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-203-52-00

Широков Михаил Александрович

Преподаватель:

Сергеева Елизавета Григорьевна

Киров

2025

**1. Цель работы**

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

**2. Формулировка задания**

1. Написать программу для работы со структурой данных «Стек»;

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.

**3. Описание алгоритма**

**Стек** — динамическая структура данных, в которой добавление и удаление элементов доступно только с одного конца (с верхнего (последнего) элемента).

**Реализация стека и алгоритмов работы с ним при помощи статической памяти**

Для реализации стека необходимо предусмотреть следующие компоненты:

1. Способ организации стека и его хранения в памяти;

2. Функцию извлечения элемента из стека;

3. Функцию добавления элемента в стек

1. Организация стека при помощи статической памяти.

Для того чтобы реализовать стек на основе статической памяти необходимо параметра – статический массив n-ого количества мест и переменная, которая отвечает за последний незаполненный элемент этого массива (переменная головы «Head»). Для удобства можно создать тип записи, присвоив ему идентификатор «Stack», в которой должно быть 2 поля – сам массив, в котором должны храниться элементы стека и «голова».

2. Функцию извлечения элемента из стека;

Для реализации этой функции необходимы 2 подпрограммы – функция, определяющая пуст ли стек и функция, выполняющая извлечение определенного количества элементов из стека.

**Дополнительная процедура stupid\_input** – запрос клавиши пользователя для выхода из разделов case-меню.

**Алгоритм работы:**

a) Процедура начинает работу;

b) Вывод пустой строки;

c) Вывод сообщения о необходимости нажать любую клавишу для выхода в меню;

d) Считывание клавиши;

e) Процедура завершает работу;

Примечание: данная процедура будет использована в конце каждого раздела меню, т.е. работы таких функций как stack\_append, stack\_pop, stack\_output.

**Функция is\_stack\_empy** – определяет пустоту стека.

**Принимает параметры:**

По значению:

S : stack – заранее созданный тип записи для реализации стека

**Тип возвращаемого значения: boolean**

**Алгоритм работы:**

a) Начало работы функции;

b) Осуществляется проверка значения «головы стека», если она равна 1, значит стек не содержит элементов и необходимо вернуть True, иначе вернуть False;

c) Конец работы функции.

**Процедура stack\_pop** – удаляет определенное количество элементов стека.

**Принимает параметры:**

По ссылке:

S : stack – заранее созданный экземпляр типа записи для реализации стека

**Алгоритм работы**

a) Начало работы функции;

b) Очистка окна консоли при помощи clrscr;

c) Запрос на ввод количество элементов, которые необходимо удалить;

d) Переменной I присваивается значение 1 и запускается цикл с предусловием (со счетчиком). Переменная цикла I будет изменятся в промежутке от 1 до количества элементов, которые необходимо удалить. В данном цикле будет поочередно происходить извлечение и удаление элементов стека. В теле цикла:

d.1) Проверяется значение функции is\_stack\_empy:

d.1.1.1) Если значение этой функции ложно, значит стек не пуст, и можно удалить элемент. Вначале меняем цвет текста консоли на зеленый;

d.1.1.2) Выводим текущий элемент стека и сообщение о его удалении;

d.1.1.3) Возвращаем обычный цвет консоли (голубой);

d.1.1.4) Элементу массива поля stack\_array записи S по индексу поля head-1 записи S присваивается нулевое значение (пустая строка). Примечание: т.к. голова стека всегда указывает на пустой элемент, то для того, чтобы обратиться к текущему необходимо использовать head-1;

d.1.1.5) Уменьшаем значение поля S.head на еденицу;

d.1.1.6) Увеличиваем переменную I на 1;

d.1.2.1) Если значение этой функции ложно, значит стек пуст, необходимо сообщить об этом пользователю. Вначале меняем цвет текста консоли на красный.

d.1.2.2) Выводим сообщение пользователю о невозможности удаления элемента стека;

d.1.2.3) Возвращаем обычный цвет консоли (голубой);

d.1.2.4) Выходим из цикла;

e) После выхода из цикла запрашиваем ввод любой клавиши для перехода в меню при помощи stupid\_input();

f) Процедура завершает работу.

3. Функция добавления элементов в стек.

Для реализации этой функции необходимы 2 подпрограммы – функция, определяющая полон ли стек и функция, добавляющая элемент в стек.

**Функция is\_stack\_full** – определяет заполненность стека.

**Принимает параметры:**

По значению:

S : stack – заранее созданный тип записи для реализации стека

**Тип возвращаемого значения: boolean**

**Алгоритм работы:**

a) Начало работы функции;

b) Осуществляется проверка значения «головы стека», если она больше значения длинны массива стека, значит стек заполнен и необходимо вернуть True, иначе вернуть False;

c) Конец работы функции.

**Функция stack\_append** – добавляет элемент в стек и возвращает результат добавления в виде логической переменной.

**Принимает параметры:**

По ссылке:

S : stack – заранее созданный экземпляр типа записи для реализации стека

**Алгоритм работы**

a) Начало работы функции;

b) Очистка окна консоли при помощи clrscr;

c) Запрос на ввод количество элементов, которое необходимо добавить;

d) Переменной I присваивается значение 1 и запускается цикл с предусловием (со счетчиком). Переменная цикла I будет изменятся в промежутке от 1 до количества элементов, которые необходимо добавить. В данном цикле будет поочередно происходить извлечение и удаление элементов стека. В теле цикла:

d.1) Проверяется значение функции is\_stack\_full(S):

d.1.1.1) Если значение этой функции ложно, значит в стеке еще осталось место для добавления нового элемента. Вначале считываем новый элемент и присваиваем его значению поля stack\_array по индексу поля haed записи S.

d.1.1.2) Изменяем значение головы текущего стека на 1: S.head +=1;

d.1.1.3) Изменяем цвет текста в консоли на зеленый;

d.1.1.4) Выводим сообщение пользователю об успешном добавлении элемента в стек;

d.1.1.5) Возвращаем обычный цвет консоли (голубой);

d.1.1.6) Уыеличиваем переменную I на 1;

d.1.2.1) Если значение этой функции истина, значит стек переполнен и необходимо вывести сообщение пользователю. Вначале меняем цвет текста в консоли на красный;

d.1.2.2) Выводим сообщении о переполнении стека;

d.1.2.3) Возвращаем обычный цвет консоли (голубой);

d.1.2.4) Выходим из цикла;

e) После выхода из цикла запрашиваем ввод любой клавиши для перехода в меню при помощи stupid\_input();

f) Процедура завершает работу.

**Процедура Stack\_output** – процедура, выводящая все элементы стека.

**Она принимает параметры:**

По значению:

S : stack – заранее созданный экземпляр типа записи для реализации стека

**Алгоритм работы:**

a) Начало работы функции;

b) Очистка окна консоли при помощи clrscr;

с) Переменной I присваивается значение 1 и запускается цикл с предусловием (со счетчиком). Переменная цикла I будет менять свое значение с 1 по значение поля head-1, записи S (т.к. поле head всегда указывает на незаполненный элемент стека). В теле цикла:

с.1) Выводиться значение массива поля stack\_array по индексу I записи S.

c.2) Происходит проверка, равно ли значение I значению S.head-1, т.е. последний ли это элемент стека. Если значение этого I не равно S.head-1, значит это не последний элемент и необходимо вывести дополнительную стрелочку.

c.3) Увеличиваем переменную I на 1;

d) Вызов процедуры stupid\_input по окончанию вывода элементов стека;

e) Конец работы процедуры;

**Основная часть программы**

Программа содержит одну глобальную переменную – main\_stack, типа stack;

**Алгоритм работы:**

a) Начало работы программы;

b) Установка голубого цвета консоли;

с) Скрытие курсора консоли;

d) Запуск бесконечного цикла, в котором будет выводиться case-меню и осуществляться выбор его элементов. В теле цикла:

d.1) Очистка окна консоли при помощи clrscr;

d.2) Вывод заголовка программы «Работа со стеком» и всех разделов меню:

«Пополнить стек», «Извлечь элемент», «Просмотр содержимого», «Выход».

d.3) Считывание клавиши клавиатуры при помощи readkey и занесение значения этой клавиши в переменную key;

d.4) Проверка значения клавиши key:

d.4.1) Если она равна 1: запустить процедуру пополнения стека stack\_input(main\_stack);

d.4.2) Если она равна 2: запустить процедуру удаления элемента из стека stack\_pop(main\_stack);

d.4.3) Если она равна 3: запустить процедуру вывода элементов из стека stack\_output(main\_stack);

d.4.4) Если она равна 4: осуществить завершение работы программы;

d.4.5) При любом другом значении перейти к следующей итерации.

e) Завершение работы программы.

**4. Схемы алгоритма с комментариями**

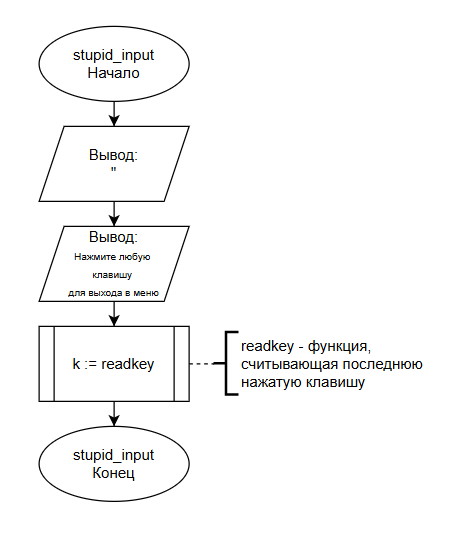


Рисунок 1 – схема алгоритма процедуры stupid\_input

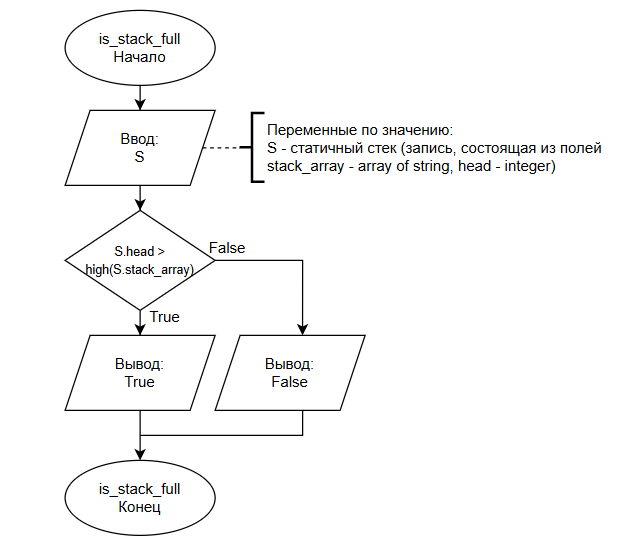


Рисунок 2 – схема алгоритма функции is\_stack\_full

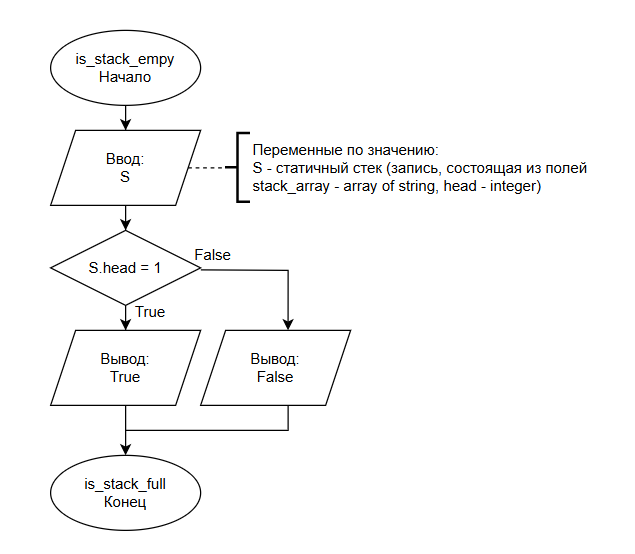


Рисунок 3 – схема алгоритма функции is\_stack\_empy

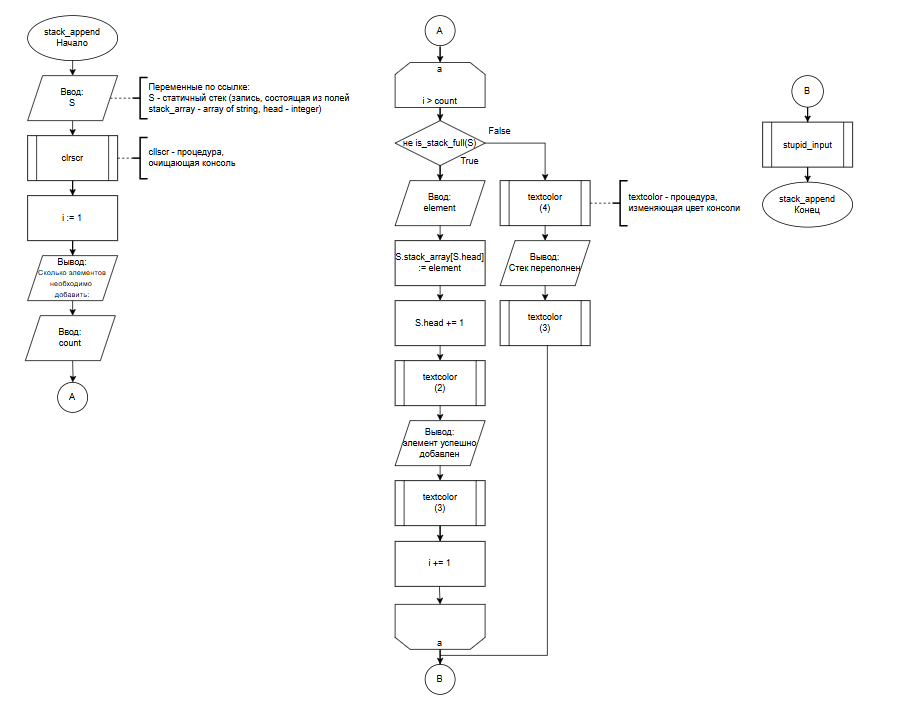


Рисунок 4 – схема алгоритма функции stack\_append

Рисунок 5 – схема алгоритма функции stack\_pop

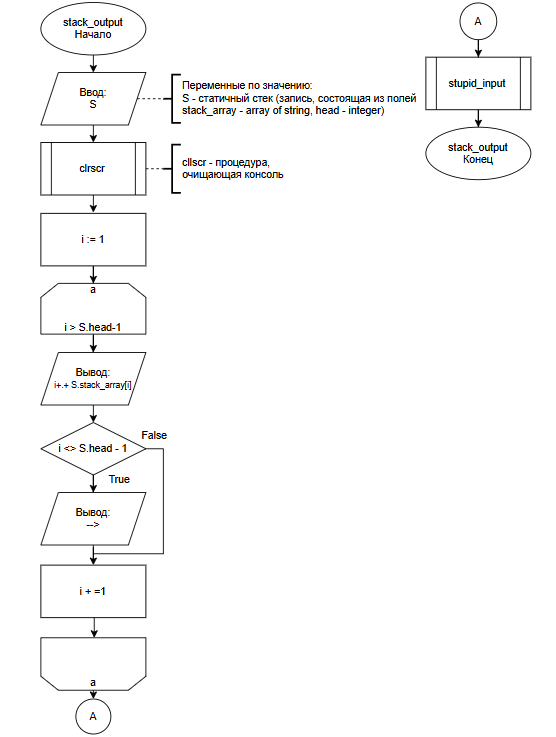
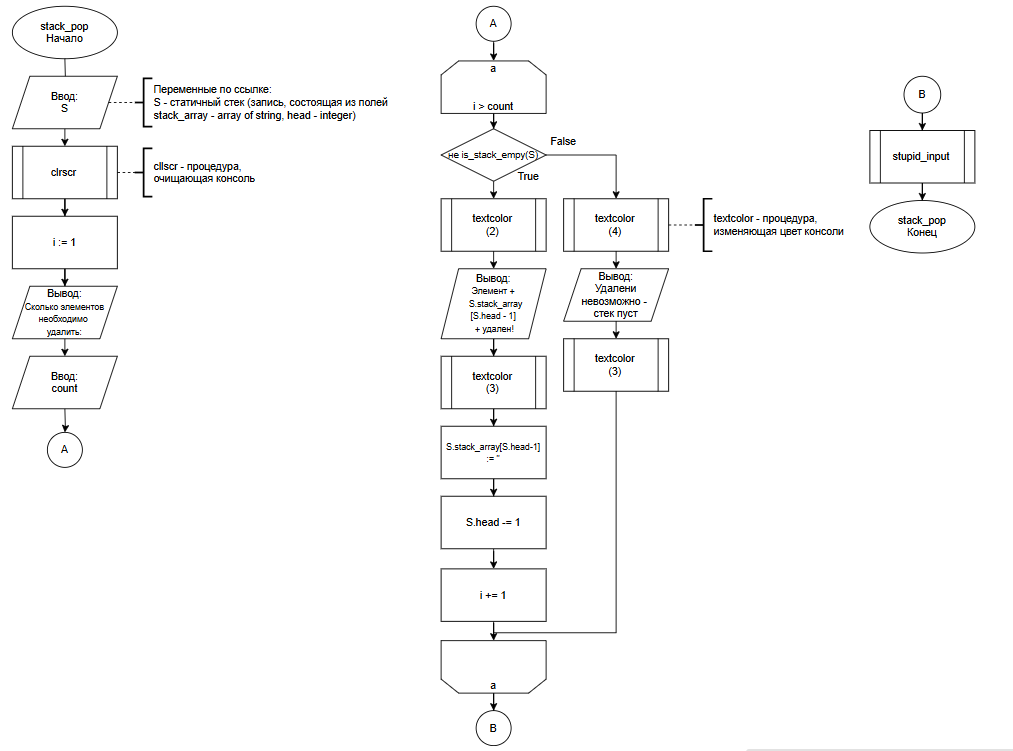


Рисунок 6 – схема алгоритма функции stack\_output

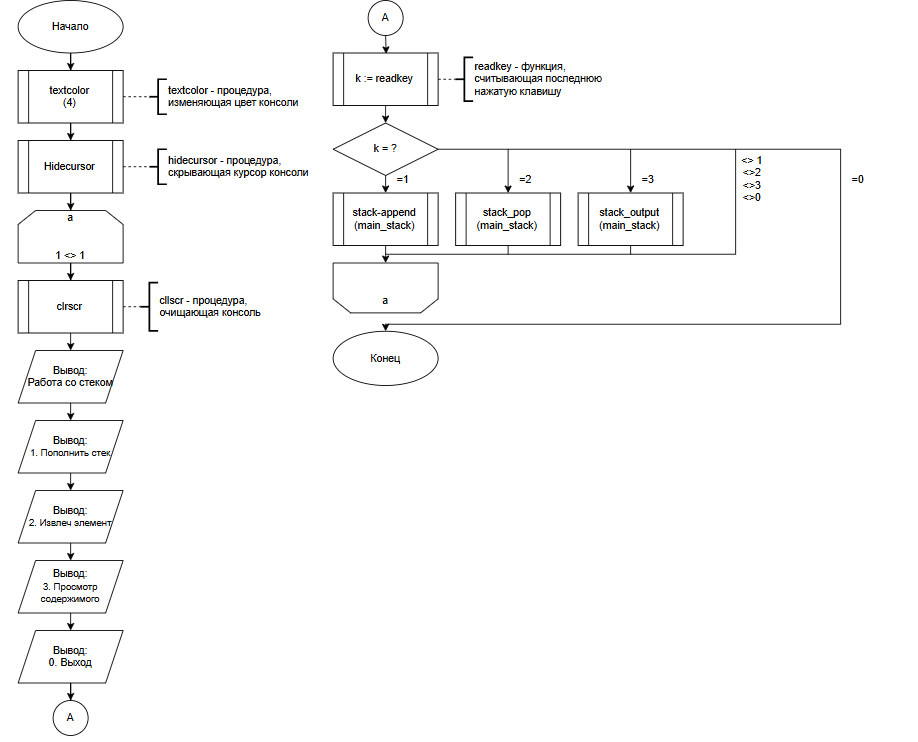


Рисунок 7 – схема основного алгоритма

**5. Код программы**

**uses** crt;

**const**

Stack\_maximum = 25;

**type**

Stack = **record**

stack\_array: **array** [1..Stack\_maximum] **of** string;

head: integer := 1;

**end**;

**var**

main\_stack:stack;

**function** is\_stack\_full(S:stack):boolean;

**begin**

**if** S.head > high(S.stack\_array) **then**

is\_stack\_full := True

**else**

is\_stack\_full := False;

**end**;

**function** is\_stack\_empy(S:stack): boolean;

**begin**

**if** S.head = 1 **then**

is\_stack\_empy := True

**else**

is\_stack\_empy := False;

**end**;

**procedure** stupid\_input();

**begin**

writeln;

println('Нажмите любую клавишу для выхода в меню');

**var** k:= readkey;

**end**;

**procedure** stack\_append(**var** S:stack);

**begin**

clrscr;

**for var** i:= 1 **to** readlninteger('Сколько элементов необходимо добавить: ') **do**

**if not**(is\_stack\_full(S)) **then**

**begin**

S.stack\_array[S.head] := readlnstring('Введите элемент стека: ');

S.head += 1;

textcolor(2);

println('Элемент успешно добавлен!');

textcolor(3);

**end**

**else**

**begin**

textcolor(4);

println('Стек переполнен!!!');

textcolor(3);

**break**;

**end**;

stupid\_input()

**end**;

**procedure** stack\_pop(**var** S:stack);

**begin**

clrscr;

**for var** i:= 1 **to** readlninteger('Сколько элементов необходимо удалить: ') **do**

**if not**(is\_stack\_empy(S)) **then**

**begin**

textcolor(2);

println('Элемент', S.stack\_array[S.head - 1],'удален!');

textcolor(3);

S.stack\_array[S.head - 1] := '';

S.head -= 1;

**end**

**else**

**begin**

textcolor(4);

println('Удаление невозможно - стек пуст!');

textcolor(3);

**break**;

**end**;

stupid\_input()

**end**;

**procedure** stack\_output(S:stack);

**begin**

clrscr;

**for var** i:= 1 **to** S.head - 1 **do**

**begin**

write(i,'.', S.stack\_array[i]);

**if** i <> S.head - 1 **then** print(' -->');

**end**;

stupid\_input()

**end**;

**begin**

textcolor(3);

Hidecursor;

**while** 1=1 **do**

**begin**

ClrScr;

println('Работа со стеком');

println('1. Пополнить стек');

println('2. Извлеч елемент');

println('3. Просмотр содержимого');

println('0. Выход');

**var** key:string := readkey;

**case** key **of**

'1': stack\_append(main\_stack);

'2': stack\_pop(main\_stack);

'3': stack\_output(main\_stack);

'0': halt;

**end**;

**end**;

**end**.

**6. Результаты работы программы**

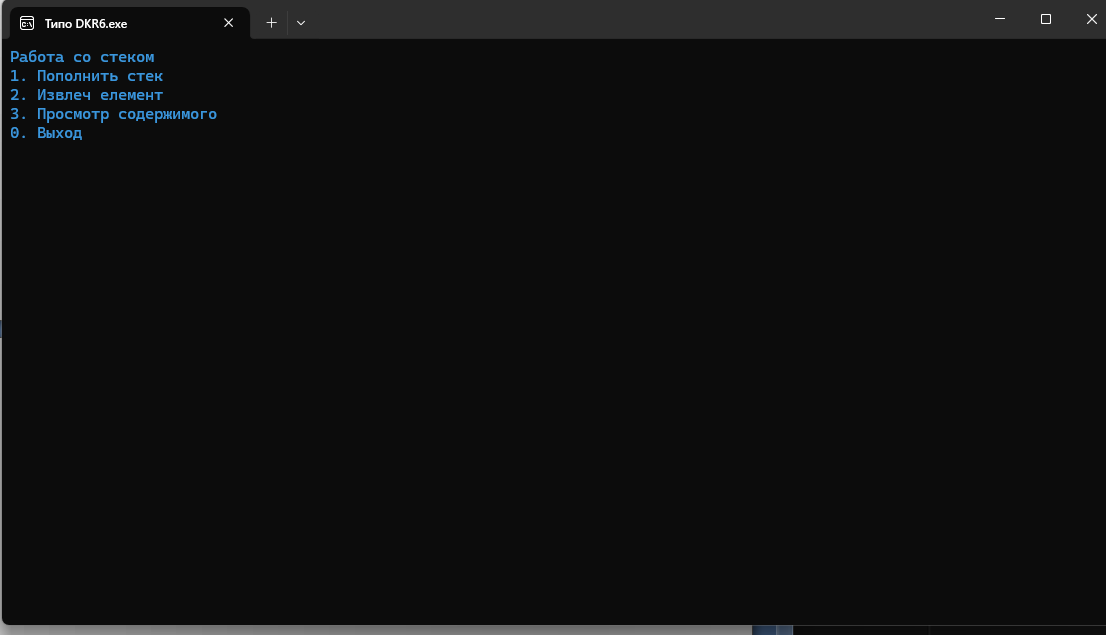


Рисунок 8 – case-меню

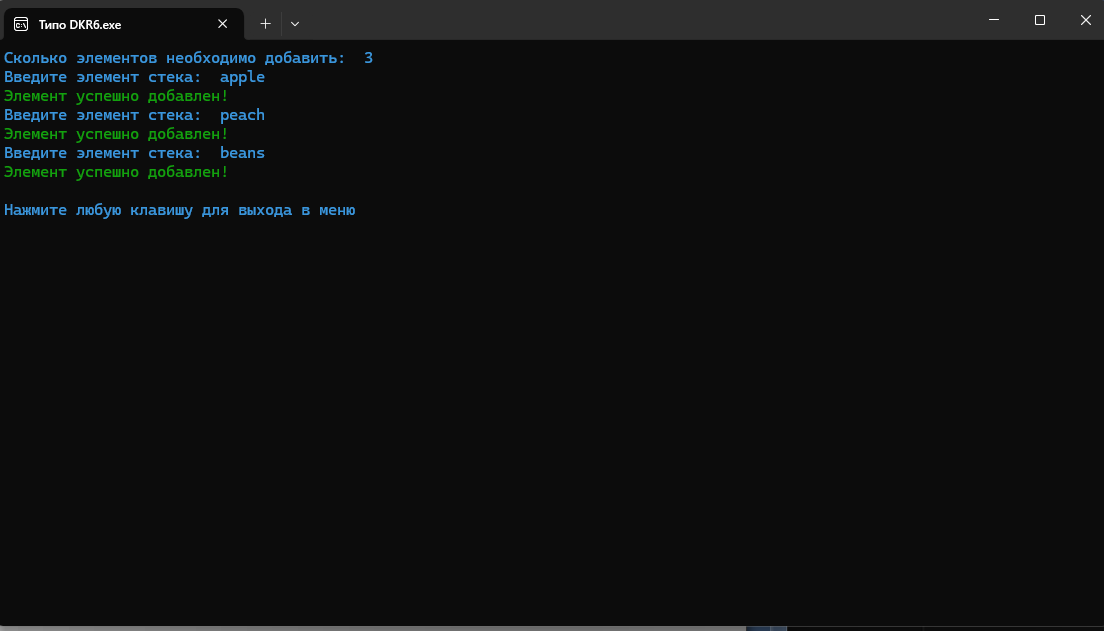


Рисунок 9 – добавление элементов в стек

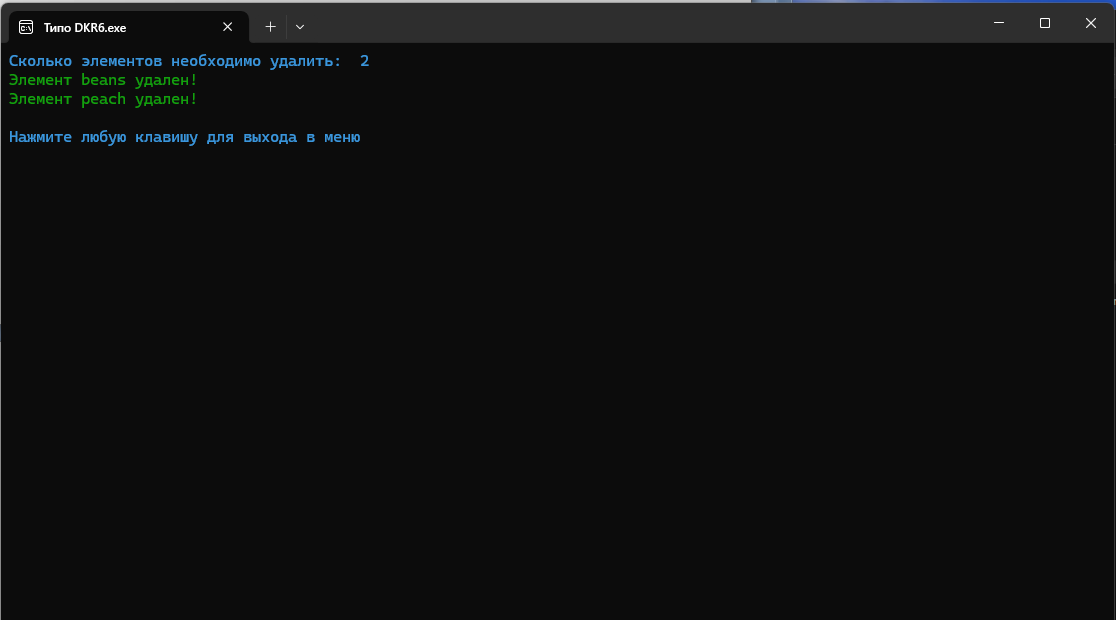


Рисунок 10 – удаление элементов стека

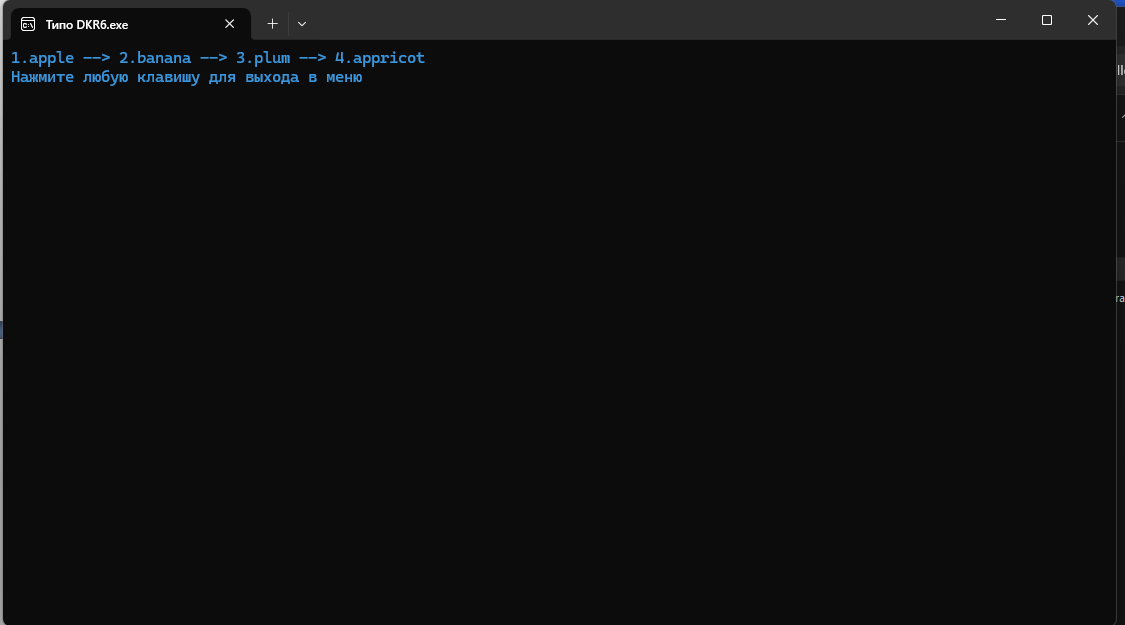


Рисунок 11 – просмотр элементов стека

**7.Вывод**

Работа завершена. В ходе работы была исследована такая динамическая структура данных как стек. В процессе работы были получены навыки по построению алгоритмов работы со стеком, а именно удалением элементов, вставке элементов и просмотре элементов стека. В результате работы была разработана программа, наглядно показывающая принципы работы стека, построенного на основе статической модели памяти.