Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

ОТЧЁТ ПО ЛАБОЛАТОРНОЙ РАБОТЕ №11

Тема: «Информационные динамические структуры»

Выполнил

Студент РИС-24-1б

Конькова С. С.

Проверил

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2025

**Постановка задачи**

Написать программу, в которой создаются динамические структуры и выполнить их обработку в соответствии со своим вариантом. Использовать однонаправленный список, двунаправленный список, стек, очередь.

Для каждого вариант разработать следующие функции:

1. Создание списка.

2. Добавление элемента в список.

3. Удаление элемента из списка.

4. Печать списка.

5. Запись списка в файл.

6. Уничтожение списка.

7. Восстановление списка из файла.

**Анализ**

1. Написать функцию для создания списка. Функция может создавать пустой список, а затем добавлять в него элементы.

2. Написать функцию для печати списка. Функция должна предусматривать вывод сообщения, если список пустой.

3. Написать функции для удаления и добавления элементов списка в соответствии со своим вариантом.

4. Выполнить изменения в списке и печать списка после каждого изменения.

5. Написать функцию для записи списка в файл.

6. Написать функцию для уничтожения списка.

7. Записать список в файл, уничтожить его и выполнить печать (при печати должно быть выдано сообщение "Список пустой").

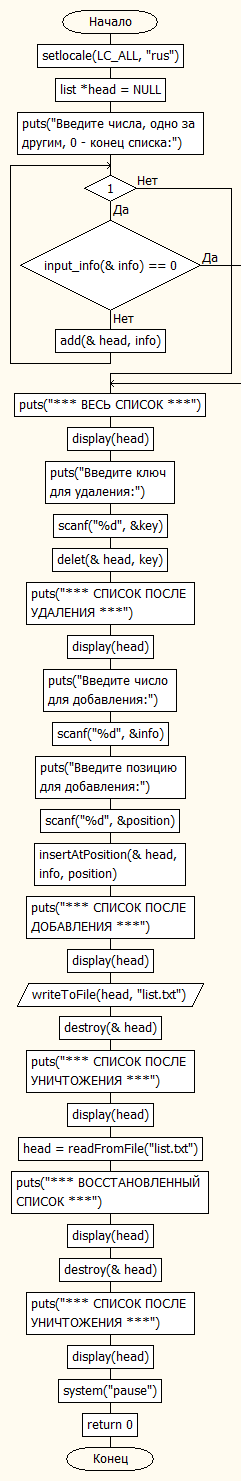
8. Написать функцию для восстановления списка из файла.

9. Восстановить список и распечатать его.

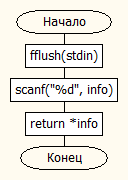
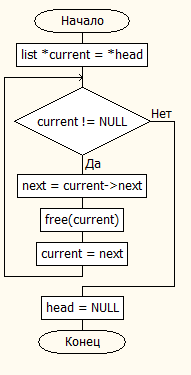
10. Уничтожить список.

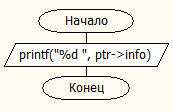
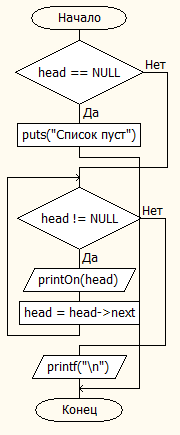
**11 вариант**

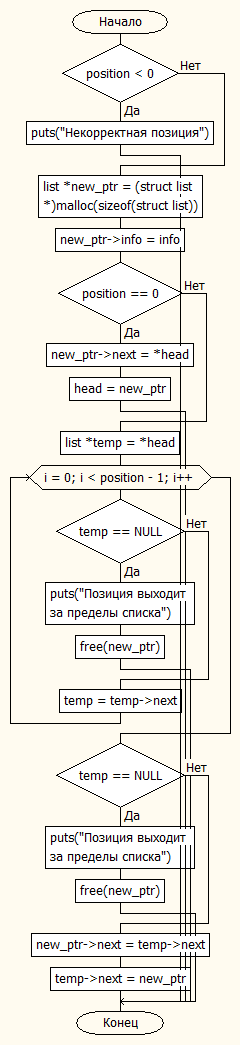
Записи в линейном списке содержат ключевое поле типа \*char(строка символов). Сформировать двунаправленный список. Удалить из него элемент с заданным ключом, добавить элемент с указанным номером.

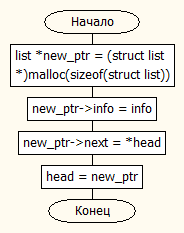
****

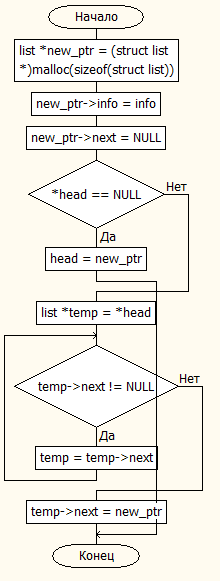
**Блок – схемы**

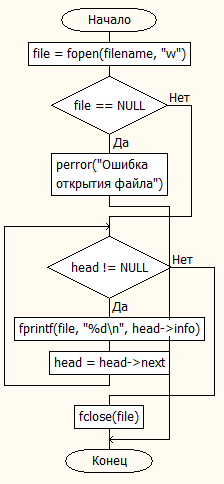
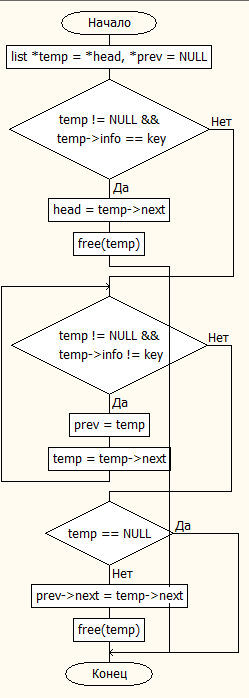
**Однонаправленный список**

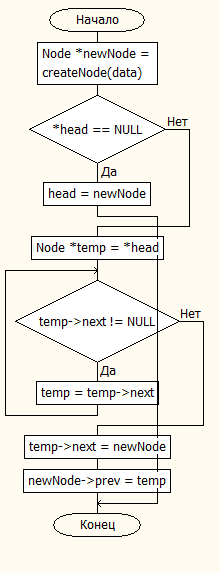
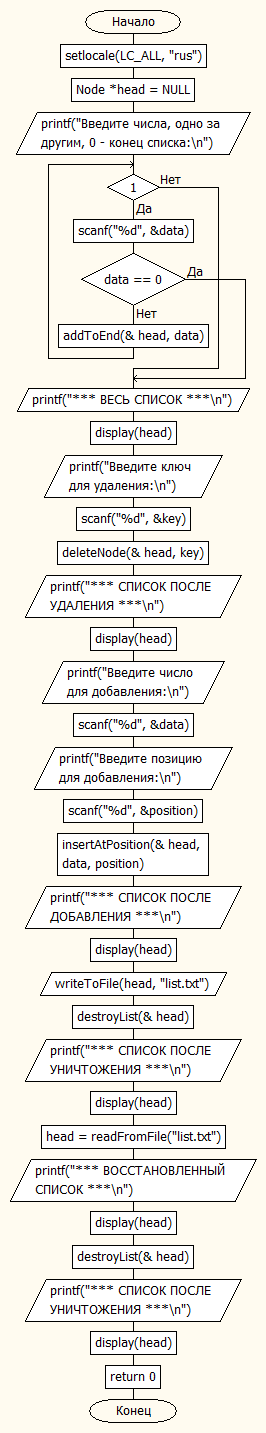
****

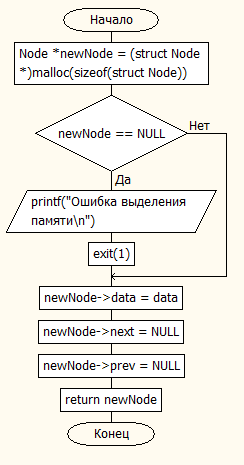
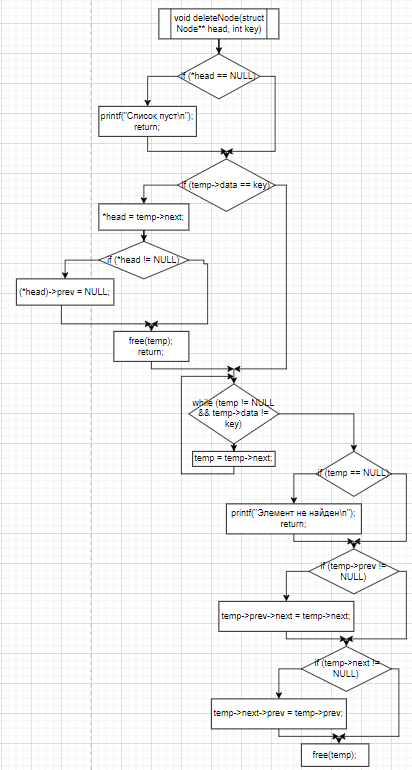
****

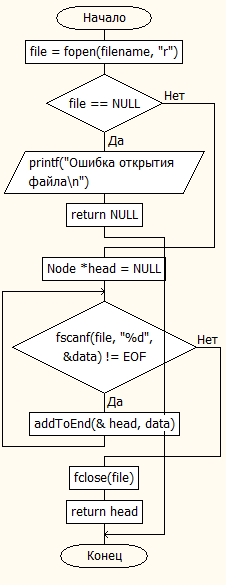
****

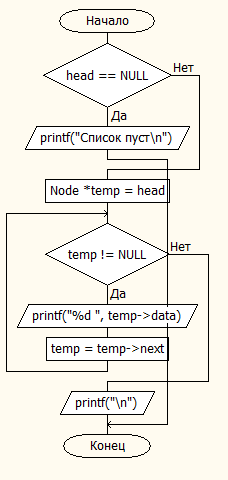
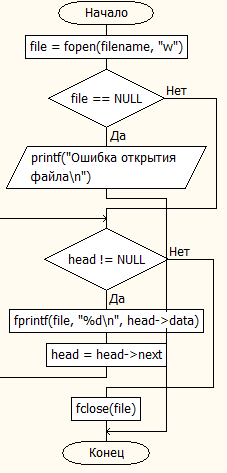
****

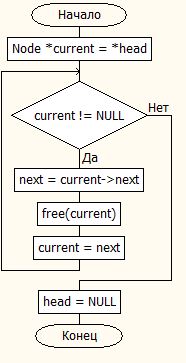
****

**Двунаправленный список**

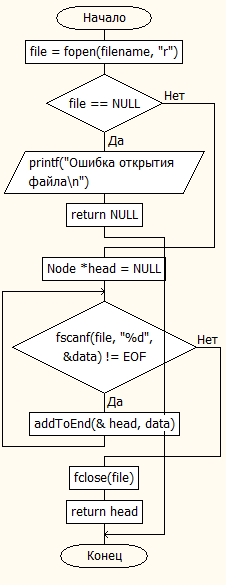
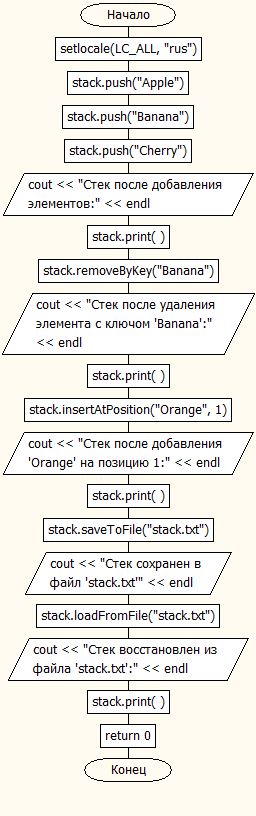
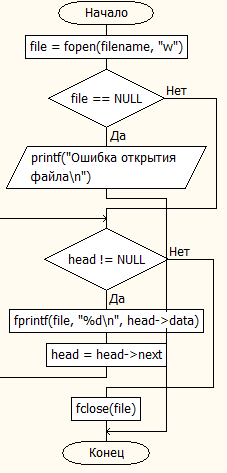
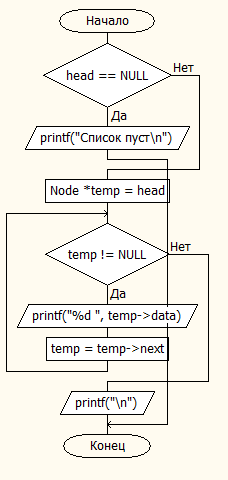
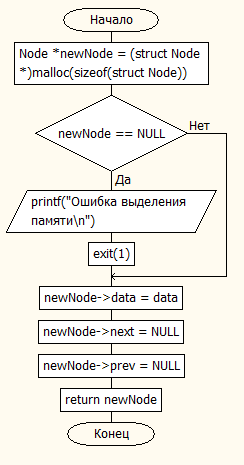
****

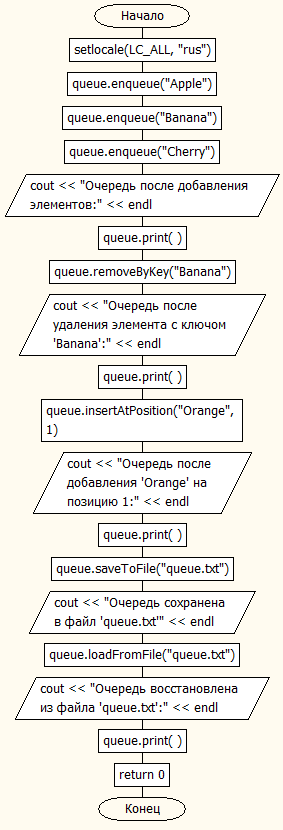
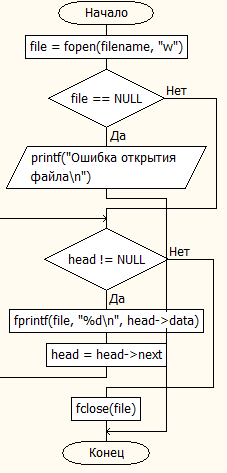
****

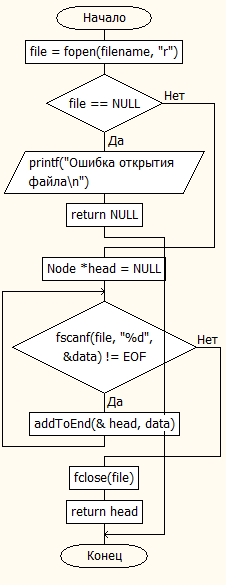
****

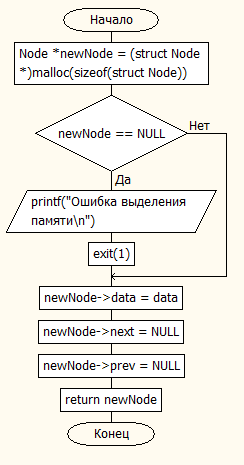
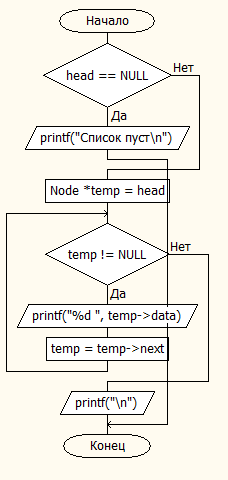
****

**Стек**

****

**Очередь**

****

****

**Коды программ**

**Однонаправленный список**

#include <stdio.h>

#include <stdlib.h>

#include <conio.h>

#include <windows.h>

#include <clocale>

struct list {

struct list\* next;

int info;

};

// Функция для ввода информации

int input\_info(int\* info) {

fflush(stdin);

scanf("%d", info);

return \*info;

}

// Функция для печати элемента списка

void printOn(struct list\* ptr) {

printf("%d ", ptr->info);

}

// Функция для отображения всего списка

void display(struct list\* head) {

if (head == NULL) {

puts("Список пуст");

return;

}

while (head != NULL) {

printOn(head);

head = head->next;

}

printf("\n");

}

// Функция для уничтожения списка

void destroy(struct list\*\* head) {

struct list\* current = \*head;

struct list\* next;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

// Функция для добавления элемента в начало списка

void add(struct list\*\* head, int info) {

struct list\* new\_ptr = (struct list\*)malloc(sizeof(struct list));

new\_ptr->info = info;

new\_ptr->next = \*head;

\*head = new\_ptr;

}

void addToEnd(struct list\*\* head, int info) {

struct list\* new\_ptr = (struct list\*)malloc(sizeof(struct list));

new\_ptr->info = info;

new\_ptr->next = NULL;

if (\*head == NULL) {

\*head = new\_ptr;

return;

}

struct list\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = new\_ptr;

}

// Функция для удаления элемента из списка по значению

void delet(struct list\*\* head, int key) {

struct list\* temp = \*head, \*prev = NULL;

if (temp != NULL && temp->info == key) {

\*head = temp->next;

free(temp);

return;

}

while (temp != NULL && temp->info != key) {

prev = temp;

temp = temp->next;

}

if (temp == NULL) return;

prev->next = temp->next;

free(temp);

}

// Функция для вставки элемента на указанную позицию

void insertAtPosition(struct list\*\* head, int info, int position) {

if (position < 0) {

puts("Некорректная позиция");

return;

}

struct list\* new\_ptr = (struct list\*)malloc(sizeof(struct list));

new\_ptr->info = info;

if (position == 0) {

new\_ptr->next = \*head;

\*head = new\_ptr;

return;

}

struct list\* temp = \*head;

for (int i = 0; i < position - 1; i++) {

if (temp == NULL) {

puts("Позиция выходит за пределы списка");

free(new\_ptr);

return;

}

temp = temp->next;

}

if (temp == NULL) {

puts("Позиция выходит за пределы списка");

free(new\_ptr);

return;

# }

new\_ptr->next = temp->next;

temp->next = new\_ptr;

}

// Функция для записи списка в файл

void writeToFile(struct list\* head, const char\* filename) {

FILE\* file = fopen(filename, "w");

if (file == NULL) {

perror("Ошибка открытия файла");

return;

}

while (head != NULL) {

fprintf(file, "%d\n", head->info);

head = head->next;

}

fclose(file);

}

// Функция для восстановления списка из файла

struct list\* readFromFile(const char\* filename) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

perror("Ошибка открытия файла");

return NULL;

}

struct list\* head = NULL;

int info;

while (fscanf(file, "%d", &info) != EOF) {

addToEnd(&head, info); // Используем addToEnd для правильного порядка

}

fclose(file);

return head;

}

int main(void) {

setlocale(LC\_ALL, "rus");

struct list\* head = NULL;

int info, key, position;

char ch;

// Создание списка

puts("Введите числа, одно за другим, 0 - конец списка:");

while (1) {

if (input\_info(&info) == 0) {

break; // Ввод завершен, если введен 0

}

add(&head, info);

}

// Печать списка

puts("\*\*\* ВЕСЬ СПИСОК \*\*\*");

display(head);

// Удаление элемента

puts("Введите ключ для удаления:");

scanf("%d", &key);

delet(&head, key);

puts("\*\*\* СПИСОК ПОСЛЕ УДАЛЕНИЯ \*\*\*");

display(head);

// Добавление элемента на указанную позицию

puts("Введите число для добавления:");

scanf("%d", &info);

puts("Введите позицию для добавления:");

scanf("%d", &position);

insertAtPosition(&head, info, position);

puts("\*\*\* СПИСОК ПОСЛЕ ДОБАВЛЕНИЯ \*\*\*");

display(head);

// Запись списка в файл

writeToFile(head, "list.txt");

// Уничтожение списка

destroy(&head);

puts("\*\*\* СПИСОК ПОСЛЕ УНИЧТОЖЕНИЯ \*\*\*");

display(head);

// Восстановление списка из файла

head = readFromFile("list.txt");

puts("\*\*\* ВОССТАНОВЛЕННЫЙ СПИСОК \*\*\*");

display(head);

// Уничтожение списка

destroy(&head);

puts("\*\*\* СПИСОК ПОСЛЕ УНИЧТОЖЕНИЯ \*\*\*");

display(head);

system("pause");

return 0;

}

**Двунаправленный список**

#include <stdio.h>

#include <stdlib.h>

#include <locale.h>

// Структура для двунаправленного списка

struct Node {

int data;

struct Node\* next;

struct Node\* prev;

};

// Функция для создания нового узла

struct Node\* createNode(int data) {

struct Node\* newNode = (struct Node\*)malloc(sizeof(struct Node));

if (newNode == NULL) {

printf("Ошибка выделения памяти\n");

exit(1);

}

newNode->data = data;

newNode->next = NULL;

newNode->prev = NULL;

return newNode;

}

// Функция для добавления элемента в конец списка

void addToEnd(struct Node\*\* head, int data) {

struct Node\* newNode = createNode(data);

if (\*head == NULL) {

\*head = newNode;

return;

}

struct Node\* temp = \*head;

while (temp->next != NULL) {

temp = temp->next;

}

temp->next = newNode;

newNode->prev = temp;

}

// Функция для отображения списка

void display(struct Node\* head) {

if (head == NULL) {

printf("Список пуст\n");

return;

}

struct Node\* temp = head;

while (temp != NULL) {

printf("%d ", temp->data);

temp = temp->next;

}

printf("\n");

}

// Функция для удаления элемента по значению

void deleteNode(struct Node\*\* head, int key) {

if (\*head == NULL) {

printf("Список пуст\n");

return;

}

struct Node\* temp = \*head;

// Если удаляемый элемент — первый

if (temp->data == key) {

\*head = temp->next;

if (\*head != NULL) {

(\*head)->prev = NULL;

}

free(temp);

return;

}

// Поиск элемента для удаления

while (temp != NULL && temp->data != key) {

temp = temp->next;

}

// Если элемент не найден

if (temp == NULL) {

printf("Элемент не найден\n");

return;

}

// Если элемент найден, перестраиваем связи

if (temp->prev != NULL) {

temp->prev->next = temp->next;

}

if (temp->next != NULL) {

temp->next->prev = temp->prev;

}

free(temp);

}

// Функция для вставки элемента на указанную позицию

void insertAtPosition(struct Node\*\* head, int data, int position) {

if (position < 0) {

printf("Некорректная позиция\n");

return;

}

struct Node\* newNode = createNode(data);

// Вставка в начало

if (position == 0) {

newNode->next = \*head;

if (\*head != NULL) {

(\*head)->prev = newNode;

}

\*head = newNode;

return;

}

// Вставка в середину или конец

struct Node\* temp = \*head;

for (int i = 0; i < position - 1; i++) {

if (temp == NULL) {

printf("Позиция выходит за пределы списка\n");

free(newNode);

return;

}

temp = temp->next;

}

if (temp == NULL) {

printf("Позиция выходит за пределы списка\n");

free(newNode);

return;

}

newNode->next = temp->next;

newNode->prev = temp;

if (temp->next != NULL) {

temp->next->prev = newNode;

}

temp->next = newNode;

}

// Функция для уничтожения списка

void destroyList(struct Node\*\* head) {

struct Node\* current = \*head;

struct Node\* next;

while (current != NULL) {

next = current->next;

free(current);

current = next;

}

\*head = NULL;

}

// Функция для записи списка в файл

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

FILE\* file = fopen(filename, "w");

if (file == NULL) {

printf("Ошибка открытия файла\n");

return;

}

while (head != NULL) {

fprintf(file, "%d\n", head->data);

head = head->next;

}

fclose(file);

}

// Функция для восстановления списка из файла

struct Node\* readFromFile(const char\* filename) {

FILE\* file = fopen(filename, "r");

if (file == NULL) {

printf("Ошибка открытия файла\n");

return NULL;

}

struct Node\* head = NULL;

int data;

while (fscanf(file, "%d", &data) != EOF) {

addToEnd(&head, data);

}

fclose(file);

return head;

}

int main() {

setlocale(LC\_ALL, "rus");

struct Node\* head = NULL;

int data, key, position;

// Создание списка

printf("Введите числа, одно за другим, 0 - конец списка:\n");

while (1) {

scanf("%d", &data);

if (data == 0) {

break;

}

addToEnd(&head, data);

}

// Печать списка

printf("\*\*\* ВЕСЬ СПИСОК \*\*\*\n");

display(head);

// Удаление элемента

printf("Введите ключ для удаления:\n");

scanf("%d", &key);

deleteNode(&head, key);

printf("\*\*\* СПИСОК ПОСЛЕ УДАЛЕНИЯ \*\*\*\n");

display(head);

// Вставка элемента на указанную позицию

printf("Введите число для добавления:\n");

scanf("%d", &data);

printf("Введите позицию для добавления:\n");

scanf("%d", &position);

insertAtPosition(&head, data, position);

printf("\*\*\* СПИСОК ПОСЛЕ ДОБАВЛЕНИЯ \*\*\*\n");

display(head);

// Запись списка в файл

writeToFile(head, "list.txt");

destroyList(&head);

printf("\*\*\* СПИСОК ПОСЛЕ УНИЧТОЖЕНИЯ \*\*\*\n");

display(head);

head = readFromFile("list.txt");

printf("\*\*\* ВОССТАНОВЛЕННЫЙ СПИСОК \*\*\*\n");

display(head);

destroyList(&head);

printf("\*\*\* СПИСОК ПОСЛЕ УНИЧТОЖЕНИЯ \*\*\*\n");

display(head);

return 0;

}

**Стек**

#include <iostream>

#include <fstream>

#include <cstring>

#include <clocale>

using namespace std;

struct Node {

char\* key;

Node\* next;

};

class Stack {

private:

Node\* top;

public:

Stack() : top(nullptr) {}

~Stack() {

while (!isEmpty()) {

pop();

}

}

bool isEmpty() {

return top == nullptr;

}

void push(const char\* key) {

Node\* newNode = new Node;

newNode->key = new char[strlen(key) + 1];

strcpy(newNode->key, key);

newNode->next = top;

top = newNode;

}

void pop() {

if (isEmpty()) {

cout << "Стек пуст!" << endl;

return;

}

Node\* temp = top;

top = top->next;

delete[] temp->key;

delete temp;

}

void removeByKey(const char\* key) {

Stack tempStack;

while (!isEmpty()) {

if (strcmp(top->key, key) == 0) {

pop();

break;

}

tempStack.push(top->key);

pop();

}

while (!tempStack.isEmpty()) {

push(tempStack.top->key);

tempStack.pop();

}

}

void insertAtPosition(const char\* key, int position) {

Stack tempStack;

int currentPos = 0;

while (!isEmpty() && currentPos < position) {

tempStack.push(top->key);

pop();

currentPos++;

}

push(key);

while (!tempStack.isEmpty()) {

push(tempStack.top->key);

tempStack.pop();

}

}

void print() {

Node\* current = top;

while (current != nullptr) {

cout << current->key << endl;

current = current->next;

}

}

void saveToFile(const char\* filename) {

ofstream file(filename);

Node\* current = top;

while (current != nullptr) {

file << current->key << endl;

current = current->next;

}

file.close();

}

void loadFromFile(const char\* filename) {

ifstream file(filename);

if (!file.is\_open()) {

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

return;

}

while (!isEmpty()) {

pop();

}

char buffer[256];

while (file.getline(buffer, 256)) {

push(buffer);

}

file.close();

}

};

int main() {

setlocale(LC\_ALL, "rus");

Stack stack;

stack.push("Apple");

stack.push("Banana");

stack.push("Cherry");

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

stack.print();

stack.removeByKey("Banana");

cout << "Стек после удаления элемента с ключом 'Banana':" << endl;

stack.print();

stack.insertAtPosition("Orange", 1);

cout << "Стек после добавления 'Orange' на позицию 1:" << endl;

stack.print();

stack.saveToFile("stack.txt");

cout << "Стек сохранен в файл 'stack.txt'" << endl;

stack.loadFromFile("stack.txt");

cout << "Стек восстановлен из файла 'stack.txt':" << endl;

stack.print();

return 0;

}

**Очередь**

#include <iostream>

#include <fstream>

#include <cstring>

#include <clocale>

using namespace std;

struct Node {

char\* key;

Node\* next;

};

class Queue {

private:

Node\* front;

Node\* rear;

public:

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

~Queue() {

while (!isEmpty()) {

dequeue();

}

}

bool isEmpty() {

return front == nullptr;

}

void enqueue(const char\* key) {

Node\* newNode = new Node;

newNode->key = new char[strlen(key) + 1];

strcpy(newNode->key, key);

newNode->next = nullptr;

if (isEmpty()) {

front = rear = newNode;

} else {

rear->next = newNode;

rear = newNode;

}

}

void dequeue() {

if (isEmpty()) {

cout << "Очередь пуста!" << endl;

return;

}

Node\* temp = front;

front = front->next;

delete[] temp->key;

delete temp;

if (front == nullptr) {

rear = nullptr;

}

}

void removeByKey(const char\* key) {

Queue tempQueue;

while (!isEmpty()) {

if (strcmp(front->key, key) == 0) {

dequeue();

break;

}

tempQueue.enqueue(front->key);

dequeue();

}

while (!tempQueue.isEmpty()) {

enqueue(tempQueue.front->key);

tempQueue.dequeue();

}

}

void insertAtPosition(const char\* key, int position) {

Queue tempQueue;

int currentPos = 0;

while (!isEmpty() && currentPos < position) {

tempQueue.enqueue(front->key);

dequeue();

currentPos++;

}

enqueue(key);

while (!tempQueue.isEmpty()) {

enqueue(tempQueue.front->key);

tempQueue.dequeue();

}

}

void print() {

Node\* current = front;

while (current != nullptr) {

cout << current->key << endl;

current = current->next;

}

}

void saveToFile(const char\* filename) {

ofstream file(filename);

Node\* current = front;

while (current != nullptr) {

file << current->key << endl;

current = current->next;

}

file.close();

}

void loadFromFile(const char\* filename) {

ifstream file(filename);

if (!file.is\_open()) {

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

return;

}

while (!isEmpty()) {

dequeue();

}

char buffer[256];

while (file.getline(buffer, 256)) {

enqueue(buffer);

}

file.close();

}

};

int main() {

setlocale(LC\_ALL, "rus");

Queue queue;

queue.enqueue("Apple");

queue.enqueue("Banana");

queue.enqueue("Cherry");

cout << "Очередь после добавления элементов:" << endl;

queue.print();

queue.removeByKey("Banana");

cout << "Очередь после удаления элемента с ключом 'Banana':" << endl;

queue.print();

queue.insertAtPosition("Orange", 1);

cout << "Очередь после добавления 'Orange' на позицию 1:" << endl;

queue.print();

queue.saveToFile("queue.txt");

cout << "Очередь сохранена в файл 'queue.txt'" << endl;

queue.loadFromFile("queue.txt");

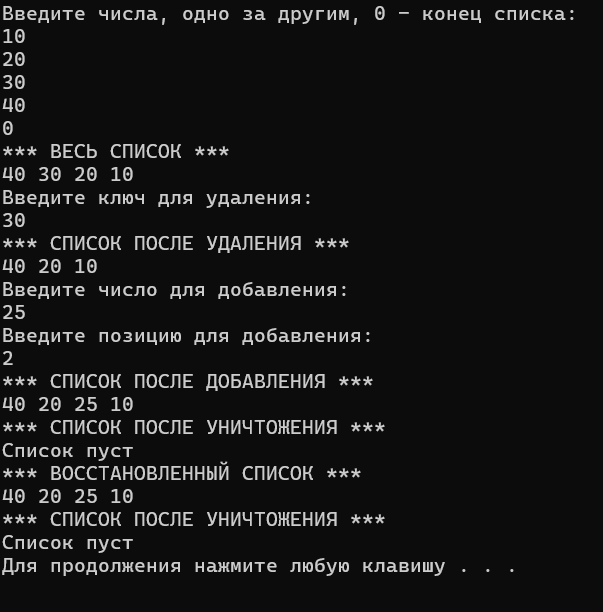
cout << "Очередь восстановлена из файла 'queue.txt':" << endl;

queue.print();

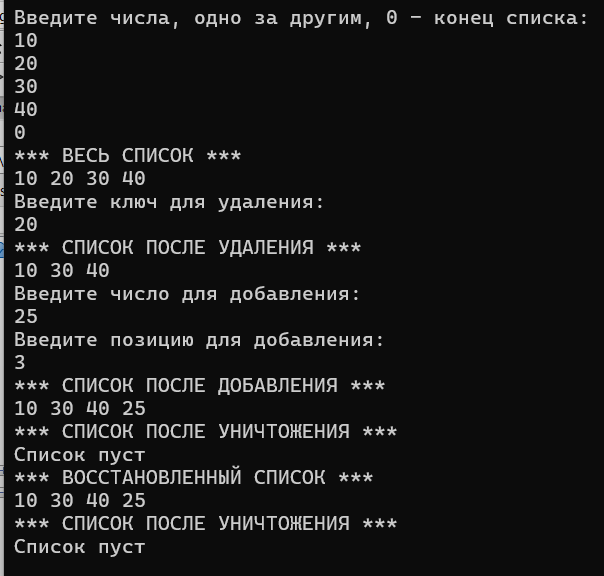
return 0;

}

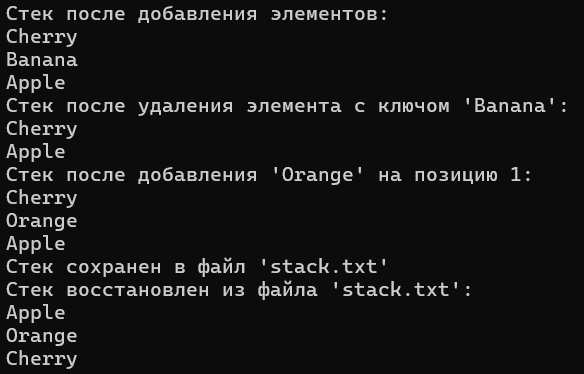
**Результаты программ:**

**Однонаправленный список**

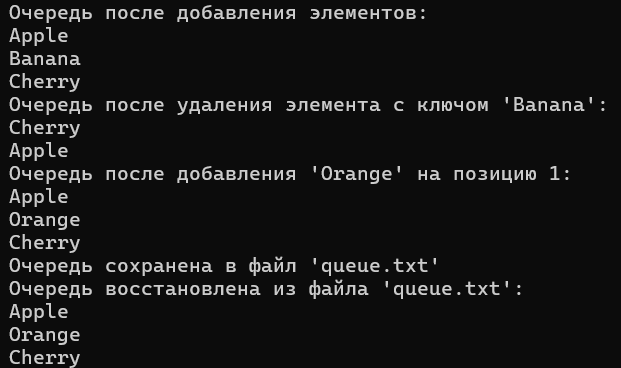
**Двунаправленный список**

****

**Стек**

****

**Очередь**

****