Практическая работа № 4

ЛИНЕЙНЫЕ ДВУСВЯЗНЫЕ СПИСКИ

Постановка задачи

Составить программу создания линейного двусвязного списка и реализовать основные алгоритмы работы с ЛДС, обеспечивающие следующие действия:

1 - Добавить элемент

2 - Удалить элемент

2.1 – Последний элемент

2.3 – Заданный элемент

3 – Проверить наличие элемента

6 - Вычислить длину ЛОС

7 - Вывести (распечатать) ЛОС на экран

Перечисленные действия оформить в виде самостоятельных режимов работы созданного ЛДС. Выбор режимов производить с помощью пользовательского меню.

Провести полное тестирование (всех режимов работы) программы на стеке, сформированном вводом с клавиатуры. Тест-примеры определить самостоятельно. Результаты тестирования в виде скриншотов экранов включить в отчет по выполненной работе.

Оформить отчет с подробным описанием созданного ЛДС, принципов программной реализации алгоритмов работы с ЛДС, описанием текста исходного кода и проведенного тестирования программы.

Сделать выводы о проделанной работе, основанные на полученных результатах.

1. Описание алгоритма

Алгоритм программы состоит из функции main и вызываемых в ней вспомогательных функций:

- **void add**– функция добавления элемента в ЛДС
- **void del_last** – функция удаления последнего элемента
- **bool in_lds** – функция проверки наличия элемента в ЛДС
- **void show –** функция вывода стека на экран
- **void del** – функция удаления выбранного элемента
- **int len –** функция вычисления длинны стека
- **Node* find –** функция возвращающая элемент по его значению

Двусвязный список — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и две ссылки («связки») на следующий и предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость: порядок элементов связного списка может не совпадать с порядком расположения элементов данных в памяти компьютера, а порядок обхода списка всегда явно задаётся его внутренними связями. Здесь ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. Как и односвязный список, двусвязный допускает только последовательный доступ к элементам, но при этом дает возможность перемещения в обе стороны. В этом списке проще производить удаление и перестановку элементов, так как легко доступны адреса тех элементов списка, указатели которых направлены на изменяемый элемент.

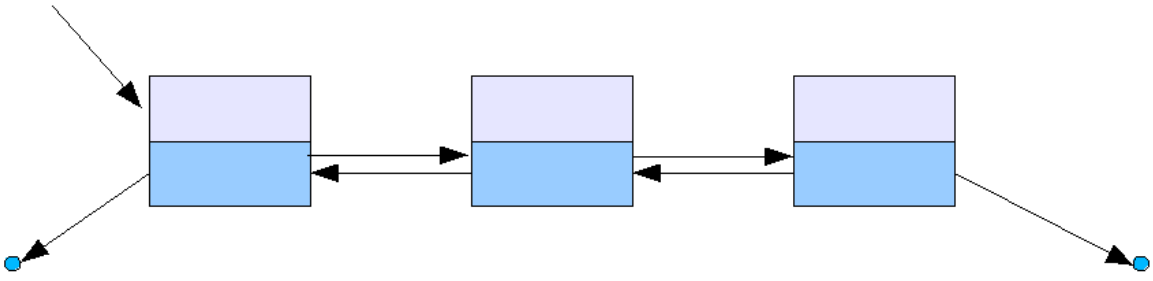

Рис.1 Принцип работы ЛДС

Функция main создает класс Lds и вызывает функцию menu.

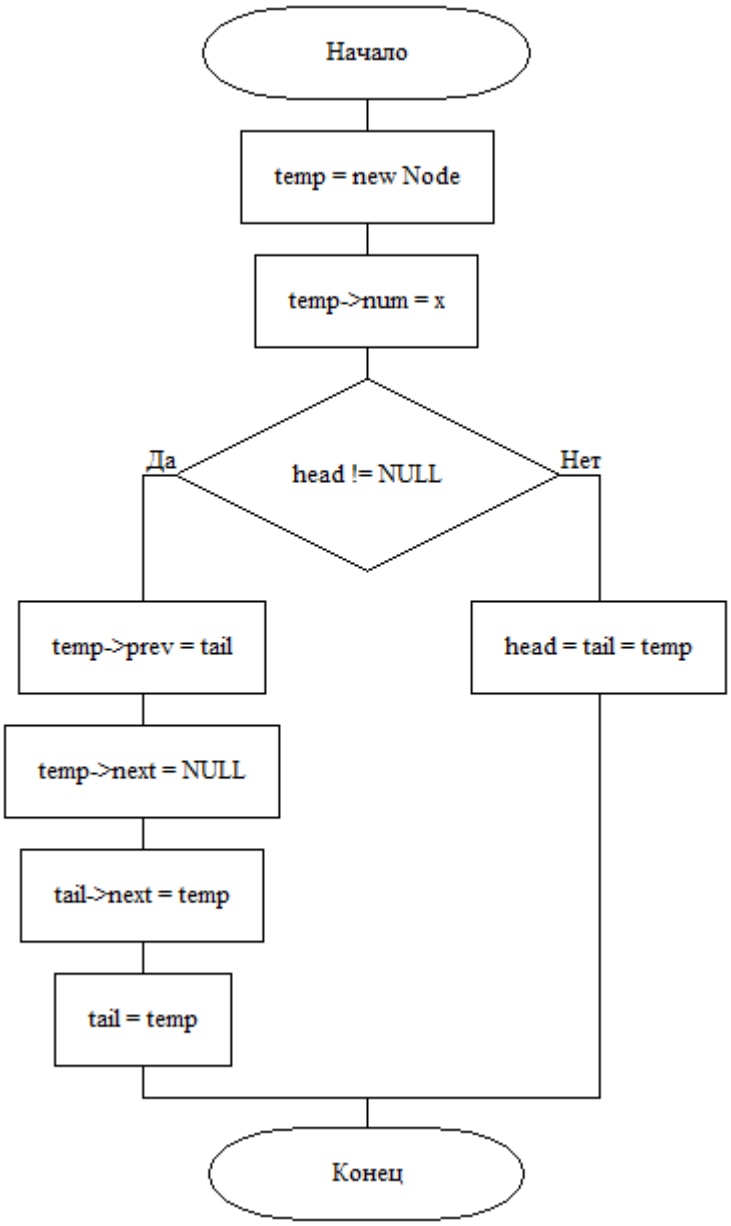

Рис.2 Схема алгоритма функции add

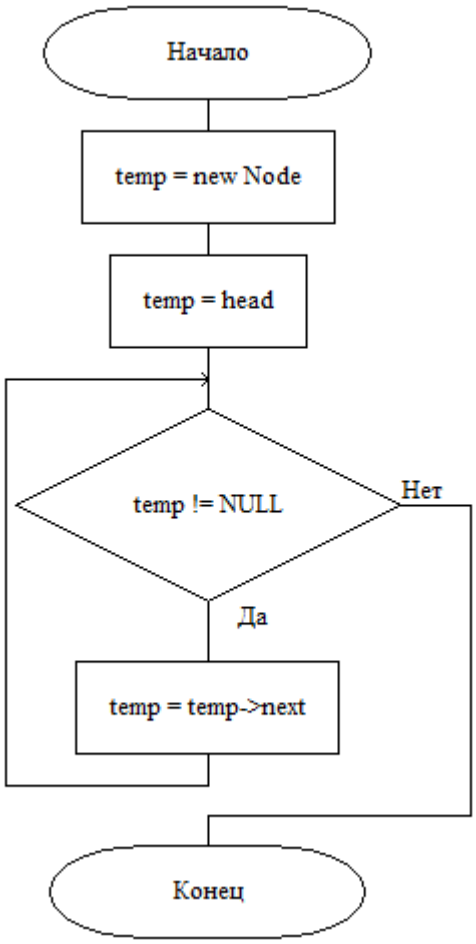

Рис.3 Схема алгоритма функции show

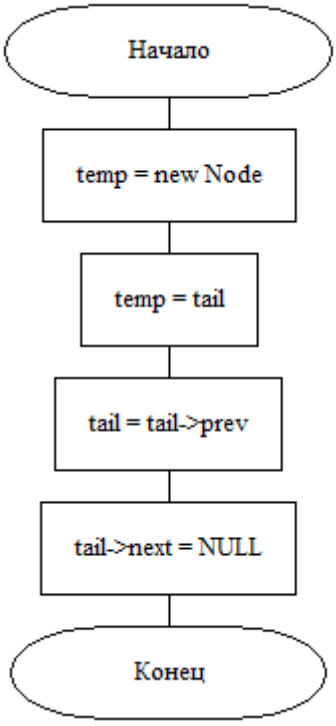

Рис.4 Схема алгоритма функции del_last

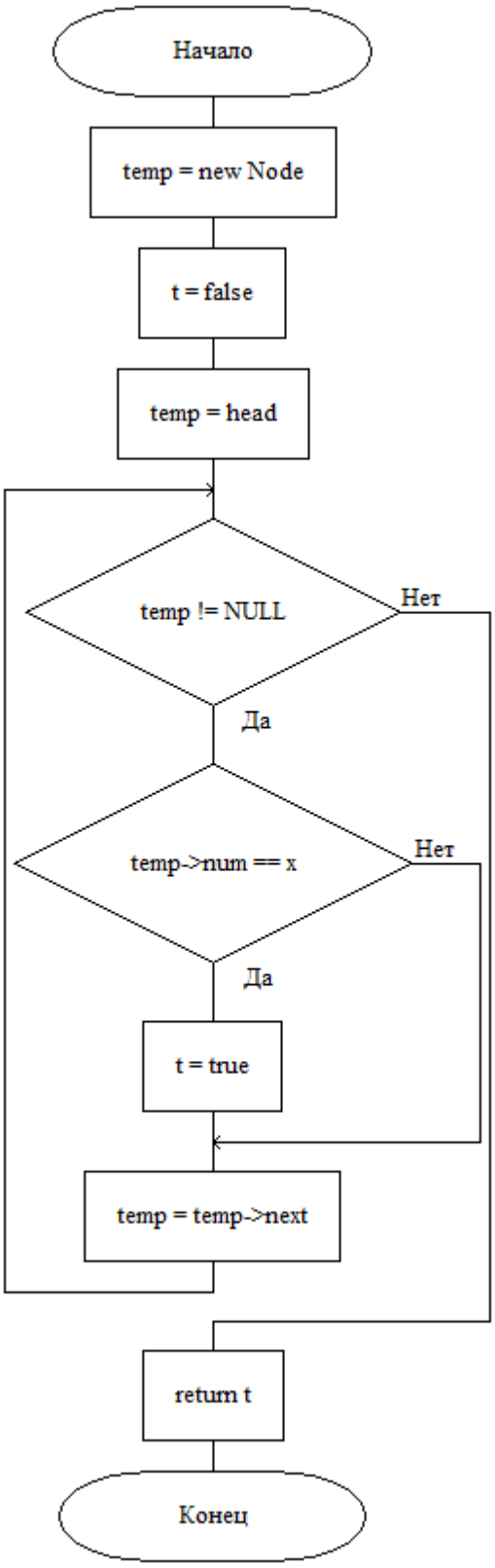

Рис.5 Схема алгоритма функции in_lds

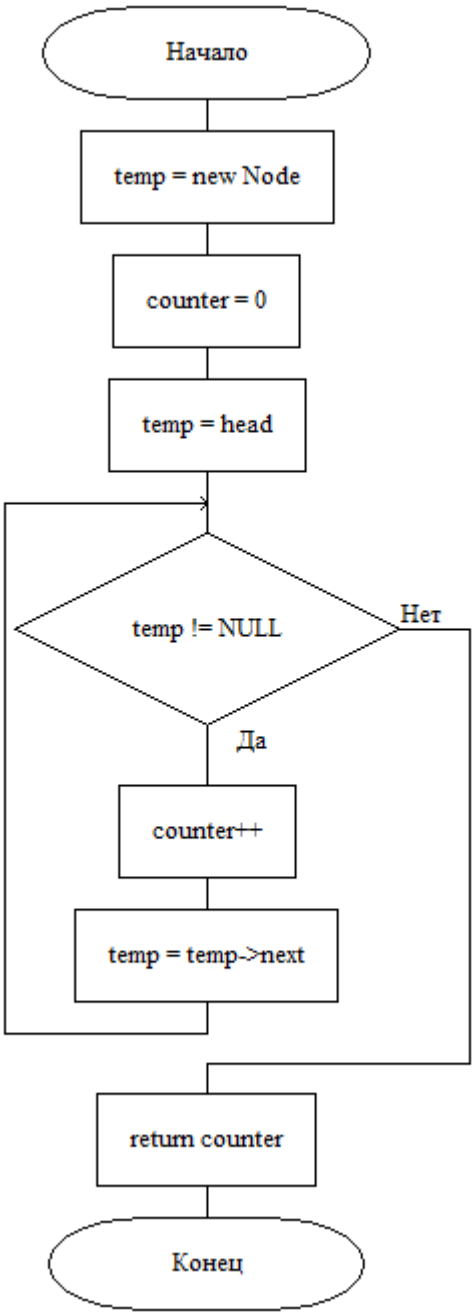

Рис.6 Схема алгоритма функции len

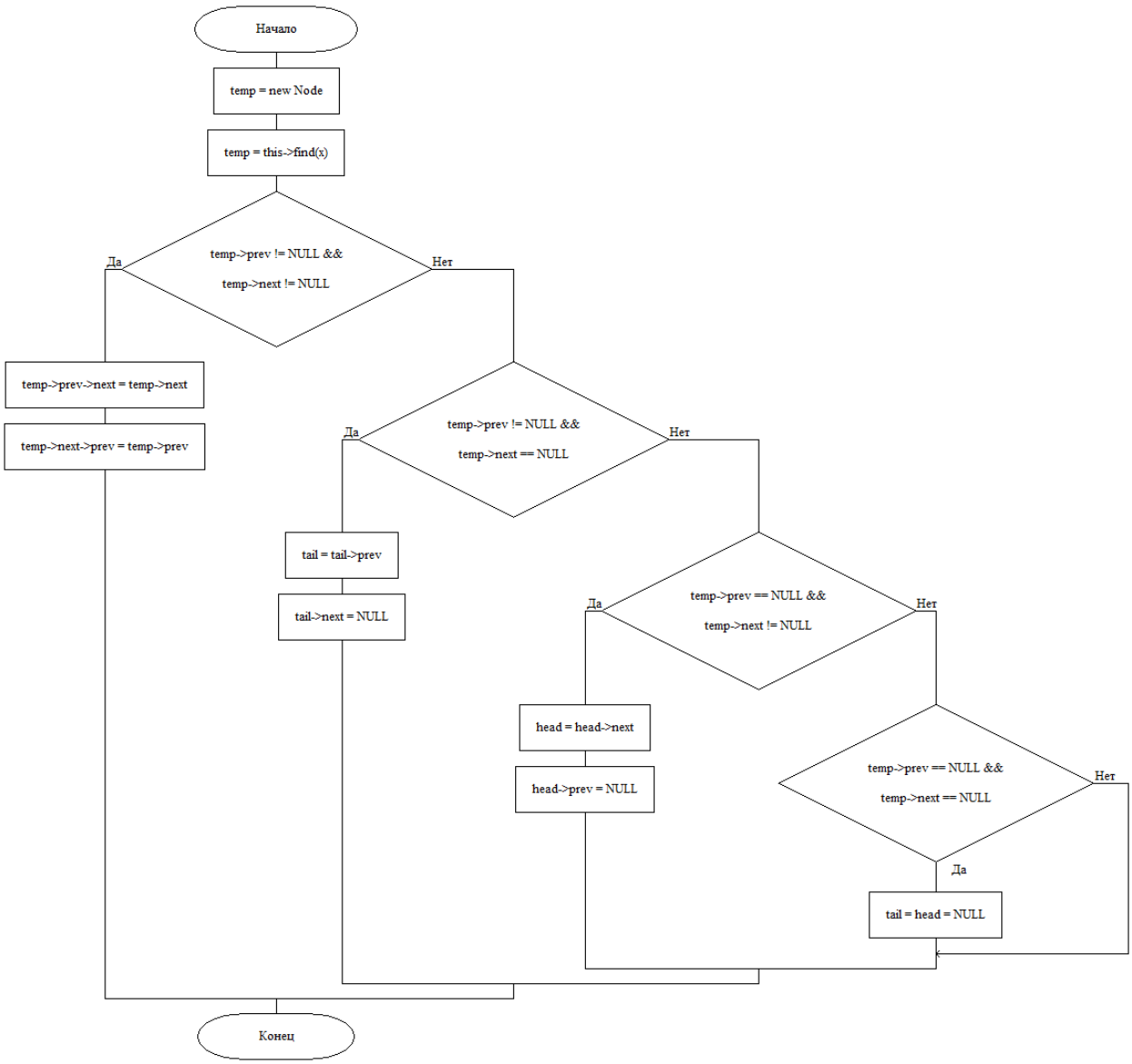

Рис.7 Схема алгоритма функции del

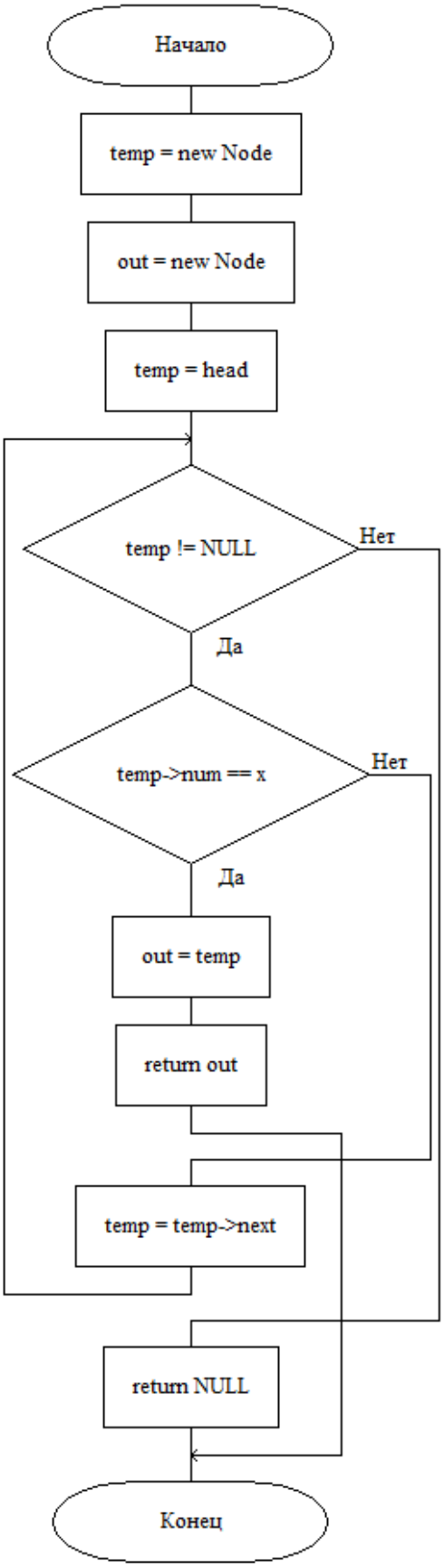

Рис.8 Схема алгоритма функции find

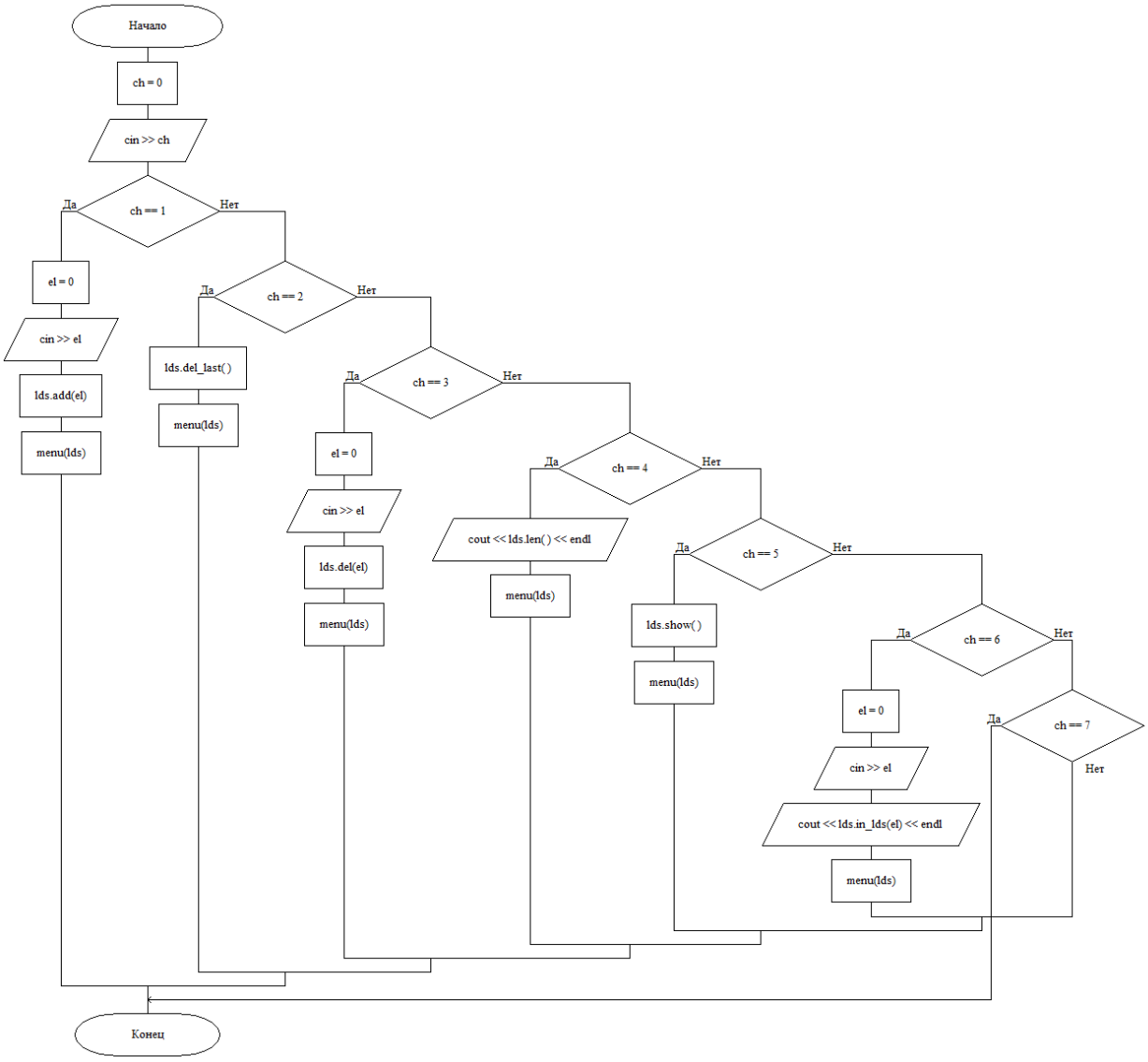

Рис.9 Схема алгоритма функции menu

Реализация алгоритма

Текст исходного кода программы

main.cpp

```
#include "Lds.h"

void menu(Lds lds) {
	cout << "Выберите команду:" << endl;
	cout << "[1] - Добавить элемент." << endl;
	cout << "[2] - Удалить последний элемент." << endl;
	cout << "[3] - Удалить элемент по значению." << endl;
	cout << "[4] - Длинна списка." << endl;
	cout << "[5] - Вывести список." << endl;
	cout << "[6] - Проверить наличие элемента в списке." << endl;
	cout << "[7] - Завершить программу." << endl;
	cout << "----> ";
	int ch = 0;
	cin >> ch;
	if (ch == 1) {
	 int el = 0;
	 cout << "Введите элемент:";
	 cin >> el;
	 lds.add(el);
	 cout << "Элемент "<<el<<" был добавлен в список." << endl;
		menu(lds);
	}
	else if (ch == 2)
	{
	 lds.del_last();
	 cout << "Последний элемент был удален" << endl;
		menu(lds);
	}
    else if (ch == 3)
    {

        int el = 0;
        cout << "Введите элемент:";
        cin >> el;
        lds.del(el);
        cout << "Элемент " << el << " был удален из списка." << endl;
        menu(lds);
    }
    else if (ch == 4)
    {
        cout << "Длинна списка = "<<lds.len()<<endl;
        menu(lds);
    }
    else if (ch == 5)
    {
        lds.show();
        menu(lds);
    }
    else if (ch == 6)
    {
        int el = 0;
        cout << "Введите элемент:";
        cin >> el;
        cout << lds.in_lds(el) << endl;
```

```
        menu(lds);
    }
    else if (ch == 7)
    {

        return;
    }

}

int main()
{
    setlocale(LC_ALL, "Russian");
    Lds * lds = new Lds;
    menu(* lds);
}
```

Lds.h

```
#include <iostream>
#pragma once
using namespace std;

struct Node {
	int num;
	Node* prev = NULL;
	Node* next = NULL;
};

class Lds
{
private:
	Node *head = NULL, *tail = NULL;
public:
	void add(int x);
	void show();
	void del_last();
	bool in_lds(int x);
	int len();
	void del(int x);
	Node* find(int x);

};
```

Lds.cpp

```
#include "Lds.h"

void Lds::add(int x) {
	Node* temp = new Node;
	temp->num = x;
	if (head != NULL) {
		temp->prev = tail;
		temp->next = NULL;
		tail->next = temp;
		tail = temp;
```

```
	}
	else head = tail = temp;
}

void Lds::show() {
	Node* temp = new Node;
	temp = head;
	while (temp != NULL) {
		cout << temp->num<<endl;
		temp = temp->next;
	}

}

void Lds::del_last() {
	Node* temp = new Node;
	temp = tail;
	tail = tail->prev;
	tail->next = NULL;
	delete temp;

}

bool Lds::in_lds(int x) {
	Node* temp = new Node;
	bool t = false;
	temp = head;
	while (temp != NULL) {
		if (temp->num == x) {
			t = true;
		}
		temp = temp->next;
	}
	return t;
}

int Lds::len() {
	Node* temp = new Node;
	int counter = 0;
	temp = head;
	while (temp != NULL) {
		counter++;
		temp = temp->next;
	}
	return counter;
}

void Lds::del(int x) {
	Node* temp = new Node;
	temp = this->find(x);
	if (temp->prev != NULL && temp->next != NULL) {
		temp->prev->next = temp->next;
		temp->next->prev = temp->prev;
	}
	else if (temp->prev != NULL && temp->next == NULL) {
		tail = tail->prev;
		tail->next = NULL;
	}
	else if (temp->prev == NULL && temp->next != NULL) {
		head = head->next;
		head->prev = NULL;
	}
	else if (temp->prev == NULL && temp->next == NULL) {
		tail = head = NULL;
```

```
        }
        delete temp;
}

Node* Lds::find(int x) {
        Node* temp = new Node;
        Node* out = new Node;
        temp = head;
        while (temp != NULL) {
                if (temp->num == x) {
                        out = temp;
                        return out;
                }
                temp = temp->next;
        }
        return NULL;
        }
```

2. Тестирование программы

Ниже представлен результат работы программы с введённым списком

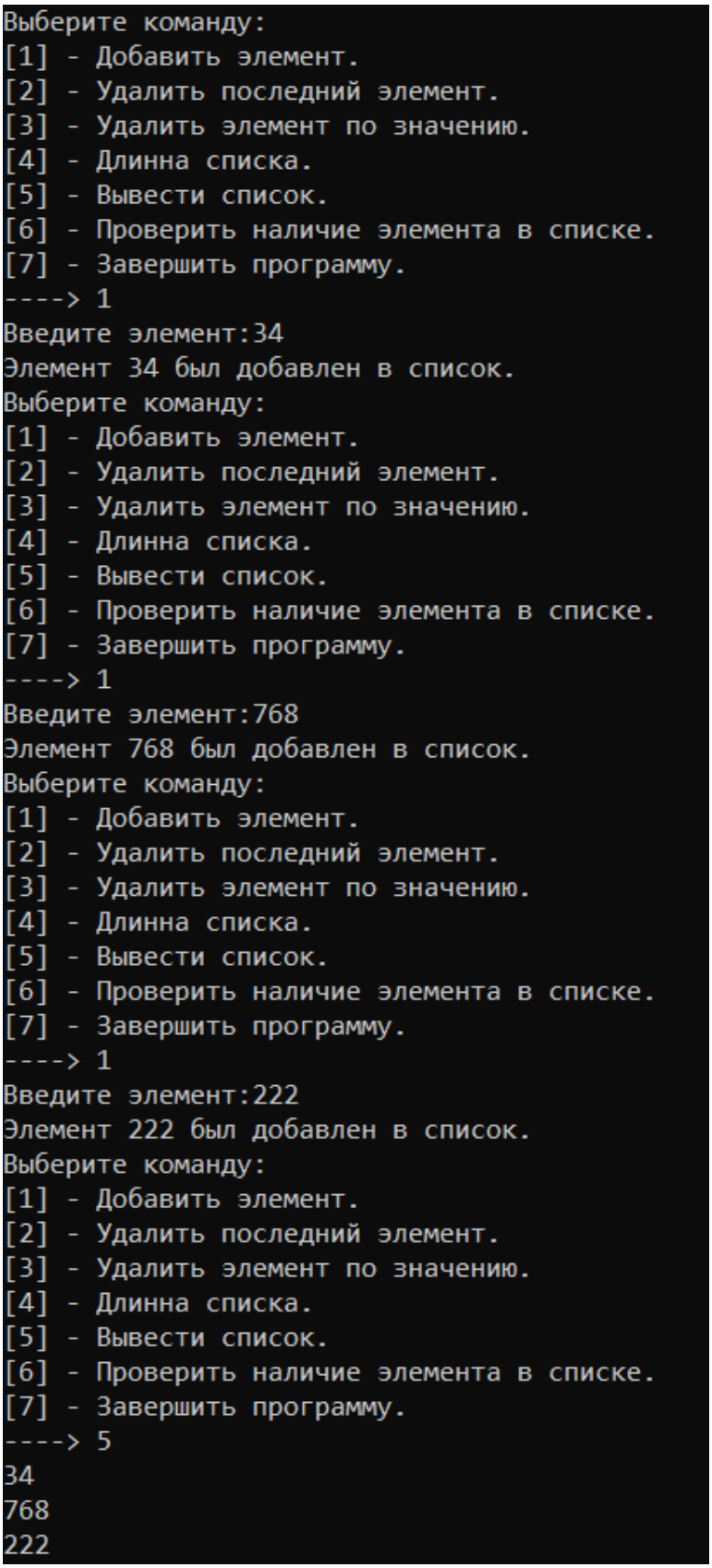

Рис.10 Скриншот добавления элементов в список и вывод списка

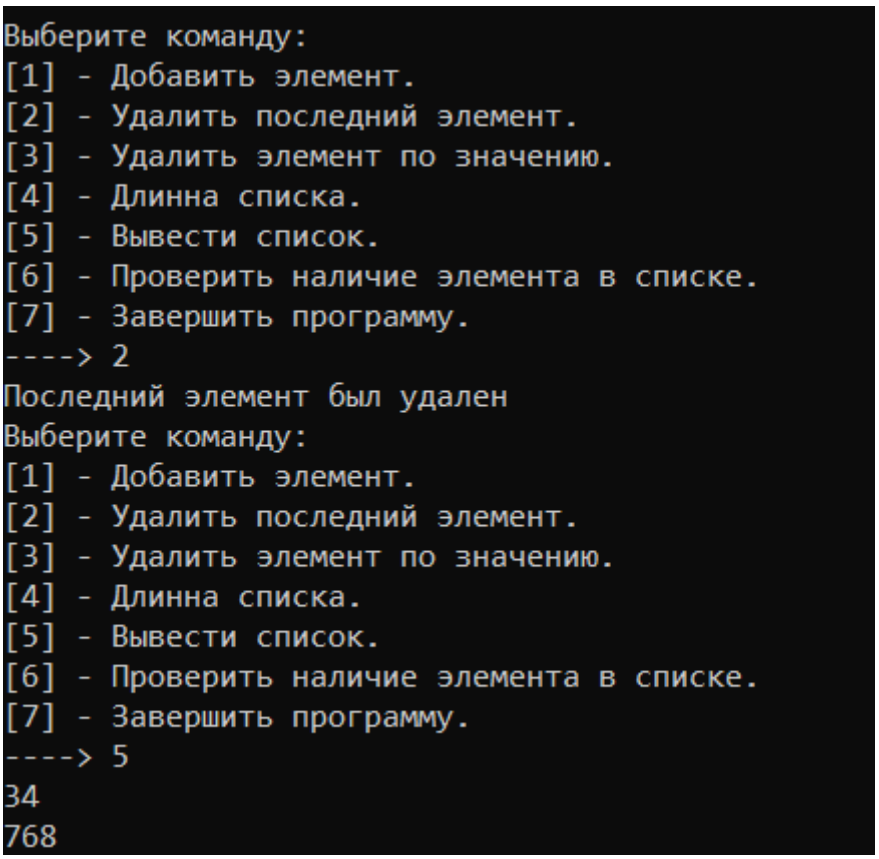

Рис.11 Скриншот удаления последнего элемента и вывод списка

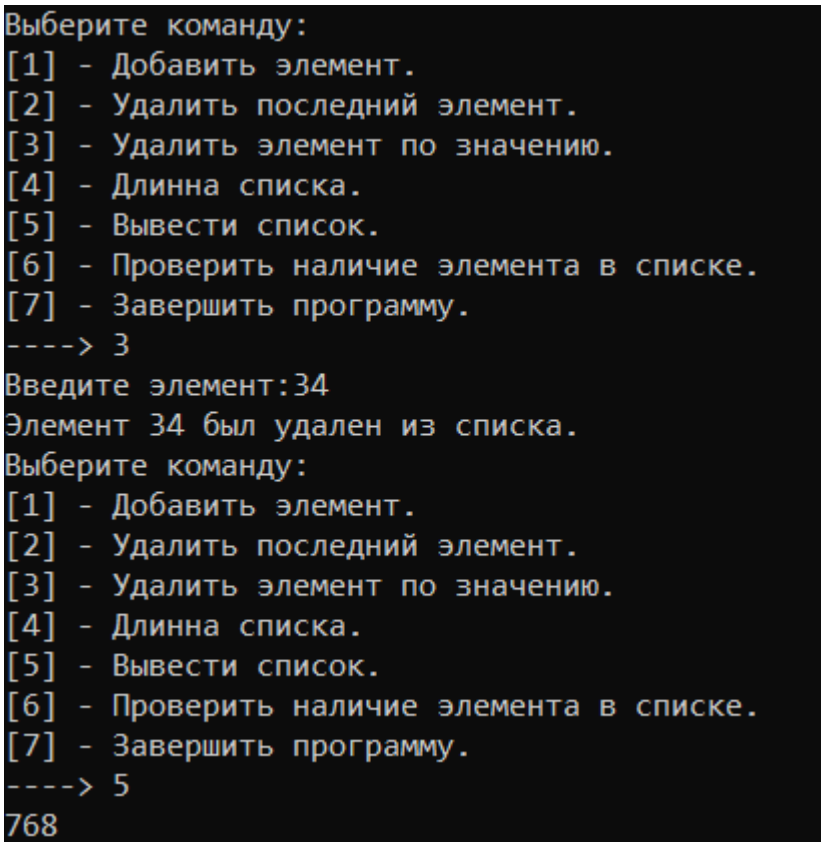

Рис.12 Скриншот удаления элемента по значению и вывод списка

```
Выберите команду:
[1] - Добавить элемент.
[2] - Удалить последний элемент.
[3] - Удалить элемент по значению.
[4] - Длинна списка.
[5] - Вывести список.
[6] - Проверить наличие элемента в списке.
[7] - Завершить программу.
----> 4
Длинна списка = 1
Выберите команду:
[1] - Добавить элемент.
[2] - Удалить последний элемент.
[3] - Удалить элемент по значению.
[4] - Длинна списка.
[5] - Вывести список.
[6] - Проверить наличие элемента в списке.
[7] - Завершить программу.
----> 5
768
```

Рис.13 Скриншот вывода длинны и вывод списка

```
Выберите команду:
[1] - Добавить элемент.
[2] - Удалить последний элемент.
[3] - Удалить элемент по значению.
[4] - Длинна списка.
[5] - Вывести список.
[6] - Проверить наличие элемента в списке.
[7] - Завершить программу.
----> 6
Введите элемент:768
1
Выберите команду:
[1] - Добавить элемент.
[2] - Удалить последний элемент.
[3] - Удалить элемент по значению.
[4] - Длинна списка.
[5] - Вывести список.
[6] - Проверить наличие элемента в списке.
[7] - Завершить программу.
----> 6
Введите элемент:123
0
```

Рис.14 Скриншот поиска элемента по значению

3. Выводы

1. В ходе работы был создан линейный двусвязный список, ссылки между элементами которого осуществляются при помощи указателей на структуры Node.
2. Также были реализованы функции работы с ЛДС: добавление элементов, удаление, вывод всех элементов списка, вывод длины, поиск элементов.
3. Были изучены положительные и негативные стороны ЛДС:
4. Преимущества: ЛДС может хранить неограниченное количество данных любого типа.
5. Недостатки: Занимает больше памяти, чем обычный массив.
6. Таким образом, была изучена работа линейного двусвязного списка и функций работы над ними и их реализация.

Список используемых информационных источников

1. Сыромятников В.П. Структуры и алгоритмы обработки данных, лекции, РТУ МИРЭА, Москва, 2020/2021 уч./год.
2. Документация по языку программирования C++, интернет-ресурс: https://en.cppreference.com/w/ (Дата обращения – 02.11.2020)
3. Интегрированная среда разработки для языков программирования C и C++, разработанная компанией JetBrains - CLion / Copyright © 2000-2020 JetBrains s.r.o., интернет-ресурс: https://www.jetbrains.com/clion/learning-center/ (Дата обращения – 02.11.2020).
4. ГОСТ 19.701-90 ЕСПД. Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения. Интернет-ресурс: http://docs.cntd.ru/document/gost-19-701-90-espd (Дата обращения – 02.11.2020).
5. Описание связных списков. интернет-ресурс: https://ru.wikipedia.org/wikiСвязный_список (Дата обращения 02.11.2020).