|  |  |
| --- | --- |
| 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)** |

**Типы и структуры данных.**

**Лабораторная работа №4.**

**«Работа со стеком»**

Студент **Леонов Владислав Вячеславович**

Группа **ИУ7-36Б**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Леонов В.В.

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Силантьева А.В.

*подпись, дата фамилия, и.о.*

*2020 г.*

Оглавление

[Условие и техническое задание 3](#_Toc56581776)

[Цель работы 3](#_Toc56581777)

[Задание 3](#_Toc56581778)

[Входные данные 3](#_Toc56581779)

[Выходные данные 3](#_Toc56581780)

[Обращение к программе 4](#_Toc56581781)

[Функции программы 4](#_Toc56581782)

[Аварийные ситуации 4](#_Toc56581783)

[Реализация 5](#_Toc56581784)

[Структуры данных 5](#_Toc56581785)

[Алгоритм 7](#_Toc56581786)

[Тестирование 8](#_Toc56581787)

[Аварийные ситуации 8](#_Toc56581788)

[Штатное поведение 9](#_Toc56581789)

[Оценка эффективности использования стека на основе вектора и списка 13](#_Toc56581790)

[Время объединения и сортировки результирующего стека (в мс) 13](#_Toc56581791)

[Объем памяти, необходимый для хранения стека 14](#_Toc56581792)

[Контрольные вопросы 15](#_Toc56581793)

[1.Что такое стек? 15](#_Toc56581794)

[2. Каким образом и сколько памяти выделяется под хранение стека при различной его реализации? 15](#_Toc56581795)

[3. Каким образом освобождается память при удалении элемента стека при различной реализации стека? 15](#_Toc56581796)

[4. Что происходит с элементами стека при его просмотре? 15](#_Toc56581797)

[5. Каким образом эффективнее реализовывать стек? От чего это зависит? 15](#_Toc56581798)

[Вывод о проделанной работе 16](#_Toc56581799)

# Условие и техническое задание

## Цель работы

Реализовать операции работы со стеком, который представлен в виде массива (статического или динамического) и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации: получить представление о механизмах выделения и освобождения памяти при работе со стеком.

## Задание

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.

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

## Входные данные

1. **Количество элементов** в стеке и **данные**, хранящиеся в стеке.
2. **Количество элементов** в стекедля автоматической генерации стека.
3. **Выбор стека** (целое число от 0 до 6)для добавления/удаления элементов из стека, а также **значение элемента** в случае добавления.
4. **Опция меню** (целое число от 0 до 9).

## Выходные данные

1. **Стеки** в векторном и списковом представлении**.**
2. **Объединенный отсортированный стек** в двух представлениях.
3. **Время выполнения операции** объединения и сортировки двух стеков**.**
4. **Массив адресов** освобождаемых элементов списка.
5. **Сравнительные таблицы использования времени и памяти** для выполнения операции (см п.3) при различном представлении стека.

## Обращение к программе

Обращение к программе **app.exe** осуществляется через запуск из терминала.

## Функции программы

1. Ввод элементов стека в ручном режиме.
2. Генерация стека в автоматическом режиме для заданного размера.
3. Добавление элемента в желаемый стек.
4. Удаление элемента из желаемого стека.
5. Объединить и отсортировать стеки, используя дополнительный.
6. Печать текущего состояния стеков.
7. Печать адресов освобождаемых элементов стека в виде списка.
8. Сравнение времени для обработки стеков в разных представлениях.
9. Сравнение памяти для обработки стеков в разных представлениях.
10. Завершить работу.

## Аварийные ситуации

1. Некорректная опция меню (буква или значение, несоответствующее ни одному пункту из меню.
2. Некорректный ввод размера или элементов стека (буква или нецелое число).
3. Превышение максимального размера стека.
4. Ошибка выделения памяти операционной системой.

# Реализация

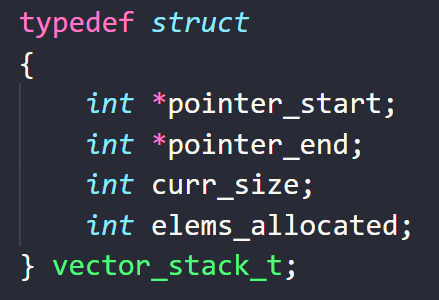
## Структуры данных



В программе максимальный размер стека ограничен константой ***MAX\_STACK\_SIZE.***

Структура **vector\_stack\_t**

Для реализации стека в виде вектора используется агрегированный тип ***vector\_stack\_t.***

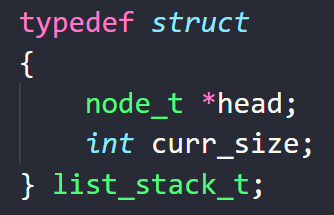


Поле ***pointer\_start/pointer\_end*** представляет собой указатель на начало/конец динамического массива, используемого для хранения данных стека.

Поле ***curr\_size*** представляет собой целое число и служит для хранения текущего размера стека.

Поле ***elems\_allocated*** представляет собой целое число и служит для хранения текущего размера выделенной памяти под вектор.

Структура **list\_stack\_t**

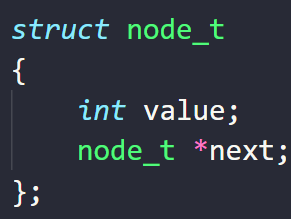
Для реализации стека в виде списка используется агрегированный тип ***list\_stack\_t***.

Поле ***head*** представляет собой указатель и служит для хранения адреса головы списка.

Поле ***curr\_size*** представляет собой целое число и служит для хранения текущего размера стека.

Структура **node\_t**

Для хранения узла списка используется агрегированный тип ***node\_t.***



Поле ***value*** представляет собой целое число и служит для хранения данных в узле списка.

Поле ***next*** представляет собой указатель и служит для хранения адреса следующего узла списка.

## Алгоритм

1. Пользователь вводит номер команды из меню.
2. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия из меню программы.

Для реализации чтения стека используются функция **read\_stack**. Выполняется запрос размерностей с последующим ввод самих элементов. Ведется параллельное заполнение стека в виде вектора и списка (функции **vector\_stack\_push** и **list\_stack\_push)**.

При генерации стека автоматически используется функция **gen\_stack.** Выполняется запрос размерностей с последующей генерацией последовательности случайных чисел.

Для добавления в стек используются функции **vector\_stack\_push** и **list\_stack\_push,** для удаления **vector\_stack\_pop** и **list\_stack\_pop** соответственно.

Для сортировки стеков используется дополнительный стек, в который сливаются два стека (функции **merge\_vector\_stacks** и **merge\_list\_stacks**). А затем происходит сортировка (функции **sort\_vector\_stack** и **sort\_list\_stack**).

Для реализации печати стека в виде вектора и в виде списка используются функции **print\_vector\_stack** и **print\_list\_stack** соответственно, для печати адресов освобождаемых элементов – **print\_free\_mem\_pointers**.

# Тестирование

## Аварийные ситуации

|  |  |  |  |
| --- | --- | --- | --- |
| № | Описание теста | Входные данные | Сообщение |
| 1 | Некорректная опция из меню | Введите опцию: abc  Введите опцию:101 | Некорректный ввод |
| 2 | Некорректный ввод стека | Введите количество  элементов: ывф  Вводите элементы  стека: d.e3e | Ошибка чтения |
| 3 | Превышение размера стека при вводе | Введите количество  элементов: 999999999 | Превышение размера стека |
| 4 | Превышение размера стека при сортировке двух стеков | Стек 1 [4000]  Стек 2 [4000] | Превышение размера стека |

## Штатное поведение

|  |  |  |
| --- | --- | --- |
| № | Описание теста | Демонстрация работы программы |
| 1 | Чтение стека |  |
| 2 | Генерация стека |  |
| 3 | Добавление элемента в стек (стек не переполнен) |  |
| 4 | Добавление элемента в стек (стек переполнен) |  |
| 5 | Удаление элемента из стека (стек не пустой) |  |
| 6 | Удаление элемента из стека (стек пустой) |  |
| 7 | Сортировка стеков  (не пустые) |  |
| 8 | Сортировка стеков  (пустые) |  |
| 9 | Печать текущего состояния стеков (пустые) |  |
| 10 | Печать текущего состояния стеков (не пустые) |  |
| 11 | Печать адресов освобождаемых элементов (без удалений) |  |
| 12 | Печать адресов освобождаемых элементов (после удалений) |  |
| 13 | Сравнение времени обработки стеков при различном представлении |  |
| 14 | Сравнение памяти обработки стеков при различном представлении |  |
| 15 | Завершение работы |  |

# Оценка эффективности использования стека на основе вектора и списка

## Время объединения и сортировки результирующего стека (в мс)

Для усреднения оценки времени и снижения погрешности, связанной с исходным состоянием стека, замеры были проведены многократно (25 раз). Таким образом, приведенное ниже время является средним для 25 операций.

|  |  |  |  |
| --- | --- | --- | --- |
| Размер стека | Список | Вектор | Преимущество использования вектора вместо списка (в раз) |
| 128 | 2 | 0 | ??? |
| 256 | 10 | 1 | ~10 |
| 384 | 22 | 2 | ~11 |
| 512 | 39 | 4 | ~9,75 |
| 640 | 62 | 7 | ~8,85 |
| 768 | 92 | 10 | ~9,2 |
| 896 | 124 | 14 | ~8,85 |
| 1024 | 163 | 18 | ~9,05 |

Исходя из приведенных выше данных можно сделать вывод, что использование вектора вместо списка для построения стека более эффективно, получаем выигрыш по времени порядка 9~10 раз, что довольно существенно.

## Объем памяти, необходимый для хранения стека

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | Объем данных в одной ячейке памяти (байт) | | | | | | | | |  |
| 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 |
| Размер стека (элементов) | 128 | 1280 | 1536 | 2048 | 3072 | 5120 | 9216 | 17408 | 33792 | 65560 | Объем занимаемой памяти (список-вектор) |
| 272 | 528 | 1040 | 2064 | 4112 | 8208 | 16400 | 32784 | 65552 |
| 256 | 2560 | 3072 | 4096 | 6144 | 10240 | 18432 | 34816 | 67584 | 133120 |
| 528 | 1040 | 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 |
| 384 | 3840 | 4608 | 6144 | 9216 | 15360 | 27648 | 52224 | 101376 | 199680 |
| 1040 | 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 |
| 512 | 5120 | 6144 | 8192 | 12288 | 20480 | 36864 | 69632 | 135168 | 266240 |
| 1040 | 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 |
| 640 | 6400 | 7680 | 10240 | 15360 | 25600 | 46080 | 87040 | 168960 | 332800 |
| 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 | 524304 |
| 768 | 7680 | 9216 | 12288 | 18432 | 30720 | 55296 | 104448 | 202752 | 399360 |
| 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 | 524304 |
| 896 | 8960 | 10752 | 14336 | 21504 | 35840 | 64512 | 121856 | 236544 | 465920 |
| 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 | 524304 |
| 1024 | 10240 | 12288 | 16384 | 24576 | 40960 | 73728 | 139264 | 270336 | 532480 |
| 2064 | 4112 | 8208 | 16400 | 32784 | 65552 | 131088 | 262160 | 524304 |

Исходя из приведенных выше данных можно сделать вывод, что при данных меньших размера указателя (8 байт) всегда выгоднее использовать вектор, чем список. Но в то же время в среднем при больших чем указатель данных и количестве элементов, при которых вектор не заполен полностью выгоднее по памяти использовать список (в данной конкретной программе объем памяти вектора пропорционален закону 2^N).

# Контрольные вопросы

## 1.Что такое стек?

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент (верхний элемент). На стек действует правило LIFO — последним пришел, первым вышел.

## 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 хранении с помощью вектора, память выделяется либо в куче, либо на стеке (в зависимости от того, динамический массив или статический). Для каждого элемента стека, реализованного списком, выделяется (4 или 8 байт) больше, чем для элемента вектора. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя (4 или 8 байт) зависит от архитектуры компьютера.

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

При хранении стека связанным списком, верхний элемент удаляется освобождением памяти для него и смещением указателя, указывающего на начало списка. При удалении из стека, реализованного вектором, смещается лишь указатель на вершину стека.

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

При просмотре элементов стека «вытаскивается» элемент, хранящийся на его вершине, и уничтожается.

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

Реализовывать стек эффективнее с помощью вектора. Он выигрывает во времени обработки. При данных меньших размера указателя (8 байт) всегда выгоднее использовать вектор, чем список. Но в то же время в среднем при больших чем указатель данных и количестве элементов, при которых вектор не заполен полностью выгоднее по памяти использовать список.

# Вывод о проделанной работе

В результате лабораторной работы была изучена структура данных – стек, методы ее реализации на основе известных ранее структур (список и вектор), а также особенности обработки подобной структуры данных. Были закреплены навыки работы с динамической памятью.

Согласно полученным экспериментальным путем данных построение стека на основе вектора дает существенный выигрыш по времени обработки информации (порядка 9-10 раз) нежели чем на списке ввиду того, что происходит меньше обращений к памяти, операция которой является очень дорогостоящей по времени. При данных меньших размера указателя (8 байт) всегда выгоднее по памяти использовать вектор, чем список. Но в то же время в среднем при больших чем указатель данных и количестве элементов, при которых вектор не заполен полностью выгоднее по памяти использовать список.

Хочется отметить, что при выделении памяти под список данные не находятся последовательно в памяти (присутствует фрагментация).