­­­­Лабораторная работа 3

Работа со стеком

Монахов Дмитрий ИУ7-34

Отчет (Вариант 13)

**Условие задачи:**

### Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации: получить представление о механизмах выделения и освобождения памяти при работе со стеком.

Используя стек, определить, является ли строка палиндромом.

Разработать программу работы со стеком, реализующую операции добавления и удаления элементов из стека и отображения текущего состояния стека. Реализовать стек: а) массивом; б) списком.

Все стандартные операции со стеком должны быть оформлены отдельными подпрограммами. В случае реализации стека в виде списка при отображении текущего состояния стека предусмотреть возможность просмотра адресов элементов стека и создания дополнительного собственного списка свободных областей (адресов освобождаемой памяти при удалении элемента, который можно реализовать как списком, так и массивом) с выводом его на экран.

**Техническое задание**

**Аварийные ситуации**

Некорректный ввод (нецифирные символы при вводе целочисленных значений)

Некорректная команда пользователя.

Удаление элемента из пустого стека.

Добавление элемента в заполненный стек.

Невозможность выделения динамической памяти при добавлении элемента в стек.

Ошибка чтения строки.

Попытка обработать пустую строку.

**Предусмотрено:**

Сообщение об ошибке при некорректном вводе.

Сообщение об ошибки удаления элемента из пустого стека.

Сообщение об ошибке добавления элемента в стек (переполнение или невозможность выделения памяти).

Сообщение при невозможности распознать команду.

Сообщение об ошибке при чтении строки.

Сообщение об ошибке при обработки пустой строки.

**Входные данные:**

Команда пользователя.

Данные для заполнения стека (ввод через консоль).

**Выходные данные:**

Сообщение об ошибке (если имеется).

Текущее состояние стеков.

Последний удаленный элемент.

Список адресов элементов стека.

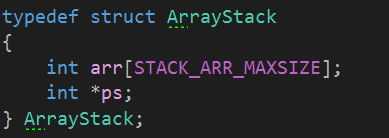
Список свободных областей.

Информация о времени выполнения операций добавления и удаления.

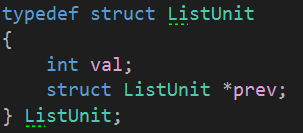
Информация о количестве используемой памяти.

**Структуры данных**

1. Представляет стек, основанный на статическом массиве.



1. Представляет элемент линейного списка, из которого формируется стек, основанный на списке.



**Функции**

Для стека-списка:

int liststack\_add(ListUnit \*\*stack, int a); // Добавляет элемент в списочный стек

int liststack\_remove(ListUnit \*\*stack, int \*result); // Удаляет элемент из списочного стека

int liststack\_is\_empty(ListUnit \*\*stack); // Проверяет, является ли списочный стек пустым

void liststack\_create(ListUnit\*\*); // Инициализирует пустой стек

Аналогичные функции для стека-массива:

int arrstack\_add(ArrayStack \*stack, int a);

int arrstack\_remove(ArrayStack \*stack, int \*result);

int arrstack\_is\_empty(ArrayStack \*stack);

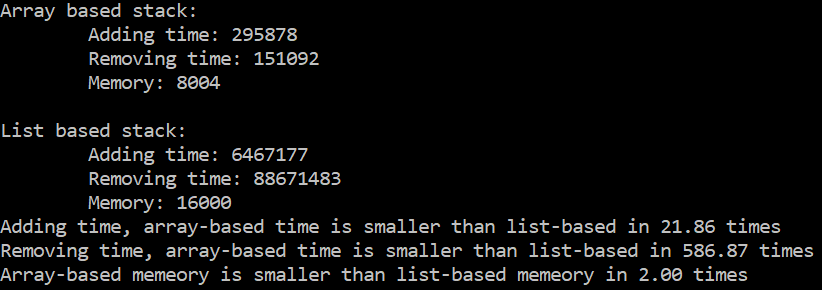
void arrstack\_create(ArrayStack \*stack);

**Тесты**

1. Некорректная команда пользователя
2. Некорректный ввод данных для добавления в стек
3. Добавление элемента в заполненный массив-стек.
4. Удаление элемента из пустых стеков.

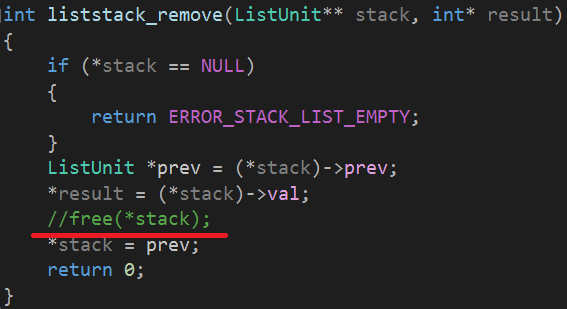
**Анализ**

**Сравнение видов реализации**

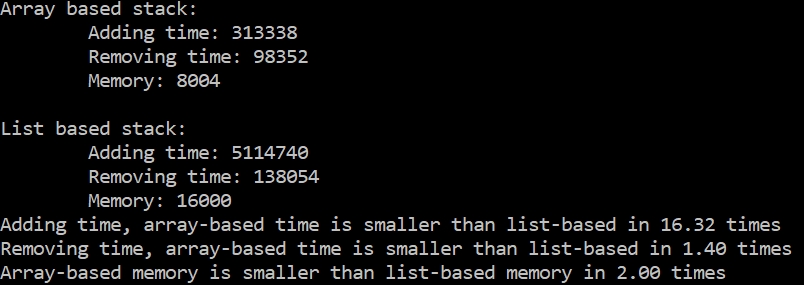


Из вывода программы видно, что по времени добавления и по времени удаления стек, основанный на статическом массиве выигрывает у стека, построенном на списке.

Однако если *ради эксперимента* отключить освобождение памяти в функции удаления элементов из стека-списка:



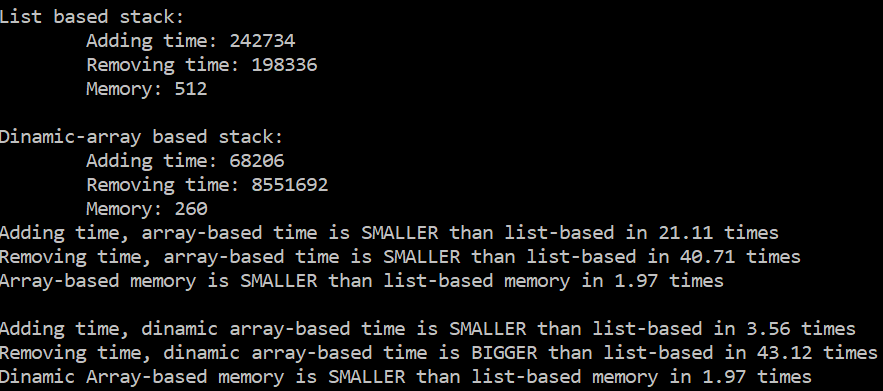
То время удаления элементов из списка будет лишь в 1.40 раза больше такового у списка-массива:



Отсюда можно сделать вывод, что если бы стек-массив был основан не на статическом, а на динамическом массиве, то временные показатели двух видов стеков были бы сравнимы. Из этого следует, что в данной ситуации выигрыш по времени стек-массива – в большей степени зависит от различия выделения памяти, и в меньшей – от вида стека.

Чтобы сравнить виды стеков в равных условиях, в проект был добавлен стек, основанный на динамическом массиве.

Для него и для стека-списка были произведены замеры времени и памяти.



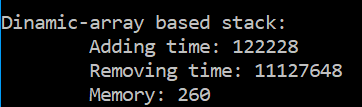
Как видно, по памяти динамический и статический стеки-массивы одинаковы.

Однако наблюдаются изменения во времени. Так, время добавления у динамического массива в 3.56 раза меньше чем у стека-списка. Время удаления же наоборот, у стека-списка время меньше в 43.12 раза чем у динамического стека-массива. Как было выяснено выше, значительная часть времени уходит на динамическое выделение памяти. Здесь же разница во времени освобождения вызвана разницей в скорости работы функций free() (в стеке-списке) и realloc() (в динамическом стеке-массиве).

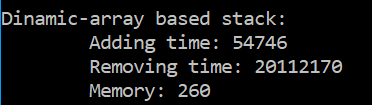
Чтобы сократить время выполнения была создана конструкция, позволяющая реже вызывать ф-ию realloc(). Так, realloc() будет выделять память ну под один элемент, а сразу под несколько. Если размер блока равен B элементов, то realloc() будет вызван N / B раз, где N – кол-во элементов, которые требуется добавить в стек (удалить из стека).

Замеры для N = 64 элемента.

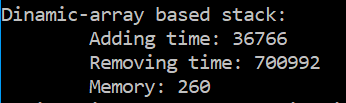
Без блочного выделения (или B = 1):



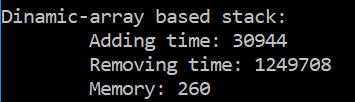
Для B = 4



Для B = 8



Для B = 16



Для B = 32

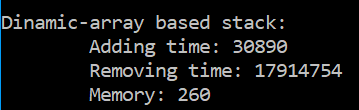


График зависимости времени добавления N = 64 элементов от количества блоков:

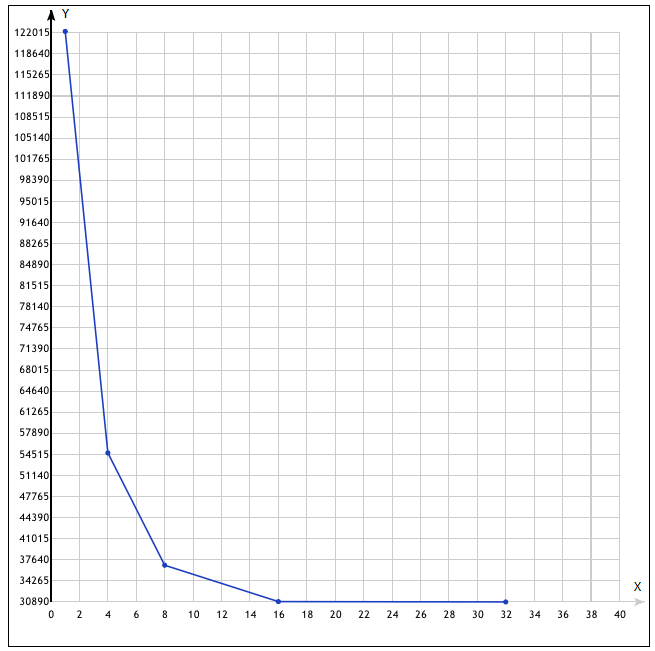
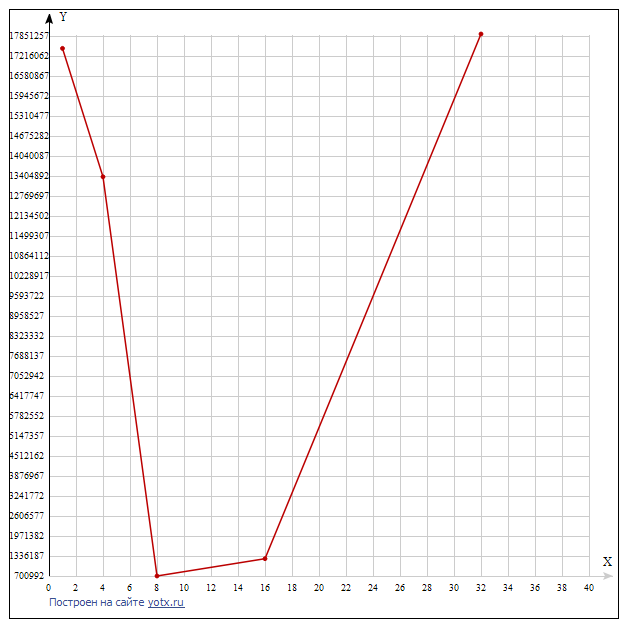
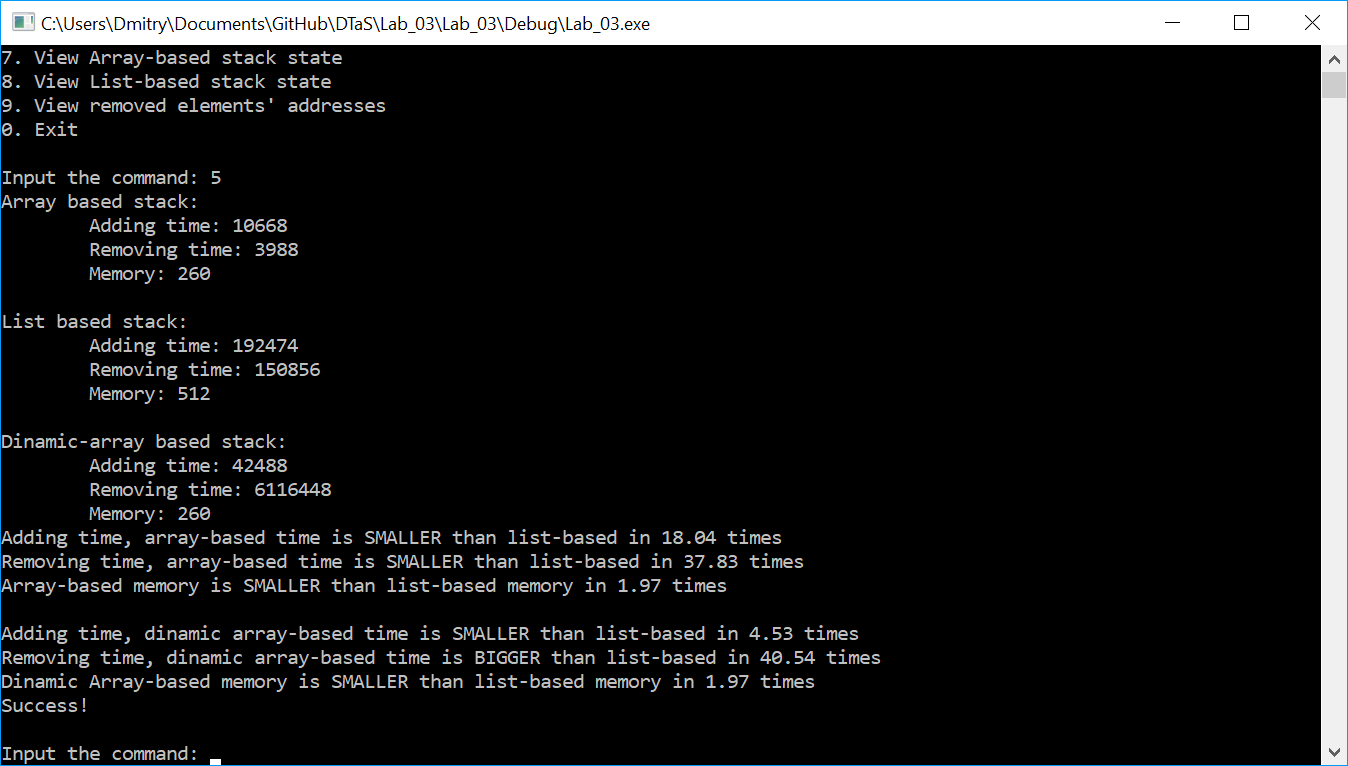


График зависимости времени удаления N = 64 элементов от количества блоков:



Видно, что наименьшее суммарное время удаления и добавления получается при B = 8.

Проведем сравнение по времени добавления и удаления стек-список и динамические стек-массив:



Таким образом, в операции добавления новых элементов при хорошо выбранном размере блока динамический стек-массив выигрывает у стека-списка примерно в 4.5 раза

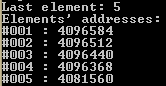
По времени удаления всех элементов динамический стек-массив проигрывает стеку-списку в сорок раз.

В общем итоге оказывается, что статический список-массив – самый эффективный по времени вид списка из трех представленных, однако он единственный имеет недостаток в виде ограничения на количество элементов (в динамических вариантах это количество ограничено оперативной памятью компьютера).

**Фрагментация памяти**

Чтобы посмотреть, фрагментируется ли память при работе со стеком, можно заполнить стек некими значениями, затем удалить и перезаписать несколько элементов.

Выведем адреса элементов стека (здесь обратная нумерация, последний добавленный элемент имеет индекс 1):

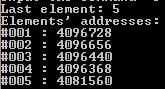


Видно, что смещение между элементами 1-2, 2-3, 3-4 одинаково и равно 72.

Размер одного элемента списка равен

Теперь удалим два элемента, а затем добавим такое же количество:

 - после удаления.

 - после добавления.

Выведем список освобожденных адресов:



Видно, что эти адреса не совпадают с адресами добавленных элементов. Данные записались в смещенную от исходной область памяти, значит наблюдается фрагментация динамической памяти.

**Контрольные вопросы**

1. Что такое стек?

Стек – это последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны – с его вершины. Стек функционирует по принципу: последним пришел – первым ушел, Last In – First Out (LIFO).

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

Если стек реализован в виде статического или динамического массива (вектора), то для его хранения обычно отводится непрерывная область памяти ограниченного размера, имеющая нижнюю и верхнюю границу. Перед началом работы указатель стека PS находится ниже левой (нижней) границы массива.

При реализации в виде списка до начала работы указатель стека показывает на нулевой, физически отсутствующий адрес (т. е. указатель - пустой). При включении элемента в стек сначала происходит выделение области памяти, адрес которой записывается в указатель стека, а затем по значению этого указателя в стек помещается информация.