

Введение в C++

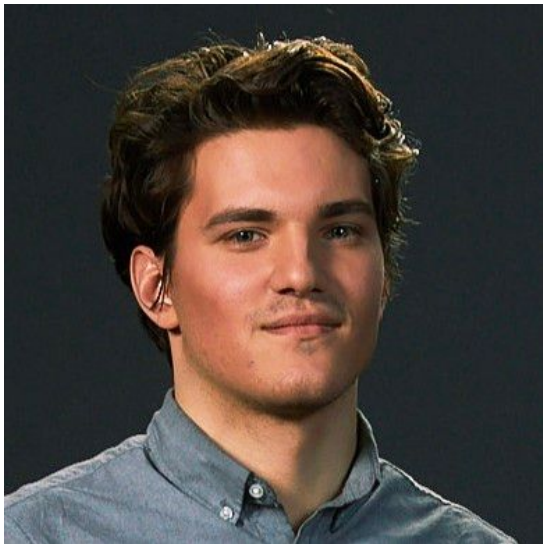
Антон Кухтичев



О преподавателях

Иван Возвахов

Руководитель команды
разработки встроенного ПО



Антон Кухтичев

Ведущий программист группы
развития Юлы



Состав курса

- Введение в C++
- Препроцессор, компилятор, компоновщик
- Память в C++
- Функции
- Классы и методы классов
- Сору и move-семантика
- Шаблоны
- STL
- Исключения
- И ещё много чего...

Лекции, задания домашних работ и примеры будут тут:

https://github.com/mailcourses/miph_basic_cpp_autumn_2023

О домашних заданиях (1)

- В вашем GitHub должен быть репозиторий `mipt_cpp_autumn_2023`;
- Репозиторий должен быть закрытым (private);
- Внутри репозитория должны быть директории из двух цифр, вида: 01, 02 и т. д. — это номера домашних заданий;
- Внутри каждой директории могут быть любые файлы реализующие задачу. Обязательным является только файл `Makefile` (зависит от семинариста);
- В `Makefile` обязательно должны быть цель `test`, которая запускает тесты вашего решения;
- Собираясь ваш код должен компилятором, поддерживающим стандарт C++20;

О домашних заданиях (2)

- Внешних зависимостей быть не должно;
- Код решения должен быть отформатирован, так проще его читать. Не забывайте про отступы;
- О том, что вы выполнили работу надо сообщать своему семинаристу, к комментарию необходимо добавить Вашу ссылку на GitHub;
- Максимальное количество попыток сдачи одного задания - 3.

Для допуска к экзамену должны быть выполнены **ВСЕ** задания!

Рекомендуемая литература



Брюс Эккель

Герб Саттер

Скотт Мейерс

Герберт Шилдт

Бьерн Страуструп

Джош Лоспинозо

Alex Allain

Практика: <https://leetcode.com/>

C++ Styleguide от Google: <https://google.github.io/styleguide/cppguide.html>

Полезные каналы: <https://t.me/cppproglib>

https://youtu.be/18c3MTX0PK0?si=YSsJprCok_H-f870

Содержание занятия

- История C++
- Объектно-ориентированное программирование
- Тривиальные типы данных
- Инициализация
- `std::string`^{чуть-чуть}
- `auto`
- Потоки (streams, не threads)
- Циклы, условные операторы
- Отладка программ

История C++



Эволюция языка C++

1979

Бьерн Страуструп
создал C с классами

1983

Были добавлены
виртуальные функции,
перегрузка функций и
операторов, ссылки,
константы

Стандартизация языка
STL

1998

2003

Следующая версия
стандарта

auto, семантика
перемещения,
constexpr

2011

2014

Вывод типа
возвращаемого
значения для
функций,
Обобщённые лямбда-
функции

Модули

2020

Объектно-ориентированное программирование



Объектно-ориентированное программирование

Все языки ООП характеризуются тремя общими признаками:

1. Инкапсуляция
2. Полиморфизм
3. Наследование

Инкапсуляция

Инкапсуляция (англ. encapsulation, от лат. in capsula) — в информатике, процесс разделения элементов абстракций, определяющих её структуру (данные) и поведение (методы); инкапсуляция предназначена для изоляции контрактных обязательств абстракции (протокол/интерфейс) от их реализации. ^{Википедия}

Полиморфизм

Полиморфизм — это свойство, позволяющее использовать один интерфейс для целого класса действий.

В C++ два вида полиморфизма:

1. Полиморфизм на этапе компиляции — шаблоны
2. Полиморфизм на этапе выполнения — виртуальные классы

Наследование

Наследование — процесс, благодаря которому один объект может приобретать свойства другого.



Тривиальные типы данных



Целочисленные типы

Тип	Знаковый	Размер в байтах				Спецификатор формата printf
		32-битная ОС		64-битная ОС		
		Windows	Linux/ macOS	Windows	Linux/ macOS	
short	Да	2	2	2	2	%hd
unsigned short	Нет	2	2	2	2	%hu
int	Да	4	4	4	4	%d
unsigned int	Нет	4	4	4	4	%u
long	Да	4	4	4	8	%ld
unsigned long	Нет	4	4	4	8	%lu
long long	Да	8	8	8	8	%lld
unsigned long long	Нет	8	8	8	8	%llu

Как получить максимальное/минимальное значения `int` в коде? На подумать...

1. Присвоить максимальное значение, если оно дано по условию

```
#define INT_MAX 10000
```

2. Самому определить переменную

```
const int MY_INT_MAX = 2147483647;
```

3. Использовать предопределённый макрос из `<climits>`

```
int max_val = INT_MAX;
```

Как получить максимальное/минимальное значения `int` в коде?

```
#include <limits> // необходимо для numeric_limits

int main()
{
    // посчитаем для типа int:
    std::cout << "min value: " << std::numeric_limits<int>::min() << "\n"
               << "max value: " << std::numeric_limits<int>::max() << "\n";
}
```

Типы с плавающей точкой

1. `float` — одинарная точность
2. `double` — двойная точность
3. `long double` — повышенная точность



Булевский тип

1. Единственный тип `bool`
2. Принимает одно из значений: `true` или `false`



Символьные типы

1. `char` — тип по умолчанию, размером всегда в 1 байт. Может быть знаковым или беззнаковым. (Пример: ASCII.)
2. `char16_t` — используется для 2-байтовых наборов символов. (Пример: UTF 16.)
3. `char32_t` — используется для 4-байтовых наборов символов. (Пример: UTF 32.)
4. `signed char` — то же, что и `char`, но гарантированно знаковый.
5. `unsigned char` — то же, что и `char`, но гарантированно беззнаковый.
6. `wchar_t` — достаточно большой, чтобы содержать самый большой символ языкового стандарта реализации.

Экранированные последовательности

Значение	Экранированная последовательность
Табуляция (горизонтальная)	\t
Новая строка	\n
Табуляция (вертикальная)	\v
Возврат на одну позицию	\b
Возврат каретки	\r
Прогон страницы	\f
Знак вопроса	? или \?
Одинарная кавычка	\'
Двойная кавычка	\"
Обратная косая	\\
Нулевой символ	\0
Оповещение	\a

Инициализация



Инициализация базового типа нулевым значением

```
int a = 0;    // Инициализируется значением 0
```

```
int b{};      // Инициализируется значением 0
```

```
int c = {};   // Инициализируется значением 0
```

```
int d;        // Инициализируется значением 0 (возможно)
```


Инициализация базового типа произвольным значением

`int e = 42;` // Инициализируется значением 42

`int f{ 42 };` // Инициализируется значением 42

`int g = { 42 };` // Инициализируется значением 42

`int h(42);` // Инициализируется значением 42

std::string

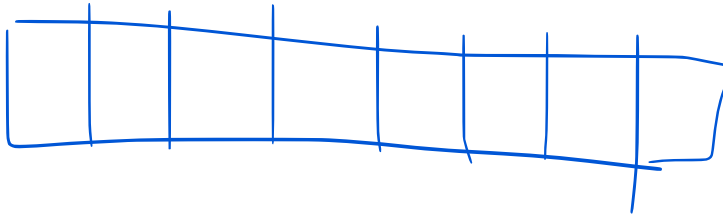


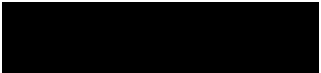
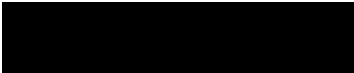
std::string

1. `std::string` для `char`; используется для кодировок, таких как ASCII;
2. `std::wstring` для `wchar_t`; достаточно большой, чтобы содержать самый большой символ языкового стандарта реализации;
3. `std::u16string` для `char16_t`; используется для кодировок, таких как UTF 16;
4. `std::u32string` для `char32_t`; используется для кодировок, таких как UTF32.

size

std::string



- `empty()`
- `size() / length()`
- `resize(size_type n)`
- 
- 
- `clear()`
- `operator+`
- `push_back(...)`
- `substr(size_type pos = 0, size_type count = npos)`



auto



auto

Позволяет статически определить тип по типу выражения.

```
auto i = 5;
```

```
auto j = foo();
```



1. Скотт Мейерс. Эффективный и современный C++. Пункт 1.2. Вывод типа auto.

Потоки (streams)



Streams

STL предоставляет несколько глобальных объектов потока в заголовке `<iostream>`, которые обрабатывают потоки ввода, вывода и ошибок стандартного ввода, вывода и вывода ошибок. Делятся на **форматированные** и **неформатированные**.

<code>cout</code>	<code>ostream</code>	Вывод, например на дисплей
<code>wcout</code>	<code>wostream</code>	
<code>cin</code>	<code>istream</code>	Ввод, например с клавиатуры
<code>wcin</code>	<code>wistream</code>	
<code>cerr</code>	<code>ostream</code>	Вывод ошибки (небуферизованный)
<code>wcerr</code>	<code>wostream</code>	
<code>clog</code>	<code>ostream</code>	Вывод ошибки (буферизованный)
<code>wclog</code>	<code>wostream</code>	

Форматированные операции

Весь форматированный ввод/вывод проходит через две функции: стандартные операторы потока, `operator<<` и `operator>>`.

```
ostream& operator<<(ostream&, char);
```

```
istream& operator>>(istream&, char);
```

Неформатированные операции ввода

Метод	Описание
<code>is.get([c])</code>	Возвращает следующий символ или записывает в символьную ссылку <code>c</code> , если тот предоставлен
<code>is.get(s, n, [d])</code>	Операция <code>get</code> считывает до <code>n</code> символов в буфер <code>s</code> , останавливаясь, если встречается символ новой строки, или <code>d</code> , если тот предоставлен.
<code>is.getline(s, n, [d])</code>	Операция <code>getline</code> делает то же самое, за исключением того, что она также читает символ новой строки. Оба пишут завершающий нулевой символ в <code>s</code> . Нужно убедиться, что в <code>s</code> достаточно места
<code>is.putback(c)</code>	Если <code>c</code> — последний извлеченный символ, выполняется <code>unget</code> .
<code>is.unget()</code>	Помещает последний извлеченный символ обратно в строку

Неформатированные операции вывода

Метод	Описание
<code>os.put(c)</code>	Записывает c
<code>os.write(s, n)</code>	Записывает n символов из s в поток
<code>os.flush()</code>	Записывает все буферизованные данные на текущее устройство

Манипуляторы

Манипуляторы — это специальные объекты, которые изменяют то, как потоки интерпретируют ввод или формат вывода.

- `std::ws` изменяет `istream`, чтобы пропустить пробелы;
- `std::flush` очищает любой буферизованный вывод непосредственно в `ostream`;
- `std::ends` отправляет нулевой байт;
- `std::endl` похож на `std::flush` за исключением того, что он отправляет новую строку перед сбросом.
- `std::setw` задаёт минимальную ширину строк
- `std::setfill` задаёт символ, которым будет заполнен недостающая ширина строки

Состояние потока

Состояние потока указывает, произошел ли сбой ввода/вывода.

Метод	Состояние	Значение
good()	goodbit	Поток находится в хорошем рабочем состоянии
eof()	eofbit	Поток достиг конца файла
fail()	failbit	Операция ввода или вывода завершилась неудачно, но поток все еще может находиться в хорошем рабочем состоянии
bad()	badbit	Произошла катастрофическая ошибка, и поток не в хорошем состоянии

Циклы, условные операторы, switch



Условные операторы

- Классический if

```
if (<выражение>)
```

- Классический if с ветвями и else

```
if ()
```

```
else if ()
```

```
else ()
```

- Инициализация локальной переменной в if/switch ^{C++17}

```
if (auto it = m.find(key); it != m.end())
```

```
    return it->second;
```



Циклы

- `for (size_t i = 0; i < items.size(); ++i)`
- `for (auto &x : items)`^{C++11}
- `for (auto thing = f(); auto& x : thing.items())`^{C++20}
- `while(<выражение>)`
- `do { //... } while(<выражение>)`

switch

```
switch (выражение-инициализации; условие) {  
    case (случай-a): {  
        // Обработка случая-a  
        break;  
    }  
    case (случай-b): {  
        // Обработка случая-b  
        break;  
    }  
    // Обработка других условий при необходимости  
    default: {  
        // Обработка случая по умолчанию  
    }  
}
```



Отладка программ



Средства исследования

1. gdb
2. valgrind
3. gprof



Домашнее задание



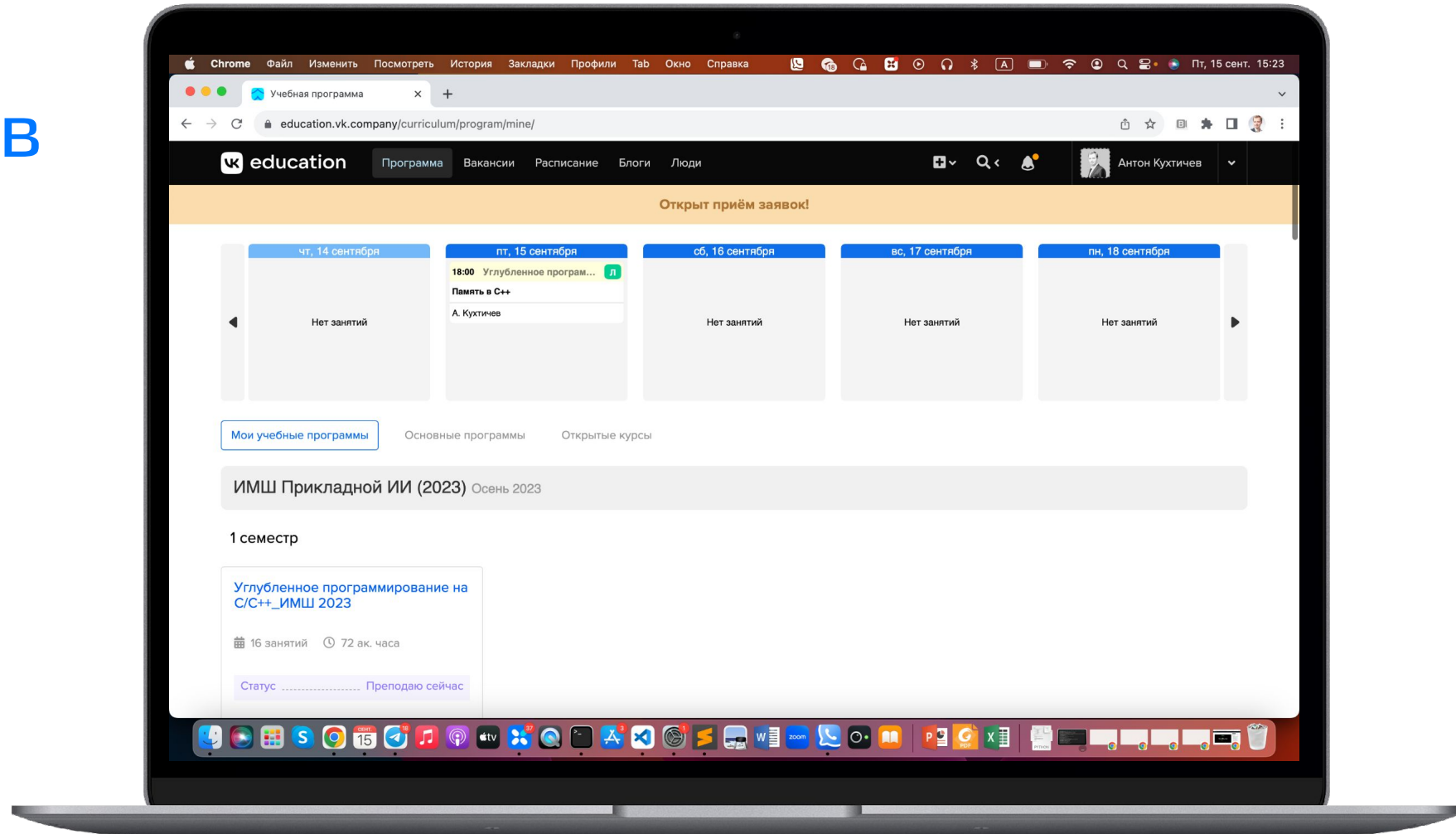
Домашнее задание #1

1. Реализовать алгоритм фонетического сходства двух строк.
2. Написать тесты, покрывающие исходный код, при помощи gtest

```
std::string text1{"Ashcraft"};  
std::string text2{"Ashcroft"};  
assert( isEqual(text1, text2) );  
assert( convertTextToSound(Ashcraft) == std::string{"A261"} );
```

Напоминание оставить отзыв

Это правда важно





Спасибо
за внимание!