Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3 ПО ДИСЦИПЛИНЕ:
ТИПЫ И СТРУКТУРЫ ДАННЫХ

ОБРАБОТКА РАЗРЕЖЕННЫХ МАТРИЦ

Студент **Попов Ю.А.**

Группа **ИУ7-32Б**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Попов Ю.А.**

Преподаватель _____ **Силантьева А.В.**

2024

Оглавление

Описание условия задачи

Реализовать операции работы со стеком, который представлен в виде статического массива и в виде односвязного списка, оценить преимущества и недостатки каждой реализации, получить представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных. Используя стек, определить, является ли строка палиндромом

Описание технического задания

Входные данные:

Пользователь вводит целое число-команду от 0 до 4. Программа выполняет операции, в зависимости от выбранной опции. Чтобы корректно завершить программу, необходимо воспользоваться соответствующей командой.

Выходные данные:

В зависимости от выбранного действия, пользователь получает разные выходные данные. Общими остаются уведомления об успешности или ошибки выполнения действия, и вывод информации на экран.

Действие программы:

Программа работает до тех пор, пока пользователь не введет число 0. Программное обеспечение позволяет ввести команды от 0 до 4, в случае если команда выходит за этот диапазон, или не подходит по типу данных, выводится соответствующее сообщение, и программа завершает работу. Разработанная программа позволяет проверить является ли строка

палиндромом и протестировать реализацию стека на массиве и на связном списке. Пользователю доступны следующие операции:

1) Вывести на экран весь стек;

2) добавить элемент в стек;

3) удалить элемент из стека;

Обращение к программе:

Программа запускается через терминал, в командной строке, с помощью команды ./app.exe. После запуска. пользователю будет доступно меню программы.

Аварийные ситуации:

1. ERR_OPERATION – Неверный выбор операции.
2. ERR_STACK_OVERFLOW – Переполнение стека, происходит при попытке добавления элемента в “заполненный” стек.
3. ERR_STACK_EMPTY – Происходит при попытке удаления из пустого стека.
4. ERR_STRING – Ошибка при вводе строки.
5. ERR_STRING_OVERFLOW - Переполнение строки.
6. ERR_MEMORY_ALLOCATION - Ошибка при выделении памяти

Во всех нештатных ситуациях, программа уведомляет о месте, где произошла ошибка.

Описание структуры данных

Структура для стека на основе массива

top - индекс вершины стека

data - массив данных, хранящихся в стеке

values - массив указателей на строки матриц.

```
typedef struct
{
    int top;
    char data[MAX_STACK_SIZE];
} static_stack_t;
```

листинг 1. Структура static_stack_t

Структура для хранения информации о “ноде”

next - указатель на следующую ноду

data - указатель на данные, хранящихся в ноде

```
typedef struct __node_t
{
    struct __node_t *next;
    void *data;
} node_t;
```

листинг 2. Структура node_t

Структура для стека на основе связного списка

count - количество элементов в стеке

top - указатель на ноду - вершину стека

```
typedef struct
{
    size_t count;
    node_t *top;
} list_stack_t;
```

листинг 2. Структура list_stack_t

Описание алгоритма

1. После запуска программы выводит приглашение к вводу и меню.
2. Программа принимает номер операции, и запускает соответствующие действия в соответствии с номер команды
3. Добавление в стек на основе списка происходит путем создания новой ноды и присваиванием указателя на нее в поле next.

4. Удаление из стека на основе списка происходит путем замены текущей ноды на следующую. После удаление элемента, память освобождается.
5. Если пользователь вводит 0, то программа завершается.

Описание основных функций

```
void static_stack_init(static_stack_t *stack);
```
Инициализация статического стека

Входные параметры: указатель на статический стек

Выходные параметры: проинициализированный стек

```
int static_stack_push(static_stack_t *stack, char element);
```
функция добавляет элемент в стек на массиве

Входные параметры: указатель на стек, элемент для добавления

Выходные параметры: стек, в который был добавлен переданный элемент

Возвращаемый результат: код возврата.

```
char static_stack_pop(static_stack_t *stack, int *rc);
```
Функция удаляет верхний элемент из стека.

Входные параметры: указатель на матрицу, указатель код возврата

Выходные параметры: удаленный элемент, код возврата

Возвращаемый результат: удаленный элемент

```
void static_stack_print(const static_stack_t stack);
```
Функция выводит стек на экран

Входные параметры: стек

Выходные параметры: Вывод стека на экран

```
int static_stack_is_palindrome(char *string);
```
Функция проверяет, является ли строка палиндромом, с помощью стека на статическом массиве

Входные параметры: строка

Возвращаемый результат: 0 - если строка не палиндром, иначе - 1

```
void list_stack_init(list_stack_t *stack);
```

Инициализация статического стека

Входные параметры: указатель на статический стек

Выходные параметры: проинициализированный стек

```
int list_stack_push(list_stack_t *stack, const void *element, size_t size);
```

функция добавляет элемент в стек на массиве

Входные параметры: указатель на стек, элемент для добавления, размер памяти, занимаемой элементом

Выходные параметры: стек, в который был добавлен переданный элемент

Возвращаемый результат: код возврата.

```
void list_stack_print_char(const list_stack_t stack);
```

Функция выводит стек, состоящий из символов, на экран

Входные параметры: стек

Выходные параметры: Вывод стека на экран

```
void list_stack_free(list_stack_t *stack);
```

Функция освобождает память, выделенную под стек

Входные параметры: стек

```
int list_stack_pop(list_stack_t *stack, void *element, size_t size);
```

Функция удаляет верхний элемент стека

Входные параметры: указатель на стек, указатель на элемент, размер элемента

Выходные параметры: стек, с удаленным элементом.

Возвращаемый результат: код возврата

```
int list_stack_is_palindrome(char *string);
```

Функция проверяет, является ли строка палиндромом, с помощью стека на списке

Входные параметры: строка

Возвращаемый результат: 0 - если строка не палиндром, иначе - 1

Набор тестов

№	Описание теста	Пользовательский ввод	Вывод
1	Некорректный ввод операции	abc	Ошибка при выборе операции.
2	Некорректный ввод операции. Выход за диапазон	10000	Ошибка при выборе операции.
3	Некорректный ввод операции. Отрицательное число	-100	Ошибка при выборе операции
4	Строка является палиндромом	tet	Строка палиндром
5	Строка не является палиндромом	test	Строка палиндром
6	Пустая строка		Ошибка при вводе строки
7	Удаление из пустого стека		Ошибка, стек пустой
8	Переполнение стека		Переполнение стека
9	Вывод пустого стека		Ошибка, стек пустой

Оценка эффективности

Временные замеры (нс)

Временные замеры осуществлялись с помощью библиотеки time.h. Замеры скорости выполнения проводились для функции, определяющей

является ли строка палиндромом. Выбор пал на эту функцию, потому что ее реализация одинакова для обоих методов, а также в реализации используется как добавление, так и удаление из стека.

Размер	Стек на статическом массиве	Стек на связном списке	Отношение скорости работы
10	1182,86	286,15	0,24
50	1148,95	1391,51	1,21
100	1390,05	2764,07	1,99
300	2029,98	8088,35	3,98
500	2589,48	13874,57	5,36
750	3193,49	21082,54	6,60
1000	4244,87	28051,26	6,61
1500	5386,98	41403,66	7,69
2000	7434,36	56455,05	7,59

Таблица 1. Сравнение времени работы исследуемых алгоритмов.

Объем занимаемой памяти (байты)

Объем, занимаемой памяти стеками в зависимости от их размера

Размер	Стек на статическом массиве	Стек на связном списке	Отношение скорости работы
10	100004	160	0,00
400	100004	6400	0,06
1000	100004	16000	0,16
1600	100004	25600	0,26

2000	100004	32000	0,32
2600	100004	41600	0,42
3400	100004	54400	0,54
4000	100004	64000	0,64
4800	100004	76800	0,77
6600	100004	105600	1,06
6800	100004	108800	1,09
7600	100004	121600	1,22
8600	100004	137600	1,38
10000	100004	160000	1,60

Таблица 2. Оценка занимаемой памяти, относительно размера.

Несмотря на то что обе операции имеют сложность O(N), операции для стека на списке выполняются дольше из-за обращения к функции для работы с выделением и освобождением памяти. По таблице 1 видно, что стек на массиве оказывается выгоднее всегда, кроме случая, когда размера стека не превышает 20 элементов, это может являться погрешностью. Однако, стек на основе списка выигрывает по памяти до 6500 элементов. Ниже приведены графики сравнения методов по скорости и по памяти, по ним видно, что оба алгоритма обладают линейной сложностью.

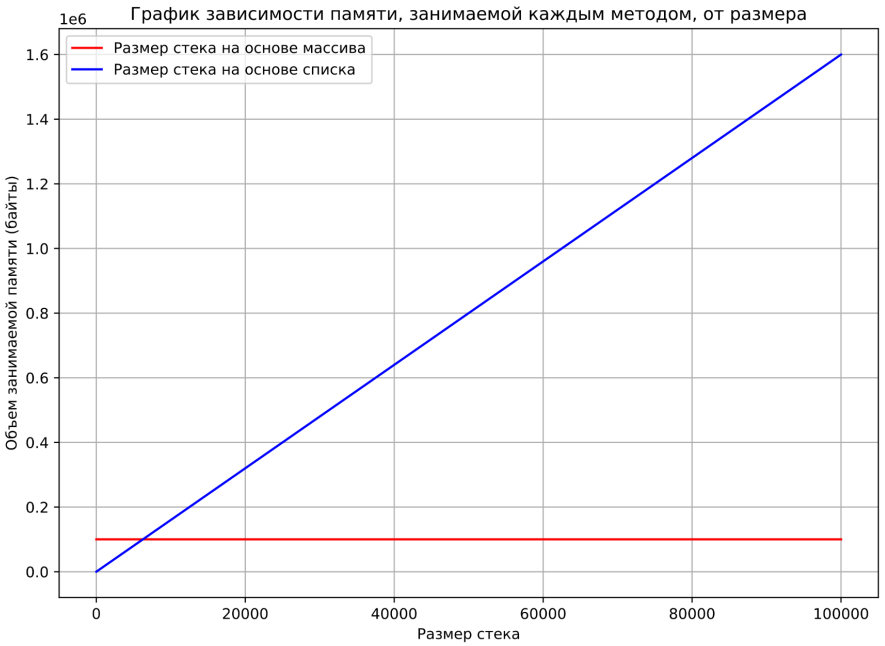


График зависимости памяти, занимаемой каждым методом, от размера
1e6
Размер стека на основе массива
Размер стека на основе списка
Объем занимаемой памяти (байты)
Размер стека
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
1.6
0
20000
40000
60000
80000
100000

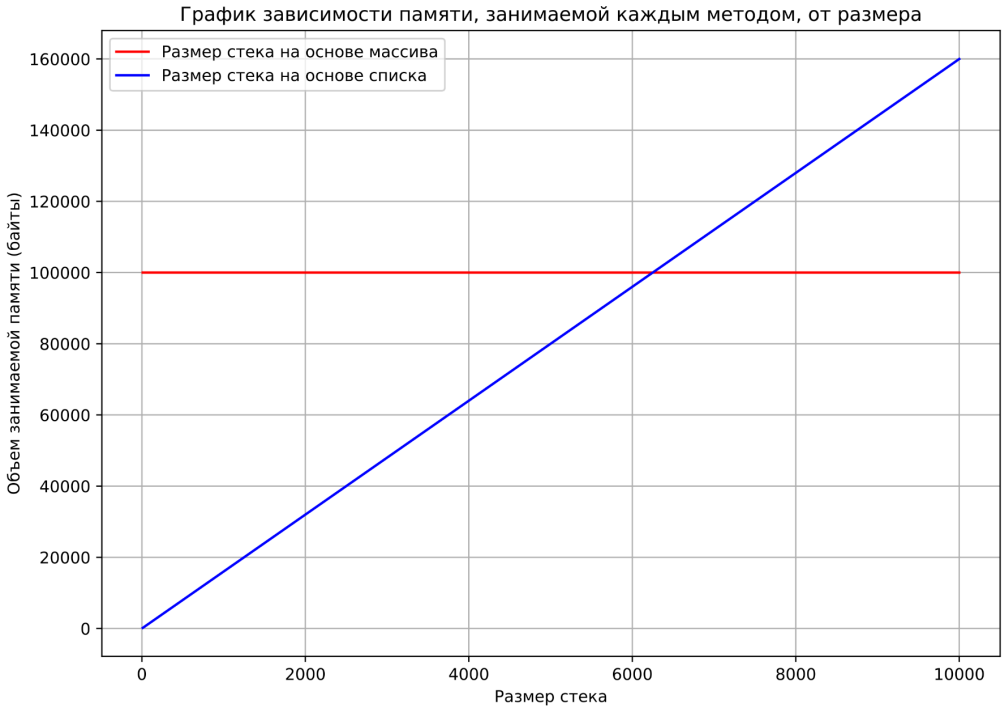


График зависимости памяти, занимаемой каждым методом, от размера
Размер стека на основе массива
Размер стека на основе списка
Объем занимаемой памяти (байты)
Размер стека
0
20000
40000
60000
80000
100000
120000
140000
160000
0
2000
4000
6000
8000
10000

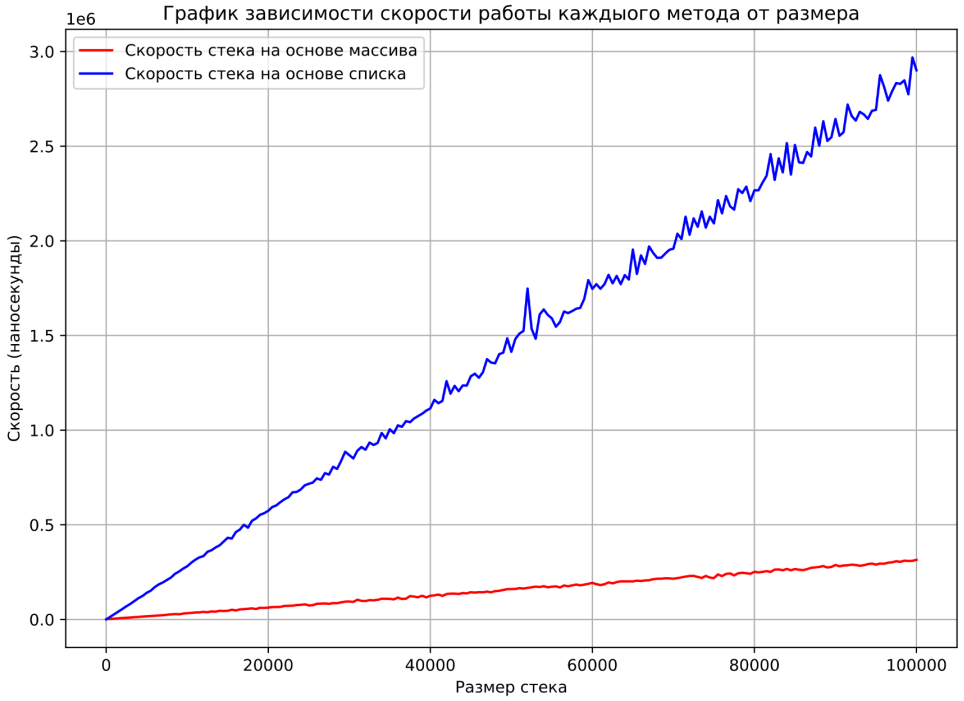


График зависимости скорости работы каждыого метода от размера
1e6
Скорость стека на основе массива
Скорость стека на основе списка
Скорость (наносекунды)
Размер стека
0.0
0.5
1.0
1.5
2.0
2.5
3.0
0
20000
40000
60000
80000
100000

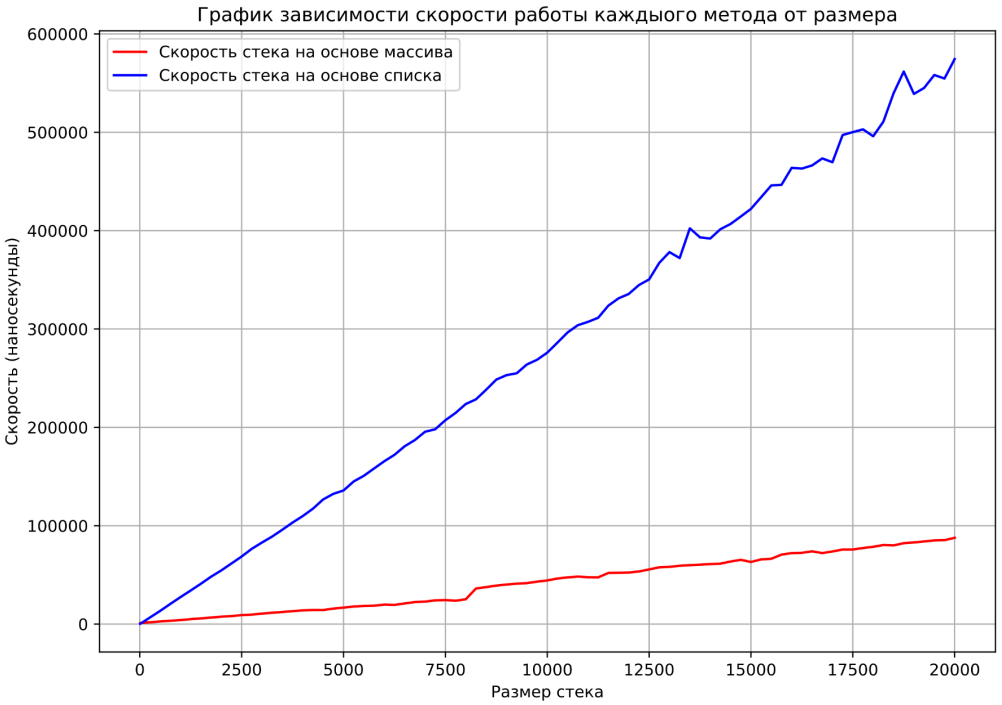
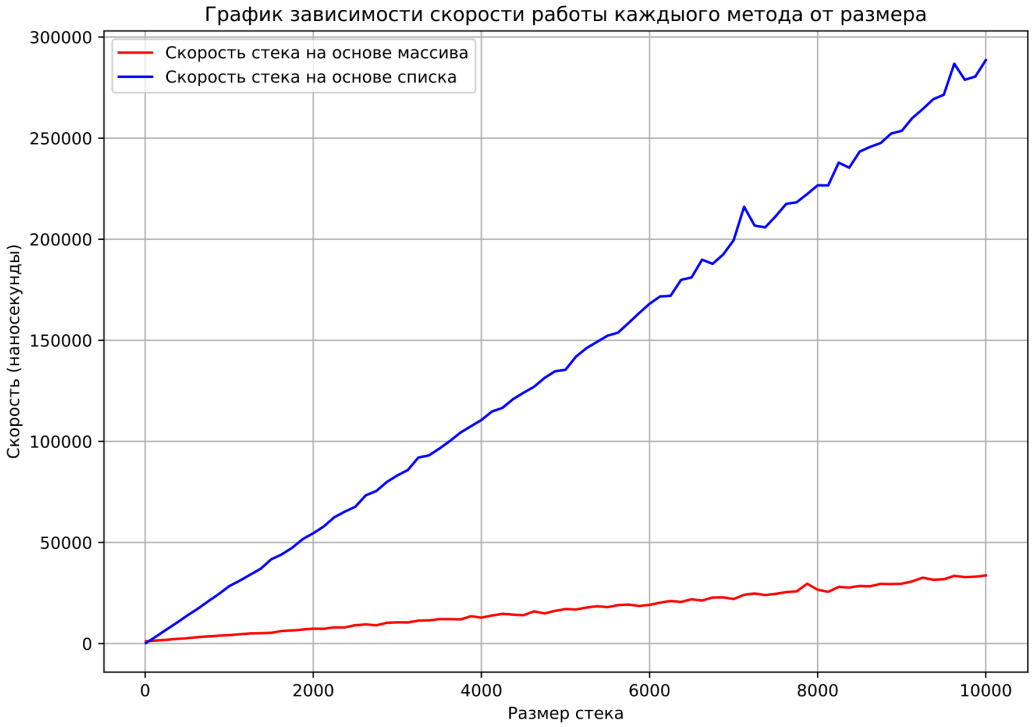


График зависимости скорости работы каждыого метода от размера
Скорость стека на основе массива
Скорость стека на основе списка
Скорость (наносекунды)
Размер стека
0
100000
200000
300000
400000
500000
600000
0
2500
5000
7500
10000
12500
15000
17500
20000



Ответы на контрольные вопросы

1. Что такое стек?

Стек - структура данных. работающая по принципу LIFO (Last In First Out) “первый пришел, последний ушел”. Это означает, что на уровне интерфейса нам доступен только последний элемент стека.

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

При реализации стека статическим массивом выделяется (N * 1 + 4), где N - это максимальное количество элементов. При реализации стека листом одна нода и элемент стека занимается 16 байт. В итоге лист занимает (n + 1) * 16 байт, где n - количество элементов. Память выделяется под каждый элемент

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

При реализации стека на основе статического массива, работа с памятью возлагается на систему. При реализации на основе списка сначала освобождается память, выделенная под данные, а потом освобождается нода.

```
free(node→data);
free(node);
```

4. Что происходит с элементами стека при его просмотре?

Для просмотра стека, в функцию передается его копия. Далее цикл проходимся по копии списка, которая в конечном итоге будет “испорчена”. Исходный стек останется нетронутым.

5. Каким образом эффективнее реализовывать стек? От чего это зависит?

Стек на массиве оказывается выгоднее всегда, кроме случая, когда размера стека не превышает 20 элементов, это может являться погрешностью. Однако, стек на основе списка выигрывает по памяти до 6500 элементов. Таким образом, оказалось, что эффективнее реализовывать стек на массиве, особенно на больших объемах данных.

ВЫВОД

Стек на массиве оказывается выгоднее стека на связном списке. Единственный минус стека на статическом массиве это инициализированние памяти на статичное число элементов. Таким образом расходуется не эффективно. Если сделать стек на динамическом массиве, то проблема с избытком проинициализированной памяти решится сама собой.