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

**Дисциплина “Типы и структуры данных”**

**Лабораторный практикум №3**

**по теме: «обработка разреженных матриц»**

Выполнил студент: \_\_*Клименко Алексей Константинович\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*фамилия, имя, отчество*

Группа: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*ИУ7-35Б*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Проверил, к.п.н.: **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

*подпись, дата*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 г.*

**цель работы**

реализовать операции работы со стеком, который представлен в виде массива (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.

**описание условия 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.

**техническое задание**

**исходные данные**

Исходными данными являются числа, запрашиваемые у пользователя в программе, и помещаемые в стеки.

*Примеры вставки и удаления чисел из двух стеков:*

|  |  |  |
| --- | --- | --- |
| **Ввод пользователя** | **Стек a** | **Стек b** |
| push a 1 2 3  pop a  push 2 5  push B 3 | (1,2,3)  (1,2)  (1,2)  (1,2) | ()  ()  (5)  (5,3) |

**результат**

Результатом работы программы являются три стэка, два из которых содержат введёные пользователем числа, а третий является отсортированным объединением чисел из двух других стеков.

**описание задачи, реализуемой программой**

Программа решает задачу слияния с одновременной сортировкой элементов из двух стеков в третий. По мере работы программы можно запросить у неё вывод текущего состояния на экран командой **show**.

**способы обращения к программе**

Для запуска программы необходимо запустить файл **app.exe**.

**возможные аварийные ситуации и ошибки пользователя**

При неверном вводе команды программа попросит ввести команду снова, а при невозможности выполнения команды (например, извлечение из пустого стека) сообщает об этом, и предлагает ввести другую команду.

**Структуры данных**

Для реализации стека на массиве была выбрана следующая структура:

**struct stack\_arr**

**{**

**uint32\_t size;** // размер стека

**uint32\_t \_\_capacity;** // вместимость стека

**int32\_t \*\_\_data;** // массив данных стека

**int32\_t \*top;** // указатель на вершину стека

**};**

При хранении в стеке N элементов объем памяти будет:

**sizeof(stack\_arr) == (24 + 4N)Б**

Реализация стека на связном списке:

**struct \_\_st\_lst\_node**

**{**

**int32\_t data;** // данные узла списка

**struct \_\_st\_lst\_node \*prev;** // указатель на предыдущий узел

**};**

**struct stack\_lst**

**{**

**uint32\_t size;** // размер стека

**struct \_\_st\_lst\_node \*top;** // указатель на вершину стека

**};**

При хранении в стеке **N** элементов объем памяти будет:

**sizeof(stack\_lst) == (12 + 12N)Б**

Структура стэка с параметризированной реализацией (универсальная):

**typedef int \_\_stack\_imp\_type;**

**#define STACK\_TYPE\_INVALID      0**

**#define STACK\_TYPE\_ARRAY        1**

**#define STACK\_TYPE\_LINKED\_LIST  2**

**struct stack**

**{**

**\_\_stack\_imp\_type \_\_type;** // тип реализации

**union**

**{**

**struct stack\_arr stack\_arr;** // реализация на массиве

**struct stack\_lst stack\_lst;** // реализация на связном списке

**} data;**

**};**

**Набор функций**

// Создаёт новый стек с заданной реализацией.

**struct stack st\_create(\_\_stack\_imp\_type type);**

// Удаляет стек, освобождая память.

**void st\_destroy(struct stack \*st);**

// Классические операции вставки и удаления из вершины.

**int st\_push(struct stack \*st, int32\_t value);**

**int st\_pop(struct stack \*st, int32\_t \*value);**

// Возвращает размер стека - кол-во хранящихся элементов.

**uint32\_t st\_get\_size(const struct stack \*st);**

// Сливает два стека в один с упорядочиванием последнего.

**int st\_merge(struct stack \*st\_out, struct stack \*st\_a, struct stack \*st\_b);**

**Описание алгоритма обработки данных**

Объединение двух стеков в один состоит из последовательного добавления в конечный стек элементов из обоих начальных:

**пока** есть элементы в начальных стеках:

**выбрать** начальный стек с наибольшим элементом в вершине (s1).

**извлечь** элемент (е) с вершины выбранного стека (s1).

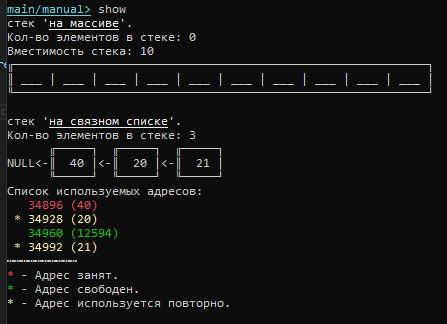
**пока** элемент в вершине конечного стека больше (е):

**переложить** вершину конечного стека в (s2)

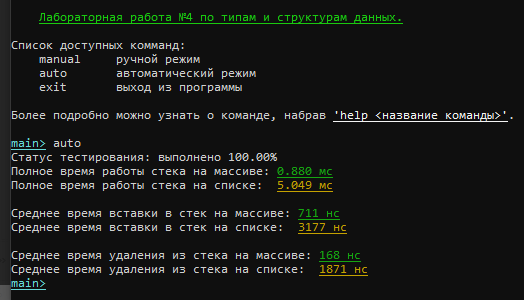
**вставить** элемент (е) в вершину конечного стека.

**пока** элемент в вершине конечного стека неменьше (е):

**переложить** вершину стека (s2) в конечный стек.

****

Результаты автоматизированного тестирования времени вставки и удаления элементов из стеков:

****

Как видно, эффективность реализации на массиве составляет в среднем **82%**. Для операции вставки эффективность составляет **78%**, а для удаления **91%**.

**набор функциональных тестов**

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Описание теста** | **Входные данные** | **Выходные данные** |
| 1 | Некорректная команда | a | Сообщение о неверной команде. Возврат в меню |
| 2 | Показ пустых стеков | manual A L A  show | Вывод информации о стеках на экран |
| 3 | Вставка элементов в стек | manual A L A  push A 10 20 30  show A | Вывод стека (10, 20, 30) на экран |
| 4 | Удаление элементов из стека | manual A L A  push A 10 20  pop A  pop A  show | Вывод пустого стека |
| 5 | Вызов автоматического тестирования | auto | Вывод временных параметров работы стека при разных реализациях |

**Выводы по проделанной работе**

По окончании работы я убедился, что связные списки для реализации небольших стеков совсем не подходят, уступая по эффективности стекам на массивах. Единственным плюсом стека на списке можно считать распределённость в памяти - нет нужды перевыделять большие фрагменты памяти для добавления новых элементов.

**Контрольные вопросы**

1. *Что такое стек?*

Стек - это абстрактная структура данных, имеющая операции вставки и удаления элементов стека. Работает по принципу LIFO - последний пришёл - первый вышел. (работаем только с вершиной стека)

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

При реализации стека на массиве память выделяется единожды в момент инициализации и впоследующем при нехватке свободного места в стеке.

При реализации стека на связном списке память выделяется каждый раз при добавлении нового элемента в стек.

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

При реализации стека на массиве память очищается при необходимости сжать стэк (уменьшить его вместимость до действительного размера), но этого можно не делать при частых и сбалансированных вставках и удалениях.

При реализации стека на связном списке память очищается каждый раз при удалении элемента из стека.

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

В классической реализации стека просмотр (как элементарное действие над стеком) невозможен. Для просмотра стека необходимо использовать дополнительную структуру данных (не обязательно другой стек), перемещая в неё последовательно все элементы из вершины стека. Тогда, складывая элементы обратно в стек, мы будем наблюдать их порядок при добавлении.

5. *Каким образом эффективнее реализовывать стек?*

Эффективнее реализовывать стек на массиве.