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 ЭЛЕМЕНТАРЫНХ СТРУКТУР ДАННЫХ НА ОСНОВЕ СТАТИЧЕСКОЙ ПАМЯТИ»**

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

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

Кривошеин Дмитрий Васильевич

Преподаватель:

Сергеева Елизавета Григорьевна

г. Киров

2024

1. **Цель работы**: изучение принципов работы с базовыми структурами данных, полученных навыков организации case-меню.
2. **Задание:**
3. Написать программу для работы со структурой данных “Дек”
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:**

Дек (или двусторонняя очередь) - это структура данных, которая позволяет добавлять и удалять элементы как в начале, так и в конце коллекции. Принцип работы дека основан на том, что операции добавления и удаления элементов могут происходить как с одного, так и с другого конца очереди.

Основные операции с деком:

1. Добавление элемента в начало (pushFront) и конец (pushBack) дека.

2. Удаление элемента из начала (popFront) и конца (popBack) дека.

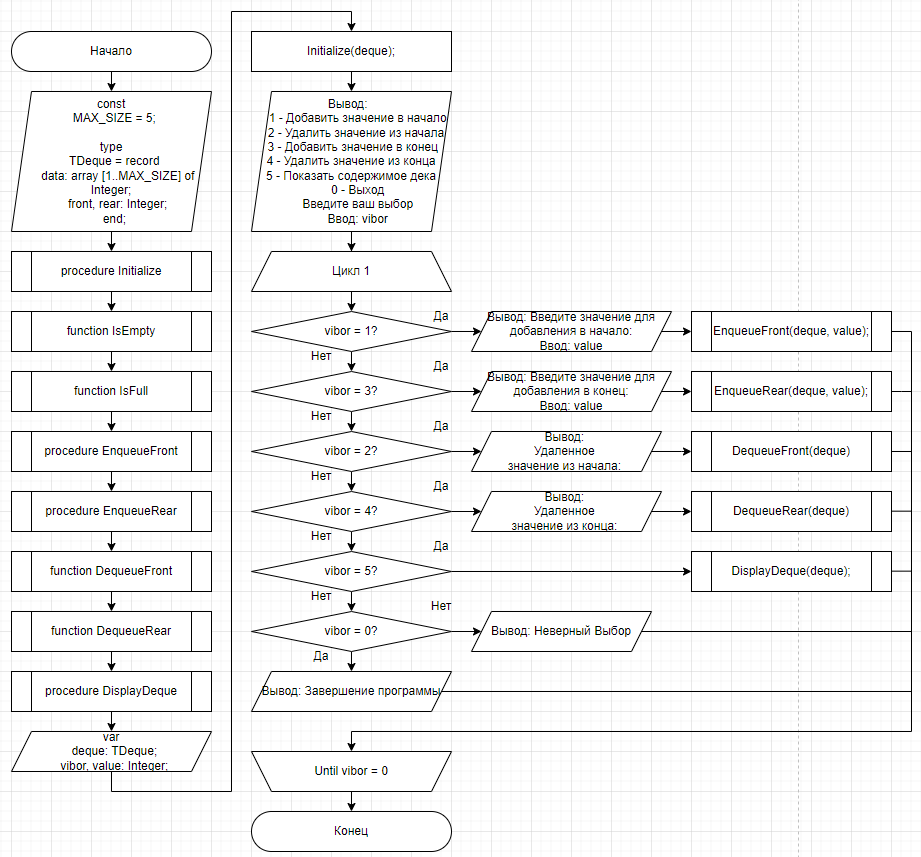
3. Получение элемента из начала (front) и конца (back) дека без его удаления.

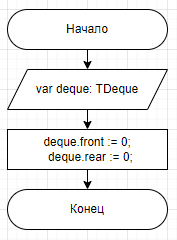
4. Проверка на пустоту дека.

Принцип работы дека можно представить как двустороннюю очередь или список, к которому можно добавлять и удалять элементы с обоих концов. Это позволяет эффективно реализовывать задачи, где требуется доступ к элементам как в начале, так и в конце коллекции.

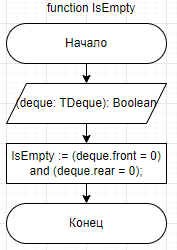
Дек может быть реализован как с использованием массива, так и с использованием связанного списка, в зависимости от специфики задачи и требований к производительности операций. Реализация с использованием массива позволяет быстро выполнить операции доступа к элементам по индексу, в то время как реализация на основе связанного списка обеспечивает динамическое изменение размеров коллекции.

1. **Схема алгоритма:**

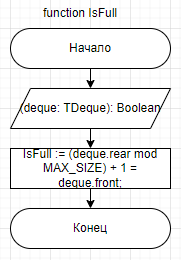
**  
Рисунок 1 – алгоритм решения**



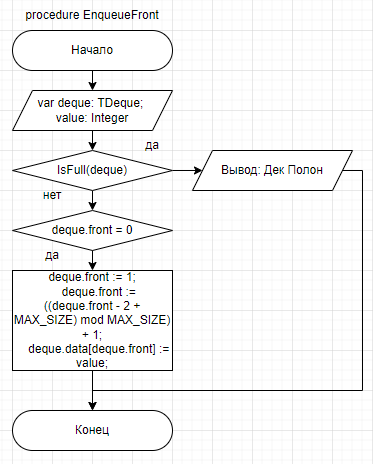
**Рисунок 2 – Процедура Initialize**

****

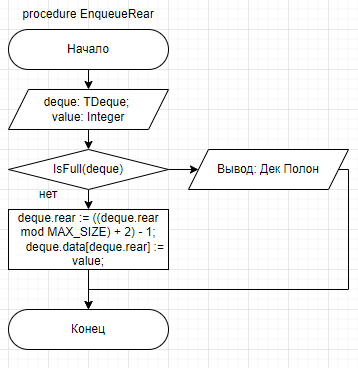
**Рисунок 3 – функция IsEmpty**

****

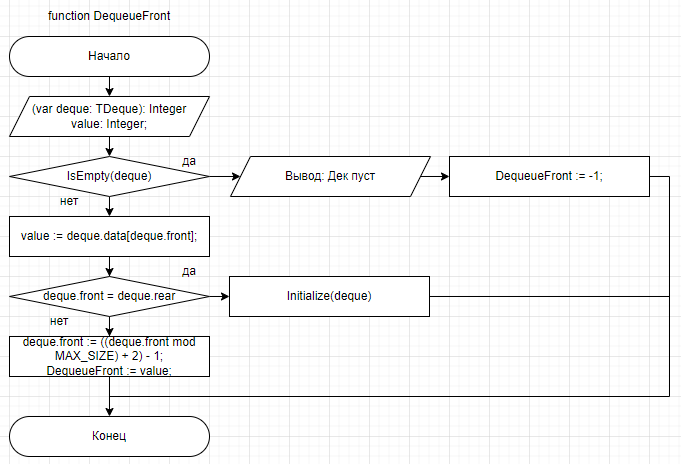
**Рисунок 4 – функция IsFull**

****

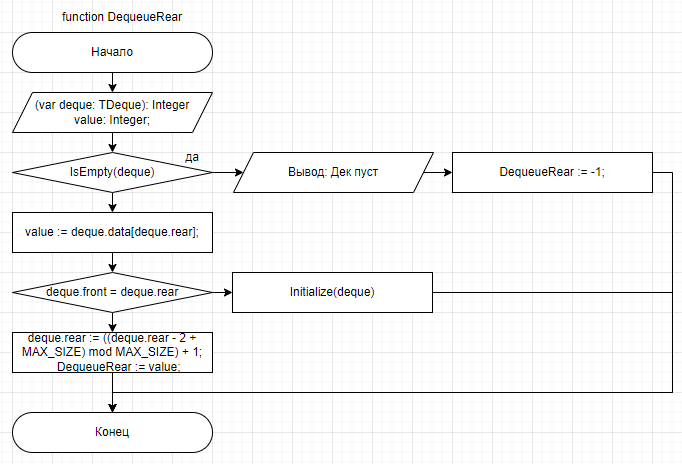
**Рисунок 5 – Процедура EnqueueFront**

****

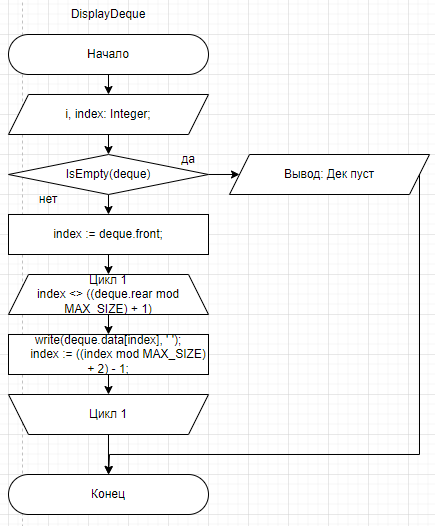
**Рисунок 6 – Процедура EnqueueRear**

****

**Рисунок 7 – Функция DequeueFront**

****

**Рисунок 8 – Функция DequeueRear**

****

**Рисунок 9 – Процедура DisplayDeque**

**Код программ:**

**program** ProgrammaDeque;

**const**

MAX\_SIZE = 5;

**type**

TDeque = **record**

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

front, rear: Integer;

**end**;

// процедура инициализации дека. Устанавливает front и rear в 0.

**procedure** Initialize(**var** deque: TDeque);

**begin**

deque.front := 0;

deque.rear := 0;

**end**;

//функция, проверяющая, пуст ли дек (если front и rear = 0, то дек пуст).

**function** IsEmpty(deque: TDeque): Boolean;

**begin**

IsEmpty := (deque.front = 0) **and** (deque.rear = 0);

**end**;

//функция, проверяющая, полон ли дек (если следующий индекс после rear равен front, то дек полон).

**function** IsFull(deque: TDeque): Boolean;

**begin**

IsFull := (deque.rear **mod** MAX\_SIZE) + 1 = deque.front;

**end**;

//процедура добавления элемента в начало дека.

**procedure** EnqueueFront(**var** deque: TDeque; value: Integer);

**begin**

**if** IsFull(deque) **then**

writeln('Дек полон')

**else begin**

**if** deque.front = 0 **then**

deque.front := 1;

deque.front := ((deque.front - 2 + MAX\_SIZE) **mod** MAX\_SIZE) + 1;

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

**end**;

**end**;

// процедура добавления элемента в конец дека.

**procedure** EnqueueRear(**var** deque: TDeque; value: Integer);

**begin**

**if** IsFull(deque) **then**

writeln('Дек полон')

**else begin**

deque.rear := ((deque.rear **mod** MAX\_SIZE) + 2) - 1;

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

**end**;

**end**;

//функция удаления элемента из начала дека.

**function** DequeueFront(**var** deque: TDeque): Integer;

**var**

value: Integer;

**begin**

**if** IsEmpty(deque) **then begin**

writeln('Дек пуст');

DequeueFront := -1;

**end**

**else begin**

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

**if** deque.front = deque.rear **then**

Initialize(deque)

**else**

deque.front := ((deque.front **mod** MAX\_SIZE) + 2) - 1;

DequeueFront := value;

**end**;

**end**;

//функция удаления элемента из конца дека.

**function** DequeueRear(**var** deque: TDeque): Integer;

**var**

value: Integer;

**begin**

**if** IsEmpty(deque) **then begin**

writeln('Дек пуст');

DequeueRear := -1;

**end**

**else begin**

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

**if** deque.front = deque.rear **then**

Initialize(deque)

**else**

deque.rear := ((deque.rear - 2 + MAX\_SIZE) **mod** MAX\_SIZE) + 1;

DequeueRear := value;

**end**;

**end**;

//процедура отображения содержимого дека.

**procedure** DisplayDeque(deque: TDeque);

**var**

i, index: Integer;

**begin**

**if** IsEmpty(deque) **then**

writeln('Дек пуст')

**else begin**

index := deque.front;

**while** index <> ((deque.rear **mod** MAX\_SIZE) + 1) **do begin**

write(deque.data[index], ' ');

index := ((index **mod** MAX\_SIZE) + 2) - 1;

**end**;

writeln;

**end**;

**end**;

**var**

deque: TDeque;

vibor, value: Integer;

**begin**

Initialize(deque);

**repeat**

writeln('1 - Добавить значение в начало');

writeln('2 - Удалить значение из начала');

writeln('3 - Добавить значение в конец');

writeln('4 - Удалить значение из конца');

writeln('5 - Показать содержимое дека');

writeln('0 - Выход');

write('Введите ваш выбор: ');

readln(vibor);

**case** vibor **of**

1: **begin**

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

readln(value);

EnqueueFront(deque, value);

**end**;

3: **begin**

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

readln(value);

EnqueueRear(deque, value);

**end**;

2: writeln('Удаленное значение из начала: ', DequeueFront(deque));

4: writeln('Удаленное значение из конца: ', DequeueRear(deque));

5: DisplayDeque(deque);

0: writeln('Завершение программы');

**else** writeln('Неверный выбор');

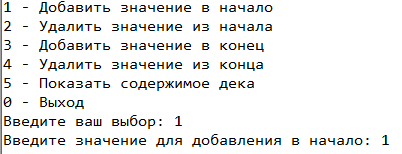
**end**;

writeln;

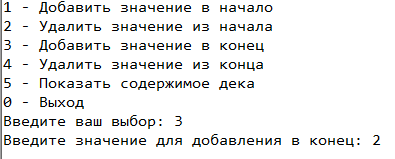
**until** vibor = 0;

**end**.

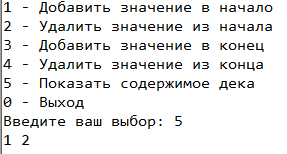
**Результат выполнения программ:**



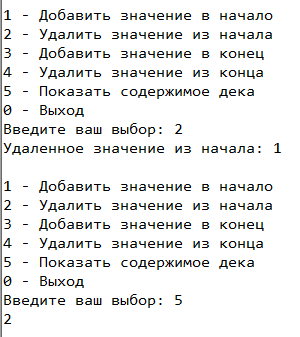
**Рисунок 10 – Добавление значения в начало Дека**



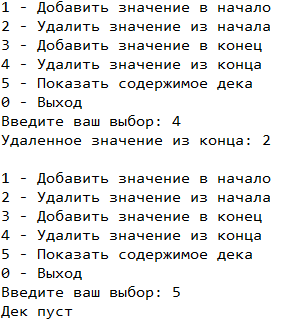
**Рисунок 11 – Добавление значения в конец Дека**



**Рисунок 12 – Вывод содержимого Дека**



**Рисунок 13 – Удаление значения из начала Дека**



**Рисунок 14 – Удаление значения из конца Дека**

1. **Вывод:**

Программа была успешно разработана и выполнена согласно поставленной задаче. Вот основные выводы:

Мы изучили принципы работы с базовыми структурами данных, а именно с “Кольцевым двусвязным списком”, освежили свои знания в работе с case-меню, а также в работе с указателями и списками, в будущем это может помочь нам при оптимизации и улучшении код. Попрактиковались в реализации структуры данных на основе статической памяти