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 (и динамической) памяти»**

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

Выполнил: студент учебной группы

ИСПк-202-52-00

Сосновский Никита Владимирович

Преподаватель:

Сергеева Елизавета Григорьевна

г. Киров

2024

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

Вариант 22

**Задание:**

1. Написать программу для работы со структурой данных “Очередь”.
2. Структура данных должна быть реализована на основе статической памяти.
3. Работа со структурой должна осуществляться с помощью case меню. Предусмотреть визуализацию содержимого структуры.
4. Структура данных должна быть реализована на основе динамической памяти.
5. Выбор структуры осуществляется с помощью case меню.

**Описание алгоритма:**

1. Определяется тип очереди: динамическая или статическая.
2. В зависимости от выбранного типа очереди инициализируется соответствующая структура данных.
3. В бесконечном цикле пользователь может выбирать действия:
4. Вставить элемент в очередь
5. Удалить элемент из очереди
6. Отобразить содержимое очереди
7. Выйти из программы
8. При вставке элемента в очередь, вводится значение и вызывается соответствующая функция EnqueueDynamic или EnqueueStatic.
9. При удалении элемента из очереди, вызывается функция DequeueDynamic или DequeueStatic, и удаленное значение отображается пользователю.
10. При отображении содержимого очереди вызывается функция DisplayDynamicQueue или DisplayStaticQueue.
11. При выборе выхода из программы, программа завершается.

**Схема алгоритма:**

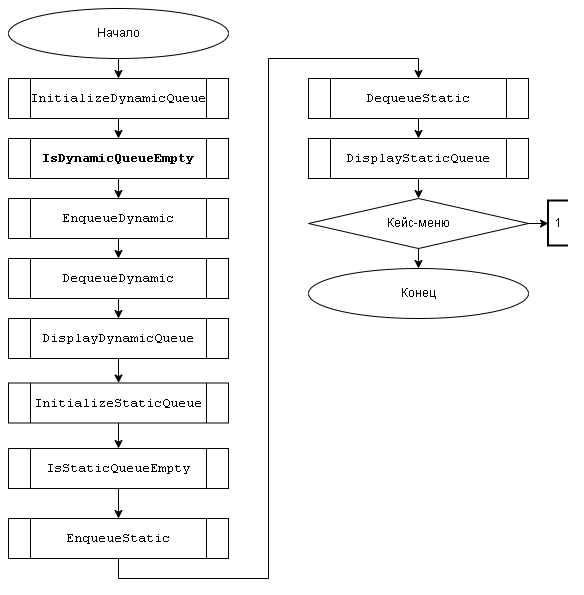


Рисунок 1 – основная программа

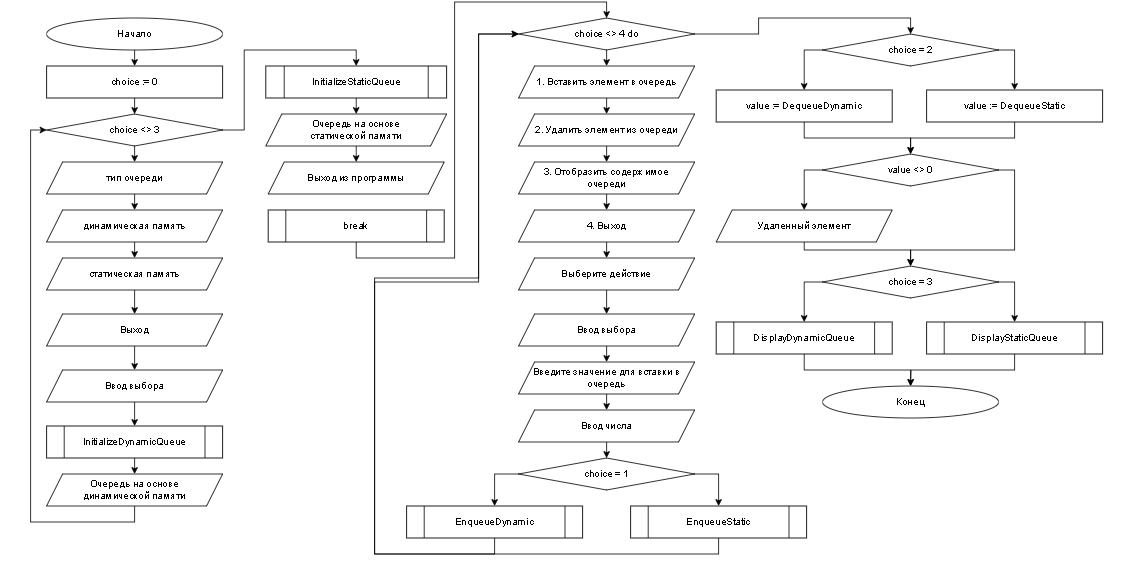


Рисунок 2 – кейс-меню

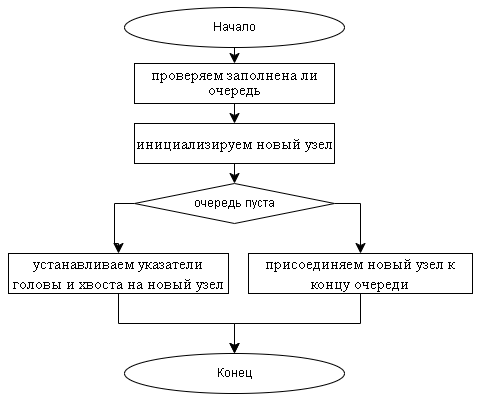


Рисунок 3 – этап добавления элемента

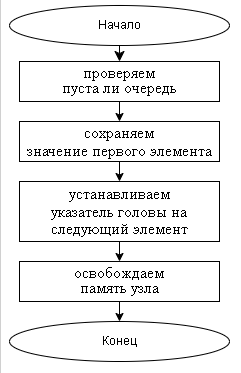


Рисунок 4 – этап удаления элемента

**Код программы:**

**program** QueueMemoryChoice;

**type**

PNode = ^TNode;

TNode = **record**

data: integer;

next: PNode;

**end**;

TQueueDynamic = **record**

head, tail: PNode;

count: integer;

**end**;

TQueueStatic = **record**

data: **array**[1..10] **of** integer;

head, tail: 1..10;

count: integer;

**end**;

**var**

choice: integer;

value: integer;

dynamicQueue: TQueueDynamic;

staticQueue: TQueueStatic;

**procedure** InitializeDynamicQueue;//Процедура инициализирует динамическую очередь, устанавливая начальные значения указателей на пустые значения

**begin**

dynamicQueue.head := nil;

dynamicQueue.tail := nil;

**end**;

**function** IsDynamicQueueEmpty: boolean;//Функция проверяет, пуста ли динамическая очередь

**begin**

IsDynamicQueueEmpty := (dynamicQueue.head = nil);

**end**;

**procedure** EnqueueDynamic(value: integer);//Процедура добавляет элемент в конец динамической очереди

**var**

newNode: PNode;

**begin**

**if** dynamicQueue.count = 5 **then**

**begin**

writeln('Очередь заполнена');

**exit**;

**end**;

**new**(newNode);

newNode^.data := value;

newNode^.next := nil;

**if** IsDynamicQueueEmpty **then**

**begin**

dynamicQueue.head := newNode;

dynamicQueue.tail := newNode;

**end**

**else**

**begin**

dynamicQueue.tail^.next := newNode;

dynamicQueue.tail := newNode;

**end**;

dynamicQueue.count := dynamicQueue.count + 1;

**end**;

**function** DequeueDynamic: integer;//Функция удаляет и возвращает первый элемент из динамической очереди

**var**

tempNode: PNode;

dequeuedValue: integer;

**begin**

**if** IsDynamicQueueEmpty **then**

**begin**

writeln('Очередь пуста');

DequeueDynamic := 0;

**end**

**else**

**begin**

tempNode := dynamicQueue.head;

dequeuedValue := tempNode^.data;

dynamicQueue.head := tempNode^.next;

dispose(tempNode);

DequeueDynamic := dequeuedValue;

**end**;

**end**;

**procedure** DisplayDynamicQueue;//Процедура выводит содержимое динамической очереди

**var**

currentNode: PNode;

**begin**

currentNode := dynamicQueue.head;

**if** IsDynamicQueueEmpty **then**

writeln('Очередь пуста')

**else**

**begin**

writeln('Содержимое очереди:');

**while** currentNode <> nil **do**

**begin**

writeln(currentNode^.data);

currentNode := currentNode^.next;

**end**;

**end**;

**end**;

**procedure** InitializeStaticQueue;//Процедура инициализирует статическую очередь, устанавливая начальные значения указателей

**begin**

staticQueue.head := 1;

staticQueue.tail := 1;

**end**;

**function** IsStaticQueueEmpty: boolean;//Функция проверяет, пуста ли статическая очередь

**begin**

IsStaticQueueEmpty := (staticQueue.head = staticQueue.tail);

**end**;

**procedure** EnqueueStatic(value: integer);//Процедура добавляет элемент в конец статической очереди

**begin**

**if** staticQueue.count = 5 **then**

**begin**

writeln('Очередь заполнена');

**exit**;

**end**;

staticQueue.data[staticQueue.tail] := value;

staticQueue.tail := staticQueue.tail + 1;

staticQueue.count := staticQueue.count + 1;

**end**;

**function** DequeueStatic: integer;//Функция удаляет и возвращает первый элемент из статической очереди

**begin**

**if** IsStaticQueueEmpty **then**

**begin**

writeln('Очередь пуста');

DequeueStatic := 0;

**end**

**else**

**begin**

DequeueStatic := staticQueue.data[staticQueue.head];

staticQueue.head := staticQueue.head + 1;

**end**;

**end**;

**procedure** DisplayStaticQueue;//Процедура выводит содержимое статической очереди

**var**

i: integer;

**begin**

**if** IsStaticQueueEmpty **then**

writeln('Очередь пуста')

**else**

**begin**

writeln('Содержимое очереди:');

**for** i := staticQueue.head **to** staticQueue.tail - 1 **do**

writeln(staticQueue.data[i]);

**end**;

**end**;

**begin**

choice := 0;

**while** choice <> 3 **do**

**begin**

writeln('Выберите тип очереди:');

writeln('1. Очередь на основе динамической памяти');

writeln('2. Очередь на основе статической памяти');

writeln('3. Выход');

readln(choice);

**case** choice **of**

1: **begin**

InitializeDynamicQueue;

writeln('Очередь на основе динамической памяти');

**end**;

2: **begin**

InitializeStaticQueue;

writeln('Очередь на основе статической памяти');

**end**;

3: **begin**

writeln('Выход из программы');

**break**;

**end**;

**end**;

**while** choice <> 4 **do**

**begin**

writeln('1. Вставить элемент в очередь');

writeln('2. Удалить элемент из очереди');

writeln('3. Отобразить содержимое очереди');

writeln('4. Выход');

writeln('Выберите действие:');

readln(choice);

**case** choice **of**

1: **begin**

writeln('Введите значение для вставки в очередь:');

readln(value);

**if** choice = 1 **then**

EnqueueDynamic(value)

**else**

EnqueueStatic(value);

**end**;

2: **begin**

**if** choice = 2 **then**

value := DequeueDynamic

**else**

value := DequeueStatic;

**if** value <> 0 **then**

writeln('Удаленный элемент: ', value);

**end**;

3: **begin**

**if** choice = 3 **then**

DisplayDynamicQueue

**else**

DisplayStaticQueue;

**end**;

**end**;

**end**;

**end**;

**end**.

**Результат выполнения программы:**

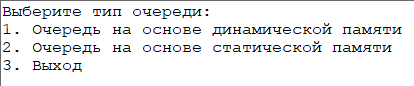
****

Рисунок 5 – выбор памяти

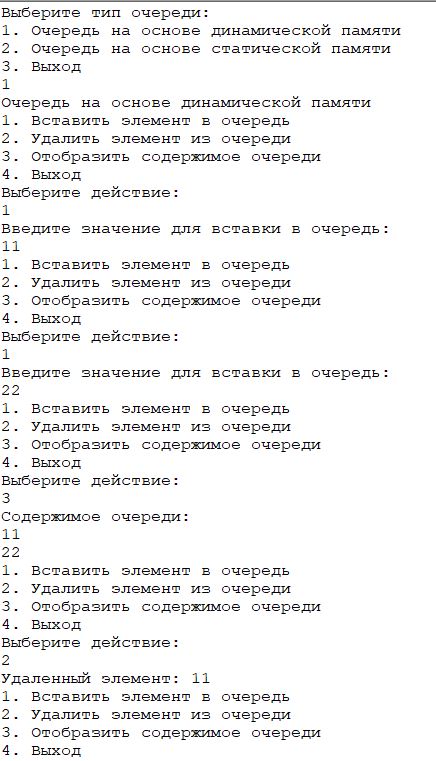
****

Рисунок 6 – работа с динамической памятью

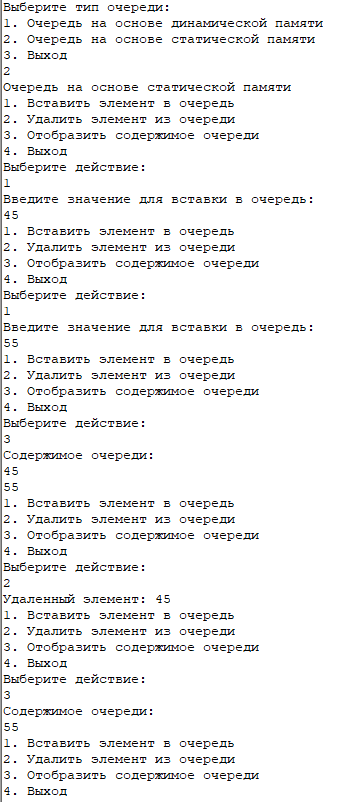
****

Рисунок 7 – работа с статической памятью

**Вывод:**

Таким образом, динамическая структура – это данные, внутреннее строение которых подчиняется определенному закону, но количество элементов, их расположение и взаимосвязи могут динамически изменятся во время выполнения программы.

Данная работа посвящена реализации очереди — это динамическая структура данных, у которой в каждый момент времени доступны только два элемента: первый и последний.

Добавление элементов возможно только с одного конца (конца очереди), а удаление элементов — только с другого конца (начала очереди).

Существует сокращение для очереди: FIFO = First In — First Out, с английского — «Кто первым вошел, тот первым вышел».

Для очереди доступны следующие операции:

* добавить элемент в конец очереди (PushTail);
* удалить элемент с начала очереди (Pop)

В ходе выполнения работы были встречены проблемы, такие как создание счетчика, который проверяет переполнение очереди.

Домашняя контрольная работа №2 была выполнена в полном объёме.