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: Раужев Павел Павлович

Группа: ИУ7-33Б

Студент _____ Раужев П. П.
подпись, дата

Преподаватель _____ Силантьева А. В.
подпись, дата

Оценка _____

2022 г

Описание условия задачи

Вариант 8

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.

Вариант 8. Ввести целые числа в 2 стека. Используя третий стек отсортировать все введенные данные

Техническое задание

Запуск программы производится из терминала. В случае ввода из файла, необходимо заранее прописать в файле последовательность команд (в том числе команду завершения программы) и перенаправить ввод из файла в программу.

Листинг 1: Пример запуска программы из коренной папки проекта

```
rauzh@mbp-pavel ~/D/P/d/lab_04> ./app.exe
rauzh@mbp-pavel ~/D/P/d/lab_04> ./app.exe < data/test_500_in.txt
```

Входные данные:

1. Пункт меню: число от 0 до 5.
2. Размерность стеков: натуральные числа
3. Элементы стеков: целые числа

Выходные данные:

1. Содержание стека-массива
2. Содержание стека-списка
3. Адреса элементов стека-списка

4. Содержание отсортированного стека-массива
5. Содержание отсортированного стека-списка
6. Адреса элементов отсортированного стека-списка

Функции программы:

0. Выйти
1. Добавить элементы в стеки с клавиатуры
2. Удалить элементы из стеков по введённому количеству
3. Вывести текущее состояние стеков
4. Отсортировать два стека в третий
5. Сравнить методы хранения и обработки стека по времени и памяти

Обращение к программе:

Запускается из терминала.

Аварийные ситуации:

- Выбор неверного пункта меню: Вход — неверный пункт меню. Выход — сообщение об ошибке.
- Некорректный ввод числовых данных: Вход: строка вместо числа. Выход — сообщение об ошибке.
- Попытка провести операцию с пустым стеком. Вход — пустой стек. Выход — сообщение об ошибке.
- Добавление элемента в заполненный стек. Выход — сообщение об ошибке.

Структуры данных

Для хранения стеков используются следующие структуры:

Стек-массив:

data — массив значений, **size** — число элементов, **top** — индекс «головы» стека

Листинг 2: Линейный односвязный список

```
typedef struct
{
    size_t size;
    size_t top;

    int *data;
} arr_stack_t; // стэк-массив
```

Стек-список:

data — элемент стека, ***next** — указатель на следующий элемент.

Листинг 3: Структура для хранения матрицы в табличном виде

```
typedef struct node_t

{
    int data;

    struct node_t *next;
} node_t; // список
```

Массив указателей на свободные элементы списка:

Листинг 4: Структура для хранения матрицы в разреженном виде

```
typedef struct
{
    node_t *free_arr_ptrs[MAX_LEN_ARR_PTRS];

    int len;
} free_ptrs_t; // массив указателей на освобождённые элементы стека
```

Алгоритм

Реализовано два способа хранения стека: с помощью списка и с помощью массива. При хранении стека с помощью списка на программиста «наложены ограничения» так, что формально доступ предоставлен только к первому элементу, т. е. «голове» списка, который является последним помещенным элементом стека, чтобы соблюдать принцип LIFO. В случае хранения стека с помощью массива доступ аналогично предоставлен только последнему элементу массива.

В программе воспроизведена сортировка двух стеков в третий: снимется верхний элемент первого стека, верхний элемент второго стека (если возможно), они сравниваются и наибольший из них помещается в результирующий стек на определенное место (в ходе этого процесса из результирующего стека «вытаскиваются» значения вплоть до нужного и временно помещаются в один из исходных стеков), а наименьший помещается в исходный стек. Таким образом, данный вид сортировки не использует дополнительной памяти.

Описание функций, используемых в программе

1. node_t *list_element_init(const int value)
   Инициализация элемента стека-списка
2. int list_stack_push(node_t **head, const int value)
   Добавление элемента на вершину стека-списка
3. int list_stack_pop(node_t **head)
   Удаление элемента с вершины стека-списка
4. int list_stack_peek(node_t *head)
   Получение значения вершины стека-списка
5. int list_stack_empty(node_t *head)
   Проверка пустоты стека-списка
6. size_t list_stack_size(node_t *head)
   Подсчёт числа элементов стека-списка
7. void list_stack_free(node_t **head)
   Освобождение памяти из-под стека-списка
8. int list_stack_copy(node_t **head, node_t **head_copy, node_t **head_temp)
   Копирование стека-списка в другой
9. size_t list_stack_counter_el(node_t *head, const int element)
   Подсчёт количества элементов, равных заданному, в стеке-списке
10. int list_stack_max_el(node_t **head, node_t **sorted_head)
    Помещение максимума из одного списка в другой список
11. int list_stack_sort(node_t **head_1, node_t **head_2, node_t **head_res)
    Сортировка стека-списка

Далее приведены функции, аналогичные предыдущим, но для стека-массива.

12. int array_stack_init(arr_stack_t *stack);
13. int array_stack_push(arr_stack_t *stack, const int value);
14. int array_stack_pop(arr_stack_t *stack);
15. int array_stack_peek(arr_stack_t *stack);
16. int array_stack_empty(arr_stack_t *stack);
17. int array_stack_copy(arr_stack_t *stack, arr_stack_t *stack_copy, arr_stack_t *stack_temp);
18. size_t array_stack_counter_el(arr_stack_t *stack, const int element);
19. int array_stack_max_el(arr_stack_t *stack, arr_stack_t *sorted_stack);
20. int load_stacks(node_t **head_1, node_t **head_2, arr_stack_t *array_stack_1, arr_stack_t *array_stack_2);
    Ввод стеков с клавиатуры и их загрузка в память компьютера
21. void print_stacks(node_t **head_1, node_t **head_2, arr_stack_t *array_stack_1, arr_stack_t *array_stack_2, node_t **head_res, arr_stack_t *array_stack_res);
    Вывод стеков на экран
22. int del_stacks(node_t **head_1, node_t **head_2, arr_stack_t *array_stack_1, arr_stack_t *array_stack_2);
    Удаление элементов из стеков
23. int efficiency_result(node_t **head_1, node_t **head_2, arr_stack_t *array_stack_1, arr_stack_t *array_stack_2, node_t **head_res, arr_stack_t *array_stack_res);
    Замер эффективности двух методов хранения и обработки стека.

Тесты

Таблица 1: Негативные тесты

№	Тип теста	Описание	Входные данные	Результат
1	Негативный	Тест меню: неправильный пункт	999	Ошибка
2	Негативный	Тест меню: неправильный пункт	sdf	Ошибка
3	Негативный	Выбор стека	Fef	Ошибка
4	Негативный	Ввод элемента стека	F0	Ошибка
5	Негативный	Ввод числа элементов стека	-2	Ошибка
6	Негативный	Попытка выполнить операцию над стеками без предварительного их заполнения	-	Ошибка

Таблица 2: Позитивные тесты

№	Тип теста	Описание теста	Входные данные	Результат
1	Позитивный	Тест меню	1 3	Ввод двух стеков с клавиатуры
2	Позитивный	Тест меню	3	Вывод двух (или трех) стеков в виде стека-массива и стека-списка
3	Позитивный	Ввод стеков	4 1 2 3 4 5 2 3 4 5 6	Записано два стека: 4 3 2 1 6 5 4 3 2
4	Позитивный	Тест меню	5	Вывод результатов сравнения эффективности методов хранения и обработки стек
5	Позитивный	Сортировка стеков	4 3 2 1 6 5 4 3 2	6 5 4 4 3 3 2 2 1
6	Позитивный	Сортировка стеков	1 1 1 2 1 1 1	2 1 1 1 1 1 1

Замеры времени и памяти

Таблица 3: Сравнение эффективности сортировки стека-массива и стека-списка

Размер стеков (первый второй)	Стеки-массивы		Стеки-списки	
	Время (мкс)	Память (Б)	Время (мкс)	Память (Б)
4 5	2	60	5	144
15 15	8	144	28	480
100 100	273	812	686	3152
500 500	3990	4024	13186	16000

Замеры времени были выполнены 100 раз, затем были усреднены.

Можно сделать вывод, что стек-массив всегда выигрывает по скорости работы и по выделяемой памяти под хранение стека. В среднем, стек-список требует почти в 4 раза больше памяти. При небольшом размере стека стеки-списки медленнее стеков-массивов немного больше, чем в 2 раза, однако при возрастании (уже при размерности в 100 элементов) размеров, возрастает и различия в скорости. Так, при 100 элементах стека, стек-список медленнее уже почти в 3 раза.

Контрольные вопросы

1. **Что такое стек?**

Стек - структура данных с ограниченным доступом: обычно реализуется с помощью массива или списка таким образом, что доступ на запись и чтение предоставлен только последнему добавленному элементу (LIFO — last in first out)

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

Стек-список хранится в куче, стек-массив: статический — на стеке, динамический — в куче. В случае хранения стека-списка хранятся независимые указатели на области памяти, в которых находятся элементы стека. В таком случае, затрачивается память на указатели (4 или 8 байт) и на сами элементы. В случае хранения стека-массива хранится натуральное число — количество элементов стека, т. е. индекс его «вершины» в массиве. Массив хранит последовательно элементы стека в памяти подряд один за другим. Получается, что стек-массив требует меньше памяти, чем стек-список.

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

В случае стека-массива просто освобождается указатель на сам массив значений элементов стека. В случае стека-списка последовательно освобождается каждый узел списка — элемент стека.

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

Элементы стека удаляются, т. к. доступна для просмотра только вершина стека.

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

Эффективнее реализовывать его в виде массива, так как в таком случае затрачивается меньше памяти и алгоритмы обработки стека работают быстрее.

Вывод

Научился реализовывать хранение и обработку стека в виде массива и списка. На примере сортировки узнал, что стек-список требует примерно в 4 раза больше памяти, чем стек-массив. Просматривая адреса элементов стека-списка при удалении и при просмотре значений определил, что повторно участки памяти не используются. Фрагментации памяти не замечено, так как, судя по адресам памяти, все объекты программы хранились в памяти последовательно в определенном участке памяти. Можно сделать вывод, что лучше хранить стек в виде массива, чем в виде списка.