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 "Информационные динамические структуры"

Выполнил студент гр. РИС-24-3б

Жиряков Леонид Антонович

Проверил:

Доцент кафедры ИТАС   
Ольга Андреевна Полякова

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (оценка) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

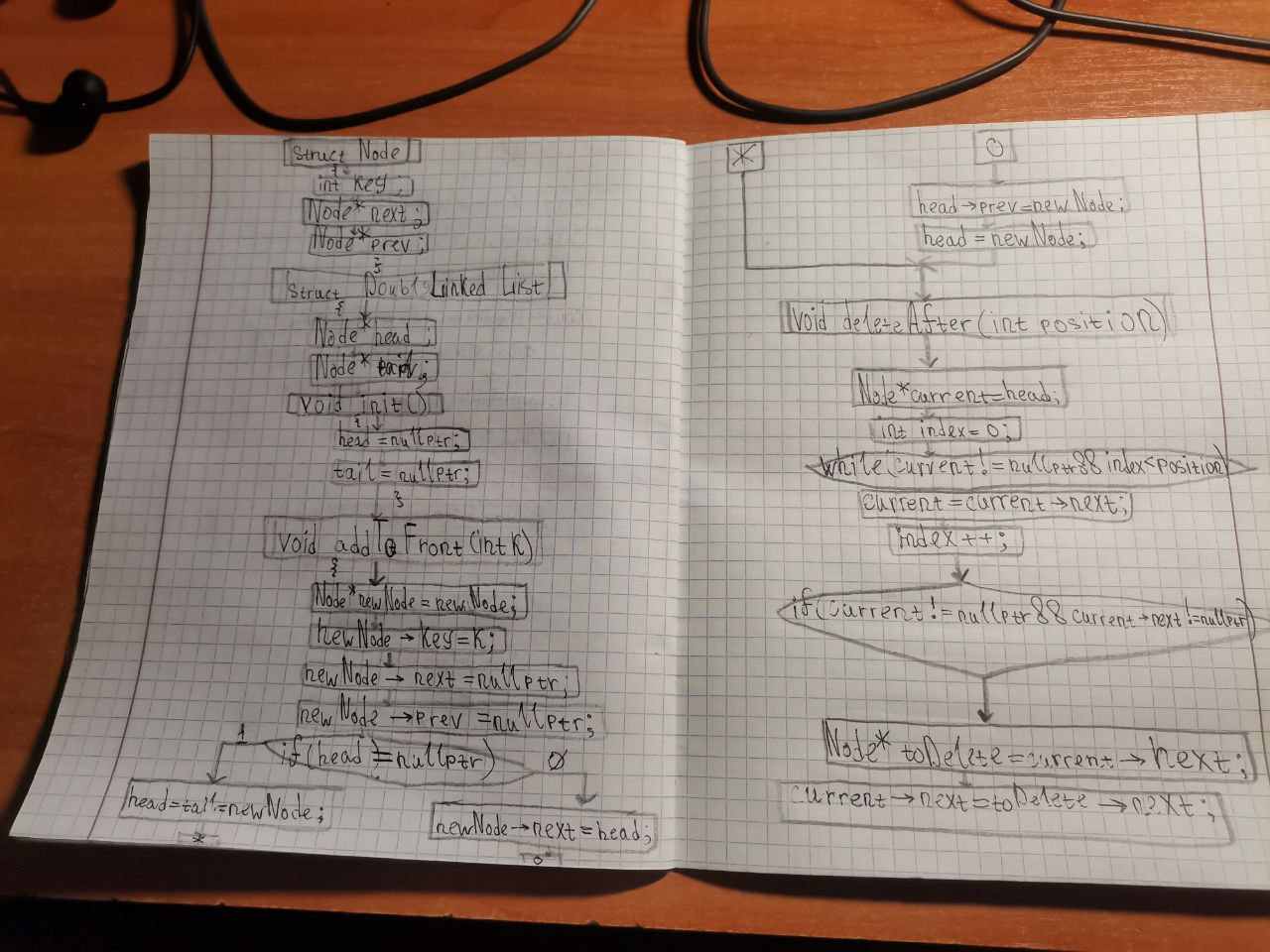
г. Пермь, 2024

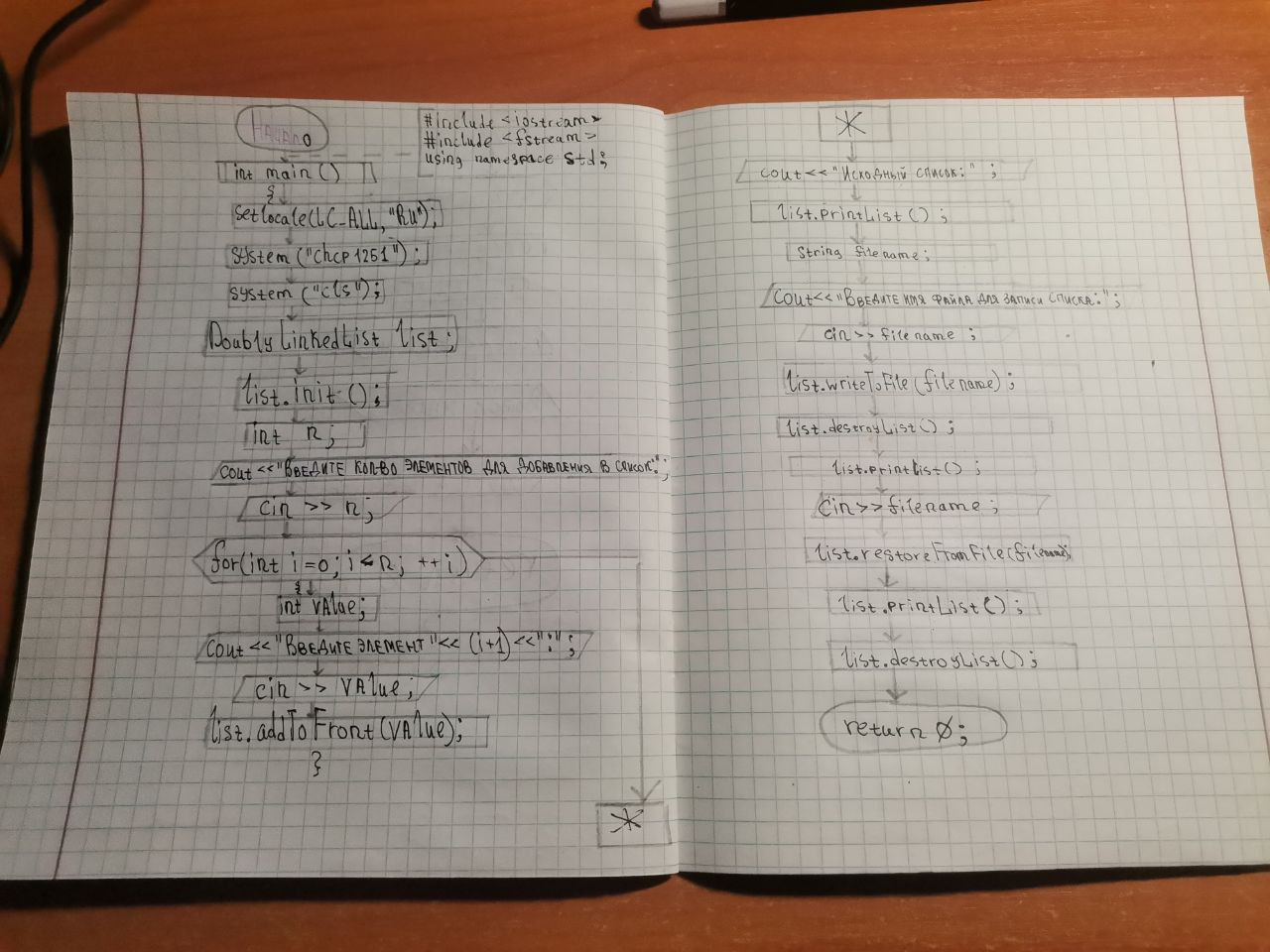
Вариант 8.

Постановка задачи:

Записи в линейном списке содержат ключевое поле типа int. Сформировать двунаправленный список. Удалить из него элемент после элемента с заданным номером, добавить К элементов в начало списка.

Блок-схема:





Код:

#include <iostream>

#include <fstream>

using namespace std;

// Структура для узла двунаправленного списка

struct Node

{

int key; // Ключевое поле

Node\* next; // Указатель на следующий узел

Node\* prev; // Указатель на предыдущий узел

};

// Структура для двунаправленного списка

struct DoublyLinkedList

{

Node\* head; // Указатель на голову списка

Node\* tail; // Указатель на хвост списка

// Инициализация списка

void init()

{

head = nullptr;

tail = nullptr;

}

// Функция для добавления элемента в начало списка

void addToFront(int k)

{

Node\* newNode = new Node; // Создаем новый узел

newNode->key = k;

newNode->next = nullptr;

newNode->prev = nullptr;

if (head == nullptr)

{

head = tail = newNode; // Если список пуст

}

else

{

newNode->next = head;

head->prev = newNode;

head = newNode;

}

}

// Функция для удаления элемента после элемента с заданным номером

void deleteAfter(int position)

{

Node\* current = head;

int index = 0;

// Поиск узла с заданным номером

while (current != nullptr && index < position)

{

current = current->next;

index++;

}

// Если узел найден и у него есть следующий узел

if (current != nullptr && current->next != nullptr)

{

Node\* toDelete = current->next;

current->next = toDelete->next; // Удаляем следующий узел

if (toDelete->next != nullptr)

{

toDelete->next->prev = current; // Обновляем указатель на предыдущий узел

}

else

{

tail = current; // Если удаляем последний узел, обновляем хвост

}

delete toDelete; // Освобождаем память

}

else

{

cout << "Нет элемента для удаления после узла с номером " << position << endl;

}

}

// Функция для печати списка

void printList()

{

if (head == nullptr)

{

cout << "Список пустой." << endl;

return;

}

Node\* current = head;

while (current != nullptr)

{

cout << current->key << " ";

current = current->next;

}

cout << endl;

}

// Функция для записи списка в файл

void writeToFile(const string& filename)

{

ofstream outFile(filename);

if (!outFile) {

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

return;

}

Node\* current = head;

while (current != nullptr)

{

outFile << current->key << endl; // Записываем каждый элемент в новую строку

current = current->next;

}

outFile.close();

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

}

// Функция для уничтожения списка

void destroyList()

{

Node\* current = head;

while (current != nullptr)

{

Node\* nextNode = current->next; // Сохраняем указатель на следующий узел

delete current; // Освобождаем память текущего узла

current = nextNode; // Переходим к следующему узлу

}

head = nullptr; // Обнуляем указатель на голову

tail = nullptr; // Обнуляем указатель на хвост

}

// Функция для восстановления списка из файла

void restoreFromFile(const string& filename)

{

ifstream inFile(filename);

if (!inFile)

{

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

return;

}

int value;

while (inFile >> value)

{

addToFront(value); // Добавляем элементы в начало списка

}

inFile.close();

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

}

};

int main()

{

setlocale(LC\_ALL, "RU");

system("chcp 1251");

system("cls");

DoublyLinkedList list;

list.init(); // Инициализация списка

// Создание списка и добавление элементов

int n; // Количество элементов для добавления

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

cin >> n;

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

{

int value;

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

cin >> value;

list.addToFront(value); // Добавляем элементы в начало списка

}

// Печать исходного списка

cout << "Исходный список: ";

list.printList();

// Удаление элемента после заданной позиции

int position;

cout << "Введите позицию узла для удаления следующего элемента: ";

cin >> position;

list.deleteAfter(position);

cout << "Список после удаления: ";

list.printList();

// Добавление K элементов в начало списка

int K;

cout << "Введите количество элементов для добавления в начало списка: ";

cin >> K;

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

{

int value;

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

cin >> value;

list.addToFront(value); // Добавляем элементы в начало списка

}

// Печать списка после добавления

cout << "Список после добавления элементов в начало: ";

list.printList();

// Запись списка в файл

string filename;

cout << "Введите имя файла для записи списка: ";

cin >> filename;

list.writeToFile(filename);

// Уничтожение списка

list.destroyList();

cout << "Список уничтожен." << endl;

// Попытка печати списка после уничтожения

cout << "Список после уничтожения: ";

list.printList();

// Восстановление списка из файла

cout << "Введите имя файла для восстановления списка: ";

cin >> filename;

list.restoreFromFile(filename);

// Печать восстановленного списка

cout << "Восстановленный список: ";

list.printList();

// Уничтожение списка перед завершением программы

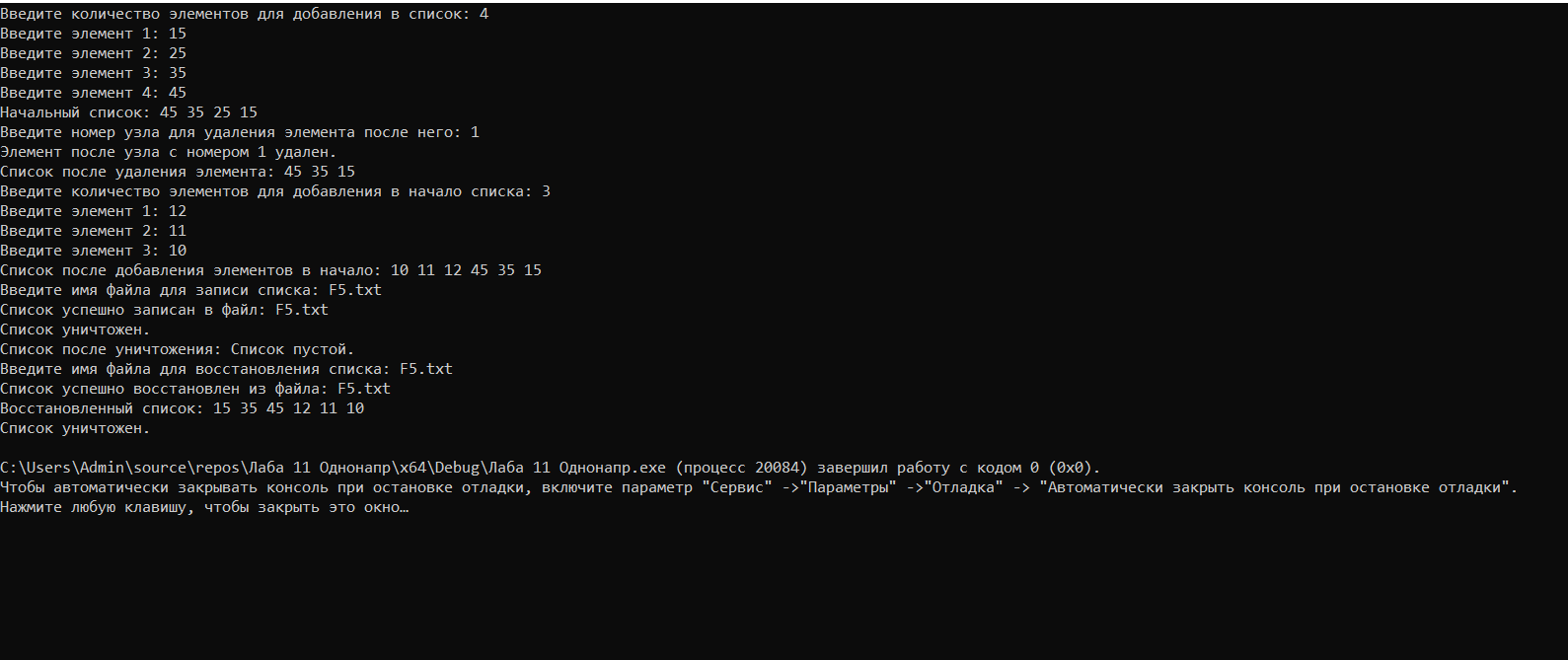
list.destroyList();

cout << "Список уничтожен." << endl;

return 0; // Завершение программы

}

Вывод:



Код(Однонаправленный):

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

// Структура для узла однонаправленного списка

struct Node

{

int key; // Ключевое поле

Node\* next; // Указатель на следующий узел

};

// Структура для однонаправленного списка

struct SinglyLinkedList

{

Node\* head; // Указатель на голову списка

// Функция для инициализации списка

void init()

{

head = nullptr; // Устанавливаем голову списка в nullptr

}

// Функция для создания списка и добавления элементов

void createList(int n)

{

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

{

int value;

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

cin >> value;

addToFront(value); // Добавляем элементы в начало списка

}

}

// Функция для печати списка

void printList()

{

if (head == nullptr)

{

cout << "Список пустой." << endl;

return;

}

Node\* current = head;

while (current != nullptr)

{

cout << current->key << " ";

current = current->next;

}

cout << endl;

}

// Функция для удаления элемента после узла с заданным номером

void deleteAfter(int position)

{

Node\* current = head;

int index = 0;

// Поиск узла с заданным номером

while (current != nullptr && index < position)

{

current = current->next;

index++;

}

// Если узел найден и у него есть следующий узел

if (current != nullptr && current->next != nullptr)

{

Node\* toDelete = current->next; // Узел для удаления

current->next = toDelete->next; // Удаляем следующий узел

delete toDelete; // Освобождаем память

cout << "Элемент после узла с номером " << position << " удален." << endl;

}

else

{

cout << "Нет элемента для удаления после узла с номером " << position << endl;

}

}

// Функция для добавления элемента в начало списка

void addToFront(int k)

{

Node\* newNode = new Node; // Создаем новый узел

newNode->key = k; // Устанавливаем значение

newNode->next = head; // Новый узел указывает на старую голову

head = newNode; // Обновляем голову

}

// Функция для записи списка в файл

void writeToFile(const string& filename)

{

ofstream outFile(filename);

if (!outFile)

{

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

return;

}

Node\* current = head;

while (current != nullptr)

{

outFile << current->key << endl; // Записываем каждый элемент в новую строку

current = current->next;

}

outFile.close();

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

}

// Функция для уничтожения списка

void destroyList()

{

Node\* current = head;

while (current != nullptr)

{

Node\* nextNode = current->next; // Сохраняем указатель на следующий узел

delete current; // Освобождаем память текущего узла

current = nextNode; // Переходим к следующему узлу

}

head = nullptr; // Обнуляем указатель на голову

}

// Функция для восстановления списка из файла

void restoreFromFile(const string& filename)

{

ifstream inFile(filename);

if (!inFile)

{

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

return;

}

int value;

while (inFile >> value)

{

addToFront(value); // Добавляем элементы в начало списка

}

inFile.close();

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

}

};

int main()

{

setlocale(LC\_ALL, "RU");

system("chcp 1251");

system("cls");

SinglyLinkedList list;

list.init(); // Инициализация списка

// Создание списка

int L;

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

cin >> L;

list.createList(L);

// Печать начального списка

cout << "Начальный список: ";

list.printList();

// Удаление элемента после заданного номера

int position;

cout << "Введите номер узла для удаления элемента после него: ";

cin >> position;

list.deleteAfter(position);

// Печать списка после удаления

cout << "Список после удаления элемента: ";

list.printList();

// Добавление K элементов в начало списка

int K;

cout << "Введите количество элементов для добавления в начало списка: ";

cin >> K;

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

{

int value;

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

cin >> value;

list.addToFront(value);

}

// Печать списка после добавления

cout << "Список после добавления элементов в начало: ";

list.printList();

// Запись списка в файл

string filename;

cout << "Введите имя файла для записи списка: ";

cin >> filename;

list.writeToFile(filename);

// Уничтожение списка

list.destroyList();

cout << "Список уничтожен." << endl;

// Попытка печати списка после уничтожения

cout << "Список после уничтожения: ";

list.printList();

// Восстановление списка из файла

cout << "Введите имя файла для восстановления списка: ";

cin >> filename;

list.restoreFromFile(filename);

// Печать восстановленного списка

cout << "Восстановленный список: ";

list.printList();

// Уничтожение списка перед завершением программы

list.destroyList();

cout << "Список уничтожен." << endl;

return 0;

}

Код(Стеки):

#include <iostream>

#include <fstream> // Для работы с файлами

using namespace std;

// Структура для узла стека

struct Node

{

int key; // Ключевое поле

Node\* next; // Указатель на следующий узел

};

// Структура для стека на основе односвязного списка

struct Stack

{

Node\* top; // Указатель на верхний элемент стека

// Функция для инициализации стека

void init()

{

top = nullptr; // Устанавливаем верхний элемент в nullptr

}

// Функция для добавления элемента на верх стека

void push(int k)

{

Node\* newNode = new Node; // Создаем новый узел

newNode->key = k; // Устанавливаем значение

newNode->next = top; // Новый узел указывает на текущий верхний элемент

top = newNode; // Обновляем верхний элемент

}

// Функция для удаления элемента после узла с заданным номером

void deleteAfter(int position)

{

Node\* current = top;

int index = 0;

// Поиск узла с заданным номером

while (current != nullptr && index < position)

{

current = current->next;

index++;

}

// Если узел найден и у него есть следующий узел

if (current != nullptr && current->next != nullptr)

{

Node\* toDelete = current->next; // Узел для удаления

current->next = toDelete->next; // Удаляем следующий узел

delete toDelete; // Освобождаем память

cout << "Элемент после узла с номером " << position << " удален." << endl;

}

else

{

cout << "Нет элемента для удаления после узла с номером " << position << endl;

}

}

// Функция для добавления K элементов в стек

void addMultipleToFront(int K)

{

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

{

int value;

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

cin >> value;

push(value); // Добавляем элемент на верх стека

}

}

// Функция для печати стека

void printStack()

{

if (top == nullptr)

{

cout << "Стек пустой." << endl;

return;

}

Node\* current = top;

while (current != nullptr)

{

cout << current->key << " ";

current = current->next;

}

cout << endl;

}

// Функция для очистки стека

void clear()

{

while (top != nullptr)

{

Node\* temp = top;

top = top->next;

delete temp; // Освобождение памяти

}

cout << "Стек очищен." << endl;

}

// Функция для сохранения стека в файл

void saveToFile(const string& filename)

{

ofstream outFile(filename);

if (!outFile)

{

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

return;

}

Node\* current = top;

while (current != nullptr)

{

outFile << current->key << endl; // Записываем каждый элемент в файл

current = current->next;

}

outFile.close();

cout << "Стек сохранен в файл " << filename << "." << endl;

}

// Функция для восстановления стека из файла

void loadFromFile(const string& filename)

{

ifstream inFile(filename);

if (!inFile)

{

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

return;

}

clear(); // Очищаем текущий стек перед загрузкой

int value;

while (inFile >> value) // Читаем значения из файла

{

push(value); // Добавляем элементы в стек

}

inFile.close();

cout << "Стек восстановлен из файла " << filename << "." << endl;

}

};

int main()

{

Stack stack;

stack.init(); // Инициализация стека

setlocale(LC\_ALL, "RU");

system("chcp 1251");

system("cls");

// Заполнение стека

int n;

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

cin >> n;

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

{

int value;

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

cin >> value;

stack.push(value);

}

// Печать стека

cout << "Текущий стек: ";

stack.printStack();

// Удаление элемента после заданного номера

int position;

cout << "Введите номер узла для удаления элемента после него: ";

cin >> position;

stack.deleteAfter(position);

// Печать стека после удаления

cout << "Стек после удаления элемента: ";

stack.printStack();

// Добавление K элементов в начало стека

int K;

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

cin >> K;

stack.addMultipleToFront(K);

// Печать стека после добавления

cout << "Стек после добавления элементов: ";

stack.printStack();

// Запись стека в файл

string filename;

cout << "Введите имя файла для записи стека: ";

cin >> filename;

stack.saveToFile(filename);

// Очистка стека

stack.clear();

// Печать стека после очистки

cout << "Стек после очистки: ";

stack.printStack();

// Восстановление стека из файла

cout << "Введите имя файла для восстановления стека: ";

cin >> filename;

stack.loadFromFile(filename);

// Печать восстановленного стека

cout << "Восстановленный стек: ";

stack.printStack();

// Очистка стека перед завершением программы

stack.clear();

return 0;

}

Код(Очередь):

#include <iostream>

#include <fstream> // Для работы с файлами

using namespace std;

// Структура для узла очереди

struct Node

{

int key; // Ключевое поле

Node\* next; // Указатель на следующий узел

Node(int k) : key(k), next(nullptr) {} // Конструктор

};

// Структура для очереди на основе односвязного списка

struct Queue

{

Node\* front; // Указатель на первый элемент очереди

Node\* rear; // Указатель на последний элемент очереди

Queue() : front(nullptr), rear(nullptr) {} // Конструктор

// Функция для добавления элемента в конец очереди

void enqueue(int k)

{

Node\* newNode = new Node(k);

if (rear)

{

rear->next = newNode; // Указываем, что следующий элемент - новый узел

}

else

{

front = newNode; // Если очередь была пустой, новый узел - это и front, и rear

}

rear = newNode; // Обновляем rear

}

// Функция для удаления элемента после узла с заданным номером

void deleteAfter(int position)

{

Node\* current = front;

int index = 0;

// Поиск узла с заданным номером

while (current != nullptr && index < position)

{

current = current->next;

index++;

}

// Если узел найден и у него есть следующий узел

if (current != nullptr && current->next != nullptr)

{

Node\* toDelete = current->next; // Узел для удаления

current->next = toDelete->next; // Удаляем следующий узел

if (toDelete == rear)

{ // Если удаляем последний элемент

rear = current; // Обновляем rear

}

delete toDelete; // Освобождаем память

cout << "Элемент после узла с номером " << position << " удален." << endl;

}

else

{

cout << "Нет элемента для удаления после узла с номером " << position << endl;

}

}

// Функция для добавления K элементов в начало очереди

void addMultipleToFront(int K)

{

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

{

int value;

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

cin >> value;

Node\* newNode = new Node(value);

newNode->next = front; // Новый узел указывает на текущий front

front = newNode; // Обновляем front

if (rear == nullptr) { // Если очередь была пустой, обновляем rear

rear = newNode;

}

}

}

// Функция для печати очереди

void printQueue()

{

if (front == nullptr)

{

cout << "Очередь пустая." << endl;

return;

}

Node\* current = front;

while (current != nullptr)

{

cout << current->key << " ";

current = current->next;

}

cout << endl;

}

// Функция для сохранения очереди в файл

void saveToFile(const string& filename)

{

ofstream outFile(filename);

if (!outFile)

{

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

return;

}

Node\* current = front;

while (current != nullptr)

{

outFile << current->key << endl; // Сохраняем каждый элемент в файл

current = current->next;

}

outFile.close();

cout << "Очередь успешно сохранена в файл " << filename << "." << endl;

}

// Функция для удаления очереди

void clear()

{

while (front != nullptr)

{

Node\* temp = front;

front = front->next;

delete temp; // Освобождение памяти

}

rear = nullptr; // Обновляем rear

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

}

// Функция для восстановления очереди из файла

void restoreFromFile(const string& filename)

{

ifstream inFile(filename);

if (!inFile)

{

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

return;

}

clear(); // Сначала очищаем очередь, чтобы избежать дублирования

int value;

while (inFile >> value)

{

enqueue(value); // Добавляем элементы в очередь

}

inFile.close();

cout << "Очередь успешно восстановлена из файла " << filename << "." << endl;

}

// Деструктор для освобождения памяти

~Queue()

{

clear(); // Освобождение памяти при уничтожении очереди

}

};

int main()

{

Queue queue;

setlocale(LC\_ALL, "RU");

system("chcp 1251");

system("cls");

// Заполнение очереди

int n;

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

cin >> n;

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

{

int value;

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

cin >> value;

queue.enqueue(value);

}

// Печать очереди

cout << "Текущая очередь: ";

queue.printQueue();

// Удаление элемента после заданного номера

int position;

cout << "Введите номер узла для удаления элемента после него: ";

cin >> position;

queue.deleteAfter(position);

// Печать очереди после удаления

cout << "Очередь после удаления элемента: ";

queue.printQueue();

// Добавление K элементов в начало очереди

int K;

cout << "Введите количество элементов для добавления в начало очереди: ";

cin >> K;

queue.addMultipleToFront(K);

// Печать очереди после добавления

cout << "Очередь после добавления элементов в начало: ";

queue.printQueue();

// Сохранение очереди в файл

string filename = "F5.txt";

queue.saveToFile(filename);

// Очистка очереди

queue.clear();

// Печать очереди после очистки

cout << "Очередь после очистки элементов: ";

queue.printQueue();

// Восстановление очереди из файла

queue.restoreFromFile(filename);

// Печать восстановленной очереди

cout << "Восстановленная очередь: ";

queue.printQueue();

return 0; // Завершение программы

}