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 №4

Группа ИУ7-35Б

Тема работы **Работа со стеком**

Студент Баранов Николай Алексеевич

Преподаватель Барышникова Марина Юрьевна

2022 г.

Цель работы: реализовать операции со стеком, который представлен в виде 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.

1. Описание условия задачи

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек массивом и списком. Все стандартные операции со стеком должны быть оформлены подпрограммами. Ввести арифметическое выражение типа: число знак ... число знак число. Вычислить значение выражения.

2. Описание ТЗ

2.1. Описание исходных данных и результатов.

При чтении данных программа принимает на вход либо число в диапазоне от 0 до 999999, либо знак операции: +, - или *. Все вычисления производятся по модулю 1000000.

При вводе числа итераций при измерении программа считывает число в указанном диапазоне.

Выбор действия происходит в меню, где вводится число – номер действия.

При вычислении значения выводится только значение.

При выводе текущего состояния стека для стека на списке выводятся значения его элементов и их адреса, а также история выделения и освобождения областей памяти под элементы стеков на списке с момента последнего вывода текущего состояния стека. Если состояние стека ни разу не выводилось, то будет выведена история выделения и освобождения памяти с момента запуска программы. История выделения и освобождения памяти не будет вестись во время измерения эффективности работы программы, так как это замедляет время работы стека на списке, но не влияет на время работы стека на массиве, тем самым искажая результаты измерений.

При сравнении эффективности для каждого способа выводится среднее время работы и затраченная память на хранение элементов стека.

2.2. Описание задачи, реализуемой программой.

Программа позволяет ввести арифметическое выражение, содержащее до 1000 чисел, и вычислить его значение по модулю 1000000. Числа и операции записываются в стек слева направо. Вычисление происходит слева направо.

2.3. Способ обращения к программе.

Запуск программы:

./app.exe

В этом случае программа начнёт выполнять свою задачу. Ввод осуществляется вручную с клавиатуры.

2.4. Описание возможных аварийных ситуаций и ошибок пользователя.

2.4.1. Переполнение буфера. Возникает, если строка при чтении оказалась длиннее чем 4 символа (1 символ при чтении знака операции).

2.4.2. Неожиданный символ. Возникает, если в строке присутствуют символы, не предусмотренные программой, или из-за неправильного формата входных данных (например, лишний пробел, лишняя точка в числе, пробел посреди числа и т. д.).

2.4.3. Число не входит в диапазон. Возникает, если пользователь ввёл число, которое не входит в диапазон допустимых значений.

2.4.4. Ошибка при выделении памяти. Практически не зависит от действий пользователя. Возникает только при неудачной попытке выделить память на куче, поэтому вероятность возникновения крайне мала.

3. Описание внутренних структур данных.

Для стеков используются 3 структуры:

```
struct astack
{
    size_t size;
    TYPE_STACK *arr;
};

typedef struct my_leaf my_leaf;

struct my_leaf
{
    TYPE_STACK value;
    my_leaf *next;
};

struct lstack
{
    size_t size;
    my_leaf *head;
};
```

astack – стек на массиве.
Её поля:
size – целое число. Хранит размер стека.
arr – указатель на тип, который зависит от значения TYPE_STACK. Хранит элементы стека.
my_leaf – листья списка.
Её поля:
value – тип зависит от значения TYPE_STACK. Хранит значение.
next – указатель на my_leaf. Хранит указатель на следующий элемент.
astack – стек на массиве.
Её поля:
size – целое число. Хранит размер стека.
head – указатель на my_leaf. Хранит начало стека.

4. Описание алгоритма.

Алгоритм умножения для стеков одинаковый. Сначала создаются два стека (назовём их левый и правый). левый инициализируется пустым, а в правый копируются элементы из стека, переданного в качестве аргумента. После этого правый стек переворачивается, чтобы операции выполнялись слева направо. Далее заходим в цикл, из которого выходим только либо при ошибке выделения памяти, либо, когда суммарный размер двух стеков не станет равный 1. В цикле сначала берём верхний элемент из правого стека (он гарантированно будет, и он гарантированно будет числом). Далее, если возможно, берём верхние элементы правого и левого стеков (если возможно, гарантированно хотя бы один из этих элементов удастся взять, также гарантированно, что это будут знаки операций). Далее сравниваем знаки операций. Если считали знак только из левого стека, то берём ещё один элемент из левого стека, выполняем над ним и самым первым элементом операцию, а результат помещаем в правый стек. Если считали знак только из правого стека, то помещаем в левый стек число, затем помещаем знак операции из правого стека. Если оба знака успешно считались и знак из правого стека приоритетнее, то сначала возвращаем в левый стек взятый из него знак, далее действуем как в случае отсутствия знака в левом стеке. Иначе возвращаем в правый стек взятый из него знак операции, а далее действуем как в случае отсутствия знака в правом стеке. После выполнения цикла в правом стеке будет лежать только результат, а левый стек будет пустой.

Копирование стека выполняется за O(n), где n – размер стека. Переворот стека также выполняется за O(n). Вычисление выполняется за O(n), так как количество перемещений в левый стек и количество выполненных операций с последующим возвратом в правый стек будет не более c * n, где c = const. Суммарная асимптотика получается O(n).

5. Сравнительный анализ разных способов хранения и перемножения.

Для замера времени и памяти программа запускала 1000 итераций, на каждой из которых при этом заново генерировалась исходная строка. Память выводится наибольшая использованная в стеках, а время работы выводится среднее за все итерации.

В таблицах ниже приведены результаты работы программы для разных размерностей.

Количество чисел	Время работы, мкс (массив)	Используемая память, байт (массив)	Время работы, мкс (список)	Используемая память, байт (список)
10	4	16008	9	320
50	9	16008	25	1600
100	14	16008	44	3200
500	62	16008	190	16000
1000	121	16008	386	32000

Для стека, реализованного на массиве, используемая память одинакова, так как под него всегда выделяется максимально возможная используемая память.

Можно было сделать размер стека на массиве динамически изменяемым, тогда памяти будет использоваться даже меньше чем для случая для списка (но при удвоении размера текущей памяти – не всегда), но время работы повысится, хотя и останется скорее всего меньше чем для стека на списке. Стек на списке работает дольше примерно на 210% (для больших размерностей, на малых разница меньше), так как каждый раз он выделяет и освобождает память под листья, а также при непоследовательном хранении переход по адресам будет дольше. Память сравнялась на 500 элементах, дальше по памяти выигрывает уже стек на массиве, а на максимальной размерности стек на списке расходует почти на 100% больше памяти.

Таким образом, стек на списке выиграл только на небольших размерах по памяти, однако из-за непоследовательного хранения у него меньше шанс, что не удастся выделить память под очередной элемент, так как один элемент требует меньше памяти чем подряд идущий массив, из-за чего фрагментация памяти меньше повлияет на него, однако из-за огромного числа небольших выделений памяти он способен вызвать эту фрагментацию памяти. Стек на массиве выиграл в остальных случаях по времени и по памяти, однако он с большей вероятностью может столкнуться с проблемой выделения памяти, которая может возникнуть из-за фрагментации памяти из-за других динамических структур, которые выделят много кусков памяти меньшей размерности.

6. Отображение текущего состояния стека.

Состояние стека для входной строки 2 * 3 + 1:

Адрес	Операция
0x563b31aa69a0	Выделение
0x563b31aa69c0	Выделение
0x563b31aa6160	Выделение
0x563b31aa6a30	Выделение
0x563b31aa6a50	Выделение
0x563b31aa6b40	Выделение
0x563b31aa6b60	Выделение
0x563b31aa6b80	Выделение

0x563b31aa6ba0	Выделение
0x563b31aa6cd0	Выделение
0x563b31aa6cd0	Освобождение
0x563b31aa6ba0	Освобождение
0x563b31aa6ba0	Выделение
0x563b31aa6cd0	Выделение
0x563b31aa6b80	Освобождение
0x563b31aa6cd0	Освобождение
0x563b31aa6b60	Освобождение
0x563b31aa6ba0	Освобождение
0x563b31aa6ba0	Выделение
0x563b31aa6b60	Выделение
0x563b31aa6b60	Освобождение
0x563b31aa6ba0	Освобождение
0x563b31aa6ba0	Выделение
0x563b31aa6b60	Выделение
0x563b31aa6b40	Освобождение
0x563b31aa6b60	Освобождение
0x563b31aa6ba0	Освобождение
0x563b31aa6ba0	Выделение

```
----------------------------
|0x563b31aa6ba0|Освобождение|
----------------------------
```

Сразу замечу, что первые 5 выделенных участков памяти – исходный стек. В функции вычисления выражения он копируется, и операции проводят уже над копией, так что исходный стек не меняется. Также элементы этого стек ещё существовали на момент просмотра стека, поэтому в таблице нет записей об освобождении этих областей памяти. Из этой таблицы видно, что память при копировании выделялась через равные промежутки по 32 байта каждый. Размер каждого «листа» - 16 байт, поэтому, вероятнее всего, рядом с выделенной памятью лежит служебная информация для её освобождения, либо же память попросту выделялась с запасом. При этом при создании стека память под все 5 элементов не лежит одной областью, а разбросана по разным местам. На их расположение в памяти могли повлиять сама история выделения и освобождения памяти, так как память под неё выделяется тоже динамически. Переворот стека выполняется перестановкой указателей, поэтому он не отображается в таблице. Однако влияние переворота не заметно, так как копирование начинается с головы стека, поэтому для элементов, память под которые была выделена раньше, память освобождалась позже. Также я заметил, что при выделении памяти первоначально занимались те адреса, которые были последними освобождены. Это наталкивает на мысль, что в куче, возможно, используется стек освобождённых адресов.

7. Набор тестов.

Тесты описаны только для ввода значений. Отсутствие выходных данных означает в данном случае что тест позитивный.

В меню проверяется только то что после каждого действия запускается нужная функция.

Входные данные	**Выходные данные**	**Что и где проверялось**
1		Ввод минимального целого числа
3		Ввод целого числа не на границе
999999		Ввод целого числа максимального размера
000003		Ввод целого числа с ведущими нулями (без переполнения буфера)
+		Ввод знака (корректного)
	Ошибка. Пустой ввод.	Ввод пустой строки при вводе числа

	Ошибка. Пустой ввод.	Ввод пустой строки при вводе знака
what	Ошибка. Встречен некорректный символ.	Ввод не числа, если нужно целое число
1234567	Ошибка. Переполнение буфера при вводе.	Переполнение буфера (при чтении целого числа)
/	Ошибка. Встречен неожиданный символ.	Неверный символ при вводе знака
++	Ошибка. Переполнение буфера при вводе.	Переполнение буфера (при чтении знака)

8. Выводы по проделанной работе.

В ходе выполнения работы я изучил операции со стеком и реализовал 2 стека: на массиве и на списке, а также реализовал вычисление выражения с помощью стеков, для которой сравнили эффективность работы при использовании вышеуказанных стеков, а также изучили состояние стека на списке при выполнении.

9. Ответы на вопросы.

9.1. Что такое стек?

Стек – структура данных, работающая по принципу LIFO. В общем случае у неё есть операции добавления и удаления элемента, но можно так же добавить такие операции, как вычисление его размера, проверка на пустоту и т. д.

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

Если реализуется массивом, то можно выделить память либо сразу под максимальный размер стека (если есть верхний предел), либо сначала выделить, а потом расширять по мере добавления элементов. Память выделяется только под элементы.

Если реализуется списком, то память выделяется каждый раз при создании листа под один лист. Однако лист помимо значения содержит ещё и адрес следующего элемента, из-за чего увеличивается расход памяти.

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

Если реализуется массивом, то память либо не освобождается до момента уничтожения стека (если размер постоянный), либо сокращается по мере удаления элементов.

Если реализуется списком, то память освобождается каждый раз при удалении листа.

9.4. Что происходит с элементами стека при его просмотре?

Обычно для того, чтобы посмотреть элемент стека, его надо оттуда вынуть. При этом вынуть можно только верхний элемент стека, и при этом он удаляется

из стека. Однако при некоторых реализациях (например, в стандартной библиотеке шаблонов в C++) можно посмотреть значение верхнего элемента, не доставая его. Просмотр остальных элементов в общем случае не предусмотрен, поэтому для их просмотра придётся вынуть из стека те элементы, которые находятся выше.

9.5. Каким образом эффективнее реализовывать стек? От чего зависит?

Необходимость той или иной реализации зависит от ситуации. Стек на статическом массиве самый эффективный, но он ограничен по памяти, из-за чего его применение ограничено, да и на малых размерах он забирает слишком много памяти. Стек на динамическом массиве решает эти проблемы, но частично, так как он может столкнуться с проблемой фрагментации памяти. Стек на списке работает медленнее, но используемая под него память при малых размерах меньше чем для стека на статическом массиве, а с проблемой фрагментации памяти он столкнётся с меньшей вероятностью, так как листа сравнительно небольшого размера и могут располагаться в разных частях памяти, однако это как раз и может создать фрагментацию памяти.