return nullptr;

}

int count;

fin >> count;

Node\* head = createQueue();

for (int i = 0; i < count; i++) {

int key;

fin >> key;

enqueue(head, key);

}

fin.close();

return head;

}

int main() {

Node\* queue = createQueue();

int n;

cout << "Сколько элементов добавить в начальную очередь? ";

cin >> n;

// Создаем очередь

for (int i = 0; i < n; i++) {

int key;

cout << "Введите ключ элемента " << i+1 << ": ";

cin >> key;

enqueue(queue, key);

}

cout << "Начальная очередь: ";

printQueue(queue);

// Выполняем удаление K элементов начиная с заданной позиции

int pos, K;

cout << "Введите позицию для удаления элементов (начиная с 1): ";

cin >> pos;

cout << "Введите количество удаляемых элементов: ";

cin >> K;

removeElements(queue, pos, K);

cout << "Очередь после удаления: ";

printQueue(queue);

// Добавляем K элементов начиная с заданной позиции

cout << "Введите позицию для добавления элементов (начиная с 1): ";

cin >> pos;

cout << "Введите количество добавляемых элементов: ";

cin >> K;

addElements(queue, pos, K);

cout << "Очередь после добавления: ";

printQueue(queue);

// Записываем очередь в файл

const char\* filename = "queue.txt";

writeQueueToFile(queue, filename);

// Уничтожаем очередь

destroyQueue(queue);

cout << "После уничтожения очереди: ";

printQueue(queue);

// Восстанавливаем очередь из файла

queue = readQueueFromFile(filename);

cout << "Восстановленная очередь: ";

printQueue(queue);

// Уничтожаем очередь

destroyQueue(queue);

return 0;

}

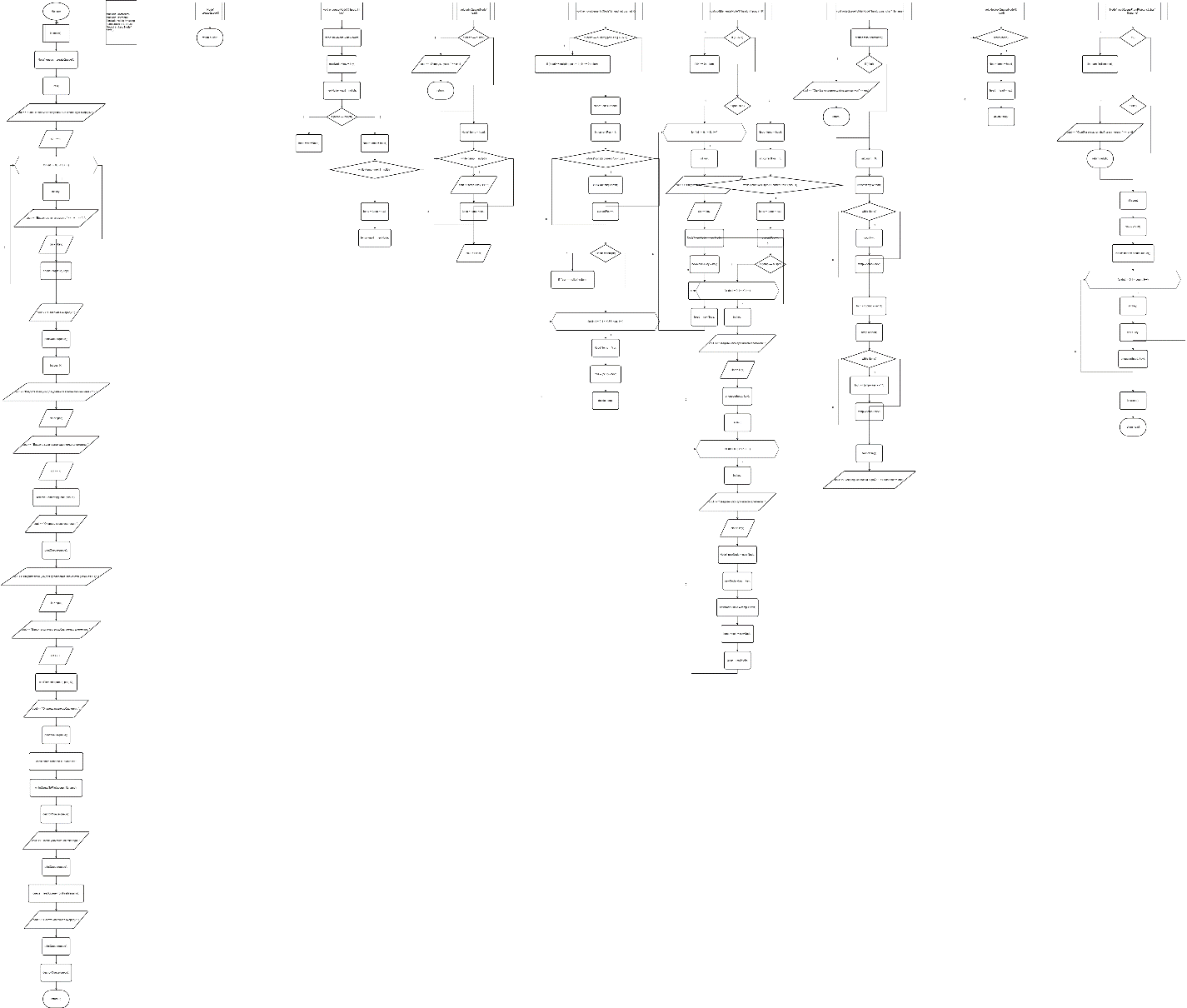
**Объяснение решения**

1. **Создание очереди и заполнение:**  
   В main() запрашивается количество начальных элементов, затем создаются узлы очереди с помощью функции enqueue.
2. **Вывод очереди:**  
   Функция printQueue выводит элементы очереди или сообщение "Очередь пустая", если очередь пуста.
3. **Удаление элементов:**  
   Функция removeElements принимает позицию и количество удаляемых элементов. Если позиция выходит за пределы очереди – никаких действий не производится.
4. **Добавление элементов:**  
   Функция addElements вставляет новые элементы начиная с указанной позиции. Если позиция меньше или равна 1, элементы добавляются в начало очереди. Если позиция превышает длину очереди, новые элементы добавляются в конец.
5. **Запись и восстановление из файла:**  
   Функция writeQueueToFile записывает в файл сначала количество элементов, затем сами ключи. Функция readQueueFromFile читает данные и формирует очередь.
6. **Уничтожение очереди:**  
   Функция destroyQueue последовательно удаляет все узлы, освобождая память.

После записи в файл очередь уничтожается, что демонстрируется вызовом printQueue (выводится сообщение о пустой очереди). Затем очередь восстанавливается из файла и выводится на экран.

Данное решение соответствует поставленным требованиям и использует только разрешённые библиотеки.

**Блок-схема**

****

**Код**

#include <iostream>

#include <fstream>

#include <stdio.h>

using namespace std;

struct Node {

int key;

Node\* next;

};

Node\* createQueue() {

return nullptr;

}

void enqueue(Node\*& head, int key) {

Node\* newNode = new Node;

newNode->key = key;

newNode->next = nullptr;

if (head == nullptr) {

head = newNode;

}

else {

Node\* temp = head;

while (temp->next != nullptr)

temp = temp->next;

temp->next = newNode;

}

}

void printQueue(Node\* head) {

if (head == nullptr) {

cout << "Очередь пустая" << endl;

return;

}

Node\* temp = head;

while (temp != nullptr) {

cout << temp->key << " ";

temp = temp->next;

}

cout << endl;

}

void removeElements(Node\*& head, int pos, int K) {

if (head == nullptr || pos < 1 || K <= 0) return;

Node\*\* cur = &head;

int currentPos = 1;

while (\*cur && currentPos < pos) {

cur = &((\*cur)->next);

currentPos++;

}

if (\*cur == nullptr) return;

for (int i = 0; i < K && \*cur; i++) {

Node\* temp = \*cur;

\*cur = (\*cur)->next;

delete temp;

}

}

void addElements(Node\*& head, int pos, int K) {

if (K <= 0) return;

if (pos <= 1) {

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

Node\* newNode = new Node;

newNode->key = key;

newNode->next = head;

head = newNode;

}

}

else {

Node\* temp = head;

int currentPos = 1;

while (temp != nullptr && currentPos < pos - 1) {

temp = temp->next;

currentPos++;

}

if (temp == nullptr) {

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

enqueue(head, key);

}

}

else {

for (int i = 0; i < K; i++) {

int key;

cout << "Введите ключ для нового элемента: ";

cin >> key;

Node\* newNode = new Node;

newNode->key = key;

newNode->next = temp->next;

temp->next = newNode;

temp = newNode;

}

}

}

}

void writeQueueToFile(Node\* head, const char\* filename) {

ofstream fout(filename);

if (!fout) {

cout << "Ошибка открытия файла для записи" << endl;

return;

}

int count = 0;

Node\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

fout << count << endl;

temp = head;

while (temp) {

fout << temp->key << " ";

temp = temp->next;

}

fout.close();

cout << "Очередь записана в файл " << filename << endl;

}

void destroyQueue(Node\*& head) {

while (head) {

Node\* temp = head;

head = head->next;

delete temp;

}

}

Node\* readQueueFromFile(const char\* filename) {

ifstream fin(filename);

if (!fin) {

cout << "Ошибка открытия файла для чтения" << endl;

return nullptr;

}

int count;

fin >> count;

Node\* head = createQueue();

for (int i = 0; i < count; i++) {

int key;

fin >> key;

enqueue(head, key);

}

fin.close();

return head;

}

int main() {

Node\* queue = createQueue();

int n;

cout << "Сколько элементов добавить в начальную очередь? ";

cin >> n;

for (int i = 0; i < n; i++) {

int key;

cout << "Введите ключ элемента " << i + 1 << ": ";

cin >> key;

enqueue(queue, key);

}

cout << "Начальная очередь: ";

printQueue(queue);

int pos, K;

cout << "Введите позицию для удаления элементов (начиная с 1): ";

cin >> pos;

cout << "Введите количество удаляемых элементов: ";

cin >> K;

removeElements(queue, pos, K);

cout << "Очередь после удаления: ";

printQueue(queue);

cout << "Введите позицию для добавления элементов (начиная с 1): ";

cin >> pos;

cout << "Введите количество добавляемых элементов: ";

cin >> K;

addElements(queue, pos, K);

cout << "Очередь после добавления: ";

printQueue(queue);

const char\* filename = "queue.txt";

writeQueueToFile(queue, filename);

destroyQueue(queue);

cout << "После уничтожения очереди: ";

printQueue(queue);

queue = readQueueFromFile(filename);

cout << "Восстановленная очередь: ";

printQueue(queue);

destroyQueue(queue);

return 0;

}