

# Конструирование программного обеспечения

ПРОГРАММНАЯ  
ИНЖЕНЕРИЯ

Лекция 1

**Введение**



НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ  
НИЖНИЙ НОВГОРОД



## Обо мне

### **AI Research Engineer / Scientist @ Intel Corp.**

C++, Python, DL/ML

### **Также работал в:**

HARMAN (a Samsung company)

ИФМ РАН

ИПФ РАН

### **Контакты:**

[+79107917732](tel:+79107917732) (phone, Telegram)



# Организационная структура курса



## Длительность

3 модуля: сентябрь -> март

## Частота

1 пара лекций в неделю

1 пара практики в неделю

## Отчетность

Устный экзамен в конце курса (последняя неделя 3-го модуля)

# Тематическая структура курса



## Классы

Базовый синтаксис и понятия

## Standard Template Library (STL)

Ключевые классы, позволяющие решать задачи без изобретения велосипеда

## Объектно-ориентированное программирование (ООП)

Общепризнанная парадигма эффективной разработки ПО на уровне исходного кода

## Паттерны ООП

Наиболее часто встречающиеся приёмы в ООП

## Особенности новых стандартов C++ и продвинутые темы

move-семантика, правые ссылки, coroutines, ranges, modules, concepts, ...

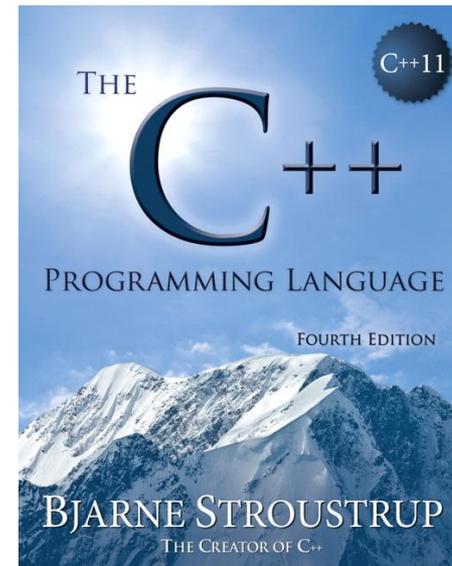
Введение

# Рекомендованная литература



## **The C++ Programming Language (4th Edition)**

Bjarne Stroustrup



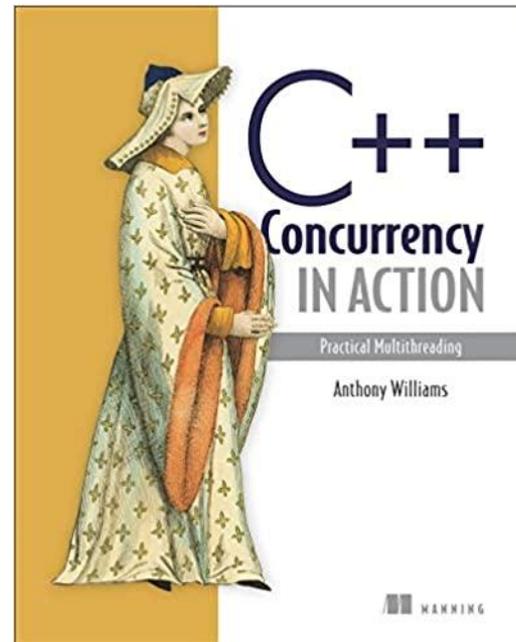
Введение

# Рекомендованная литература



## **C++ Concurrency In Action (2<sup>nd</sup> edition, 2019)**

Anthony Williams





cppreference.com

Create account

Search

Page Discussion

View View source History



CppCon 2021

It's the annual, week-long gathering for the entire C++ community. Register now!

## C++ reference

C++98, C++03, C++11, C++14, C++17, C++20, C++23 | Compiler support C++11, C++14, C++17, C++20, C++23

Freestanding implementations

## Language

- Basic concepts
- Keywords
- Preprocessor
- Expressions
- Declaration
- Initialization
- Functions
- Statements
- Classes
- Overloading
- Templates
- Exceptions

## Headers

## Named requirements

## Feature test macros (C++20)

## Language support library

- Type support – traits (C++11)
- Program utilities
- Coroutine support (C++20)
- Three-way comparison (C++20)
- numeric\_limits – type\_info
- initializer\_list (C++11)

## Concepts library (C++20)

## Diagnostics library

## General utilities library

- Smart pointers and allocators
- unique\_ptr (C++11)
- shared\_ptr (C++11)
- Date and time
- Function objects – hash (C++11)
- String conversions (C++17)
- Utility functions
- pair – tuple (C++11)
- optional (C++17) – any (C++17)
- variant (C++17) – format (C++20)

## Strings library

- basic\_string
- basic\_string\_view (C++17)
- Null-terminated strings:
- byte – multibyte – wide

## Containers library

- array (C++11) – vector – deque
- map – unordered\_map (C++11)
- set – unordered\_set (C++11)
- priority\_queue – span (C++20)
- Other containers:
- sequence – associative
- unordered associative – adaptors

## Iterators library

## Ranges library (C++20)

## Algorithms library

## Numerics library

- Common math functions
- Mathematical special functions (C++17)
- Numeric algorithms
- Pseudo-random number generation
- Floating-point environment (C++11)
- complex – valarray

## Localizations library

## Input/output library

- Stream-based I/O
- Synchronized output (C++20)
- I/O manipulators

## Filesystem library (C++17)

## Regular expressions library (C++11)

- basic\_regex – algorithms

## Atomic operations library (C++11)

- atomic – atomic\_flag
- atomic\_ref (C++20)

## Thread support library (C++11)

- thread – mutex
- condition\_variable

<https://en.cppreference.com>

# История C++



- 1979 - появление нового языка под названием “С с классами”
- 1983 - языку дают название “C++”, большая часть функционала современного C++ уже реализована
- 1987 - впервые выпущен GNU C++ компилятор как часть GCC
- 1990-1993 - поддержка шаблонов, исключений, пространств имен
- 1998 - опубликован **первый стандарт C++ (C++98)**
- 2011 - опубликован **стандарт C++11 published** (серьезные улучшения в качестве жизни разработчиков C++)
- 2014 - опубликован стандарт C++14
- 2017 - опубликован стандарт C++17



## C++ - стандартизованный язык

Существуют конкретные документы, описывающие ключевую функциональность языка, которую *должны* поддерживать имеющиеся компиляторы и стандартные библиотеки C++

<https://isocpp.org/std/the-standard>

## Q: Why is the standard hard to read? I'm having trouble learning C++ from reading it.

The standard is not intended to teach how to use C++. Rather, it is an international treaty – a formal, legal, and sometimes mind-numbingly detailed technical document intended primarily for people writing C++ compilers and standard library implementations.

# Стандарты C++



Новые ревизии стандартов добавляют новый функционал и ключевые слова (и иногда меняют имеющееся поведение)

**C++98**

**C++03**

**C++11**

**C++14**

**C++17**

**C++20**

How should we call C++20 and the beyond area?

C++98 - C++03: Legacy C++

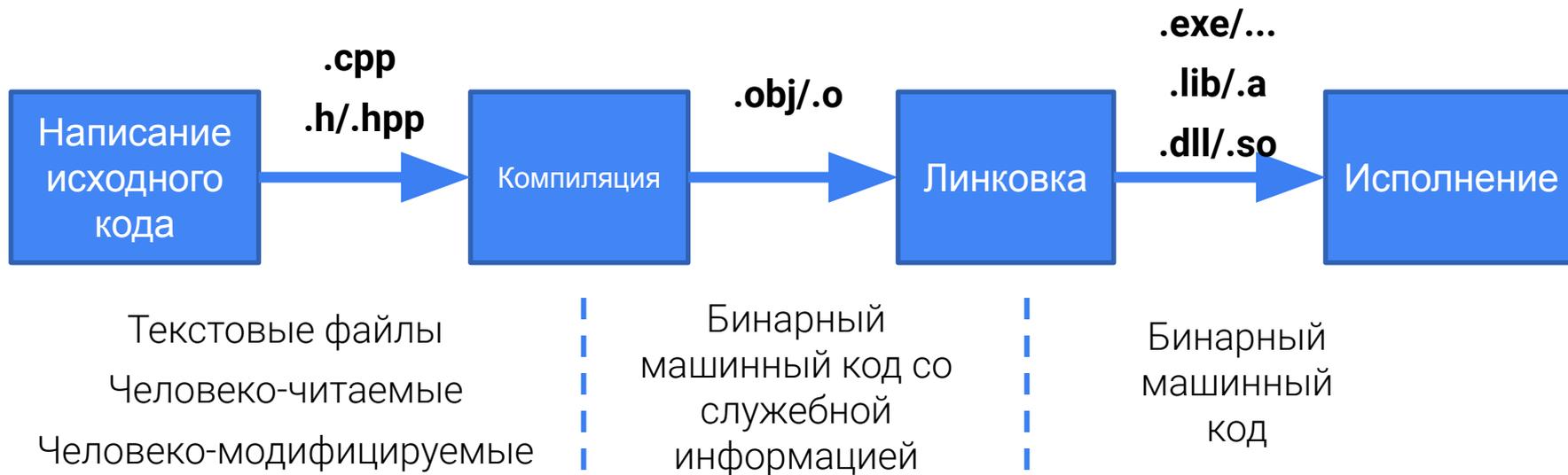
C++11 - C++17: Modern C++

C++20 - C++.. :

Honestly, I don't know.

— Rainer Grimm (@rainer\_grimm) [April 28, 2019](#)

# Базовый процесс программирования на C++





## C++ - «надмножество» языка C

Работающий код на C почти всегда будет работающим, будучи скомпилированным C++ компилятором, и выдавать тот же самый результат

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("Hello, World!\n");
    int foo;
    scanf("%d", &foo);
    printf("%d", foo + 2);
    return 0;
}
```



## Встроенные типы данных

Тип	Ключевое слово	Типичный диапазон значений
Символ (однобайтный)	char	-128 ... 127
Целое число	int	-2147483648 ... 2147483647
Число с плавающей точкой (одинарная точность)	float	+ $-3.4E-38$ ... + $3.4E+38$
Число с плавающей точкой (двойная точность)	double	+ $-1.7E-308$ .. + $1.7E+308$
<b>Булевское значение</b>	bool	<b>true, false</b>
<b>Символ («широкий»)</b>	wchar_t	-32768...32767



## C - ~32 ключевых слова

<u>auto</u>	<u>break</u>	<u>case</u>	<u>char</u>
<u>const</u>	<u>continue</u>	<u>default</u>	<u>do</u>
<u>double</u>	<u>else</u>	<u>enum</u>	<u>extern</u>
<u>float</u>	<u>for</u>	<u>goto</u>	<u>if</u>
<u>int</u>	<u>long</u>	<u>register</u>	<u>return</u>
<u>short</u>	<u>signed</u>	<u>sizeof</u>	<u>static</u>
<u>struct</u>	<u>switch</u>	<u>typedef</u>	<u>union</u>
<u>unsigned</u>	<u>void</u>	<u>volatile</u>	<u>while</u>



### C++ - >60 ключевых слов

<a href="#">alignas</a> (C++11) <a href="#">alignof</a> (C++11) <a href="#">and</a> <a href="#">and_eq</a> <a href="#">asm</a> <a href="#">auto</a> (1) <a href="#">bitand</a> <a href="#">bitor</a> <a href="#">bool</a> <a href="#">break</a> <a href="#">case</a> <a href="#">catch</a> <a href="#">char</a> <a href="#">char8_t</a> (C++20) <a href="#">char16_t</a> (C++11) <a href="#">char32_t</a> (since C++11) <a href="#">class</a> (1) <a href="#">compl</a>	<a href="#">concept</a> (C++20) <a href="#">const</a> <a href="#">constexpr</a> (C++20) <a href="#">constexpr</a> (C++11) <a href="#">constinit</a> (C++20) <a href="#">const_cast</a> <a href="#">continue</a> <a href="#">co_await</a> (C++20) <a href="#">co_return</a> (C++20) <a href="#">co_yield</a> (C++20) <a href="#">decltype</a> (C++11) <a href="#">default</a> (1) <a href="#">delete</a> (1) <a href="#">do</a> <a href="#">double</a>	<a href="#">dynamic_cast</a> <a href="#">else</a> <a href="#">enum</a> <a href="#">explicit</a> <a href="#">export</a> (1) (3) <a href="#">extern</a> (1) <a href="#">false</a> <a href="#">float</a> <a href="#">for</a> <a href="#">friend</a> <a href="#">goto</a> <a href="#">if</a> <a href="#">inline</a> (1) <a href="#">int</a> <a href="#">long</a>	<a href="#">mutable</a> (1) <a href="#">namespace</a> <a href="#">new</a> <a href="#">noexcept</a> (C++11) <a href="#">not</a> <a href="#">not_eq</a> <a href="#">nullptr</a> (C++11) <a href="#">operator</a> <a href="#">or</a> <a href="#">or_eq</a> <a href="#">private</a> <a href="#">protected</a> <a href="#">public</a>	<a href="#">register</a> (2) <a href="#">reinterpret_cast</a> <a href="#">requires</a> (C++20) <a href="#">return</a> <a href="#">short</a> <a href="#">signed</a> <a href="#">sizeof</a> (1) <a href="#">static</a> <a href="#">static_assert</a> (C++11) <a href="#">static_cast</a> <a href="#">struct</a> (1) <a href="#">switch</a> <a href="#">template</a> <a href="#">this</a> <a href="#">thread_local</a> (C++11) <a href="#">throw</a> <a href="#">true</a>	<a href="#">try</a> <a href="#">typedef</a> <a href="#">typeid</a> <a href="#">typename</a> <a href="#">union</a> <a href="#">unsigned</a> <a href="#">using</a> (1) <a href="#">virtual</a> <a href="#">void</a> <a href="#">volatile</a> <a href="#">wchar_t</a> <a href="#">while</a> <a href="#">xor</a> <a href="#">xor_eq</a>
--	---	--	--	--	--



**C++ отличается от C добавлением функционала, упрощающего написание программ в разных парадигмах («стилях» программирования)**

- Объектно-ориентированное программирование
- Обобщенное программирование
- Оперирование абстрактными контейнерами данных

# Введение

## C++ и C



В C++ «принят» отличный от C стиль взаимодействия с вводом/выводом программы

### C

```
#include <stdio.h>
int main(int argc, char* argv[]) {
    printf("Hello, World!\n");
    int foo;
    scanf("%d", &foo);
    printf("%d", foo + 2);
    return 0;
}
```

### C++

```
#include <iostream>
int main(int argc, char* argv[]) {
    std::cout << "Hello, World!\n";
    int foo;
    std::cin >> foo;
    std::cout << (foo + 2);
    return 0;
}
```

# Введение

## C++ и C



C++ использует иные ключевые слова для работы с динамической памятью

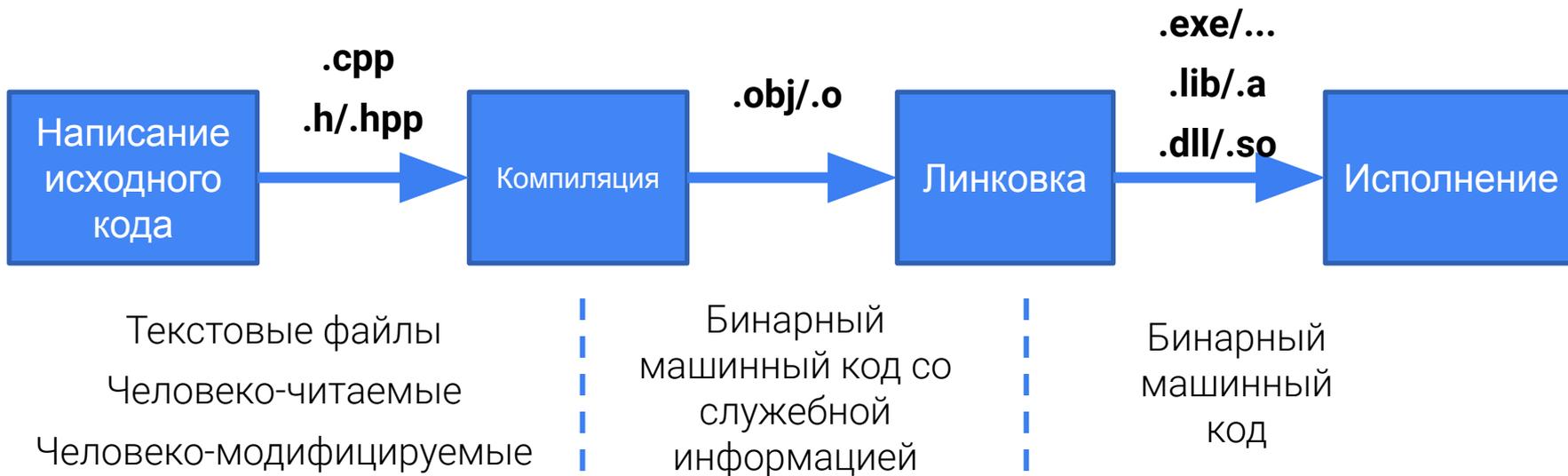
### C

```
#include <stdio>
int main(int argc, char* argv[]) {
    char* my_dyn_arr = NULL;
    my_dyn_arr = (char*) malloc(6 * sizeof(char));
    my_dyn_arr[0] = 'H'; my_dyn_arr[1] = 'e';
    my_dyn_arr[2] = 'l'; my_dyn_arr[3] = 'l';
    my_dyn_arr[4] = 'o'; my_dyn_arr[5] = 0;
    printf(my_dyn_arr);
    free(my_dyn_arr);
    return 0;
}
```

### C++

```
#include <iostream>
int main(int argc, char* argv[]) {
    char* my_dyn_arr = nullptr;
    my_dyn_arr = new char[6];
    my_dyn_arr[0] = 'H'; my_dyn_arr[1] = 'e';
    my_dyn_arr[2] = 'l'; my_dyn_arr[3] = 'l';
    my_dyn_arr[4] = 'o'; my_dyn_arr[5] = 0;
    std::cout << my_dyn_arr;
    delete[] my_dyn_arr;
    return 0;
}
```

# Базовый процесс программирования на C++





## Windows - cl.exe/link.exe

Microsoft Visual C++ Compiler (MSVC) - обычно входит в состав Visual Studio

КОМПИЛЯЦИЯ



```
C:\ cl.exe /c hello_world.cpp
C:\ dir
2021-09-01 15:43          332 hello_world.cpp
2021-09-01 15:45      52,471 hello_world.obj
```

ЛИНКОВКА



```
C:\ link.exe /c hello_world.cpp
C:\ dir
2021-09-01 15:43          332 hello_world.cpp
2021-09-01 15:45     185,856 hello_world.exe
2021-09-01 15:45      52,471 hello_world.obj
```



## Linux - g++

GNU Compiler Collection - доступно в качестве пакета во всех версиях Linux, зачастую предустановлена

КОМПИЛЯЦИЯ



```
$ g++ -c hello_world.cpp
$ ls
hello_world.cpp hello_world.o
```

ЛИНКОВКА



```
$ g++ hello_world.o -o hello_world
$ ls
hello_world.cpp hello_world.o hello_world
```



## Кросс-платформенный - clang

Фронт-энд (т.е. приложение, взаимодействующее с пользователем) для компиляции C/C++ с помощью фреймворка компиляторов LLVM; устанавливается дополнительно

КОМПИЛЯЦИЯ



```
$ clang -c hello_world.cpp
$ ls
hello_world.cpp hello_world.o
```

ЛИНКОВКА



```
$ clang hello_world.o -o hello_world
$ ls
hello_world.cpp hello_world.o hello_world
```



## Windows:

- MS Visual Studio
- Visual Studio Code
- Qt Creator
- Code::Blocks
- Notepad++

## Linux:

- Eclipse
- CLion
- Qt Creator
- Code::Blocks
- Vim



## Программа скомпилировалась, но работает неправильно - что делать?

- C++ поддерживает строчную отладку, точки останова и т. д.
- Отладчик (debugger) - дополнительная программа, с помощью которого производится отладка
- Отладчик естественным образом дополняет компилятор и линковщик; часто всё вместе называют "tool chain"
- GDB - отладчик из GCC, LLDB - отладчик clang/LLVM, Microsoft Visual Studio Debugger - отладчик MSBuild/Visual Studio

## Компиляторы для C++ - оптимизирующие

- Компиляторы обязаны преобразовать исходный код в машинный корректным образом, но не обязаны, например, сохранять порядок машинных операций относительно порядка соответствующих строк в коде, или сохранять неиспользуемые куски кода



- Для того, чтобы отладить программу, необходимо скомпилировать ее специальным образом - обычно это указывается с помощью флага для компилятора

```
$ g++ -g hello_world.cpp -o hello_world_for_debug
```

- Такими же флагами контролируются и применяемые оптимизации:

```
$ g++ hello_world.cpp -O3 -o hello_world_for_release
```

```
$ g++ hello_world.cpp -O1 -o hello_world_with_less_opts
```

# std::cin, std::cout



## std::cout

- Объект класса ostream, предназначенный для взаимодействия со стандартным потоком вывода
- В коде, в основном, используется вместе с оператором << (aka “inserter”)
- Работает с большинством встроенных типов и с некоторыми STL-типами; можно приспособить и для вывода пользовательских классов
- Формат вывода настраивается с помощью использования объектов из заголовка `<iomanip>`

## std::cin

- Объект класса istream, предназначенный для взаимодействия со стандартным потоком ввода
- В коде, в основном, используется вместе с оператором >> (aka “extractor”)
- - // -