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 №04 ПО ДИСЦИПЛИНЕ: ТИПЫ И СТРУКТУРЫ ДАННЫХ

Работа со стеком.

Студент **Батуев А.Г.**

Группа **ИУ7-36Б**

Вариант **8**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Батуев А.Г.**

Преподаватель _____ **Никульшина Т.А.**

2024 г.

Условие задачи

Цель работы: реализовать операции работы со стеком, который представлен в виде статического массива (при желании и динамического) и в виде односвязного списка, оценить преимущества и недостатки каждой реализации, получить представление о механизмах выделения и освобождения памяти при работе с динамическими структурами данных.
Вариант 8: ввести целые числа в 2 стека. Используя третий стек отсортировать все введенные данные.

Техническое задание

На вход **программа получает**:

- опция из меню (цифра от 0 до 6)
- 2 стека
- значения для хранения в стеке

Программа выводит:

- текущее состояние стека
    - для стека, реализованного списком, дополнительно выводятся адреса удаленных элементов
- отсортированный стек, состоящих из всех элементов из 1 и 2 стека
- результаты автоматического тестирования по добавлению, удалению, сортировке

Программа должна реализовывать создание, хранение, вывод стека, а также операции добавления, удаления элемента из стека и сортировку объединенного стека.

Обращение к программе осуществляется по указанию названия программы (./app.exe).

Необходимо внимательно **обработать возможные аварийные ситуации**, которые могут включать:

- неправильный ввод данных
- ошибка выделения памяти
- переполнение стека
- удаление из пустого стека

Описание внутренних структур данных

stack_array_t представляет стек, хранящий элементы в массиве фиксированного размера values. Эта структура включает:

- int values[STACK_SIZE] — массив фиксированного размера, где хранятся значения стека.
- int *ep (end pointer) — указатель на конец стека, который может указывать за пределы массива.
- int *sp (stack pointer) — указатель на текущий элемент (вершину) стека. Он изменяется при добавлении или извлечении элементов из стека.

stack_list_t представляет узел связного списка, используемого для хранения элементов стека в связном списке. Она включает:

- int val — значение текущего элемента стека.
- struct stack_list *next — указатель на следующий элемент стека. Это поле позволяет связать узлы в односвязный список, что позволяет динамически добавлять или удалять элементы стека.

type_stack определяет типы стека. Оно может принимать два значения:

- ARRAY — указывает, что стек представлен массивом.
- LIST — указывает, что стек представлен связным списком.

deleted_node_t хранит информацию об удаленных узлах списка. Поля:

- void *address — указатель на адрес удаленного узла. Поле void * делает его универсальным для хранения любых типов указателей.
- struct deleted_node *next — указатель на следующий элемент в списке удаленных узлов.

stack_t — это основная структура данных, представляющая стек. Поля:

- type_stack type — указывает текущий тип стека (ARRAY или LIST).
- union stack_u — объединение для хранения одного из типов стека:
- stack_array_t array_stack — используется, если стек реализован на массиве.
- stack_list_t *list_stack — указатель на вершину стека, если стек реализован на связном списке.
- deleted_node_t *deleted — указатель на голову списка удаленных узлов для связного

```
typedef struct
{
    int values[STACK_SIZE];
    int *ep;
    int *sp;
}stack_array_t;

typedef struct stack_list
{
    int val;
    struct stack_list *next;
} stack_list_t;

typedef enum
```

```
{
    ARRAY = 1,
    LIST
}type_stack;

typedef struct deleted_node
{
    void *address;
    struct deleted_node *next;
} deleted_node_t;

typedef struct
{
    type_stack type;
    union
    {
        stack_array_t array_stack;
        stack_list_t *list_stack;
    }stack_u;
    deleted_node_t *deleted;
}stack_t;
```

Основные функции программы

Программа реализует следующие основные функции:

- stack_t *stack_init(int type); Создает и инициализирует стек указанного типа (массив или связный список), возвращая указатель на структуру stack_t.
- void switch_stack(stack_t **stack1, stack_t **stack2); Меняет указатели двух стеков, позволяя переключить их содержимое.
- void push_arr(stack_t *stack, int value); Добавляет элемент в стек в виде массива по значению элемента.
- int *pop_arr(stack_t *stack); Удаляет последний добавленный элемент из стека в виде массива. Возвращает указатель на хранившееся значение.
- void push_list(stack_t *stack, int value, stack_list_t *node); Добавляет элемент в стек, в виде списка, по значению или сразу по адресу элемента.
- stack_list_t *pop_list(stack_t *stack); Удаляет последний добавленный элемент из стека, в виде списка. Возвращает указатель на него
- void sort_list(stack_t *stack1, stack_t *stack2, stack_t *sorted); Сортирует стек, реализованный на связном списке (stack_list_t), упорядочивая элементы по возрастанию.
- void sort_arr(stack_t *stack1, stack_t *stack2, stack_t *sorted); Аналогично sort_list, только для массивной реализации.
- void measure(void); Измеряет время добавления, удаления, сортировки в стеки с разной реализацией.
- void choice(char option, stack_t **stack1, stack_t **stack2); Функция меню
    - 1. Сменить рабочий стек
        - Если новый стек еще не был проинициализирован, то он инициализируется
    - 2. Добавить элемент в стек
    - 3. Удалить элемент из стека
    - 4. Отобразить состояние стека
    - 5. Сортировать
        - Работает, только тогда, когда оба стека проинициализированы и имеют одинаковые виды представления.
    - 6. Автотестирование
    - 0. Выход

Описание основных алгоритмов

Для стека-массива:

- **Добавление** (push_arr и add_to_sarray): при добавлении значения указатель sp (вершина стека) увеличивается, и новое значение записывается по этому адресу. Если достигается граница массива (ep), добавление прекращается с сообщением о переполнении.
- **Удаление** (pop_arr и delete_from_sarray): при удалении sp уменьшается, указывая на предыдущий элемент. Если sp совпадает с базой массива (bp), то стек пуст, и удаление прекращается с предупреждением.

Для стека-связного списка:

- **Добавление** (push_list и add_to_slist): создается новый узел со значением, который становится вершиной стека и указывает на предыдущий элемент.
- **Удаление** (pop_list и delete_from_slist): вершина стека перемещается на следующий элемент, а удаленный узел сохраняется в списке удаленных узлов для возможного восстановления или анализа. Если стек пуст, удаление прекращается с предупреждением.

Алгоритм сортировки идентичен для обоих реализация стеков, меняются только специфические функции для работы с определенным типом стека.

1. Элементы из второго стека помещаются в первый.
2. Элементы из первого стека начинают помещаться во второй, кроме найденного минимального элемента
3. Если находится новый минимальный элемент, старый минимальный помещается во второй, новый минимальный не помещается
4. Найденный минимальный элемент помещается в 3 стек.
5. Повторяются пункты 2-4 пока 1 и 2 стек не окажутся полностью пусты (то есть обработаны все элементы)

Набор тестов

Позитивные тесты:	Входные данные	Выходные данные
Добавление элементов в стек (массив)	ARRAY элементы: 1, 2, 3	Стек содержит: 1, 2, 3
Добавление элементов в стек (список)	LIST элементы: 5, 6, 7	Стек содержит: 5, 6, 7
Удаление элементов из стека (массив)	ARRAY элементы: 3, 2, 1	Элементы удалены, стек пуст
Удаление элементов из стека (список)	LIST элементы: 7, 6, 5	Элементы удалены, стек пуст
Переключение между стеками	stack1 и stack2	Стек1 и Стек2 поменялись местами
Объединение и сортировка двух стеков	Два стека с элементами	Отсортированный стек с объединенными элементами

Негативные тесты:	Входные данные	Выходные данные
Добавление элементов при переполнении (массив)	ARRAY заполненный стек	Ошибка: Переполнение стека
Удаление из пустого стека (массив)	Пустой стек	Ошибка: Стек пуст, удаление прекращено
Удаление из пустого стека (список)	Пустой стек	Ошибка: Стек пуст, удаление прекращено
Сортировка неинициализированного стека	NULL	Ошибка: Стек пуст

Методология работы

Стек — это структура данных типа LIFO (Last In, First Out), в которой последний добавленный элемент извлекается первым. Основные операции со стеком:

- push – добавление элемента на вершину
- pop – удаление элемента с вершины

Стек хранится в двух возможных вариациях: в виде списка и в виде массива. Но общая логика взаимодействия со стеком сохраняется вне зависимости от реализации, то есть единовременно доступен только верхний элемент.
Если стек реализован в виде статического или динамического массива (вектора), то для его хранения обычно отводится непрерывная область памяти ограниченного размера, имеющая нижнюю и верхнюю границу.
При реализации стека в формате связанного списка каждый элемент стека представляет собой узел, содержащий:

- значение элемента
- указатель на следующий элемент

Стек хранится как односвязный список, где каждый новый элемент добавляется в начало (вершину) списка. Операции добавления и удаления осуществляются на вершине списка:

- при добавлении (push) создается новый узел, который указывает на текущую вершину стека, а затем обновляется указатель вершины.
- при удалении (pop) указатель вершины смещается на следующий элемент.

Аналитическая часть

Все замеры происходили не ноутбуке с подключенным питанием и в режиме высокой производительности.
Характеристики ноутбука:

- ОС: Windows 11
- Оперативная память: 16 Гб DDR4
- Процессор: Intel core i5-12500H

Предположим, что на этапе инициализации стек рассчитан на 1000 элементов. Без учёта хранения указателя на такую структуру:

Метод хранения стека	Формула для расчета занимаемой памяти	Количество занимаемой памяти (байт)
Массив	Sizeof(int) * len(values) + sizeof(int *) * 3	4024
Список	(Sizeof(stack_list *) + sizeof(int)) * val_amount	12

Размер стека, выполненного в виде массива, остается статичным, а вот размер списка динамичный и зависит от кол-ва элементов. Так что при полной заполненности ситуация становится такой:

Метод хранения	Количество занимаемой памяти (байт)
Массив	4024
Список	12000

Список становится невыгоден уже на 336 элементах (33% от максимального размера)

Метод хранения	Количество занимаемой памяти (байт)
Массив	4024
Список	4032

Теперь приведем результаты временных затрат на выполнение операций добавления (push) в стек. Каждое измерение проводилось 100 раз для усреднения результирующего времени.

Метод хранения	Размерность	Время (с)	Отношение массива к списку
Массив	25000	0.000100	13%
Список		0.000730	
Массив	50000	0.000160	11%
Список		0.001450	
Массив	75000	0.000230	10%
Список		0.002230	

Массив	100000	0.000320	11%
Список		0.002850	

Проведем результаты сравнения удаления. Каждое измерение проводилось 100 раз для усреднения результирующего времени.

Метод хранения	Размерность	Время (с)	Отношение массива к списку
Массив	25000	0.000030	33%
Список		0.000090	
Массив	50000	0.000090	36%
Список		0.000250	
Массив	75000	0.000110	33%
Список		0.000330	
Массив	100000	0.000150	37%
Список		0.000400	

Приведем результаты сравнения сортировки 2 стеков с использованием 3 стека. Каждое измерение проводилось 25 раз для усреднения результатов.

Метод хранения	Размерность	Время (с)	Отношение массива к списку
Массив	25000	1.027040	70%
Список		1.461680	
Массив	50000	4.350680	62%
Список		6.962200	
Массив	75000	10.140720	53%
Список		19.157080	
Массив	100000	18.280040	42%
Список		42.696520	

Фрагментация

Фрагмент ниже описывает элементы, которые располагаются в памяти и адреса, по которым описывались элементы ранее, но были очищены.

```
Элемент: 10, адрес: 000002194D623BF0
Элемент: 9, адрес: 000002194D5BCED0
Элемент: 8, адрес: 000002194D5BD150
Элемент: 7, адрес: 000002194D5BD110
Элемент: 6, адрес: 000002194D5BCE90
Элемент: 5, адрес: 000002194D5BCE70
Элемент: 4, адрес: 000002194D5BCFD0
Элемент: 3, адрес: 000002194D5BD030
Элемент: 2, адрес: 000002194D5BD050
Элемент: 1, адрес: 000002194D5BCF90
Удаленный адрес: 000002194D5BCED0
Удаленный адрес: 000002194D5BCEF0
Удаленный адрес: 000002194D5BCE90
```

```
Удаленный адрес: 000002194D5BCE30
Удаленный адрес: 000002194D5BCF90
Удаленный адрес: 000002194D5BD110
Удаленный адрес: 000002194D5BCFB0
Удаленный адрес: 000002194D5BD190
Удаленный адрес: 000002194D5BCE70
Удаленный адрес: 000002194D5BD030
```

Данные указывают на то, что новые элементы частично располагаются по ранее очищенным адресам, но произошло это только в 2 случаях из 10, что может указывать на фрагментацию памяти.

Контрольные вопросы

1. Что такое стек?

Стек – это последовательный список с переменной длиной, в котором включение и исключение элементов происходит только с одной стороны – с его вершины. Стек функционирует по принципу: последним пришел – первым ушел, Last In – First Out (LIFO).

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

В массиве: память выделяется единовременно при инициализации. Размер стека фиксирован, и его емкость определена размером массива.

В связанном списке: память выделяется динамически при добавлении каждого элемента. Размер стека не ограничен и зависит только от доступной памяти в системе.

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

В массиве: указатель на вершину стека просто сдвигается на одну позицию вниз, что позволяет "убрать" элемент из видимости. Память не освобождается.

В связанном списке: удаляемый узел стека освобождается с помощью функции free(), и указатель вершины сдвигается на следующий элемент.

4. Что происходит с элементами стека при его просмотре?

При просмотре элемент выносится из стека функцией pop и считывается его значение. Так продолжатся пока в стеке есть элементы.

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

Эффективность реализации стека по сути зависит от того, ограничены мы в памяти или нет, знаем необходимый размер стека или нет. В зависимости от этих параметров делаем вывод о подходящей реализации.

Заключение

На основе приведённых данных можно сделать следующие выводы. Реализация стека на массиве оказывается более эффективной с точки зрения времени выполнения операций, но более затратным по памяти до 33% заполнения от максимального значения.

Что касается времени выполнения операций, массив демонстрирует лучшие результаты по сравнению со списком. При добавлении элементов (push) разница в производительности очевидна: для размера в 100000 элементов массив выполняет операцию за 0.000320 с, тогда как список требует 0.002850с. В операциях удаления ситуация аналогична — массив работает быстрее. Например, при размере 100000 элементов массив справляется за 0.00015 с, в то время как список — за 0.0004 с.

При сортировке двух стеков с использованием третьего массив также показывает явное преимущество. Для 100000 элементов время выполнения составляет 18.28 с у массива против 42.70 с у списка, что более чем в два раза быстрее. Таким образом, массив оказывается предпочтительным выбором для задач, требующих высокой производительности, особенно при большом объёме данных. Список, благодаря своей гибкости, подходит для хранения неизвестного количества данных.