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
Лабораторная работа №4
«Работа со стеком»

Группа ИУ7-34Б

Дисциплина Типы и структуры данных

Вариант 14

Студент Козлитин Максим Александрович

Преподаватель Барышникова Марина Юрьевна

2022г.

Цель работы

Реализовать операции работы со стеком, который представлен в виде динамического 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.

Условие задачи

Создать программу работы со стеком, выполняющую операции добавление, удаления элементов и вывод текущего состояния стека.
Реализовать стек:

A. Массивом
B. 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.

Ввести целые числа в 2 стека. Используя третий стек отсортировать все введенные данные.

Описание исходных данных

1. Выбор действия:

   Целое число от 0 до 15.

2. Стек (Ввод):

   Целые числа разделенные пробелом.
   Чтобы прекратить ввод нужно ввести любой непробельный символ и не цифру.

3. Число (добавление элемента в стек):

   Целое число.

Способ обращения к программе

Работа с программой осуществляется с помощью консоли.

Программа запускается с помощью команды: **./app.exe**

Далее пользователь выполняет взаимодействие с помощью меню.
Меню выводится в начале и, далее, каждые 3 введенные команды.

Меню:

```
|>    0 - Завершить.
|>   1 - Поменять тип стека [сейчас: Классический]
|>   2 - Сравнить эффективность для пункта 12.
|>   3 - Сравнить объем памяти.


|Первый стек
|
|>   4 - Ввести.
|
|>   5 - Напечатать.
|
|>   6 - Добавить элемент.
|
|>   7 - Удалить элемент.


|Второй стек
|
|>   8 - Ввести.
|
|>   9 - Напечатать.
|
|>  10 - Добавить элемент.
|
```

```
|>  11 - Удалить элемент.


|Третий стек
|
|>  12 - Слить первый и второй стеки в третий.
|
|>  13 - Напечатать.
|
|>  14 - Добавить элемент.
|
|>  15 - Удалить элемент.
```

Описания возможных аварийных ситуаций и ошибок пользователя

1. Ввод не совпадающий с форматом входных данных (описано в Описание исходных данных).
2. Ввод целых чисел превышающих максимальное допустимое значение типа данных int.

Описание внутренних структур данных

Стек в виде динамического массива:

```
typedef struct
{
    size_t size; // размер стека
    int *head; // указатель на последний элемент
    int *data; // числа, добавленные в стек
} mystack_t;
```

size_t *используется для задания размеров наших массивов, так как является особым макросом, значение которого позволяет адресовать массив любой теоретически возможной длины для данной машины.*

int *- тип данных знакового целого числа.*

Стек в виде односвязного списка:

```
struct node_t
{
    int data; // число, добавленное в стек
    node_t *next; // указатель на следующий узел
};

typedef struct
{
    node_t *head; указатель на последний элемент-узел
} linked_list_stack_t;
```

int *- тип данных знакового целого числа.*

Описание алгоритма

Сортировка стека - **stack_sort**:

1. Создается временный стек.
2. Пока исходный стек не пустой достаем из него элемент.
3. Достаем элементы из временного стека, пока он либо не опустеет, либо элемент из исходного стека не встанет на своем место - больше или равен элемента из временного стека.
4. Добавляем во временный стек наш элемент исходного стека.
5. После того как исходный стек опустел, переносим содержимое временного стека в него.

Слияние двух стеков - **stack_merge**:

1. Пока оба стека не опустеют достаем наибольший из их последних элементов.
   Если один из стеков пуст достаем оттуда, где элементы есть.
2. Добавляем в результирующий стек очередной элемент.

Для стека, реализованного в виде односвязного списка алгоритм не отличается.

Оценка эффективности

Для оценки эффективности в программе реализованы функции, позволяющие создать набор различных данных: возрастающая последовательность, убывающая последовательность, последовательность случайных чисел. Данные последовательности добавляются в стек таким образом, что данные получаются развернуты. Т. е. первый элемент последовательности будет добавлен первым, а выйдет из стека последним.

Размеры: [10x10] [100x100]
Исходные данные: [Возрастающая последовательность] [Убывающая последовательность] [Случайная последовательность]

Для выполнения операции требуется два стека поэтому рассматриваются комбинации.

Комбинации:
[Возрастающая + Возрастающая] [Возрастающая + Убывающая]
[Возрастающая + Случайная]
[Убывающая + Убывающая][Убывающая + Случайная]
[Случайная + Случайная]

Эффективность методов

	Возрастающая Возрастющая		**Возрастающая Убывающая**		**Возрастающая Случайная**		**Убывающая Убывающая**		**Убывающая Случайная**		**Случайная Случайная**	
Размер	**Массив**	**Список**	**Массив**	**Список**	**Массив**	**Список**	**Масив**	**Список**	**Массив**	**Список**	**Массив**	**Список**
1	780	512	674	509	664	423	643	423	843	422	643	424
10	8867	9523	12689	15268	10392	10519	17606	17993	13724	14083	12153	11677
20	15276	16535	39299	40488	26576	25657	71956	81198	54964	59263	36264	37412
30	23572	25430	107110	90047	62436	69674	15396	159481	114705	127080	93318	104505
40	32082	32936	145461	154169	88118	91046	25677	270964	215731	219978	138121	143919
50	36225	40347	209157	239656	121142	138504	40135	419731	312714	329648	206158	221909
60	46376	48643	296499	319101	164915	175436	56886	613025	404676	447896	258975	293717
70	60298	55334	398926	436736	264766	284935	82045	884005	550135	595576	380127	407271
80	62789	65058	515081	532853	276757	287541	97646	992805	723093	743907	479119	576223
90	72964	73268	650504	706428	303271	313957	12463	124401	984651	971364	695059	720556
100	96472	95251	858410	905204	518079	570141	17171	184841	131317	134036	797348	944778

Разница эффективности по отношению к стандартному:

	Возрастающая Возрастающая	Возрастающая Убывающая	Возрастающая Случайная	Убывающая Убывающая	Убывающая Случайная	Случайная Случайная
Размер	**Процент**					
1	34,4	24,5	36,3	34,2	49,9	34,1
10	-7,4	-20,3	-1,2	-2,2	-2,6	3,9
20	-8,2	-3,0	3,5	-12,8	-7,8	-3,2
30	-7,9	15,9	-11,6	-3,6	-10,8	-12,0
40	-2,7	-6,0	-3,3	-5,5	-2,0	-4,2
50	-11,4	-14,6	-14,3	-4,6	-5,4	-7,6
60	-4,9	-7,6	-6,4	-7,8	-10,7	-13,4
70	8,2	-9,5	-7,6	-7,7	-8,3	-7,1
80	-3,6	-3,5	-3,9	-1,7	-2,9	-20,3
90	-0,4	-8,6	-3,5	0,2	1,3	-3,7
100	1,3	-5,5	-10,0	-7,6	-2,1	-18,5
Максимум	34,4	24,5	36,3	34,2	49,9	34,1
Минимум	-11,4	-20,3	-14,3	-12,8	-10,8	-20,3
Среднее	-0,2	-3,5	-2,0	-1,7	-0,1	-4,7

По данной таблице, можем заметить, что максимальная прирост эффективности достигает 36,3%, в пользу связного списка, но это только для размера 1.
При этом, минимальная разница составляет -20,3%, то есть максимальная прирост эффективности в пользу стандартной реализации достигает 20,3%.
В среднем же значения отличаются не более чем на 4,7% в пользу реализации на массиве

Данное явление можно объяснить более быстрым доступом к элементу, очищение элемента также производится быстрее.

Так как с увеличением числа элементов в связном списке они становятся все более разбросаны по памяти, то можно проследить как скорость доступа влияет на производительность - если на стеке из одного элемента связный список выигрывает по эффективности, то при увеличении размера реализация на массиве работает быстрее.

Разница эффективности алгоритма для разных последовательностей:

	Убывающая Убывающая Возрастающая Возрастающая	Убывающая Убывающая Возрастающая Убывающая	Убывающая Случайная Возрастающая Случайная	Убывающая Убывающая Случайная Случайная	Убывающая Случайная Случайная Случайная	Случайная Случайная Возрастающая Случайная
Размер	**Процент**					
1	17,6	-4,8	21,2	0,0	23,7	-21,3
10	49,6	27,9	24,3	31,0	11,4	27,0
20	78,8	45,4	51,6	49,6	34,0	57,9
30	84,7	30,4	45,6	39,4	18,6	74,7
40	87,5	43,4	59,2	46,2	36,0	76,8
50	91,0	47,9	61,3	48,6	34,1	82,4
60	91,8	47,9	59,2	54,5	36,0	82,1
70	92,7	51,4	51,9	53,7	30,9	84,1
80	93,6	47,3	61,7	50,9	33,7	86,9
90	94,1	47,8	69,2	44,2	29,4	89,5
100	94,4	50,0	60,5	53,6	39,3	87,9
Максимум	94,4	51,4	69,2	54,5	39,3	89,5
Минимум	17,6	-4,8	21,2	0,0	11,4	-21,3
Среднее	79,6	39,5	51,4	42,9	29,7	66,2

По данной таблице можно заметить, что убывающая последовательность является худшим случаем для нашего алгоритма. Это можно объяснить тем. что стеки сортируются сначала по убыванию, а потом добавляются в стек так, что получается отсортированные по возрастанию данные. Соотвественно добавляя в исходные стеки две возрастающие последовательности мы формально сразу получаем готовый результат.

Две возрастающие последовательности в среднем быстрее на 79,6%, чем две убывающие.

При этом две убывающие последовательности на 42,9% уступают по эффективности двум случайным, а в случае с одной убывающей и одной случайно на 29,7% - что все еще является весомое разницей.

Лучший случай - возрастающая последовательность. Возрастающая и случайная в сравнении с двумя случайными в среднем работает эффективнее на 66,2%.

Это говорит о том, что в худшем и лучше случае прирост эффективности по сравнению со случайными данными сильно отличается в худшую и лучшую сторону, соотвественно.

Сравнение памяти:

Размер	Массив, байт	Список, байт	Разница
1	32	24	-33,3
10	104	168	38,1
20	184	328	43,9
30	264	488	45,9
40	344	648	46,9
50	424	808	47,5
60	504	968	47,9
70	584	1128	48,2
80	664	1288	48,4
90	744	1448	48,6
100	824	1608	48,8

В случае с одним элементов массив уступает по памяти, из-за того что сама структура требует больше чем выделенная память на один элемент, как массива, так и списка. Разница достигает 33,3% в пользу списка.

Список уступает по памяти, с увеличением размера, так как элементы массива менее требовательны к памяти, нежели узлы связного списка.
Таким образом разница достигает от 38,1% до 48,8% что является весомым недостатком данной реализации.

Вывод

В ходе проделанной работы, я приобрел навыки обработки стека, который представлен в виде динамического массива и в виде односвязного списка. Был произведен анализ преимуществ и недостатков каждой реализации. Получено представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных.

По результатам, проведенной оценки эффективности, было выявлено, что стек, реализованный с помощью связного списка требует на 38,1% - 48,8% памяти больше, чем с помощью массива. При этом при выполнение слияния двух стеков с сортировкой разница для двух данных методов незначительна и не превышает 5% в пользу реализации, с помощью массива.

Подробные результаты и методы оценки эффективности программы можно изучить в пункте «Оценка эффективности».

Также стоит отметить что при использовании стека на связном списке происходит фрагментация памяти, т.е данные не лежат непрерывным блоком в памяти, а разбросаны по ней и освобожденные ранее адреса повторно не используются.

Контрольные вопросы

1. **Что такое стек?**

   Последовательный список с переменной длиной, в котором операции добавления и удаления элементов возможно только с одной стороны - вершины. Стек функционирует по принципу: Last In - First Out (LIFO) или последним пришел - первым вышел.

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

   Массив:
   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. Физическая структура обычно дополняется дескриптором стека, в котором хранится имя стека, адрес нижней границы стека, адрес верхней границы стека

   Список:
   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. При физической реализации стека в виде односвязного линейного списка дескриптор будет отличаться отсутствием верхней границы стека. В этом случае объем стека ограничивается только объемом доступной оперативной памяти.

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

   Массив:
   При исключении элемента из стека сначала считываются данные, а затем происходит перемещение указателя на верхний элемент к предыдущему элементу. Если указатель стека выходит за нижнюю границу массива, то

стек пуст.

Список:
При исключении элемента сначала по указателю стека считывается информация об исключаемом элементе, а затем указатель смещается к предыдущему элементу. После чего освобождается память, выделенная под элемент. Если указатель имеет значение нулевого адреса, то стек пуст.

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

Принцип организации стека предполагает, что в текущий момент времени доступен элемент, индекс которого хранится в указателе на верхний элемент стека. То есть для просмотра стека нужно поочередно исключать элементы, таким образом получая их значение.
Соответсвенно, стек при просмотре всего содержимого очистится.

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

В случае статического массива реализация с помощью списка будет эффективнее по памяти, в зависимости от заполненности данными по отношению к максимально возможному размеру.

В случае с динамическим массивом эффективность стека на связном списке по времени не уступает, но требуемая память значительно увеличивается с увеличением размера.

Таким образом требования к памяти - ключевой фактор при выборе реализации. Если заполненность данных будет не столь велика, то стоит отдавать предпочтение стеку на связном списке.