Яндекс

Заглядываем под капот «Поясов по C++»

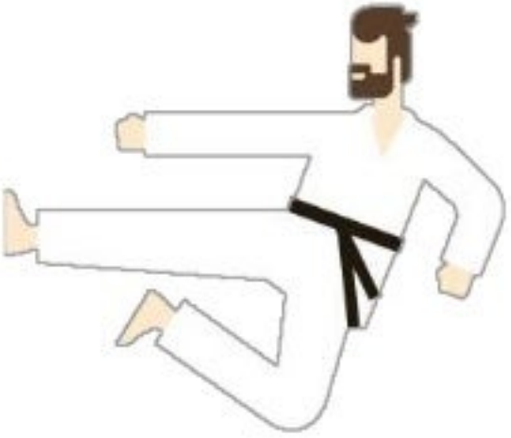
Илья Шишков, старший разработчик

Состав специализации

Рейтинг
4.8/5

Рейтинг
4.9/5

Рейтинг
4.8/5



Целевая аудитория специализации

Люди, владеющие любым языком программирования

- надо знать, что такое переменные, условный оператор и циклы
- иметь представление о принципах объектно-ориентированного программирования
- знать базовые алгоритмы и структуры данных: сортировка, поиск, массив, словарь

Чему хотим научить

После прохождения нашей специализации слушатели должны уметь:

- самостоятельно решать практические задачи на языке C++
- применять естественные для C++ идиомы и конструкции
- самостоятельно находить ответы на свои вопросы и изучать язык глубже
- писать на C++ эффективный код без ущерба для его качества

Наша команда

Антон Полднев
руководитель службы

Евгений Парамонов
руководитель группы

Илья Шишков
старший разработчик

Иван Лежанкин
старший разработчик

Михаил Матросов
Align Technology

Особенности создания онлайн курсов

- Полностью автоматическая проверка работ
- В каждый момент люди должны понимать, зачем они тратят время и деньги
- Важно обеспечить ранний эффект
- Мы передаём свой опыт

Программа курсов

Неделя	Белый пояс	Жёлтый пояс	Красный пояс	Коричневый пояс
1	Обзор возможностей C++ Тестирование и отладка	Целые типы pair и tuple	Введение в макросы Шаблоны классов	Устройство ассоциативных
2	vector, map, set	тестирование	Модель памяти	«Умные» указатели Владение
3	sort, count, count_if Лямбды Пользовательские типы	Многофайловые проекты ODR	Move-семантика	RAII exception safety
4	<fstream> Перегрузка операторов Исключения	Итераторы <algorithm> deque, queue	Устройство линейных контейнеров	const-correctness Compile time vs runtime
5	Финальный проект	Наследование Полиморфизм	Введение в многопоточность	«Хороший» код
6		Финальный проект	Финальный проект	Финальный проект

48 задач

33 задачи

35 задач

31 задача

Программа курсов

Неделя	Белый пояс	Жёлтый пояс	Красный пояс	Коричневый пояс
1	Обзор возможностей C++ Тестирование и отладка if, for, while	Целые типы pair и tuple Шаблоны функций	Введение в макросы Шаблоны классов Принципы оптимизации	Устройство ассоциативных контейнеров
2	Функции, ссылки, const vector, map, set	Юнит-тестирование	Сложность алгоритмов Модель памяти	Пространства имён «Умные» указатели Владение
3	sort, count, count_if Лямбды Пользовательские типы	Многофайловые проекты ODR	Move-семантика	RAII exception safety
4	<fstream> Перегрузка операторов Исключения	Итераторы <algorithm> deque, queue		ss vs
5	Финальный проект	Наследование Полиморфизм	многопоточность	«Хороший» код
6		Финальный проект	Финальный проект	Финальный проект

Тестирование и профилирование

Описание тестирующей системы

Устройство тестирующей системы

Устройство тестирующей системы

Юнит-тест фреймворк

Юнит-тест фреймворк

В видеолекциях мы разработали свой юнит-тест фреймворк

- чтобы показать, что текущих знаний уже достаточно, чтобы сделать что-то полезное
- чтобы люди понимали, как он работает и как устроен внутри
- чтобы они могли вносить в него изменения

Пример применения юнит-тест фреймворка

```
#include "test_runner.h"

int Abs(int x);

void TestPositive() {
  ASSERT_EQUAL(Abs(5), 5);
}

void TestNegative() {
  ASSERT_EQUAL(Abs(-5), 5);
}

int main() {
  TestRunner tr;
  RUN_TEST(tr, TestPositive);
  RUN_TEST(tr, TestNegative);
}
```

```
TestPositive fail: Assertion failed: -5 != 5
hint: Abs(5) != 5, main.cpp:5
TestNegative OK
1 unit tests failed. Terminate
```

Класс TestRunner

```
class TestRunner {
private:
  int fail_count = 0;
public:
  template <class TestFunc>
  void RunTest(TestFunc func, const std::string& test_name) {
    try {
      func();
      std::cerr << test_name << " OK" << std::endl;
    } catch (std::exception& e) {
      ++fail_count;
      std::cerr << test_name << " fail: " << e.what() << std::endl;
    } catch (...) {
      ++fail_count;
      std::cerr << "Unknown exception caught" << std::endl;
    }
  }
```

```
  ~TestRunner() {
    if (fail_count > 0) {
      std::cerr << fail_count << " unit tests failed. Terminate" << std::endl;
      exit(1);
    }
  }
};
```

Стимулируем писать юнит-тесты

- К каждой задаче выдаём заготовку решения с юнит-тестами

```
// Реализуйте этот шаблон
template <typename T> void Swap(T* first, T* second);

void TestSwap() {
  int a = 1;
  int b = 2;
  Swap(&a, &b);
  ASSERT_EQUAL(a, 2);
  ASSERT_EQUAL(b, 1);
}

int main() {
  TestRunner tr;
  RUN_TEST(tr, TestSwap);
}
```

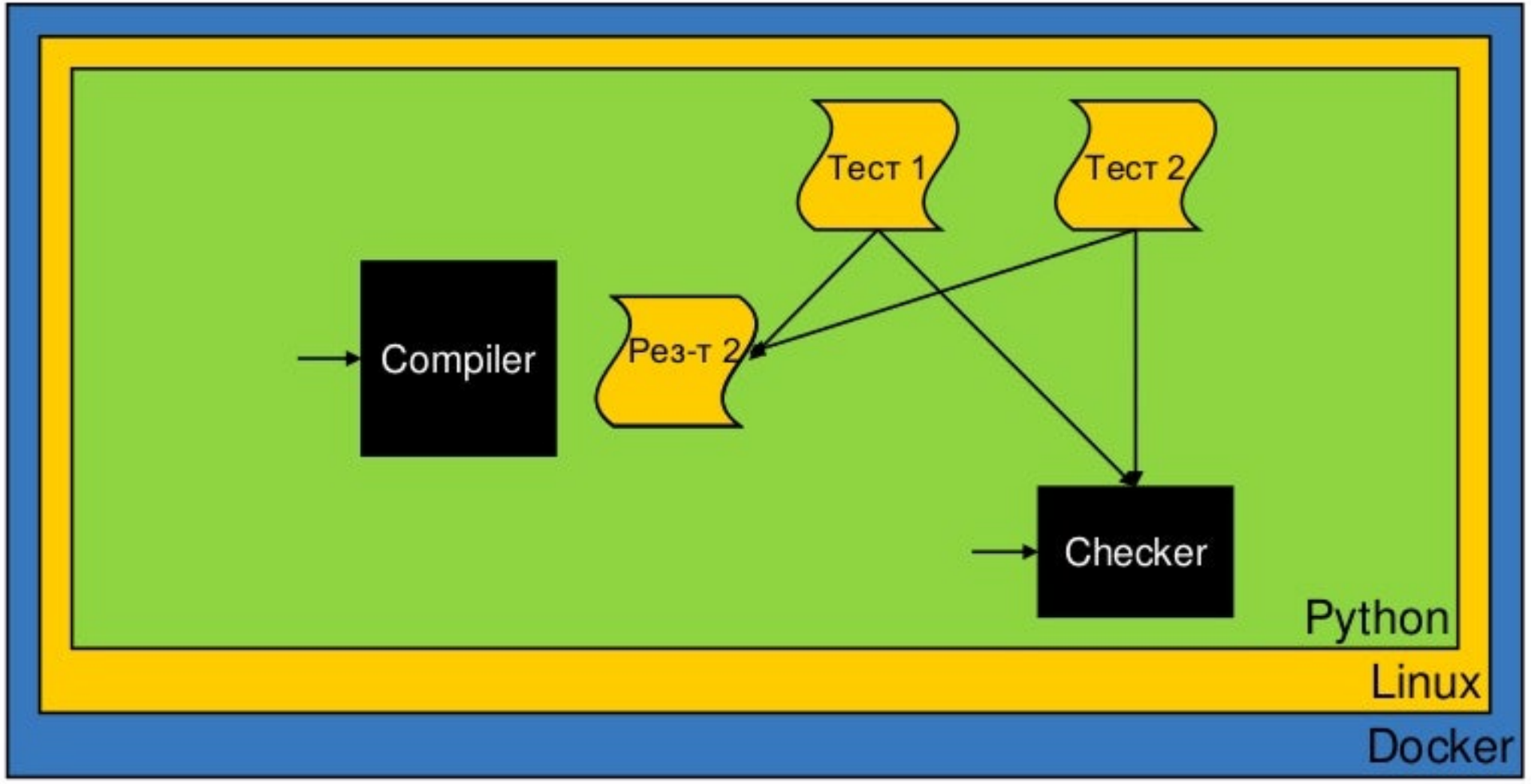
Тестирование решений участников

Типы задач на курсах

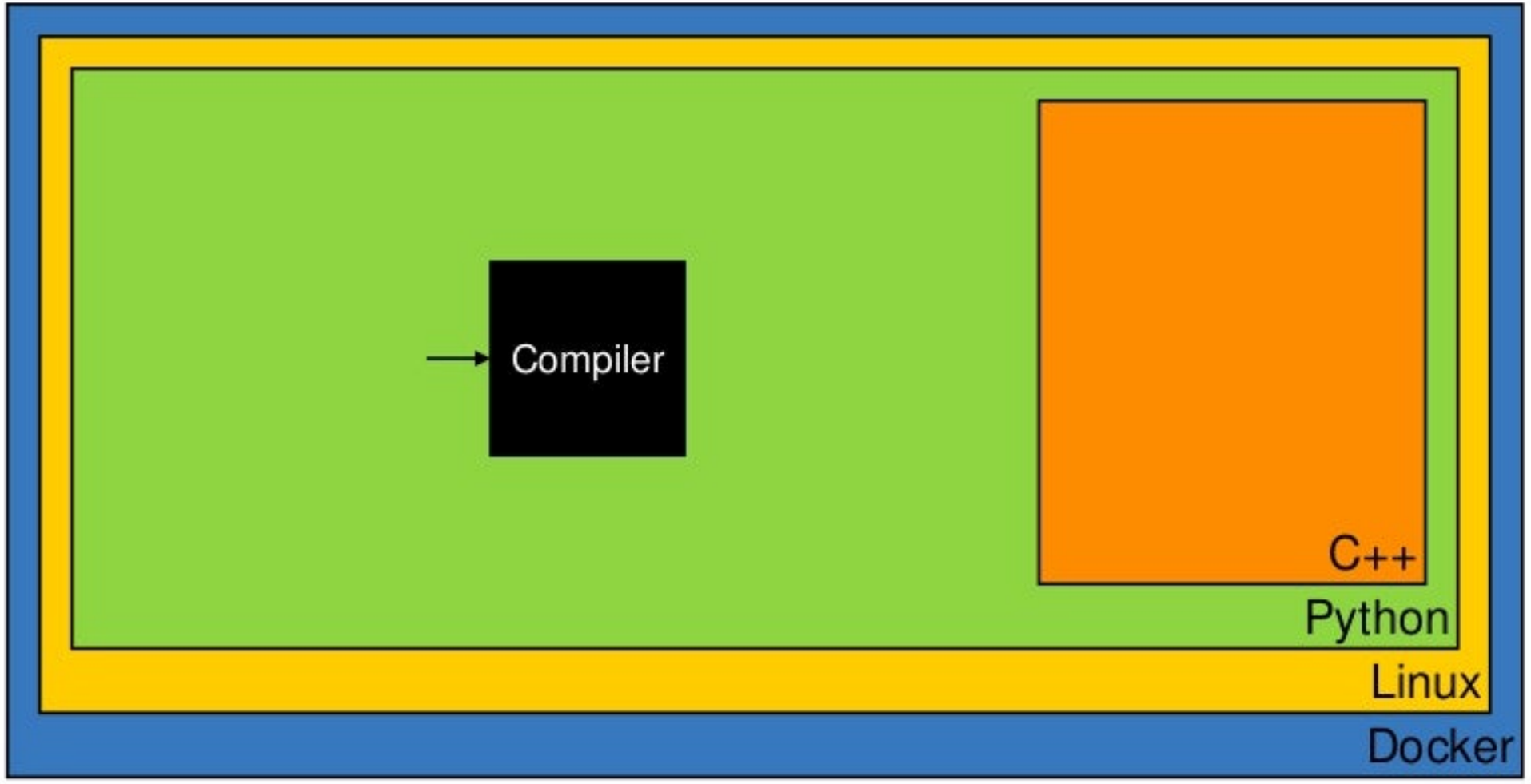
Большинство задач относится к одной из двух категорий:

- Написать программу stdin → stdout
- Реализовать функцию/класс/шаблон с заданным интерфейсом

Тестирование задач stdin → stdout



Тестирование реализации интерфейса



Проблема с функцией main

```
// Реализуйте этот шаблон
template <typename T>
void Swap(T* first, T* second) {
  std::iter_swap(first, second);
}

void TestSwap() {
  int a = 1;
  int b = 2;
  Swap(&a, &b);
  ASSERT_EQUAL(a, 2);
  ASSERT_EQUAL(b, 1);
}

int main() {
  TestRunner tr;
  RUN_TEST(tr, TestSwap);
}
```

```
int main() {
  string left = "leftstring";
  string right = "rightstring";

  Swap(&left, &right);
  belts::Assert(left == "rightstring");
  belts::Assert(right == "leftstring");
}
```

Как удалить функцию main?

- Просить участников удалять её из своих файлов перед посылкой
  - «Файл, присланный на проверку, не должен содержать функцию main. Если в нём будет функция main, вы получите ошибку компиляции»
- Неудобно!

Как удалить функцию main?

- Автоматически удалять из присланного файла
  - Регуляркой находим строку «int main(»
  - Двигаемся дальше, считая баланс фигурных скобок
  - Как только он стал нулевым, удаляем выбранный фрагмент

- Хрупкое решение — баланс скобок не учитывает комментарии

```
int main() {
  int  x;
  cin >> x;
//  if (x > 0) {
  return x;
}
```

Как удалить функцию main?

Её не надо удалять — её достаточно переименовать!

- Это решение работает как часы уже 1,5 года

```
template <typename T>
void Swap(T* first, T* second) {
  std::iter_swap(first, second);
}

void TestSwap() { … }

int main() {
  TestRunner tr;
  RUN_TEST(tr, TestSwap);
}
```

```
int main() {
  string left = "leftstring";
  string right = "rightstring";

  Swap(&left, &right);
  belts::Assert(left == "rightstring");
  belts::Assert(right == "leftstring");
}
```

Тестирование интерфейса. Итоги

- «Удаление» функции main делает работу с тестирующей системой удобнее
- Трюк с переименованием функции помог нам сэкономить время
- Это простое решение, которое надёжно работает более 1,5 лет

Ограничение использования стандартных контейнеров

Задача SimpleVector

- В задаче надо реализовать сильно упрощённый вектор

- Простейший способ — использовать std::vector<T>

```
template <typename T> class SimpleVector {
public:
  SimpleVector() = default;
  explicit SimpleVector(size_t size);
  ~SimpleVector();
  T& operator[](size_t index);
  T* begin();
  T* end();
  size_t Size() const;
  size_t Capacity() const;
  void PushBack(const T& value);
};
```

Как запретить использовать std::vector?

Парсить решение участника и анализировать секцию private шаблона SimpleVector

- Сложно
- Хрупко

Как запретить использовать std::vector?

Как запретить использовать std::vector?

У нас фиксирована версия компилятора!

- Смотрим на имена include-guard'ов в нашей реализации стандартной библиотеки
- Вызываем #error, если они видны

```
__SUBMISSION__    // Заменяется Python'ом на содержимое файла участника
#ifdef _GLIBCXX_VECTOR
#error "You are not allowed to use header file <vector> in this problem"
#endif
#ifdef _GLIBCXX_DEQUE
#error "You are not allowed to use header file <deque> in this problem"
#endif
```

```cpp
int main() {
  SimpleVector<int> values(5);
  belts::Assert(values.Size() == 5u);
  belts::Assert(values.Size() <= values.Capacity());
  values[2] = 123;
  belts::Assert(values[2] == 123);
}
```

Краткий итог

- Ограничиваем использование стандартных контейнеров, анализируя видимые include-guard’ы
- Переход от общей задачи к её частному случаю позволил создать простое решение
- Оно сэкономило нам массу времени и надёжно работает

Compile-time проверка интерфейса

Задача SimpleVector

Как проверить, что присланный шаблон имеет требуемый интерфейс?

- Например, что operator[] возвращает T&

```
template <typename T> class SimpleVector {
public:
  SimpleVector() = default;
  explicit SimpleVector(size_t size);
  ~SimpleVector();
  T& operator[](size_t index);
  T* begin();
  T* end();
  size_t Size() const;
  size_t Capacity() const;
  void PushBack(const T& value);
};
```

Тестирование интерфейса SimpleVector

Можно запускать компиляцию и возвращать сообщение компилятора, если она не удалась

```
__SUBMISSION__    // Заменяется Python'ом на содержимое файла участника
int main() {
    SimpleVector<int> v(1);
    int& x = v[0];      // Проверяем, чт      озвращает T&
}

error: cannot bind non-c                    'int&' to an rvalue of type 'int'
int& x = v[0];                                  int&,
  ~~~^
```

Compile-time проверка интерфейса

Если человек ошибся в интерфейсе, не стоит отдавать ему сообщения компилятора:

- они могут быть громоздкими
- могут сбивать с толку

Нужен способ проверять интерфейс в compile time и отдавать внятные сообщения

- И мы нашли такой способ — Detector idiom

Detector idiom

- Walter E. Brown «Modern Template Metaprogramming: A Compendium», CppCon 2014
- Marshall Clow «The 'Detection idiom:' A Better Way to SFINAE», C++Now 2017
- Ivan Čukić «2020: A void_t odyssey», C++ Russia 2018

Detector idiom

```
#include "detector.h"  // Реализация Detector idiom

template <typename T>
using HasIndexOperator = decltype(std::declval<T>()[0]);

static_assert(
  std::is_same_v<
    detected_t<HasIndexOperator, SimpleVector<int>>,
    int&
  >,
  "Member function T& operator[](size_t) not found in SimpleVector<T>"
);
```

Detector idiom

Если слушатель будет возвращать не ссылку из operator[], он получит внятное сообщение

```
error: static assertion failed: Member function T& operator[](size_t) not found in SimpleVector<T>
```

Compile-time проверка интерфейса. Итоги

- Сообщения компилятора могут сбивать с толку начинающих программистов на C++
- Мы стараемся более внятно сообщать участникам, что не так в их коде
- Detector idiom сильно упрощает исследование интерфейса класса в compile time

Результаты

Курс	Белый пояс	Жёлтый пояс	**Красный пояс**
Дата запуска	Июнь 2017	Декабрь 2017	Июль 2018
Активные участники	17 529	2590	582
Выпускники	1 061	269	28
Оценка	4,8/5	4,9/5	4,8/5

> Этот курс показал мне нормальный C++, а не тот, которому меня учили в университете.

Отзыв на «Белый пояс по C++»

На работе пригодились знания по декомпозиции, алгоритмам и юнит-тестированию

Отзыв на «Жёлтый пояс по C++»

> Один из лучших курсов, что я проходил в жизни. Не только по языку программирования.

Отзыв на «Красный пояс по C++»

Практичность

```
void PrintSize(const vector<Person>& people) {
  cout << people.size() << endl;
}
int main() {
  vector<Person> people(15'000'000);
  PrintSize(people);
}
```

Простота

Вы сделали курс настолько понятным, что даже семиклассница смогла его пройти. Благодаря вам, я сделала первый шаг к своей мечте!

Спасибо

Илья Шишков

Старший разработчик компании Яндекс

 ishfb@yandex-team.ru

 ishfb

 telegram: ishfb