Стек. Способы реализации стека. Основные операции и их вычислительная сложность.

Стек

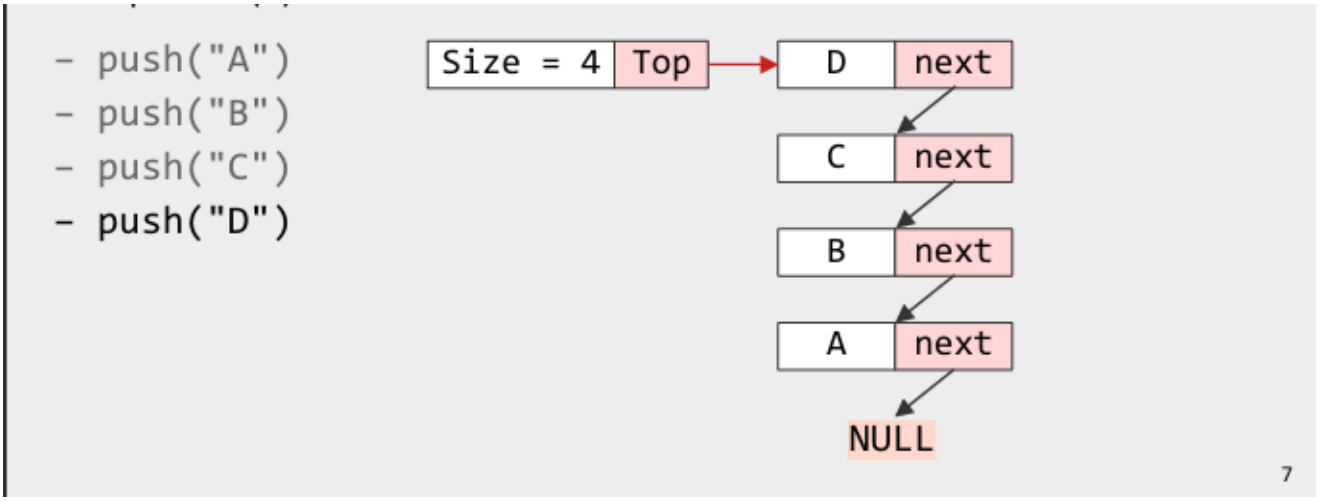
- **Стек** *(stack)* – это структура данных для хранения элементов с дисциплиной доступа «последним пришел – первым вышел» *(Last In – First Out, LIFO).*
- Элементы помещаются и извлекаются из головы стека *(top).*

Способы реализации стека

1. **На основе связных списков** *(длина стека ограничена объемом доступной памяти).*
2. **На основе статических массивов** *(длина стека фиксирована, задана его максимальная длина – количество элементов в массиве).*

Реализация стека на основе связных списков

- Элементы стека хранятся в **односвязном списке** *(singly linked list)*
- Операции добавления (push) и удаления *(pop)* выполняются за время **O(1)**



```c
#include "llist.h" // Реализация связного списка
struct stack {
    struct listnode *top; // Вершина стека
    int size;
};

struct listnode {
    int value; // Значение элемента в стеке
    struct listnode *next;
};
```

Создание пустого стека

```c
struct stack *stack_create()
{
    struct stack *s = malloc(sizeof(*s));
    if (s != NULL) {
        s->size = 0;
        s->top = NULL;
    }
    return s;
}
```

Удаление стека

```c
void stack_free(struct stack *s) {
    while (s->size > 0)
        stack_pop(s); // Удалить все элементы
    free(s);
}
```

Вернуть размер стека

```c
int stack_size(struct stack *s)
{
    return s->size;
}
```

Добавление элемента в стек

```c
int stack_push(struct stack *s, int value)
{
    s->top = list_addfront(s->top, value);
    if (s->top == NULL) {
        fprintf(stderr, "stack: Stack overflow\n");
        return -1;
```

```
    }
    s->size++;
    return 0;
}
```

Извлечение элемента из стека

```
int stack_pop(struct stack *s)
{
    struct listnode *next;
    int value;
    if (s->top == NULL) {
        fprintf(stderr, "stack: Stack underflow\n");
        return -1;
    }
    next = s->top->next;
    value = s->top->value;
    free(s->top);
    s->top = next;
    s->size--;
    return value;
}
```

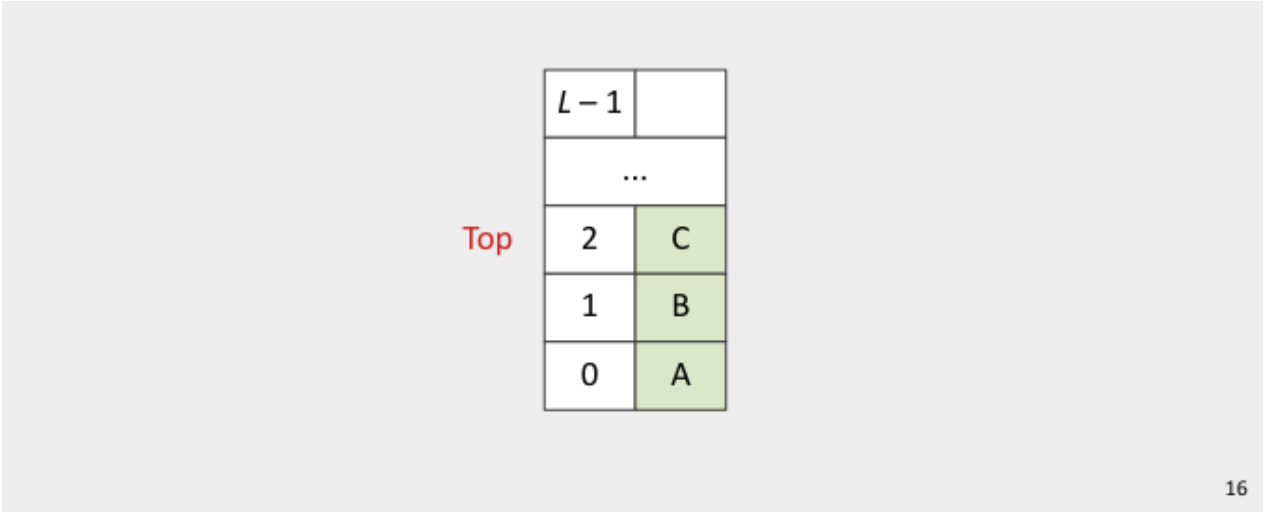
Пример работы со стеком

```
int main()
{
    struct stack *s;
    int i, val;
    s = stack_create();
    for (i = 1; i <= 10; i++)
        stack_push(s, i);
    for (i = 1; i <= 11; i++) {
        val = stack_pop(s);
        printf("pop: %d\n", val);
    }
    stack_free(s);
    return 0;
}
```

```
pop: 10
pop: 9
pop: 8
pop: 7
pop: 6
pop: 5
pop: 4
pop: 3
pop: 2
pop: 1
pop: -1
```

Реализация стека на основе массива

- Элементы стека хранятся в массиве фиксированной длины **L**
- Добавление элемента и удаление выполняется за время **O(1)**




```c
struct stack {
    int *v;
    int top;
    int size;
    int maxsize;
};
```

Создание пустого стека

```c
struct stack *stack_create(int maxsize)
{
    struct stack *s = malloc(sizeof(*s));
```

```
    if (s != NULL) {
        s->v = malloc(sizeof(int) * maxsize);
        if (s->v == NULL) {
            free(s);
            return NULL;
        }
        s->size = 0;
        s->top = 0;
        s->maxsize = maxsize;
    }
    return s;
}
```

Удаление стека

```
void stack_free(struct stack *s)
{
    free(s->v);
    free(s);
}
```

Вернуть размер стека

```
int stack_size(struct stack *s)
{
    return s->size;
}
```

Добавление элемента в стек

```
int stack_push(struct stack *s, int value)
{
    if (s->top < s->maxsize) {
        s->v[s->top++] = value;
        s->size++;
    } else {
        fprintf(stderr, "stack: Stack overflow\n");
        return -1;
    }
    return 0;
}
```

Извлечение элемента из стека

```
int stack_pop(struct stack *s)
{
```

```
    if (s->top == 0) {
        fprintf(stderr, "stack: Stack underflow\n");
        return -1;
    }
    s->size--;
    return s->v[--s->top];
}
```