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 ДИСЦИПЛИНЕ**

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

Выполнил: студент учебной группы

ИСПк-203-52-00

Четвериков Д. А.

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**Цель работы:**

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

**Формулировка задания:**

1. Написать программу для работы со структурой данных "Дек".

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.

**Описание алгоритма:**

Программа реализует дек (двустороннюю очередь) с использованием кольцевого массива. Дек — это структура данных, которая позволяет добавлять и удалять элементы как с начала, так и с конца. Программа предоставляет пользователю интерактивное меню для управления деком.

Основные шаги алгоритма

1. Инициализация дека:

- Дек представлен в виде записи (record), содержащей:

- Массив data для хранения элементов.

- Индексы front и rear для обозначения начала и конца дека.

- При инициализации (InitializeDeque) индексы front и rear устанавливаются в -1, что означает, что дек пуст.

2. Проверка состояния дека:

- IsEmpty: Проверяет, пуст ли дек. Если front = -1, дек пуст.

- IsFull: Проверяет, заполнен ли дек. Дек полон, если:

- front находится в начале массива (0), а rear — в конце (MAX\_SIZE - 1).

- Или front находится сразу за rear (кольцевая структура).

3. Добавление элементов:

- InsertFront:

- Если дек полон, выводится сообщение.

- Если дек пуст, front и rear устанавливаются в 0.

- Иначе front перемещается назад (с учетом кольцевой структуры).

- Новый элемент добавляется в начало дека.

- InsertRear:

- Аналогично InsertFront, но элемент добавляется в конец дека.

4. Удаление элементов:

- DeleteFront:

- Если дек пуст, выводится сообщение.

- Если в деке только один элемент, front и rear устанавливаются в -1.

- Иначе front перемещается вперед (с учетом кольцевой структуры).

- DeleteRear:

- Аналогично DeleteFront, но удаляется элемент с конца дека.

5. Отображение дека:

- DisplayDeque:

- Если дек пуст, выводится сообщение.

- Иначе выводятся все элементы дека, начиная с front и заканчивая rear.

6. Интерактивное меню:

- Программа отображает меню с возможными действиями:

- Добавить элемент в начало.

- Добавить элемент в конец.

- Удалить элемент с начала.

- Удалить элемент с конца.

- Отобразить элементы дека.

- Выйти из программы.

- Пользователь выбирает действие, нажимая соответствующую клавишу (1 – 6).

- После выполнения действия программа ждет нажатия любой клавиши для продолжения.

**Схемы алгоритмов:**

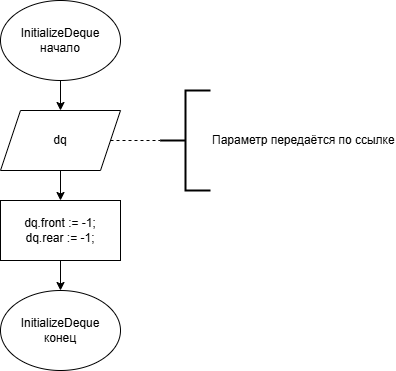


Рисунок 1 – схема алгоритма процедуры InitializeDeque

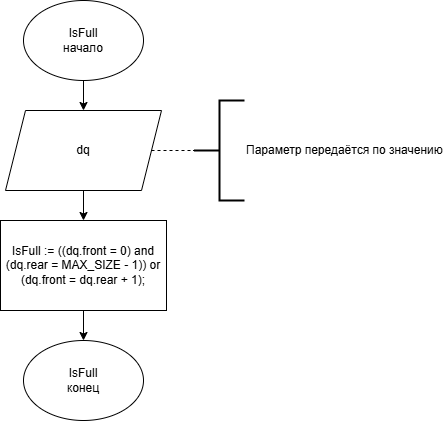


Рисунок 2 – схема алгоритма функции IsFull

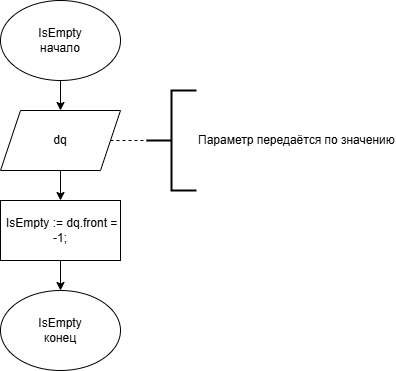


Рисунок 3 – схема алгоритма функции IsEmpty

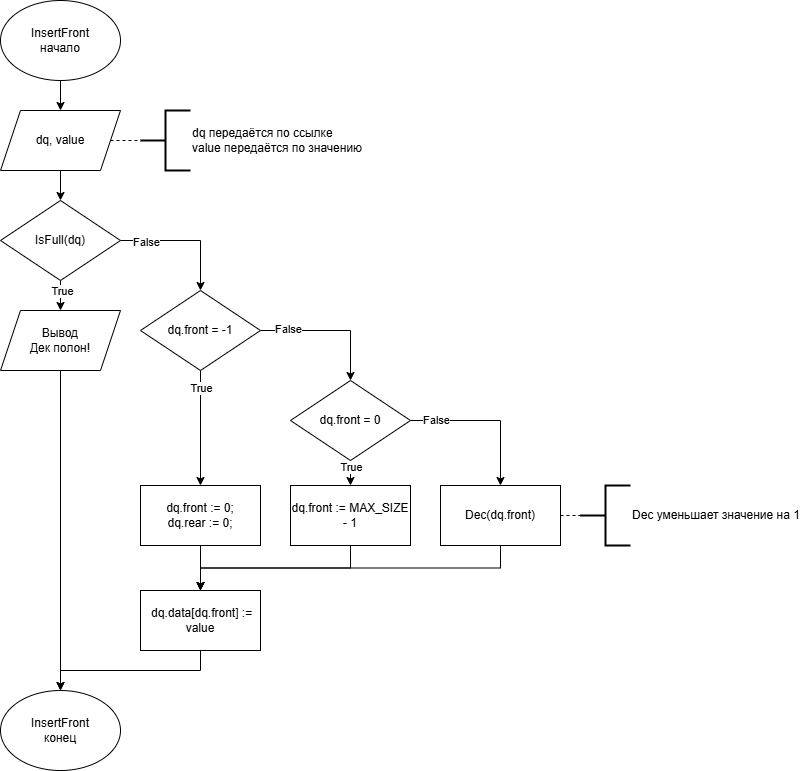


Рисунок 4 – схема алгоритма процедуры InsertFront

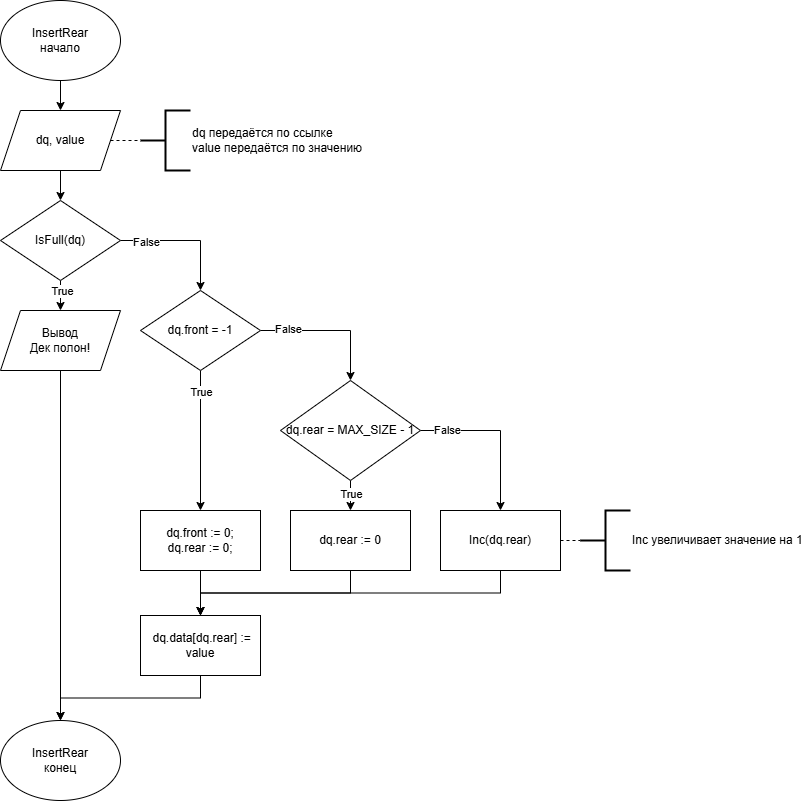


Рисунок 5 – схема алгоритма процедуры InsertRear

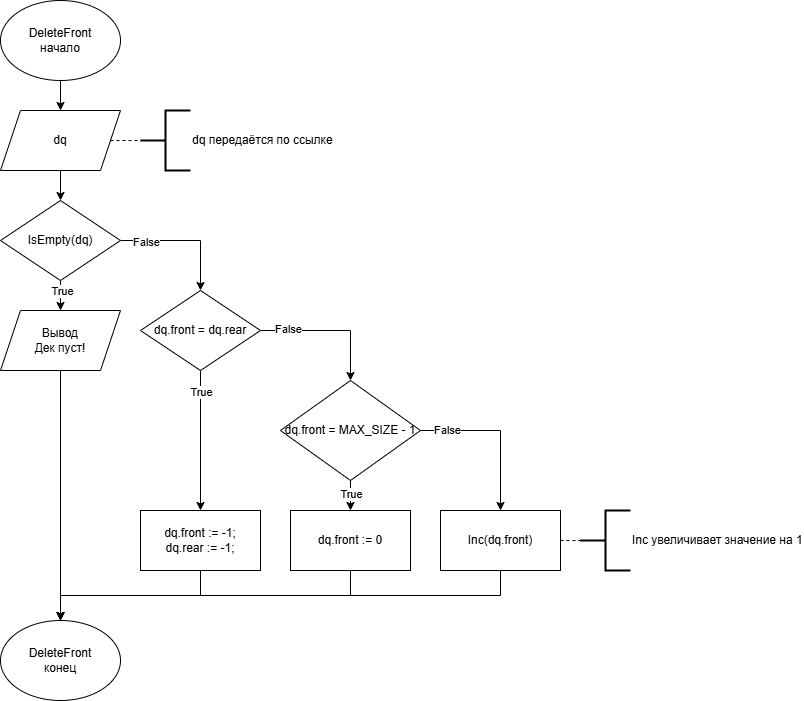


Рисунок 6 – схема алгоритма процедуры DeleteFront

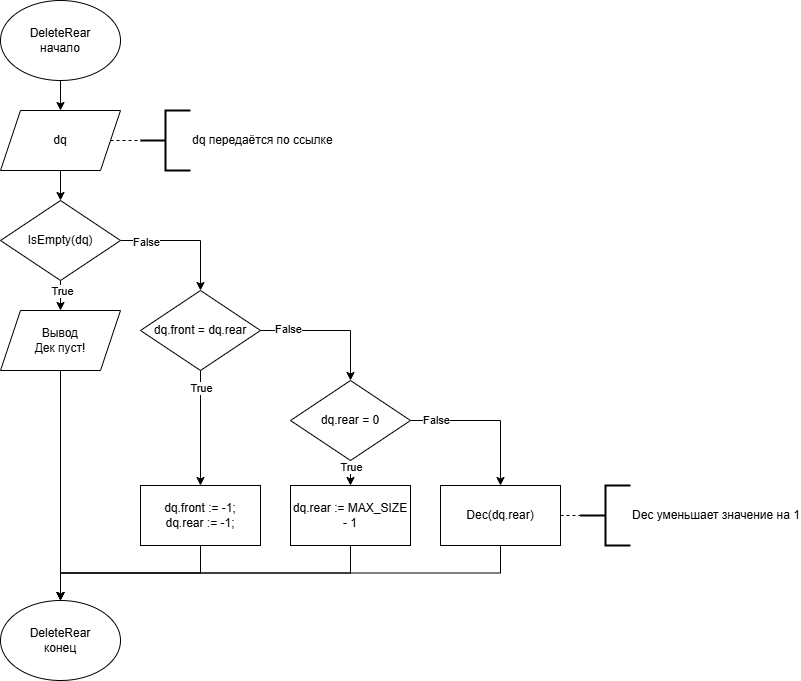


Рисунок 7 – схема алгоритма процедуры DeleteRear

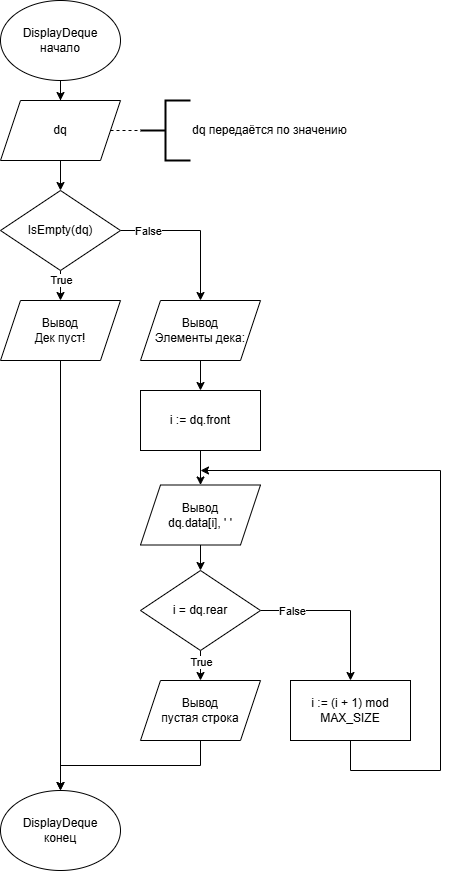


Рисунок 8 – схема алгоритма процедуры DisplayDeque

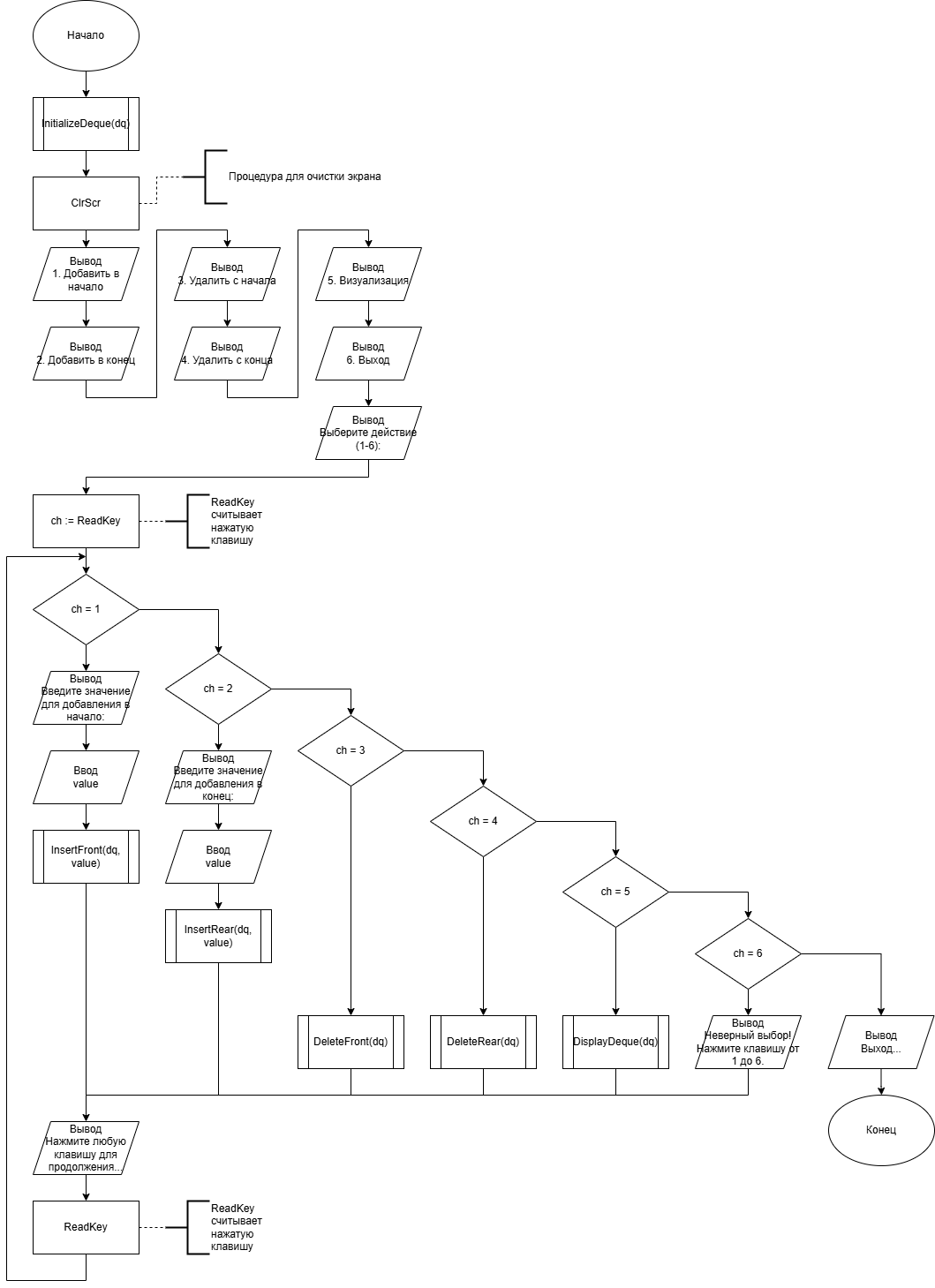


Рисунок 9 – схема алгоритма основной программы

**Код программы:**

**uses** crt;

**const**

MAX\_SIZE = 10;

**type**

Deque = **record**

data: **array**[0..MAX\_SIZE-1] **of** Integer;

front, rear: Integer;

**end**;

**procedure** InitializeDeque(**var** dq: Deque);

**begin**

dq.front := -1;

dq.rear := -1;

**end**;

**function** IsFull(dq: Deque): Boolean;

**begin**

IsFull := ((dq.front = 0) **and** (dq.rear = MAX\_SIZE - 1)) **or** (dq.front = dq.rear + 1);

**end**;

**function** IsEmpty(dq: Deque): Boolean;

**begin**

IsEmpty := dq.front = -1;

**end**;

**procedure** InsertFront(**var** dq: Deque; value: Integer);

**begin**

**if** IsFull(dq) **then**

**begin**

WriteLn('Дек полон!');

**Exit**;

**end**;

**if** dq.front = -1 **then**

**begin**

dq.front := 0;

dq.rear := 0;

**end**

**else if** dq.front = 0 **then**

dq.front := MAX\_SIZE - 1

**else**

Dec(dq.front);

dq.data[dq.front] := value;

**end**;

**procedure** InsertRear(**var** dq: Deque; value: Integer);

**begin**

**if** IsFull(dq) **then**

**begin**

WriteLn('Дек полон!');

**Exit**;

**end**;

**if** dq.front = -1 **then**

**begin**

dq.front := 0;

dq.rear := 0;

**end**

**else if** dq.rear = MAX\_SIZE - 1 **then**

dq.rear := 0

**else**

Inc(dq.rear);

dq.data[dq.rear] := value;

**end**;

**procedure** DeleteFront(**var** dq: Deque);

**begin**

**if** IsEmpty(dq) **then**

**begin**

WriteLn('Дек пуст!');

**Exit**;

**end**;

**if** dq.front = dq.rear **then**

**begin**

dq.front := -1;

dq.rear := -1;

**end**

**else if** dq.front = MAX\_SIZE - 1 **then**

dq.front := 0

**else**

Inc(dq.front);

**end**;

**procedure** DeleteRear(**var** dq: Deque);

**begin**

**if** IsEmpty(dq) **then**

**begin**

WriteLn('Дек пуст!');

**Exit**;

**end**;

**if** dq.front = dq.rear **then**

**begin**

dq.front := -1;

dq.rear := -1;

**end**

**else if** dq.rear = 0 **then**

dq.rear := MAX\_SIZE - 1

**else**

Dec(dq.rear);

**end**;

**procedure** DisplayDeque(dq: Deque);

**var**

i: Integer;

**begin**

**if** IsEmpty(dq) **then**

**begin**

WriteLn('Дек пуст!');

**Exit**;

**end**;

Write('Элементы дека: ');

i := dq.front;

**while** True **do**

**begin**

Write(dq.data[i], ' ');

**if** i = dq.rear **then Break**;

i := (i + 1) **mod** MAX\_SIZE;

**end**;

WriteLn;

**end**;

**var**

dq: Deque;

choice, value: Integer;

ch: Char;

**begin**

InitializeDeque(dq);

**repeat**

ClrScr; // Очистка экрана

WriteLn('1. Добавить в начало');

WriteLn('2. Добавить в конец');

WriteLn('3. Удалить с начала');

WriteLn('4. Удалить с конца');

WriteLn('5. Визуализация');

WriteLn('6. Выход');

Write('Выберите действие (1-6): ');

ch := ReadKey; // Считываем нажатую клавишу

**case** ch **of**

'1': **begin**

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

ReadLn(value);

InsertFront(dq, value);

**end**;

'2': **begin**

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

ReadLn(value);

InsertRear(dq, value);

**end**;

'3': DeleteFront(dq);

'4': DeleteRear(dq);

'5': DisplayDeque(dq);

'6': **begin**

WriteLn('Выход...');

**Break**;

**end**;

**else** WriteLn('Неверный выбор! Нажмите клавишу от 1 до 6.');

**end**;

WriteLn('Нажмите любую клавишу для продолжения...');

ReadKey; // Ожидание нажатия любой клавиши

**until** False;

**end**.

**Результат выполнения программы:**

Рисунок 10 – результат выполнения одной из кнопок

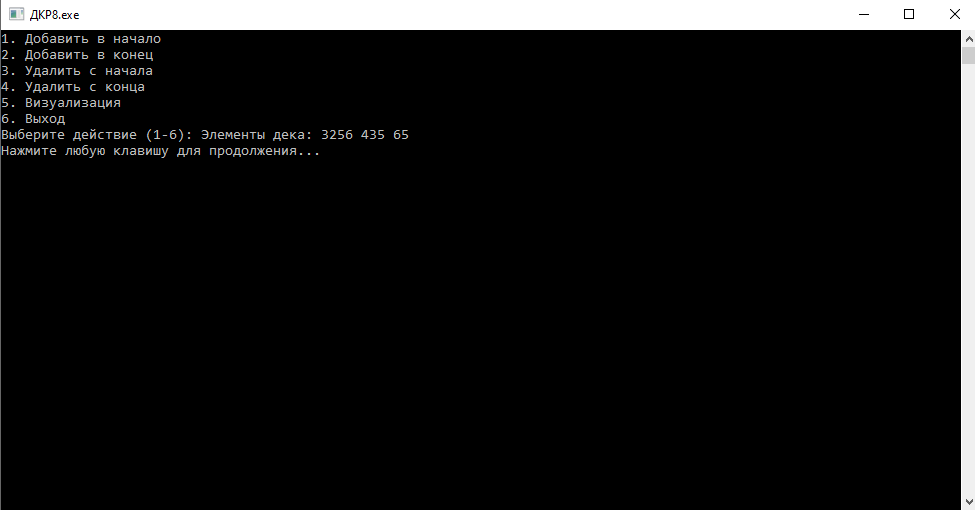


Рисунок 11 – результат выполнения одной из кнопок

**Вывод:**

В ходе выполнения данной работы принципs работы с базовыми структурами данных, получение навыков организации case-меню. Была написана программа для работы со структурой данных "Дек", реализованная на основе статической памяти, с возможностью посмотреть наглядную визуализацию содержимого структуры.