|  |  |
| --- | --- |
| **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»**

Студент: Чепиго Дарья Станиславовна

Группа: ИУ7-34Б

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Чепиго Д.С

*подпись, дата фамилия, и.о.*

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

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Условие задачи**

**Вариант 2**

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.

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.

**Техническое задание**

*Входные данные*

- Целое число от 0 до 10 – пункт меню

- Вещественные числа – элементы стеков.

*Допущения:*

- Максимальный размер стека, который может выбрать пользователь – 5000 элементов

*Пункты меню:*

Для массива:

1 - добавить элемент в левый стек

2 - добавить элемент в правый стек

3 - удалить элемент в левом стеке

4 - удалить элемент в правом стеке

5 - посмотреть текущее состояние стеков в массиве

Для списка:

6 - добавить элемент в стек

7 - удалить элемент в стеке

8 - посмотреть текущее состояние списка

9 - посмотреть массив адресов, в которых хранились элементы

10 - посмотреть сравнение реализации двух подходов

0 - выйти из программы

*Выходные данные:*

При выборе пункта меню, связанным с просмотром стеков на экран выводится информация о стеке: количество свободных элементов(для массива), сами элементы. При реализации стека с помощью списка также возможен просмотр адресов, которые используются для элементов в данное время и адресов, которые были использованы элементами, которые позже были удалены.

При выборе меню 10 выводится информация о времени и памяти, которые нужны для реализации стека массивом и списком.

*Действие программы:*

*Реализация стека с помощью списка, реализация двух стеков в одном массиве.*

*Обращение к программе:*

Запускается командой ./app.exe через терминал, находясь в директории, содержащей программу.

*Описание алгоритма*

1. Пользователь выбирает пункт меню

2. В зависимости от выбора пользователь может вводить и удалять элементы в стеках, а также смотреть текущую информацию о стеках и затрачиваемых ресурсах.

3. Стеки, реализованные с помощью массива и стек, реализованный списком не синхронизируются.

4. При вводе первого элемента в любой из стеков, которые реализуются с помощью массива пользователю нужно ввести максимальный размер массива. Поменять его во время программы нельзя.

5. Если пользователь захочет завершить программу, то для этого есть отдельный пункт меню.

*Аварийные ситуации:*

- Выбор несуществующего пункта меню

- Неправильный размер массива с двумя стеками

- Неправильные элементы стека

- Ошибки выделения памяти

- Переполнение стеков

- Работа с пустым стеком

*Описание структур данных*

*Структура для реализации двух стеков в одном массиве:*

*struct arr\_stack*

{

int size\_arr; //размер массива

int count\_left; //последний элемент в левом стеке

int count\_right; //последний элемент в правом стеке

double \*arr; //массив с правым и левом стеке

};

Структуры для реализации стека с помощью списка:

*struct list\_element — структура для одного элемента*

{

double elem; //значение элемента списка

int count; //количество элементов в списке

struct list\_element \*next; //указатель на след ячейку списка

};

struct list\_stack — структура для всего списка

{

struct list\_element \*element;

};

*Сравнение эффективности*

Время — в тактах

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Добавление элементов в массив | Удаление элементов в массиве | Добавление элементов в список | Удаление элементов в списке |
| 10 | 7 | 7 | 4 | 5 |
| 100 | 11 | 14 | 41 | 16 |
| 1000 | 30 | 85 | 153 | 377 |

Память — в байтах

|  |  |  |
| --- | --- | --- |
| Количество элементов | Массив | Список |
| 10 | 200 | 240 |
| 100 | 2000 | 2400 |
| 1000 | 20000 | 24000 |

Выводы из таблицы измерений:

Стек, реализованный связанным списком, проигрывает как по памяти, так и по времени обработки, при размерностях больше 50. При этом по памяти массив занимает всего лишь в 1.2 раза меньше, чем список, так как при моей реализации двух стеков в одном массиве нужно хранить два указателя на начало каждого стека.

Что касается времени:

- до 100 элементов обработка массива быстрее в 1.5 — 2 раза.

-от 100 до 1000 элементов, то обработка массива быстрее в 2 — 4 раза.

-при 1000+ элементов обработка массива быстрее в 5 — 6 раз.

Следовательно при увеличении элементов относительная скорость массива над списком также растёт.

Таким образом, можно сделать вывод, что если нужно реализовать такую структуру данных как стек, то лучше использовать массив, а не связанный список, если количество элементов больше 50.

**Тестирование**

*Позитивные тесты.*

|  |  |  |
| --- | --- | --- |
| Входные данные | Действия программы | Выходные данные |
| Пункт 2  Все элементы массива принадлежат правому стеку | Корректная работа программы | Ожидание следующего ключа |
| Пункт 1  Все элементы массива принадлежат левому стеку | Корректная работа программы | Ожидание следующего ключа |
| Пункт 3 или 4, стек пуст | Информация, что стек пуст | Ожидание следующего ключа |
| Пункт 7 или 8 | Информация, что стек пуст | Ожидание следующего ключа |
| Пункт 2 или 4, после пункт 3 | Удаление последнего элемента из левого/правого стека | Ожидание следующего ключа |
| Пункт 5, после пунктов 1 или 2 | Вывод информации о состоянии массива | Ожидание следующего ключа |
| Пункт 6 | Добавление элементов в список | Ожидание следующего ключа |
| Пункт 7 | Удаление элемента из списка | Ожидание следующего ключа |
| Пункт 8, после пунктов 6 и 7 | Вывод на экран текущее состояние массива | Ожидание следующего ключа |
| Пункт 9 | Вывод на экран свободных адресов | Ожидание следующего ключа |
| Пункт 10  Выбор количества элементов | Вывод на экран сравнение по памяти и по времени для данного количества элементов | Ожидание следующего ключа |

*Негативные тесты*

|  |  |  |
| --- | --- | --- |
| Входные данные | Действия программы | Выходные данные |
| Пункт 1, ввод числа 6000 | Информация о неверном количестве элементов | Завершение программы |
| Пункт 1 - 2, ввод букв | Информация о неверном элементе массива | Завершение программы |
| Пункт 6, ввод букв | Информация о неверном элементе списка | Завершение программы |
| Пункт 12 — неверный ключ | Информация о неверном ключе | Завершение программы |

**Контрольные вопросы**

*Что такое стек?*

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

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

При хранении стека с помощью списка, память выделяется под каждый новый элемент. При хранении с помощью динамического массива, память выделяется под определенное количество элементов и не факт, что вся эта память будет задействована программой.

Для массива необходимо (sizeof(double) + 3 \* sizeof(int))\* n , где n — количество элементов в стеке.

Для каждого элемента стека, реализованного списком, выделяется на sizeof(int) байт больше, чем для элемента массива.

В моем случае стек, реализованный массивом не сильно выигрывает в памяти, так как в нем хранятся указатели на начало правого и левого стека.

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

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

*Что происходит с элементами стека при его просмотре?*

Элементы стека меняются и доступен только один, так как каждый раз достается верхний элемент стека.

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

Реализовывать стек эффективнее с помощью массива. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти. Вариант хранения списка может выигрывать только в том случае, если у нас малое количество элементов или нам нужно занять всю оперативную память элементами в программе(массив, вероятно не сможет так выделить память из-за фрагментации).