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 СТАТИСТИЧЕСКИЙ ПАМЯТИ»**

**«МДК 05.02 РАЗРАБОТКА КОДА ИНФОРМАЦИОННЫХ СИСТЕМ»**

Выполнил: студент учебной группы

ИСПк-202-52-00

Подпорин А.А.

Преподаватель:

Сергеева Е.Г.

Киров

2025

**Цель работы:** изучение принципов работы с базовыми структурами данных, получение навыков организации case-меню.

**Вариант 14.**

**Задание:**

1. Написать программу для работы со структурой данных «Односвязный список»
2. Структура данных должна быть реализована на основе статической памяти
3. Работа со структурой должна осуществляться с помощью case-меню. Предусмотреть наглядную визуализацию содержимого структуры.

**Формулировка задания:**

Написать программу для работы со структурой данных «Односвязный список», реализованная на основе статической памяти, и через case-меню.

**Алгоритм схемы**

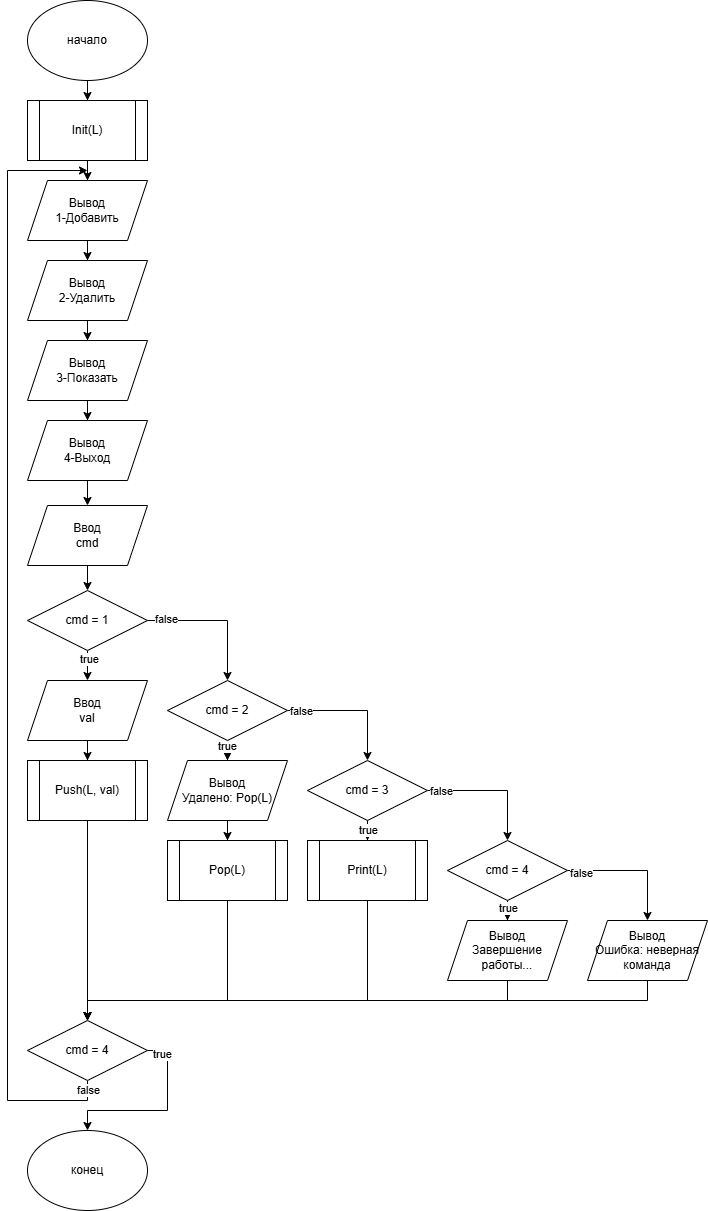
****

Рисунок 1 – Основная схема алгоритма

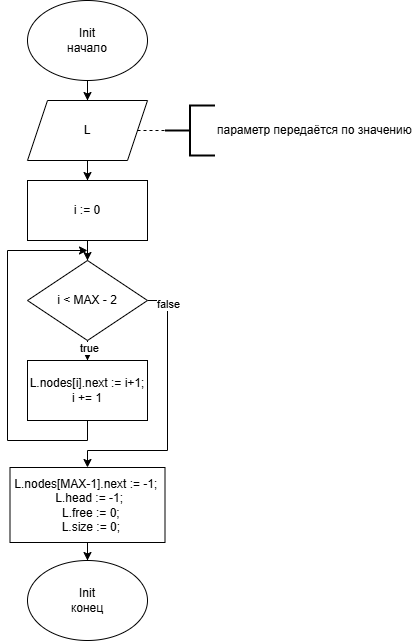
****

Рисунок 2 – Схема алгоритма процедуры init

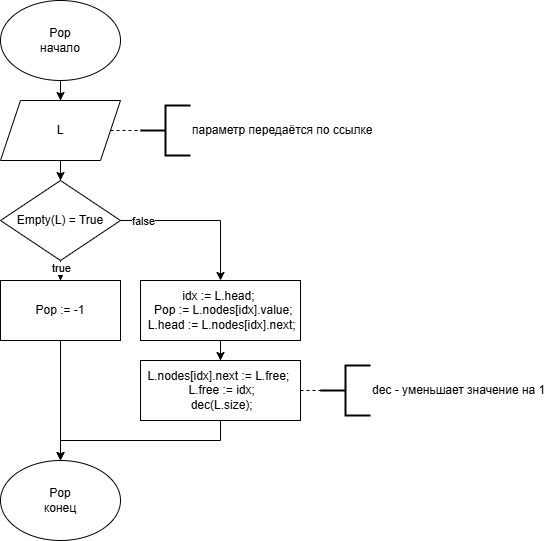
****

Рисунок 3 – Схема алгоритма процедуры pop

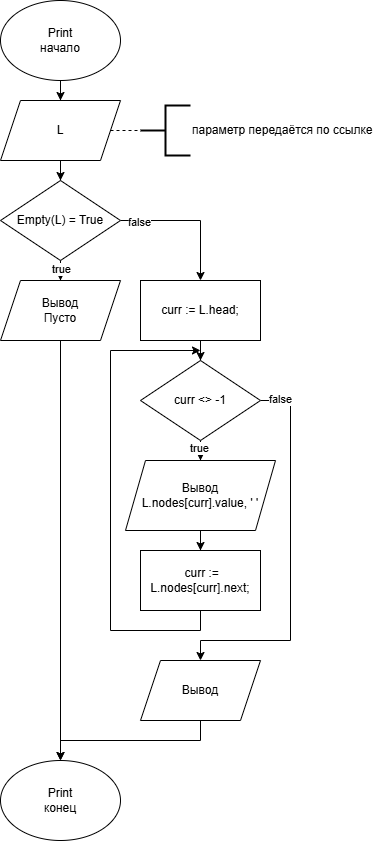


Рисунок 4 – Схема алгоритма процедуры print

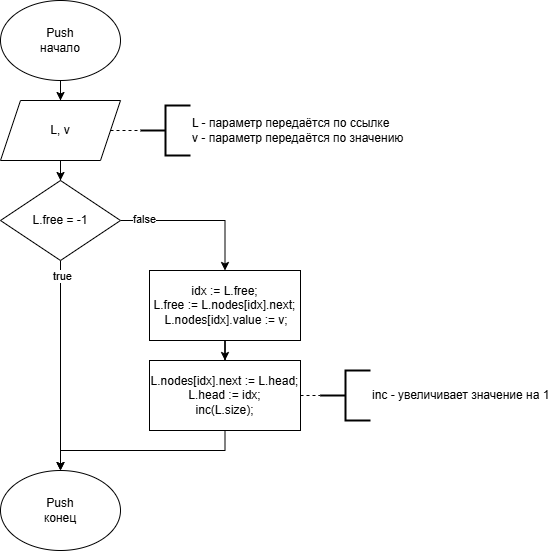


Рисунок 5 – Схема алгоритма процедуры Push

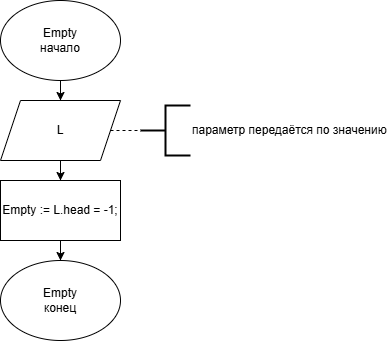


Рисунок 6 – Схема алгоритма процедуры Empty

**Код программы:**

**program** SimpleLinkedList;

**const**

MAX = 100; // Максимальный размер списка

**type**

TNode = **record** // Структура элемента списка

value: integer; // Значение элемента

next: integer; // Индекс следующего элемента (-1 = конец)

**end**;

TList = **record** // Структура списка

nodes: **array**[0..MAX-1] **of** TNode; // Массив элементов

head, free, size: integer; // Голова, свободный элемент, размер

**end**;

// Инициализация списка

**procedure** Init(**var** L: TList);

**var** i: integer;

**begin**

**for** i := 0 **to** MAX-2 **do** L.nodes[i].next := i+1; // Связываем элементы

L.nodes[MAX-1].next := -1; // Последний элемент без следующего

L.head := -1; // Пустой список

L.free := 0; // Первый свободный

L.size := 0; // Размер 0

**end**;

// Проверка пустоты

**function** Empty(**var** L: TList): boolean;

**begin**

Empty := L.head = -1; // Если голова -1, список пуст

**end**;

// Добавление в начало

**procedure** Push(**var** L: TList; v: integer);

**var** idx: integer;

**begin**

**if** L.free = -1 **then exit**; // Если нет свободных мест - выход

idx := L.free; // Берем первый свободный индекс

L.free := L.nodes[idx].next; // Обновляем свободный список

L.nodes[idx].value := v; // Записываем значение

L.nodes[idx].next := L.head; // Новый элемент указывает на старую голову

L.head := idx; // Обновляем голову

inc(L.size); // Увеличиваем размер

**end**;

// Удаление из начала

**function** Pop(**var** L: TList): integer;

**var** idx: integer;

**begin**

**if** Empty(L) **then**

**begin**

Pop := -1; // Возвращаем -1 при пустом списке

**exit**;

**end**;

idx := L.head; // Запоминаем индекс головы

Pop := L.nodes[idx].value; // Возвращаемое значение

L.head := L.nodes[idx].next; // Новая голова - следующий элемент

L.nodes[idx].next := L.free; // Освобождаемый элемент в свободный список

L.free := idx; // Обновляем указатель на свободный список

dec(L.size); // Уменьшаем размер

**end**;

// Печать списка

**procedure** Print(**var** L: TList);

**var** curr: integer;

**begin**

**if** Empty(L) **then**

**begin**

writeln('Пусто');

**exit**;

**end**;

curr := L.head; // Начинаем с головы

**while** curr <> -1 **do** // Пока не конец списка

**begin**

write(L.nodes[curr].value, ' '); // Печатаем значение

curr := L.nodes[curr].next; // Переходим к следующему

**end**;

writeln;

**end**;

// Основная программа

**var**

L: TList;

cmd, val: integer;

**begin**

Init(L); // Инициализация списка

**repeat**

writeln('1-Добавить');

writeln('2-Удалить');

writeln('3-Показать');

writeln('4-Выход');

readln(cmd); // Чтение команды

**case** cmd **of**

1: **begin**

readln(val); // Чтение значения

Push(L, val); // Добавление

**end**;

2: writeln('Удалено: ', Pop(L)); // Удаление элемента

3: Print(L); // Вывод списка

4: writeln('Завершение работы...'); // Выход из программы

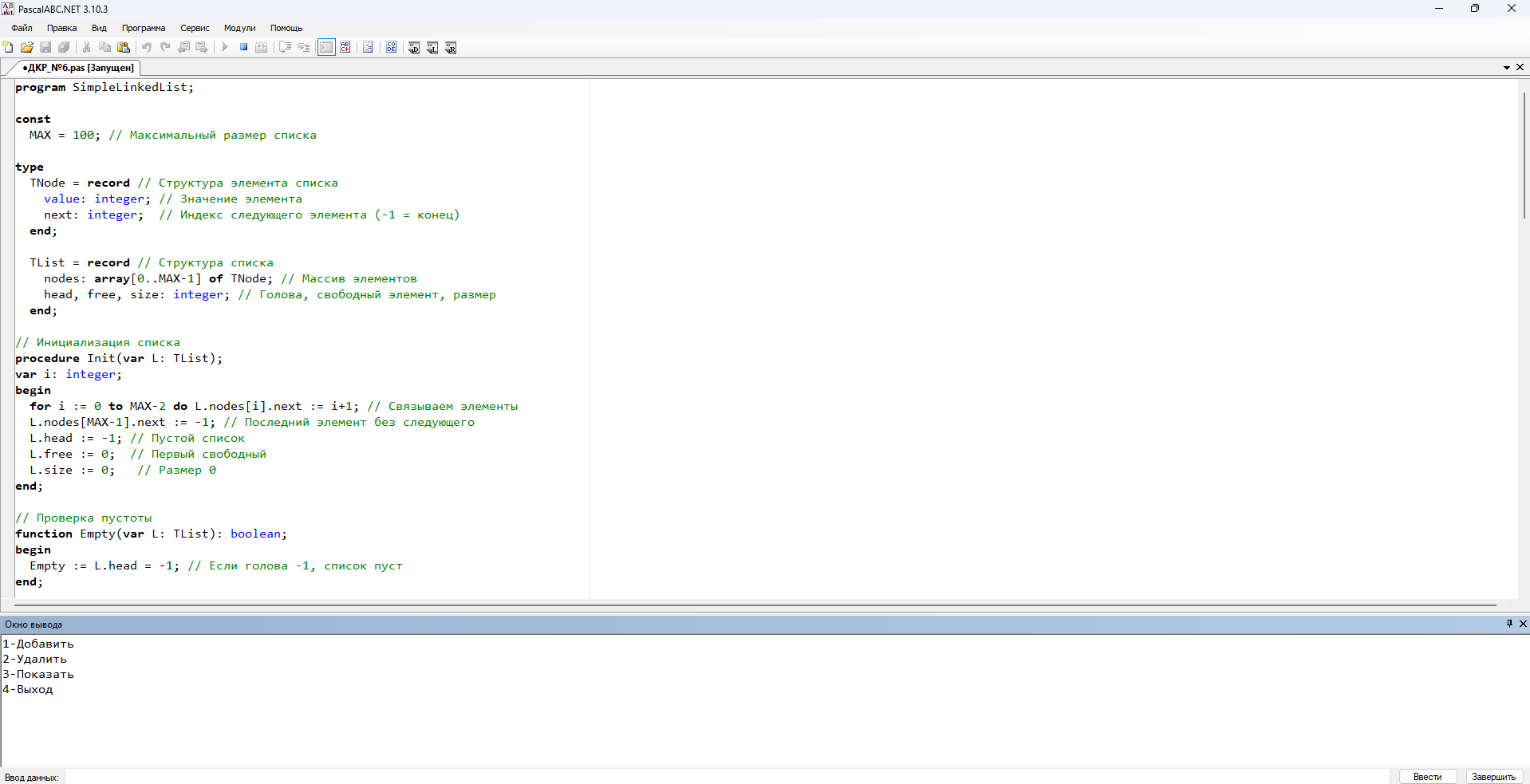
**else** writeln('Ошибка: неверная команда'); // Неверная команда

**end**;

**until** cmd = 4; // Выход по команде 4

**end**.

**Результат выполнения программы:**

  
Рисунок 7 – Запуск программы

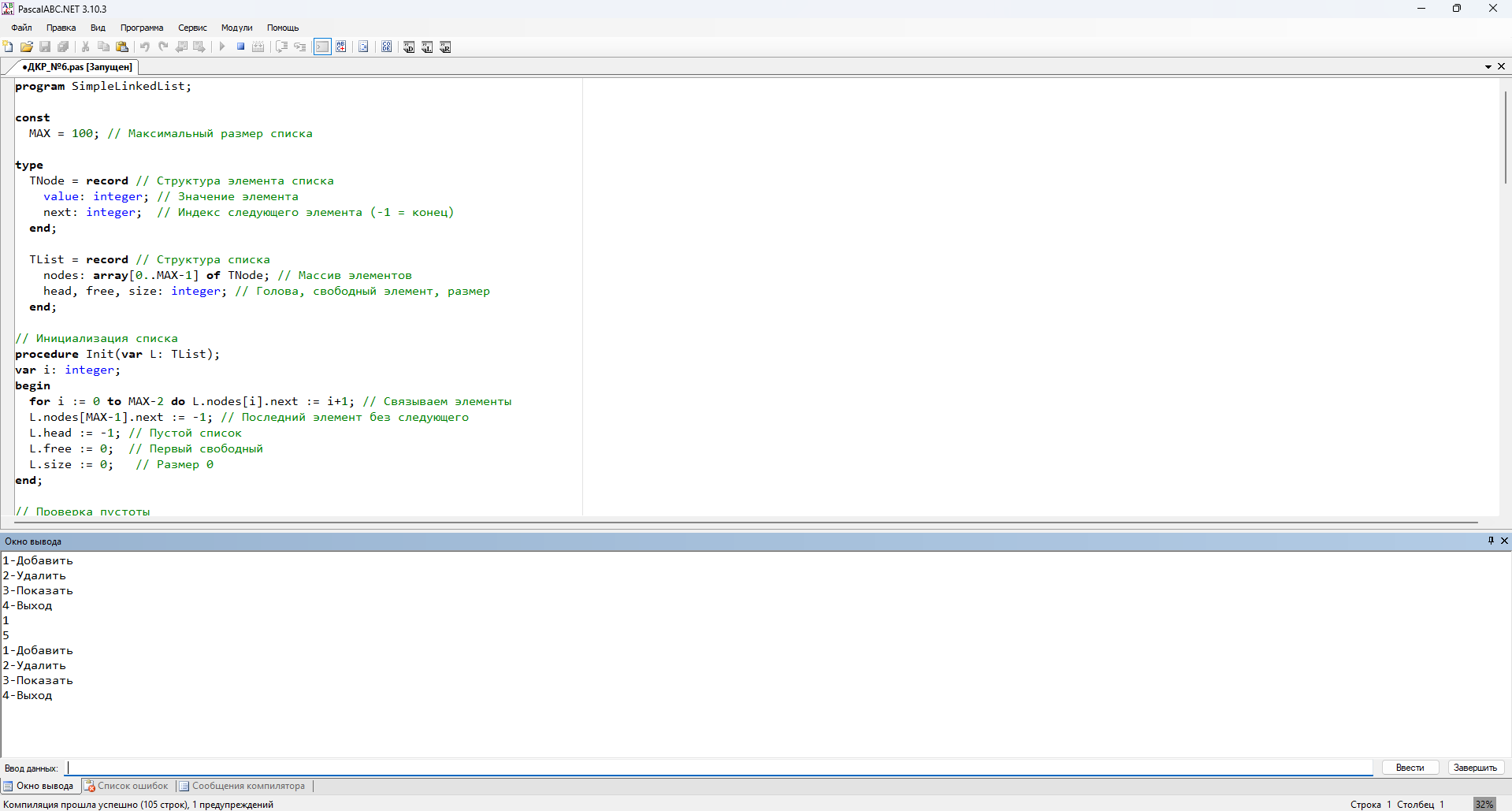


Рисунок 8 – Добавил число

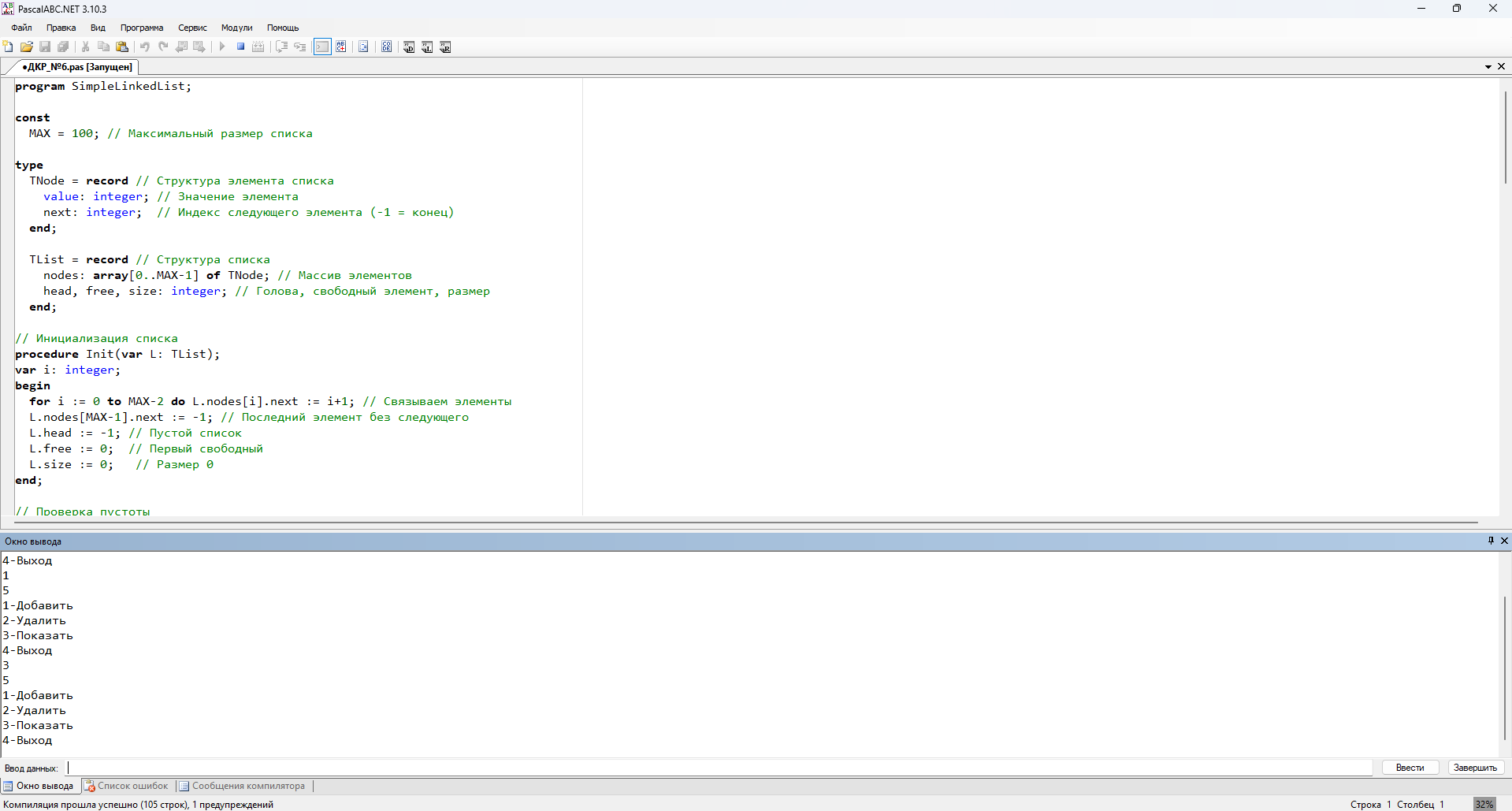


Рисунок 9 – Вывел число, которое показано на прошлом рисунке

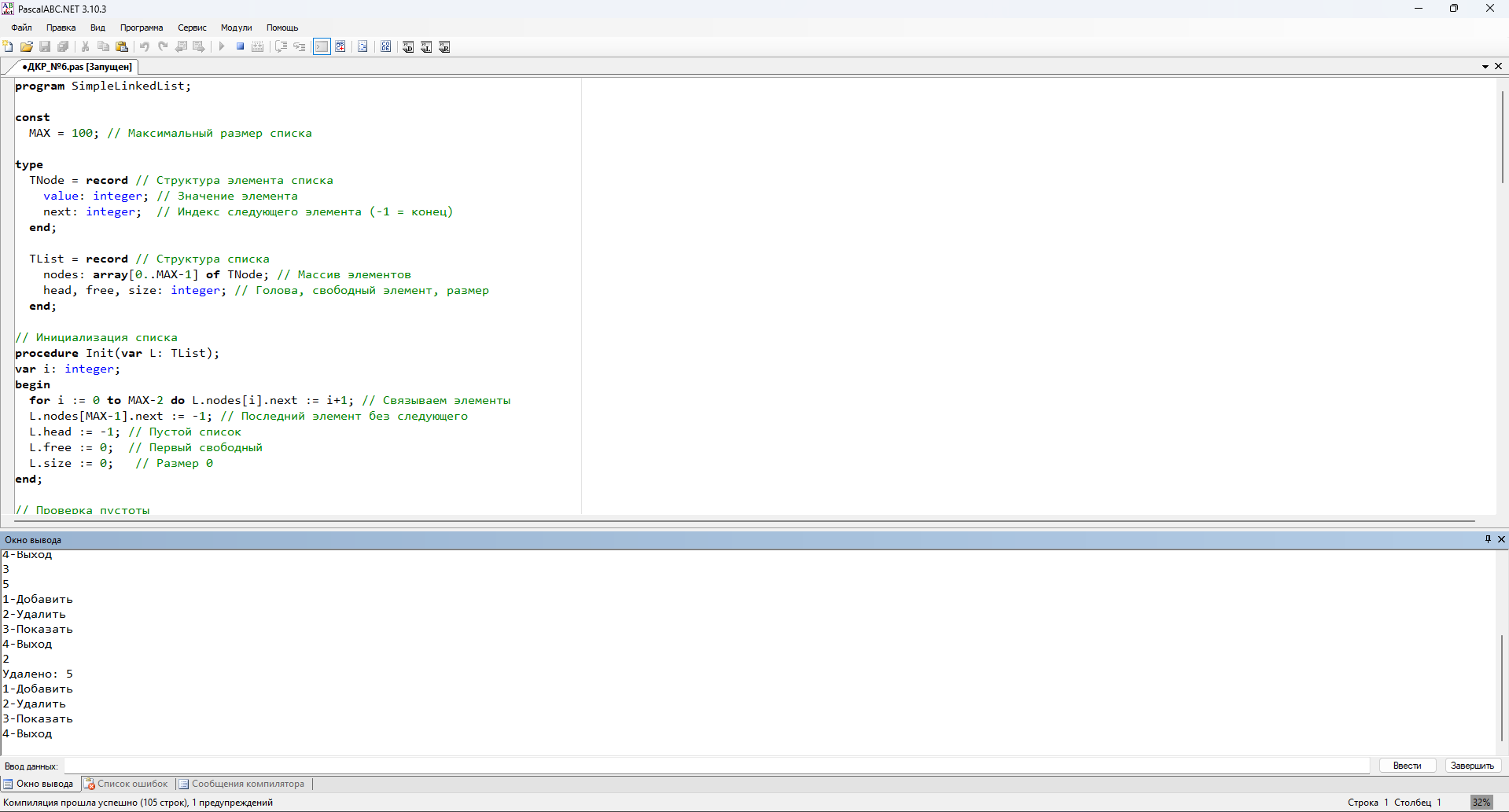


Рисунок 10 – Удалил текущее число

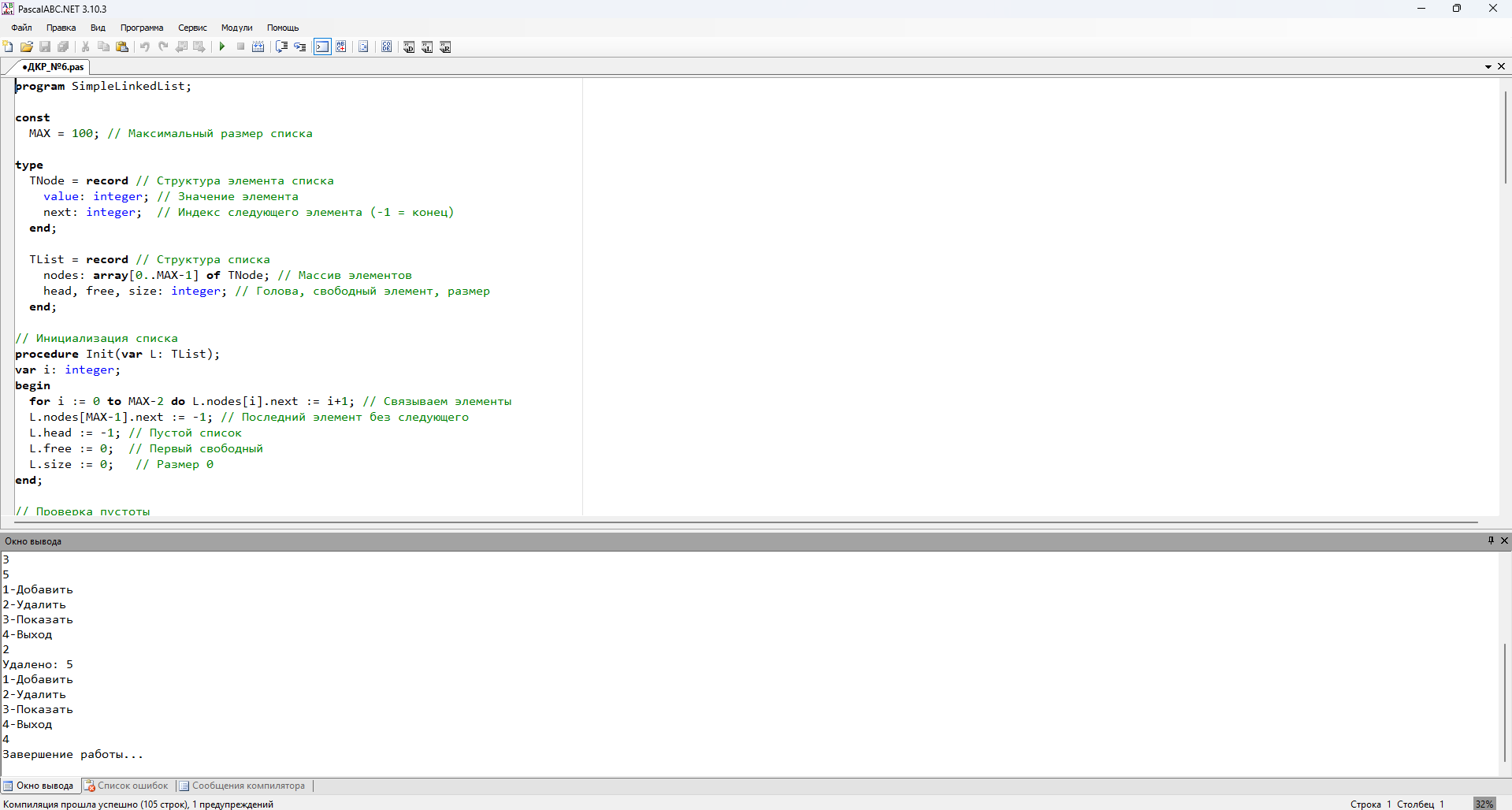


Рисунок 11 – Закончил работу программы

**Вывод:** Данная реализация демонстрирует принципы работы односвязного списка на статической памяти. Код соответствует поставленной задаче, обладает четкой структурой и может служить основой для более сложных модификаций.