Стек, очередь и дек

Булгаков Илья, Гусев Илья, Виталий Ерошин

Московский физико-технический институт

Москва, 2023

Содержание

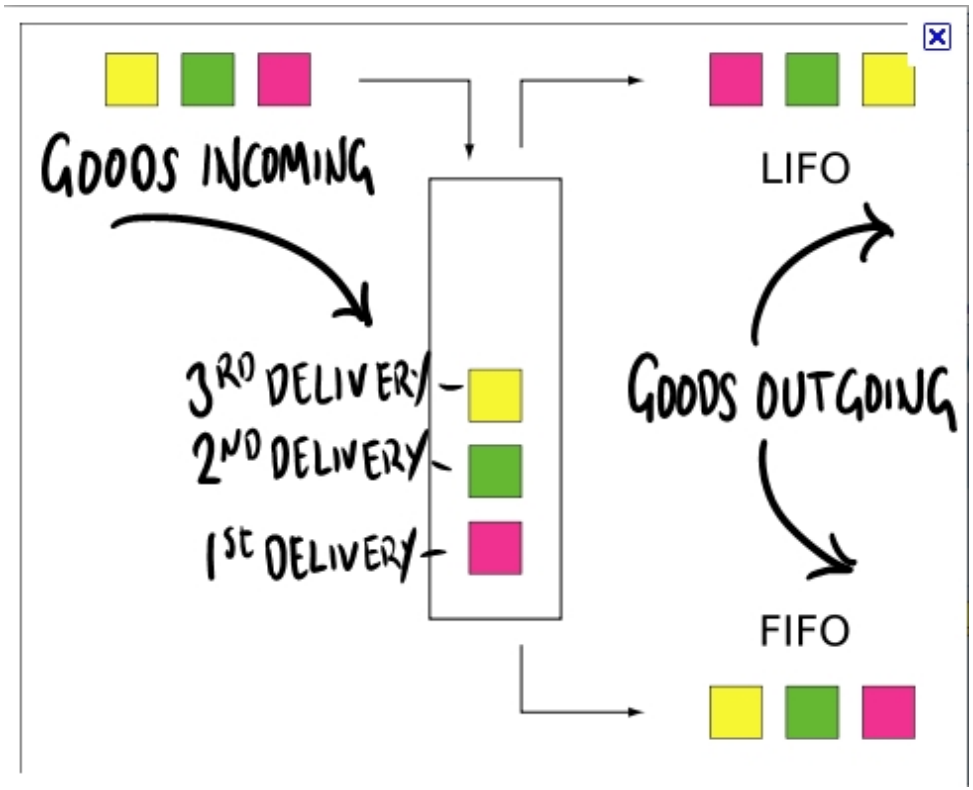
Принципы LIFO vs FIFO

Пусть есть упорядоченные структуры элементов. Рассмотрим несколько абстрактных принципов их обработки.
LIFO и FIFO - подходы к порядку обработки элементов. Применяются в разных сферах (даже бухгалтерском учете!)

- LIFO = last in first out (последний вошёл, первый вышел)
- FIFO = first in first out (первый вошёл, первый вышел)

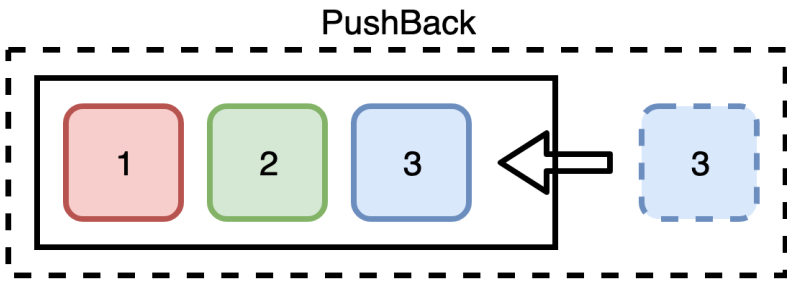
Принципы LIFO vs FIFO

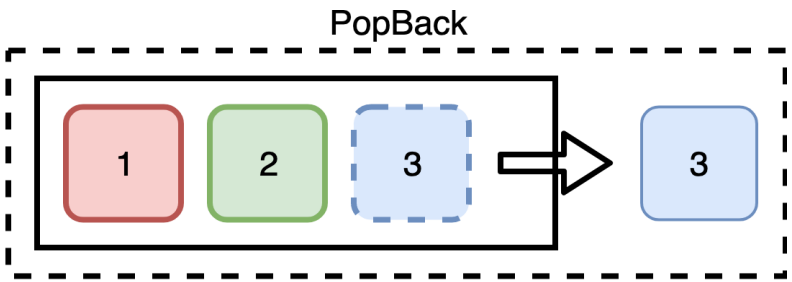
Порядок обработки товаров.
LIFO - как стопка книжек.
FIFO - как на прием к врачу.



Стек

Структура данных, которая поддерживает LIFO, часто называется **Стек.** Как могут выглядеть методы работы с такой структурой, которая поддерживает LIFO?



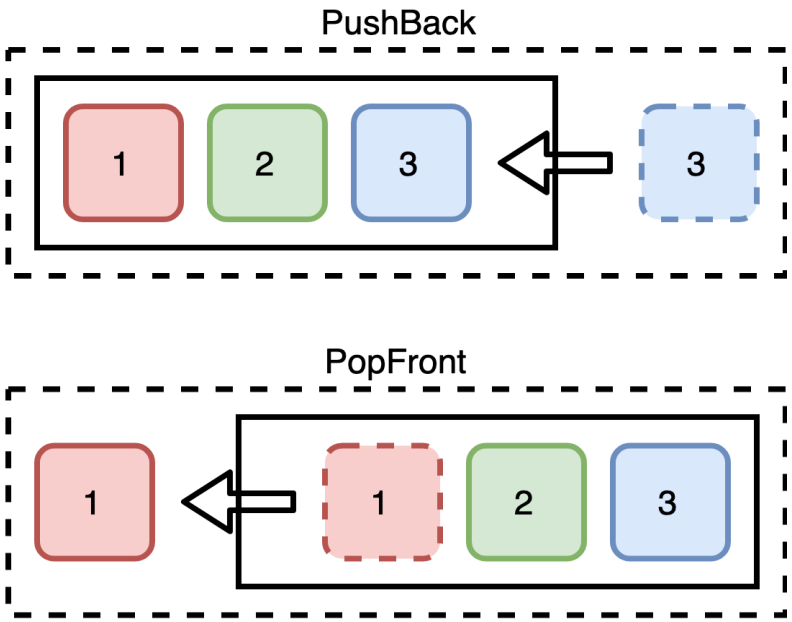


Возможный набор методов (интерфейс) для **стека**:

```
// Вставка элемента в конец
void PushBack(Stack* stack, int
    element);
// Снятие элемента с конца
int PopBack(Stack* stack);
// Проверка на пустоту
bool IsEmpty(Stack* stack);
```

Очередь

Структура данных, которая поддерживает LIFO, часто называется **Очередь**

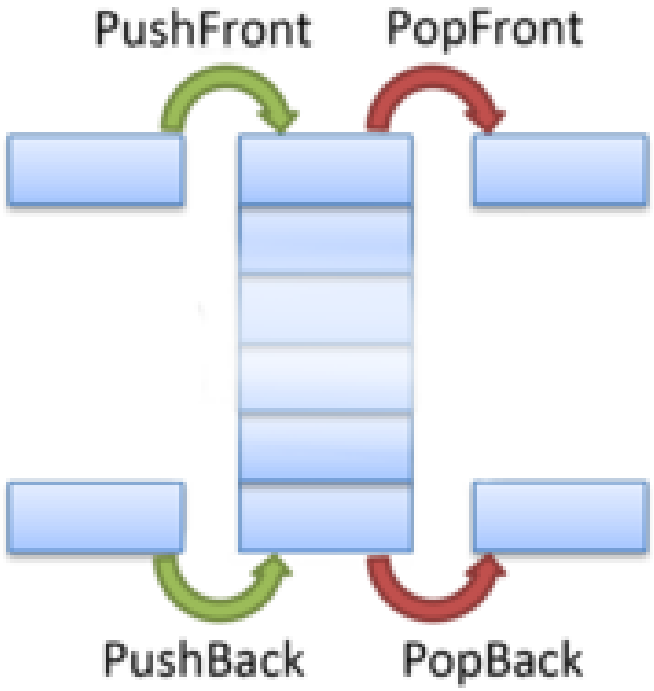


Возможный набор методов (интерфейс) для **очереди**:

```
// Вставка элемента в конец
void PushBack(Queue* queue, int
    element);
// Снятие элемента с начала
int PopFront(Queue* queue);
// Проверка на пустоту
bool IsEmpty(Queue* queue);
```

Дек

Структура данных, которая поддерживает и LIFO, и FIFO называется **Дек**



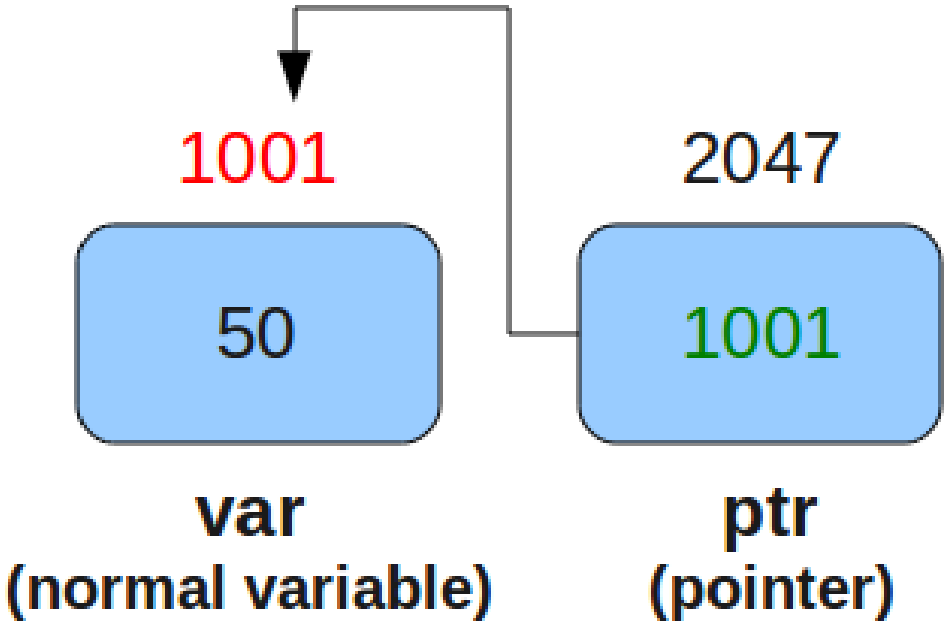
Реализация стека, очереди и дека

Как можно реализовать стек и очередь?

- С помощью однонаправленного связного списка
- С помощью статического массива

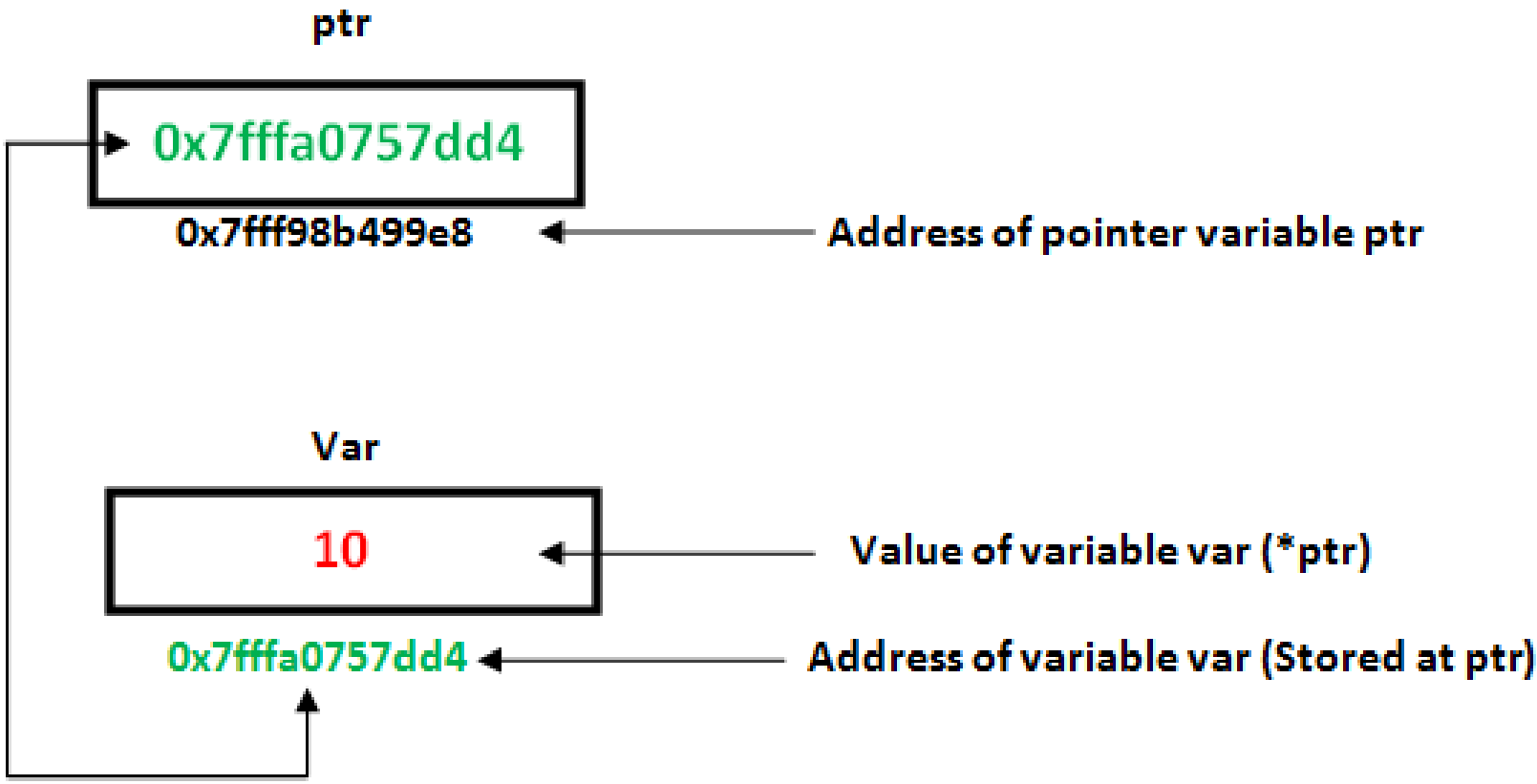
Указатели

```
int  var  =  50;
int* prt  =  &var;
```

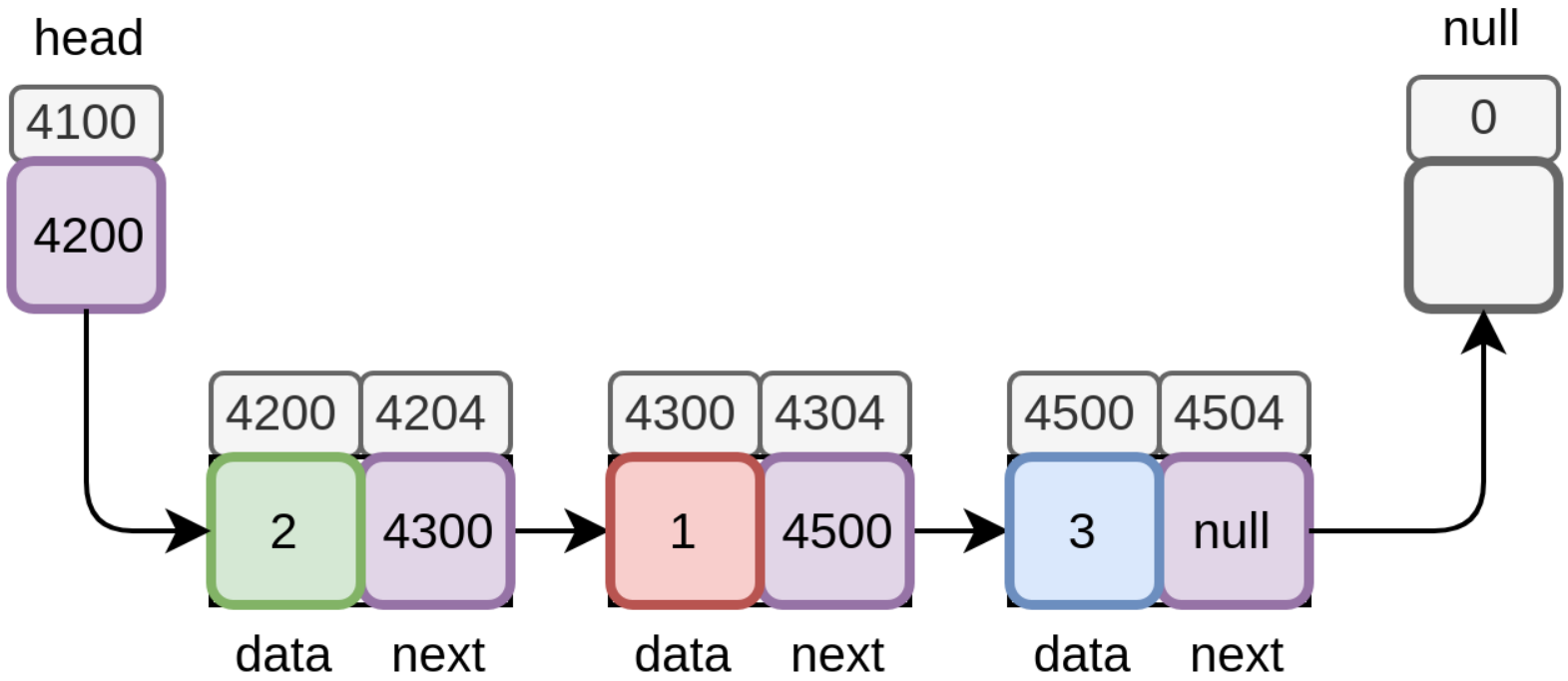


Указатели

На самом деле адреса не такие красивые



Однонаправленный связный список



Однонаправленный связный список

Как используется память при реализации списков?

Адрес	Данные	Комментарий
4100	**4200**	Адрес другой ячейки
4104	<данные по адресу>	
4108	<данные по адресу>	
...	...	
4200	**2**	Данные
4204	**4300**	Адрес другой ячейки
...	...	
4300	**1**	Данные
4304	**4500**	Адрес другой ячейки
...	...	
4500	**3**	Данные
4504	**0**	Нулевой адрес

Обратите внимание: данные лежат не подряд!

Однонаправленный связный список

Как реализовать?

```
struct Node {
    int data; // Данные, которые храним
    Node* next; // Следующий элемент списка
};

struct SingleLinkedList {
    Node* head; // Первый элемент списка
};
```

Однонаправленный связный список

Реализация стека

```
void PushBack(SingleLinkedList* list, int element) {
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = element;
    newNode->next = list->head;
    list->head = newNode;
}

int PopBack(SingleLinkedList* list) {
    Node* backNode = list->head;
    assert(backNode != NULL);
    list->head = backNode->next;
    int data = backNode->data;
    free(backNode);
    return data;
}

bool IsEmpty(SingleLinkedList* list) {
    return (list->head == NULL);
}
```

Однонаправленный связный список

Сложность

Интерфейс стека:

- PushBack: $O(1)$
- PopBack: $O(1)$
- IsEmpty: $O(1)$

Другие операции:

- PushFront: $O(n)$
- PopFront: $O(n)$
- GetByIndex: $O(n)$
- Find: $O(n)$
- ExtractMax: $O(n)$

Реализация на массиве

Если известно число элементов в стеке, либо число элементов небольшое (например $<= 10^5$ элементов), то его можно реализовать на простом массиве.

Реализация на массиве

```
int stack[100];
int n = 100;
int top = -1;
void push(int val) {
    if(top>=n-1)
    cout<<"Stack Overflow"<<endl;
    else {
        top++;
        stack[top]=val;
    }
}
void pop() {
    if(top<=-1)
    cout<<"Stack Underflow"<<endl;
    else {
        cout<<"The popped element is "<< stack[top] <<endl;
        top--;
    }
}
```

Как же все это устроено в стандартной библиотеке C++?

Немного слов про контейнеры

Контейнер управляет памятью, выделенной для хранения элементов, и предоставляет набор методов для доступа к этим элементам напрямую или через итераторы.

Контейнер может хранить элементы одного произвольного типа.

Примеры контейнеров в C++:

- `std::array`
- `std::vector`
- `std::deque`
- `std::list`

std::deque (double-ended queue)

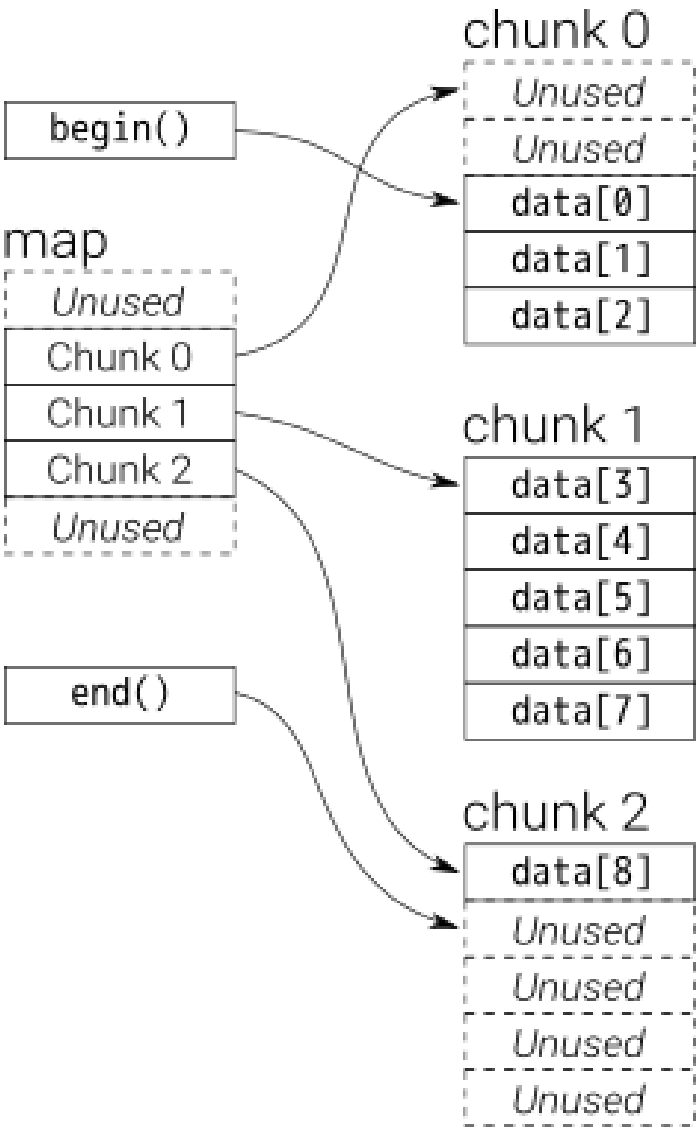
Поддерживает следующие операции:

- Добавление элемента в конец/начало за O(1)
- Удаление элемента с конца/начала за O(1)
- Обращение к произвольному элементу по индексу за O(1)

Пример использования:

```
deque<int> d = {1, 2, 3};
cout << d[1];       // 2
d.push_front(0);
cout << d.size(); // 4
cout << d[1];       // 1
d.pop_front();
d.pop_back();       // d = {1, 2}
```

Внутреннее устройство std::deque



- Элементы располагаются по несколько штук в так называемых "чанках"
- Адреса чанков хранятся в отдельном динамическом массиве
- Чтобы вычислить адрес элемента по индексу, достаточно хранить номер первого чанка и номер первого непустого элемента в чанке
- При добавлении новых элементов увеличиваем количество чанков

Реализация стека и очереди в библиотеке C++

В языке C++ стек и очередь - это лишь обертки над контейнерами (*container adaptors*), поддерживающими нужные операции. По умолчанию в качестве такого контейнера берется `std::deque`.

При желании, можно указать, какой контейнер мы хотим использовать вместо `std::deque`. Например, для стека подойдет `std::vector` и `std::list`. Для очереди `std::list`.

```
// Стек строк, как обертка над std::deque
stack<string> stack_over_deque;

// В качестве контейнера можно указать std::list
stack<string, list<string>> stack_over_list;
```

Стек и очередь в библиотеке C++

std::stack

`std::stack` поддерживает такие операции:

- `top()`
- `empty()`
- `size()`
- `push()` - добавляет элемент в стек
- `pop()` - удаляет элемент с вершины стека

Пример использования:

```
stack <int> s;
s.push(10);
s.push(30);
cout << s.size();
cout << s.top();
s.pop();
```

Стек и очередь в библиотеке C++

std::queue

`std::queue` поддерживает такие операции:

- `front()`
- `empty()`
- `size()`
- `push()` - добавляет элемент в конец очереди
- `pop()` - удаляет элемент из начала очереди

Полезные ссылки I

Nearly All Binary Searches and Mergesorts are Broken
https://bit.ly/2MdGqfU

Т.Кормен, Ч.Лейзерсон, Р.Ривест, К.Штайн - Алгоритмы. Построение и анализ. Глава 3
https://bit.ly/2wFzphU