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 СТРУКТУРЫ ДАННЫХ

<Работа со стеком>

Студент ***<Ермаков И.Г>***

Группа ***<ИУ7-32Б>***

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

Студент _____ **<Ермаков И.Г>**

Преподаватель _____ **<Барышникова М.Ю>**

2024

Содержание

Цель работы

Реализовать операции работы со стеком, который представлен в виде статического массива и в виде односвязного линейного списка; оценить преимущества и недостатки каждой реализации; получить представление о механизмах выделения и освобождения памяти при работе со стеком

Условие задачи

Создать программу работы со стеком, выполняющую операции добавления, удаления элементов и вывод текущего состояния стека. Реализовать стек:
а) статическим массивом (дополнительно можно реализовать динамическим 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.

Проверить правильность расстановки скобок трех типов (круглых, квадратных и фигурных) в выражении.

Описание технического задания

Входные данные:

Изначально, пользователя встречает окно выбора секции

a - использование стека посредством статического массива b - использование стека посредством динамического массива c - использование стека посредством списка t - Просмотр времени выполнения и затраченной памяти

После корректного выбора секции для каждой секции пользователя встречает свое окно выбора режима работы:

При выборе опции “a”:

```
Операции со с стеком в виде статического массива:
1  - создать стек;
2  - добавить элемент в стек;
3  - удалить элемент из стека;
4  - вывести текущее состояние стека;
5  - проверить правильность расстановки скобок в выражении;
6  - очистить стек;
7 - просмотр адресов удаленных переменных
```

При выборе опции “b”:

```
Операции со с стеком в виде динамического массива:
1  - создать стек;
2  - добавить элемент в стек;
3  - удалить элемент из стека;
4  - вывести текущее состояние стека;
5  - проверить правильность расстановки скобок в выражении;
6  - очистить стек;
7 - просмотр адресов удаленных переменных
```

При выборе опции “c”:

```
Операции со с стеком в виде списка:
1  - создать стек;
2  - добавить элемент в стек;
3  - удалить элемент из стека;
4  - вывести текущее состояние стека;
5  - проверить правильность расстановки скобок в выражении;
6  - очистить стек;
7 - просмотр адресов удаленных переменных
```

При выборе опции “t” пользователю дается выбор размера автогенерируемой строки для анализа.

В случае ошибки ввода пользователю выводится сообщение об ошибке и предлагается ко вводу последний список опций, на котором произошла ошибка

Выходные данные

В зависимости от каждой опции выводится соответствующий результат. В случае ошибки на экран выводится что пользователь сделал не так.

Действие программы:
Программа работает до тех пор пока пользователь не введет 0, в случае некорректного ввода программа будет предлагать пользователю ввести еще раз.

Обращение к программе:

Программа собирается с помощью утилиты make
Затем запускается ./app.exe

Аварийные ситуации:

Некорректный ввод секции или опции
Переполнение стека

Описание структур данных:

```
typedef struct
{
    char data[MAX_LEN_STACK];
    int top;
} static_array_stack_t;
```

Данная структура описывает стек, реализованный на статическом массиве. Поле top указывает на последний элемент стека
MAX_LEN_STACK – максимальная длина стека, если кол-во элементов превысит это кол-во, будет переполнение стека.

```
typedef struct
{
    char *data;
    int top;
    int capacity;
} dynamic_array_stack_t;
```

Данная структура описывает стек, реализованный на динамическом массиве, поле top все так же является указателем на последний элемент стека, поле capacity создано для динамического расширение стека.

```
typedef struct
{
    char data;
    struct list_stack_t *next;
} list_stack_t;
```

Данная структура описывает стек, реализованный на односвязном списке. Структура содержит символ data, который хранит в себе символ скобки, а так же указатель next – указатель на следующий элемент стека.

```
typedef struct
{
    char *removed_addresses[MAX_REMOVED];
    int count;
} removed_addresses_tracker_t;
```

Данная структура создана для удобства реализации задачи о сохранении освобожденных адресов на стеке, данная структура содержит поле removed_addresses – массив удаленных адресов. А так же поле count – их кол-во.

MAX_REMOVED – максимальная длина массива с удаленными адресами.

```
typedef enum { STATIC_ARRAY, DYNAMIC_ARRAY, LIST } stack_type_t;
```

Данное перечисление так же является очень важной частью для удобства работы со структурными типами. Это перечисление содержит выбор типов стека с которыми будет происходить работа

STATIC_ARRAY – стек, реализованный через статический массив.
DYNAMIC_ARRAY – стек, реализованный через динамический массив.
LIST – стек, реализованный через односвязный список.

Описание основных сигнатур функций

```
void push_st(static_array_stack_t *stack, char value)

void push_dn(dynamic_array_stack_t *stack, char value)

int push_list(list_stack_t **stack, char value)
```

Функции добавление элемента в стек (на вершину стека), принимает стек и значение для добавления

```
char pop_st(static_array_stack_t *stack, removed_addresses_tracker_t *tracker)

char pop_dn(dynamic_array_stack_t *stack, removed_addresses_tracker_t
*tracker)

char pop_list(list_stack_t **stack, removed_addresses_tracker_t *tracker)
```

Функции для удаления элемента с вершины стека. принимают стек и структуру в которую будут записываться освобожденные адреса.

```
bool check_brackets_st(static_array_stack_t *stack, stack_type_t type, const char
*expr)

bool check_brackets_dn(dynamic_array_stack_t *stack, stack_type_t type, const
char *expr)

bool check_brackets_list(list_stack_t **stack, const char *expr)
```

Функция для проверки правильности расстановки скобок для стека на статике/динамики/списки. Принимают стек, тип стека и строку для обработки

Описание алгоритма

Функция check_brackets_st которая работает со статическим массивом сначала инициализируется проход по строке выражения для каждого символа если это открывающая скобка то она добавляется в стек с помощью функции push_st если это закрывающая скобка то проверяется пуст ли стек с помощью функции is_empty если стек пуст то возвращается false так как закрывающая скобка встретилась без пары если стек не пуст то извлекается верхний элемент стека с помощью pop_st и проверяется соответствует ли он закрывающей скобке если нет то возвращается false после обработки строки проверяется пуст ли стек если в стеке остались элементы значит есть лишние открывающие скобки и возвращается false иначе true

Функция check_brackets_dn которая работает с динамическим массивом алгоритм аналогичен предыдущему сначала добавляются открывающие скобки в стек с помощью push_dn при этом если стек заполнен его емкость увеличивается перед добавлением нового элемента если встречается закрывающая скобка проверяется пуст ли стек если пуст то возвращается false если не пуст то извлекается верхний элемент с помощью pop_dn и проверяется соответствие с закрывающей скобкой если они не совпадают возвращается false после обработки строки проверяется пуст ли стек если остались элементы возвращается false иначе true

Функция check_brackets_list которая работает со стеком на основе списка проходится по каждому символу выражения если символ открывающая скобка она добавляется в стек с помощью push_list если символ закрывающая скобка проверяется пуст ли стек если пуст возвращается false иначе извлекается верхний элемент с помощью pop_list и проверяется соответствует ли он закрывающей скобке если нет возвращается false после обработки строки проверяется пуст ли стек если остались элементы возвращается false если стек пуст возвращается true в конце стек очищается с помощью free_list_stack чтобы освободить память

Сравнение эффективности работы алгоритмов

Методология исследования:
Для исследования времени работы функции на каждый размер проводится по 10000 измерений, затем значение усредняется.

Таблица замеров времени для функций, проверяющих правильность расстановки скобок:

Размер	Стек на статическом массиве (такты)	Стек на динамическом массиве (такты)	Стек на односвязном списке (такты)
100	384	528	2366
250	740	1095	8169
500	1433	1518	9150
1000	3434	3810	33522
5000	7828	8021	97044
10000	13754	15324	191888

Таблица замеров памяти для функций, проверяющих правильность расстановки скобок:
Уточнение : для динамического массива память расширяется следующим образом – изначальное значение длины массива устанавливается в 10, затем при превышении этого лимита величина удваивалась.

Размер	Стек на статическом массиве (байты)	Стек на динамическом массиве (байты)	Стек на односвязном списке (байты)
100	104	164	1600
250	254	324	4000
500	504	644	8000
1000	1004	1284	16000
5000	5004	5124	80000
10000	10004	10244	160000

Тесты:

Терминология:
Секция это первоначальный выбор пользователя:

a - использование стека посредством статического массива b - использование стека посредством динамического массива c - использование стека посредством списка t - Просмотр времени выполнения и затраченной памяти

Набор опций коррелируется в зависимости от опции, пример:

Операции со с стеком в виде статического массива: 0 – возврат к меню секций 1 - создать стек; 2 - добавить элемент в стек; 3 - удалить элемент из стека; 4 - вывести текущее состояние стека; 5 - проверить правильность расстановки скобок в выражении; 6 - очистить стек; 7 - просмотр адресов удаленных переменных

Описание теста	Входные данные	Выходные данные
Некорректный ввод секции	-1	Ошибка ввода секции
Некорректный ввод секции	w	Ошибка ввода секции
Некорректный ввод опции	-1	Ошибка ввода опции
Некорректный ввод опции	8	Ошибка ввода опции
Некорректное использования функции добавления на верхушку стека на статическом массиве	a	Ожидалась скобка!
Некорректное использования функции добавления на верхушку стека на динамическом массиве	a	Ожидалась скобка!

Некорректное использования функции добавления на верхушку стека на списке	a	Ожидалась скобка!
Попытка добавления на стек без инициализации		Стек еще не создан!
Попытка удаления элемента со стека без элементов		Стек пуст!
Вывод элементов неинициализированного стека		Стек еще не создан!
Вывод элементов пустого стека		Стек пуст!
Переполнение стека		Стек переполнен!
Проверка правильности расстановок скобок	{{}}	Скобки расставлены правильно!
Проверка правильности расстановок скобок	{[()]}	Скобки расставлены правильно!
Проверка правильности расстановок скобок	{{}	Ошибка в расстановке скобок!
Проверка правильности расстановок скобок	{{]]	Ошибка в расстановке скобок!

Вывод:

Анализ времени выполнения

Стек на статическом массиве показал лучшие результаты по времени выполнения для всех размеров входных данных. Например, при размере 10000 тиков время выполнения составило 13754 тактов, что значительно быстрее, чем время выполнения для стека на односвязном списке (191888 тактов).

Стек на динамическом массиве занимает промежуточное положение по производительности. Он уступает стеку на статическом массиве (время выполнения **15324 такта** при размере 10000), но значительно опережает реализацию на односвязном списке.

Стек на односвязном списке имеет худшие показатели времени выполнения. Его структура предполагает дополнительные операции выделения и освобождения памяти для каждого узла, что замедляет выполнение. При размере 10000 тактов время выполнения составило **191888 тактов**.

Выводы по времени выполнения

Стек на статическом массиве наиболее эффективен по времени, так как операции добавления и удаления имеют сложность O(1) без накладных расходов на выделение памяти.

Динамический стек требует дополнительного времени для перераспределения памяти при расширении, что увеличивает его общее время выполнения.

Односвязный список из-за операций выделения памяти имеет наибольшие временные затраты.

Анализ затраченной памяти

Стек на статическом массиве потребляет фиксированное количество памяти, независимо от размера входных данных. Например, для размера 10000 потребление составляет 10004 байта.

Динамический стек увеличивает свою память по мере роста данных. Для размера 10000 потребление составляет 10244 байта, что чуть больше, чем у статического стека.

Односвязный список требует значительного объема памяти для хранения указателей на следующий элемент. Потребление памяти значительно возрастает

с увеличением размера входных данных и достигает 160000 байт при размере 10000.

Выводы по затраченной памяти

Стек на статическом массиве имеет преимущество в предсказуемости использования памяти, что делает его эффективным для небольших и средних объемов данных.

Динамический стек потребляет больше памяти из-за динамического увеличения, но может быть полезен при неизвестном заранее размере данных. Односвязный список имеет самое высокое потребление памяти из-за хранения указателей, что делает его менее эффективным.

Теоретические расчеты и выводы

Время выполнения всех операций для всех трех реализаций: O(1) для вставки и удаления, однако в реальных условиях время может увеличиваться за счет особенностей реализации.

Стек на статическом массиве использует фиксированный объем памяти O(n), где n — максимальная длина стека.

Динамический стек может требовать больше памяти в процессе перераспределения, сложность в худшем случае — O(n + k), где k это остаток выделенной памяти.

Односвязный список требует дополнительную память для хранения указателей, что увеличивает использование памяти до O(16 * n + n) (для каждого элемента по два поля: данные и указатель).

Выбор реализации стека зависит от конкретной задачи. Для задач с высокими требованиями к производительности и известным объемом данных статический стек предпочтительнее. Динамические структуры лучше использовать при необходимости гибкой работы с памятью, а односвязные списки — лишь в случае, когда это оправдано логикой приложения.

Ответы на контрольные вопросы:

Что такое стек?
Стек — это структура данных, работающая по принципу "последним пришел — первым вышел" (LIFO, Last In, First Out). Добавление и удаление элементов происходит только с одного конца, называемого вершиной стека.

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

Стек на статическом массиве: выделяется фиксированный объем памяти заранее, равный максимальному размеру стека. Память выделяется один раз и не изменяется.

Стек на динамическом массиве: память выделяется по мере роста данных. При заполнении текущего объема стек увеличивается, удваивая объем памяти.

Стек на односвязном списке: память выделяется под каждый элемент при его добавлении, включая указатель на следующий элемент, что требует дополнительной памяти для указателей.

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

Стек на статическом массиве: память не освобождается, просто сдвигается указатель на вершину.

Динамический стек: память освобождается при удалении элементов, но может быть перераспределена, если стек значительно уменьшается.

Односвязный список: память освобождается для каждого удаляемого, так как каждый элемент хранится в отдельном блоке.

Что происходит с элементами стека при его просмотре?

При просмотре элементов стека содержимое остается неизменным. Обычно просматриваются элементы, начиная с вершины до основания, без их удаления.

Каким образом эффективнее реализовывать стек? От чего это зависит?

Стек на статическом массиве эффективен для задач с известным заранее фиксированным размером данных, так как он обеспечивает быструю работу с памятью и меньшие накладные расходы.

Динамический стек удобен для задач с переменным объемом данных, так как он может автоматически расширяться.

Стек на односвязном списке полезен, если важна гибкость и частые операции добавления/удаления в условиях неизвестного заранее размера, но он требует больше памяти для хранения указателей.