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»**

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

Выполнил: студент учебной группы

ИСПк-206-52-00

Петухов Егор Игоревич

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

Цель работы:

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

Формулировка задания (с вариантом)

Вариант: 3

Цель работы: изучение принципов работы с базовыми структурами данных получение навыков организации саsе-меню.

Задание:

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

2.Структура данных должна быть реализована на основе статической памяти

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

**Описание алгоритма:**

1. Определяется константа MAX\_SIZE, которая задает максимальный размер дека.

2. Объявляется тип TDeque, который представляет собой запись с тремя полями: Data (массив для хранения элементов дека), Front (индекс первого элемента) и Rear (индекс последнего элемента).

3. Процедура Initialize инициализирует дек, устанавливая Front и Rear в 0, что означает пустой дек.

4. Функция IsEmpty проверяет, пуст ли дек, сравнивая Front и Rear с 0.

5. Функция IsFull проверяет, заполнен ли дек, сравнивая Rear с MAX\_SIZE.

6. Процедура EnqueueFront добавляет элемент в начало дека, если дек не заполнен. Если дек был пуст, то Front и Rear устанавливаются на 1, и элемент помещается в Data[1]. В противном случае Front уменьшается на 1, и элемент помещается в Data[Front].

7. Процедура EnqueueRear добавляет элемент в конец дека, если дек не заполнен. Если дек был пуст, то Front и Rear устанавливаются на 1, и элемент помещается в Data[1]. В противном случае Rear увеличивается на 1, и элемент помещается в Data[Rear].

8. Функция DequeueFront удаляет и возвращает элемент из начала дека. Если дек пуст, функция возвращает -1. Если в деке остается один элемент, он удаляется, и Front и Rear обнуляются.

9. Функция DequeueRear удаляет и возвращает элемент из конца дека. Если дек пуст, функция возвращает -1. Если в деке остается один элемент, он удаляется, и Front и Rear обнуляются.

10. Процедура PrintDeque выводит содержимое дека на экран.

11. В основной части программы создается дек MyDeque, который инициализируется с помощью Initialize.

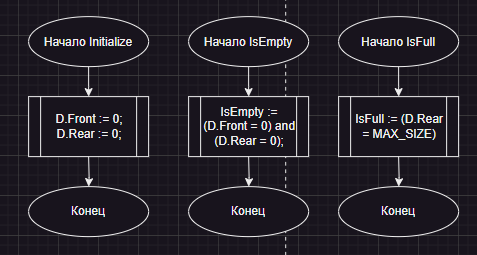
12. Программа входит в цикл, в котором выводится меню с возможными операциями над деком: 1) добавить элемент в начало, 2) добавить элемент в конец, 3) удалить элемент из начала, 4) удалить элемент из конца, 5) вывести содержимое дека, 0) выход.

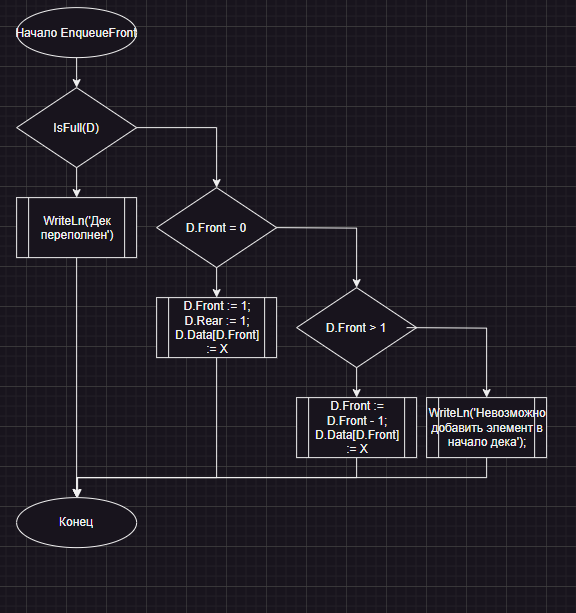
13. В зависимости от выбора пользователя вызывается соответствующая процедура или функция для выполнения операции над деком.

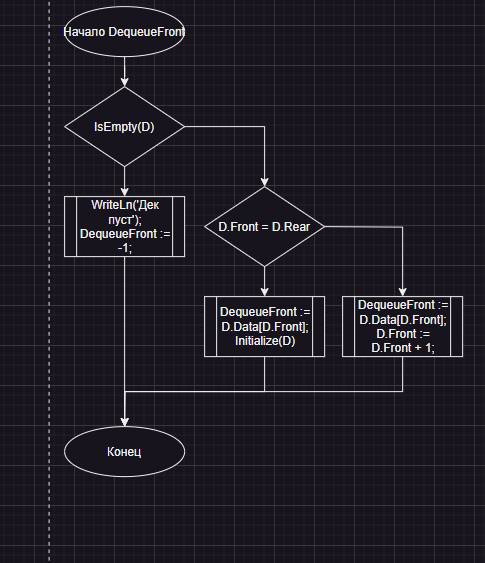
14. Цикл продолжается до тех пор, пока пользователь не выберет вариант "0" для выхода из программы.

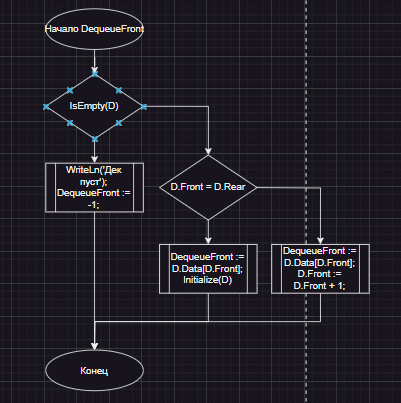
**Схема алгоритма:**

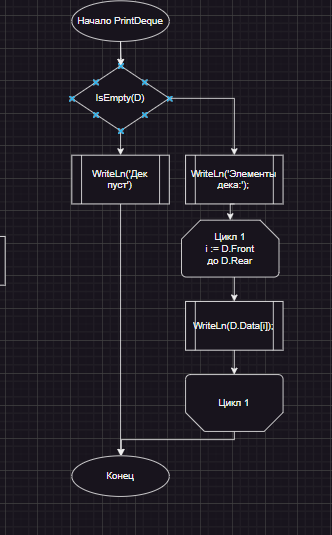


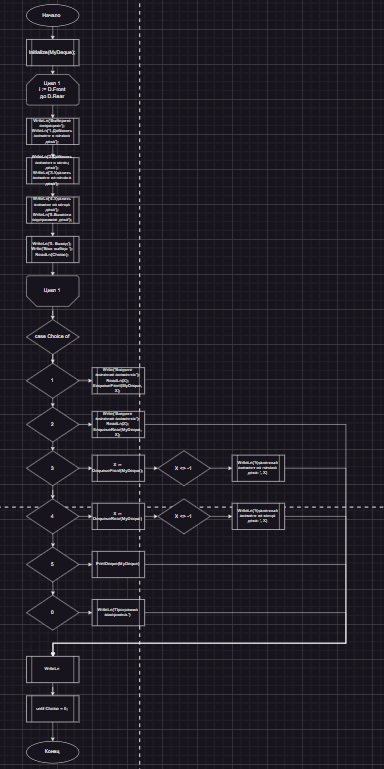






****

****

****

**Код программы:**

**const**

MAX\_SIZE = 100;

**type**

TDeque = **record**

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

Front, Rear: Integer;

**end**;

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

**begin**

D.Front := 0;

D.Rear := 0;

**end**;

**function** IsEmpty(**const** D: TDeque): Boolean;

**begin**

IsEmpty := (D.Front = 0) **and** (D.Rear = 0);

**end**;

**function** IsFull(**const** D: TDeque): Boolean;

**begin**

IsFull := (D.Rear = MAX\_SIZE);

**end**;

**procedure** EnqueueFront(**var** D: TDeque; X: Integer);

**begin**

**if** IsFull(D) **then**

WriteLn('Дек переполнен')

**else if** D.Front = 0 **then**

**begin**

D.Front := 1;

D.Rear := 1;

D.Data[D.Front] := X;

**end**

**else if** D.Front > 1 **then**

**begin**

D.Front := D.Front - 1;

D.Data[D.Front] := X;

**end**

**else**

WriteLn('Невозможно добавить элемент в начало дека');

**end**;

**procedure** EnqueueRear(**var** D: TDeque; X: Integer);

**begin**

**if** IsFull(D) **then**

WriteLn('Дек переполнен')

**else if** D.Rear = 0 **then**

**begin**

D.Front := 1;

D.Rear := 1;

D.Data[D.Rear] := X;

**end**

**else if** D.Rear < MAX\_SIZE **then**

**begin**

D.Rear := D.Rear + 1;

D.Data[D.Rear] := X;

**end**

**else**

WriteLn('Невозможно добавить элемент в конец дека');

**end**;

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

**begin**

**if** IsEmpty(D) **then**

**begin**

WriteLn('Дек пуст');

DequeueFront := -1;

**end**

**else if** D.Front = D.Rear **then**

**begin**

DequeueFront := D.Data[D.Front];

Initialize(D);

**end**

**else**

**begin**

DequeueFront := D.Data[D.Front];

D.Front := D.Front + 1;

**end**;

**end**;

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

**begin**

**if** IsEmpty(D) **then**

**begin**

WriteLn('Дек пуст');

DequeueRear := -1; // значение если дек пуст

**end**

**else if** D.Front = D.Rear **then**

**begin**

DequeueRear := D.Data[D.Rear];

Initialize(D);

**end**

**else**

**begin**

DequeueRear := D.Data[D.Rear];

D.Rear := D.Rear - 1;

**end**;

**end**;

**procedure** PrintDeque(**const** D: TDeque);

**var**

i: Integer;

**begin**

**if** IsEmpty(D) **then**

WriteLn('Дек пуст')

**else**

**begin**

WriteLn('Элементы дека:');

**for** i := D.Front **to** D.Rear **do**

WriteLn(D.Data[i]);

**end**;

**end**;

**var**

MyDeque: TDeque;

Choice, X,i: Integer;

**begin**

Initialize(MyDeque);

**repeat**

WriteLn('Выберите операцию:');

WriteLn('1. Добавить элемент в начало дека');

WriteLn('2. Добавить элемент в конец дека');

WriteLn('3. Удалить элемент из начала дека');

WriteLn('4. Удалить элемент из конца дека');

WriteLn('5. Вывести содержимое дека');

WriteLn('0. Выход');

Write('Ваш выбор: ');

ReadLn(Choice);

**case** Choice **of**

1: **begin**

Write('Введите значение элемента: ');

ReadLn(X);

EnqueueFront(MyDeque, X);

**end**;

2: **begin**

Write('Введите значение элемента: ');

ReadLn(X);

EnqueueRear(MyDeque, X);

**end**;

3: **begin**

X := DequeueFront(MyDeque);

**if** X <> -1 **then**

WriteLn('Удаленный элемент из начала дека: ', X);

**end**;

4: **begin**

X := DequeueRear(MyDeque);

**if** X <> -1 **then**

WriteLn('Удаленный элемент из конца дека: ', X);

**end**;

5: **begin**

PrintDeque(MyDeque);

**end**;

0: **begin**

WriteLn('Программа завершена.');

**end**;

**else**

WriteLn('Неверный выбор.');

**end**;

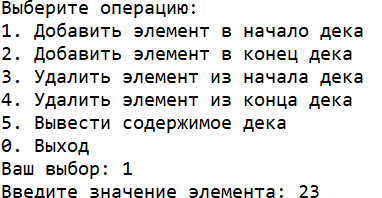
WriteLn;

**until** Choice = 0;

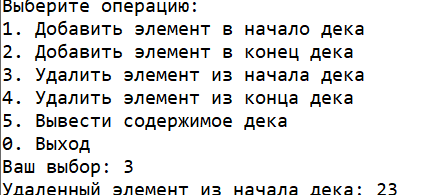
**end**.

**Результат выполнения программы:**

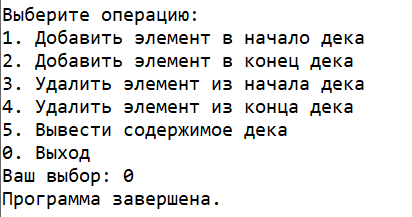
Меню



Кнопка “Добавить”



Кнопка “Удалить”

****

Кнопка “Выход”

**Вывод**

Изучил принцип работы с базовыми структурами данных, получил навыки организации case-меню. Написал программу для работы со структурой данных "Дэк". Структура данных реализована на основе статической памяти. Работа со структурой осуществляется с помощью case-меню. Предусмотрел наглядную визуализацию содержимого структуры.

Ответы на вопросы:

1. структура данных стек (StackT), реализованная с помощью массива. Она содержит два поля: Data, массив целых чисел размером MaxSize, для хранения элементов стека, и TopIdx, индекс вершины стека (вершина стека указывает на последний добавленный элемент).

2. Структура стека хранится с помощью одномерного массива Data размером MaxSize, где каждый элемент массива соответствует элементу стека. Вершина стека (TopIdx) указывает на индекс последнего добавленного элемента в массиве. Начальное значение TopIdx равно 0, что означает пустой стек.

3. Основные операции со стеком:

- Init: инициализация стека (установка TopIdx в 0).

- Push: добавление элемента в стек. Если стек не полон, увеличиваем TopIdx на 1 и записываем элемент в массив Data по новому значению TopIdx.

- Pop: удаление и возврат верхнего элемента стека. Если стек не пуст, уменьшаем TopIdx на 1 и возвращаем элемент из массива Data по новому значению TopIdx. Если стек пуст, возвращаем -1.

- Print: вывод содержимого стека на экран, начиная с вершины и до дна.

4. Кейс-меню реализован с помощью цикла repeat..until, где пользователю предлагается выбрать операцию (добавить, удалить, печать, выход). С помощью оператора case выполняется выбранная операция. Выход из меню происходит при выборе пункта "4".