5. Стандартная библиотека, динамический массив, std::string

Программирование и алгоритмизация

Практические занятия

БИВТ-24-17

Надежда Анисимова

ms teams m2102039@edu.misis.ru

Проверка себя

1. Чем отличаются git merge от git rebase?
2. Чем стек отличается от кучи?
3. Принцип работы стека?

Что делают команды?

- git branch
- git cheackout
- git checkout -b
- git fetch
- git pull

Массивы в c++

01

Массивы стандартной библиотеки

Массив в стиле С

std::array

std::vector

C-style массив

std::array<int, N> a;

std::vector<int> v;

T a[N];

тип

кол-во элементов

тип

кол-во элементов

тип

Фиксированный размер

Массив в стиле С

- Одномерный массив

```
int array[5];
int array[5]{1, 2, 3, 4, 5};
```

- N-мерный массив

```
int array[3][4];
```

0	1	2	3
1			
2			

- operator[] - произвольный доступ к элементам
- N - длину массива лучше выделять в отдельную переменную, если позже надо использовать
- Выделяется **на стеке** с фиксированным размером, известным на этапе компиляции

Массив в стиле С

- Объект массива “забывает” свой размер

```
void SomeFunc(int Array[]){
    sizeof(Array); - вернет размер указателя в байтах
}
```

- Чтобы помнить длину - нужно её везде с собой передавать

```
void SomeFunc(int Array[], int arraySize);
```

!!! Избегать использования !!!

std::array

Обертка над C-style массивом, позволяющая пользоваться доп возможностями стандартной библиотеки

- Помнит свой размер
- Оператор at() с проверкой выхода из границ
- Поддержка итераторов
- и тд

(public member function)

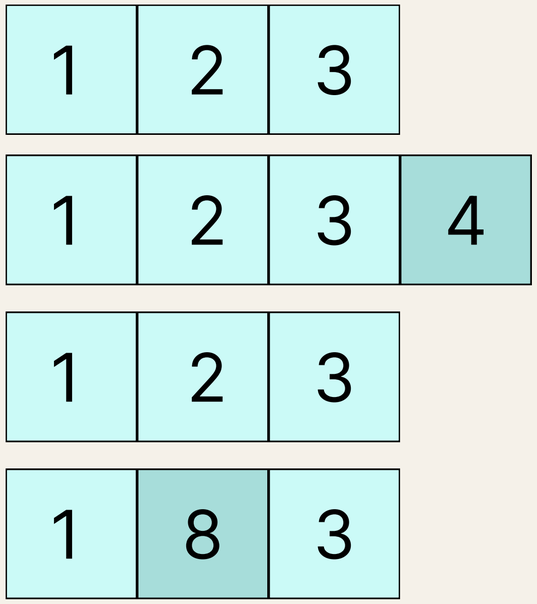
Element access	
`at`	access specified element with bounds checking (public member function)
`operator[]`	access specified element (public member function)
`front`	access the first element (public member function)
`back`	access the last element (public member function)
`data`	direct access to the underlying contiguous storage (public member function)

std::vector **Динамический массив**

Динамический массив - массив, размер которого может изменятся в run-time, в процессе исполнения программы

Память выделяется - **в куче**

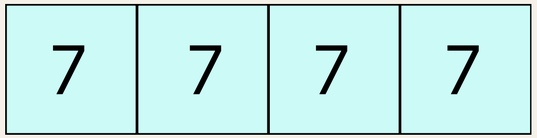
```
std::vector<int> v {1, 2, 3};
v.push_back(4);
v.pop_back();
v[1] = 8;
```



Некоторые операции

- **инициализация через N одинаковых элементов**

```
std::vector<int> v (4, 7);
```



- **методы вектора**

```
vector.erase(vector.begin(), vector.end());
```

итераторы задают интервал

- **алгоритмы из стандартной библиотеки <algorithm>**

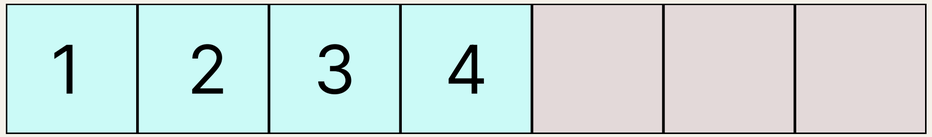
```
std::reverse(vector.begin(), vector.end());
```

применяются к вектору - не нужно заново присваивать

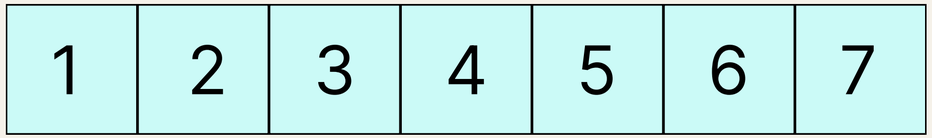
Size vs Capacity

Size - количество элементов в массиве

Capacity - сколько элементов выделено по факту под капотом

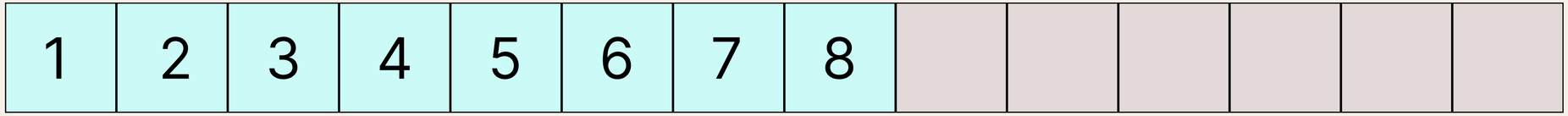


size = 4; capacity = 7;



size = 7; capacity = 7;

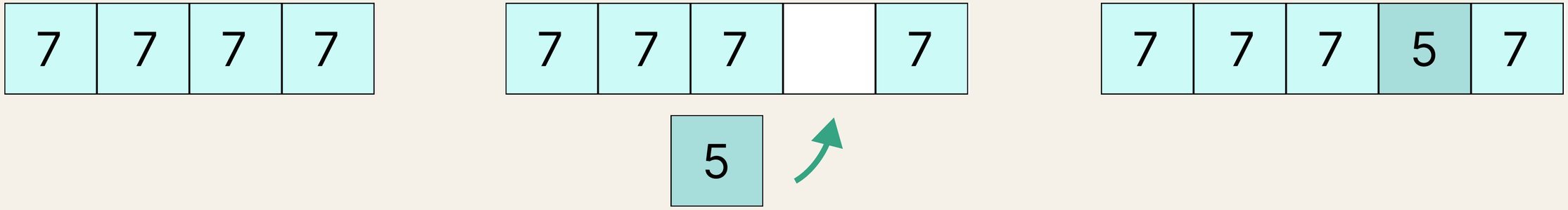
Элементы копируются в новую непрерывную область памяти



size = 8; capacity = 14; - увеличилось в 2 раза

Сложность операций

- Свободный доступ к элементам – O(1)
- Добавление/удаление в конце (push_back/pop_back) – в большинстве O(1), если превысилось capacity – O(n)
- Добавление/удаление в другое место – линейно в количестве элементов от нужного до конца вектора O(n)



Строки в c++

02

C-style строки

- статический массив char, который хранит на один больше символ

char []

- \0 - последний доп символ, указывает конец строки

std::string

- динамический массив символов

std::vector<char>

```
std::string hw = "Hello";
hw += " World!";
hw[4] = “A”
```

std::string

Класс с удобными методами для работы со строками

- size(), length() — возвращают длину строки.
- substr() — извлекает подстроку.
- find() — ищет подстроку.
- append(), push_back() — добавляют символы или строки

Defined in header <string>

Type	Definition
std::string	std::basic_string<char>
std::wstring	std::basic_string<wchar_t>
std::u8string (since C++20)	std::basic_string<char8_t>
std::u16string (since C++11)	std::basic_string<char16_t>
std::u32string (since C++11)	std::basic_string<char32_t>

Контейнеры стандартной библиотеки

03

Контейнеры

- Последовательные контейнеры
- Ассоциативные контейнеры
- Неупорядоченные ассоциативные контейнеры

+ есть адапторы на контейнеры – предоставляют другой интерфейс

Последовательные контейнеры

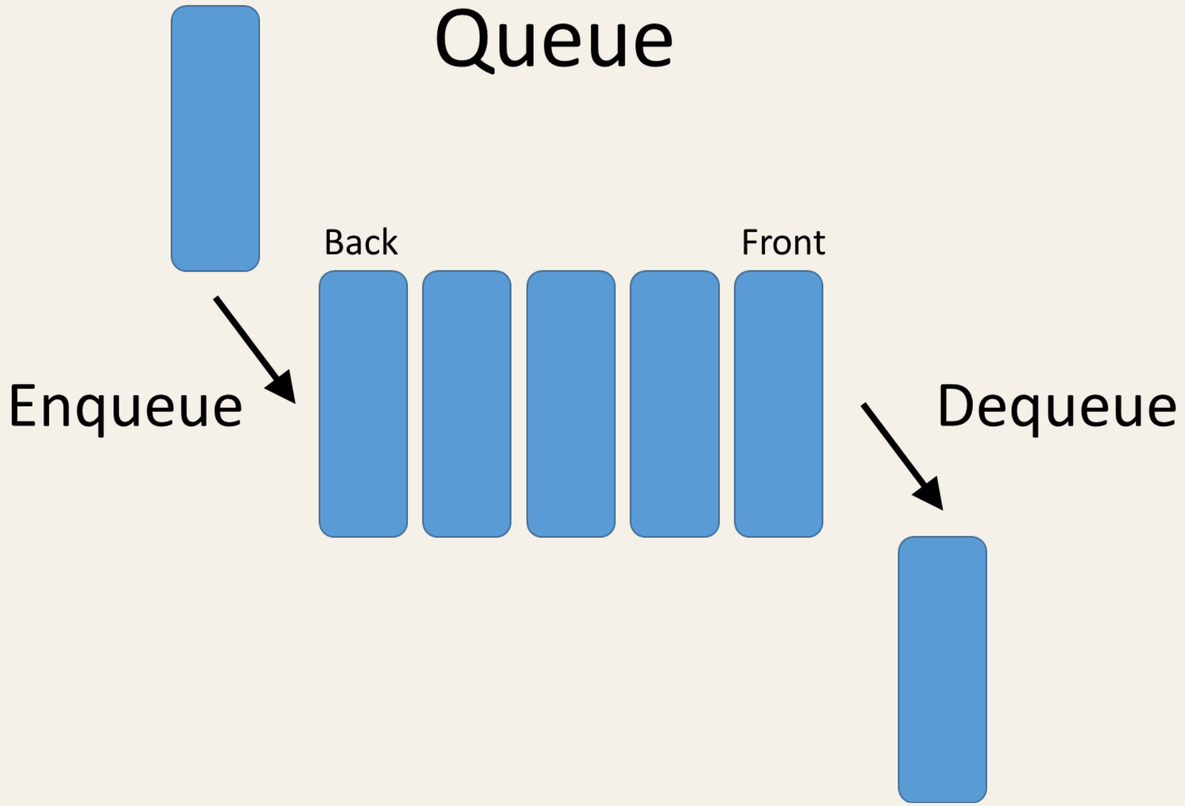
array (C++11)	fixed-sized inplace contiguous array (class template)
vector	dynamic contiguous array (class template)
inplace_vector (C++26)	dynamically-resizable, fixed capacity, inplace contiguous array (class template)
deque	double-ended queue (class template)
forward_list (C++11)	singly-linked list (class template)
list	doubly-linked list (class template)

Очередь

FIFO

First in – first out

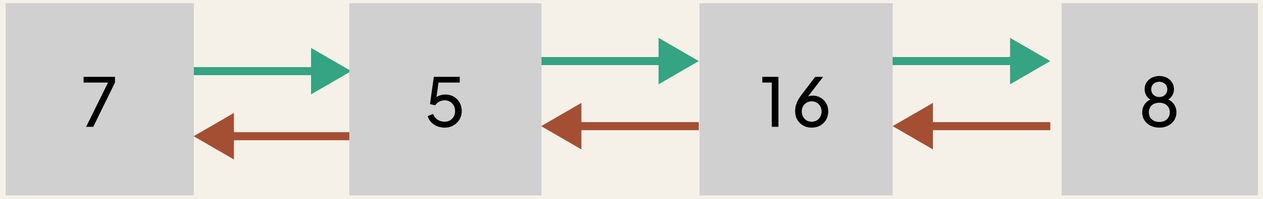
Новый элемент – первым зашел и первым выходит



std::list<int> l = {7, 5, 16, 8}; – Двусвязный список

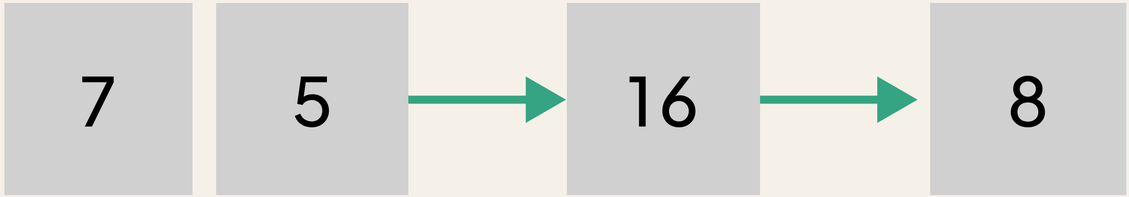
std::forward_list fList { 1, 2, 3, 4, 5 }; – Односвязный список

- Нет прямого доступа по индексу, так как элементы не хранятся в смежной памяти. Для доступа к элементу нужно итерировать по списку, поэтому доступ к элементу по индексу — O(n)
- Добавление/удаление – O(n)
- Перебор по элементам O(n)



```
std::deque<int> d = {7, 5, 16, 8};
```

- В блоках доступ к элементам осуществляется за O(1), но требуется доступ к нужному блоку, что может потребовать O(log n) в худшем случае.
- Добавление/удаление элементов в начале или конце дека — O(1), а в середине — O(n) из-за необходимости поиска позиции для вставки или удаления.
- Перебор по элементам выполняется за O(n)



Домашнее задание

Контест ДЗ3

Три задачи, каждую нужно решить двумя способами:

1. **В ручную, циклами и тд**
2. **С помощью методов/алгоритмов** std