

## 1. Описание условия задачи.

Разработать программу работы со стеком, реализующую операции добавления и удаления элементов из стека и отображения текущего состояния стека. Распечатать убывающие серии последовательности целых чисел в обратном порядке.

## 2. Описание технического задания.

Разработать программу работы со стеком, реализующую операции добавления и удаления элементов из стека и отображения текущего состояния стека.

Распечатать убывающие серии последовательности целых чисел в обратном порядке. Реализовать стек: а) массивом; б) списком.

Все стандартные операции со стеком должны быть оформлены отдельными подпрограммами. В случае реализации стека в виде списка при отображении текущего состояния стека предусмотреть возможность просмотра адресов элементов стека и создания дополнительного собственного списка свободных областей (адресов освобождаемой памяти при удалении элемента, который можно реализовать как списком, так и массивом) с выводом его на экран.

### Входные данные.

*Целое число: пункт меню (от 0 до 12).*

- |   |
|---|
| <ul style="list-style-type: none"><li>1) Добавить элемент в стек на массиве</li><li>2) Добавить элемент в стек на списке</li><li>3) Извлечь элемент из стека на массиве</li><li>4) Извлечь элемент из стека на списке</li><li>5) Вывести текущее состояние стека на массиве</li><li>6) Вывести текущее состояние стека на списке</li><li>7) Вывести список свободных областей</li><li>8) Ввести последовательность целых чисел в стек на массиве</li><li>9) Ввести последовательность целых чисел в стек на списке</li><li>10) Вывести убывающие серии в стеке на массиве.</li><li>11) Вывести убывающие серии в стеке на списке.</li><li>12) Вывести результат замеров</li><li>0) Завершить работу программы</li></ul> |
|---|

Целое число: элемент стека.

### Выходные данные.

Сообщение об успехе или ошибке выполнения пункта, текущее состояние стека, убывающая серия в обратном порядке, список освобождённых областей, таблица замеров по времени и памяти.

### Способ обращения к программе.

Запускается через кнопку «run» в среде разработки «Clion» .

## **Возможные аварийные ситуации и ошибки пользователя.**

Некорректный ввод данных, пустота или переполненность стека. Ошибка выделения памяти.

## **3. Описание внутренних структур данных.**

### **Структура для стека на массиве.**

```
MAX_ARR_SIZE = 500

typedef struct
{
    int data[MAX_ARR_SIZE]; // Элементы стека
    int max_size; // Максимальный размер стека
    int ps; // Указатель на верхушку стека
} array_stack_t;
```

### **Узел списка.**

```
typedef struct node
{
    int data;
    struct node *next;
} node_t;
```

### **Узел списка для освобожденных адресов.**

```
typedef struct free_node
{
    void *address;
    struct free_node *next;
} free_node_t;
```

### **Структура для стека на списке.**

```
typedef struct
{
    node_t *ps; // Указатель на верхушку стека
    int size; // Размер стека
    int max_size; // Максимальный размер стека

    free_node_t *free_list; // Список освобожденных элементов
    int free_elems_size; // Кол-во освобожденных элементов
} list_stack_t;
```

## Интерфейс для работы со структурами.

### Для работы со стеком на массиве

```
// Инициализация стека
void arr_stack_init(array_stack_t *stack, int max_size);

// Проверка стека на пустоту
int is_arr_stack_empty(array_stack_t *stack);

// Проверка стека на заполненность
int is_arr_stack_full(array_stack_t *stack);

// Добавление элемента в стек
void arr_stack_push(array_stack_t *stack, int value);

// Удаление элемента из стека
int arr_stack_pop(array_stack_t *stack);

// Просмотр верхнего значения стека
int arr_stack_peek(array_stack_t *stack);

// Вывод стека
void print_arr_stack(FILE *file, array_stack_t *stack);

// Вывод убывающих серий последовательности
void arr_stack_print_sequence(FILE *file, array_stack_t *stack);
```

### Для работы со стеком на списке

```
// Инициализация стека
void list_stack_init(list_stack_t *stack, int max_size);

// Проверка стека на пустоту
int is_list_stack_empty(list_stack_t *stack);

// Проверка стека на заполненность
int is_list_stack_full(list_stack_t *stack);

// Добавление элемента в стек
int list_stack_push(list_stack_t *stack, int value);

// Удаление элемента из стека
int list_stack_pop(list_stack_t *stack, int *value);
```

```
// Просмотр верхнего значения стека
int list_stack_peek(list_stack_t *stack);

// Вывод стека
void print_list_stack(FILE *file, list_stack_t *stack);

// Вывод освобожденной области
void print_free_list(list_stack_t *stack);

// Очищение стека
void free_list_stack(list_stack_t *stack);

// Вывод убывающих серий последовательности
void list_stack_print_sequence(FILE *file, list_stack_t *stack);
```

## 4. Описание алгоритма.

Берется первый элемент из основного стека, и если стек пуст, записывается значение элемента в файл.

- **Обработка чисел:** Начинается цикл, который работает, пока в основном стеке остаются элементы.
- **Извлечение и сравнение:** Каждый новый элемент извлекается из стека и сравнивается с предыдущим. Если новый элемент меньше предыдущего, это означает, что элементы образуют убывающую последовательность.
- Если текущая серия пустая (в ней пока нет чисел), добавляются оба элемента, и длина серии увеличивается на 2.
- Если серия уже начата, просто добавляется новый элемент, и длина серии увеличивается.
- **Завершение серии:** Если найден элемент, который не меньше предыдущего, текущая убывающая серия завершается:
- Если серия была начата, она заканчивается выводом новой строки, и длина серии обнуляется.

Пример работы:

```
Введите кол-во чисел в последовательность (не больше 20): 7
Введите последовательность:
Введите 1 член последовательности: 4
Введите 2 член последовательности: 5
Введите 3 член последовательности: 3
```

Введите 4 член последовательности: 1  
Введите 5 член последовательности: 6  
Введите 6 член последовательности: 3  
Введите 7 член последовательности: 2

Убывающие серии последовательности в обратном порядке:

2 3 6

1 3 5

## 5. Замеры.

Количество итераций = 50.

### Временные замеры

#### Вывод последовательностей.

Длина последовательности	Время для стека на массиве	Время для стека на списке	Преимущество выполнения стека на массиве к стеку на списке (%)
10	2.54	3.52	27.84
50	6.44	19.36	66.74
100	12.22	29.20	58.16
130	14.60	34.16	57.24
500	54.68	134.20	59.25

#### Добавление элементов

Количество элементов	Время для стека на массиве	Время для стека на списке	Преимущество выполнения стека на массиве к стеку на списке (%)
10	0.12	0.78	84.62
50	0.42	5.10	91.76
100	0.84	6.04	86.09
130	1.00	7.24	86.18
500	3.82	26.26	85.45

### Удаление элементов

Количество элементов	Время для стека на массиве	Время для стека на списке	Преимущество выполнения стека на массиве к стеку на списке (%)
10	0.18	0.38	52.63
50	0.43	2.22	80.18
100	0.58	4.20	86.19
130	0.72	4.98	85.54
500	2.60	18.18	85.69

### Замеры по памяти

Длина последовательности	Память для стека на массиве (байт)	Память для стека на списке (байт)	Преимущество по памяти стека на массиве к стеку на списке (%)
10	2000	160	92.5
50	2000	800	60
100	2000	1600	20
130	2000	2080	3.84
500	2000	8000	75

## 6. Выводы по проделанной работе.

Стек на массиве оказывается более эффективным и по времени, и по памяти, особенно для длинных последовательностей. Его фиксированный размер и доступ по индексу обеспечивают высокую скорость операций, что позволяет обрабатывать данные быстрее, чем в стеке на списке, где работа с указателями создает дополнительную нагрузку. При этом стек на массиве может показаться неэкономичным по памяти при малых объемах данных, но с ростом последовательности его преимущество по сравнению со стеком на списке становится очевидным. Стек на списке, хотя и требует больше памяти из-за хранения указателей, подходит для ситуаций, когда длина последовательности заранее неизвестна и может меняться. Также при работе со стеком на списке при выделении памяти для новых элементов использовалась ранее освобождаемая память, и можно предположить, что фрагментации памяти не наблюдалось.

## 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?

При реализации стека с помощью списка каждый узел требует динамического выделения памяти, что приводит к большому объему занимаемой памяти из-за дополнительных указателей (4 байта на 32-битной и 8 байт на 64-битной архитектуре).

При использовании статического массива память выделяется заранее и занимает фиксированное количество байт в зависимости от типа данных, что обеспечивает меньшие накладные расходы и более эффективное использование памяти.

### 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, указывающего на начало стека. При удалении из стека, реализованного массивом, смещается лишь указатель на вершину стека, а сами данные остаются в памяти, но считаются «стёртыми».

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

При просмотре стека его элементы извлекаются из него, причем классическая реализация стека предполагает, что просмотреть содержимое стека без извлечения (удаления) его элементов невозможно.

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

Эффективность реализации стека массивом или списком зависит от ресурсов памяти и начального размера стека. Если нам важна занимаемая стеком память

и скорость обработки, то эффективнее использовать стек на массиве, а если мы не знаем сколько элементов может храниться в стеке, то лучше использовать стек на списке.