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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

О Т Ч Е Т

По лабораторной работе № 4

Название: Работа со стеком

Дисциплина: Типы и структуры данных

Вариант: 2

Студент ИУ7-31Б (Группа) Савинова М. Г. (Фамилия И. О.)

Преподаватель Силантьева А. В.

Москва, 2022

Оглавление

Цель работы

Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) 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.

Условие задачи

При реализации стека массивом располагать два стека в одном массиве. Один стек располагается в начале массива и растет к концу, а другой располагается в конце массива и растет к началу. Заполнять и освобождать стеки произвольным образом с экрана. Элементами стека являются вещественные числа. Списком реализовать один стек.

Описание ТЗ

1. Описание входных данных

Исходными данными, хранящимися в стеке, являются вещественные числа.

Ввод данных происходит с помощью выбора 0-10 пункта в меню.

```
-------------------------MENU-------------------------

ARRAY
    1  - Init stack
    2  - Add element to stack
    3  - Remove element from stack
    4  - Output current stack

LIST
    5  - Init stack
    6  - Add element to stack
    7  - Remove element from stack
    8  - Output current stack
    9  - Output current array of freed adresses

    10 - Check up time

0  - EXIT

Input an option (e.g. 1):
```

Некорректный ввод вызывает завершение программы с ненулевым кодом возврата.

Ограничения:

- Элементы стека:

- **Данные:** вещественные числа **от -1000000 до 1000000**

- Пункт меню: целое число **от 0 до 10**

2. Описание выходных данных

Опции 1-4:

В результате выполнения программы будет сформирован стек на основе массива.

Опция 5-10:

В результате выполнения программы будет сформирован стек на основе односвязанного стека.

3. Описание задачи, реализуемой в программе

Программа реализует:

- добавление элемента в стек
- удаление элемента из стека
- вывод текущего состояния стека
- измерение времени на добавление/удаление элемента из стека разными способами.

4. Способ обращения к программе

Обращение к программе происходит через консоль, путём запуска *.exe файла.

5. Описание возможных аварийных ситуаций и ошибок пользователя

Аварийная ситуация	Код завершения	Сообщение
Неверный размер стека	1	Wrong capacity :(
Не удалось выделить память	2	Memory error :(
Переполнение стека	3	Stack is overflowed :(
Стек пуст	4	Stack is empty :(
Стек не инициализирован	5	No avaliable stack :(
Нет доступных адресов памяти	6	No avaliable adress :(
Неверный выбор пункта меню	7	Wrong choice :(
Неверный тип данных для стека	8	Wrong number :(

Описание внутренних структур данных

- Тип данных элементов, хранящихся в стеке;

```
typedef double data;   // тип данных
```

- Структура, хранящая 2 стека в одном массиве;

```
typedef struct
{
    int top_1;     // номер головы первого стека
    int top_2;     // номер головы второго стека
    int capacity;  // размер стека
    data *pointer; // указатель на массив
} array_t;
```

- Структура, хранящая массив освобожденных адресов адресов;

```
typedef struct
{
    int len;        // текущий размер стека
    int capacity;   // размер стека
    node **pointer; // указатель на массив адресов
} array_adress_t;
```

- Структура, представляющая собой один элемент связанного списка

```
typedef struct node
{
    data value;        // значение
    struct node *next; // указатель на следующий элемент
} node;
```

- Структура, хранящая стек в виде односвязанного списка

```
typedef struct
{
    node *begin;
    int size;
    int capacity;
} list_t;
```

Описание алгоритма

1. Выводится меню данной программы
2. Пользователь выбирает пункт меню
3. Выход из программы осуществляется в случае ошибки или пункта меню - «0»

Тестовые данные

1. Позитивные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Создание стека в виде **массива**	1 10	Well done :)
2.	Добавление элемента в «первый» стеком (стек не переполненный)	1 10 2 1 100	Well done :)
3.	Добавление элемента во «второй» стеком (стек не переполненный)	1 10 2 2 100	Well done :)
4.	Удаление элемента из «первого» стека (стек не пуст)	1 10 2 1 10 2 1 100 3 1	This elem 100.000000 was removed from 1 stack
5.	Удаление элемента из «второго» стека (стек не пуст)	1 10 2 2 10 2 2 100 3 2	This elem 100.000000 was removed from 2 stack
6.	Вывод текущего состояния стека (стек не пуст)	1 10 2 1 10 2 2 100 4 3	| WHOLE STACK | | 10.000000 | | - | | - | | - | | - | | - | | - | | - | | - | | 100.000000 |

7.	Создание стека в виде **списка**	5 10	Well done :)
8.	Добавление элемента стек (стек не переполненный)	5 10 6 -1.67	Well done :)
9.	Удаление элемента из стека (стек не пуст)	5 10 6 -1.67 7	This elem -1.670000 was removed from stack
10.	Вывод текущего состояния стека (стек не пуст)	5 10 6 -1.67 6 100 8	| STACK | ADRESS | |----------|----------| | 100.000000|0x5590d40b97d0| | -1.670000|0x5590d40b97b0|
11.	Вывод массива свободных адресов (массив не пуст)	5 10 6 -1.67 7 9	Avaliable adresses: | 0x55a539cb77b0|

2. Негативные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Неверно выбранный пункт меню	11	Wrong choice :(
2.	Добавление элемента в заполненный стек	1 1 2 1 10 2 1 0	Stack is overflowed :(
3.	Удаление элемента из пустого массива	1 1 3 1 0	Stack is empty :(
4.	Неверно введенное значение элемента, хранящегося в стеке	1 1 2 1 a	Wrong number :(

		0	
5.	Вывод не инициализированного стека	5 10 8 0	Stack is empty :(
6.	Вывод не инициализированного массива свободных адресов	5 10 6 1 6 2 6 3 9 0	No avaliable adress :(

Оценка эффективности

Были проведены тесты с размерностями стека: 100, 1000, 5000, 8000, 10000

Результаты замеров представлены в виде таблицы:

```
------------------TEST 100 SIZE------------------
|             |   ARRAY_1  |  ARRAY_2  |   LIST    |
|SIZE(bytes)  |        8008|       8008|       1608|
|PUSH(mcs)    |           1|          1|          5|
|POP(mcs)     |           0|          1|          4|
------------------TEST 1000 SIZE------------------
|             |   ARRAY_1  |  ARRAY_2  |   LIST    |
|SIZE(bytes)  |        8008|       8008|      16008|
|PUSH(mcs)    |          10|         10|         52|
|POP(mcs)     |           5|          5|         81|
------------------TEST 5000 SIZE------------------
|             |   ARRAY_1  |  ARRAY_2  |   LIST    |
|SIZE(bytes)  |        8008|       8008|      80008|
|PUSH(mcs)    |          60|         58|        271|
|POP(mcs)     |          25|         24|        200|
------------------TEST 8000 SIZE------------------
|             |   ARRAY_1  |  ARRAY_2  |   LIST    |
|SIZE(bytes)  |        8008|       8008|     128008|
|PUSH(mcs)    |          82|        110|        605|
|POP(mcs)     |          39|         39|        323|
------------------TEST 10000 SIZE-----------------
|             |   ARRAY_1  |  ARRAY_2  |   LIST    |
|SIZE(bytes)  |        8008|       8008|     160008|
|PUSH(mcs)    |         109|        143|        610|
|POP(mcs)     |          50|         49|        407|
-------------------------------------------------
```

Эффективность по времени:

Из полученных данных видно, что эффективность по времени при работе стека в виде массива больше: при добавление в 5-7 раз; при удалении в 8-10 раз.

Эффективность по памяти:

Стек в виде массива занимает примерно в 2 раза меньше объема по памяти, из-за того что для реализации списка нам необходимо хранить еще и указатель на следующий элемент.

Выводы

Хранение стека в виде массива эффективнее: как по памяти, так и по времени. По памяти выигрыш в 2 раза; по времени выполнения: ~7-8 раз. Это объясняется тем, что при реализации в виде массива доступ к нужном у элементу осуществляется передвижением на него указателя. При реализации же в виде списка требуется дополнительное время для удаления верхушки стека, перестановки указателя.

Но если подразумевается неполное заполнение стека (~40-50%), то целесообразнее использовать реализацию стека в виде списка.

Так же можно сделать вывод о том, что в результате тестирования фрагментация не наблюдается (см. приложение).

*Ответы на контрольные вопросы

1. *Что такое стек?*

Стек – последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны. Функционирует по принципу LIFO: «последний зашел – первый вышел».

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

При реализации стека **списком** память выделяется динамически по мере добавления новых элементов и освобождается при завершении программы.

При реализации стека **массивом** выделяется фиксированный участок памяти; в стеке не может быть больше заданного числа элементов. Добавление нового элемента происходит путём смещения указателя на последний элемент.

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

При реализации **списком** память из-под элемента освобождается при его удалении.

При реализации **массивом** память из-под элемента не освобождается, происходит лишь изменение значения указателя на последний элемент.

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

Мы храним только указатель на последний элемент, поэтому чтобы пройтись по стеку нужно обойти все элементы стека по указателям на предыдущий элемент.

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

Реализация стека **массивом** даёт 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.

Реализация в виде **списка** выигрывает по памяти лишь в случае частичного заполнения стека.

Приложение

1. Создать стек на основе списка из 4 элементов

2. Вывести состояние текущего стека

```
|       STACK        |       ADRESS       |
|--------------------|--------------------|
|            4.000000|      0x55ffa95a57b0|
|            3.000000|      0x55ffa95a5790|
|            2.000000|      0x55ffa95a5770|
|            1.000000|      0x55ffa95a5750|
```

3. Удаление всех элементов стека

```
This elem 4.000000 was removed from stack
This elem 3.000000 was removed from stack
This elem 2.000000 was removed from stack
This elem 1.000000 was removed from stack
```

4. Вывести состояние текущего массива адресов

```
Avaliable adresses:
|      0x55ffa95a57b0|
|      0x55ffa95a5790|
|      0x55ffa95a5770|
|      0x55ffa95a5750|
```

5. Снова заполним стек 4 элементами

```
|       STACK        |       ADRESS       |
|--------------------|--------------------|
|            4.000000|      0x55ffa95a57b0|
|            3.000000|      0x55ffa95a5790|
|            2.000000|      0x55ffa95a5770|
|            1.000000|      0x55ffa95a5750|
```

Видим, что новые добавленные элементы размещаются на тех же адресах, на которых размещались и удаленные элементы.