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 И ДИНАМИЧЕСКОЙ ПАМЯТИ»**

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

Выполнил: студент учебной группы

ИСПк-203-52-00

Леушина Анна Станиславовна

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

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

**2. Формулировка задания (Вариант: 15)**

1. Написать программу для работы со структурой данных «Дек».

2. Структура данных должна быть реализована на основе статической памяти (массивы) и динамической памяти (указатели).

3. Работа со структурой должна осуществляться с помощью case-меню. Предусмотреть наглядную визуализацию содержимого структуры.

**3. Описание алгоритма**

Дек – структура данных, которая позволяет добавлять и удалять элементы как с начала, так и с конца очереди.

В статической реализации дека используется фиксированный массив для хранения элементов. Этот массив имеет заранее определенный размер, который не изменяется в процессе работы программы. Когда элемент добавляется в дек в конец или в начало, он помещается в следующую свободную ячейку массива. При этом осуществляется увеличение переменной, отслеживающей текущее количество элементов в деке. Операции удаления элементов также приводят к изменению размера этой переменной. В статической реализации нет необходимости в выделении или освобождении памяти при добавлении или удалении элементов, так как все пространство для хранения уже выделено заранее.

В динамической реализации дека используются связанные узлы для хранения элементов. Каждый узел содержит данные элемента и указатели на предыдущий и следующий узлы. Когда новый элемент добавляется в дек в начало или в конец, создается новый узел, который затем связывается с соответствующими узлами. При удалении элементов из дека, узлы могут быть динамически удалены из памяти, и связи между узлами обновляются. Это позволяет динамически изменять размер дека в зависимости от количества элементов, обеспечивая более гибкое управление памятью. Каждый новый узел требует отдельного выделения памяти, а при удалении узла его память освобождается.

**4. Схема алгоритма с комментариями**

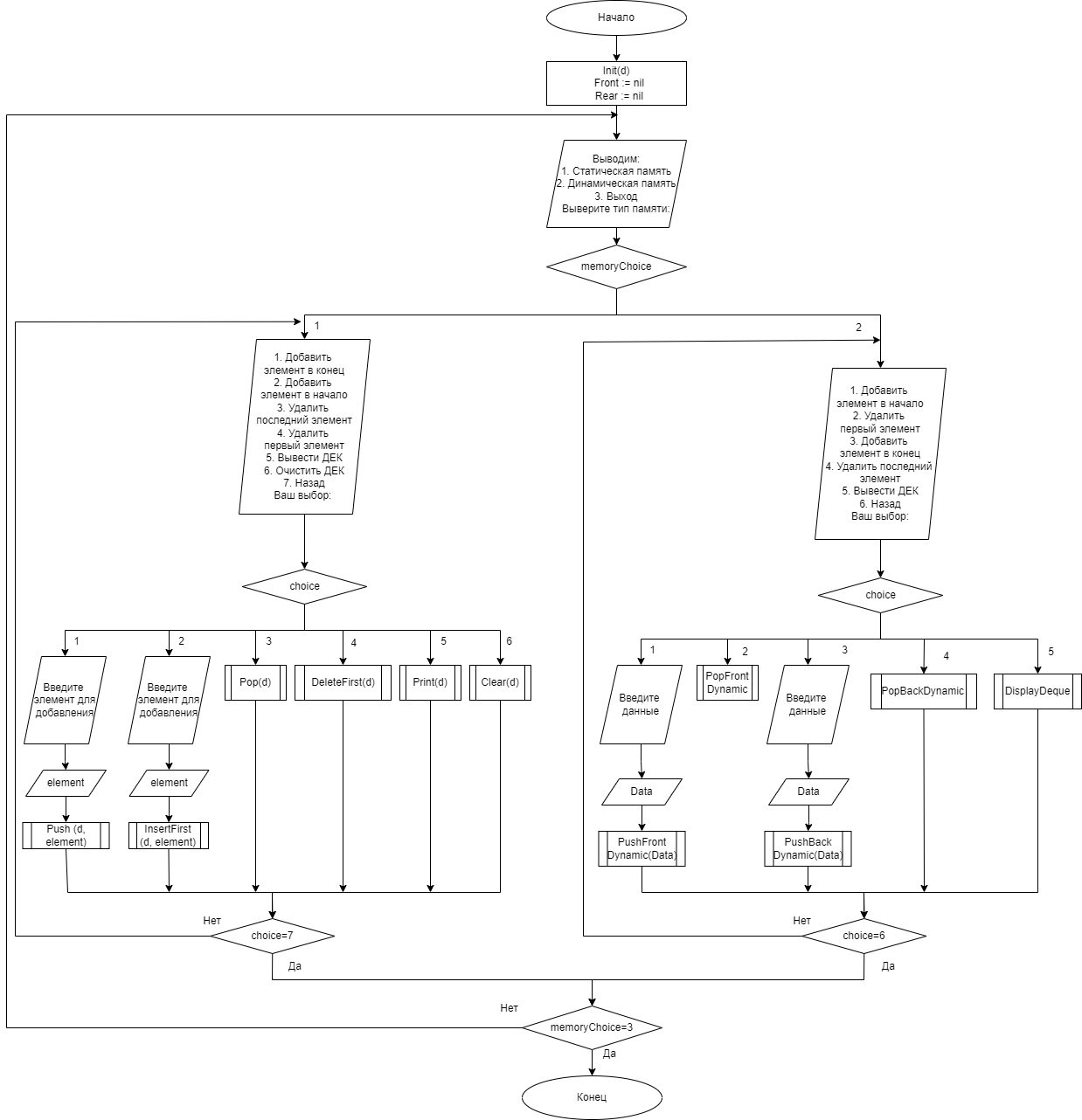


Рисунок 1 – Схема алгоритма case-меню

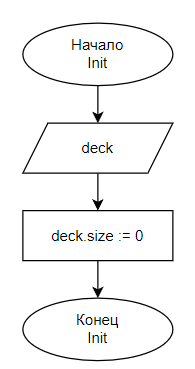


Рисунок 2 – Процедура Init

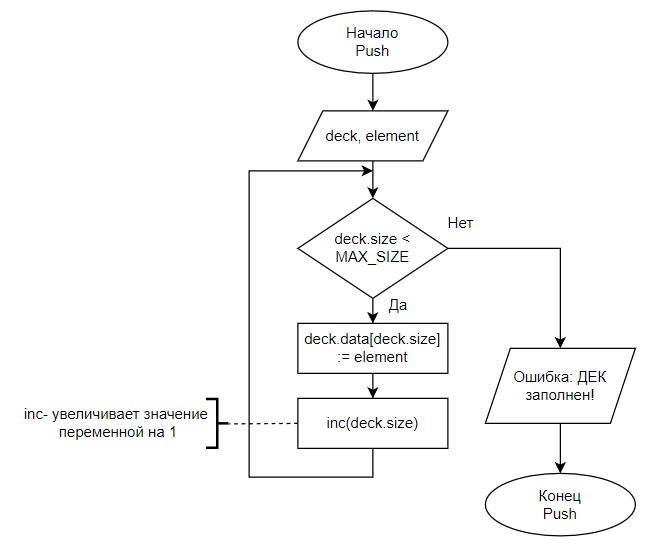


Рисунок 3 – Процедура Push

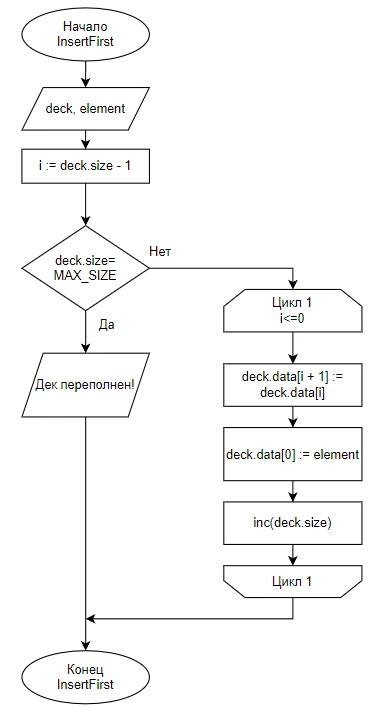


Рисунок 4 – Процедура InsertFirst

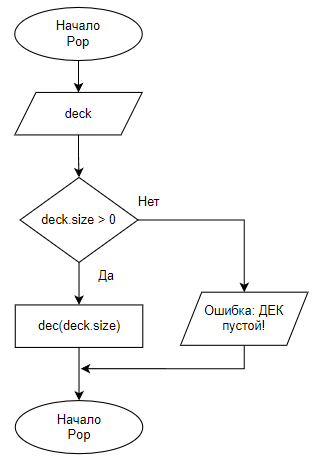


Рисунок 5 – Процедура Pop

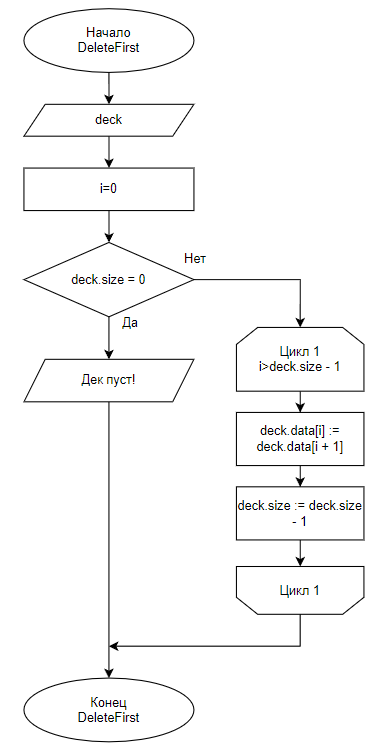


Рисунок 6 – Процедура DeleteFirst

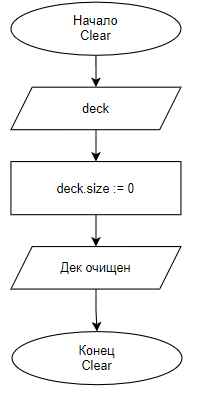


Рисунок 7 – Процедура Clear

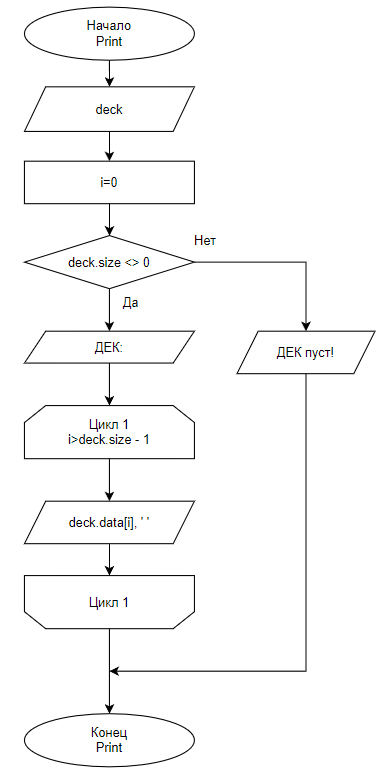


Рисунок 8 – Процедура Print

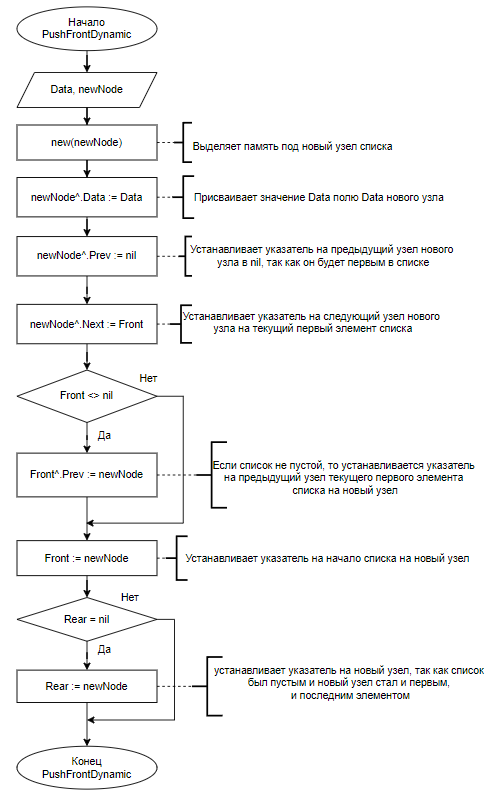


Рисунок 9 – Процедура PushFrontDynamic

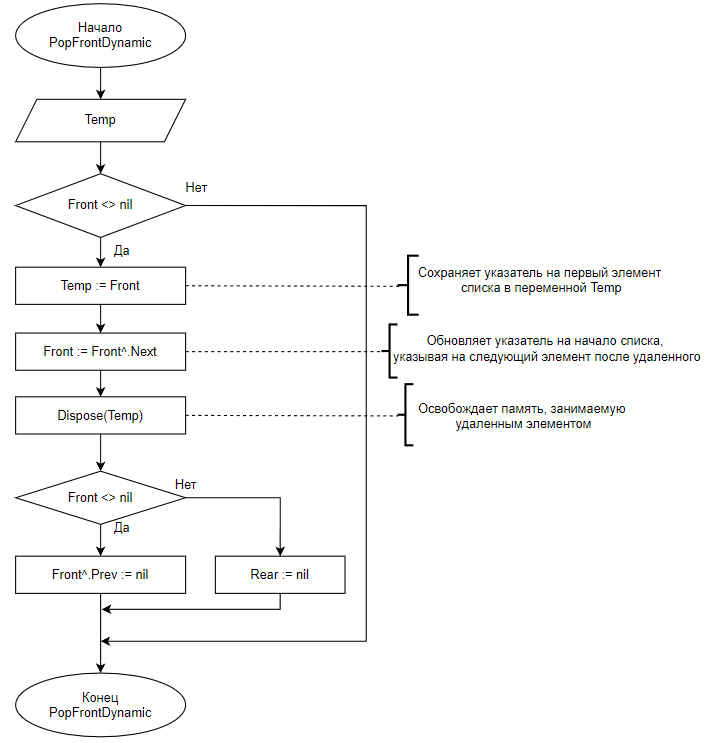


Рисунок 10 – Процедура PopFrontDynamic

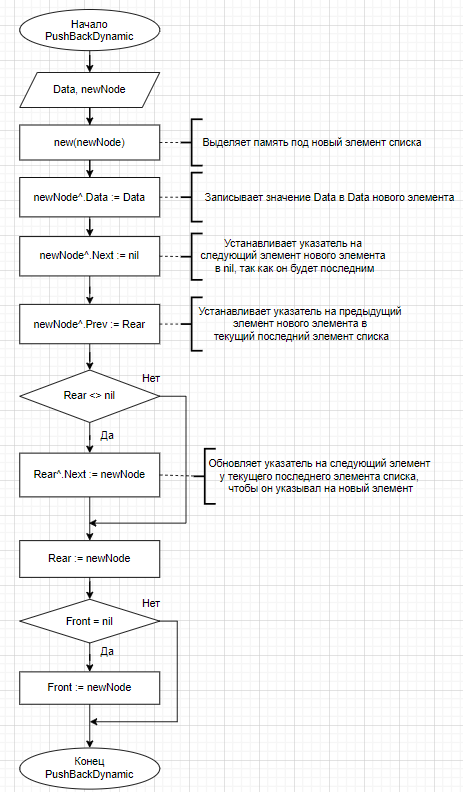


Рисунок 11 – PushBackDynamic

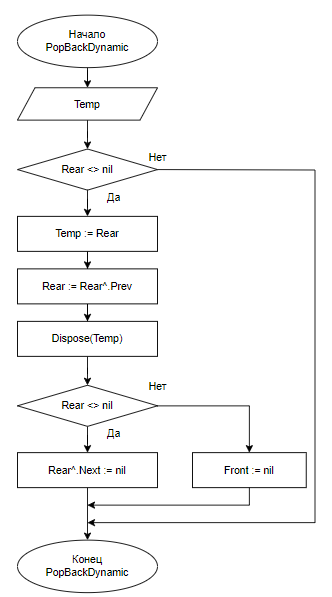


Рисунок 12 – Процедура PopBackDynamic

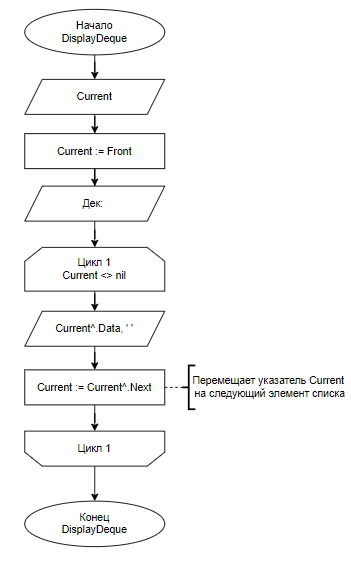


Рисунок 13 – Процедура DisplayDeque

**5. Код программы:**

**uses** crt;

**const**

MAX\_SIZE = 3;

**type**

PNode = ^Node;

Node = **record**

Data: Integer;

Next: PNode;

Prev: PNode;

**end**;

Deck = **record**

data: **array**[0..MAX\_SIZE] **of** Integer;

size: Integer;

**end**;

**var**

d: Deck;

Front, Rear: PNode;

choice, memoryChoice: Integer;

**procedure** Init(**var** deck: Deck);

**begin**

deck.size := 0;

**end**;

**procedure** Push(**var** deck: Deck; element: Integer);

**begin**

**if** deck.size < MAX\_SIZE **then**

**begin**

deck.data[deck.size] := element;

inc(deck.size);

**end**

**else**

writeln('Ошибка: ДЕК заполнен!');

**end**;

**procedure** InsertFirst(**var** deck: Deck; element: Integer);

**var**

i: Integer;

**begin**

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

writeln('Дек переполнен!')

**else**

**begin**

**for** i := deck.size - 1 **downto** 0 **do**

deck.data[i + 1] := deck.data[i];

deck.data[0] := element;

inc(deck.size)

**end**;

**end**;

**procedure** Pop(**var** deck: Deck);

**begin**

**if** deck.size > 0 **then**

dec(deck.size)

**else**

writeln('Ошибка: ДЕК пустой!');

**end**;

**procedure** DeleteFirst(**var** deck: Deck);

**var**

i: Integer;

**begin**

**if** deck.size = 0 **then**

writeln('Дек пуст!')

**else**

**begin**

**for** i := 0 **to** deck.size - 1 **do**

deck.data[i] := deck.data[i + 1];

deck.size := deck.size - 1;

**end**;

**end**;

**procedure** Clear(**var** deck: Deck);

**begin**

deck.size := 0;

writeln('Дек очищен');

**end**;

**procedure** Print(**const** deck: Deck);

**var**

i: Integer;

**begin**

**if** deck.size <> 0 **then**

**begin**

write('ДЕК: ');

**for** i := 0 **to** deck.size - 1 **do**

write(deck.data[i], ' ');

writeln

**end**

**else**

writeln('ДЕК пуст!');

**end**;

**procedure** PushFrontDynamic(Data: Integer);

**var**

newNode: PNode;

**begin**

**new**(newNode);

newNode^.Data := Data;

newNode^.Prev := nil;

newNode^.Next := Front;

**if** Front <> nil **then**

Front^.Prev := newNode;

Front := newNode;

**if** Rear = nil **then**

Rear := newNode;

**end**;

**procedure** PopFrontDynamic;

**var**

Temp: PNode;

**begin**

**if** Front <> nil **then**

**begin**

Temp := Front;

Front := Front^.Next;

Dispose(Temp);

**if** Front <> nil **then**

Front^.Prev := nil

**else**

Rear := nil;

**end**;

**end**;

**procedure** PushBackDynamic(Data: Integer);

**var**

newNode: PNode;

**begin**

**new**(newNode);

newNode^.Data := Data;

newNode^.Next := nil;

newNode^.Prev := Rear;

**if** Rear <> nil **then**

Rear^.Next := newNode;

Rear := newNode;

**if** Front = nil **then**

Front := newNode;

**end**;

**procedure** PopBackDynamic;

**var**

Temp: PNode;

**begin**

**if** Rear <> nil **then**

**begin**

Temp := Rear;

Rear := Rear^.Prev;

Dispose(Temp);

**if** Rear <> nil **then**

Rear^.Next := nil

**else**

Front := nil;

**end**;

**end**;

**procedure** DisplayDeque;

**var**

Current: PNode;

**begin**

Current := Front;

Write('Дек: ');

**while** Current <> nil **do**

**begin**

write(Current^.Data, ' ');

Current := Current^.Next;

**end**;

writeln;

**end**;

**begin**

Init(d);

Front := nil;

Rear := nil;

**repeat**

writeln();

writeln('1. Статическая память');

writeln('2. Динамическая память');

writeln('3. Выход');

write('Выберите тип памяти: ');

readln(memoryChoice);

**case** memoryChoice **of**

1:

**repeat**

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

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

writeln('3. Удалить последний элемент');

writeln('4. Удалить первый элемент');

writeln('5. Вывести ДЕК');

writeln('6. Очистить ДЕК');

writeln('7. Назад');

write('Ваш выбор: ');

readln(choice);

**case** choice **of**

1:

**begin**

write('Введите элемент для добавления: ');

**var** element: Integer;

readln(element);

Push(d, element);

**end**;

2:

**begin**

write('Введите элемент для добавления: ');

**var** element: Integer;

readln(element);

InsertFirst(d, element);

**end**;

3: Pop(d);

4: DeleteFirst(d);

5: Print(d);

6: Clear(d);

**end**;

**until** choice = 7;

2:

**repeat**

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

writeln('2. Удалить первый элемент');

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

writeln('4. Удалить последний элемент');

writeln('5. Вывести ДЕК');

writeln('6. Назад');

write('Ваш выбор: ');

readln(choice);

**case** choice **of**

1:

**begin**

write('Введите данные: ');

**var** Data: Integer;

readln(Data);

PushFrontDynamic(Data);

**end**;

2: PopFrontDynamic;

3:

**begin**

write('Введите данные: ');

**var** Data: Integer;

readln(Data);

PushBackDynamic(Data);

**end**;

4: PopBackDynamic;

5: DisplayDeque;

**end**;

**until** choice = 6;

**end**;

**until** memoryChoice = 3;

**end**.

**6. Результат выполнения программы**

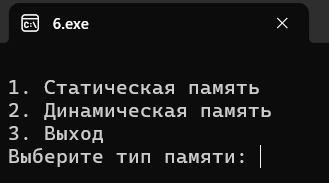


Рисунок 14 – Основное меню

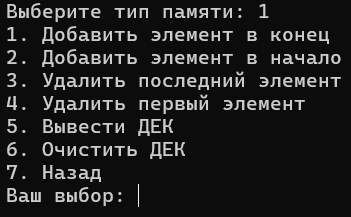


Рисунок 15 – Выбор статистической памяти

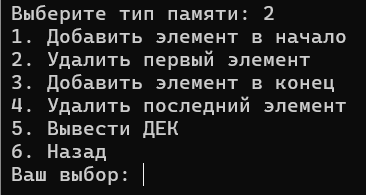


Рисунок 16 – Выбор динамической памяти

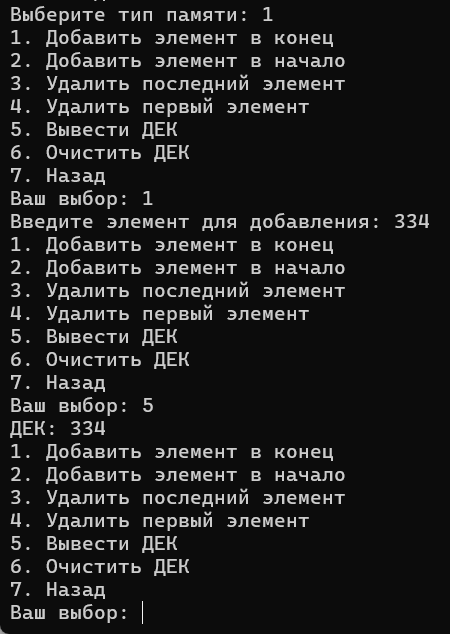


Рисунок 17 – Добавление элемента в конец (статистическая память)

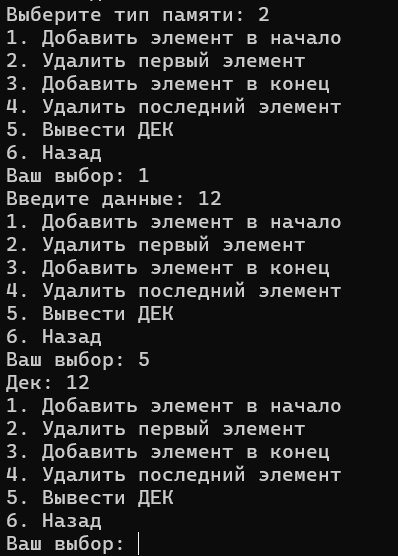


Рисунок 18 – Добавление элемента в начало (динамическая память)

**7. Вывод**

В ходе выполнения лабораторной работы были изучены основные принципы работы с базовыми структурами данных, а также получены навыки организации case-меню.

Для реализации дека использовались как статическая память, так и динамическая память. Программа была организована с помощью меню, основанного на конструкции «case», что обеспечило удобство пользовательского интерфейса и позволило проще взаимодействовать с программой.

В работе была использована такая программа, как Draw.io. Draw.io - это онлайн-редактор диаграмм и схем, который позволяет создавать различные виды диаграмм, схем, планов и других графических объектов. Он широко используется в различных отраслях, таких как бизнес, образование, наука и т.д., для создания визуализаций, презентаций и документации. С помощью Draw.io можно легко и быстро создавать профессиональные диаграммы, схемы и другие графические объекты, а также экспортировать их в различные форматы, такие как PNG, JPEG, SVG.