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

Студент **Паламарчук А.Н.**

Группа **ИУ7-33Б**

Предмет **Типы и структуры данных**

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

Студент _____ **Паламарчук А.Н.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

2023 г.

Условие задачи
Создать программу работы для работы с типом данных «стек». Используя созданный тип данных, распечатать убывающие серии последовательности целых чисел в обратном порядке.

Техническое задание
Реализовать стек: двумя способами – с помощью массива и списка. Оформить операции работы со стеком в виде подпрограмм. Ввести арифметическое выражение типа: число|знак| … число|знак| число. Вычислить значение выражения. (Приоритетность операций необязательна). Оценить преимущества и недостатки каждой реализации.

Входные данные

- Команда (число) из меню программы (см. описание алгоритма) – обозначение необходимой операции.

Выходные данные

- Таблица эффективности
- Исходные стеки
- Результат вычисления выражения

Возможные аварийные ситуации

- Некорректный исходный файл
- Ошибка открытия файла
- Отсутствие исходного файла
- Нехватка памяти для работы программы
- Некорректная команда

Описание внутренних структур данных

```
typedef struct node_t node_t;
struct node_t
{
    node_t *next;
    int val;
};

int stack_arr[MAX_LEN_ARR];
int *ps = stack_arr;
node_t *stack_list = NULL;
node_t *pl = stack_list;
```

Константы

```
#define MAX_LEN_ARR 10000
#define COUNT_TESTS 100
```

Используемые функции

```
unsigned long long cur_mks_gettimeofday(void);
```

Возвращает текущее время в мкс

```
int pop_stack_arr(int *stack_arr, int **ps, int *el)
```

Pop из стека (массив)

```
int push_stack_arr(int *stack_arr, int **ps, int el)
```

Push в стек (массив)

```
int pop_stack_list(node_t *list, node_t **pl, int *el)
```

Pop из стека (список)

```
int push_stack_list(node_t **list, node_t **pl, int el)
```

Push в стек (список)

```
int fscanf_exp(char *src, node_t **list, node_t **pl, int *stack_arr, int
**ps, size_t print_flag)
```

Считывает выражение из файла в стеки

```
int input_exp(node_t **list, node_t **pl, int *stack_arr, int **ps)
```

Ручное заполнение стеков

```
int print_stack_arr(int *stack_arr, int *ps, int i)
```

Распечатать стек (массив)

```
int print_stack_list(node_t *stack_list, node_t *pl, int i)
```

Распечатать стек (лист)

```
int calc_exp_arr(int *stack_arr, int *ps, size_t print_flag)
```

Вычисляет значение выражения в стеке (массив)

```
int calc_exp_list(node_t *stack_list, node_t *pl, size_t print_flag)
```

Вычисляет значение выражения в стеке (лист)

```
void efficency_table(void);
```

Вывод таблицы эффективности

Описание алгоритма

Программа обрабатывает нужную команду:
Комманды:
1 - Заполнить стек используя данные из файла

2 - Заполнить стек вручную
3 - Вывести исходный стек
4 - Вычислить выражение (стек - статический массив)
5 - Вычислить выражение (стек - список)
6 - push (стек - статический массив)
7 - pop (стек - статический массив)
8 - push (стек - список)
9 - pop (стек - список)
10 - Получения таблицы производительности
0 - Завершить работу программы

Записываем выражение в стек. Переписываем во временный стек и тем самым переворачиваем исходное выражение. Берём первый элемент записываем в результирующую переменную. Далее циклически идем по стеку и в соответствии с операцией вычисляем значение результирующего выражения.

Производительность
Замеры проводились по 100 раз.

MEMORY

N	Stack array	Stack list
199	40000	1600
399	40000	3200
599	40000	4800
799	40000	6400
999	40000	8000
1199	40000	9600
1399	40000	11200
1599	40000	12800
1799	40000	14400
1999	40000	16000
2199	40000	17600
2399	40000	19200
2599	40000	20800
2799	40000	22400
2999	40000	24000
3199	40000	25600
3399	40000	27200
3599	40000	28800
3799	40000	30400
3999	40000	32000
4199	40000	33600
4399	40000	35200
4599	40000	36800
4799	40000	38400
4999	40000	40000
5199	40000	41600
5399	40000	43200
5599	40000	44800
5799	40000	46400
5999	40000	48000
6199	40000	49600
6399	40000	51200
6599	40000	52800
6799	40000	54400
6999	40000	56000
7199	40000	57600
7399	40000	59200
7599	40000	60800
7799	40000	62400
7999	40000	64000
8199	40000	65600
8399	40000	67200
8599	40000	68800
8799	40000	70400
8999	40000	72000
9199	40000	73600
9399	40000	75200
9599	40000	76800
9799	40000	78400
9999	40000	80000

Время указано в мкс, затраты памяти в байтах
TIME

N	Stack array	Stack list
199	2	59
399	4	207
599	8	469
799	9	770
999	11	1183
1199	13	1691
1399	16	2292
1599	18	2964
1799	21	3741
1999	23	4597
2199	25	5552
2399	28	6587
2599	31	7717
2799	33	8961
2999	36	10249
3199	38	11674
3399	40	13117
3599	43	14693
3799	46	16369
3999	48	18135
4199	50	20862
4399	53	21860
4599	54	23897
4799	57	25983
4999	59	28158
5199	62	30442
5399	65	32813
5599	67	35271
5799	71	42456
5999	75	41199
6199	76	43418
6399	79	49890
6599	82	49415
6799	82	51962
6999	85	55066
7199	87	58129
7399	91	61526
7599	93	64838
7799	94	73540
7999	98	75450
8199	100	92198
8399	103	99325
8599	105	100493
8799	107	86877
8999	109	90718
9199	113	135717
9399	115	98990
9599	117	112082
9799	119	107511
9999	123	119433

Фрагментация
Фрагментация редко наблюдается.

Выводы по проделанной работе
Стек на основе связанного списка, стабильно проигрывает стеку на основе статического массива по скорости работы, однако при объёме элементов, меньшем, чем 50%, стек, выполненный в виде статического массива проигрывает стеку, выполненному в виде списка по памяти. Таким образом, можно сделать вывод, что если нужно реализовать стек, то нужно, ориентируясь на объём доступной памяти и ожидаемое количество элементов выбрать либо стек в виде массива – для прироста в скорости, либо для меньшего объёма занимаемой памяти при объёме элементов, больше чем 50%. Однако если вы рассчитываете обрабатывать меньшее количество элементов и вам критически важен объём занимаемой памяти, то можно выбрать стек на основе списка. Если же нам известно количество элементов в списке, то следует использовать статический массив для реализации стека, иначе следует использовать стек на основе списка.

Контрольные вопросы

1. **Что такое стек?**

   Стек – структура данных, работающая по правилу Last In First Out, в которой можно обрабатывать только последний добавленный элемент.

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

   При хранении стека с помощью списка, мы выделяем блок динамической памяти для каждого узла. При хранении с помощью статического массива, память выделяется на стеке. Для каждого элемента стека, реализованного списком, выделяется значительно большее количество памяти, чем для очередного элемента массива. Эти дополнительные байты занимает указатель на следующий элемент списка. Размер указателя (4 или 8 байт) зависит от архитектуры.
   Для стека на основе статического массива память выделяется до начала работы со стеком и сразу на все элементы стека, а для списка память выделяется по мере необходимости на каждый элемент стека.

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

При хранении стека связанным списком, верхний элемент удаляется путем освобождения памяти для него и смещения указателя. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека, а сами данные на самом деле остаются в памяти, но мы считаем, что их там нет (затираем данные). Для стека на основе статического массива память освобождается после окончания работы со стеком, а для списка память очищается при удалении элемента из стека.

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

   Элементы стека уничтожаются, так как каждый раз достается верхний элемент стека.

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

   Реализовывать стек эффективнее с помощью массива. Он выигрывает как во времени обработки, так и в количестве занимаемой памяти. Если же нам известно количество элементов в списке, то следует использовать статический массив для реализации стека, иначе следует использовать стек на основе списка.