Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №2**

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

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

Выполнил: студент учебной группы ИСПк- 204-52-00

Никулин Арсений Игоревич

Преподаватель:

Сергеева Елизавета Григорьевна

г. Киров

2024

1. **Цель работы:** изучение принципов работы с базовыми структурами данных, получение навыков организации case-меню.
2. **Задание:**
3. Написать программу для работы со структурой данных «Двусвязный список».
4. Структура данных должна быть реализована на основе статической памяти.
5. Работа со структурой должна осуществляться с помощью case-меню. Предусмотреть наглядную визуализацию содержимого структуры.
6. **Описание алгоритма:**

1) Создание списка. Программа начинает с создания пустого двусвязного списка. В зависимости от режима работы (статической или динамической памяти) выделяется необходимое количество памяти для начальных структур данных.

2) Вставка элемента в начало списка. Пользователь может выбрать опцию вставки нового элемента в начало списка. Программа запрашивает данные у пользователя, создает новый узел списка, и затем устанавливает указатель на начало списка на новый узел.

3) Вставка элемента в конец списка. Эта функция позволяет пользователю добавить новый элемент в конец списка. Программа запрашивает данные у пользователя, создает новый узел, и затем корректно устанавливает соответствующие указатели.

4) Вставка элемента в середину списка. Пользователь указывает позицию, куда необходимо вставить новый элемент, а также предоставляет данные для этого элемента. Программа находит указанный узел и вставляет новый узел перед ним, корректно меняя соответствующие указатели.

5) Удаление элемента из списка. Пользователь может удалить элемент из списка, указав его позицию. Программа находит указанный узел и правильно перенаправляет указатели, чтобы исключить его из списка.

6) Вывод списка. Эта функция выводит содержимое списка на экран. Программа перебирает все узлы списка, выводя их данные на экран, а также ссылки на предыдущий и следующий узлы.

7) Очистка списка. Все элементы списка удаляются, а выделенная для них память освобождается.

8) Выход из программы. Пользователь может завершить выполнение программы, выбрав соответствующую опцию из меню.

1. **Схема алгоритма:**

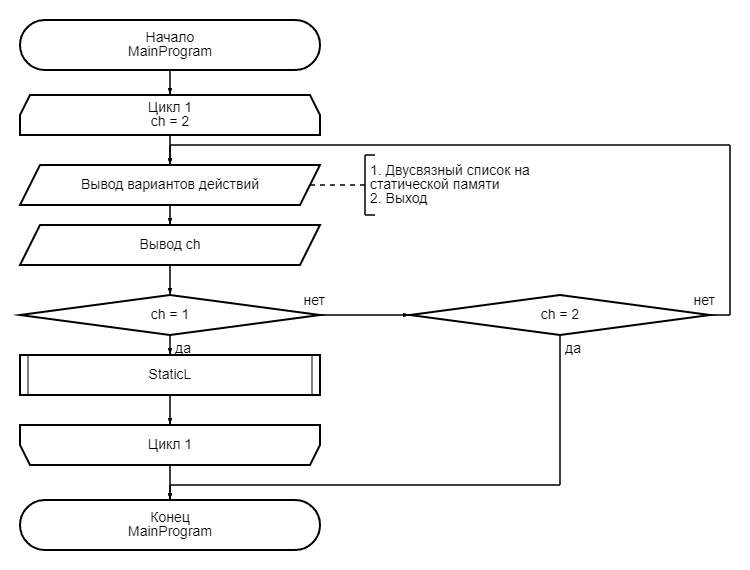
****

Рисунок 1 – Основная программа

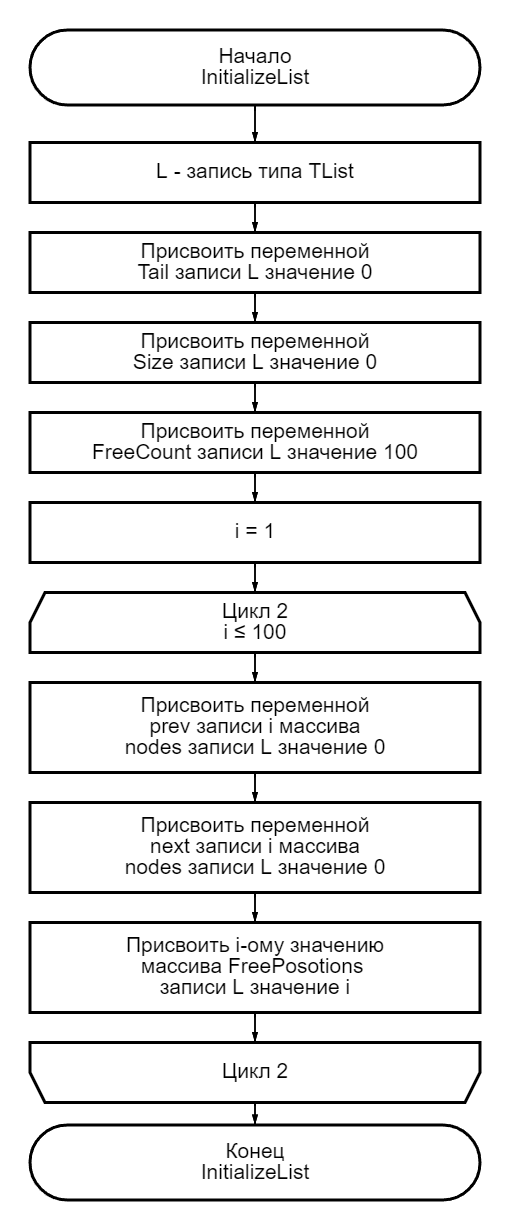
****

Рисунок 2 – Схема алгоритма процедуры InitializeList

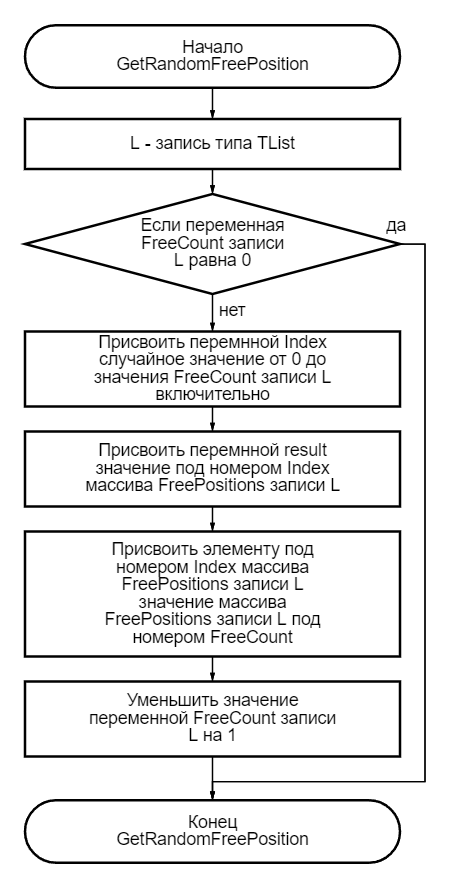


Рисунок 3 – Схема алгоритма функции GetRandomFreePosition

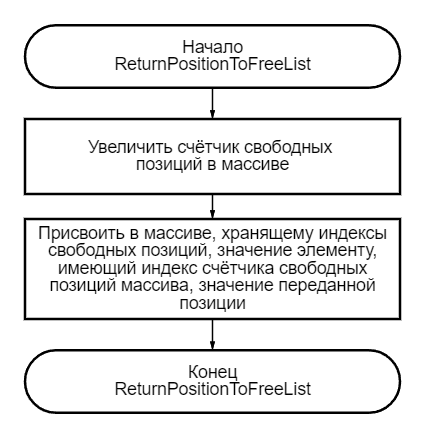


Рисунок 4 – Схема алгоритма процедуры ReturnPositionToFreeList

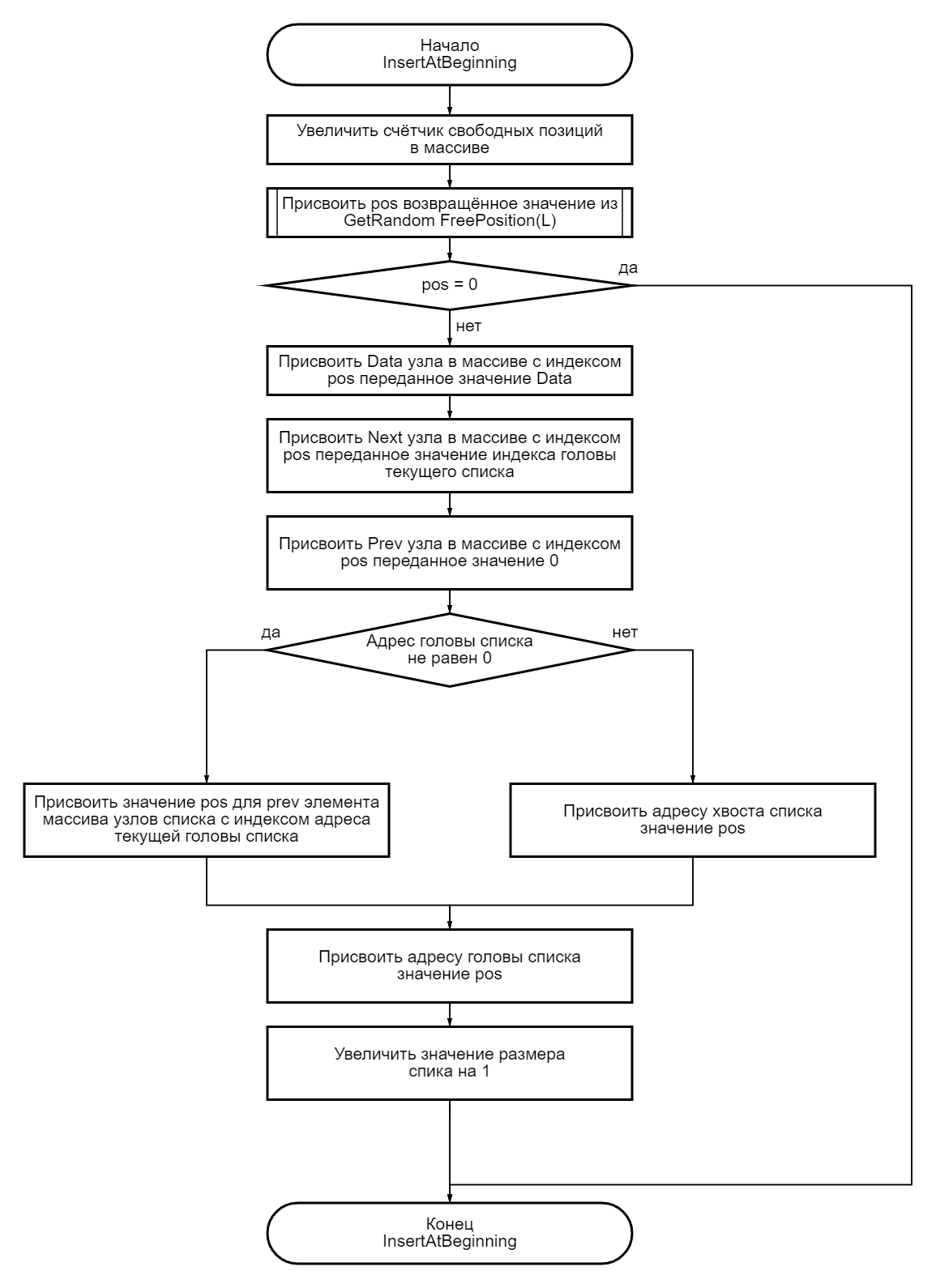


Рисунок 5 – Схема алгоритма процедуры InsertAtBeginning

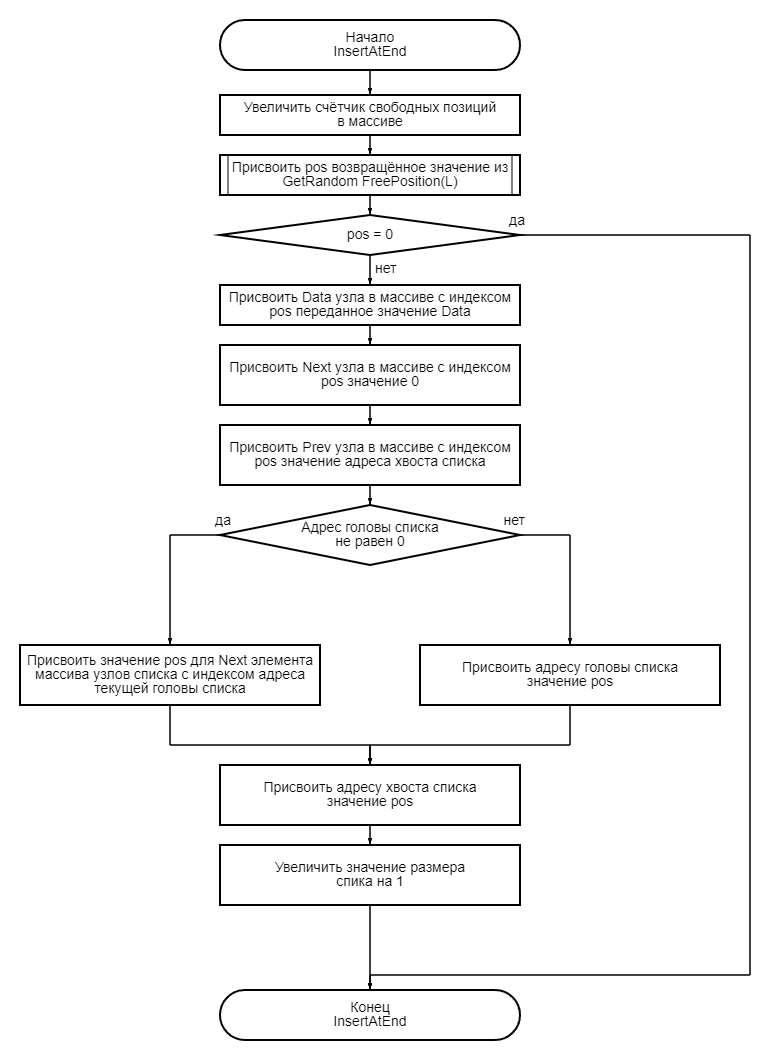


Рисунок 6 – Схема алгоритма процедуры InsertAtEnd

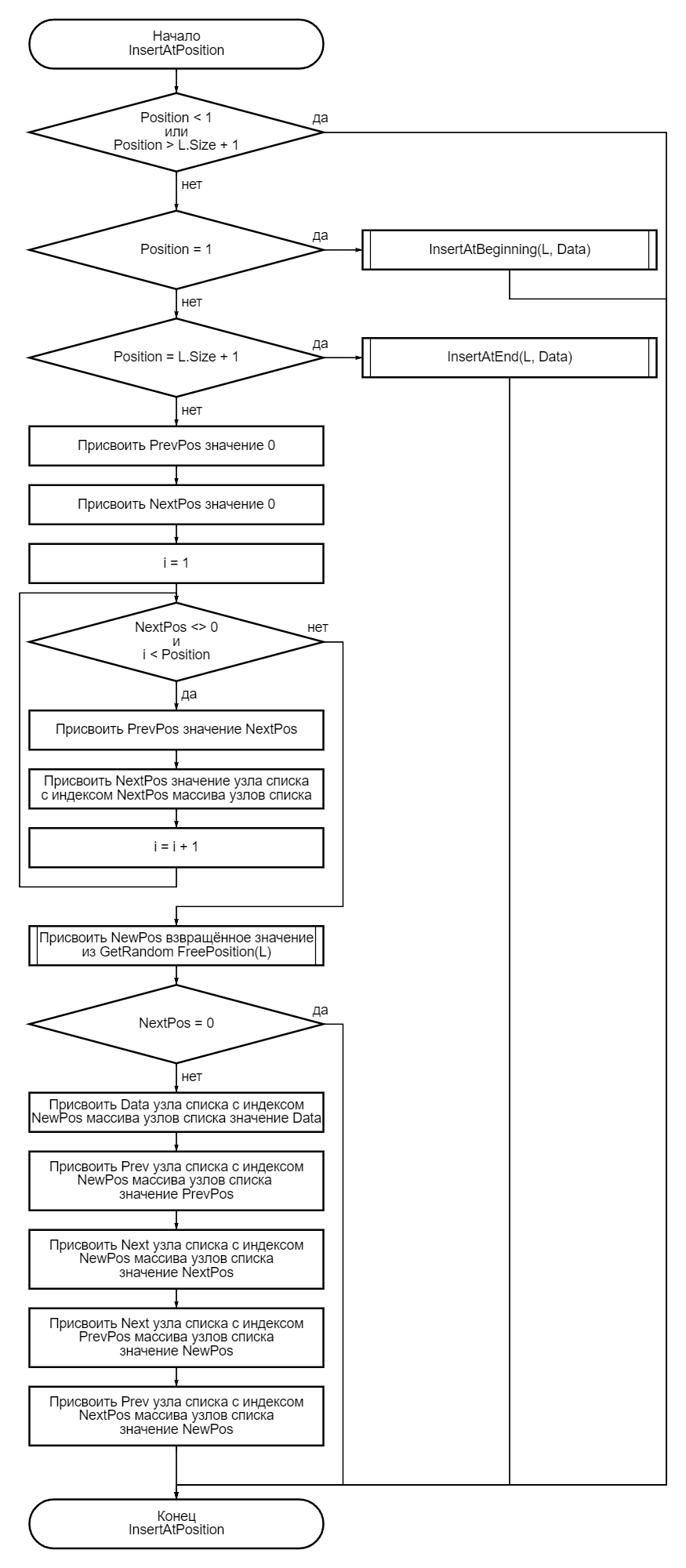


Рисунок 7 – Схема алгоритма процедуры InsertAtPosition

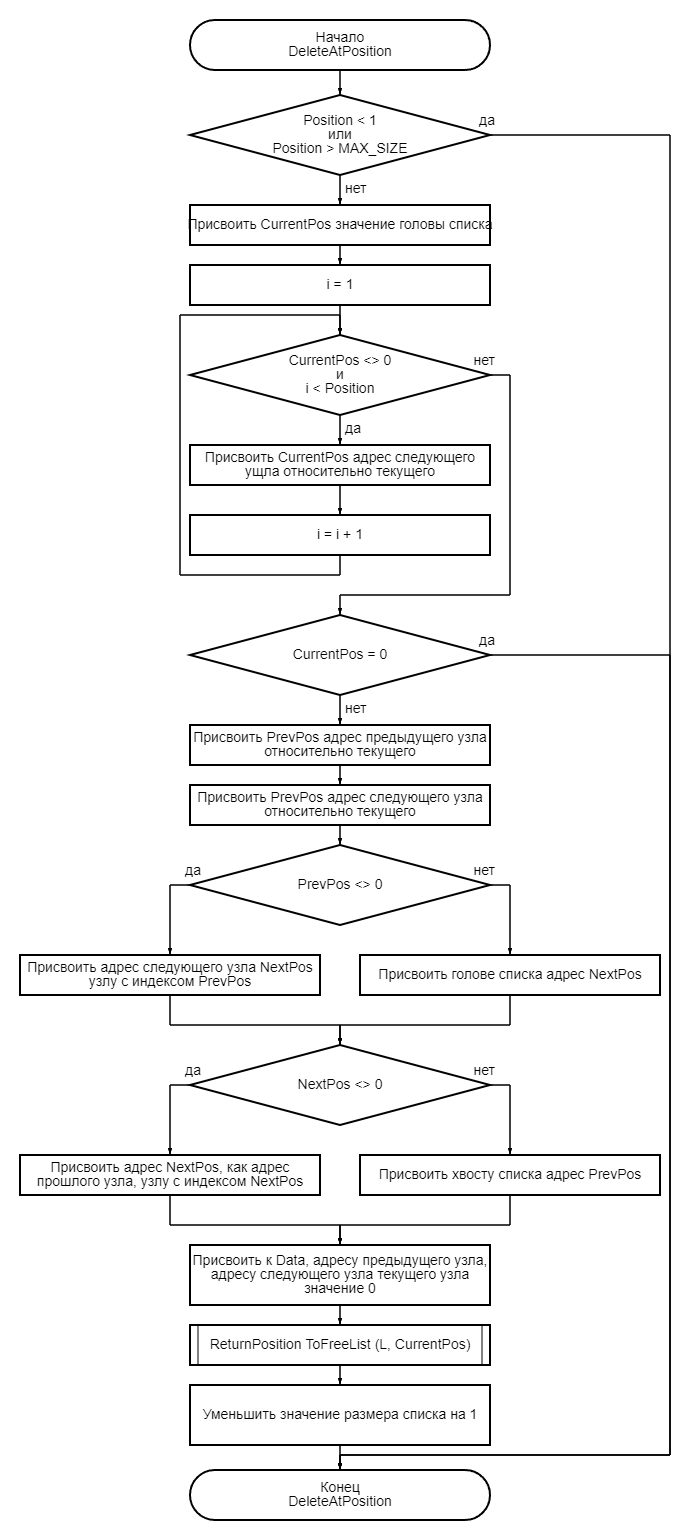


Рисунок 8 – Схема алгоритма процедуры DeleteAtPosition

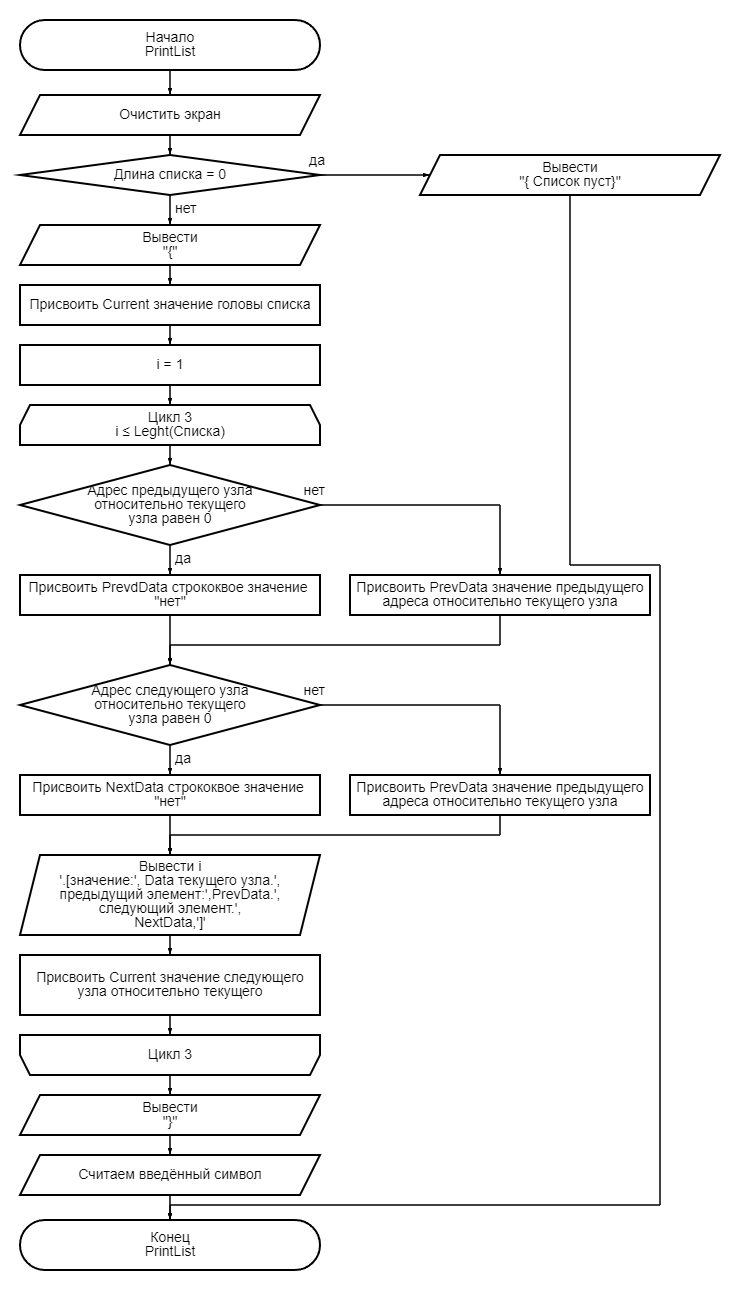


Рисунок 9 – Схема алгоритма процедуры PrintList

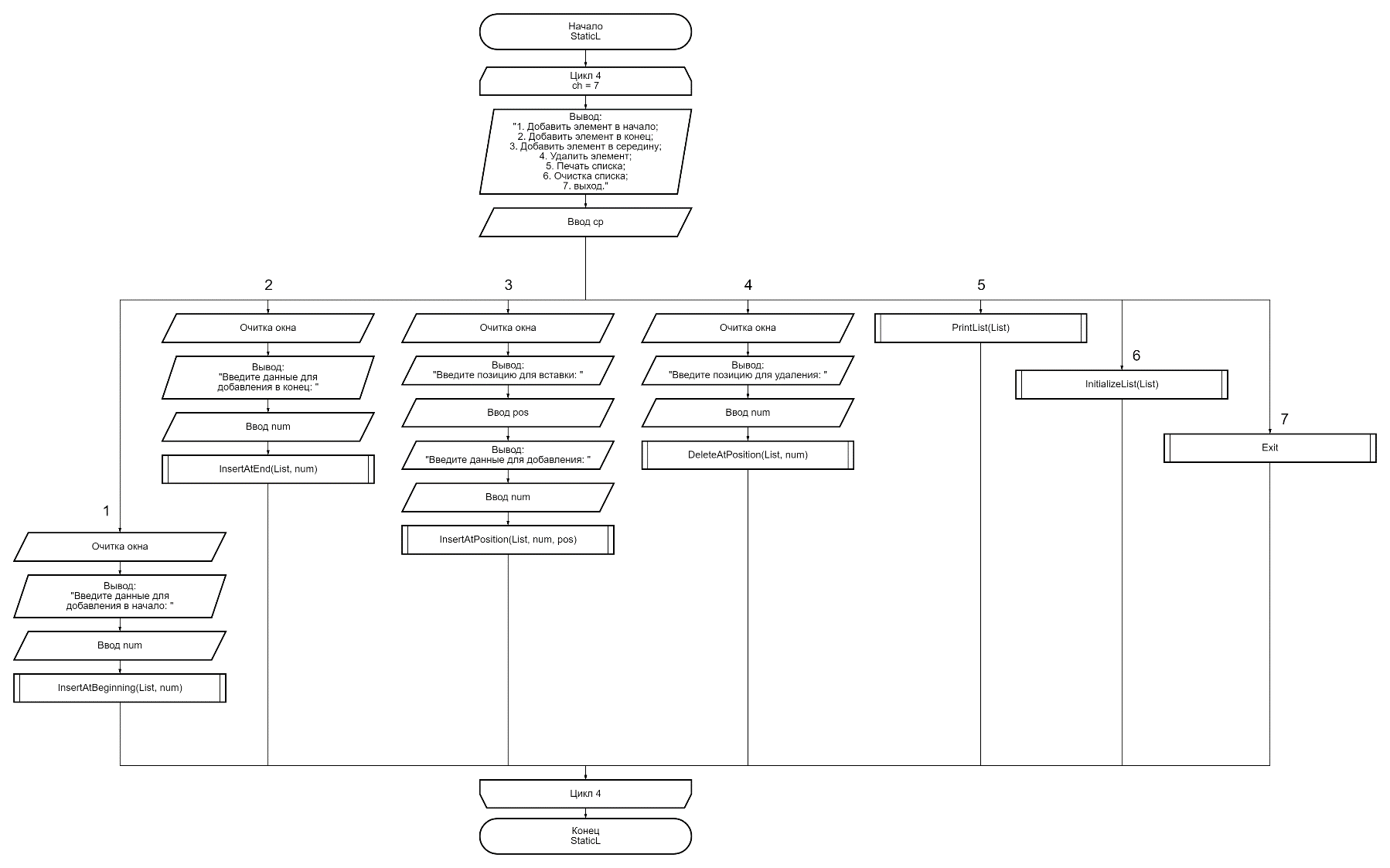


Рисунок 10 – Схема алгоритма процедуры StaticL

1. **Код программы**

**MainProgram:**

**program** MainProgram;

**uses** crt, StaticLs;

**var**

choice, punkt: integer;

ch: integer;

**begin**

**repeat**

ClrScr;

writeln('1. Двусвязный список на статической памяти');

writeln('2. Выход');

readln(ch);

**case** ch **of**

1: StaticL;

2: **Exit**

**end**;

**until** ch = 2;

**end**.

**StaticLs:**

**unit** StaticLs;

**interface**

**uses** crt;

**procedure** StaticL;

**implementation**

**const**

MAX\_SIZE = 100;

**type**

TNode = **record**

Data: Integer;

Prev, Next: Integer;

**end**;

TList = **record**

Nodes: **array**[1..MAX\_SIZE] **of** TNode;

Head, Tail: Integer;

Size: Integer;

FreePositions: **array**[1..MAX\_SIZE] **of** Integer;

FreeCount: Integer;

**end**;

**var**

List: TList;

punkt, num, pos: integer;

ch: integer;

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

**var**

i: Integer;

**begin**

L.Head := 0;

L.Tail := 0;

L.Size := 0;

L.FreeCount := MAX\_SIZE;

**for** i := 1 **to** MAX\_SIZE **do**

**begin**

L.Nodes[i].Prev := 0;

L.Nodes[i].Data:= 0;

L.Nodes[i].Next := 0;

L.FreePositions[i] := i;

**end**;

**end**;

**function** GetRandomFreePosition(**var** L: TList): Integer;

**var**

Index: Integer;

**begin**

**if** L.FreeCount = 0 **then Exit**;

Index := Random(L.FreeCount) + 1;

Result := L.FreePositions[Index];

L.FreePositions[Index] := L.FreePositions[L.FreeCount];

Dec(L.FreeCount);

**end**;

**procedure** ReturnPositionToFreeList(**var** L: TList; Position: Integer);

**begin**

Inc(L.FreeCount);

L.FreePositions[L.FreeCount] := Position;

**end**;

**procedure** InsertAtBeginning(**var** L: TList; Data: Integer);

**var**

Pos: Integer;

**begin**

Pos := GetRandomFreePosition(L);

**if** Pos = 0 **then Exit**;

L.Nodes[Pos].Data := Data;

L.Nodes[Pos].Next := L.Head;

L.Nodes[Pos].Prev := 0;

**if** L.Head <> 0 **then**

L.Nodes[L.Head].Prev := Pos

**else**

L.Tail := Pos;

L.Head := Pos;

Inc(L.Size);

**end**;

**procedure** InsertAtEnd(**var** L: TList; Data: Integer);

**var**

Pos: Integer;

**begin**

Pos := GetRandomFreePosition(L);

**if** Pos = 0 **then Exit**;

L.Nodes[Pos].Data := Data;

L.Nodes[Pos].Prev := L.Tail;

L.Nodes[Pos].Next := 0;

**if** L.Tail <> 0 **then**

L.Nodes[L.Tail].Next := Pos

**else**

L.Head := Pos;

L.Tail := Pos;

Inc(L.Size);

**end**;

**procedure** InsertAtPosition(**var** L: TList; Data, Position: Integer);

**var**

PrevPos, NextPos, NewPos, i: Integer;

**begin**

**if** (Position < 1) **or** (Position > L.Size + 1) **then Exit**;

**if** Position = 1 **then**

InsertAtBeginning(L, Data)

**else if** Position = L.Size + 1 **then**

InsertAtEnd(L, Data)

**else**

**begin**

PrevPos := 0;

NextPos := L.Head;

i := 1;

**while** (NextPos <> 0) **and** (i < Position) **do**

**begin**

PrevPos := NextPos;

NextPos := L.Nodes[NextPos].Next;

Inc(i);

**end**;

NewPos := GetRandomFreePosition(L);

**if** NewPos = 0 **then Exit**;

L.Nodes[NewPos].Data := Data;

L.Nodes[NewPos].Prev := PrevPos;

L.Nodes[NewPos].Next := NextPos;

L.Nodes[PrevPos].Next := NewPos;

L.Nodes[NextPos].Prev := NewPos;

Inc(L.Size);

**end**;

**end**;

**procedure** DeleteAtPosition(**var** L: TList; Position: Integer);

**var**

PrevPos, NextPos, CurrentPos, i: Integer;

**begin**

**if** (Position < 1) **or** (Position > MAX\_SIZE) **then Exit**;

CurrentPos := L.Head;

i := 1;

**while** (CurrentPos <> 0) **and** (i < Position) **do**

**begin**

CurrentPos := L.Nodes[CurrentPos].Next;

Inc(i);

**end**;

**if** CurrentPos = 0 **then Exit**;

PrevPos := L.Nodes[CurrentPos].Prev;

NextPos := L.Nodes[CurrentPos].Next;

**if** PrevPos <> 0 **then**

L.Nodes[PrevPos].Next := NextPos

**else**

L.Head := NextPos;

**if** NextPos <> 0 **then**

L.Nodes[NextPos].Prev := PrevPos

**else**

L.Tail := PrevPos;

L.Nodes[CurrentPos].Data := 0;

L.Nodes[CurrentPos].Prev := 0;

L.Nodes[CurrentPos].Next := 0;

ReturnPositionToFreeList(L, CurrentPos);

Dec(L.Size);

**end**;

**procedure** PrintList(**const** L: TList);

**var**

i, Current: Integer;

PrevData, NextData: string;

**begin**

ClrScr;

**if** L.Size = 0 **then**

**begin**

WriteLn('{ Список пуст }');

**Exit**;

**end**;

WriteLn('{');

Current := L.Head;

**for** i := 1 **to** L.Size **do**

**begin**

**if** L.Nodes[Current].Prev = 0 **then**

PrevData := '\*нет\*'

**else**

PrevData := IntToStr(L.Nodes[Current].Prev);

**if** L.Nodes[Current].Next = 0 **then**

NextData := '\*нет\*'

**else**

NextData := IntToStr(L.Nodes[Current].Next);

WriteLn(i, '. [значение:', L.Nodes[Current].Data,

', предыдущий элемент:', PrevData,

', следующий элемент:', NextData, ']');

Current := L.Nodes[Current].Next;

**end**;

WriteLn('}');

readln;

**end**;

**var**

Option, Data, Position: Integer;

**procedure** StaticL;

**begin**

InitializeList(List);

**repeat**

writeln('1. Добавить элемент в начало');

writeln('2. Добавить элемент в конец');

writeln('3. Добавить элемент в середину');

writeln('4. Удалить элемент');

writeln('5. Печать списка');

writeln('6. Очистка списка');

writeln('7. выход');

Readln(ch);

**case** ch **of**

1:

**begin**

clrscr;

Write('Введите данные для добавления в начало: ');

ReadLn(num);

InsertAtBeginning(List, num);

**end**;

2:

**begin**

clrscr;

Write('Введите данные для добавления в конец: ');

ReadLn(num);

InsertAtEnd(List, num);

**end**;

3:

**begin**

clrscr;

Write('Введите позицию для вставки: ');

ReadLn(pos);

Write('Введите данные для добавления: ');

ReadLn(num);

InsertAtPosition(List, num, pos);

**end**;

4:

**begin**

clrscr;

Write('Введите позицию для удаления: ');

ReadLn(num);

DeleteAtPosition(List, num);

**end**;

5: PrintList(List);

6: InitializeList(List);

7: **Exit**;

**end**;

**until** ch = 7;

**end**;

**end**.

1. **Результат выполнения программы**

****

Рисунок 11 ­– Результат выполнения программы



Рисунок 12 ­– Результат выполнения программы



Рисунок 13 ­– Результат выполнения программы

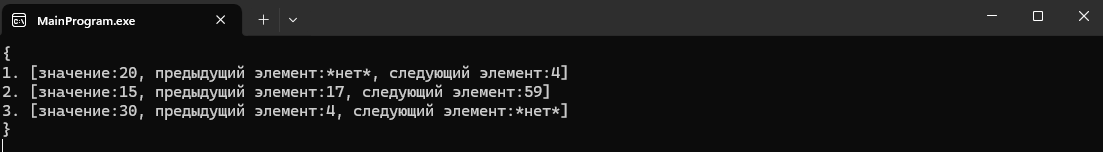


Рисунок 14 ­– Результат выполнения программы

1. **Вывод**

В процессе проведения данной лабораторной работы была успешно достигнута главная цель - изучение основ работы с базовыми структурами данных и освоение навыков создания case-меню. Разработанная программа позволяет пользователю манипулировать двусвязным списком, реализованным с использованием статической памяти.

Основные функции программы включают добавление элементов в начало, конец и середину списка, их удаление, а также отображение содержимого списка и его очистку. Каждая из этих операций выполнена в соответствии с описанными алгоритмами, что гарантирует правильное выполнение действий с данными.