

Основы Программирования

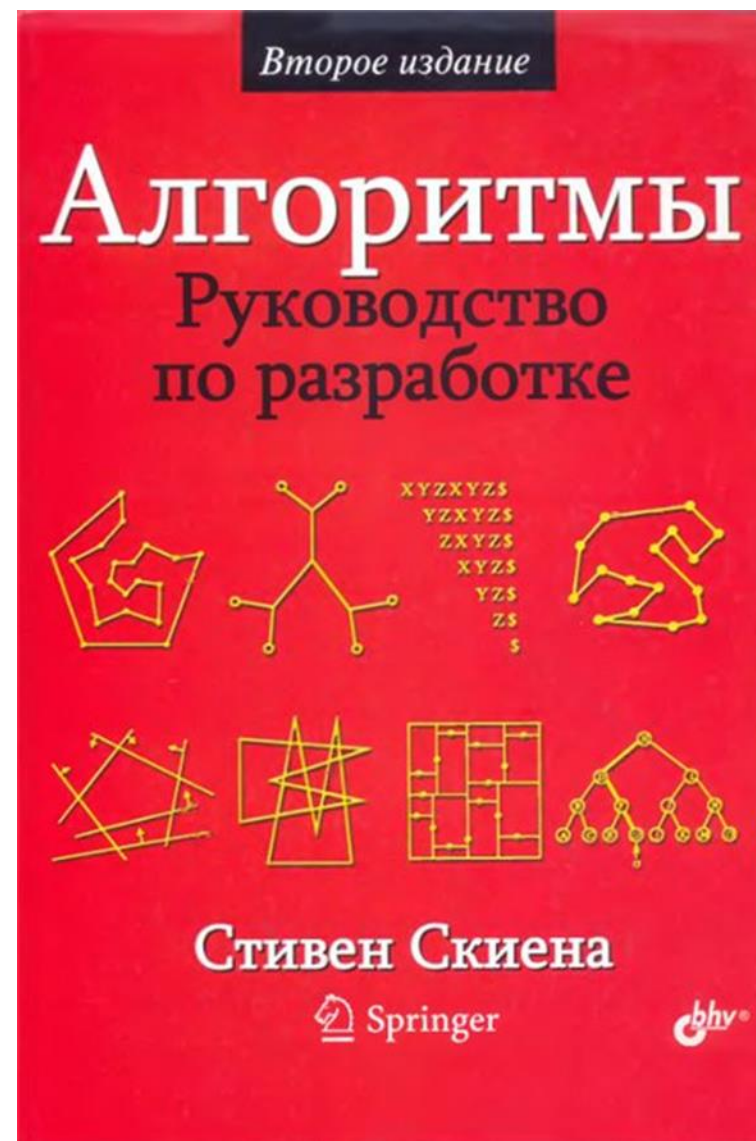
Литература

- Томас Кормен “Алгоритмы построение и анализ”



Литература

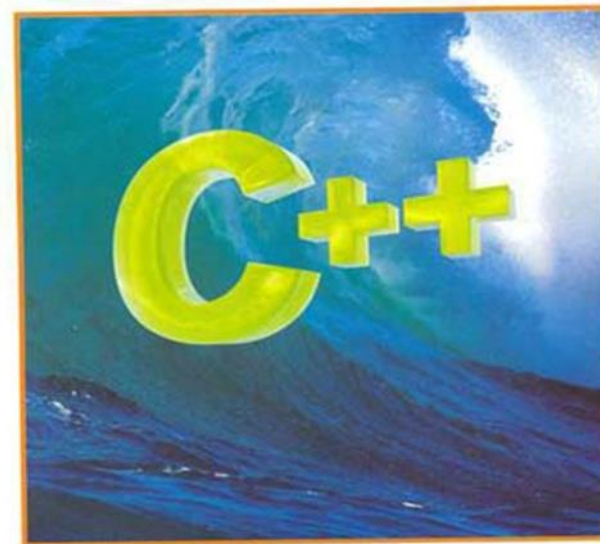
- Стивен Скиена “Алгоритмы. Руководство по разработке”



Литература

- Бьерн Страуструп “Язык программирования C++”

ЯЗЫК C++ ПРОГРАММИРОВАНИЯ специальное издание



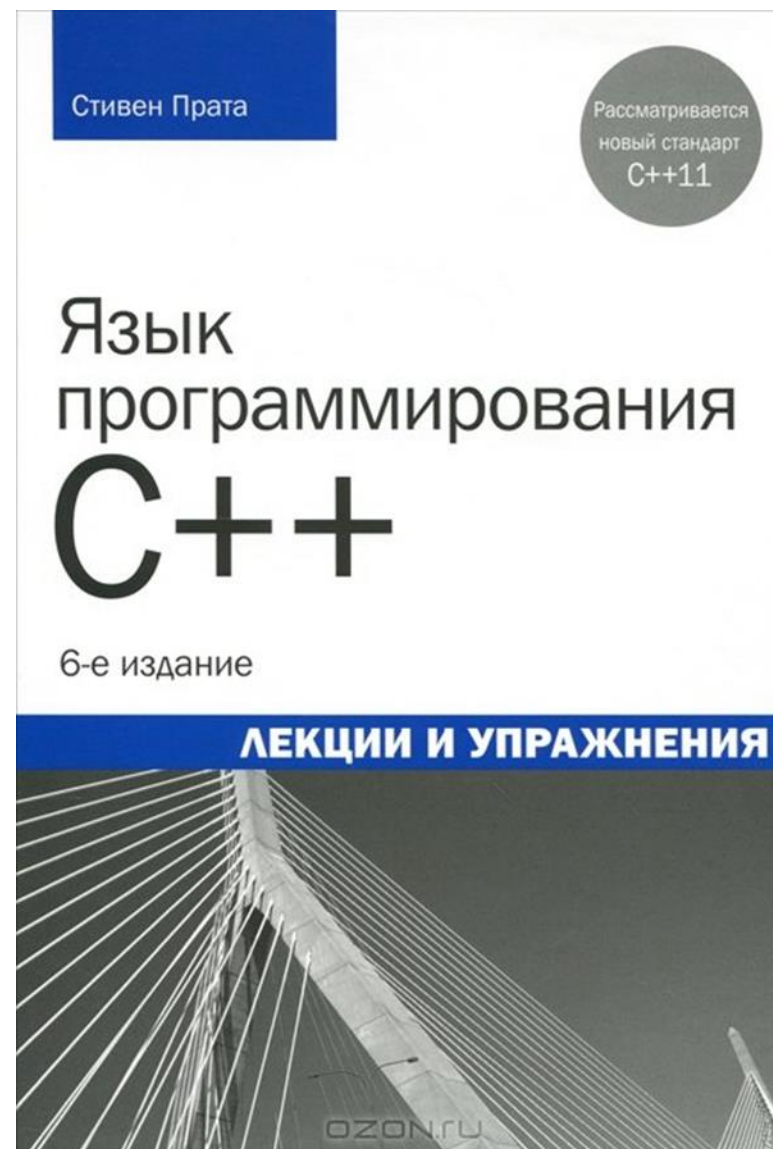
Бьерн Страуструп
создатель C++



с авторскими изменениями
и дополнениями

Литература

- Стивен Прата “Язык программирования C++”

























Литература

- В.В. Воеводин, Вл. В. Воеводин
“Параллельные вычисления”



Рейтинг языков (spectrum.ieee.org), July 2017

Language Rank	Types	Spectrum Ranking
1. Python	 	100.0
2. C	  	99.7
3. Java	  	99.4
4. C++	  	97.2
5. C#	  	88.6
6. R		88.1
7. JavaScript	 	85.5
8. PHP		81.4
9. Go	 	76.1
10. Swift	 	75.3

Язык С

- Язык программирования С разработан в начале 1972 года в компании Bell Labs Кеном Томпсоном и Деннисом Ритчи.
- Язык С был создан для использования в операционной системе UNIX.
- В связи с успехом UNIX язык С получил широкое распространение.
- На данный момент С является одним из самых распространённых языков программирования (доступен на большинстве платформ).
- С — основной язык для низкоуровневой разработки.

Особенности C

- Эффективность.

Язык C позволяет писать программы, которые напрямую работают с железом.

- Стандартизированность.

Спецификация языка C является международным стандартом.

- Относительная простота.

Стандарт языка C занимает 230 страниц.

Создание C++

- Разрабатывается с начала 1980-х годов.
- Создатель — сотрудник Bell Labs Бьёрн Страуструп.
- Изначально это было расширение языка C для поддержки работы с классами и объектами.
- Это позволило проектировать программы на более высоком уровне абстракции.
- Ранние версии языка назывались “C with classes”.

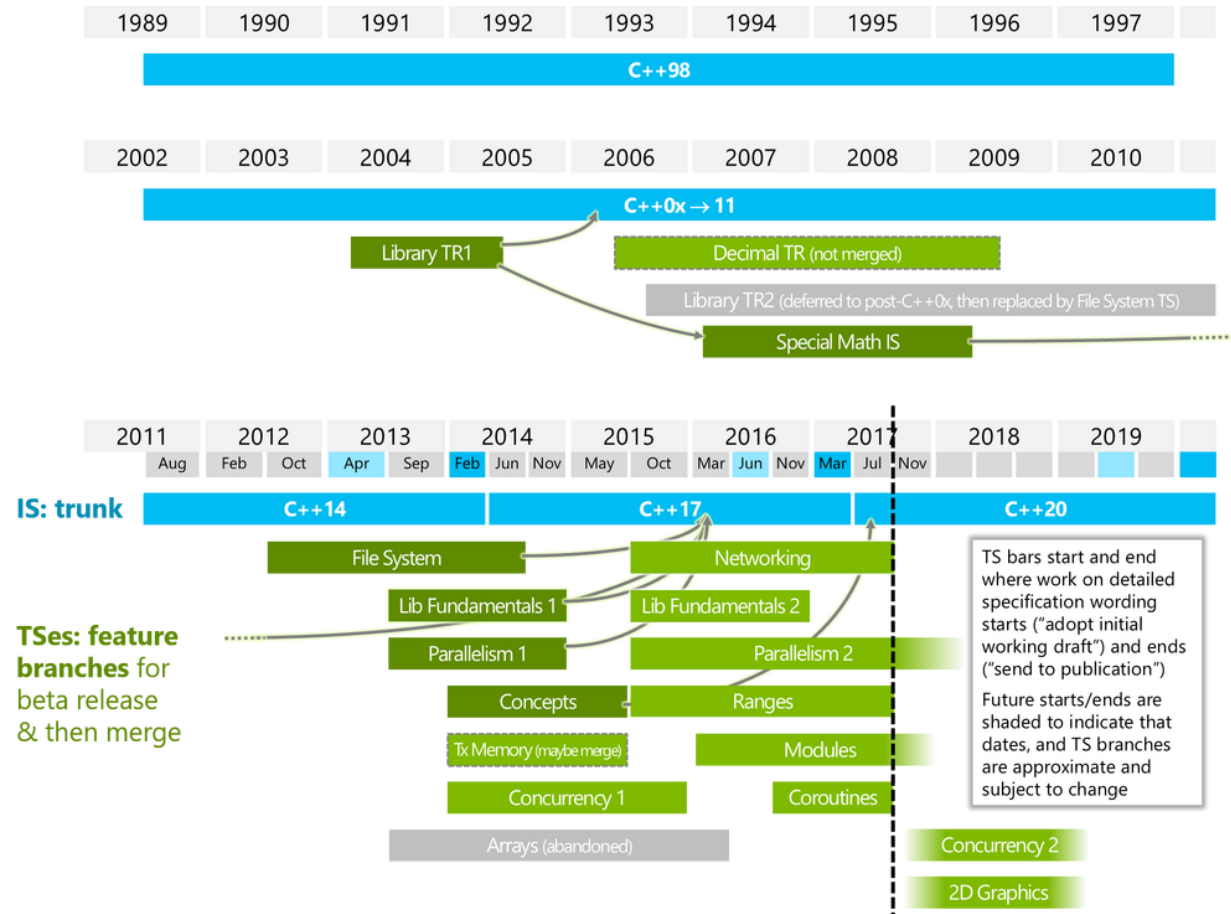
Развитие C++

- К 1983 году в язык было добавлено много новых возможностей (виртуальные функции, перегрузка функций и операторов, ссылки, константы, . . .)
- Получившийся язык перестал быть просто дополненной версией классического C и был переименован из “C с классами” в C++.
- Имя языка, получившееся в итоге, происходит от оператора унарного постфиксного инкремента C '++'.
- Язык также не был назван D, поскольку “является расширением C и не пытается устранять проблемы путём удаления элементов C”.

Стандартизация C++

- Лишь в 1998 году был ратифицирован международный стандарт языка C++: ISO/IEC 14882:1998 “Standard for the C++ Programming Language”.
- В 2003 году был опубликован стандарт языка ISO/IEC 14882:2003, где были исправлены выявленные ошибки и недочёты предыдущей версии стандарта.
- В 2005 году был выпущен Library Technical Report 1 (TR1) и началась работа над новой версией стандарта, которая получила кодовое название C++0x.
- В 2011 году стандарт был принят и получил название C++11 ISO/IEC 14882:2011. И началась работа нас C++1y.
- В 2014 году принят стандарт C++14. В 2017 принят стандарт C++17.

Эволюция C++



Совместимость С и С++

- Один из принципов разработки стандарта С++ — это сохранение совместимости с С.
- Синтаксис С++ унаследован от языка С.
- С++ не является в строгом смысле надмножеством С.
- Можно писать программы на С так, чтобы они успешно компилировались на С++.
- С и С++ сильно отличаются как по сложности, так и по принятым архитектурным решениям, которые используются в обоих языках.

Характеристики языка C++

- сложный,
- мультипарадигмальный,
- эффективный,
- низкоуровневый,
- компилируемый,
- статически типизированный.

Сложность

- Описание стандарта занимает более 1300 страниц текста.
- Нет никакой возможности рассказать “весь C++” в рамках одного, пусть даже очень большого курса.
- В C++ программисту позволено очень многое, и это влечёт за собой большую ответственность.
- На плечи программиста ложится много дополнительной работы:
 - проверка корректности данных,
 - управление памятью,
 - обработка низкоуровневых ошибок.

Мультипарадигмальный

На C++ можно писать программы в рамках нескольких парадигм программирования:

- процедурное программирование (код “в стиле C”),
- объектно-ориентированное программирование (классы, наследование, виртуальные функции, . . .),
- обобщённое программирование (шаблоны функций и классов),
- функциональное программирование (функторы, безымянные функции, замыкания).

Эффективный

Одна из фундаментальных идей языков С и С++ — отсутствие неявных накладных расходов, которые присутствуют в других более высокоуровневых языках программирования.

- Программист сам выбирает уровень абстракции, на котором писать каждую отдельную часть программы.
- Можно реализовывать критические по производительности участки программы максимально эффективно.
- Эффективность делает С++ основным языком для разработки приложений с компьютерной графикой (к примеру, игры).

Низкоуровневый

- Язык C++, как и C, позволяет работать напрямую с ресурсами компьютера.
 - Позволяет писать низкоуровневые системные приложения (например, драйверы операционной системы).
 - Неаккуратное обращение с системными ресурсами может привести к падению программы.
- В C++ отсутствует автоматическое управление памятью.
 - Позволяет программисту получить полный контроль над программой.
 - Необходимость заботиться об освобождении памяти.

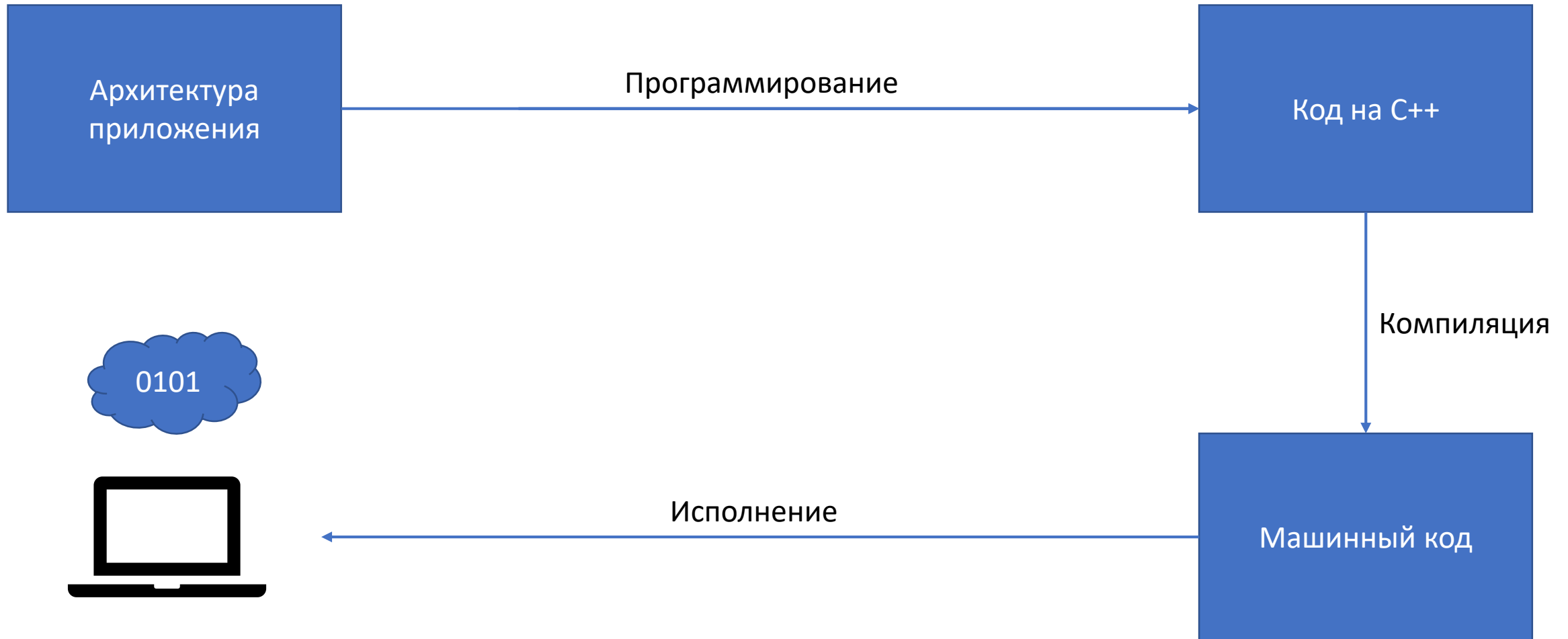
Компилируемый

- С++ является компилируемым языком программирования.
- Для того, чтобы запустить программу на С++, её нужно сначала скомпилировать.
- Компиляция — преобразование текста программы на языке программирования в машинный код.
 - Нет накладных расходов при исполнении программы.
 - При компиляции можно отловить некоторые ошибки.
 - Требуется компилировать для каждой платформы отдельно.

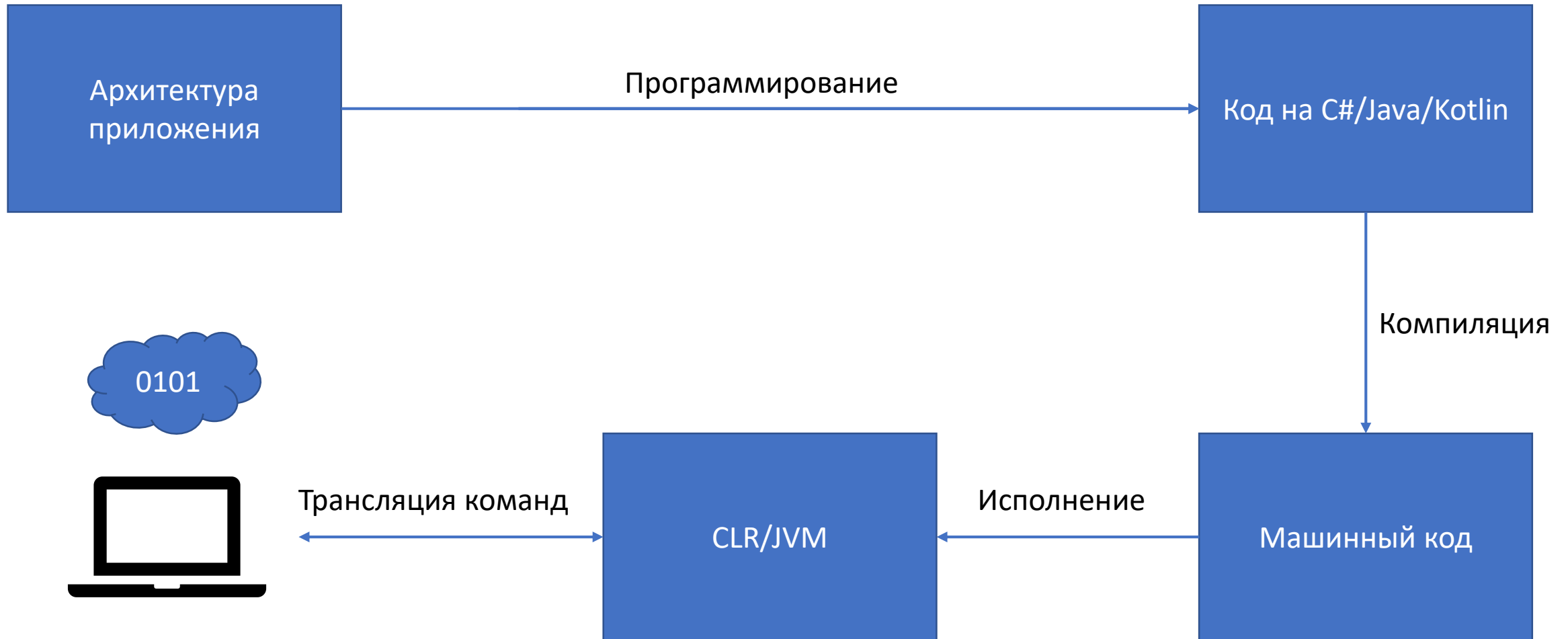
Статическая типизация

- C++ является статически типизированным языком.
 1. Каждая сущность в программе (переменная, функция и пр.) имеет свой тип,
 2. и этот тип определяется на момент компиляции.
- Это нужно для того, чтобы
 1. вычислить размер памяти, который будет занимать каждая переменная в программе,
 2. определить, какая функция будет вызываться в каждом конкретном месте.
- Всё это определяется на момент компиляции и “зашивается” в скомпилированную программу.
- В машинном коде никаких типов уже нет — там идёт работа с последовательностями байт.

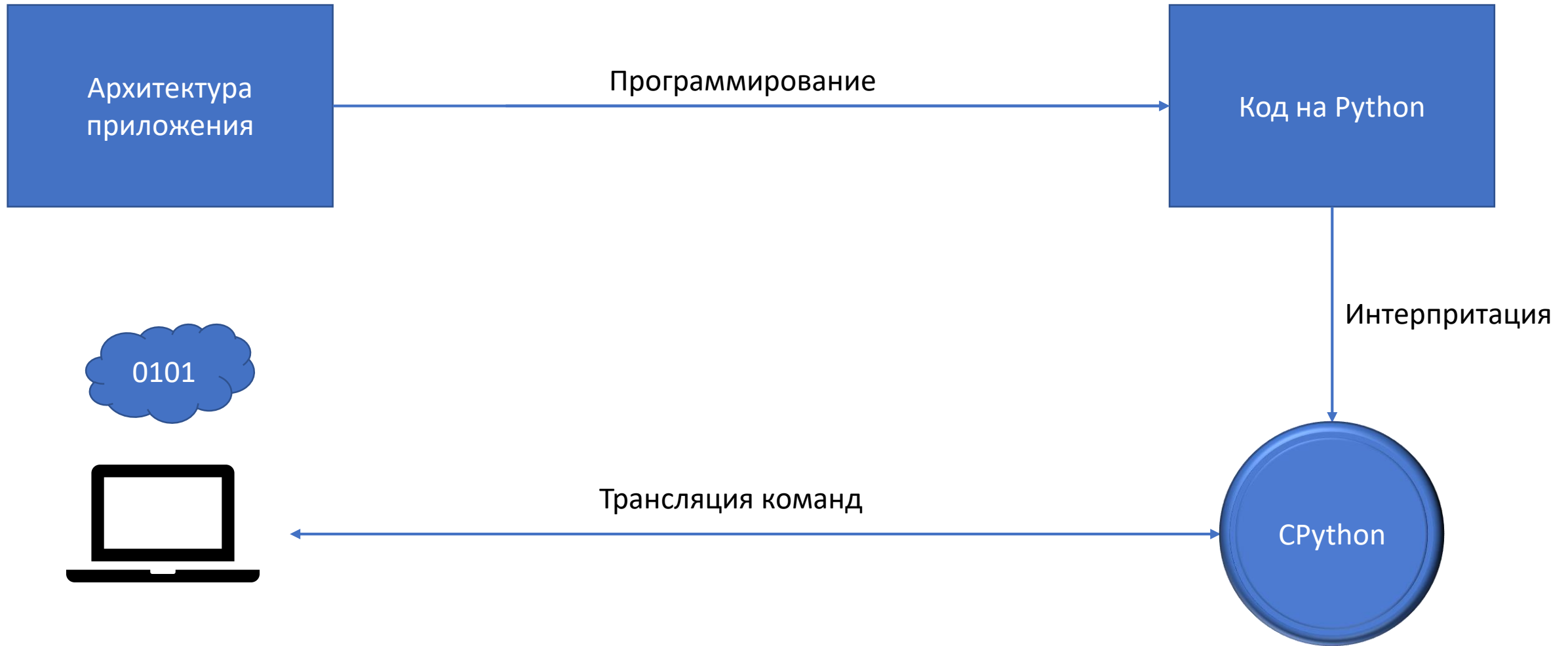
Что такое компиляция?



Что такое компиляция?



Что такое интерпретация?



Плюсы и минусы компилируемости в машинный код

Плюсы:

- эффективность: программа компилируется и оптимизируется для конкретного процессора,
- нет необходимости устанавливать сторонние приложения (такие как интерпретатор или виртуальная машина).

Минусы:

- нужно компилировать для каждой платформы,
- сложность внесения изменения в программу — нужно перекомпилировать заново.

Важно: компиляция — преобразование одностороннее, нельзя восстановить исходный код.

Разбиение программы на файлы

Зачем разбивать программу на файлы?

- С небольшими файлами удобнее работать.
- Разбиение на файлы структурирует код.
- Позволяет нескольким программистам разрабатывать приложение одновременно.
- Ускорение повторной компиляции при небольших изменениях в отдельных частях программы.

Файлы с кодом на C++ бывают двух типов:

1. файлы с исходным кодом (расширение *.cpp, иногда *.c,),
2. заголовочные файлы (расширение *.hpp или *.h).

Этап №1: препроцессор

Язык препроцессора – это специальный язык программирования, встроенный в C++.

- Препроцессор работает с кодом на C++ как с текстом.
- Команды языка препроцессора называют директивами, все директивы начинаются со знака #.
- Директива `#include` позволяет подключать заголовочные файлы к файлам кода.
 1. `#include <foo.h>` — библиотечный заголовочный файл,
 2. `#include "bar.h"` — локальный заголовочный файл.
- Препроцессор заменяет директиву `#include "bar.h"` на содержимое файла `bar.h`.

Этап 2: компиляция

На вход компилятору поступает код на C++ после обработки препроцессором.

- Каждый файл с кодом компилируется отдельно и независимо от других файлов с кодом.
- Компилируются только файлы с кодом (т.е. *.cpp).
- Заголовочные файлы сами по себе ни во что не компилируются, только в составе файлов с кодом.
- На выходе компилятора из каждого файла с кодом получается “объектный файл” — бинарный файл со скомпилированным кодом (с расширением *.o или *.obj).

Этап 3: линковка (компоновка)

На этом этапе все объектные файлы объединяются в один исполняемый (или библиотечный) файл.

- При этом происходит подстановка адресов функций в места их вызова.
- По каждому объектному файлу строится таблица всех функций, которые в нём определены.

Этап 3: линковка (компоновка)

- На этапе компоновки важно, что каждая функция имеет уникальное имя.
- В C++ может быть две функции с одним именем, но разными параметрами.
- Имена функций искажаются (mangle) таким образом, что в их имени кодируются их параметры. Например, компилятор GCC превратит имя функции foo

```
void foo (int , double ) {}
```

в `_Z3fooid`.

- По каждому объектному файлу строится таблица всех функций, которые в нём определены.

Этап 3: линковка (компоновка)

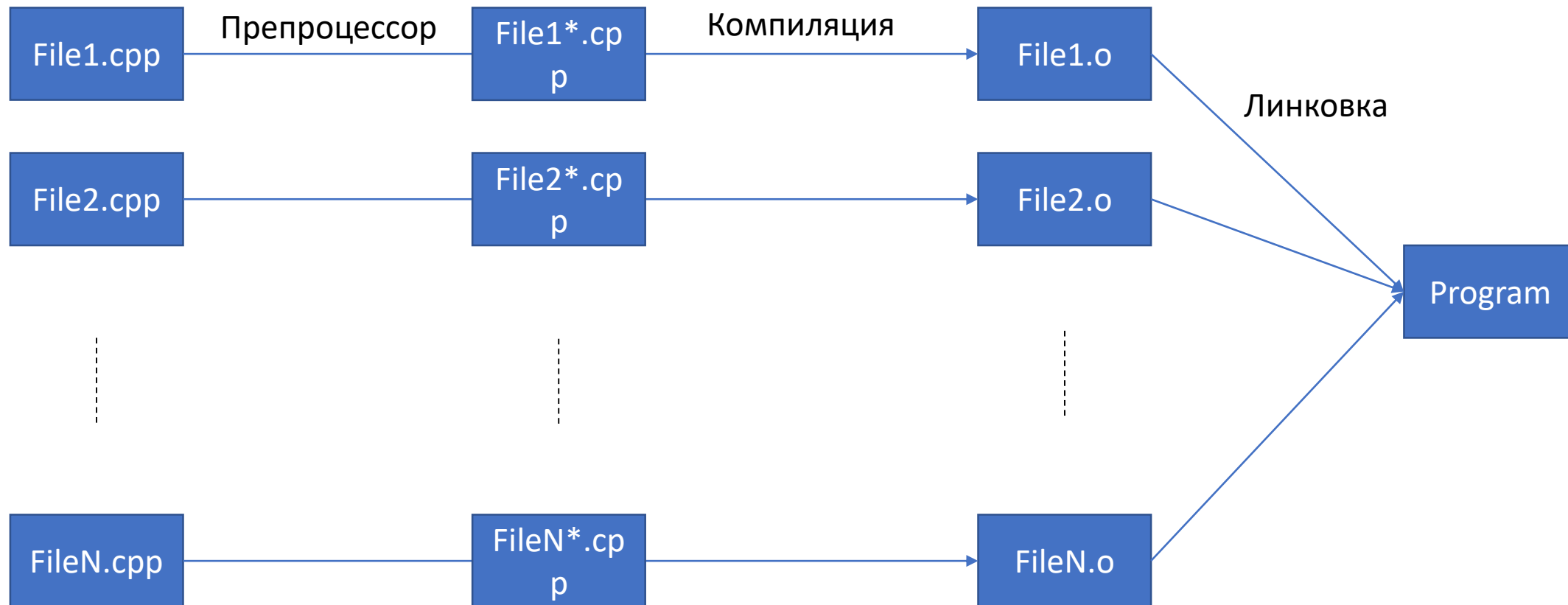
- Точка входа — функция, вызываемая при запуске программы. По умолчанию — это функция `main`:

```
int main()  
{  
    return 0;  
}
```

// Или

```
int main(int argc , char** argv)  
{  
    return 0;  
}
```

Общая схема



Типы данных

Тип данных	Типичный размер в битах	Минимально допустимый диапазон значений
char	8 (или 1 байт)	от −127 до 127
unsigned char	8	от 0 до 255
signed char	8	от −127 до 127
int	16 или 32	от −32 767 до 32 767
unsigned int	16 или 32	от 0 до 65 535
signed int	16 или 32	от −32 767 до 32 767
short int	16	от −32 767 до 32 767
unsigned short int	16	от 0 до 65 535
signed short int	16	от −32 767 до 32 767
long int	32	от −2 147 483 647 до 2 147 483 647
long long int	64	от $-(2^{63} - 1)$ до $(2^{63} - 1)$ для C99
signed long int	32	от −2 147 483 647 до 2 147 483 647
unsigned long int	32	от 0 до 4 294 967 295
unsigned long long int	64	от 0 до $(2^{64} - 1)$ для C99
float	32	от $1E - 37$ до $1E + 37$ (с точностью не менее 6 значащих десятичных цифр)
double	64	от $1E - 37$ до $1E + 37$ (с точностью не менее 10 значащих десятичных цифр)
long double	80	от $1E - 37$ до $1E + 37$ (с точностью не менее 10 значащих десятичных цифр)

Типы данных

Логический тип данных `bool`.

Пустой тип `void`.

Указатели...

- При вычислении размера типа используется `sizeof`

```
#include <iostream>

int main()
{
    int* x = new int(1);
    std::cout << sizeof(x) << std::endl;
}
```

Литералы

- Целочисленные:

1. 'a' — код буквы 'a', тип `char`,
2. 42 — все целые числа по умолчанию типа `int`,
3. 1234567890L — суффикс 'L' соответствует типу `long`,
4. 1703U — суффикс 'U' соответствует типу `unsigned int`,
5. 2128506UL — соответствует типу `unsigned long`.

- Числа с плавающей точкой:

1. 3.14 — все числа с точкой по умолчанию типа `double`,
2. 2.71F — суффикс 'F' соответствует типу `float`,
3. 3.0E8 — соответствует $3.0 * 10^8$.

- `true` и `false` — значения типа `bool`.

- Строки задаются в двойных кавычках: "Text string".

Переменные

- При определении переменной указывается её тип. При определении можно сразу задать начальное значение (инициализация).

```
int i = 10;  
short j = 20;  
bool b = false ;  
unsigned long l = 123123;  
double x = 13.5 , y = 3.1415;  
float z ;
```

- Нужно всегда инициализировать переменные.
- Нельзя определить переменную пустого типа `void`

Операции

- Оператор присваивания: `=`.
- Арифметические:
 1. бинарные: `+` `-` `*` `/` `%`,
 2. унарные: `++` `--`.
- Логические:
 1. бинарные: `&&` `||`,
 2. унарные: `!`.
- Сравнения: `==` `!=` `>` `<` `>=` `<=`.
- Приведения типов: `(type)`.
- Сокращённые версии бинарных операторов: `+=` `-=` `*=` `/=` `%=`.

```
int i = 10;  
i = (20 * 3) % 7;
```

```
int k = i++;  
int l = --i;
```

```
bool b = !(k == 1);  
b = (a == 0) || (1 / a < 1);
```

```
double d = 3.1415;  
float f = (int) d;  
// d = d * (i + k)  
d *= i + k ;
```

Инструкции

- Выполнение состоит из последовательности инструкций.
- Инструкции выполняются одна за другой.
- Порядок вычислений внутри инструкций не определён

```
/* unspecified behavior */  
int i = 10;  
i = ( i += 5) + ( i * 4);
```

- Блоки имеют вложенную область видимости:

```
int k = 10;  
{  
    int k = 5 * i ; // не видна за пределами блока  
    i = ( k += 5) + 5;  
}  
k = k + 1;
```

Условные операторы

- Оператор if:

```
int d = b * b - 4 * a * c ;  
if ( d > 0 ) {  
    roots = 2;  
} else if ( d == 0 ){  
    roots = 1;  
} else {  
    roots = 0;  
}
```

- Тернарный условный оператор:

```
int roots = 0;  
if ( d >= 0 )  
    roots = ( d > 0 ) ? 2 : 1;
```

Циклы

```
for (int k = 0; k < 10; k++) {  
    squares += k * k;  
}
```

```
int squares = 0;  
int k = 0;  
while ( k < 10 ) {  
    squares += k * k;  
    k = k + 1;  
}
```


Циклы

```
int i = 10;  
int sum = 0;  
while (i < 10) {  
    sum += i;  
}  
// sum = 0
```

```
int i = 10;  
int sum = 0;  
do {  
    sum += i;  
} while(i < 10);  
// sum = 10
```

Функции

- В сигнатуре функции указывается тип возвращаемого значений и типы параметров.
- Ключевое слово **return** возвращает значение.

```
double square (double x) {  
    return x * x ;  
}
```

- Переменные, определённые внутри функций, — локальные.
- Функция может возвращать **void**.
- Параметры передаются по значению (копируются).

```
void strange (double x , double y) {  
    x = y ;  
}
```

Макросы

- Параметры макросов нужно оборачивать в скобки:
- Это не избавляет от всех проблем:

```
#define max3(x , y) ((x) > (y) ? (x) : (y))
```

- Определять функции через макросы — плохая идея

```
int a = 1;  
int b = 1;  
int c = max3 (++ a , b);  
// c = ((++a) > (b) ? (++a) : (b))
```

- Макросы можно использовать для условной компиляции:

```
#ifdef DEBUG  
    // дополнительные проверки  
#endif
```

ВВОД-ВЫВОД

```
#include <stdio.h>
```

- В С используется библиотека `stdio.h`

```
#include <stdio.h>
```

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

- В C++ используется библиотека `iostream.h`

- Ввод:

```
int a = 0;  
int b = 0;  
cin >> a >> b;
```

- Вывод:

```
cout << "a + b = " << (a + b) << endl;
```

Простая программа

```
#include <iostream>
using namespace std;
int main () {
    int a = 0;
    int b = 0;
    cout << "Enter a and b : ";
    cin >> a >> b;
    cout << "a + b = " << ( a + b ) << endl;
    return 0;
}
```