|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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**

**«РАБОТА СО СТЕКОМ»**

**по курсу «Типы и структуры данных»**

**Вариант 1**

Студент: Писаренко Дмитрий Павлович

Группа: ИУ7-34Б

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Писаренко Д.П.

*подпись, дата фамилия, и.о.*

Преподаватель \_\_**\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Барышникова М.Ю.

*подпись, дата фамилия, и.о.*

# Цель работы

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

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.

Распечатайте убывающие серии последовательности целых чисел в обратном порядке.

# Техническое задание

## Исходные данные

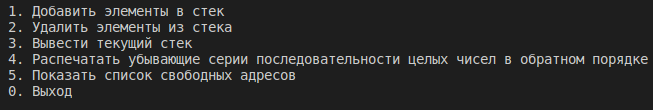
*Выбор действия*: целое число от 0 до 3.

0. Выход из программы

Не требует ввода от пользователя.

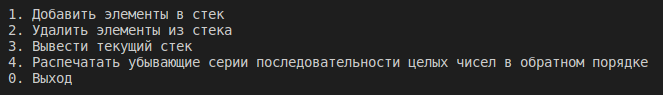
1. Заполнение стека с помощью списка:

Пользователю на выбор дается 6 действий, каждое из которых выполняется при указании целого числа, написанного слева от действия:



2. Заполнение стека с помощью массива:

Пользователю на выбор дается 5 действий, каждое из которых выполняется при указании целого числа, написанного слева от действия:

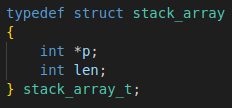


3. Сравнение по времени и памяти двух представлений

Пользователю на выбор дается 2 действия, каждое из которых выполняется при указании целого числа, написанного слева от действия:

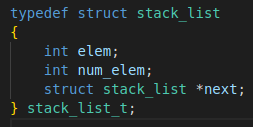


## Структуры данных



|  |
| --- |
| int \*p – указатель на элемент стека  int len – количество заполненных элементов стека |

Описание полей структуры stack\_array\_t



|  |
| --- |
| int elem – текущий элемент стека  int num\_elem – количество заполненных элементов стека  struct stack\_list \*next – указатель на следующий элемент стека |

Описание полей структуры stack\_list\_t

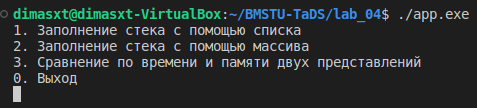
# Способ обращения к программе

Работа с программой осуществляется с помощью консоли.

Сборка осуществляется c помощью команды **make build**

Запуск выполняется с помощью команды **./app.exe**

Дальнейшая работа производится с помощью меню:

\

# Описание алгоритма

1. Вводится пункт первого меню, пользователь переходит на одно из трех последующих меню с остальными командами.

2. Во втором меню пользователь выбирает команду, которое отвечает за определенное действие. При добавлении сначала проверяется, возможно ли добавить заданное пользователем количество элементов, если да, то при вводе в стек-массив происходит сдвиг указателя каждый введенный элемент, также увеличивается значение количества элементов в списке, если нет – выход с ненулевым кодом возврата. При вводе в стек-список выделяется память под конкретный элемент, далее туда записывается адрес предыдущего значения, указатель стека сдвигается на новый элемент. В задании по варианту сравнивается последний элемент стека с предпоследним: если предпоследний оказывается больше последнего, то на экран выводится последний элемент. Конец серии регулируется флагом flag. Он сообщает, когда надо переносить элемент на новую строку (начата новая серия).

# Тестирование

## Позитивные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| # | Входные данные | Выходные данные | Результат |
| 1 | Ключ = 2  Ключ = 1  Кол-во элементов: 4  1 2 3 4  Ключ = 3 | 4 3 2 1 | Ожидание следующего ключа |
| 2 | Ключ = 2  Ключ = 1  Кол-во элементов: 5  7 5 7 4 3  Ключ = 4 | 3 4 7  5 7 | Ожидание следующего ключа |
| 3 | Ключ = 2  Ключ = 1  Кол-во элементов: 4  5 1 3 4  Ключ = 2  Кол-во удаляемых: 2  Ключ = 3 | 1 5 | Ожидание следующего ключа |
| 4 | Ключ = 1  Ключ = 1  Кол-во элементов: 4  1 2 3 4  Ключ = 4 | 4 3 2 1 | Ожидание следующего ключа |
| 5 | Ключ = 1  Ключ = 3 | Стек пуст | Ожидание следующего ключа |
| 6 | Ключ = 1  Ключ = 1  Кол-во элементов: 5  1 2 3 4 5  Ключ = 2  Кол-во удаляемых: 2  Ключ = 5 | Номера по порядку и адреса двух удаленных элементов  Например:  1 | 0x559b78c14b40  2 | 0x559b78c14b20 | Ожидание следующего ключа |

## Негативные тесты

|  |  |  |  |
| --- | --- | --- | --- |
| # | Входные данные | Выходные данные | Результат |
| 1 | Ключ = 4 | Номер меню - целое число от 0 до 3 | Код ошибки 2 |
| 2 | Ключ = 2  Ключ = 1  Кол-во элементов: 1000000 | Слишком много элементов для добавления | Код ошибки 7 |
| 3 | Ключ = 2  Ключ = 2  Кол-во элементов: 5  Кол-во удаляемых: 10 | Слишком много элементов для удаления | Код ошибки 7 |
| 4 | Ключ = 1  Ключ = 1  Кол-во элементов: dsfgew | Количество добавляемых элементов - натуральное число | Код ошибки 5 |
| 5 | Ключ = 1  Ключ = 1  Кол-во элементов: 5  1 4 1 2 dd | Ошибка в чтении элемента | Код ошибки 8 |
| 6 | Ключ = 2  Ключ = 122 | Действие - целое число от 0 до 4 | Код ошибки 4 |

# Таблицы с результатами измерения времени и памяти

Время замерялось при 100 выполнениях функции.

Время в таблицах указано в мс.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Стек** | **Размер** | **Количество заполненных элементов (%)** | | | | |
| 10 | 20 | 30 | 40 | 50 |
| Массив | 10.000 | 1 | 2 | 4 | 5 | 8 |
| Список | 3 | 7 | 10 | 15 | 19 |
| Массив | 100.000 | 18 | 34 | 50 | 70 | 89 |
| Список | 50 | 99 | 157 | 195 | 274 |
| Массив | 1.000.000 | 169 | 276 | 364 | 482 | 657 |
| Список | 408 | 730 | 966 | 1257 | 1534 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Стек** | **Размер** | **Количество заполненных элементов (%)** | | | | |
| 60 | 70 | 80 | 90 | 100 |
| Массив | 10.000 | 10 | 10 | 12 | 13 | 16 |
| Список | 23 | 29 | 32 | 39 | 59 |
| Массив | 100.000 | 105 | 143 | 159 | 169 | 219 |
| Список | 345 | 370 | 388 | 402 | 454 |
| Массив | 1.000.000 | 694 | 716 | 832 | 894 | 986 |
| Список | 1750 | 2026 | 2212 | 2595 | 2860 |

Выделенная память в таблицах указана в байтах.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Стек** | **Размер** | **Количество заполненных элементов (%)** | | | | |
| 10 | 20 | 30 | 40 | 50 |
| Список | 10.000 | 40.000 | 80.000 | 120.000 | 160.000 | 200.000 |
| Массив | 160.000 | | | | |
| Список | 100.000 | 400.000 | 800.000 | 1.200.000 | 1.600.000 | 2.000.000 |
| Массив | 1.600.000 | | | | |
| Список | 1.000.000 | 4000000 | 8000000 | 12000000 | 16000000 | 20000000 |
| Массив | 16.000.000 | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Стек** | **Размер** | **Количество заполненных элементов (%)** | | | | |
| 60 | 70 | 80 | 90 | 100 |
| Список | 10.000 | 240.000 | 280.000 | 320.000 | 360.000 | 400.000 |
| Массив | 160.000 | | | | |
| Список | 100.000 | 2.400.000 | 2.800.000 | 3.200.000 | 3.600.000 | 4.000.000 |
| Массив | 1.600.000 | | | | |
| Список | 1.000.000 | 24000000 | 28000000 | 32000000 | 36000000 | 40000000 |
| Массив | 16.000.000 | | | | |

По приведенным выше таблицам можно сделать следующие выводы:

- По времени выполнения: во всех тестах при любой заполненности стек-массив выигрывает в скорости у стек-списка в 2-3 раза.

- По памяти: выбор способа реализации стека не зависит от его размера. Хранение в виде стека-списка выгоднее при количестве заполненных элементов до 40% от общего.

# Контрольные вопросы

*1. Что такое стек?*

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент, работает по правилу LIFO — Last Input First Output.

*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 этом сначала выделяется память под конкретный указатель, туда записывается адрес предыдущего, затем указатель стека сдвигается на новый, и уже по новому адресу стека записываются данные элемента.

При хранении стека с помощью массива, память выделяется либо в куче, либо на стеке в начале программы, при этом данные записываются последовательно. Для каждого элемента стека, реализованного списком, памяти выделяется больше, чем для элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка.

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

При хранении стека 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. Память из-под массива освобождается только в конце работы программы.

*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?*

Исходя из выполнения лабораторной работы, стек на основе массива выигрывает как в скорости, так и в памяти у стека на основе списка (кроме случаев заполненности стека менее чем на 40%).

Однако в случае реализации перевыделения памяти при переполнении, добавление новых элементов может занять значительное время даже по сравнению со списком. В таком случае есть смысл задуматься о реализации стека на основе списка. Во всех остальных случаях стоит использовать стек на основе массива, так как количество памяти растет быстрее мощности процессоров.

# Вывод

В результате выполнения лабораторной работы я научился работать со стеком и реализовал различные его версии: на основе массива и на основе списка. Стек на основе массива выигрывает как в скорости, так и в памяти у стека на основе списка (кроме случаев заполненности стека менее чем на 40%).  
 Фрагментация данных отсутствует, т.е. при удалении элемента из стека, а затем добавлении, новый элемент будет записан на место удаленного, то есть иметь его адрес.