|  |  |
| --- | --- |
|  | **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

**ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4**

Название предмета: Типы и структуры данных

Студент: Баринов Никита

Группа: ИУ7-31Б

*2021г.*

**Описание 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**

***Описание исходных данных***

Программа постоянно запрашивает команду на выполнение того или иного действия.

Пункты меню:

0 - выход из программы

1 - добавить слово

2 - удалить слово

3 - вывести в обратном порядке

4 - вывести текущее состояние стеков

При добавлении элемента требуется ввести слово (без пробелов) размером до 30 символов.

***Описание результата программы***

При удалении элемента из стека выводится его значение и адрес освобождаемой памяти.

При выборе 3 пункта меню выводятся только перевернутые слова в обратном порядке через пробел из двух стеков.

Также выведется время выполнения для каждого из способов реализации стека и затраченная при этом память.

***Описание задачи, реализуемой программой***

Программа выполняет обработку стека, хранящегося в виде динамического массива или линейного односвязного списка, а также анализ скорости и занимаемой памяти при данных способах хранения и обработки стека.

При выполнении поставленной задачи происходит полное очищение стеков методом (pop), само определение стека (первый зашел – последний ушел) позволяет естественным образом вывести слова в обратном порядке.

Вывод слова наоборот производится дополнительной подпрограммой.

***Способ обращения к программе***

Способ обращения к программе — консольный.

Компиляция: make app.exe

Имя программы: app.exe

Как запустить: ./app.exe

Дальнейшие инструкции будут выведены после запуска.

***Описание возможных аварийных ситуаций и ошибок пользователя***

* Ошибка выделения памяти для хранения
* Попытка добавить элемент в заполненный стек
* Попытка удалить элемент из пустого стека
* Неверный пункт меню
* Попытка записать пустое слово
* Попытка записать слишком большое слово
* Попытка записать слово, содержащее пробелы

#define CHOICE\_ERROR 1

#define ALLOCATE\_ERROR 2

#define STACK\_OVERFLOW\_ERROR 3

#define EMPTY\_STACK\_ERROR 4

Во всех указанных случаях программа вернет сообщение об ошибке и вернется в режим меню или режим ввода слова, исключая ошибку памяти (кроме ошибки выделения памяти).

**Описание внутренних структур данных**

**1) Стек, основанный на массиве**

typedef struct Arr\_Stack Arr\_Stack;

struct Arr\_Stack

{

void \*\*Stack\_Start; // Указатель на начало массива

int max\_size; // Максимальный размер массива

int curr\_size; // Текущий размер массива

int curr; // Указатель стека

size\_t el\_size; // Размер элемента стека

};

**2) Стек, основанный на списке**

typedef struct List\_Stack List\_Stack;

struct List\_Stack

{

int curr\_size; // Текущий размер стека

int max\_size; // Максимальный размер стека

Node \*Stack\_Pointer; // Указатель на голову списка

};

**3) Узел списка**

typedef struct Node Node;

struct Node

{

char \*word;

Node \*next;

};

**4) Дополнительный список для хранения удаленных элементов списка**

typedef struct List List;

struct List

{

char \*word;

void \*ptr;

List \*next;

};

**Подпрограммы и их описание**

**Подпрограмма для добавления нового элемента стеков**

int add\_to\_stacks(Arr\_Stack \*arr\_stack, List\_Stack \*list\_stack);

**Подпрограмма для удаления элемента их стеков**

int remove\_from\_stacks(Arr\_Stack \*arr\_stack, List \*\*arr\_deleted, List\_Stack \*list\_stack, List \*\*list\_deleted);

**Подпрограмма для записи слов в обратном порядке**

int print\_backwards(Arr\_Stack \*arr\_stack, List \*\*arr\_deleted, List\_Stack \*list\_stack, List \*\*list\_deleted);

**Подпрограмма для вывода текущего состояния стеков**

int print\_current(Arr\_Stack \*arr\_stack, List \*arr\_deleted, List\_Stack \*list\_stack, List \*list\_deleted);

**Подпрограмма для вывода слова наоборот**

int print\_backwards\_word(char \*word);

**Подпрограммы для работы со стеком в виде массива**

Arr\_Stack \*create\_arr\_stack();

int push\_arr\_stack(Arr\_Stack \*s, char \*word);

char \*pop\_arr\_stack(Arr\_Stack \*s);

void free\_arr\_stack(Arr\_Stack \*s);

**Подпрограммы для работы со стеков в виде односвязного линейного списка**

List\_Stack \*create\_list\_stack();

Node \*create\_node(char \*word);

int push\_list\_stack(List\_Stack \*stack, char \*word);

Node \*pop\_list\_stack(List\_Stack \*stack);

void free\_list\_stack(List\_Stack \*stack);

void free\_node(Node \*node);

**Подпрограммы для работы с дополнительным списком для хранения удаленных элементов стеков**

List \*create\_list\_node(char \*word, void \*\*ptr);

List \*add\_to\_list(List \*head, List \*node);

void list\_apply(List \*head, void (\*f)(List\*, void\*), void \*arg);

void node\_print(List \*pers, void \*arg);

void free\_list(List \*head);

**Описание алгоритма**

1. Читается и выполняется команда пользователя
2. Если выбран пункт добавления слова, то данное слово записывается сразу в два стека
3. Если выбран пункт удаления элемента из стеков, то удаляются по одному элементу из обоих стеков, при этом адреса и значения элементов записываются в дополнительные списки освобожденных элементов и выводится как адрес, так и значение удаленного элемента.
4. Если выбран пункт вывода слов в обратном порядке, то удаляются все элементы обоих стеков, при этом каждое слово выводится наоборот, все удаленные элементы записываются в соответствующие списки.
5. Если выбран пункт вывода текущего состояния стеков, то выводятся каждый из стеков в формате “Указатель\nСлово” и все удаленные элементы из соответствующих списков в схожем формате
6. При выходе из программы очищаются все стеки и списки.

**Тестирование**

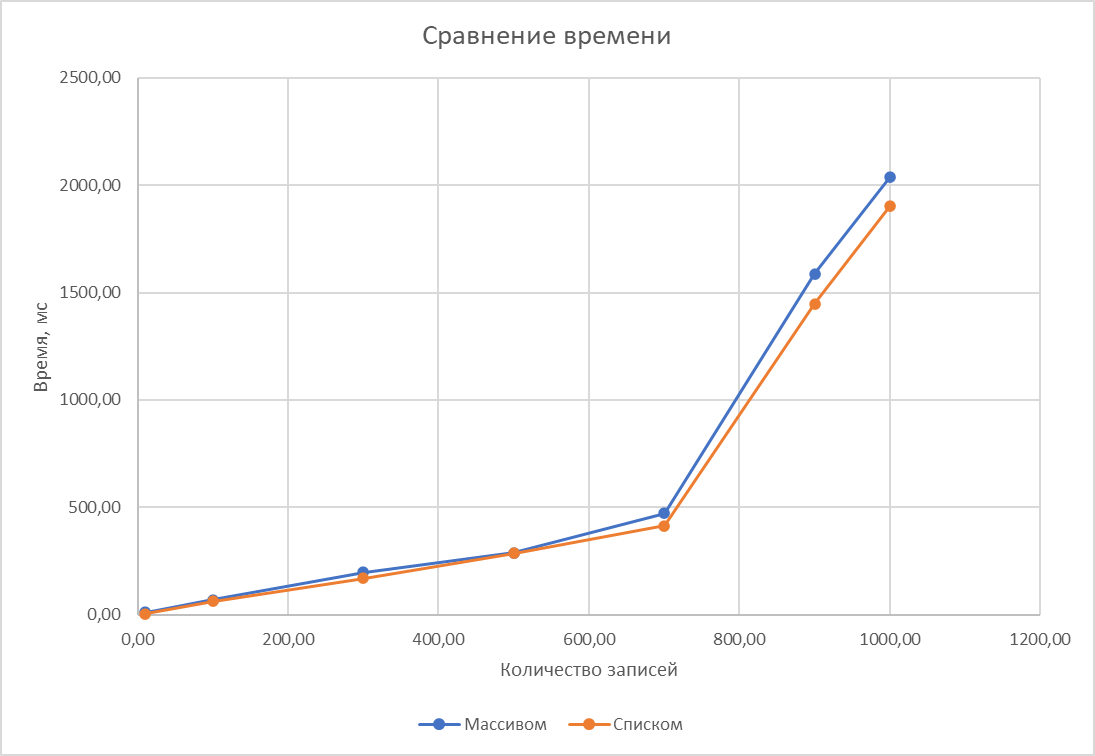
1. Ввод неверного пункта меню – “Неверный пункт меню”
2. Ввод слишком большого слова – “Слишком большое слово!”
3. Ввод пустого слова – “Пустое слово!”
4. Ввод слова с пробелами – “В слове есть пробелы!”
5. Попытка удалить слово из пустого стека – “Стек пуст!”
6. Попытка вывести слова в обратном порядке при пустом стеке – “Стек пуст!”
7. Неверный ввод пункта меню (не число) – Выход из программы с ненулевым кодом возврата
8. Попытка ввести более 1024 слов в стек – Выход из программы с ненулевым кодом возврата

**Оценка эффективности**

Замеры времени и памяти проводились на основе функции вывода слов в обратном порядке, время измерялось в мс, размер в байтах.

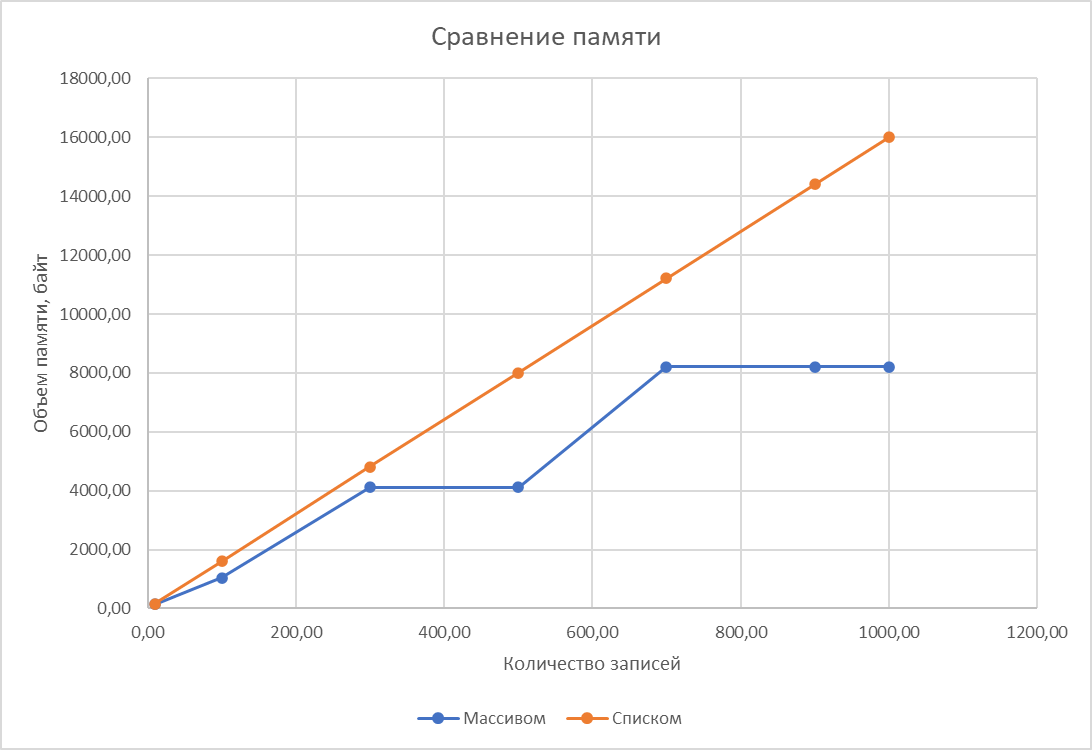
Сравнение времени(мс)

|  |  |  |
| --- | --- | --- |
| Размер | Массивом | Списком |
| 10,00 | 12,00 | 4,00 |
| 100,00 | 70,00 | 62,00 |
| 300,00 | 197,00 | 169,00 |
| 500,00 | 289,00 | 285,00 |
| 700,00 | 472,00 | 414,00 |
| 900,00 | 1587,00 | 1447,00 |
| 1000,00 | 2038,00 | 1901,00 |



Сравнение памяти(байт)

|  |  |  |
| --- | --- | --- |
| Размер | Массивом | Списком |
| 10,00 | 144,00 | 168,00 |
| 100,00 | 1040,00 | 1608,00 |
| 300,00 | 4112,00 | 4808,00 |
| 500,00 | 4112,00 | 8008,00 |
| 700,00 | 8208,00 | 11208,00 |
| 900,00 | 8208,00 | 14408,00 |
| 1000,00 | 8208,00 | 16008,00 |



Исходя из результатов измерений, становится очевидно, что хранение стека в виде массива эффективнее по памяти. Память в массиве выделяется большими кусками (это видно по графику), а в списке – линейно, к тому же массив - экономичная по памяти структура данных, так как не требуется хранить указатель на следующий элемент. По времени же результаты примерно равны, так как в моей программе при удалении элемента из стека массивом выполняется обращение по индексу, а при удалении из стека списком – обращение по конкретному адресу, при этом при записи слов в обратном порядке иных операций не выполняется, соответственно время оказалось примерно одинаковым.

**Ответы на 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 пришёл – первым ушёл, Last In – First Out (LIFO).

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

При хранении стека в виде массива под него выделяется определённый объем памяти. При хранении стека в виде списка объем памяти ограничен объёмом доступной оперативной памяти, но может быть ограничен искусственно.

Память под массив выделяется len\*sizeof(элемент). Память для списка выделяется не только под элементы самого списка, но также и под указатели.

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

При реализации стека массивом память освобождается в конце области видимости массива при уничтожении соответствующей переменной. При реализации стека списком память освобождается при каждом удалении элемента.

4. Что происходит с элементами стека при его просмотре?

При просмотре стека он очищается.

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

Эффективнее реализовывать стек массивом, поскольку добавление/удаление осуществляются просто по индексу, а память выделяется один раз если стек не расширяется в программе.