Знакомство с C++

Механизмы абстракции. Стандартная библиотека.

Краткая хронология

1980: C With Classes

1983: C++

1998: Международный стандарт языка

2003: C++03

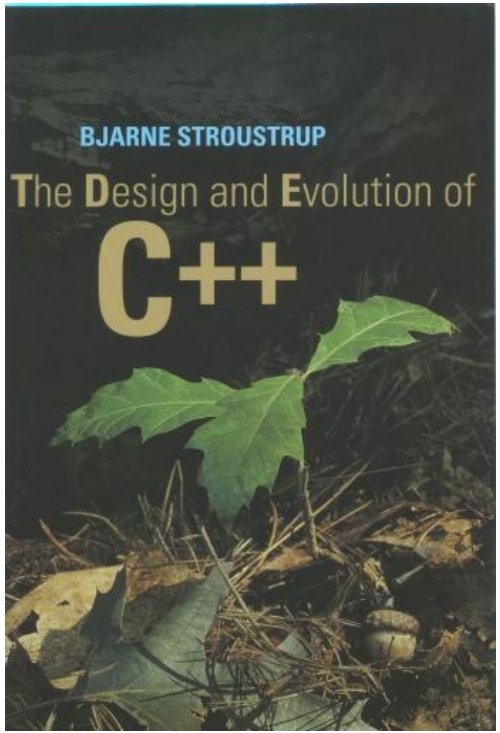
C++11, C++14, C++17, C++20, C++23

Bjarne Stroustrup

Дизайн и эволюция C++

- История разработки языка
- Мотивация технических решений
- Продвижение языка
- Стандартизация
- Планы (В C++20 — уже реальность)

Стоит читать, но значительно позже.



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_print(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)", 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_print(const struct Point* point) {
  printf("POINT(%.6lf %.6lf)", 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_;
  }
};
```

Но здесь можно было бы обойтись передачей по значению

Константность this

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

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

Константность this

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

struct Point {
  ...

  void move(const Vec2d& vec);
  void print() const;
};
```


Constants and immutability

Обобщение классов

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

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

Тип T и, следовательно, его размер неизвестны

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 = ?
```

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
```

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 = ?
```

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
```

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. **Динамический массив**
2. Строка
3. Ассоциативный контейнер (ключ → значение)
4. Множество
5. Алгоритмы
6. Ввод-вывод

Динамический массив

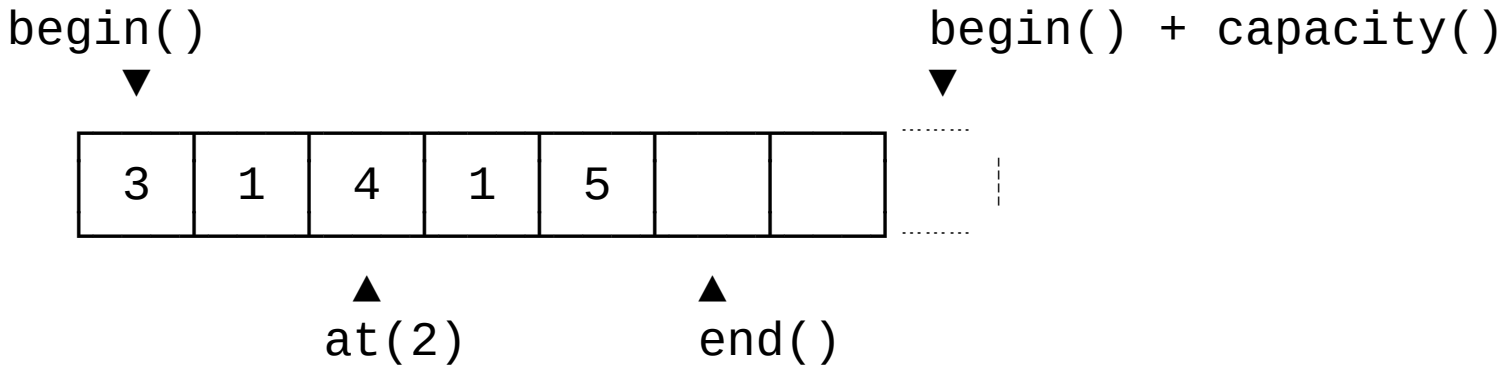
`std::vector<T>`

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]

Устройство std::vector



Обход вектора

```
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*

– *В C++17 можно опустить указание типа элементов*

Обход вектора

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

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


Range-based for loop

Обход вектора

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

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

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

[Meyers01, Item 14]

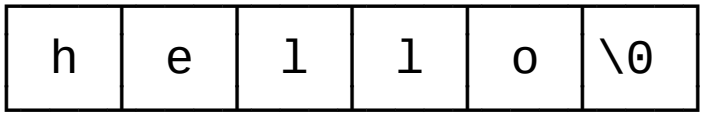
Минимальный набор

1. Динамический массив
2. **Строка**
3. Ассоциативный контейнер (ключ → значение)
4. Множество
5. Алгоритмы
6. Ввод-вывод

Строка

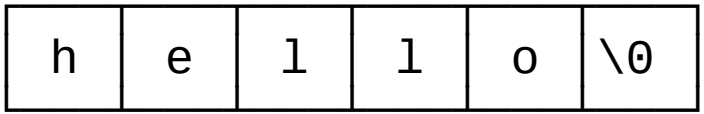
Q: Как представлены строки в языке C?

Устройство C-string



`strlen(const char*) – O(?)`

Устройство C-string

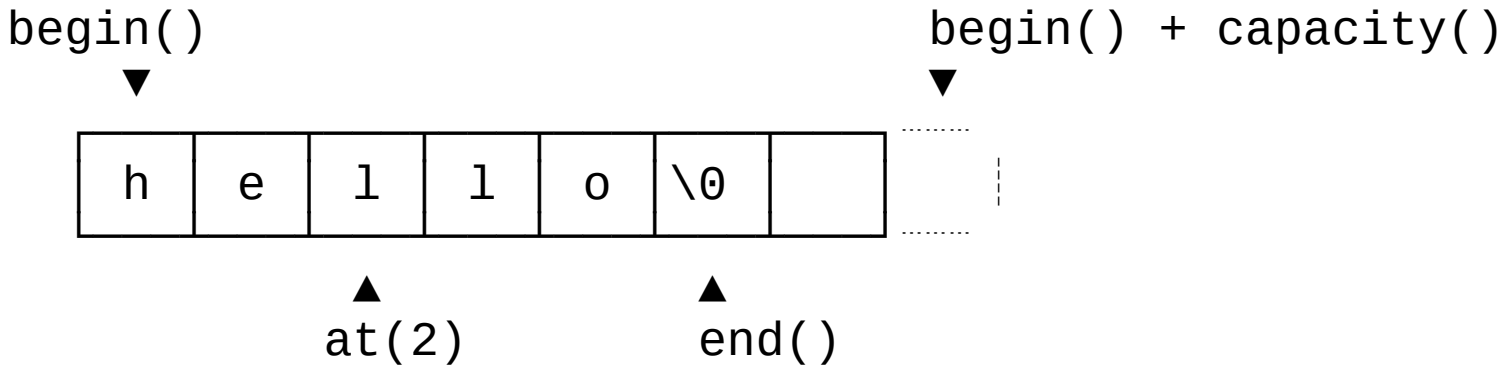



```
strlen(const char*) – O(N)

char *strncpy(char *dest, const char *src, size_t n);
```

Разработчик контролирует выход за границы

Устройство std::string



Обход строки

```
std::string s = “hello”;

for (auto c : s) {
  std::cout << c << ' ';
}
```

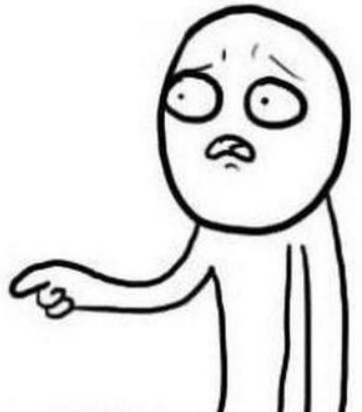
Паттерн: reserve

```
std::size_t n = ...;
std::string s;

s.reserve(n);

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

Всего одна аллокация



Возникают вопросы

Q1: Зачем нужен тип std::string, если есть std::vector<char>?

Q2: Зачем хранить ‘\0’, если размер уже хранится отдельно?

std::string vs std::vector<char>

```
template<
    class CharT,
    class Traits = std::char_traits<CharT>,
    class Allocator = std::allocator<CharT>
> class basic_string;

using string = basic_string<char>;
```


std::char_traits

std::string vs std::vector<char>

SSO: Small String Optimization

```
void* operator new(std::size_t n) {
  std::cout << "Allocate " << n << '\n';
  return malloc(n);
}


int main() {
  std::string s = "hello";
  //std::vector v{'h', 'e', 'l', 'l', 'o'};
}
```


https://godbolt.org/z/KKdrEeMeK

Зачем нужен .c_str() и ‘\0’?

```
const char* c_str() const;
```

Returns a pointer to a null-terminated character array [...]

Настоящее объявление:

```
constexpr const CharT* c_str() const noexcept;
```

Зачем нужен .c_str() и ‘\0’?

```
const char* c_str() const;
```

Returns a pointer to a null-terminated character array [...]

Настоящее объявление:

```
constexpr const CharT* c_str() const noexcept;
```

Мотивация: использование C-API

Проблемы std::string

```
void f(const std::string& s);


std::string s = "long enough to allocate";

f(s);
```

Принимаем параметр по ссылке, нет копирования и аллокаций

Проблемы std::string

```
void f(const std::string& s);


std::string s = "long enough to allocate";

f(s);
f("long enough to allocate");
```

Как избежать аллокаций?

Вариант 1: перегрузка

```
void f(const char* s);
void f(const std::string& s) { f(s.c_str()); }

std::string s = "long enough to allocate";

f(s);
f("long enough to allocate");
```

Вариант 2: std::string_view

```
void f(std::string_view s);


std::string s = "long enough to allocate";

f(s);
f("long enough to allocate");
```


https://godbolt.org/z/n6hx3jWzv

std::string_view

string_view — невладеющий указатель на строку

```
class string_view {
  const char* data;
  size_t length;
};
```


https://godbolt.org/z/759sfdb58

Минимальный набор

1. Динамический массив
2. Строка
3. **Ассоциативный контейнер (ключ → значение)**
4. Множество
5. Алгоритмы
6. Ввод-вывод

Ассоциативные контейнеры

- std::map — RB-tree
- std::unordered_map — Hash table
- std::flat_map — Sorted array (C++23)

Обращение к элементам

```
std::map<std::string, int> text_to_number;

// Вставка
text_to_number[“one”] = 1;
text_to_number.insert({“two”, 2});

// Чтение
auto v1 = text_to_number[“two”];
auto v2 = text_to_number.at(“one”);
auto it = text_to_number.find(“three”);
```


std::map, Unexpected Modification

Обход std::map (C++03)

```
std::map<std::string, int> name_to_number
    {{"one", 1}, {"two", 2}, {"three", 3}};

for (std::map<std::string, int>::const_iterator
     it = name_to_number.begin();
     it != name_to_number.end(); ++it) {
  const std::string& name = it->first;
  const int value = it->second;
  std::cout << name << ": " << number << '\n';
}
```

Обход std::map (C++11, v1)

```
std::map<std::string, int> name_to_number
    {{"one", 1}, {"two", 2}, {"three", 3}};

for (auto it = name_to_number.cbegin();
     it != name_to_number.cend(); ++it) {
  const auto& name = it->first;
  const auto value = it->second;
  std::cout << name << ": " << number << '\n';
}
```

Обход std::map (C++11, v2)

```
std::map<std::string, int> name_to_number
    {{"one", 1}, {"two", 2}, {"three", 3}};

for (const auto& item : name_to_number) {
  const auto& name = item.first;
  const auto number = item.second;
  std::cout << name << ": " << number << '\n';
}
```

auto — не просто синтаксический сахар!

Обход std::map (C++11)

```
for (const std::pair<std::string, int>& item : ...)
```

Обход std::map (C++11)

Создание копии обьекта

```
for (const std::pair<std::string, int>& item : ...)
```

Верный тип:

```
for (const std::pair<const std::string, int>& item : ...)
```


[Meyers14, Item 5]

Обход std::map (C++17)

```
std::map<std::string, int> name_to_number
    {{"one", 1}, {"two", 2}, {"three", 3}};

for (const auto& [name, number] : name_to_number) {
  std::cout << name << ": " << number << '\n';
}
```


Structured binding declaration

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>;
```

Минимальный набор

1. Динамический массив
2. Строка
3. Ассоциативный контейнер (ключ → значение)
4. **Множество**
5. Алгоритмы
6. Ввод-вывод

Множество

- std::set — RB-tree
- std::unordered_set — Hash table
- std::flat_set — Sorted array (C++23)

Минимальный набор

1. Динамический массив
2. Строка
3. Ассоциативный контейнер (ключ → значение)
4. Множество
5. **Алгоритмы**
6. Ввод-вывод

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

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';
```

Минимальный набор

1. Динамический массив
2. Строка
3. Ассоциативный контейнер (ключ → значение)
4. Множество
5. Алгоритмы (find, sort, map, filter, reduce, …)
6. Управление памятью
7. **Ввод-вывод**

#include <iostream> / <fstream>

std::cin, std::cout, std::cerr — stdin, stdout, stderr

std::ifstream, std::ofstream

https://en.cppreference.com/w/cpp/io

fmtlib / std::fmt (C++20)

```
std::cout
  << std::setprecision(6) << std::fixed
  << "POINT(" << pt.x_ << ' ' << pt.y_ << ')'
  << '\n';

// vs

std::cout << fmt::format(
    "POINT({:6f} {:6f})\n", pt.x_, pt.y_);
```

Рекомендации по работе с STL

1. Используйте синонимы (using)*
2. Используйте auto*
3. Не используйте using namespace std;

* Если это облегчает чтение кода