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-204-52-00

Титков Дмитрий Михайлович

Преподаватель:

Сергеева Елизавета Григорьевна

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.

Описание алгоритма

Была реализована программа для работы со структурой данных «Двусвязный список», использующая статическое распределение памяти.  
Двусвязный список — это структура данных, в которой каждый элемент (узел) содержит ссылки как на предыдущий, так и на следующий элемент, что позволяет эффективно вставлять и удалять элементы с обеих сторон.

Память для узлов списка заранее выделяется в виде массива фиксированной длины. Для управления свободными ячейками используется связный список свободных узлов. Это позволяет обойтись без динамического выделения памяти в процессе выполнения программы.

Программа содержит следующие основные процедуры:

* + - InitList — инициализирует список, создаёт цепочку свободных узлов и подготавливает список к работе.
    - AllocateNode — реализует алгоритм выделения узла из пула свободных ячеек.
    - DeallocateNode — возвращает освобождённый узел обратно в пул.
    - InsertFront и InsertBack — добавляют элементы в начало и конец списка соответственно, корректируя указатели prev и next соседних элементов.
    - DeleteFront и DeleteBack — удаляют элементы с соответствующих концов списка, освобождая узлы.
    - DisplayList — наглядно выводит содержимое списка на экран в виде цепочки значений, соединённых стрелками.

Все действия реализованы в основном теле программы с использованием case-меню, которое предлагает пользователю выбрать нужное действие: вставка элемента, удаление, вывод списка или выход. Благодаря использованию статической памяти, программа не зависит от операционной системы для выделения памяти и хорошо подходит для работы в ограниченных средах.

1. Схема алгоритма с комментариями.

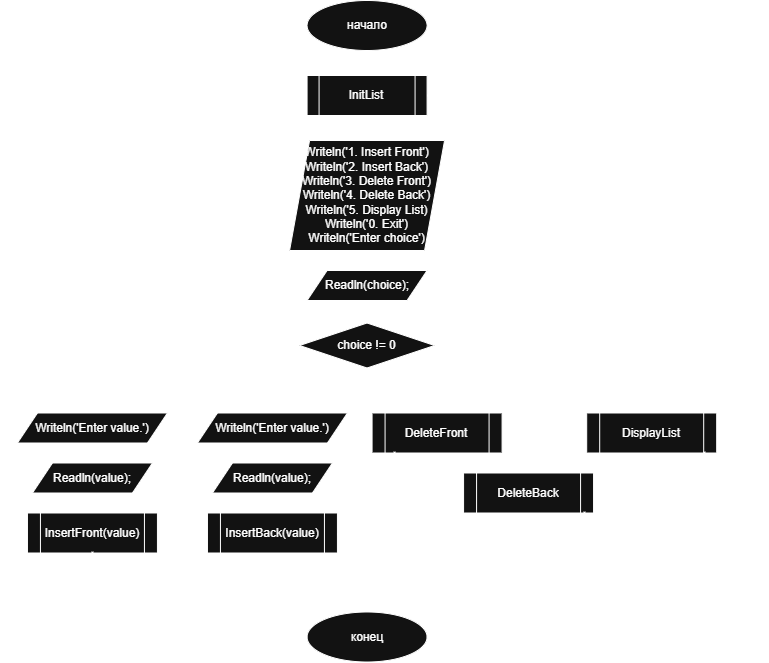


Рисунок 1 — Тело программы с case-меню

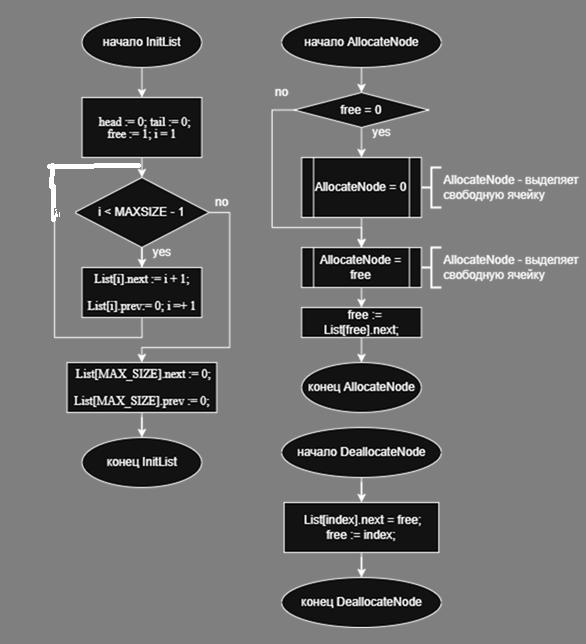


Рисунок 2 — Процедуры инициализации листа, выделения свободной ячейки и освобождения ячейки памяти

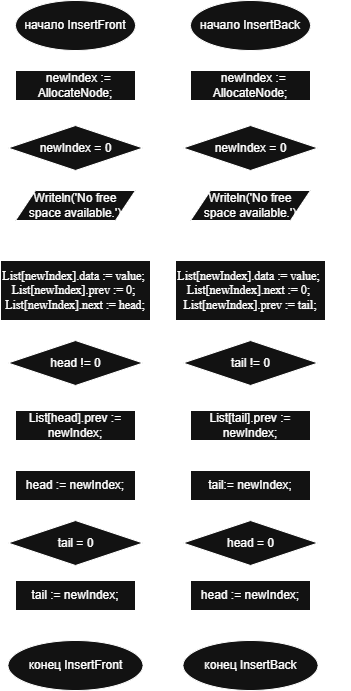


Рисунок 3 — Процедуры добавления элемента в начало и конец списка

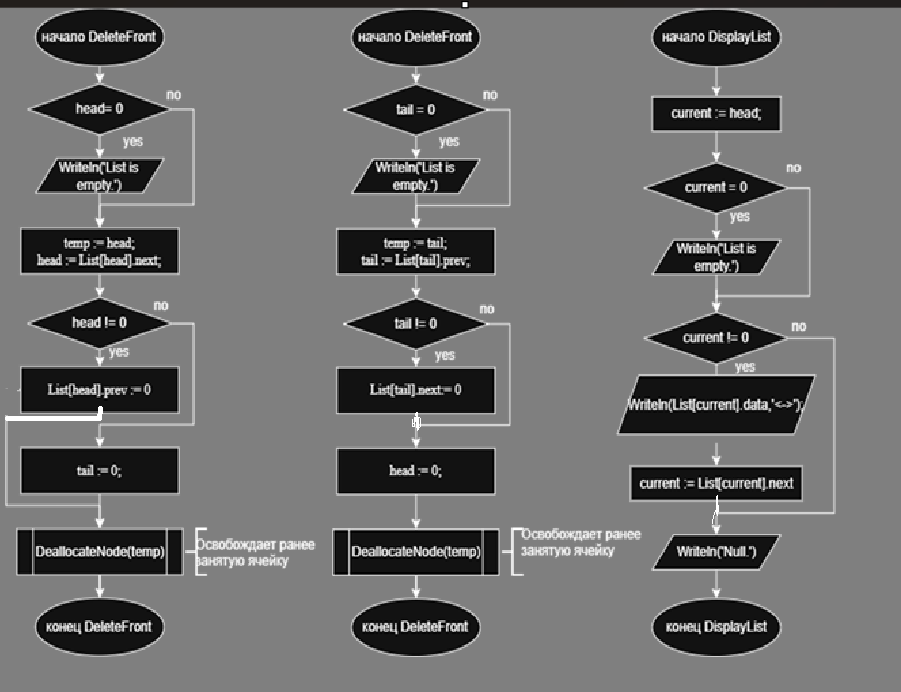


Рисунок 4 — Процедуры удаления элемента из начала и конца списка, отображения списка

Код программы

program StaticDoublyLinkedList;

const

MAX\_SIZE = 10;

type

Node = record

data: Integer;

prev, next: Integer;

end;

var

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

head, tail, free: Integer;

procedure InitList;

var

i: Integer;

begin

head := 0;

tail := 0;

free := 1;

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

begin

List[i].next := i + 1;

List[i].prev := 0;

end;

List[MAX\_SIZE].next := 0;

List[MAX\_SIZE].prev := 0;

end;

function AllocateNode: Integer;

begin

if free = 0 then

begin

AllocateNode := 0;

Exit;

end;

AllocateNode := free;

free := List[free].next;

end;

procedure DeallocateNode(index: Integer);

begin

List[index].next := free;

free := index;

end;

procedure InsertFront(value: Integer);

var

newIndex: Integer;

begin

newIndex := AllocateNode;

if newIndex = 0 then

begin

Writeln('No free space available.');

Exit;

end;

List[newIndex].data := value;

List[newIndex].prev := 0;

List[newIndex].next := head;

if head <> 0 then

List[head].prev := newIndex;

head := newIndex;

if tail = 0 then

tail := newIndex;

end;

procedure InsertBack(value: Integer);

var

newIndex: Integer;

begin

newIndex := AllocateNode;

if newIndex = 0 then

begin

Writeln('No free space available.');

Exit;

end;

List[newIndex].data := value;

List[newIndex].next := 0;

List[newIndex].prev := tail;

if tail <> 0 then

List[tail].next := newIndex;

tail := newIndex;

if head = 0 then

head := newIndex;

end;

procedure DeleteFront;

var

temp: Integer;

begin

if head = 0 then

begin

Writeln('List is empty.');

Exit;

end;

temp := head;

head := List[head].next;

if head <> 0 then

List[head].prev := 0

else

tail := 0;

DeallocateNode(temp);

end;

procedure DeleteBack;

var

temp: Integer;

begin

if tail = 0 then

begin

Writeln('List is empty.');

Exit;

end;

temp := tail;

tail := List[tail].prev;

if tail <> 0 then

List[tail].next := 0

else

head := 0;

DeallocateNode(temp);

end;

procedure DisplayList;

var

current: Integer;

begin

current := head;

if current = 0 then

begin

Writeln('List is empty.');

Exit;

end;

while current <> 0 do

begin

Write(List[current].data, ' <-> ');

current := List[current].next;

end;

Writeln('NULL');

end;

var

choice, value: Integer;

begin

InitList;

repeat

Writeln('1. Insert Front');

Writeln('2. Insert Back');

Writeln('3. Delete Front');

Writeln('4. Delete Back');

Writeln('5. Display List');

Writeln('0. Exit');

Write('Enter choice: ');

Readln(choice);

case choice of

1: begin

Write('Enter value: ');

Readln(value);

InsertFront(value);

end;

2: begin

Write('Enter value: ');

Readln(value);

InsertBack(value);

end;

3: DeleteFront;

4: DeleteBack;

5: DisplayList;

end;

until choice = 0;

end.

Результат выполнения программы

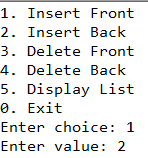


Рисунок 5 — результат выполнения 1 команды

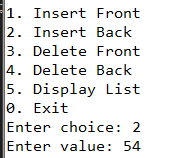


Рисунок 6 — результат выполнения 2 команды

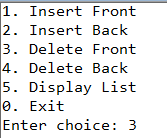


Рисунок 7 — результат выполнения 3 команды

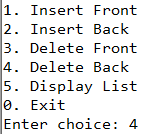


Рисунок 8 — результат выполнения 4 команды

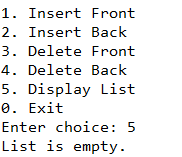


Рисунок 9 — результат выполнения 5 команды

Вывод

В ходе выполнения контрольной работы по реализации двусвязного списка были изучены основные понятия и принципы организации линейных динамических структур данных. Особое внимание было уделено реализации списка на статической памяти, что позволило закрепить знания о работе с массивами и управлении памятью без использования динамического распределения.

В рамках работы были реализованы основные алгоритмы вставки и удаления элементов с обеих сторон списка, а также методы выделения и освобождения узлов из заранее подготовленного пула памяти. Программа снабжена интерактивным case-меню, позволяющим пользователю управлять списком и наглядно визуализировать его текущее состояние.

В процессе выполнения задания были получены практические навыки построения структур данных с учётом ограниченных ресурсов памяти, а также укреплены навыки структурного программирования на языке Pascal. Программа построена модульно, с разделением логики работы с памятью, элементами списка и пользовательским интерфейсом, что делает её легко расширяемой и понятной.