Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Вятский государственный университет»

Колледж ВятГУ

**ОТЧЕТ**

**ПО ДОМАШНЕЙ КОНТРОЛЬНОЙ РАБОТЕ №2**

**«Реализация элементарных структур данных на основе статической памяти»**

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

Выполнил: студент учебной группы ИСПк-202-52-00

Причинин Вадим Андреевич

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2024

Цель работы: изучение принципов работы с базовыми структурами данных, получение навыков организации case-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 case-меню. Предусмотреть наглядную визуализацию содержимого структуры.

**Описание алгоритма:**

**О́чередь —** абстрактный тип данных с дисциплиной доступа к элементам «первый пришёл — первый вышел» ([FIFO](https://ru.wikipedia.org/wiki/FIFO), [англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *first in, first out*). Добавление элемента (принято обозначать словом enqueue — поставить в очередь) возможно лишь в конец очереди, выборка — только из начала очереди (что принято называть словом dequeue — убрать из очереди), при этом выбранный элемент из очереди удаляется.

При динамической реализации используется двухсвязный список, у которого есть голова и хвост. При удалении элемента мы смещаем голову на следующий элемент очереди, при добавлении элемента мы к хвосту добавляем ссылку на новый элемент (тем самым он становится хвостом).

При статической реализации используется массив. При удалении элемента мы сдвигаем элементы очереди на один, при добавлении элемента мы добавляем элемент в конец массива.

**Схема алгоритма:**

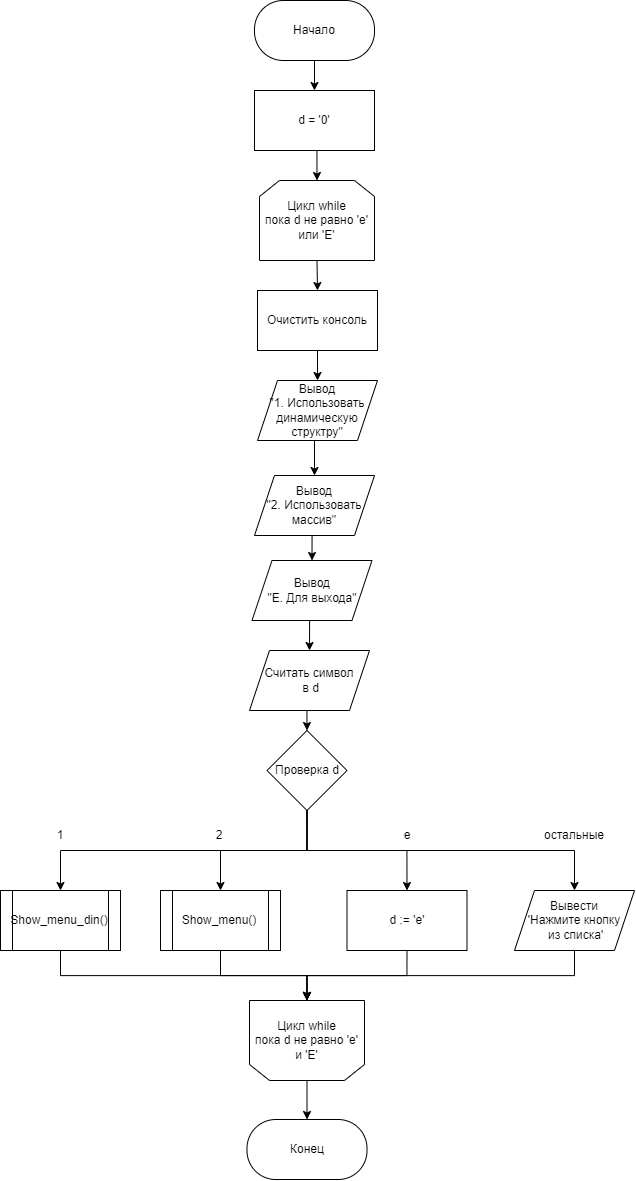


Рисунок 1 — Основное тело программы

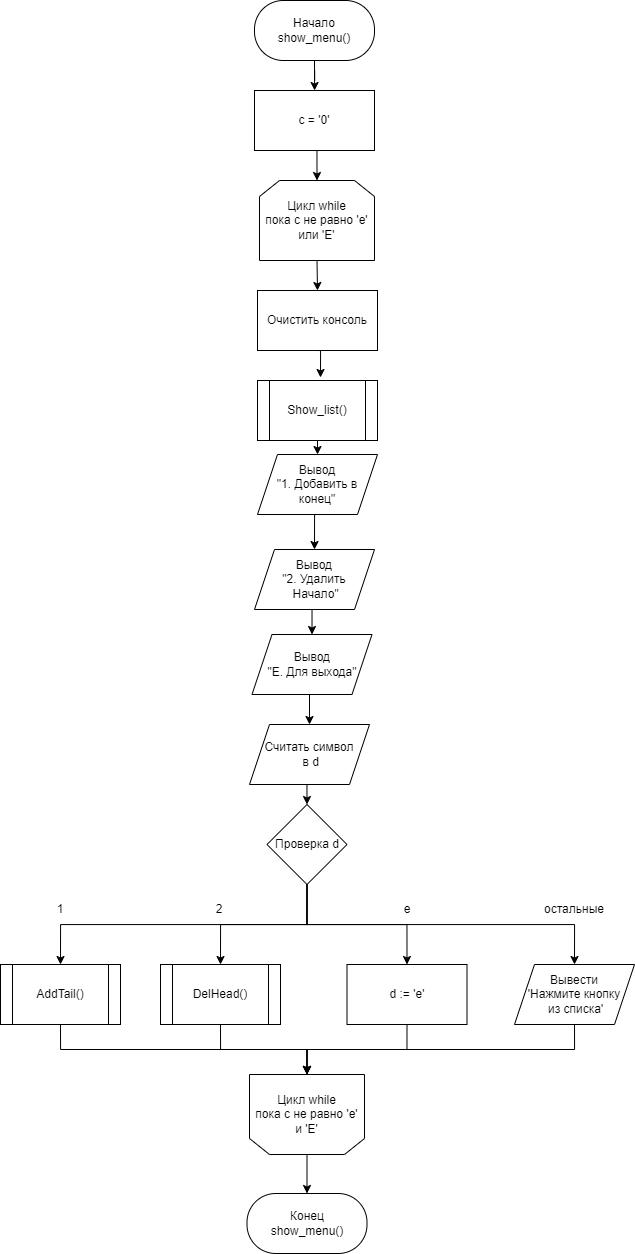


Рисунок 2 — подпрограмма show\_menu

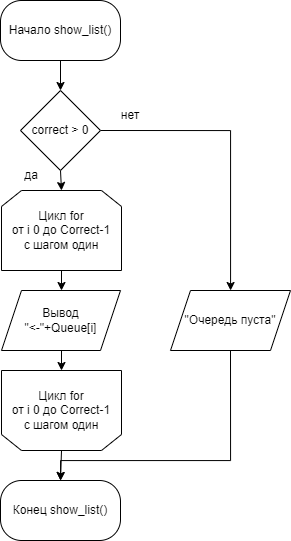


Рисунок 3 — Подпрограмма show\_list

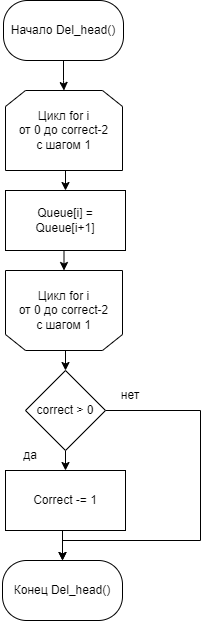


Рисунок 4 — подпрограмма del\_head

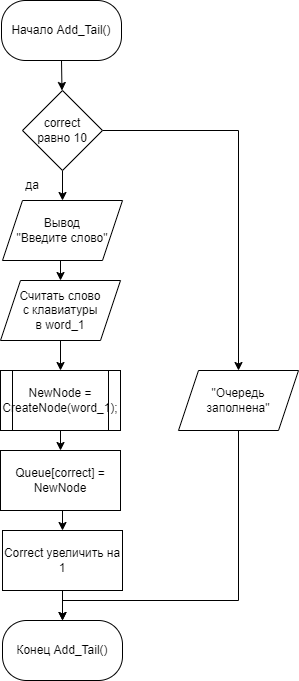


Рисунок 5 — подпрограммы add\_tale

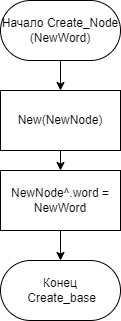


Рисунок 6 — подпрограммы Create\_node

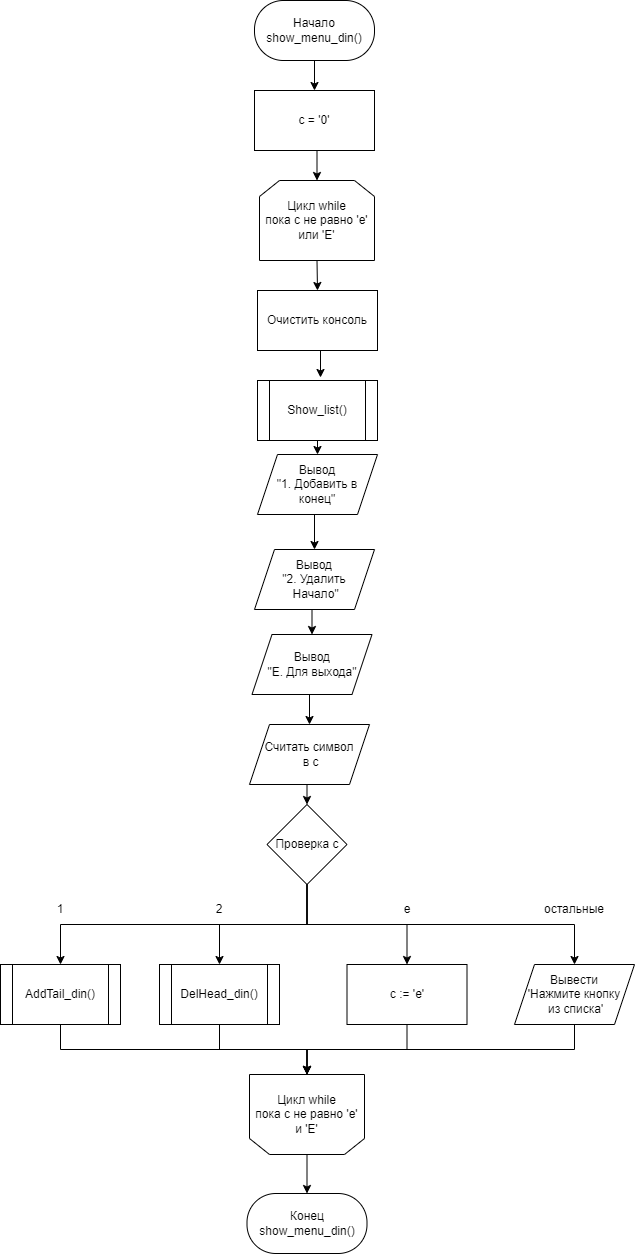


Рисунок 7 — подпрограммы show\_menu\_din

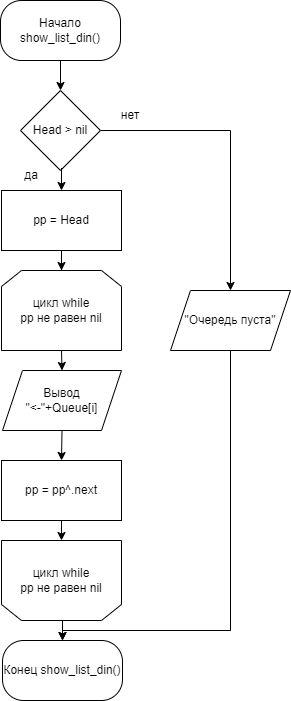


Рисунок 8 — подпрограммы show\_list\_din

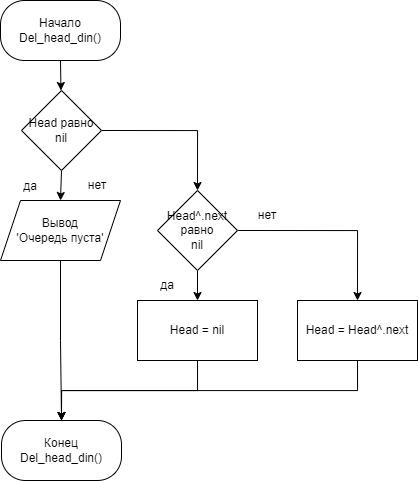


Рисунок 9 — подпрограммы Del\_head\_din

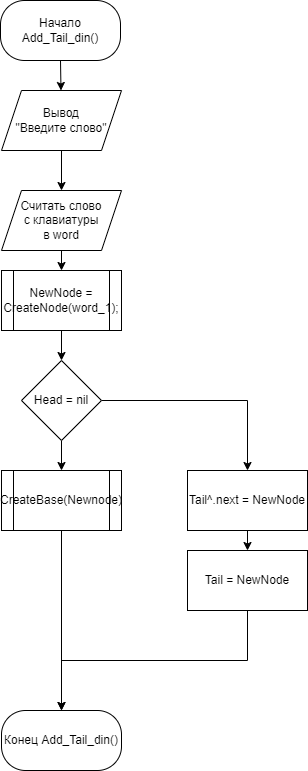


Рисунок 10 — подпрограммы add\_tale\_din

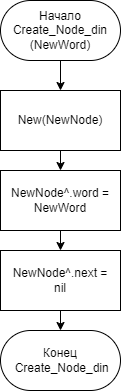


Рисунок 11 — подпрограммы Create\_node\_din

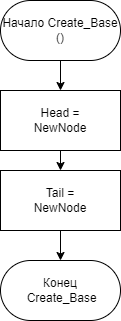


Рисунок 12 — подпрограммы Create\_base

**Код программы**

**program** Queue\_com;

**uses** crt;

**type** PNode\_din = ^Node\_din; { указатель на узел }

Node\_din = **record** { узел и его структура }

word: string[40]; { поле для слова }

next: PNode\_din; { поле - ссылка на следующий узел }

prev: PNode\_din; { поле - ссылка на предыдущий узел }

**end**;

**procedure** CreateBase ( **var** Head, Tail: PNode\_din; NewNode: PNode\_din);

**begin**

Head := NewNode;

Tail := NewNode;

**end**;

**procedure** DelHead\_din ( **var** Head: PNode\_din);

**begin**

**if** Head = nil **then**

Writeln('Очередь пуста')

**else if** Head^.next = nil **then**

**begin**

**var** a := Head;

Head := nil;

dispose(a);

**end**

**else**

**begin**

**var** a := Head;

Head := Head^.next;

dispose(a);

**end**

**end**;

**procedure** show\_list\_din(Head: PNode\_din);

**var** pp: PNode\_din; correct: integer;

**begin**

**if** Head = nil **then**

Write('Очередь пуста')

**else**

pp := Head;

**while** pp <> nil **do**

**begin**

Write('<-',pp^.word);

pp := pp^.next;

**end**;

**end**;

**function** CreateNode\_din(NewWord: string): PNode\_din;

**var** NewNode: PNode\_din;

**begin**

**New**(NewNode);

NewNode^.word := NewWord;

NewNode^.next := nil;

Result := NewNode;

**end**;

**procedure** AddTail\_din ( **var** Head,Tail: PNode\_din);

**begin**

Writeln('Введите слово');

**var** word:string;

Readln(word);

**var** NewNode := CreateNode\_din(word);

**if** Head = nil **then**

CreateBase(Head,Tail,Newnode)

**else**

**begin**

Tail^.next := NewNode;

Tail := NewNode;

**end**

**end**;

**var** c := '0';

Head, Tail: PNode\_din;

**Procedure** Show\_menu\_din();

**begin**

Head := nil;

Tail := nil;

**while**((c <> 'e') **and** (c<>'E')) **do**

**begin**

Clrscr();

show\_list\_din(Head);

Writeln();

Writeln('1. Добавить в конец');

Writeln('2. Удалить Начало');

Writeln('e. Выход');

c:=ReadKey();

**if** (c = '1') **then**

AddTail\_din(Head,Tail)

**else if** (c = '2') **then**

DelHead\_din(Head)

**else if** (c = 'e') **then**

**begin**

c := '0';

**break**;

**end**

**else** Writeln('Нажмите кнопку из списка');Sleep(1000);

**end**;

**end**;

**type** PNode = ^Node; { указатель на узел }

Node = **record** { узел и его структура }

word: string[40]; { поле для слова }

**end**;

**function** CreateNode(NewWord: string): PNode;

**var** NewNode: PNode;

**begin**

**New**(NewNode);

NewNode^.word := NewWord;

Result := NewNode;

**end**;

**var**

Correct:= 0;

Queue: **array of** PNode;

**procedure** DelHead ( **var** Queue: **array of** PNode);

**begin**

**for var** i:=0 **to** correct-2 **do**

Queue[i] := Queue[i+1];

**if** correct > 0 **then**

Correct -= 1

**end**;

**procedure** AddTail (**var** Queue: **array of** PNode; **var** Correct: integer);

**var** word\_1:string;NewNode:PNode;

**begin**

**if** correct <> 10 **then**

**begin**

Writeln('Введите слово');

Readln(word\_1);

NewNode := CreateNode(word\_1);

Queue[correct] := NewNode;

Correct += 1;

**end**

**else**

Writeln('Очередь заполнена')

**end**;

**procedure** show\_list(Queue: **array of** PNode; Correct:integer);

**begin**

**if** correct > 0 **then**

**for var** i:=0 **to** Correct-1 **do**

Write('<-',Queue[i]^.word)

**else**

write('Очередь пуста')

**end**;

**Procedure** show\_menu();

**begin**

**var** stat := '0';

SetLength(Queue,10);

**while**((stat <> 'e') **and** (stat<>'E')) **do**

**begin**

Clrscr();

show\_list(Queue, Correct);

Writeln();

Writeln('1. Добавить в конец');

Writeln('2. Удалить Начало');

Writeln('e. Выход');

stat:=ReadKey();

**if** (stat = '1') **then**

AddTail(Queue,Correct)

**else if** (stat = '2') **then**

DelHead(Queue)

**else if** (stat = 'e') **then**

**begin**

stat := '0';

**break**;

**end**

**else** Writeln('Нажмите кнопку из списка');Sleep(1000);

**end**;

**end**;

**var** d:='0';

**begin**

**while**((d <> 'e') **and** (d <>'E')) **do**

**begin**

Clrscr();

Writeln('1. Использовать динамическую структуру');

Writeln('2. Использовать массив(10 чисел)');

Writeln('e. Выход');

d:=ReadKey();

**if** (d = '1') **then**

Show\_menu\_din()

**else if** (d = '2') **then**

Show\_menu()

**else if** (d = 'e') **then**

d := 'e'

**else** Writeln('Нажмите кнопку из списка');Sleep(1000);

**end**;

**end**.

**Результаты выполнения программы**

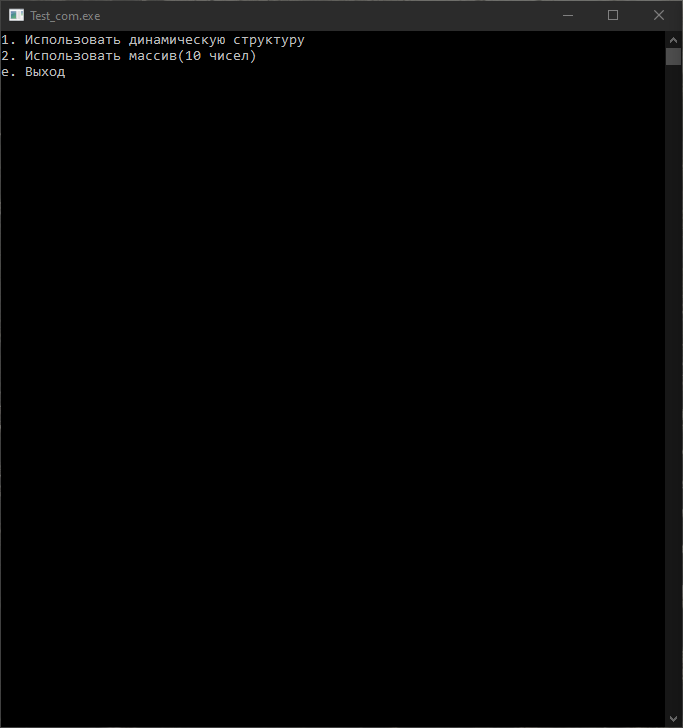


Рисунок 13 — Начальное case-menu

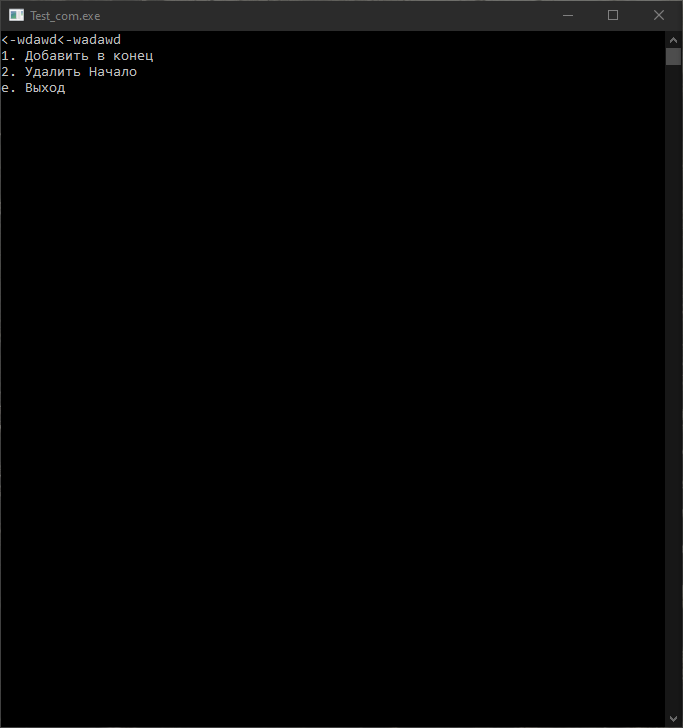


Рисунок 14 — case-menu динамической части

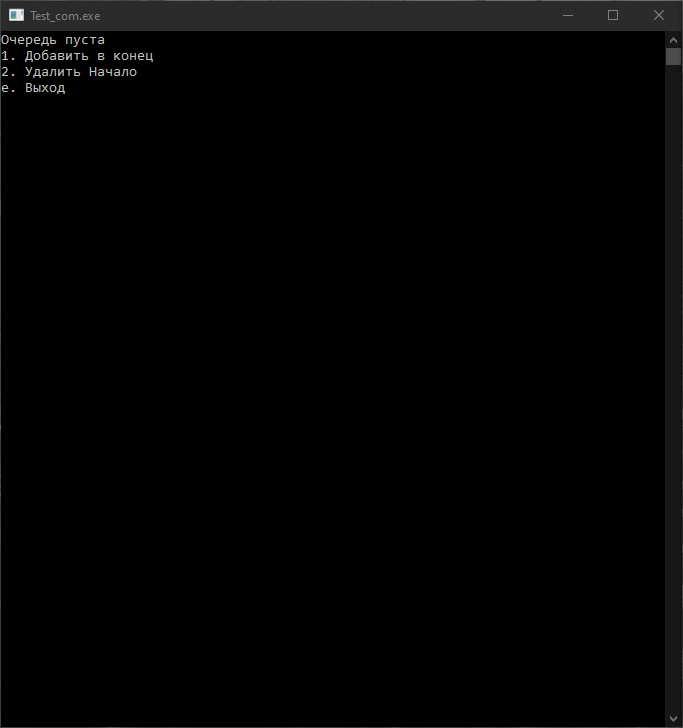


Рисунок 15— case-menu статической части

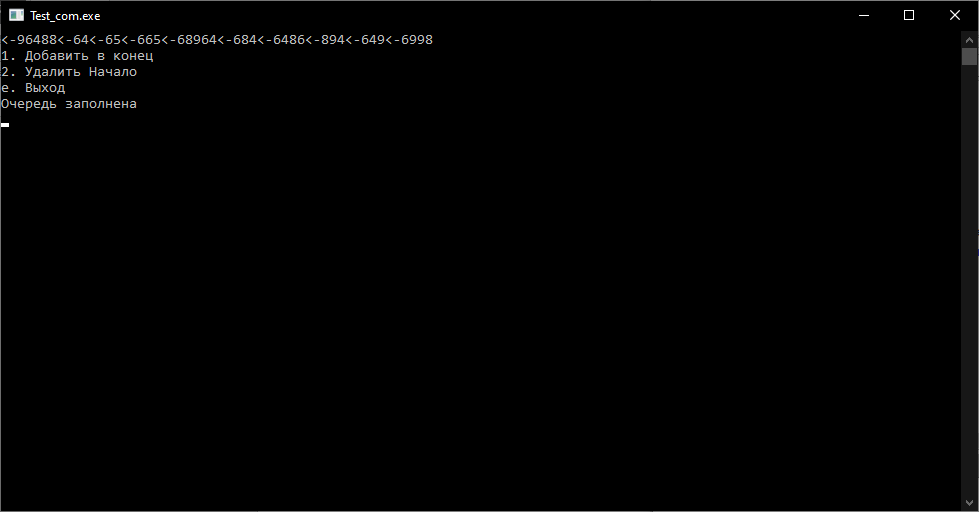


Рисунок 16— попытка добавить новый элемент при полном массиве

**Вывод:**

В ходе работы был изучена реализация динамической структуры данных — очереди. При реализации алгоритма алгоритм был реализован в двух вариациях в статической(массив) и динамической (односвязный список), что позволило лучше понять структуру данных. Также в программе реализовано управление с помощью case-menu: для управления режимом работы программы, а также для взаимодействия с очередью (структурой данных). В ходе работы возникли проблемы с реализацией case\_menu, а также вынесением кода в отдельные модули.