Лекция 1. Языки C и C++

Александр Смаль

CS центр
5 сентября 2017
Санкт-Петербург

Язык C

- Язык программирования C++ создан на основе языка C.
- Язык программирования C разработан в начале 1973 года в компании Bell Labs *Кеном Томпсоном* и *Деннисом Ритчи*.
- Язык C был создан для использования в операционной системе UNIX.
- В связи с успехом UNIX язык C получил широкое распространение.
- На данный момент C является одним из самых распространённых языков программирования (доступен на большинстве платформ).
- C — 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
  (против 700+ для Java и 1300+ для C++).

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”.
- Первый компилятор `cfront` перерабатывающий исходный код “C с классами” в исходный код на C.

Развитие C++

- К 1983 году в язык были добавленно 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 ’++’ (увеличение значения переменной на единицу).
- Язык также не был назван D, поскольку “является расширением C и не пытается устранять проблемы путём удаления элементов C”.
- Язык начинает активно развиваться. Появляются новые компиляторы и среды разработки.

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).
- С 2005 года началась работа над новой версией стандарта, которая получила кодовое название C++0x.
- В конце концов в 2011 году стандарт был принят и получил название C++11 ISO/IEC 14882:2011.
- В 2014 году вышел C++14: ISO/IEC 14882:2014.
- В данный момент готовится к публикации C++17.

Совместимость C и C++

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

Характеристики языка C++

Характеристики 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),
- **генеративное программирование**
  (метапрограммирование на шаблонах).

Эффективный

Одна из фундаментальных идей языков C и C++ — *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 C++ 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

C++ является компилируемым языком программирования.

Для того, чтобы запустить программу на C++, 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.

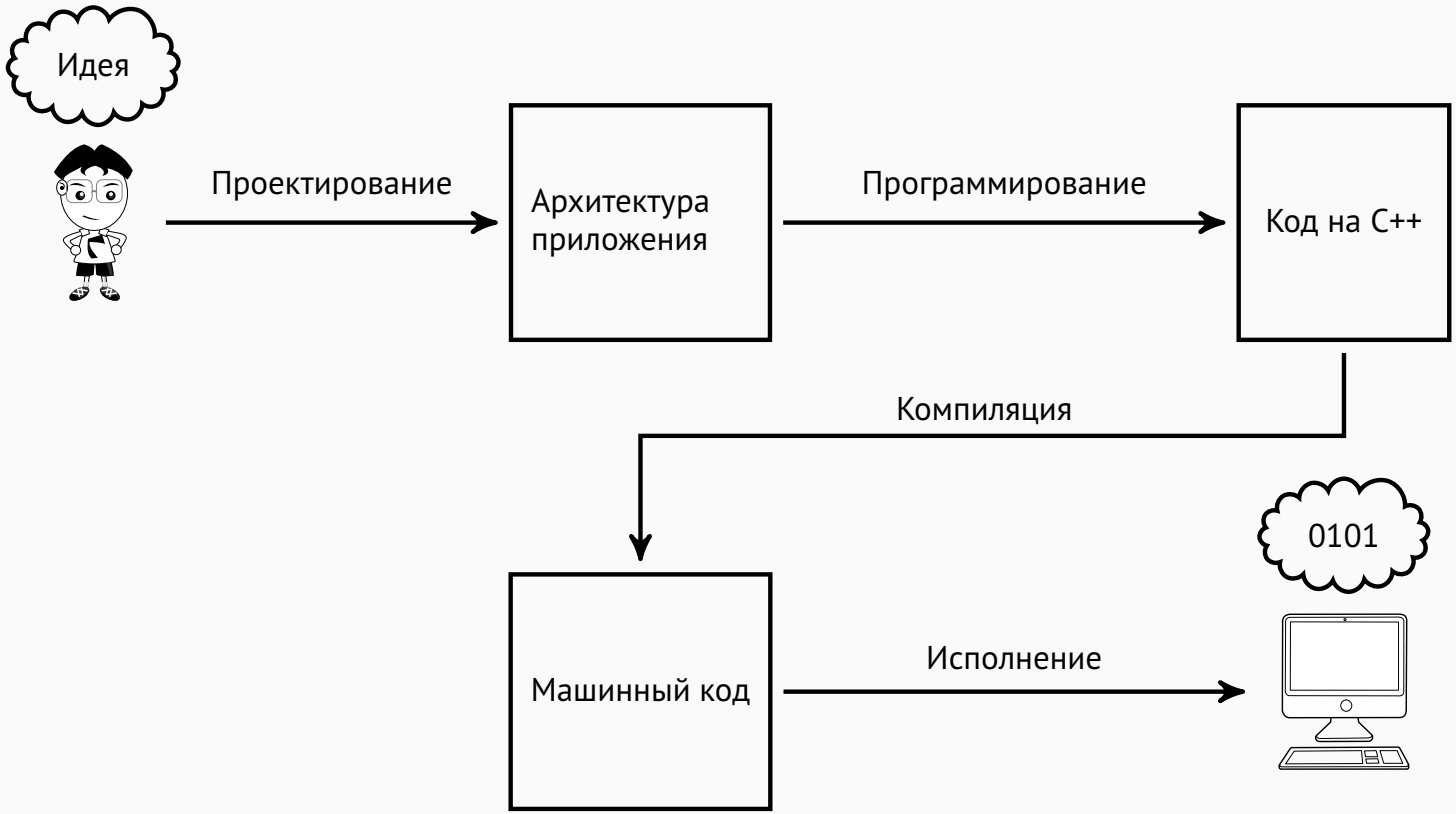
1. Каждая сущность в программе (переменная, функция и пр.) имеет свой тип,
2. и этот тип определяется на момент компиляции.

Это нужно для того, чтобы

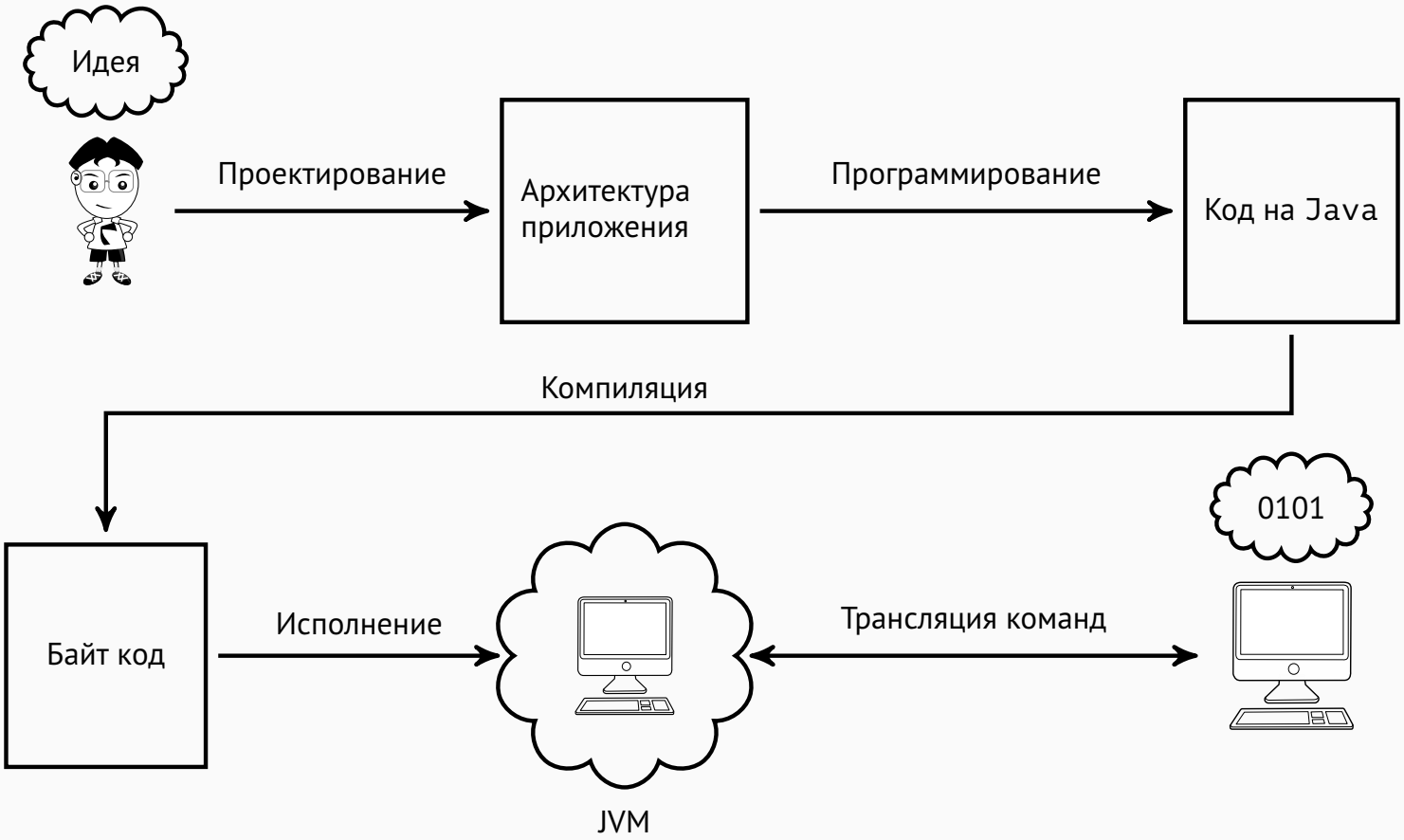
1. вычислить размер памяти, который будет занимать каждая переменная в программе,
2. определить, какая функция будет вызываться в каждом конкретном месте.

Всё это определяется на момент компиляции и “зашивается” в скомпилированную программу.
В машинном коде никаких типов уже нет — там идёт работа с последовательностями байт.

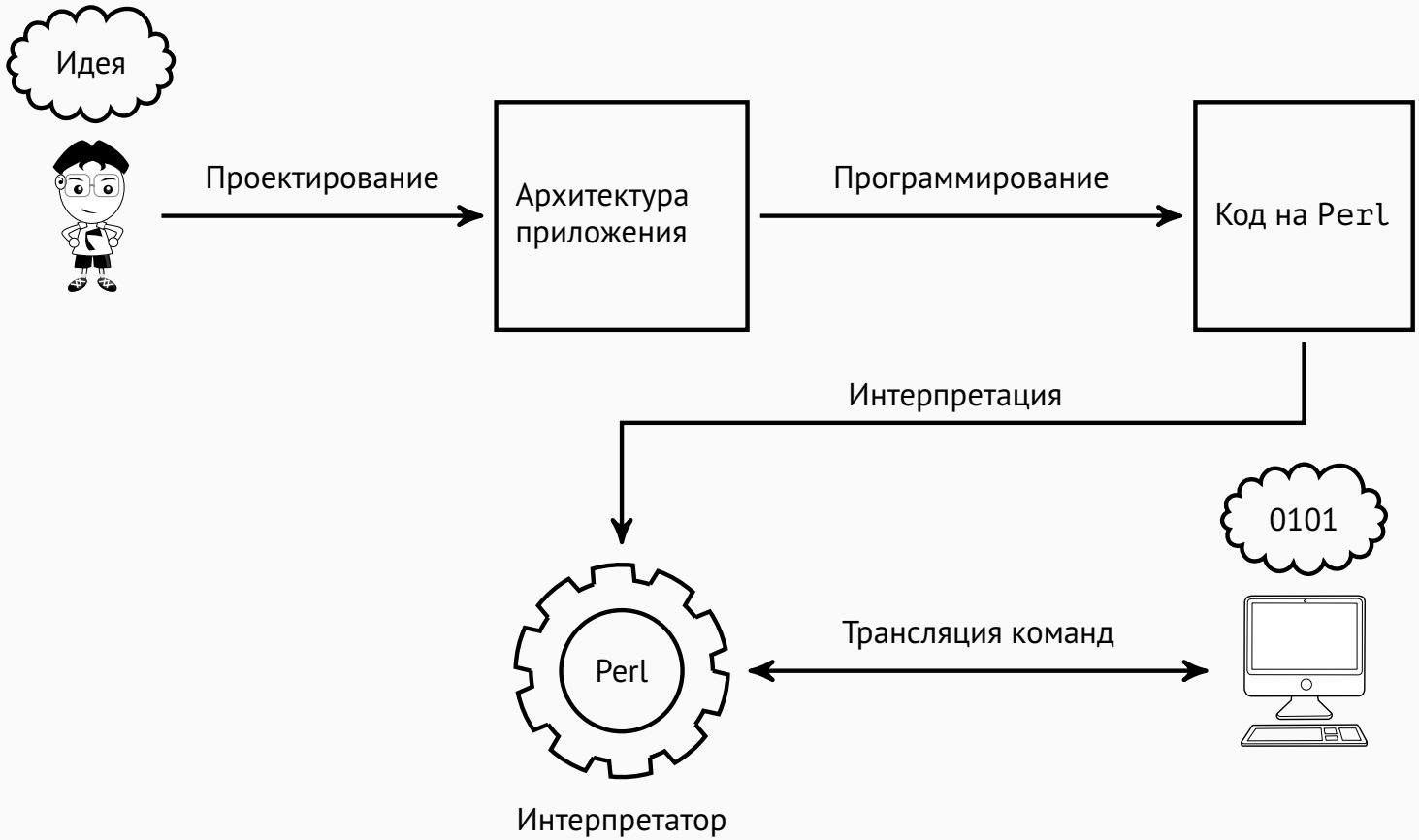
Что такое компиляция?



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`).

Заголовочные файлы

- Файл `foo.cpp`:

```
// определение (definition) функции foo
void foo()
{
    bar();
}
```

- Файл `bar.cpp`:

```
// определение (definition) функции bar
void bar() { }
```

Компиляция этих файлов выдаст ошибку.

Заголовочные файлы

- Файл `foo.cpp`:

```
// объявление (declaration) функции bar
void bar();

// определение (definition) функции foo
void foo()
{
    bar();
}
```

- Файл `bar.cpp`:

```
// определение (definition) функции bar
void bar() { }
```

Заголовочные файлы

Предположим, что мы изменили функцию `bar`.

- Файл `foo.cpp`:

```
void bar();

void foo()
{
    bar();
}
```

- Файл `bar.cpp`:

```
int bar() { return 1; }
```

Данный код некорректен — объявление отличается от определения. (Неопределённое поведение.)

Заголовочные файлы

Добавим заголовочный файл `bar.hpp`.

- Файл `foo.cpp`:

```
#include "bar.hpp"

void foo()
{
    bar();
}
```

- Файл `bar.cpp`:

```
int bar() { return 1; }
```

- Файл `bar.hpp`:

```
int bar();
```

Двойное включение

Может случиться двойное включение заголовочного файла.

- Файл `foo.cpp`:

```
#include "foo.hpp"
#include "bar.hpp"

void foo()
{
    bar();
}
```

- Файл `foo.hpp`:

```
#include "bar.hpp"

void foo();
```

Стражи включения

Это можно исправить двумя способами:

- (наиболее переносимо) Файл `bar.hpp`:

```
#ifndef BAR_HPP
#define BAR_HPP

int bar();
#endif
```

- (наиболее просто) Файл `bar.hpp`:

```
#pragma once

int bar();
```

Резюме: `.cpp` — для определений, `.hpp` — для объявлений.

Этап №1: препроцессор

- Язык препроцессора – это специальный язык программирования, встроенный в C++.
- Препроцессор работает с кодом на C++ как с текстом.
- Команды языка препроцессор 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.
- Компилируется 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.

```
void foo()
{
    bar();
}
```

```
void bar() {  }
```

- 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`.
- Аналогично функциям в линковке нуждаются глобальные переменные.

Этап 3: линковка (компоновка)

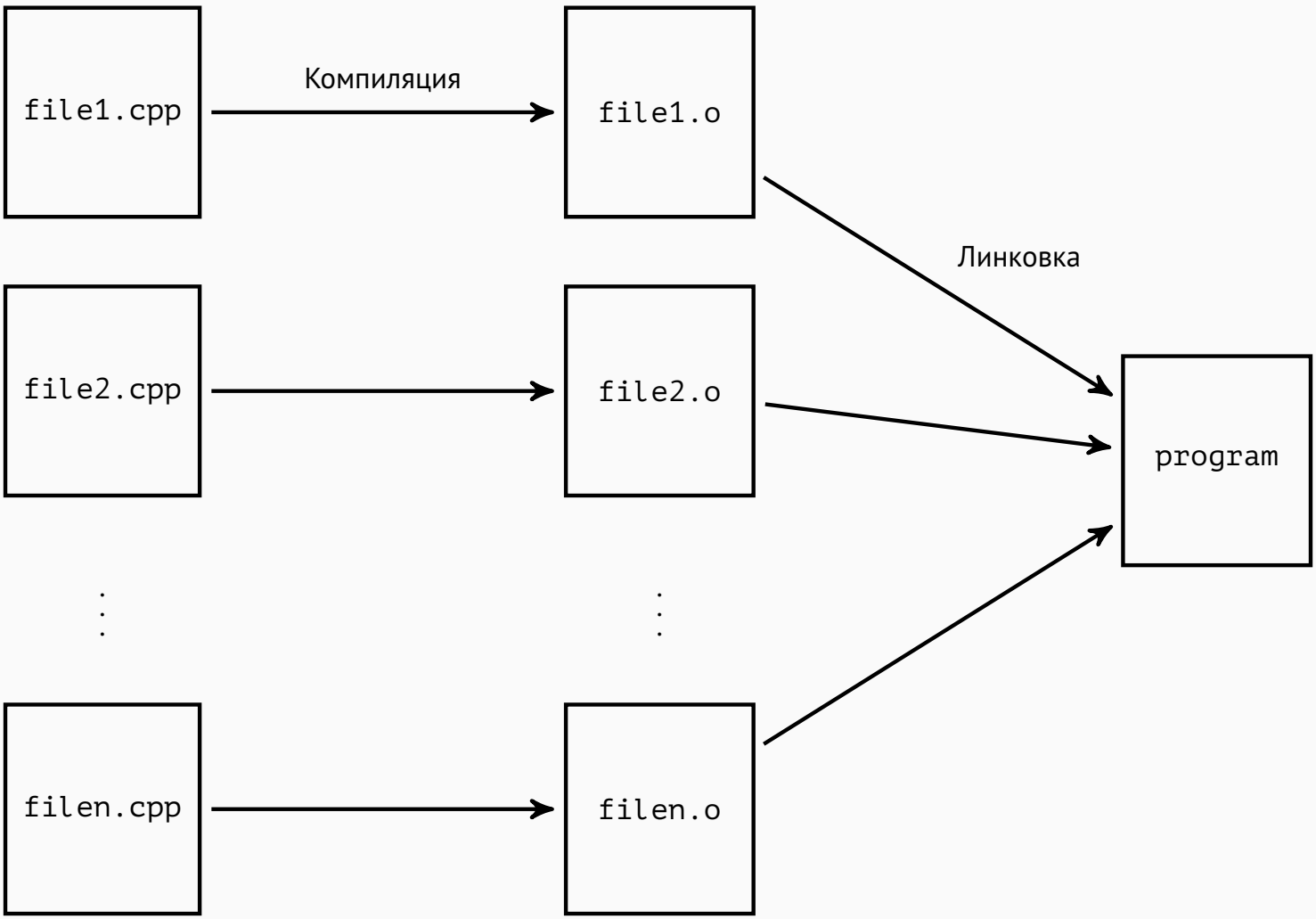
- *Точка входа* — функция, вызываемая при запуске программы. По умолчанию — это функция `main`:

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

или

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

Общая схема


file1.cpp
Компиляция
file1.o
Линковка
file2.cpp
file2.o
program
filen.cpp
filen.o