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

По дисциплине «Типы и структуре данных»

Название: Работа со стеком

Студент Дремин Кирилл

Группа ИУ7 – 36Б

Тип лабораторной работы: Учебная

Вариант №7

Преподаватель Барышникова Марина Юрьевна

2022 г.

Содержание

Описание условия задачи

Реализовать операции работы со стеком, представленным в виде массива и односвязного линейного списка, оценив преимущества и недостатки каждой реализации (механизмы работы с памятью и стеком).

Описание ТЗ

Описание исходных данных и результатов:

Программа реализует коллекцию однотипных объектов, работающую по принципу LIFO, или стек. Предоставляются операции добавления элемента в стек, удаления элемента из стека, а также расчёта его длины, выделения и освобождения памяти.

Способ обращения к программе

Обращение происходит посредством вызова заранее скомпилированного файла и ввода через консоль.

Описание входных данных

Программа позволяет создать стеки, основанные на массиве и списке, произвести с ними основные операции и измерить эффективность операций над ними.

```
1. Создать стек на основе списка заданного размера
2. Создать стек на основе массива заданного размера
3. Создать стек на основе массива заданного размера (заполнить случайно)
4. Произвести замеры эффективности стеков (оба стека должны быть заданы, операция опустошит их!)
5. Добавить элемент в стек на основе списка
6. Добавить элемент в стек на основе массива
7. Удалить элемент из стека на основе списка
8. Удалить элемент из стека на основе массива
9. Просмотреть информацию о стеке на основе списка
10. Просмотреть информацию о стеке на основе массива
0. Выход из программы
```

Корректный ввод - цифра от 0 до 10.

Описание возможных аварийных ситуаций и ошибок пользователя:

1. Ввод некорректных данных.
2. Ошибки выделения динамической памяти.
3. Удаление элемента из пустого стека.

Описание внутренних СД:

Программа оперирует АТД, описывающим операции над стеком:

```
#define stack_def(subname, type) \
typedef struct stack_name(subname, type) stack_name(subname,
type); \
// Добавить элемент в стек\
err_t stack_push(subname, type) (stack_name(subname, type)
*stack, type element); \
// Удалить элемент из стека\
err_t stack_pop(subname, type) (stack_name(subname, type)
*stack, type *result);   \
// Выделить память под стек\
stack_name(subname, type) *stack_alloc(subname, type)();
\
// Получить длину стека\
size_t stack_len(subname, type) (stack_name(subname, type)
*stack);\
// Освободить память, выделенную под стек\
void stack_free(subname, type)(stack_name(subname, type) *
stack);  \
// Просмотреть верхний элемент стека\
err_t stack_peek(subname, type)(stack_name(subname, type)
*stack, type *result); \
// Получить размер стека в байтах\
size_t stack_sizeof(subname, type)(stack_name(subname, type)
*stack);
```

Данный АТД является полиморфным по названию (используется для определения механизма работы) и по типу хранимых данных. Также описанный

стек подразумевается, как не владеющий - он не производит освобождения памяти, выделенной под свои элементы, при своём освобождении.

Сравнение различных реализаций стека

В ходе замеров замерялась эффективность трёх операций: измерения длины, пары pop/push одного элемента и полной очистки. Также измерено потребление динамической памяти.

10000 замеров для длины, 10000000 замеров для pop/push, размеры обоих стеков - 100000 элементов, результат:

Тип	Длина (total)	pop/push (total)	Длина (av)	pop/push (av)	Очистка	Память
Массив	2 ms	489 ms	0.000200	0.00004890	1 ms	800024 bytes
Список	6269 ms	742 ms	0.626900	0.00007420	2 ms	1600016 bytes

10000 замеров для длины, 10000000 замеров для pop/push, размеры обоих стеков 500000, результат:

Тип	Длина (total)	pop/push (total)	Длина (av)	pop/push (av)	Очистка	Память
Массив	1 ms	338 ms	0.000100	0.00003380	3 ms	4000024 bytes
Список	35659 ms	502 ms	3.565900	0.00005020	9 ms	8000016 bytes

Вывод

Исходя из полученных результатов, можно сделать вывод, что эффективность реализации стека на основе массива гораздо выше, чем эффективность стека, основанного на связном списке, особенно при выполнении операций, для которых необходима обработка всего стека. Операции pop/push эффективнее в среднем на 50%, операция получения длины фактически не зависит от размера стека в

реализации на основе массива и линейно зависит от длины стека для реализации на основе списка**.**

Ответы на вопросы

1. ***Что такое стек?***

Стек - структура данных, обеспечивающая их хранение и доступ вида LIFO (последний вошёл - первый вышел).

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

Под обычную матрицу выделяется n * m * sizeof(тип данных матрицы) байт памяти, где n - количество строк, m - количество столбцов матрицы, то есть память выделяется под все элементы. Память под хранение разреженной матрицы выделяется только для информации о ненулевых элементах. Для CSC обозначим число ненулевых элементов за e, тогда объём требуемой памяти: e * (sizeof(size_t) + sizeof(тип данных матрицы)) + m (кол-во столбцов) * sizeof(size_t).

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

Реализация через массив - память при удалении элемента стека не высвобождается Реализация через односвязный список - освобождается память под узел списка, содержащий элемент.

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

При просмотре стека элементы из него удаляются, при использовании метода pop и не удаляются, при использовании метода push.

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

Более эффективной является реализация стека на основе массива, т.к. не требуестack_shrink(void_ptr)(arr_stack); size_t arr_size = stack_sizeof(arr, void_ptr)(arr_stack); size_t list_size = stack_sizeof(list, void_ptr)(list_stack);т частого

выделения динамической памяти и занимает меньше памяти. Также адресная арифметика является более эффективной для обработки данных, чем работа с узлами списка, что позволяет получить прирост к скорости.