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»**

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

Выполнил: студент учебной группы

ИСПк-202-52-00

Карманова Мария Игоревна

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

**Цель работы** – изучение принципов работы с базовыми структурами данных, получение навыков организации сазе-меню.

**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 сазе-меню. Предусмотреть наглядную визуализацию содержимого структуры.

## **1 Описание алгоритма**

1. Программа начинает работу с главного меню, предлагая пользователю выбрать одно из следующих действий:

- Добавить элемент в очередь (динамическая структура)

- Извлечь элемент из очереди (динамическая структура)

- Добавить элемент в очередь (статическая структура)

- Извлечь элемент из очереди (статическая структура)

- Выйти из программы

2. В зависимости от выбора пользователя, программа вызывает соответствующую процедуру или функцию:

Для динамической структуры:

- Процедура PushTail создает новый узел в динамической очереди и добавляет его в конец очереди.

- Функция Pop извлекает элемент из головы очереди, обновляет указатели на голову и хвост очереди, и возвращает извлеченное значение

Для статической структуры:

- Процедура Enqueue добавляет новый элемент в конец статической очереди, увеличивая значение переменной хвост.

- Функция Dequeue извлекает элемент из начала статической очереди, увеличивая значение переменной голова.

3. Программа продолжает выполнять запросы пользователя, пока тот не выберет пункт "Выход".

## **2 Схема алгоритма с комментариями**

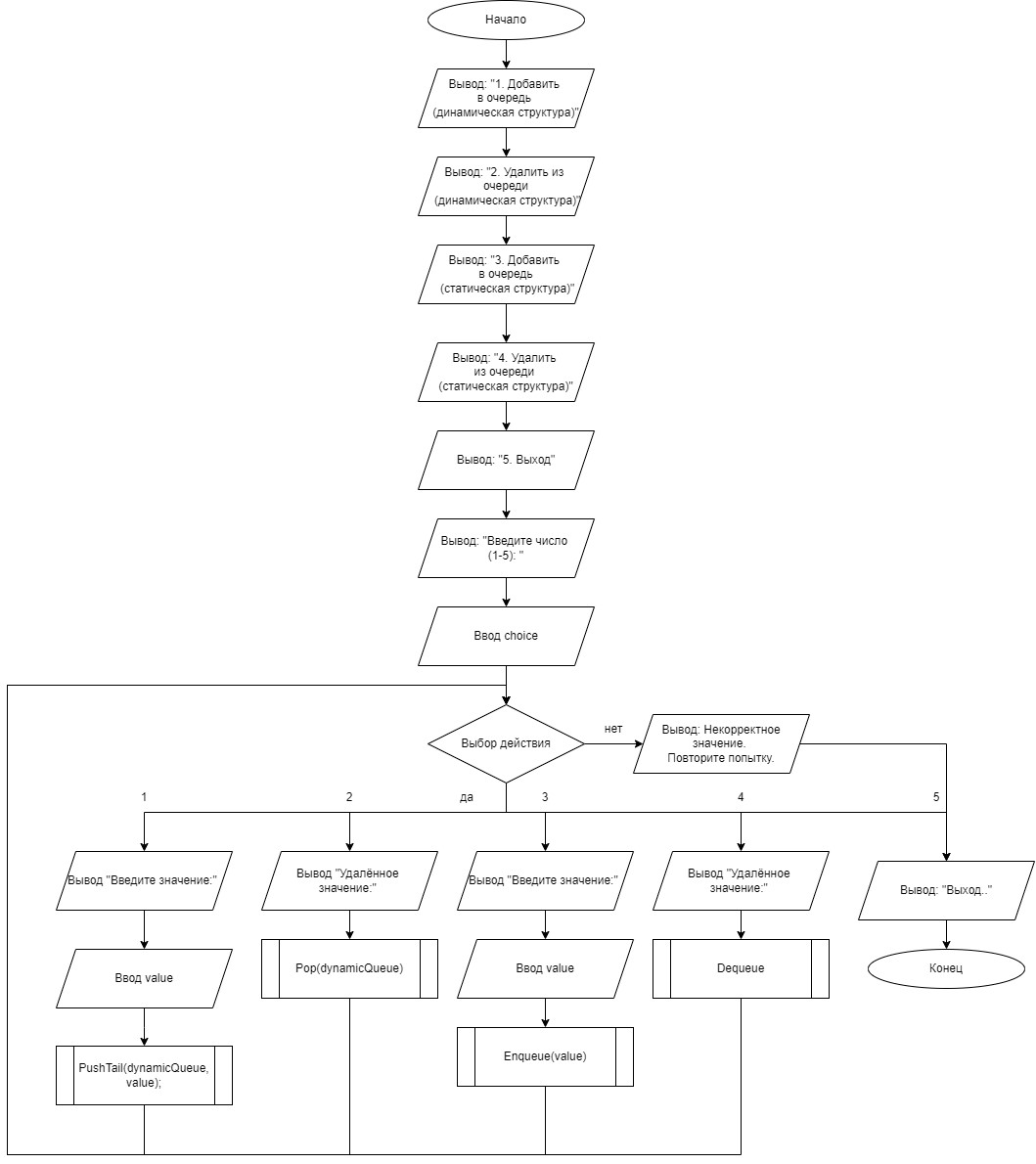


Рисунок 1 – Схема алгоритма основного тела программы

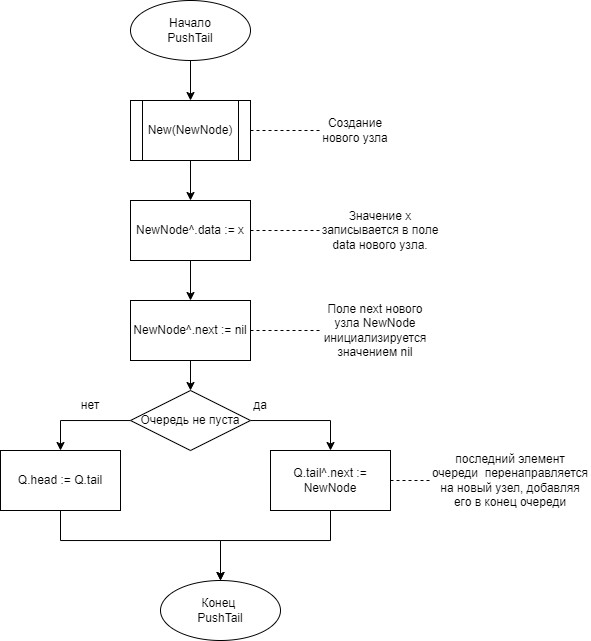


Рисунок 2 – Схема алгоритма процедуры PushTail

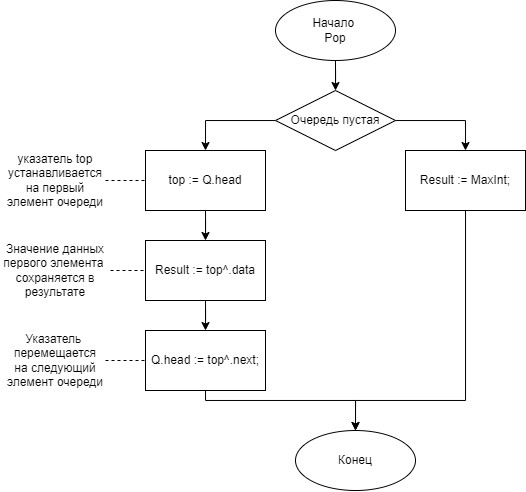


Рисунок 3 – Схема алгоритма функции Pop

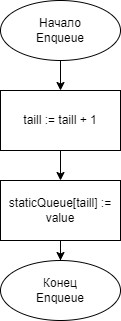


Рисунок 4 – Схема алгоритма процедуры Enqueue

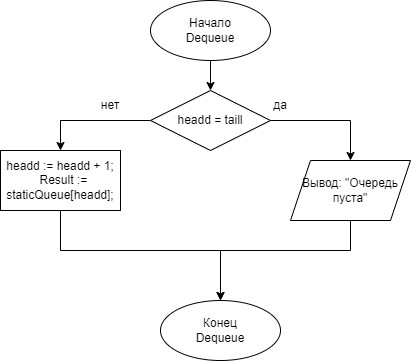


Рисунок 5 – Схема алгоритма функции Dequeue

## **3 Код программы**

**program** Queue;

**uses**

crt;

**type**

PNode = ^Node;

Node = **record**

data: integer;

next: PNode;

**end**;

TQueue = **record**

head, tail: PNode;

Cnt: Integer;

**end**;

**var**

staticQueue: **array**[1..5] **of** integer;

dynamicQueue: TQueue;

choice, value, headd, taill: integer;

**procedure** PushTail(**var** Q: TQueue; x: integer);

**var**

NewNode: PNode;

**begin**

**New**(NewNode);

NewNode^.data := x;

NewNode^.next := nil;

**if** Q.tail <> nil **then**

Q.tail^.next := NewNode;

Q.tail := NewNode;

**if** Q.head = nil **then**

Q.head := Q.tail;

Inc(Q.Cnt);

**end**;

**function** Pop(**var** Q: TQueue): integer;

**var**

top: PNode;

**begin**

**if** Q.head = nil **then**

**begin**

Result := MaxInt;

**Exit**;

**end**;

top := Q.head;

Result := top^.data;

Q.head := top^.next;

**if** Q.head = nil **then**

Q.tail := nil;

Dispose(top);

Dec(Q.Cnt);

**end**;

**procedure** ShowQueue(Q: TQueue);

**var**

tmp: PNode;

**begin**

tmp := Q.head;

**while** tmp <> nil **do**

**begin**

Write(tmp^.data, ' ');

tmp := tmp^.next;

**end**;

Writeln;

**end**;

**procedure** Enqueue(value: integer);

**begin**

taill := taill + 1;

staticQueue[taill] := value;

**end**;

**function** Dequeue: integer;

**begin**

**if** headd = taill **then**

**raise** Exception.Create('Очередь пуста');

headd := headd + 1;

Result := staticQueue[headd];

**end**;

**begin**

**repeat**

writeln('1. Добавить в очередь (динамическая структура)');

writeln('2. Удалить из очереди (динамическая структура)');

writeln('3. Добавить в очередь (статическая структура)');

writeln('4. Удалить из очереди (статическая структура)');

writeln('5. Просмотр очереди');

writeln('6. Выход');

write('Введите число (1-6): ');

readln(choice);

**case** choice **of**

1:

**begin**

write('Введите значение: ');

readln(value);

PushTail(dynamicQueue, value);

**end**;

2:

**begin**

writeln('Удалённое значение: ', Pop(dynamicQueue));

**end**;

3:

**begin**

write('Введите значение: ');

readln(value);

Enqueue(value);

**end**;

4:

**begin**

writeln('Удалённое значение: ', Dequeue);

**end**;

5:

**begin**

writeln('Очередь (динамическая структура):');

ShowQueue(dynamicQueue);

writeln;

writeln('Очередь (статическая структура):');

**for** headd := 1 **to** taill **do**

Write(staticQueue[headd], ' ');

writeln;

**end**;

6: writeln('Выход...');

**else**

writeln('Некорректное значение. Повторите попытку.');

**end**;

**until** choice = 6;

**end**.

## **4 Результат выполнения программы**

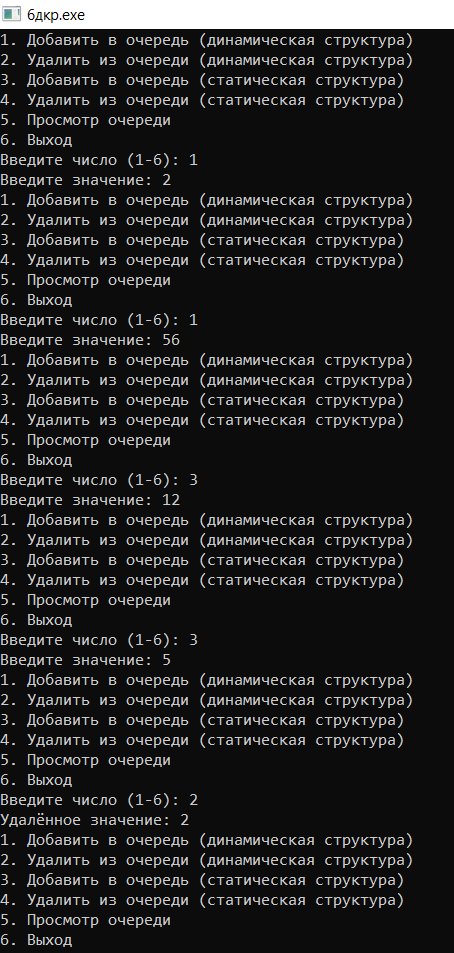


Рисунок 6 – Результат выполнения программы

## **5 Вывод**

В ходе выполнения данной контрольной работы были изучены принципы работы с очередью, получены навыки организации case-меню. Очередь — 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).