


Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

	«Московский государственный технический университет имени Н.Э. Баумана» (МГТУ им. Н.Э. Баумана)
---	--

ФАКУЛЬТЕТ _____ ИУ _____

КАФЕДРА _____ ИУ7 _____

Отчет

по лабораторной работе № 2

Дисциплина: Типы и Структуры Данных

Название лабораторной работы: Обработка больших чисел

Студент гр. **ИУ7-34Б Малышев Илья Николаевич**

(Подпись, дата) (И.О. Фамилия)

Преподаватель **Силантьева Александра Васильевна**

(Подпись, дата) (И.О. Фамилия)

Москва, 2021

Цель работы

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.

Условия задачи

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека. Реализовать стек: а) массивом; б) списком. Все стандартные операции со стеком должны быть оформлены подпрограммами. При реализации стека списком в вывод текущего состояния стека добавить просмотр адресов элементов стека и создать свой список или массив свободных областей (адресов освобождаемых элементов) с выводом его на экран.

Элементами стека являются слова. Распечатайте слова в обратном порядке в перевернутом виде.

Техническое задание

Входные данные:

- Целое число, которое обозначает действие, которое должна выполнить программа (смотри “Задача программы”).
- Запрашиваемые данные, необходимые для выполнения некоторых действий;
- Файл с статистикой работы программы.

Выходные данные:

- Состояние стека в формате: <адрес> : строка;
- Таблица с результатами измерений времени и памяти при разных объемах стека;
- Результат моделируемой операции.

Задача программы:

Программа позволяет выполнить перечисленные ниже операции:

1. [СПИСОК] Добавить элемент в стек (push);
2. [СПИСОК] Удалить элемент из стека (pop);
3. [СПИСОК] Вывести состояние стека;
4. [СПИСОК] Вывести список освобожденных областей;
5. [СПИСОК] Очистить стек;
6. [СПИСОК] Сбросить список освобожденных областей;
7. [СПИСОК] Распечатать слова в обратном порядке в перевернутом виде;
8. [МАССИВ] Добавить элемент в стек (push);
9. [МАССИВ] Удалить элемент из стека (pop);
10. [МАССИВ] Вывести состояние стека;
11. [МАССИВ] Вывести список освобожденных областей;
12. [МАССИВ] Сбросить стек;
13. [МАССИВ] Сбросить список освобожденных областей;
14. [МАССИВ] Распечатать слова в обратном порядке в перевернутом виде;
15. Сгенерировать статистику по одной размерности;
16. Вывести готовую статистику из файла ('statistics.txt');
0. Выход.

Способ обращения к программе:

Программа является консольной. После запуска исполняемого файла из консоли, программа выведет справку и приглашение на ввод.

Аварийные ситуации:

- Некорректный ввод номера команды;
- Неверный ввод строки;
- Отсутствие требуемых для работы программы текстовых файлов.

Структуры данных:

string_t - структура, представляющая динамическую строку. В ней хранится указатель на начало строки и длину области памяти, выделенной под строку.

```
typedef struct
{
    char *string;
    size_t allocated;
} string_t;
```

array_stack_t - структура, представляющая стек, выполненный на базе динамического массива. В ней хранится указатель на первый элемент массива, указатель на элемент следующий за последним в массиве, а также указатель на элемент массива, в который будет происходить запись при следующем вызове команды push.

```
typedef struct
{
    void **start_pointer;
    void **end_pointer;
    void **current_position_pointer;
} array_stack_t;
```

stack_note_t - структура, представляющая узел связанного списка. В ней хранится указатель на информацию и указатель на предшествующий узел.

```
typedef struct
{
    void *data;
    void *previous;
} stack_note_t;
```

stack_t - структура, представляющая “упаковку” для связанного списка. В ней хранится указатель на последний добавленный узел и общее количество добавленных узлов.

```
typedef struct
{
    size_t stack_len;
    stack_note_t *current_note;
} stack_t;
```

line_t - тип, представляющий статическую строку.

```
#define LINE_LEN 1024
typedef char line_t[LINE_LEN];
```

list_t - структура, представляющая динамический массив. Хранит указатель на начало массива с данными, количество записанных данных и длину выделенной области по данные.

```
typedef struct
{
    void **deallocated_memory;
    size_t len;
    size_t allocated;
} list_t;
```

base_t - структура, представляющая “хранилище” для некоторых структур, необходимых для работы программы.

```
typedef struct
{
    stack_t *stack;
    list_t *stack_deallocated_memory;
    array_stack_t *array_stack;
    list_t *array_stack_deallocated_memory;
} base_t;
```

Тесты

#	Ошибка	Пример
1	Неправильный ввод команды	q
2	Неправильный ввод команды	20
3	Неправильный ввод строки	123
4	Попытка работы со стеком (массивом), которые еще не инициализирован	без 8 9
5	Отсутствие необходимых для работы программы системных файлов	statistics.txt

Оценка эффективности

Измерения эффективности способов сложения будут производиться в единицах измерения – тактах процессора.

* - Смотри условие задачи, последний абзац.

Стек в виде связного списка				
Количество элементов в стеке	Среднее время добавления элемента (в тактах)	Среднее время удаления элемента (в тактах)	Среднее время на решение поставленной задачи* (в тактах)	Затраченная память на хранение структур (не учитывая понятия под информацией) (в байтах)
1	214	188	4265	32
5	101	83	14301	96
10	120	92	29891	176
50	154	123	144712	816
100	173	124	265907	1616
500	147	86	1274848	8016
1000	144	123	2498019	16016

Стек в виде массива				
Количество элементов в стеке	Среднее время добавления элемента (в тактах)	Среднее время удаления элемента (в тактах)	Среднее время на решение поставленной задачи* (в тактах)	Затраченная память на хранение структур (не учитывая понятие под информацией) (в байтах)
1	18	19	3050	32
5	17	19	13024	64
10	20	21	29156	104
50	26	28	144093	424
100	26	27	264521	824
500	18	19	1263726	4024
1000	26	28	2499874	8024

Контрольные вопросы

1. Что такое стек?

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент (верхний элемент). На стек действует правило LIFO — последним пришел, первым вышел.

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

При хранении стека с помощью списка, память всегда выделяется в куче. При хранении с помощью массива, память выделяется либо в куче, либо на стеке (в зависимости от того, динамический массив или статический). Для каждого элемента стека, реализованного списком, выделяется на 4 или 8 байт (на большинстве современных ПК) больше, чем для элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя (4 или 8 байт) зависит от архитектуры.

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

При хранении стека связанным списком, верхний элемент удаляется освобождением памяти для него и смещением указателя, указывающего на начало стека. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека.

4. Что происходит с элементами стека при его просмотре?

Элементы стека уничтожаются, так как каждый раз достается верхний элемент стека.

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

Реализовывать стек эффективнее с помощью массива. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти (в классическом случае). Вариант хранения

списка может выигрывать только в том случае, если стек реализован статическим массивом. В этом случае, память для списка ограничена размером оперативной памяти (так как память выделяется в куче), а память для статического массива ограничена размером стека.

Вывод

1. Как видно из таблиц оценки эффективности программы:
 1. В общем случае стек, выполненный связанным списком, требует в два раза больше памяти, чем стек - массив.
 2. Стек-массив и стек-список позволяют выполнять операцию добавления и удаления элемента с относительно стабильными показателями времязатрат. Можно говорить, что время добавления и удаления элемента для стека-списка больше времени добавления и удаления элемента для стека-массива примерно на один порядок. Также можно отметить, что время добавления и время удаления элемента для одного способа организации стека отличаются не более чем на один порядок.
 3. Последнее наблюдение - время, потраченное на решение поставленной задачи, для двух способов реализации стека, для разных объемов стеков отличается менее чем на порядок, что говорит о том, что в сравнении со временем обработки данных, время на работу со стеком пренебрежимо мало.
2. На основании вышесказанного и более детального рассмотрения таблиц оценки эффективности, можно также сказать:
 1. Стек-список проигрывает стеку-массиву в подавляющем большинстве временных тестов и тестов по памяти (по времени - на один порядок, по памяти в два раза).

2. Стек-список практически не уступает стеку-массиву по стабильности работы (см. время добавления и удаления элемента).
3. При решении практических задач, отставание стека-списка от стека-массива пренебрежимо мало.

Учитывая тот факт, что стек-список, позволяет решить проблему фрагментации памяти, можно сделать вывод, а также выводы 2.1, 2.2, 2.3 можно утверждать, что использование стека-списка рекомендовано всегда, кроме тех случаев, когда временные показатели играют первостепенную задачу, или в условиях дефицита памяти.