Знакомство с C++

Механизмы абстракции. Стандартная библиотека.

Еще раз о задании

Реализовать систему полнотекстового поиска.

Индексация

- Парсинг документов
- Построение обратного индекса

Поиск

- Парсинг поискового запроса
- Извлечение документов из индекса
- Оценка и ранжирование документов

Принцип работы индексатора

Вход: [doc_id => text]

```
1 => The Matrix
2 => The Matrix Reloaded
3 => The Matrix Revolutions
```

Выход: [word => [doc_id]]

```
matrix      => [1, 2, 3]
reloaded    => [2]
revolutions => [3]
```

Поддержка нечеткого поиска

```
mat          => [1, 2, 3]
matr         => [1, 2, 3]
matri        => [1, 2, 3]
rel          => [2]
relo         => [2]
reloa        => [2]
rev          => [3]
revo         => [3]
revol        => [3]
```

Генерируем n-граммы для каждого слова

Поиск

```
query> MatriX: revl
>> matrix rev
>> mat matr matri rev

mat          => [1, 2, 3]
matr         => [1, 2, 3]
matri        => [1, 2, 3]
rev          => [3]
revl         => []
```

Поиск

```
query> MatriX: revl
>> matrix rev
>> mat matr matri rev

3: The Matrix Revolutions
1: The Matrix
2: The Matrix Reloaded
```

Обзор всей системы

См. диаграмму https://csc-cpp.readthedocs.io/ru/2023/s1/3-indexer.html

Выводы

Нужны:

1. Средства для моделирования предметной области
2. Структуры данных
3. Алгоритмы

Цель:

1. Знать, что есть в языке и стандартной библиотеке
2. Делать аргументированный выбор
3. Уметь пользоваться

Краткая хронология

1980: C With Classes

1983: C++

1998: Международный стандарт языка

2003: C++03

C++11, C++14, C++17, C++20, C++23

Bjarne Stroustrup

Как написать программу?

1. Изучить предметную область.
2. Описать ее в системе типов языка:
    a. Определить типы
    b. Реализовать операции над типами
3. ???
4. PROFIT!

Типы и операции

```
struct Point { double x_, y_; };

void point_move(struct Point* point,
                const struct Vec2d* vec) {
  point->x_ += vec->x_;
  point->y_ += vec->y_;
}

void point_println(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)\n", point->x_, point->y_);
}
```

Объясните расстановку const

```
struct Point { double x_, y_; };

void point_move(struct Point* point,
                const struct Vec2d* vec) {
  point->x_ += vec->x_;
  point->y_ += vec->y_;
}

void point_println(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)\n", point->x_, point->y_);
}
```

Покритикуйте **эти const**

```
struct Point { /* const */ double x_, y_; };

void point_move(/* const */ struct Point* /* const */ point,
                const struct Vec2d* /* const */ vec) {
  point->x_ += vec->x_;
  point->y_ += vec->y_;
}

void point_println(const struct Point* /* const */ point) {
  printf("POINT(%.6lf %.6lf)\n", point->x_, point->y_);
}
```

Покритикуйте этот код

```
struct Point { double x_, y_; };

void point_move(struct Point* point,
                const struct Vec2d* vec) {
  point->x_ += vec->x_;
  point->y_ += vec->y_;
}

void point_println(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)\n", point->x_, point->y_);
}
```

Покритикуйте этот код

```
struct Point { double x_, y_; };

void point_move(struct Point* point,
                const struct Vec2d* vec) {
  point->x_ += vec->x_;
  point->y_ += vec->y_;
}

void point_println(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)\n", point->x_, point->y_);
}
```

Субъективные проблемы

1. Ручное аннотирование функции типом данных: `point_*`.
2. Все указатели могут быть `NULL`.
3. Явное указание `struct`.
   Ранее мы уходили от него typedef-идиомой:

   ```
   typedef struct { ... } Point;
   ```

Объединение данных и методов

```
struct Point {
  double x_, y_;

  void move(/*Point* this, */ const Vec2d* vec) {
    /*this->*/ x_ += vec->x_;
    /*this->*/ y_ += vec->y_;
  }
};
```

Объект, для которого вызывается метод

Явное указание this считается дурным тоном

Объединение данных и методов

```
struct Point {
  double x_, y_;

  void move(const Vec2d* vec) {
    x_ += vec->x_;
    y_ += vec->y_;
  }
};
```

Нет необходимости в указателе:

- Нет адресной арифметики
- Нет optional-семантики

Объединение данных и методов

```
struct Point {
  double x_, y_;

  void move(const Vec2d& vec) {
    x_ += vec.x_;
    y_ += vec.y_;
  }
};
```

Есть ли смысл экономить на копировании?

Объединение данных и методов

```
struct Point {
  double x_, y_;

  void move(Vec2d vec) {
    x_ += vec.x_;
    y_ += vec.y_;
  }
};
```

Можно обойтись

передачей по значению

Константность this

```
void point_move(          struct Point* point, ...);
void point_println(const struct Point* point);

struct Point {
  ...

  void move(Vec2d vec);
  void println();
};
```

Если this не указывается в сигнатуре, как управлять его константностью?

Constants and immutability

Константность this

```
void point_move(          struct Point* point, ...);
void point_println(const struct Point* point);

struct Point {
  ...

  void move(Vec2d vec);
  void println() const;
};
```


Constants and immutability

Обобщение классов

```
struct Point {
  double x_, y_;

  void move(Vec2d vec) {
    ...
  }
};
```

x, y: double? float? int? BigInteger?

Обобщение классов

```
template <typename T>
struct Point {
  T x_, y_;

  void move(const Vec2d<T>& vec) {
    ...
  }
};
```

Тип T и, следовательно, его размер неизвестны

C vs C++

C

```
struct Point pt = {};
point_println(&pt);

struct Vec2d v = {1, 2};
point_move(&pt, &v);
point_println(&pt);
```

C++

```
Point<double> pt;
pt.println();

Vec2d<double> v{1, 2};
pt.move(v);
pt.println();
```


https://godbolt.org/z/7ronvxWbq

C vs C++: предварительные выводы

✓ C++ позволяет создавать более универсальные типы
- `struct Point` — только для double
- `Point<T>` — для любого подходящего T

? Пока неясна мотивация перемещения функций внутрь класса.

! `Point::println()` — только пример, не делайте так в своих классах.

Обобщение функций

C-style: макросы

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))
```

Покритикуйте этот подход

Обобщение функций

C-style: макросы

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))
```

Покритикуйте этот подход

```
int a = 1, b = 2;
int m = MAX(a++, b++); // m = ? b = ?
```

Обобщение функций

C-style: макросы

```
#define MAX(a, b) (((b) < (a)) ? (a) : (b))
```

Покритикуйте этот подход

```
int a = 1, b = 2;
int m = MAX(a++, b++); // m = 3 b = 4
```

Обобщение функций в C++

```
template <typename T>
T max(T a, T b) { return b < a ? a : b; }


int a = 1, b = 2;
int m = max(a++, b++); // m = ? b = ?
```

Обобщение функций в C++

```
template <typename T>
T max(T a, T b) { return b < a ? a : b; }


int a = 1, b = 2;
int m = max(a++, b++); // m = 2 b = 3
```


https://godbolt.org/z/as5zMv7od

Обобщение функций

C:

```
void qsort(void *base, size_t nmemb, size_t size,
           int (*compar)(const void *, const void *))
```

C++:

```
template <typename RandomIt, typename Compare>
void sort(RandomIt first, RandomIt last,
          Compare comp);
```


Benchmark: qsort vs std::sort

STL

1993–1994 Создание STL

Standard Template Library

(STepanov Library :))

Alexander Stepanov

Минимальный набор

1. Последовательные контейнеры
   a. Динамический массив: std::vector
   b. Строка: std::string
2. Ассоциативные контейнеры
   a. std::unordered_map, std::unordered_set
   b. std::map, std::set
3. Алгоритмы: поиск, сортировка, …
4. Ввод-вывод и ФС

Операции стандартных контейнеров

CRUD

- **Create**
- Read
- Update
- Delete

Динамический массив: std::vector<T>

```
std::vector<int> v{3, 1, 4, 1, 5};
```

1. Обращение к элементу по индексу за O(1)
2. Вставка в конец за амортизированное O(1)

“When choosing a container, remember vector is best; leave a comment to explain if you choose from the rest!”

[sequence.reqmts]

Строка: std::string

```
std::string s{“Hello!”};
```

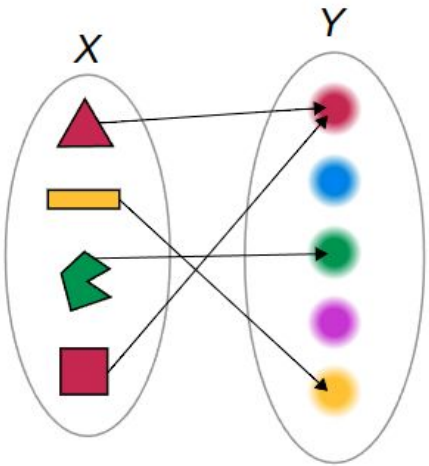
1. Обращение к элементу по индексу за O(1)
2. Вставка в конец за амортизированное O(1)
3. Не определяет кодировку, хранит байты
4. Последний символ — ‘\0’
5. Есть std::char_traits[1]
6. Допускает Small String Optimization[1]

[1] об этом в другой раз

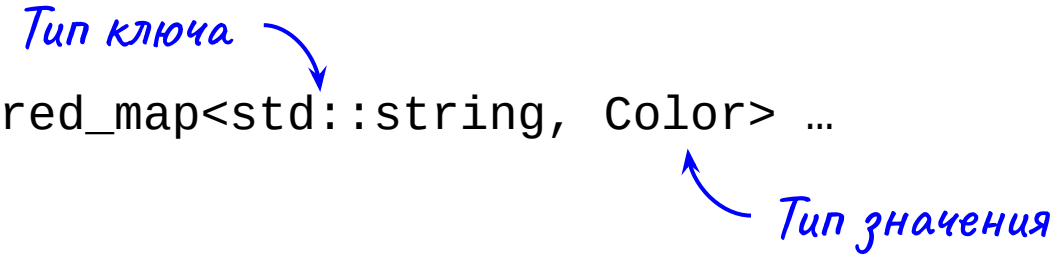
[sequence.reqmts]

Ассоциативный контейнер: std::unordered_map

```
std::unordered_map<std::string, Color> shape_to_color{
    {"triangle",  Color::Red},
    {"rectangle", Color::Yellow},
    {"polygon",   Color::Green},
    {"square",    Color::Red},
};
```



Ассоциативный контейнер: std::unordered_map

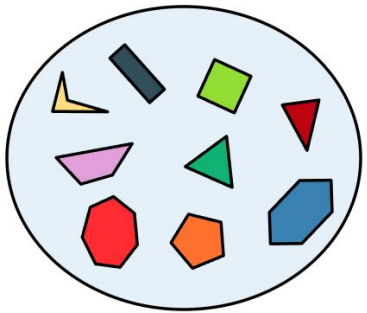


- Хранит пары {ключ: значение}
- Реализация — хеш-таблица
- Поиск, вставка и удаление элемента в среднем за O(1)
- Ключи уникальны

Множество: std::unordered_set

```
std::unordered_set<std::string> s{“one”, “two”, ...};
```

- Хранит уникальные элементы
- Реализация — хеш-таблица
- Поиск, вставка и удаление элемента в среднем за O(1)



Создание экземпляра std::vector

```
std::vector<int> v0;           // []

std::vector<int> v1{3, 1, 4};  // [3, 1, 4]

std::vector<int> v2(3);        // [0, 0, 0]
std::vector<int> v3(3, 42);    // [42, 42, 42]
std::vector<int> v4{3, 42};    // [3, 42]

std::vector<int> v5(v3);       // [42, 42, 42]
```

vector (constructor)

Создание экземпляра std::string

```
std::string s0;                 // “”

std::string s1{"Hello!"};       // “Hello!”
std::string s2("Hello!");       // -/-
std::string s3 = "Hello!";      // -/-

std::string s4(3, 'a');         // “aaa”
std::string s5("Hello!", 5);    // “Hello”
std::string s6(s5);             // “Hello”
```

basic_string (constructor)

Создание экземпляра std::unordered_map

```
std::unordered_map<std::string, int> empty;

std::unordered_map<std::string, Color> shape_to_color{
    {"triangle",  Color::Red},
    {"rectangle", Color::Yellow},
    ...
};

std::unordered_map<std::string, Color> shape_to_color_copy(sh
```

Oops...

unordered_map (constructor)

Используйте синонимы для читаемости

```
using ShapeToColor
    = std::unordered_map<std::string, Color>;

ShapeToColor shape_to_color{
    {"triangle",  Color::Red},
    {"rectangle", Color::Yellow},
    ...
};

ShapeToColor shape_to_color_copy(shape_to_color);
```

Создание экземпляра std::unordered_set

```
std::unordered_set<std::string> empty;

std::unordered_set<std::string> s1{“one”, “two”};

std::unordered_set<std::string> s2(s1);
```

Здесь ничего принципиально нового

unordered_set (cosntructor)

Sleeping test

```
std::unordered_set<int> s{3, 1, 4, 1, 5};

std::cout << s.size();  // Что будет выведено?
```

Sleeping test

```
std::unordered_set<int> s{3, 1, 4, 1, 5};

std::cout << s.size();  // 4
```

Операции стандартных контейнеров

CRUD

- ✓ Create
- Read
- Update
- Delete

Обход std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (std::size_t i = 0; i < v.size(); ++i) {
  std::cout << v[i] << ' ';
}
```

– *Индекс используется только для обращения к элементу*

– *v[i] доступен для записи*

Обход std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (const int item : v) {
  std::cout << item << ' ';
}
```


Range-based for loop

Обход std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (const auto item : v) {
  std::cout << item << ' ';
}
```

Компилятор может вывести тип элемента

Обход std::vector

```
std::vector v{3, 1, 4, 1, 5};

for (const auto item : v) {
  std::cout << item << ' ';
}
```

И здесь тоже

Компилятор может вывести тип элемента

Изменение элементов std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (std::size_t i = 0; i < v.size(); ++i) {
  v[i] *= 2;
}
```

– *Индекс используется только для обращения к элементу*

– *v[i] доступен для записи (но нам это и нужно?)*

Обход std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (int& item : v) {
  item *= 2;
}

// v == [6, 2, 8, 2, 10]
```

Обход std::vector

```
std::vector<int> v{3, 1, 4, 1, 5};

for (auto& item : v) {
  item *= 2;
}

// v == [6, 2, 8, 2, 10]
```

Обход std::string

```
std::string s = “Hello!”;
for (std::size_t i = 0; i < s.size(); ++i) {
  std::cout << s[i];
}
```

+/– как у std::vector

```
for (const char c : s) {
  std::cout << c;
}

std::cout << s;  // Вывести можно и так
```

Обход std::unordered_map

```
std::unordered_map<std::string, std::string> en_to_ru{
    {"one",   "один"},
    {"two",   "два"},
    {"three", "три"},
};
```

Обход std::unordered_map

```
for (std::unordered_map<
        std::string, std::string>::const_iterator
     it = en_to_ru.begin();
     it != en_to_ru.end();
     ++it) {

  const auto& en = it->first;
  const auto& ru = it->second;
  std::cout << en << " -> " << ru << '\n';
}
```

Обход std::unordered_map

```
for (std::unordered_map<
        std::string, std::string>::const_iterator
     it = en_to_ru.begin();
     it != en_to_ru.end();
     ++it) {

  const auto& en = it->first;
  const auto& ru = it->second;
  std::cout << en << " -> " << ru << '\n';
}
```

Обход std::unordered_map

```
for (auto it = en_to_ru.cbegin();
     it != en_to_ru.cend();
     ++it) {

  const auto& en = it->first;
  const auto& ru = it->second;
  std::cout << en << " -> " << ru << '\n';
}
```

Обход std::unordered_map

```
std::unordered_map<std::string, std::string> ...;

for (const std::pair<std::string, std::string>&
     item : en_to_ru) {
  const auto& en = item.first;
  const auto& ru = item.second;
  std::cout << en << " -> " << ru << '\n';
}
```

Кто видит ошибку?

Обход std::unordered_map

```
std::unordered_map<std::string, std::string> ...;

for (const std::pair<const std::string, std::string>&
     item : en_to_ru) {
  const auto& en = item.first;
  const auto& ru = item.second;
  std::cout << en << " -> " << ru << '\n';
}
```

Забыли const, без него будет копирование пары

Обход std::unordered_map

```
for (const auto& [en, ru] : en_to_ru) {
    std::cout << en << " -> " << ru << '\n';
}
```

https://godbolt.org/z/6vqf9o866

Обращение к элементу по ключу

```
// Если ключа нет, то исключение std::out_of_range
en_to_ru.at("two") = "два";
```

Операции стандартных контейнеров

CRUD

- ✓ Create
- ✓ Read
- \- Update
- \- Delete

Добавление элемента в std::vector

```
std::size_t n = ...;
std::vector<int> v;



for (std::size_t i = 0; i < n; ++i) {
  v.push_back(f(i));
}
```

Паттерн: reserve

```
std::size_t n = ...;
std::vector<int> v;

v.reserve(n);

for (std::size_t i = 0; i < n; ++i) {
  v.push_back(f(i));
}
```

Всего одна аллокация

Добавление элемента в std::unordered_map

```
// Если ключа нет, то оператор [] создаст его
en_to_ru["four"] = "четыре";

// Если ключ есть, то без эффекта
en_to_ru.insert({"five", "пять"});

// Если ключа нет – добавить, если есть – обновить.
en_to_ru.insert_or_assign("six", "шесть");
```

https://godbolt.org/z/b3zhb5Pb1

Остерегайтесь случайных модификаций

```
std::map<std::string, bool> name_to_flag;

for (const std::string name
     : {"two", "three", "four"}) {

  if (name_to_flag[name]) { /* some op */ }
}
```

https://godbolt.org/z/x9efW7oTb

Операции стандартных контейнеров

CRUD

- ✓ Create
- ✓ Read
- ✓ Update
- \- Delete

Удаление элемента из std::unordered_map

```
en_to_ru.erase(“seven”);
```

С вектором все сложнее.

Итератор

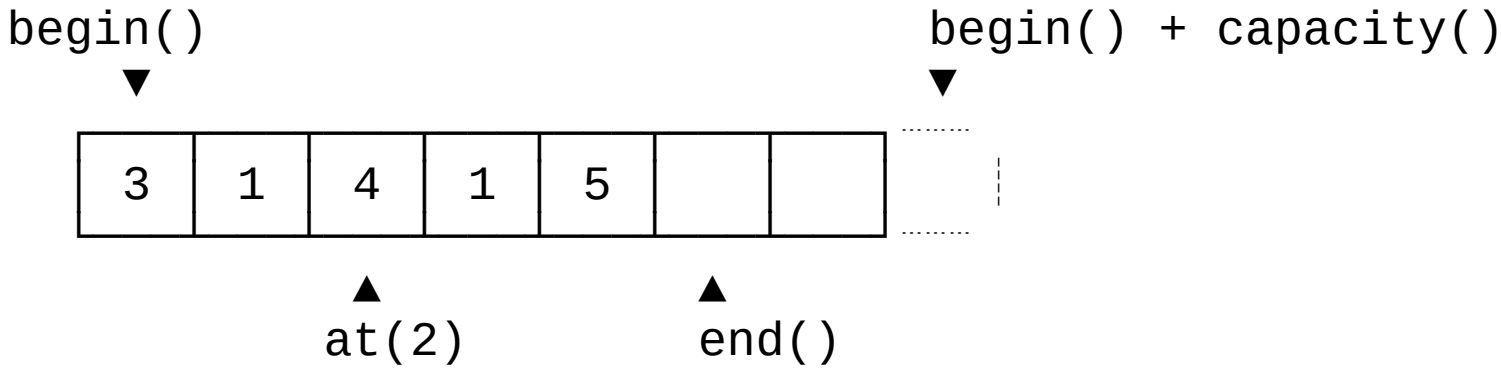
Пока будем считать, что итератор — это нечто похожее на указатель.

```
template <typename InputIt, typename T>
InputIt find(InputIt first, InputIt last,
             const T& value);
```

Здесь параметры шаблона — любые типы,
удовлетворяющие статическому интерфейсу

std::find

Устройство std::vector



std::find

```
int xs[] = {3, 1, 4, 1, 5};
std::vector v{3, 1, 4, 1, 5};

auto it1 = std::find(
    xs, xs + sizeof(xs) / sizeof(*xs), 4);

auto it2 = std::find(v.begin(), v.end(), 4);

std::cout << *it1 << ' ' << *it2 << '\n';
```

std::find_if

```
bool is_even(int x) {
  return x % 2 == 0;
}

std::vector v{3, 1, 4, 1, 5};
auto it1 = std::find_if(v.begin(), v.end(), is_even);
std::cout << *it1 << '\n';
```

std::find_if + lambda

```
std::vector v{3, 1, 4, 1, 5};

auto it2 = std::find_if(
  v.begin(), v.end(),
  [](int x) -> bool { return x % 2 == 0; });

std::cout << *it1 << '\n';
```

erase-remove idiom

```
std::vector v{3, 1, 4, 1, 5};

auto it = std::remove(v.begin(), v.end(), 1);

//                  v-- end()
// [3, 4, 5, ?, ?]
//            ^-- it

v.erase(it, v.end());
```

Remove+lambda+capture

```
std::unordered_set numbers{1, 3};
std::vector v{3, 1, 4, 1, 5};

auto it = std::remove(
  v.begin(), v.end(),
  [](int item) {
    return numbers.find(item) != numbers.end();
  });

v.erase(it, v.end());
```

Remove+lambda+capture

```
std::unordered_set numbers{1, 3};
std::vector v{3, 1, 4, 1, 5};

auto it = std::remove_if(
  v.begin(), v.end(),
  [&numbers](int item) {
    return numbers.find(item) != numbers.end();
  });

v.erase(it, v.end());
```

https://godbolt.org/z/fTd73nxMq

Операции стандартных контейнеров

CRUD

- ✓ Create
- ✓ Read
- ✓ Update
- ✓ Delete

Везде одно и то же (нет)

C++: std::vector, std::unordered_map, std::unordered_set

Python: list, dict, set

Java: ArrayList, HashMap, HashSet

Go: slice, map, map*

Последние ремарки

Можно получить указатель на внутренний буфер:

```
T* std::vector<T>::data();

const char* std::string::c_str();
```

Для путей используйте std::filesystem::path.

Какие данные здесь хранятся?

```
std::map<
  std::string,
  std::map<std::string, std::vector<int>>
>
```

Какие данные здесь хранятся?

```
using Group = std::string;
using Name = std::string;
using Grade = int;
using Grades = std::vector<Grade>;

using StudentToGrades = std::map<Name, Grades>;

using GroupToStudentToGrades
  = std::map<Group, StudentToGrades>;
```

Осталось за кадром

- std::map, std::set
- std::array
- std::stack, std::queue
- std::string_view
- std::flat_map, std::flat_set
- …

Containers library