Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №6**

**«Реализация элементарных структур данных на основе статической памяти»**

**ПО «МДК 05.02 Разработка кода информационных систем»**

Выполнил: студент учебной группы

ИСПк-202-52-00

Лаптев Владимир Антонович

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2025

**Цель работы**

Изучение принципов работы с базовыми структурами данных, получение навыков организации саѕе-меню.

**Формулировка задания (с вариантом)**

Вариант: 10

Задание:

1. Написать программу для работы со структурой данных "Двусвязный список".

2. Структура данных должна быть реализована на основе статической памяти. 3. Работа со структурой должна осуществляться с помощью саѕе-меню. Предусмотреть наглядную визуализацию содержимого структуры.

**Описание алгоритма**

В данной программе реализована функция работы со структурой данных Двусвязный список. Двусвязный список — список, в котором элемент содержит ссылки на предыдущий и следующий элемент. Была создана процедура, которая инициализирует список. Также были написаны процедуры, которые добавляют элементы в начало и в конец. Были написаны функции для вывода с конца и начала. Также была написана функция для удаления элементов. В основном теле программы написано case меню, которое позволяет выбрать действие. Процедуры были реализованы с статическим типом данных. Заключается это в том, что был создан массив, который хранит числовое значение и индексы следующего и предыдущего элементов.

**Схема алгоритма с комментариями**

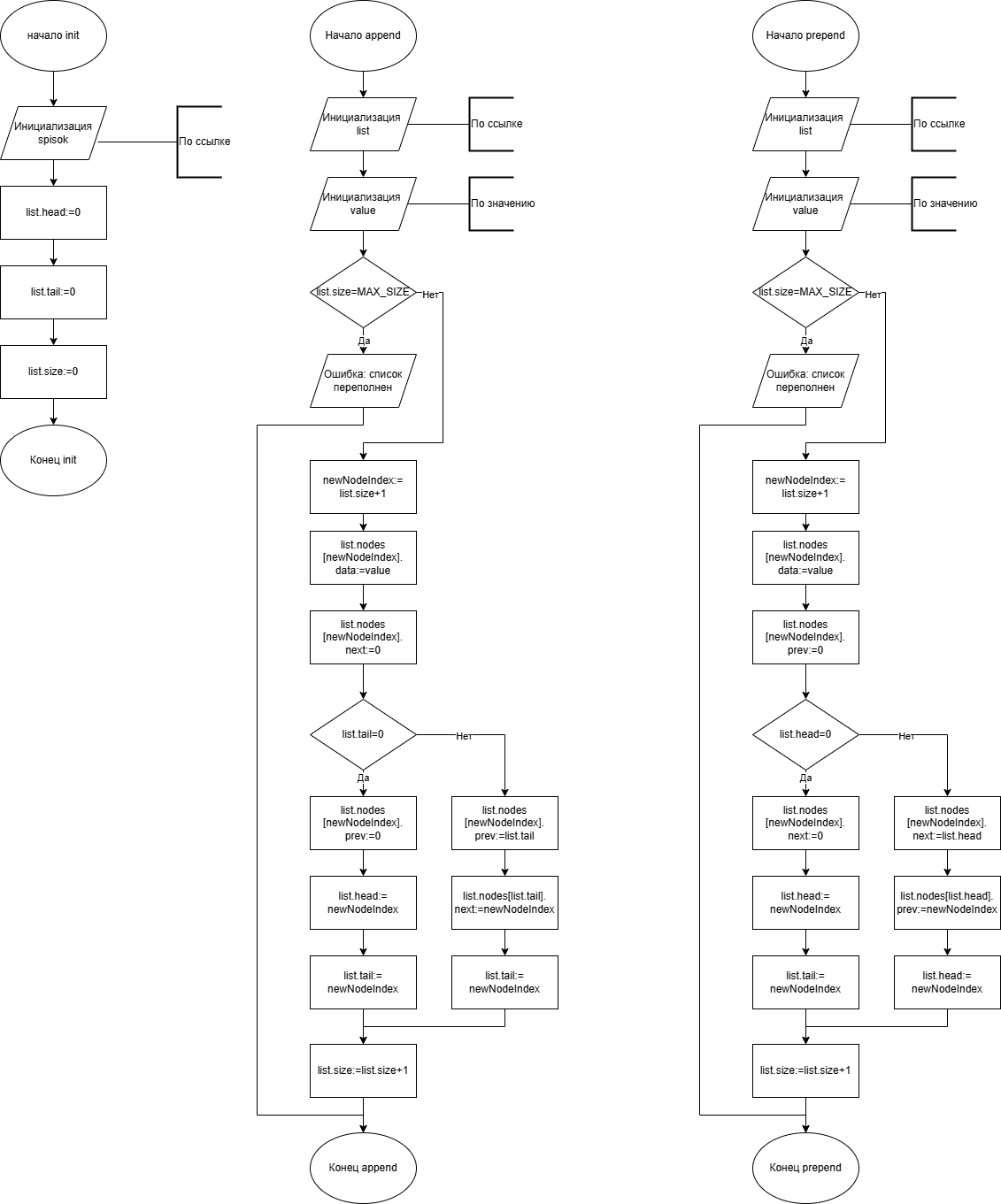


Рисунок 1 — Алгоритм решения программы

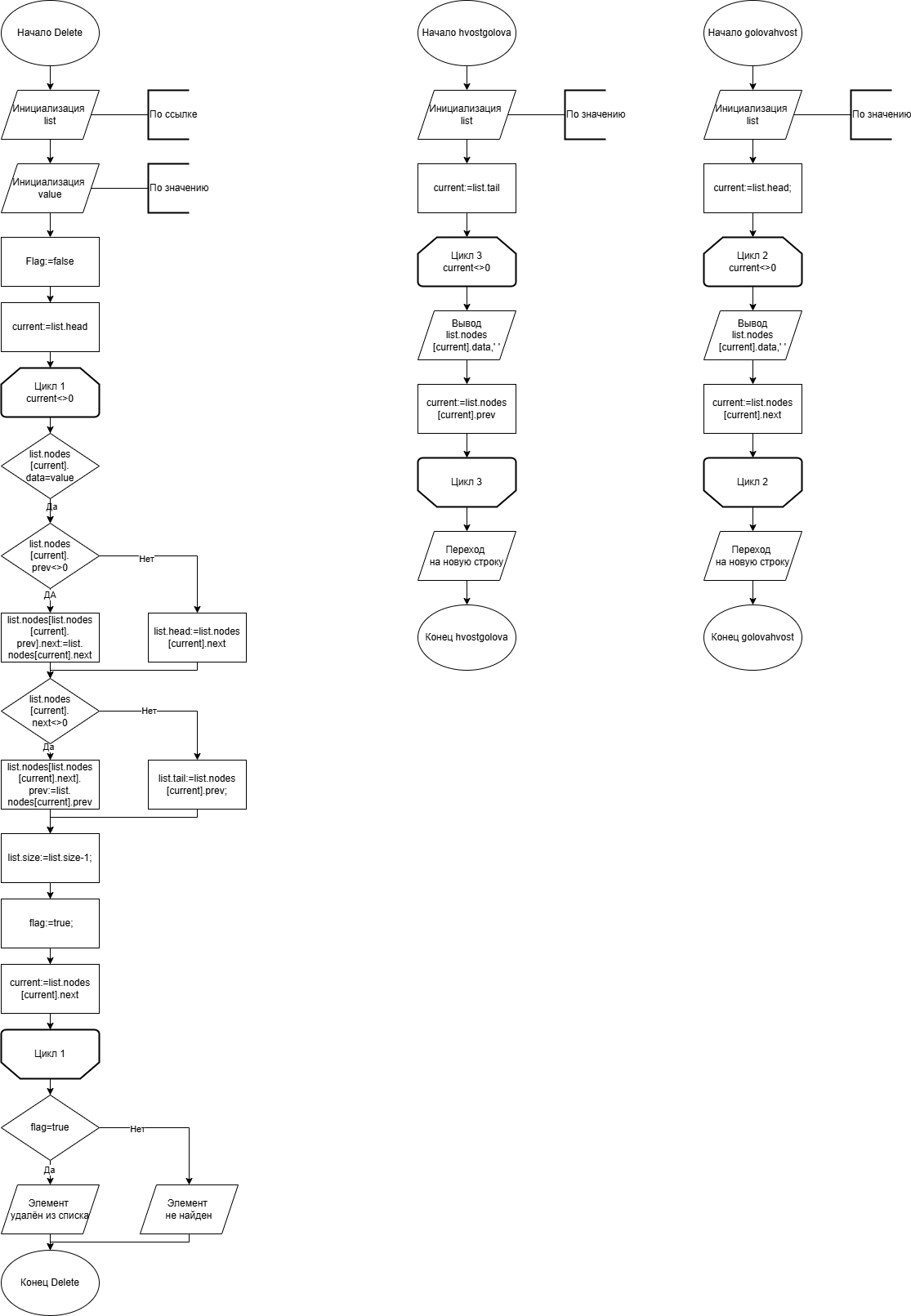


Рисунок 2 — Алгоритм решения программы

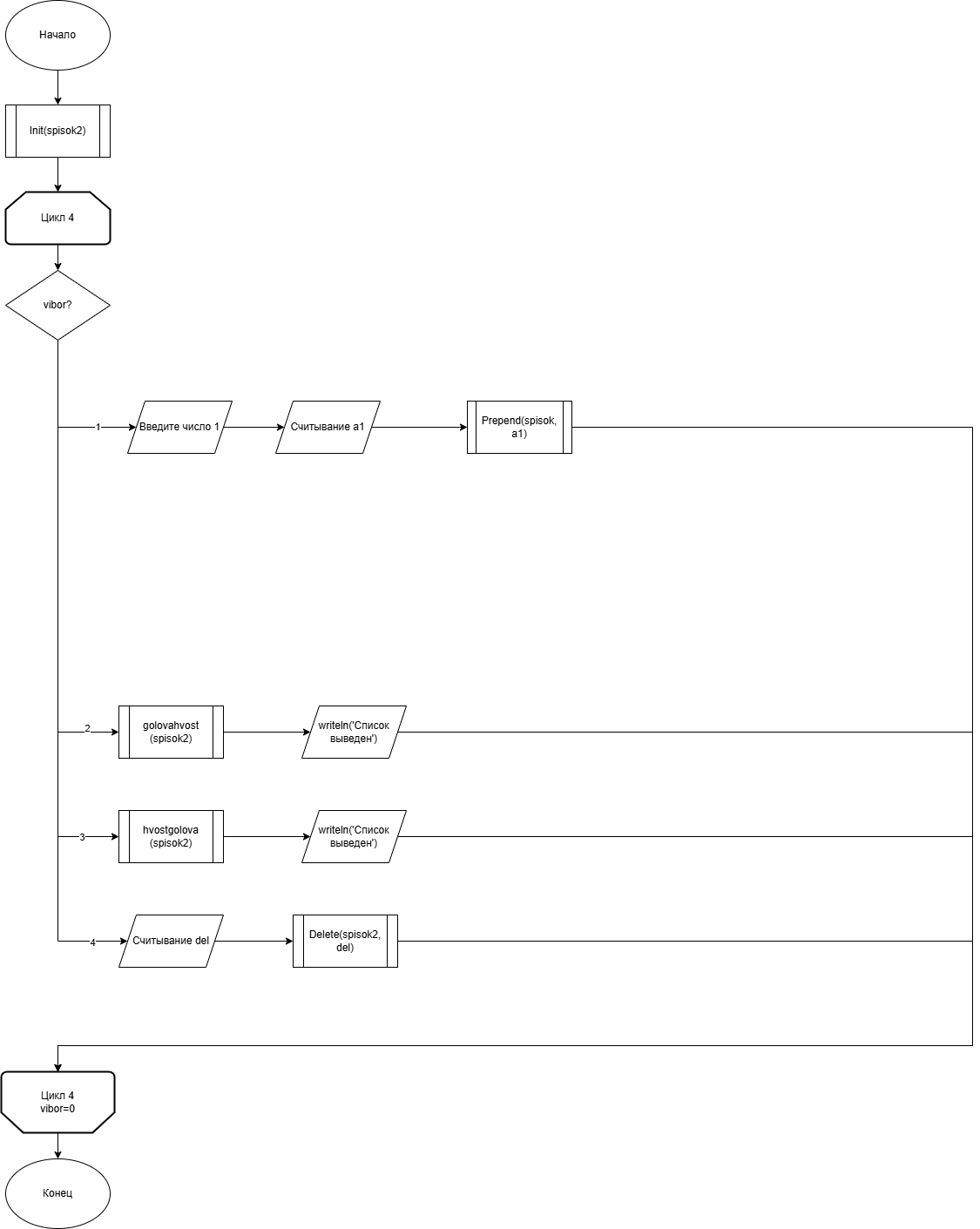


Рисунок 3 — Алгоритм решения программы

**Код программы**

**program** qq;

**uses** Crt;

**const**

MAX\_SIZE=100;

**type**

Node=**record**

data:integer;

prev,next:integer;

**end**;

spisok=**record**

nodes:**array**[1..MAX\_SIZE] **of** Node;

head,tail:integer;

size:integer;

**end**;

**procedure** init(**var** list:spisok);

**begin**

list.head:=0;

list.tail:=0;

list.size:=0;

**end**;

**procedure** append(**var** list:spisok;value:integer);

**var**

newNodeIndex:integer;

**begin**

**if** list.size=MAX\_SIZE **then**

**begin**

writeln('Ошибка: список переполнен');

**exit**;

**end**;

newNodeIndex:=list.size+1;

list.nodes[newNodeIndex].data:=value;

list.nodes[newNodeIndex].next:=0;

**if** list.tail=0 **then**

**begin**

list.nodes[newNodeIndex].prev:=0;

list.head:=newNodeIndex;

list.tail:=newNodeIndex;

**end**

**else**

**begin**

list.nodes[newNodeIndex].prev:=list.tail;

list.nodes[list.tail].next:=newNodeIndex;

list.tail:=newNodeIndex;

**end**;

list.size:=list.size+1;

**end**;

**procedure** prepend(**var** list:spisok;value:integer);

**var**

newNodeIndex:integer;

**begin**

**if** list.size=MAX\_SIZE **then**

**begin**

writeln('Ошибка: список переполнен');

**exit**;

**end**;

newNodeIndex :=list.size+1;

list.nodes[newNodeIndex].data:=value;

list.nodes[newNodeIndex].prev:=0;

**if** list.head=0 **then**

**begin**

list.nodes[newNodeIndex].next:=0;

list.head:=newNodeIndex;

list.tail:=newNodeIndex;

**end**

**else**

**begin**

list.nodes[newNodeIndex].next:=list.head;

list.nodes[list.head].prev:=newNodeIndex;

list.head:=newNodeIndex;

**end**;

list.size:=list.size+1;

**end**;

**procedure** Delete(**var** list:spisok;value:integer);

**var**

current:integer;

flag:boolean;

**begin**

flag:=false;

current:=list.head;

**while** current<>0 **do**

**begin**

**if** list.nodes[current].data=value **then**

**begin**

**if** list.nodes[current].prev<>0 **then**

list.nodes[list.nodes[current].prev].next:=list.nodes[current].next

**else**

list.head:=list.nodes[current].next;

**if** list.nodes[current].next<>0 **then**

list.nodes[list.nodes[current].next].prev:=list.nodes[current].prev

**else**

list.tail:=list.nodes[current].prev;

list.size:=list.size-1;

flag:=true;

**end**;

current:=list.nodes[current].next;

**end**;

**if** flag=true **then**

writeln('Элемент удалён из списка')

**else**

writeln('Элемент не найден');

**end**;

**procedure** golovahvost(list:spisok);

**var**

current:integer;

**begin**

current:=list.head;

**while** current<>0 **do**

**begin**

write(list.nodes[current].data,' ');

current:=list.nodes[current].next;

**end**;

writeln;

**end**;

**procedure** hvostgolova(list:spisok);

**var**

current:integer;

**begin**

current:=list.tail;

**while** current<>0 **do**

**begin**

write(list.nodes[current].data,' ');

current:=list.nodes[current].prev;

**end**;

writeln;

**end**;

**var**

spisok2:spisok;

a1,vibor,del:integer;

**begin**

init(spisok2);

**repeat**

writeln('Выберите действие');

writeln('1. Ввод данных');

writeln('2. Вывод списка в нормальном порядке');

writeln('3. Вывод списка в обратном порядке');

writeln('4. Удаление элемента списка');

writeln('0. Выход из программы');

readln(vibor);

**case** vibor **of**

1: **begin**

readln(a1);

append(spisok2,a1);

**end**;

2: **begin**

golovahvost(spisok2);

writeln('Список выведен');

**end**;

3: **begin**

hvostgolova(spisok2);

writeln('Список выведен');

**end**;

4: **begin**

writeln('Введите число, которое нужно удалить');

readln(del);

Delete(spisok2,del);

**end**;

**end**;

**until** vibor = 0;

**end**.

**Результат выполнения программы**

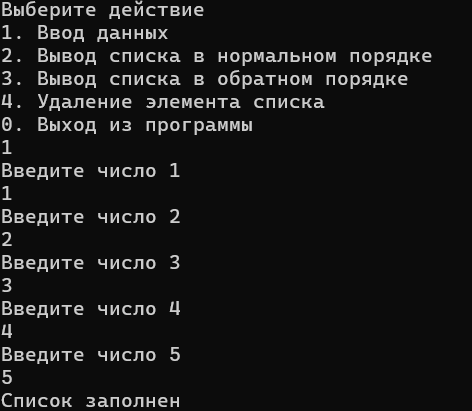


Рисунок 4 — Результат выполнения программы

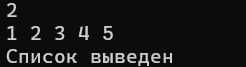


Рисунок 5 — Результат выполнения программы



Рисунок 6 — Результат выполнения программы

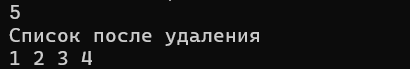


Рисунок 7 — Результат выполнения программы

**Вывод**

В данной контрольной работе мы изучили реализацию элементарных структур данных на основе статической памяти, с фокусом на двусвязный список.

В процессе работы мы освоили основные операции с двусвязным списком, такие как создание, добавление и удаление узлов, а также поиск элементов. Определили структуру узла, который содержит данные и ссылки на предыдущий и следующий узел, что обеспечивает двусторонний доступ к элементам списка.

Также отметили преимущества и недостатки использования двусвязного списка по сравнению с другими структурами данных, такими как массивы и односвязные списки. Например, двусвязный список позволяет эффективнее вносить изменения в середине структуры, но требует больше памяти на хранение ссылок.

В итоге, работа над данной темой помогла лучше понять принципы работы с динамическими структурами данных и привела к осознанию важности выбора подходящей структуры данных в зависимости от требований задачи.