Лекция 1. Введение в C++

Программирование на C++

Мехмат ЮФУ

Осенний семестр 2026

Содержание лекции

1. Что такое C++? Рейтинг TIOBE. Философия, вехи развития. Сравнение с Python.
2. Компиляция и исполнение. Препроцессор. Выбор стандарта языка.
3. Простейшая программа, пространства имён.
4. Ввод и вывод. print и println.
5. Типы данных, sizeof, предельные значения. Литералы.
6. Объявление и инициализация переменных, auto.
7. Явное и неявное преобразование типов. Переполнение и сравнение вещественных чисел.
8. Константы (const, constexpr).
9. Управляющие конструкции: if, switch, тернарная условная операция, операция запятая.
10. Основные операции. Приоритет операций.
11. Циклы: for, while, do-while.
12. Математические функции.
13. Утверждения (assert).
14. Структуры. Передача в функцию.
15. Генерация случайных чисел.
16. Объявление функций (прототипы).

Что такое C++?

- Компилируемый, статически типизированный язык общего назначения.
- Проект начался у Бьярна Страуструпа в 1979 году как *C with Classes*; название C++ появилось в 1983 году.
- Мультипарадигмальный: процедурный, ООП, обобщённый, функциональный.
- Применяется в высокопроизводительных системах: игры, ОС, компиляторы, встраиваемые системы, финансы, HPC.
- Один из самых популярных языков (TIOBE Index, 2026).

Aug 2026	Aug 2025	Change	Programming Language	Ratings	Change
1	1		Python	18.53%	-7.61%
2	3	↑	C	11.10%	+2.07%
3	2	↓	C++	8.62%	-0.56%
4	4		Java	8.25%	-0.34%
5	5		C#	4.09%	-1.43%
6	6		JavaScript	2.63%	-0.52%
7	7		Visual Basic	2.18%	-0.15%
8	12	⇈	SQL	1.88%	+0.16%
9	14	⇈	R	1.56%	+0.19%
10	18	⇈	Rust	1.45%	+0.31%

Философия C++

Принцип zero-overhead

«Не платите за то, что не используете».

- Эффективность — минимальные накладные расходы.
- Переносимость — код работает на разных платформах.
- Совместимость с C — многие C-идиомы и библиотеки доступны, но C++ не является строгим надмножеством C.
- Мультипарадигмальность — выбор стиля под задачу.
- «C++ даёт вам контроль, но требует ответственности» (Страуструп).

Вехи развития C++

- 1979 — *C with Classes*: классы и идеи ООП поверх C.
- 1983 — язык получил название C++.
- 1998 — C++98 (первый стандарт).
- 2011 — C++11 (move, auto, лямбды, умные указатели).
- 2014 — C++14 (доработки).
- 2017 — C++17 (структурное связывание, if constexpr).
- 2020 — C++20 (концепты, ranges, модули).
- 2023 — C++23 (std::print и др.).

В курсе стараемся использовать C++23 (по возможности).

Современные стандарты C++

Стандарт	Ключевые нововведения
C++11	move-семантика, `auto`, лямбды, умные указатели, `nullptr`, `constexpr`
C++14	бинарные литералы, разделители разрядов, обобщённые лямбды, `std::make_unique`
C++17	структурное связывание, `if constexpr`, `std::optional`, `std::string_view`, `std::filesystem`
C++20	концепты, ranges, модули, `std::format`, трёхстороннее сравнение (<=>)
C++23	`std::print`, `std::println`, `std::expected`, многомерный `operator[]`

C++ vs Python

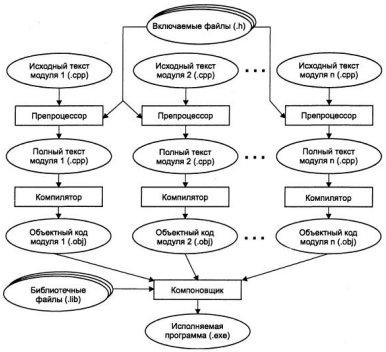
Аспект	Python	C++
Типизация	динамическая, утиная	статическая, строгая
Компиляция	интерпретируемый	компилируемый
Управление памятью	сборщик мусора	RAII, контейнеры, умные указатели; вручную — при необходимости
Скорость	медленнее (10–100 раз)	быстрее
Синтаксис	лаконичный, отступы	многословный, {} и ;
Область применения	скрипты, ML, веб	системное, игры, HPC, встраиваемые

Компиляция и исполнение

- Python: исходник → байт-код → интерпретатор.
- C++: исходник → **препроцессор** → **компилятор** → **линковщик** → .exe.

Этапы:

1. Препроцессинг: #include, #define, удаление комментариев.
2. Компиляция: перевод в объектный код (.obj/.o).
3. Линковка: сборка объектных файлов и библиотек.



Пример работы препроцессора

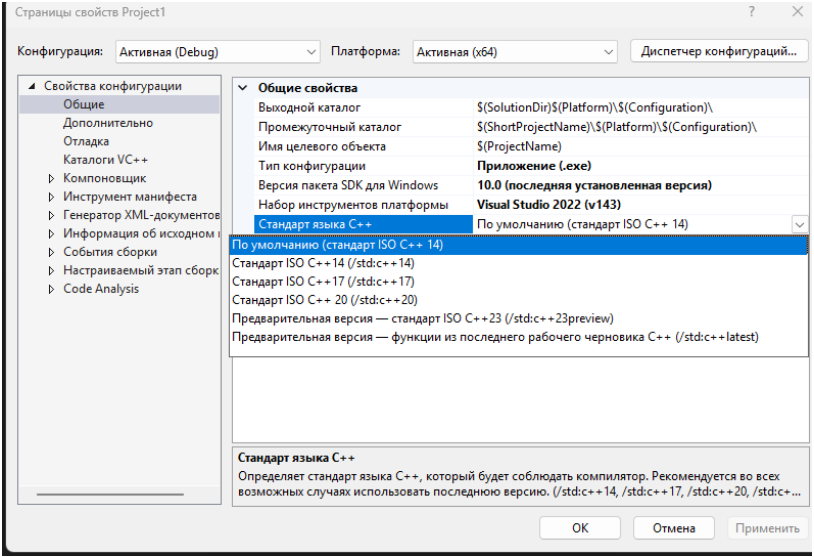
Исходник

```
#define PI 3.14
#include <iostream>
int main() {
    std::cout << "PI = " << PI;
}
```

После препроцессинга

```
// содержимое iostream
int main() {
    std::cout << "PI = " << 3.14;
}
```

Стандарт C++ в опциях проекта



Структура простейшей программы

Код

```cpp
#include <iostream>
int main() {
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

- #include — директива препроцессора.
- main — точка входа.
- std::cout — вывод.
- std::endl — перевод строки + сброс буфера (используйте \n для скорости).
- return 0 — успешное завершение.

Вопрос: что выведет этот код?

Код

```cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello";
    cout << "World";
    return 0;
}
```

Вопрос: что выведет этот код?

Код

```cpp
#include <iostream>
using namespace std;
int main() {
    cout << "Hello";
    cout << "World";
    return 0;
}
```

Ответ: HelloWorld (без пробела)

Вопрос: что выведет этот код?

Код

```
#include <iostream>
int main()
{
    std::cout << "Привет, мир!\n";
    return 0;
}
```

Вопрос: что выведет этот код?

Код

```
#include <iostream>
int main()
{
    std::cout << "Привет, мир!\n";
    return 0;
}
```

Ответ: абракадабра

Необходимо подключение русской локали:

```
setlocale(LC_ALL, "Russian");
```

std::print (C++23)

Код

```cpp
#include <print>
int main() {
    std::print("Hello, World!\n");
    return 0;
}
```

- Удобный форматированный вывод.
- Поддерживается в GCC 14+, Clang 17+.
- В курсе используем cout, но знайте о существовании.

Примеры

```cpp
#include <print>
using namespace std;

int main()
{
    int num = 255;
    double pi = 3.1415926535;
    string name = "Alice";

    println("{1} {0}", num, pi);  // 3.1415926535 255
    println("Hex: {:x}", num);    // ff
    println("Bin: {:b}", num);    // 11111111
    println("|{:10}|", num);      // |       255|
    println("{:.2f}", pi);        // 3.14
    println("|{:10.3f}|", pi);    // |     3.142|
    println("|{:<10.3f}|", pi);   // |3.142     |
    println("|{:^10.3f}|", pi);   // |  3.142   |
    println("|{:<10}|", name);    // |Alice     |
    println("|{:>10}|", name);    // |     Alice|

    return 0;
}
```

Пространства имён (namespace)

- Все стандартные сущности в `std`.

```
std::cout << "Hello";
```

- Подключение всех имён: `using namespace std;`
- Подключение одного имени: `using std::cout;`

Рекомендация

В заголовочных файлах не использовать `using namespace std`.

Вывод в консоль

```cpp
int age = 20;
std::cout << "Age: " << age << "\n";
std::cout << "Next year: " << age + 1 << std::endl;
```

- \n — быстрее, чем endl.
- Можно выводить несколько значений подряд.

Ввод с консоли

```
int x;
std::cin >> x;
```

- `cin` пропускает пробелы и переводы строк.
- Для чтения строки с пробелами: `std::getline(std::cin, str);`

Базовые типы данных

Тип	Размер (обычно)	Диапазон
int	4 байта	$\pm 2 \cdot 10^9$
double	8 байт	$\pm 1.7 \cdot 10^{308}$
char	1 байт	-128..127 или 0..255 (unsigned)
bool	1 байт	true/false

- Размеры могут зависеть от платформы.
- Для точных размеров: `<cstdint>` (int32_t и др.).
- Предельные значения: `std::numeric_limits<T>::max()`

sizeof

```cpp
#include <iostream>
int main() {
    std::cout << "int: " << sizeof(int) << " bytes\n";
    std::cout << "double: " << sizeof(double) << " bytes\n";
}
```

Вывод (обычно): int: 4 bytes, double: 8 bytes

Предельные значения

```cpp
#include <limits>
std::cout << "int max: " << std::numeric_limits<int>::max() << "\n"; // 2147483647
std::cout << "int max (2): " << INT_MAX << "\n"; // 2147483647
std::cout << "double min: " << std::numeric_limits<double>::min() << "\n"; // 2.22507e-308
std::cout << "double min (2): " << DBL_MIN << "\n"; // 2.22507e-308
std::cout << "double epsilon: " << std::numeric_limits<double>::epsilon() << "\n"; // 2.22045e-16
```

Литералы

```
int a = 42;        // десятичный
int b = 0x2A;      // шестнадцатеричный (0x)
int c = 052;       // восьмеричный (0) - не рекомендуется
int d = 0b101010; // двоичный (C++14)

double pi = 3.14159; // вещественный
float f = 3.14f;     // суффикс f
long double ld = 3.14L;

char ch = 'A';       // символьный
const char* s = "Hello"; // строковый
```

Бинарные литералы и разделители разрядов (C++14)

Бинарные литералы

Запись чисел в двоичной системе с помощью префикса 0b:

```
int a = 0b1010;   // 10
int b = 0b11111111; // 255
int c = 0b10101010; // 170
```

Разделители разрядов

Апостроф для улучшения читаемости больших чисел:

```
int million = 1'000'000;
int binary = 0b1010'1010'1010;
int hex = 0xDEAD'BEEF;
long long big = 9'223'372'036'854'775'807LL;
```

- Разделители не влияют на значение, только на читаемость.
- Можно использовать в любых числовых литералах.

Объявление переменных

```
int a;          // объявление без инициализации
double pi = 3.14;
char ch = 'A';
bool flag = true;
```

- Явное указание типа обязательно (кроме `auto`).
- Неинициализированные переменные содержат мусор (UB).

auto — вывод типа

Для переменных (C++11)

```
auto x = 5;        // int
auto y = 3.14;     // double
auto z = "Hello"; // const char*
```

Для возврата функций (C++14)

```
auto square(int x) {
    return x * x;  // тип выводится как int
}
auto add(double a, double b) {
    return a + b;  // тип выводится как double
}
```

- Удобно для сложных типов (итераторы, лямбды).
- Для функций — только если тело состоит из одного выражения `return`.
- Не злоупотребляйте в простых случаях — читаемость важнее.

Инициализация переменных

Способы:

1. Копированием: `int a = 5;`
2. Прямая: `int a(5);`
3. Универсальная: `int a{5};` — **рекомендуется**.

Важно

Универсальная инициализация запрещает сужающие преобразования.

```
int x { 3.14 }; // ошибка
char c { 1000 }; // ошибка
bool b { -1 }; // ошибка

int j { true }; // OK: j = 1 (bool → int)
int y = 3.14; // OK: y = 3 (потеря данных)
```

Явное и неявное преобразование типов

Неявное

```
double d = 3.14;
int i = d; // i = 3 (потеря данных)
```

Явное (C-style)

```
int i = (int)3.14;
```

Явное (C++)

```
int i = static_cast<int>(3.14);
```

- C++-cast предпочтительнее — лучше читается и ищется.

Переполнение и сравнение вещественных

Переполнение целых

```
int max = std::numeric_limits<int>::max();
int overflow = max + 1; // UB (переполнение)
```

Сравнение вещественных

```
double a = 0.1 + 0.2;
double b = 0.3;
if (a == b) // false!
if (std::abs(a - b) < 1e-9) // правильно
```

Вопрос: что выведет?

```
double pi = 3.14;
int a = pi;
int b{pi};
cout << a << " " << b;
```

Вопрос: что выведет?

```
double pi = 3.14;
int a = pi;
int b{pi};
cout << a << " " << b;
```

Ответ: Ошибка компиляции на `int b{pi};`

const и constexpr

```
const double pi = 3.14159;   // может вычисляться во время выполнения
constexpr int size = 10;      // обязательно на этапе компиляции
```

- `constexpr` — для настоящих констант.
- `const` — для защиты от изменений.
- `constexpr` может быть применён к функциям.

if-else

```
if (x > 0) {
    cout << "Positive";
} else if (x < 0) {
    cout << "Negative";
} else {
    cout << "Zero";
}
```

- Фигурные скобки рекомендуются всегда.
- В C++ `else if` — это просто вложенный if.

Вопрос: что выведет?

```
if (a > b)
    cout << "a > b";
    cout << "a <= b";
```

Вопрос: что выведет?

```
if (a > b)
    cout << "a > b";
    cout << "a <= b";
```

Ответ: всегда a <= b (второй cout не входит в if)

Тернарная условная операция

```
int max = (a > b) ? a : b;
```

Сокращённая форма if-else для выражений.

```
cout << ((a > b) ? "a > b" : "a <= b");
```

Операция запятая

```
int s = 0, i = 5, j = 6;
cout << (s = i * i, i); // выведет 5
cout << (s = j * j, s); // выведет 36
```

Возвращает значение последнего выражения. Иногда используется в for.

switch

```
switch (day) {
    case 1: cout << "Mon"; break;
    case 2: cout << "Tue"; break;
    default: cout << "Unknown";
}
```

```
int x{2};
switch (x) {
    case 1:
    case 2:
        std::cout << "x = 1 or 2" << "\n"; break;
     case 5:
         std::cout << "x = 5" << "\n"; break;
    default: cout << "x is not 1, 2 or 5" << "\n";
}
```

- `break` обязателен, иначе проваливание.
- `default` — для всех остальных значений.

Вопрос: что выведет?

```
switch (x) {
    case 1: cout << "One";
    case 2: cout << "Two";
    case 3: cout << "Three";
}
```

Вопрос: что выведет?

```
switch (x) {
    case 1: cout << "One";
    case 2: cout << "Two";
    case 3: cout << "Three";
}
```

Ответ: TwoThree (проваливание)

Основные операции

```
i << 3    // побитовый сдвиг влево
i >> 2    // побитовый сдвиг вправо
a += 2;   // a = a + 2;
b = a++;  // t = a; a++; b = t;
b = ++a;  // a++; b = a;
a = b++ + --c;
7 / 3     // целочисленное деление
7 % 3     // остаток от целочисленного деления
(i < 0 || i > 2)   // или
(i >= 2 && i <= 3) // и
!(i>2)             // не
& | ^ ~ // побитовые and, or, xor, not
min = a < b ? a : b; // тернарная условная операция
a = b = c; // множественное присваивание b = c, a = b
```

Приоритет операций

Операторы	Приоритет
`::`	1
`()`, `[]`, `->`, `++`, `--` (постфикс)	2
`++`, `--` (префикс), `!`, `&`, `-` (унарный), `*`	3
`*`, `/`, `%`	5
`+`, `-`	6
`>>`, `<<`	7
`<`, `>`, `<=`, `>=`	8
`==`, `!=`	9
`&`	10
`^`	11
`|`	12
`&&`	13
`||`	14
`=`, `+=`, `-=` и др.	15

Цикл for

```
for (int i = 0; i < 10; ++i) {
    cout << i << " ";
}
```

- Переменная `i` видна только внутри цикла.
- Бесконечный цикл: `for(;;) { ... }`
- Три части: инициализация, условие, шаг.

for с несколькими переменными

```
for (int i = 0, j = 10; i < j; ++i, --j) {
    cout << i << " " << j << "\n";
}
```

Вывод:

```
0 10
1 9
2 8
3 7
4 6
```

Вопрос: что выведет?

```
for (int i = 0; i < 3; ++i) cout << i;
cout << i;
```

Вопрос: что выведет?

```
for (int i = 0; i < 3; ++i) cout << i;
cout << i;
```

Ответ: Ошибка компиляции (i не видна за пределами цикла)

while и do-while

```
while (i < 10) { ... ++i; }
do { ... ++i; } while (i < 10);
```

- `do-while` гарантирует хотя бы одно выполнение.

Бесконечный цикл и выход

```
while (true) {
    int x;
    cin >> x;
    if (x == 0) break; // выход
    if (x < 0) continue; // пропуск
    cout << x;
}
```

break и continue

```
for (int i=0; i<10; ++i) {
    if (i == 5) break;    // выход из цикла
    if (i % 2 == 0) continue; // пропуск итерации
    cout << i;
}
```

Вывод: 1 3 (нечётные до 5)

Вложенные циклы (пример)

```
for (int i=1; i<=3; ++i) {
    for (int j=1; j<=3; ++j)
        cout << i*j << " ";
    cout << "\n";
}
```

Вывод:

```
1 2 3
2 4 6
3 6 9
```

Вопрос: что выведет?

```
int i = 0;
while (i < 3) {
    cout << i << " ";
    i++;
}
cout << i;
```

Вопрос: что выведет?

```
int i = 0;
while (i < 3) {
    cout << i << " ";
    i++;
}
cout << i;
```

Ответ: 0 1 2 3

Математические функции (<cmath>)

```
#include <cmath>
double x = 16.0;
std::sqrt(x);   // 4.0
std::pow(x,0.5); // 4.0
std::fabs(-3.5); // 3.5
std::sin(3.14159); // 0
std::log(2.71828); // 1
std::exp(1.0);   // 2.718
// cos, tan, asin acos, atan
// sinh, cosh, tanh, asinh, acosh, atanh
// exp2, log10, log2, cbrt, hypot
// ceil, floor, trunc, round
```

Все функции в пространстве std.

assert

```cpp
#include <cassert>
double area(double r) {
    assert(r >= 0);
    return 3.14 * r * r;
}
```

- Если условие ложно — аварийное завершение.
- В релизе можно отключить: `#define NDEBUG`
- `assert` — для отладки, не для обработки ошибок в релизе.

Вопрос: что выведет?

```
assert(2 + 2 == 5);
```

Вопрос: что выведет?

```
assert(2 + 2 == 5);
```

Ответ: Аварийное завершение.

Структуры

```
struct Point {
    double x;
    double y;
};
Point p1 = {1.0, 2.0};
cout << p1.x << " " << p1.y;
```

- Поля по умолчанию `public`.
- Можно передавать в функции по значению или по ссылке.
- Подробнее — на следующей лекции.

Передача структуры в функцию

```
void printPoint(Point p) {
    cout << "(" << p.x << ", " << p.y << ")";
}
// лучше по ссылке:
void printPoint(const Point& p) { ... }
```

- По значению — копия (может быть дорого).
- По ссылке — эффективно, без копирования.
- `const` — защита от изменений.

Генерация случайных чисел (<random>)

```cpp
#include <random>
std::random_device rd;
std::mt19937 gen(rd());
std::uniform_int_distribution<int> dist(1, 6);
int dice = dist(gen); // 1..6
```

- `random_device` — аппаратный источник энтропии.
- `mt19937` — явно выбранный псевдослучайный генератор.
- `uniform_int_distribution` — равномерное распределение.
- Подробно изучим на следующей лекции.

Объявление функций (прототипы)

```cpp
#include <iostream>
using namespace std;

double area(double r); // прототип (объявление)

int main() {
    cout << area(5.0); // OK
    return 0;
}

double area(double r) { // определение
    return 3.14 * r * r;
}
```

- Прототип нужен, если функция вызывается до определения.
- Обычно прототипы помещают в заголовочные файлы.
- Подробнее — на следующей лекции.

Пример: площадь круга

```
#include <iostream>
#include <cmath>
#include <cassert>
int main() {
    double radius;
    cout << "Enter radius: ";
    cin >> radius;
    assert(radius >= 0);
    const double PI = 3.1415926535;
    double area = PI * radius * radius;
    cout << "Area: " << area << endl;
    return 0;
}
```

Разбор примера

- Ввод радиуса.
- `assert` проверяет неотрицательность.
- Константа `PI`.
- Вычисление и вывод.
- Обратите внимание: `assert` сработает только в Debug.

Типичные ошибки начинающих

1. Забыли #include <iostream>
2. Забыли return 0;
3. Забыли break в switch
4. Путают = и ==
5. Забыли фигурные скобки в if/for
6. Использование неинициализированной переменной
7. Сужающее преобразование (особенно с {})

Вопрос: что выведет?

```
if (x = 3) cout << "True";
else cout << "False";
```

Вопрос: что выведет?

```
if (x = 3) cout << "True";
else cout << "False";
```

Ответ: True (присваивание, не сравнение)

Стиль кода и Core Guidelines

- Имена переменных: осмысленные, с маленькой буквы.
- Константы: обычно с большой буквы или с префиксом k.
- Фигурные скобки всегда ставить (даже для одной строки).
- Использовать `auto` для сложных типов.
- Избегать сужающих преобразований.
- Инициализировать переменные при объявлении.

Вложенный тернарный оператор

```
int x = 10, y = 20, z = 15;
int max = (x > y) ? ((x > z) ? x : z) : ((y > z) ? y : z);
```

- Работает, но быстро становится нечитаемым.
- Для сложных условий лучше использовать if-else.

static_cast и reinterpret_cast

static_cast — безопасное приведение

```
double d = 3.14;
int i = static_cast<int>(d); // OK
```

reinterpret_cast — опасное приведение

```
int* p = new int(42);
long long addr = reinterpret_cast<long long>(p); // опасно
```

Предупреждение

reinterpret_cast почти всегда указывает на проблему дизайна. Используйте только при крайней необходимости.

Итоги лекции

- C++ — мощный компилируемый язык.
- Основные типы, ввод/вывод.
- Управляющие конструкции: if, switch, циклы.
- Математические функции, assert.
- Универсальная инициализация {}.
- Структуры и ссылки.
- Прототипы функций — мостик к следующей лекции.

Что дальше?

- Следующая лекция: функции, ссылки, многофайловая компоновка.
- На практике: создание проектов из нескольких файлов.
- ДЗ: Установите среду разработки Visual Studio.

Вопросы?