| 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

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №4 «Работа со стеком»**

Студент Романов Семен Константинович

Группа ИУ7 – 35Б

Проверено Никульшина Татьяна Александрова

***Описание задачи***

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

1. *массивом*;
2. *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:**

1. **Целое число(номер команды):** целое число от 0 до 9 (см. Функции программы)
2. **Числовые обозначения:** количество чисел в стеке, значения стека

**Выходные данные:**

1. **Результаты работы со стеком:** Количество добавленных элементов, значение удаленного элемента, текущее состояние стека, освобожденные адреса при удалении
2. **Эффективность работы команды:** Затраченное время и память

**Способ обращения к программе:** запускается из терминала командой ./app.exe

**Функции программы:**

1. Добавить числа в первый стек
2. Добавить числа во второй стек
3. Удалить число из первого стека
4. Удалить число из второго стека
5. Вывод обоих стеков

п. 1-5 относятся к стекам, реализованным с помощью массива.

1. Добавить числа в стек
2. Удалить число из стека
3. Показать текущее состояние стека
4. Показать освобожденные адреса
5. Сравнение эффективности массива и связанного списка
6. Выход

**Аварийные ситуации:**

1. Неверный тип данных при вводе числовых значений: Вывод сообщения “Invalid Input”
2. Неверная команда: Сообщение “Unknown command”
3. (Для стека на основе массива) Выход за пределы стека: Вывод оставшихся ячеек и сообщение “Stack overflow”
4. Попытка удаления из пустого стека: Сообщение “Stack already empty”

**Структуры данных:**

**Для стека на основе связанного списка:**

typedef struct node

{

float data;

struct node \*next;

} node\_t;

**Поля структуры:**

* **Float data:** значение конкретного “нода”
* **Struct node \*next:** указатель на предыдущее значение стека (в случае первого элемента – указатель NULL)

**Адреса свободных областей (на основе связанного списка):**

typedef struct frmem

{

long address;

struct frmem \*next;

} frmem\_t;

**Поля структуры:**

* **Long address:** значение освободившегося адреса
* **Struct frmem \*next:** указатель на предыдущее расположение адреса (в случае первого элемента – указатель NULL)

**Для двух стеков на основе одного массива;**

**float \*array, \*stack\_a\_array, \*stack\_b\_array;**

**Размер массива size:** указывается при начале программы, перевыделения памяти не происходит.

1. **float \*array:** указатель на массив
2. **float \*stack\_a\_array:** указатель на первый стек
3. **float \*stack\_b\_array:** указатель на второй стек

**В случае пустого стека:**

1. указатель на первый стек становится равен array – 1 (левее начала массива)
2. указатель на второй стек становится равен array + size (правее конца массива)

**Алгоритм:**

1. При вводе стек-массива, будь то первого или второго, вычисляется количество свободных элементов (stack\_b\_array – stack\_a\_array - 1) – и если их количество не меньше количества вводимых, то производится поэлементный ввод со сдвигом указателя
2. При выборе команды удалить элемент из стек-массива, элемент удаляется путем сдвига указателя
3. При вводе стек-списка, сначала выделяется память под конкретный “нод”, туда записывается адрес предыдущего, затем указатель стека сдвигается на новый “нод”, и уже по новому адресу стека записываются данные элемента.
4. При выборе команды удалить элемент из стек-списка, указатель на элемент записывается во временную переменную, указатель стека перемещается на предыдущий элемент и по указателю временной переменной производится освобождение памяти.
5. При выборе команды вывода массива свободных областей, выводится массив свободных областей в том случае, если какие-либо элементы были удалены из стека.
6. Пользователь выполняет действия с таблицей, пока не введет команду 0 (Выход)

**Тесты**

|  |  |  |  |
| --- | --- | --- | --- |
| N | Test | Input | Output |
| 1 | Некорректная комманда | 11 | “Unknown command” |
| 2 | Некорректная комманда | Abc | “Invalid Input” |
| 3 | Некорректное количество вводимых значений | -1 | “Invalid Input” |
| 4 | Некорректный ввод значений | 5 4 a | “Number of stack'd elements: 2  Invalid Input” |
| 5 | Удаление из пустого стэка | **Удаление** (стэк пустой) | “Stack already empty” |
| 6 | (При реализации стэка массивом) Переполнение стэка | *Input numbers count*: 1 | “Free space remaining: 0 element(s)  STACK OVERFLOW!” |
| 7 | Корректное добавление в стэк | 1 2 3 | “Number of stack'd elements: 3” |
| 8 | Корректное удаление из стэка | **Удаление** | Deleted element: Верхнее значение стека |

**Оценка эффективности:**

Все время измеряется в clocks

**Время добавления элементов:**

|  |  |  |
| --- | --- | --- |
| Size | Array | List |
| 100 | 19 | 27 |
| 1000 | 69 | 88 |
| 10000 | 556 | 691 |

**Время удаления элементов:**

|  |  |  |
| --- | --- | --- |
| Size | Array | List |
| 100 | 20 | 48 |
| 1000 | 76 | 251 |
| 10000 | 621 | 2163 |

**Занимаемая память в байтах:**

|  |  |  |  |
| --- | --- | --- | --- |
| Size | Количество элементов от размера массива | Array | List |
| 100 | 20% | 400 + 16 | 320 |
| 40% | 640 |
| 60% | 960 |
| 80% | 1280 |
| 100% | 1600 |
| 1000 | 20% | 4000 + 16 | 3200 |
| 40% | 6400 |
| 60% | 9600 |
| 80% | 12800 |
| 100% | 16000 |
| 10000 | 20% | 40000 + 16 | 32000 |
| 40% | 64000 |
| 60% | 96000 |
| 80% | 128000 |
| 100% | 160000 |

Таким образом, список является более эффективным по памяти решением, чем массив, только при условии заполненности массива не более чем на 25%

**Контрольные вопросы**

1. **Что такое стек?**

Стек – структура данных, в которой можно обрабатывать только последний добавленный элемент (верхний). На стек действует правило LIFO — “Last Input, First Output” (“Последним пришел, первым вышел”).

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

При хранении стека с помощью списка, то память всегда выделяется в куче, при этом сначала выделяется память под конкретный “нод”, туда записывается адрес предыдущего, затем указатель стека сдвигается на новый “нод”, и уже по новому адресу стека записываются данные элемента. При хранении с помощью массива, память выделяется либо в куче, либо на стеке в начале программы (в зависимости от того, динамический массив или статический), при этом данные записываются последовательно. Для каждого элемента стека, реализованного списком, выделяется на 8 байт больше, чем для элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя зависит от архитектуры.

1. **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 стека. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека. Память из под массива освобождается в конце работы программы.

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

Элементы стека уничтожаются, так как каждый раз достается верхний элемент стека.

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

Если говорить о быстроте обработки и количестве занимаемой памяти, то реализовывать стек эффективнее с помощью массива при условии отсутствия перевыделения памяти. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти. Однако в случае реализации перевыделения памяти при переполнении, то добавление новых элементов может занять значительное время даже по сравнению со связанным списком. Также вариант хранения списком может выигрывать в том случае, если заполненность массива является небольшой относительно размера самого массива. Также память для списка ограничена лишь размером оперативной памяти (так как память выделяется в куче).

**Вывод**

Стек, реализованный связанным списком, внушительно проигрывает как по времени, так и по памяти в данной реализации в случае полной заполненности массива, однако при заполненности меньше 25%, список является более эффективным по памяти решением. Таким образом, можно сделать вывод, что если нужно реализовать такую структуру данных как стек, то лучше использовать массив, а не связанный список. Однако всегда нужно держать в голове тот факт, что массив обладает ограничениями, в то время как связанный список – нет