МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
Институт математики и информационных систем
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Дата сдачи на проверку:

«__»_____ 2024 г.

Проверено:

«__»_____ 2024 г.

РЕАЛИЗАЦИЯ И РАБОТА С ДВУСВЯЗНЫМ СПИСКОМ

Отчёт по лабораторной работе №3

по дисциплине

«Программирование»

Разработал студент гр. ИВТб-1301-05-00 _____/Черкасов А. А./

(подпись)

Заместитель кафедры ЭВМ _____/Долженкова М. Л./

(подпись)

Работа защищена «__»_____ 2024 г.

Киров

2024

Цель

Цель работы: Изучить структуру данных "двусвязный список"и принципы работы с ним. Реализовать операции добавления, удаления и вывода элементов списка. Разработать программу с интерактивным вводом данных и поддержкой управления списком через команды.

Задание

- Реализовать двусвязный список с помощью структур: Узел списка должен содержать значение (символ) и указатели на следующий и предыдущий узлы. Список должен содержать указатели на первый и последний узлы, а также размер списка.
- Реализовать функции для создания узлов и списка, добавления символов в конец списка, удаления узлов, вывода содержимого списка.
- Организовать интерактивную работу программы с командами управления: ввод символов, удаление символов (с помощью '&'), завершение ввода (с помощью '.') и выход из программы ("exit").

Теоретическая часть

Двусвязный список

Двусвязный список — это структура данных, состоящая из элементов, называемых узлами, где каждый узел хранит:

- **Значение (данные)** — информация, которую хранит узел (например, символ или число).

– **Ссылка на следующий узел (next)** — указатель на следующий элемент в списке.

– **Ссылка на предыдущий узел (prev)** — указатель на предыдущий элемент в списке.

Основное преимущество двусвязного списка — возможность проходить как в одну, так и в другую сторону, что даёт гибкость при реализации алгоритмов.

Преимущества

– **Добавление/удаление**: Быстрое добавление и удаление элементов как в начале, так и в конце списка, а также в середине. Операции вставки и удаления не требуют сдвига элементов, как это происходит в массивах.

– **Доступ**: Доступ к элементам двусвязного списка может быть как в прямом, так и в обратном направлении (вперёд/назад), что расширяет возможности для обхода и манипуляций с данными.

Недостатки

– **Память**: Каждый узел требует дополнительной памяти для хранения ссылки на предыдущий и следующий узел. Это увеличивает расход памяти по сравнению с односвязными списками или массивами.

– **Сложность управления**: Необходимо тщательно управлять ссылками на предыдущие и следующие узлы, особенно при удалении элементов, что требует дополнительных проверок и вычислений.

Сравнение

Таблица 1 – Сравнение структур.

Критерий	Двусвязный список	Односвязный список
Память	Больше	Меньше
Добавление/удаление	Быстро	Быстро
Доступ по индексу	Медленно (O(n))	Медленно (O(n))
Обход	Двунаправленный	Однонаправленный
	Массив	Стек
Память	Определена заранее	Столько же, сколько и в списке
Добавление/удаление	Медленно	Быстро
Доступ по индексу	Быстро (O(1))	Нет
Обход	Нет	Нет

Решение

Схема алгоритма решения задачи представлена на рисунках 1.1 и 1.2. Исходный код представлен в Приложении А1.

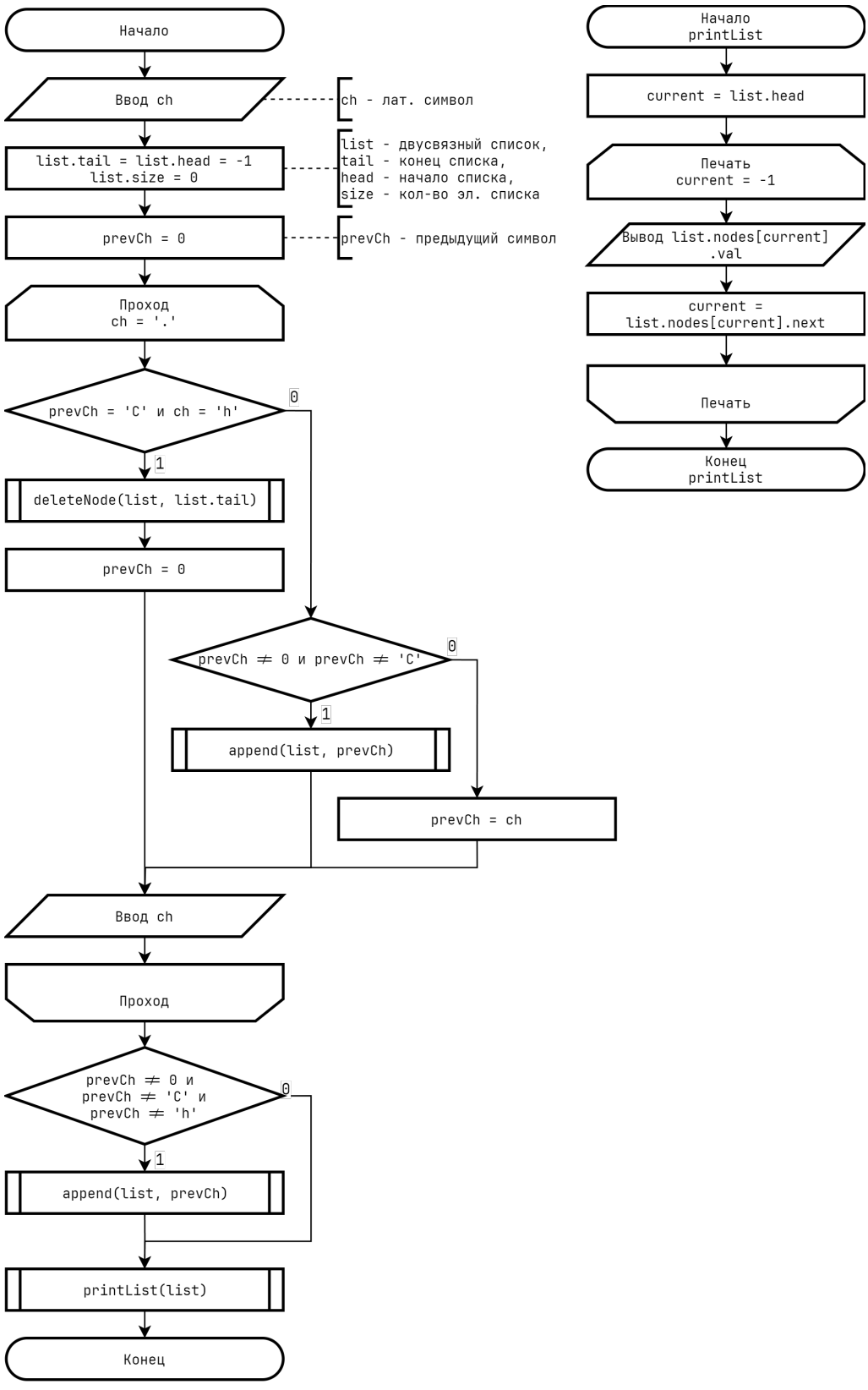

Рисунок 1.1 - Схема алгоритма.

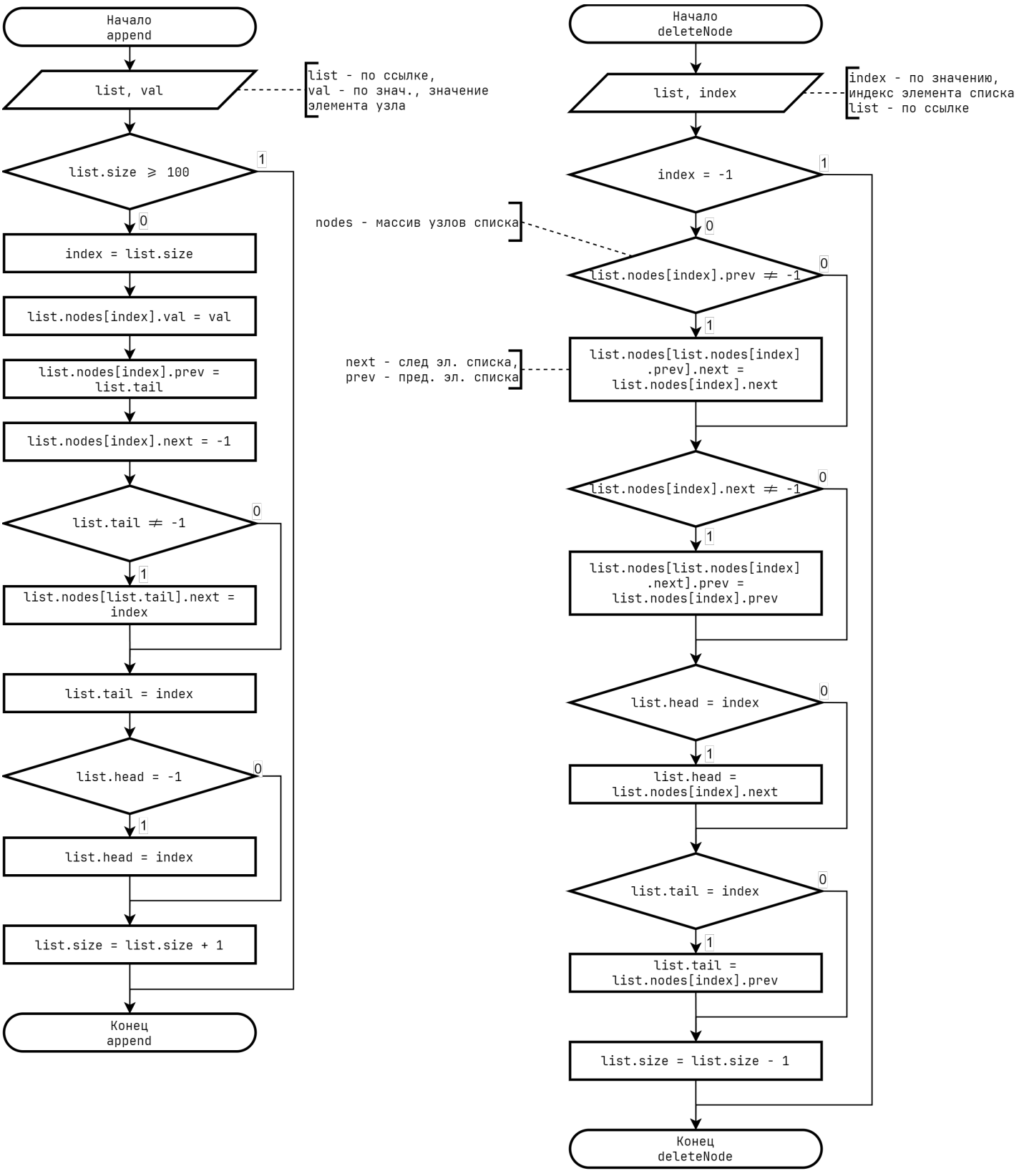

Рисунок 1.2 - Схема алгоритма.

Вывод

В ходе выполнения работы изучены основы работы с двусвязным списком, реализованы операции добавления и удаления элементов, а также организован интерактивный пользовательский интерфейс для управления списком.

Приложение А1. Исходный код

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Структура узла списка
typedef struct Node {
    char value;
    int next;
    int prev;
} Node;

// Структура списка
typedef struct LList {
    int size;
    int head;
    int tail;
    Node nodes[100];
} LList;

// Глобальная переменная для списка
LList list;

// Создает новый список
void createList() {
    list.size = 0;
    list.head = -1;
    list.tail = -1;
}

// Добавляет символ в список
void append(char value) {
    if (list.size >= 100) {
```

```
        printf("Переполнение списка.\n");
        return;
    }
    int index = list.size;
    list.nodes[index].value = value;
    list.nodes[index].prev = list.tail;
    list.nodes[index].next = -1;
    if (list.tail != -1)
        list.nodes[list.tail].next = index;
    list.tail = index;
    if (list.head == -1)
        list.head = index;
    list.size++;
}

// Удаляет узел из списка
void deleteNode(int index) {
    if (index == -1 || list.size == 0) return;
    if (list.nodes[index].prev != -1)
        list.nodes[list.nodes[index].prev].next = list.nodes[index].next;
    if (list.nodes[index].next != -1)
        list.nodes[list.nodes[index].next].prev = list.nodes[index].prev;
    if (list.head == index)
        list.head = list.nodes[index].next;
    if (list.tail == index)
        list.tail = list.nodes[index].prev;
    list.size--;
}

// Печатает список
void printList() {
    int current = list.head;
    while (current != -1) {
```

```
        printf("%c", list.nodes[current].value);
        current = list.nodes[current].next;
    }
    printf("\n");
}

int main() {
    createList();
    char ch, prevCh = '\0';

    printf("Введите последовательность лат. символов оканчивающюся '.',\
 'Ch' - для удаления предыдущего символа.\n");

    ch = getchar();
    while (ch != '.') {
        if (prevCh == 'C' && ch == 'h') {
            deleteNode(list.tail);
            prevCh = '\0';
        } else {
            if (prevCh != '\0' && prevCh != 'C') {
                append(prevCh);
            }
            prevCh = ch;
        }
        ch = getchar();
    }
    if (prevCh != '\0' && prevCh != 'C' && prevCh != 'h') {
        append(prevCh);
    }
    printList();

    return 0;
}
```