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»**

Выполнил: студент учебной группы

ИСПк-204-52-00

Исупов Максим Олегович

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**Цель работы**

Цель работы: изучение принципов работы с базовыми структурами данных, получение навыков организации case-menu

**Формулировка задания**

Вариант: 8

Лабораторная работа №6

Реализация элементарных структур данных на основе статической памяти.

Цель работы: изучение принципов работы с базовыми структурами данных, получение навыков организации case-menu

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 первой программы**

Необходимо построить фрактал «Снежинка Коха» с реализацией возможности масштабирования, изменения глубины прорисовки

1. создаём функцию **Initialize** для инициализации дека. Устанавливает front и rear в 0;
2. создаём функцию IsEmpty. Это функция, проверяющая, пуст ли дек;
3. создаём функцию IsFull, проверяющую, полон ли дек;
4. создаём функции EnqueueFront и EnqueueRear для добавления элемента в начало и в конец дека;
5. создаём функции DequeueFront и DequeueRear для удаления элемента с начала и с конца дека;
6. Далее создаём процедуру DisplayDeque для отображения содержимого дека
7. Следующим шагом осуществляем взаимодействие с функциями с помощью case-menu.

**Схема алгоритма первой программы**

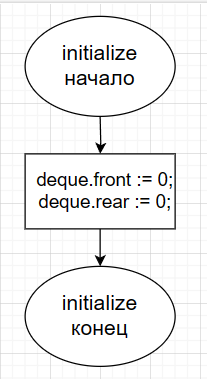


Рисунок 1 – Схема алгоритм функции initialize

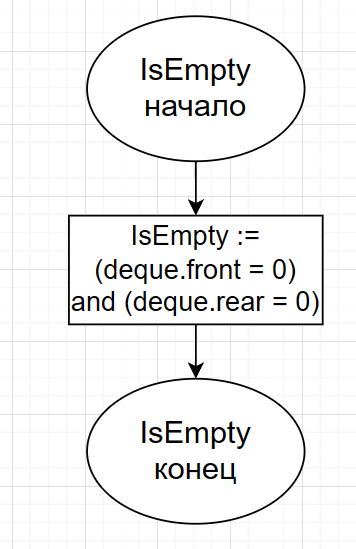
****

Рисунок 2 – схема алгоритма функции IsEmpty

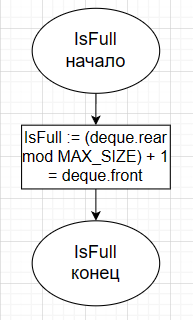


Рисунок 3 – схема алгоритма функции IsFull

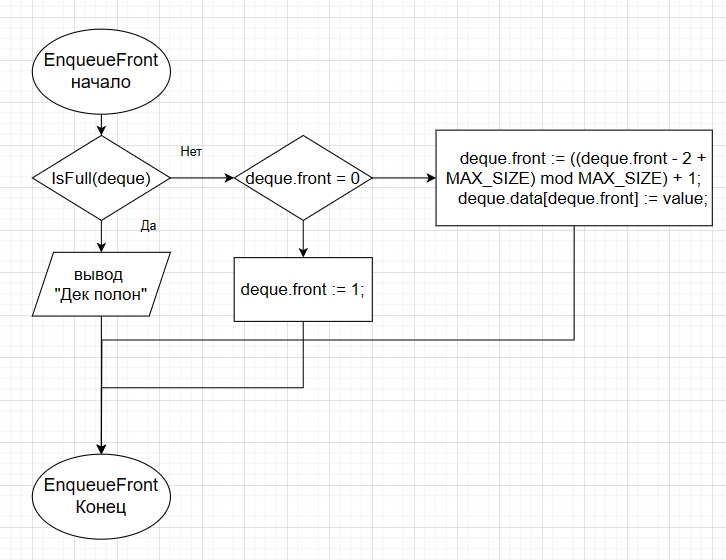


Рисунок 4 – схема алгоритм функции EnqueueFront

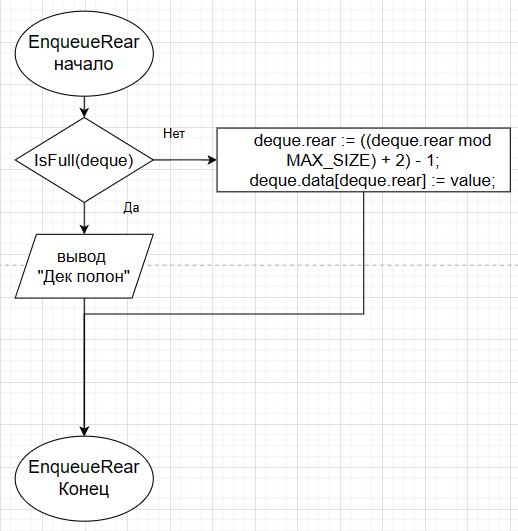
****

Рисунок 5 – схема алгоритма функции EnqueueRear

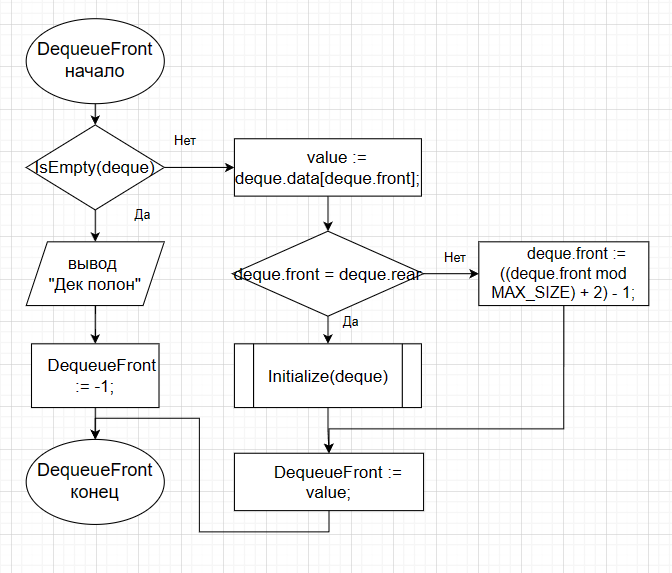


Рисунок 6 – схема алгоритма функции DequeueFront

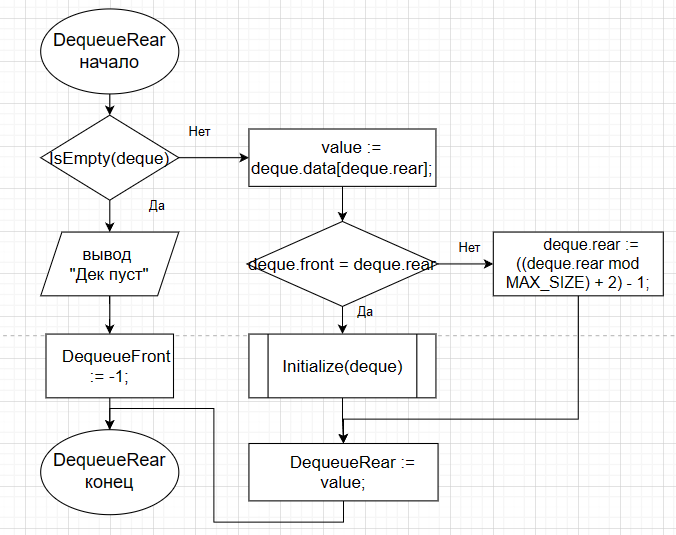


Рисунок 7 – схема алгоритма функции DequeueRear

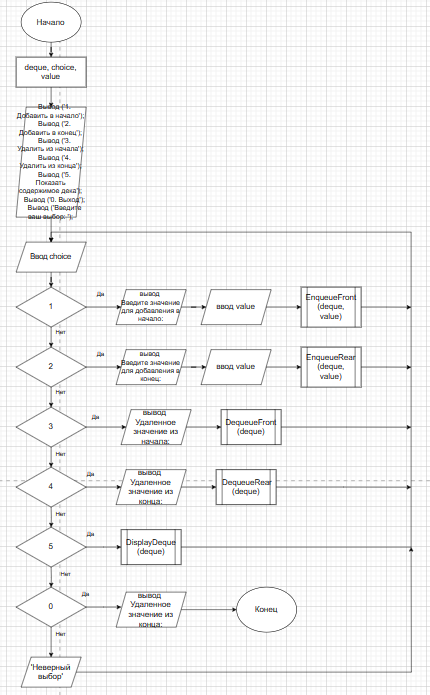


Рисунок 8 – схема алгоритм программы

**Код программы**

**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;

choice, value: Integer;

**begin**

Initialize(deque);

**repeat**

writeln('1. Добавить в начало');

writeln('2. Добавить в конец');

writeln('3. Удалить из начала');

writeln('4. Удалить из конца');

writeln('5. Показать содержимое дека');

writeln('0. Выход');

write('Введите ваш выбор: ');

readln(choice);

**case** choice **of**

1: **begin**

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

readln(value);

EnqueueFront(deque, value);

**end**;

2: **begin**

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

readln(value);

EnqueueRear(deque, value);

**end**;

3: 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** choice = 0;

**end**.

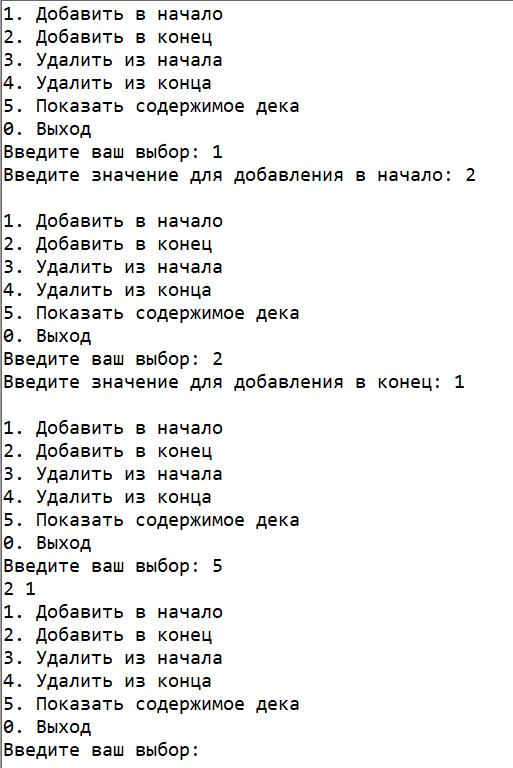


Рисунок 9 – Результат выполнения программы 1

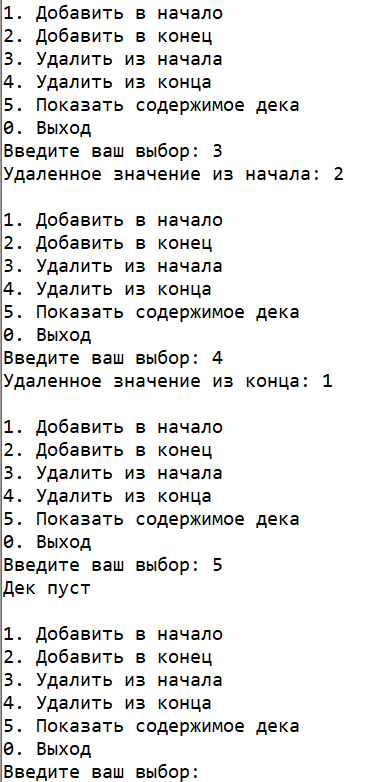


Рисунок 10 – Результат выполнения программы 2

**Вывод**

В ходе выполнения домашней лабораторной работы мы закрепили знания полученные при выполнение лабораторной работы №15-16. Научились реализовывать рекурсивные алгоритмы, изучили фракталы, их свойства, освоили использование кнопок для изменения параметров фрактала. Таким образом, полученные знания могут быть полезны в дальнейшем.

Так же для написания схемы алгоритма был использован интернет -ресурс «draw.io». Его использование упрощает написание схем алгоритмов, уменьшает вероятность ошибок, а также в случае пропуска какого-либо цикла его можно без каких-либо проблем вставить в любое место схемы. Опыт написания схем-алгоритмов будет полезен в будущем для решения новых задач а так же понимание программ, написанных другими пользователями