Отчет по 4 Лабораторной работе

Предмет «Типы и структуры данных»

Вариант 16

Выполнено Маргаритой Мищенко

Студенткой группы ИУ7-35Б

Тема «Стек и операции над ним»

# Техническое задание

**Задача:**

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека, выполнить особое задание – определить, является ли строка палиндромом.

**Входные данные:**

Символы ASCII, вводимые для занесения в стек, либо номера команд.

**Выходные данные:**

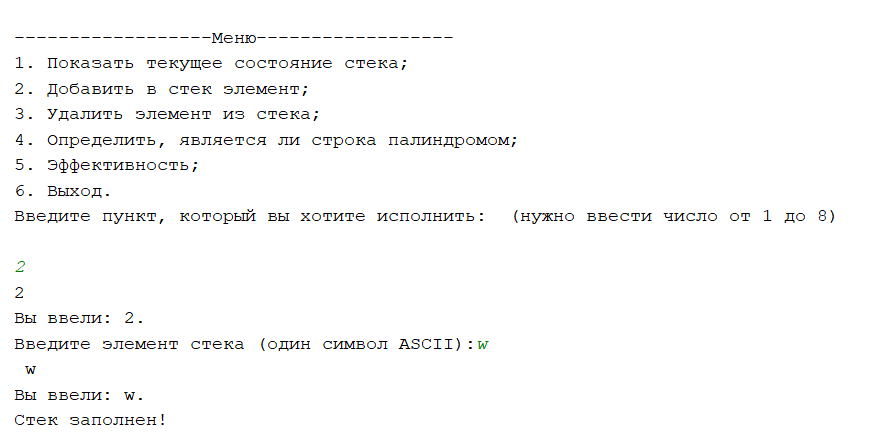
Элементы стека, список адресов, либо время выполнения, затрачиваемое на разное количество разных операциий, также области освобожденной памяти в виде односвязного списка.

**Функция программы:**

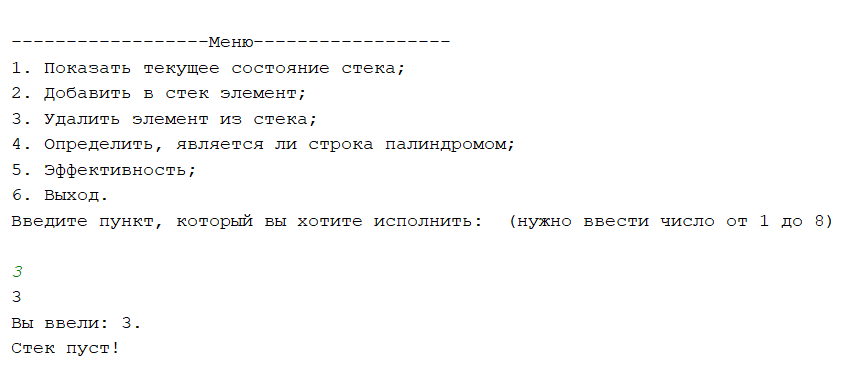
Реализация работы со стеком. Демонстрация распределения памяти при работе со стеком. Анализ времени и обьема памяти, которое требуется на добавление и удаление элементов из стека при реализации статическим массивом и стеком. Реализация специального задания.

# Аварийные ситуации

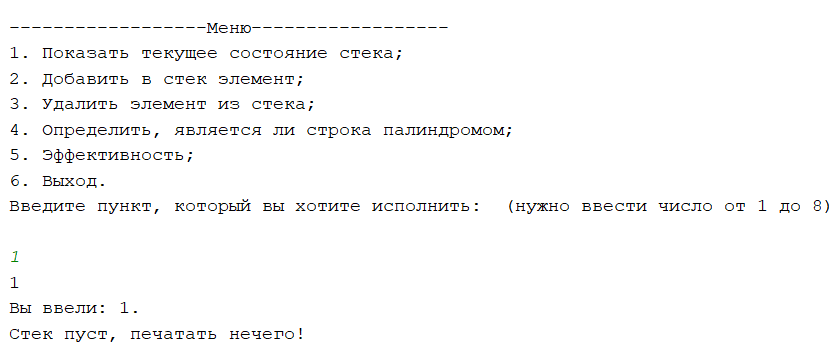
1. Попытка добавить элемент в заполненный стек.



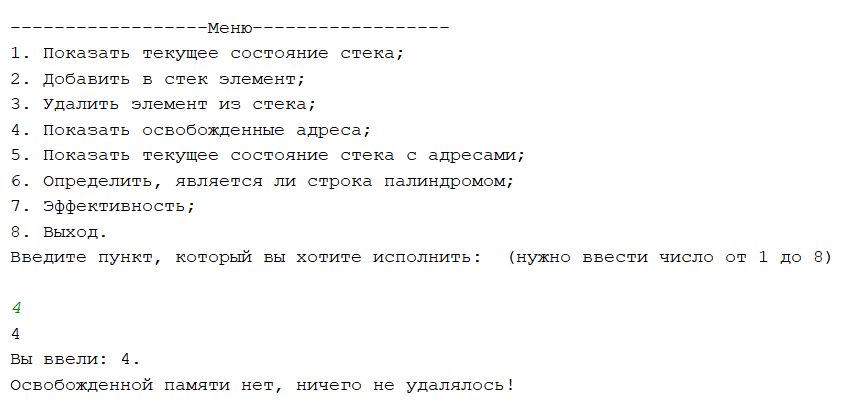
1. Попытка убрать элемент из пустого стека.



1. Попытка вывести элементы пустого стека



1. Попытка вывести освобожденную память, когда ничего не освобождалось



# Используемые структуры

# Для реализации структур в программе используется две константы: (это максимальный размер вектора и списка) #define **MAX\_SIZE\_VEC** 2 #define **MAX\_SIZE\_LIST** 2

# Для реализации вектора используется массив символов arr размера **MAX\_SIZE\_VEC**.

# При реализации односвязного списка для хранения стека используется следующая: (в ней символ стека и указатель на следующий символ)

**typedef struct** single\_linked\_list  
{  
 **char** symb;  
 **struct** single\_linked\_list \*next;  
}list\_t;

Приведенная ниже структура используется для хранения освобожденной памяти (памяти, в которой ранее хранились переменные из односвязного списка, но были удалены):  
  
**typedef struct** deleted\_memory  
{  
 **void** \*p;  
 **struct** deleted\_memory \*next;  
}del\_mem;

Алгоритм

Все алгоритмы в данной лабораторной работе являются стандартными алгоритмами работы со стеком.

Функции

Данная функция без входных параметров используя ранее обьявленные структуры и ниже приложенные функции считает эффективность операций со стеком на векторе и односвязном списке.

**void** efficience();

Данная функция определяет, является ли строка, записанная в стеке, палиндромом. List – односвязный список, list\_del – удаленная память, size – размер стека, size\_del – размер стека удаленной памяти, string[] – вектор.

**void** ispoly(list\_t \*list, del\_mem \*list\_del, **int** size, **int** size\_del, **char** string[]);

Функции для работы с вектором:

1. Печатает содержимое вектора:

**void** print\_arr(**char** \*string, **int** size);

1. Удаляет элемент вектора, возвращая его:

**char** arr\_pop(**char** \*\*string, **int** size);

1. Добавляет элемент в вектор:

**void** arr\_push(**char** \*\*string, **char** symb, **int** size);

Функции для работы с односвязным списком:

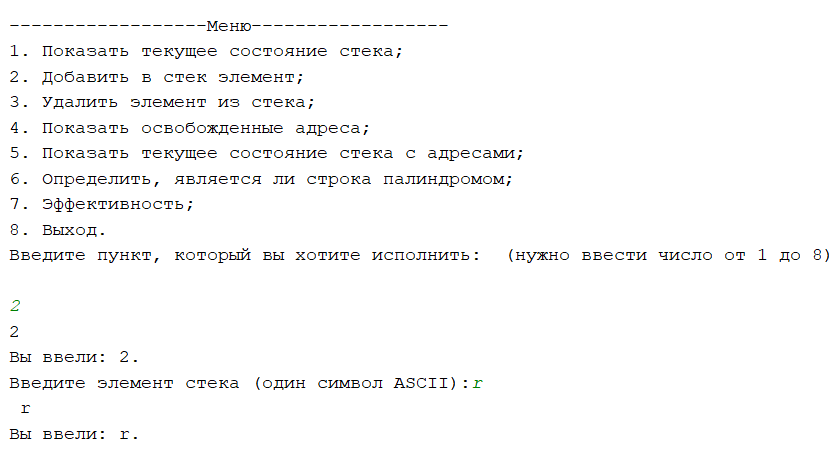
Функция добавления элемента, удаления элемента, печатания удаленной памяти, добавления удаленной памяти, печатании стека с хвоста, печатания стека с головы, печатания стека с адресами с головы.

**void** push(list\_t\*\* list, **char** elem, **int** \*size);  
**char** pop(list\_t\*\* list, del\_mem \*\*list\_del, **int** \*size, **int** \*size\_del);  
**void** print\_from\_head\_mem(**const** del\_mem \*list, **int** size);  
**void** push\_mem(del\_mem \*\*list, **void** \*p);  
**void** print\_from\_tail(**const** list\_t \*list);  
**void** print\_from\_head(**const** list\_t \*list, **int** size);  
**void** print\_from\_head\_with\_addresses(**const** list\_t \*list, **int** size);

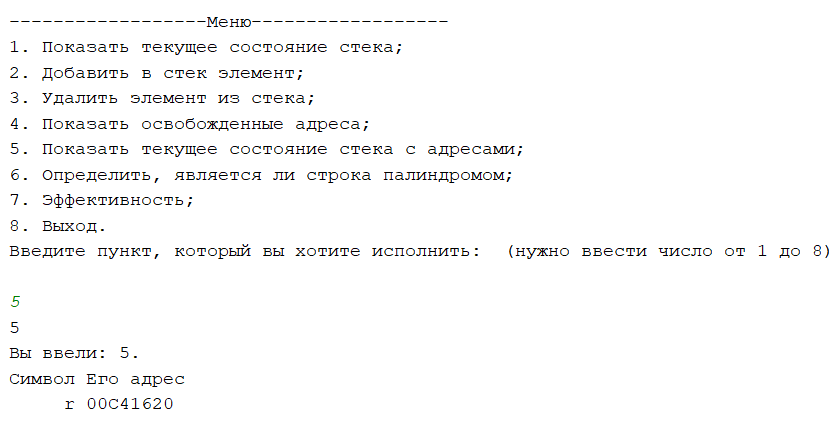
Вспомогательные функции: безопасный ввод целых чисел и символов.

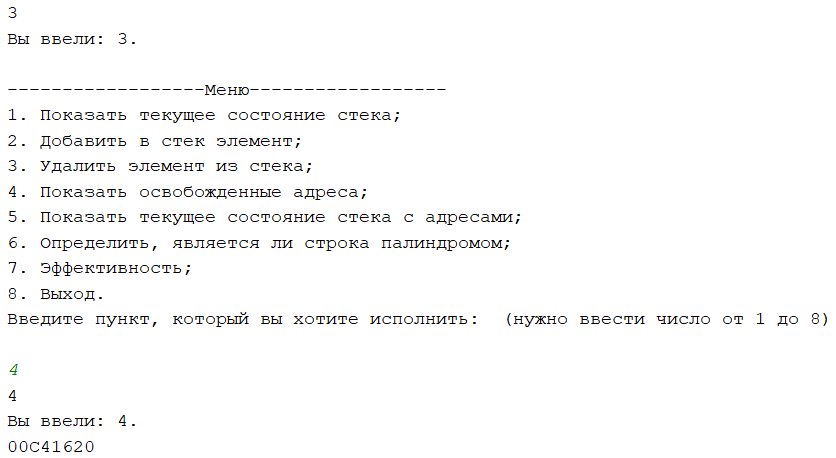
**void** scan\_char(**char** \*num);  
**void** scan\_int(**int** \*num);

# Тесты

Добавляем элемент в стек (односвязный список) 

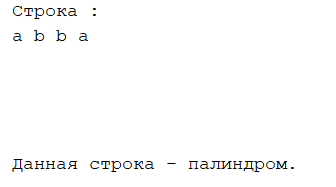
Выводим данные стека с адресами:



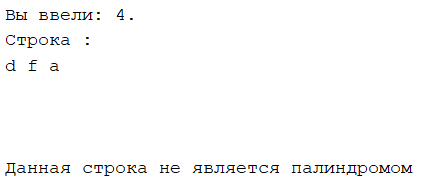
Удаляем элемент из стека, затем выводим освобожденные адреса 

Вводим строку и определяем, является ли она палиндромом.

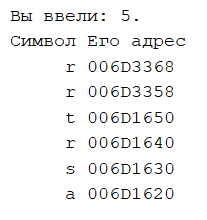
Палиндром:



Не палиндром:



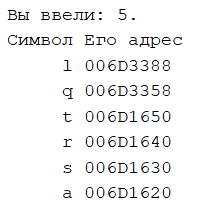
Демонстрация фрагментации памяти:

- стек.

Освобожденная память далее (удалим два элемента).



Далее введем два новых элемента и выведем стек:



Мы видим, что новые элементы занимают новые участки памяти, хотя могут занять и старые. Следовательно, память выделяется фрагментировано.

# Эффективность

# Эффективность того или иного способа задания стека проверяется на выполнении 10000, 100000 и 1000000 операций добавления и удаления элемента. Производится сравнение по времени и по памяти.

# 

# 

# Мы видим, что работа с односвязным списком происходит намного медленнее, для 100000 элементов в 9 раз, для 1000000 в более чем 12 раз. По памяти вектор тоже намного эффекктивнее, почти всегда в два раза (ведь для каждого элемента стека в односвязном списке мы храним еще и указатель).

# Вывод

Можно сделать вывод, что использование массива (или вектора) в целом более целесообразно. Этот способ быстрее и тратит в два раза меньше памяти. Однако в некоторых случаях можно использовать и односвязный список, он менее ограничен в размере, чем массив. Также в ходе лабораторной было выявлено, что при использовании односвязного списка происходит дефрагментация памяти (элементы стека располагаются в памяти не друг за другом).

# Ответы на вопросы

1. **Что такое стек?**

Стек – последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны. Функционирует по Last In First Out (LIFO).

1. **Каким образом и сколько памяти выделяется под хранение стека при различной его реализации?**

При реализаци стека списом, память выделяется динамически по мере добавления новых элементов; число элементов в стеке ограничено толь9ко количеством доступной ОП.

При реализации стека массивом, выделяется фиксированный участок памяти; в стеке не может быть больше заданного числа элементов. Добавлние нового элемента происходит путём смещения индекса вершины.

1. **Каким образом освобождается память при удалении элемента стека при различной реализации стека?**

При реализации списком память из-под элемента освобождается после его удаления.

При реализации массивом память из-под элемента не освобождается, происходит лишь изменение значения индекса вершины.

1. **Что происходит с элементами стека при его просмотре?**

В общем случае доступ есть только к вершине стека; при просмотре она удаляется из стека, а указатель смещается далее. Для отображения состояния стека требуется последовательно проходить по всем его элементам, не «снимая» их.

1. **Каким образом эффективнее реализовывать стек? От чего это зависит?**

Реализация стека массивом даёт огромный выигрыш во времени, поскольку не нужно каждый раз заново выделять и освобождать память. Тем не менее, в этом случае количество элементов в стеке жёстко ограничено – возможно либо переполнение стека, либо постоянный «излишек» памяти, отведённой под него.

Способ реализации напрямую зависит от условий решаемой задачи – нужно знать примерное число элементов, которые могут храниться в стеке; можно ли пренебрегать переполнением; ограничен ли объём памяти.