Лекция 1 – введение в язык C++

Воронин Андрей Андреевич

Кафедра прикладной математики и информатики

30 сентября 2019 г.

Язык 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 распространенных языков программирования (доступен на большинстве платформ).
- C – основной язык низкоуровневой разработки.
- Язык программирования C++ создан на основе языка C.

Особенности языка 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 520 страниц (Java 772, C++ 1605).

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, перерабатывал исходный код ”C with classes” в исходный код на C.

Развитие C++

- К 1983 году в язык были добавлены множество новых возможностей (виртуальные функции, перегрузка функций и операторов, ссылки, константы, ...)
- Получившийся язык перестал быть просто дополненной версией классического C и был переименован из ”C with classes” в C++.
- Имя языка, получившегося 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 путем 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 недочеты 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.
- C++14 ISO/IEC 14882:2014 небольшое расширение и исправление ошибок предыдущего стандарта.
- Стандарт C++17 ISO/IEC 14882:2017 принес изменения направленные на большую безопасность языка.
- C++20 находится в разработке и обещает серьёзный набор изменений включающих изменение процесса сборки.

Совместимость C и C++

- Один из принципов разработки стандарта C++ – это сохранение совместимости с C.
- Синтаксис C++ унаследован от языка C.
- C++ в строгом смысле не является 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.

Сложность C++

C makes it easy to shoot yourself in the foot. C++ makes it harder, but when you do, it blows away your whole leg.
(В языке C легко прострелить себе ногу. В C++ это сложнее, но если вы сделаете это, то отстрелите всю ногу целиком.)

Bjarne Stroustrup

Every extension proposal should be required to be accompanied by a kidney. People would submit only serious proposals, and nobody would submit more than two.
(Нужно, чтобы к каждому предложению о расширении языка обязательно прилагалась почка. Тогда люди присылали бы только очень важные предложения, и никто не прислал бы более двух.)

Jim Waldo

Сложность C++

- Описание стандарта занимает 1605 страниц текста.
- Нет никакой возможности рассказать ”весь C++” в рамках одного, путь даже очень большого курса.
- В C++ программисту позволено делать очень многое, и это влечет за собой большую ответственность.
- На плечи программиста ложится много дополнительной работы:
  - проверка корректности данных,
  - управление памятью,
  - обработка низкоуровневых ошибок.

Мультипарадигменность

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

- процедурное программирование (код ”в стиле C”);
- объектно-ориентированное программирование (классы, наследование, виртуальные функции, ...);
- обобщенное программирование (шаблоны функций и классов);
- функциональное программирование (функторы, безымянные функции, замыкания);
- генеративное программирование (метапрограммирование на шаблонах).

Эффективность

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).

Низкоуровневость

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.

Компилируемость

C++ является компилируемым языком программирования. Для того, чтобы запустить программу на C++, ее 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 идет работа с последовательностями байтов.

Выберите все верные утверждения из списка.

- ☐ C++ ориентирован на написание эффективных приложений.
- ☐ C++ поддерживает объектно-ориентированное программирование.
- ☐ C++ не поддерживает процедурное программирование.
- ☐ C++ поддерживает процедурное программирование.
- ☐ C++ интерпретируемый язык программирования
- ☐ C++ ориентирован на безопасность работы с памятью.

Выберите все верные утверждения из списка.

- ☒ C++ ориентирован на написание эффективных приложений.
- ☒ C++ поддерживает объектно-ориентированное программирование.
- ☐ C++ не поддерживает процедурное программирование.
- ☒ C++ поддерживает процедурное программирование.
- ☐ C++ интерпретируемый язык программирования
- ☐ C++ ориентирован на безопасность работы с памятью.

Что такое компиляция?

Проектирование → Архитектура программы

Программирование → Код на C++

Компиляция → Машинный код

Исполнение →

Что такое компиляция?

Проектирование

Архитектура программы

Программирование

Код на Java

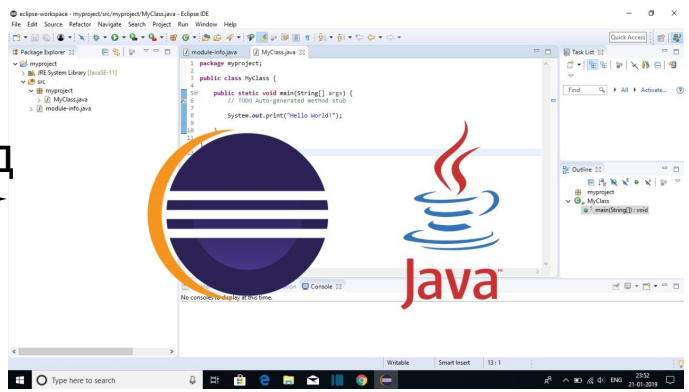
Компиляция

Байт код

Исполнение

Java Virtual Machine (JVM)

Трансляция команд



Что такое компиляция?

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 изменений в программу – нужно перекомпилировать заново.

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

Выберите все верные утверждения из списка.

- ☐ Код программы, написанный на интерпретируемом языке, можно без предварительной компиляции запустить на любой платформе, где есть интерпретатор этого языка.
- ☐ Код программы, написанный на языке, который компилируется в байт код виртуальной машины, достаточно скомпилировать однажды, чтобы программу можно было запускать на любой платформе, где есть соответствующая виртуальная машина.
- ☐ Для запуска программы, код которой был написан на компилируемом языке, на компьютере должен быть установлен компилятор этого языка.
- ☐ Код программы, написанный на языке, который компилируется в машинный код, достаточно скомпилировать однажды, и потом программу можно будет запустить на любой платформе.
- ☐ Для запуска программы, код которой был написан на интерпретируемом языке, на компьютере должен быть установлен интерпретатор этого языка.
- ☐ Скомпилировать программу на C++ для некоторой архитектуры X можно только на компьютере с архитектурой X.

Выберите все верные утверждения из списка.

- ☒ Код программы, написанный на интерпретируемом языке, можно без предварительной компиляции запустить на любой платформе, где есть интерпретатор этого языка.
- ☒ Код программы, написанный на языке, который компилируется в байт код виртуальной машины, достаточно скомпилировать однажды, чтобы программу можно было запускать на любой платформе, где есть соответствующая виртуальная машина.
- ☐ Для запуска программы, код которой был написан на компилируемом языке, на компьютере должен быть установлен компилятор этого языка.
- ☐ Код программы, написанный на языке, который компилируется в машинный код, достаточно скомпилировать однажды, и потом программу можно будет запустить на любой платформе.
- ☒ Для запуска программы, код которой был написан на интерпретируемом языке, на компьютере должен быть установлен интерпретатор этого языка.
- ☐ Скомпилировать программу на C++ для некоторой архитектуры X можно только на компьютере с архитектурой X.

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`, `.cxx`, `.cc` иногда `.C`)
2. заголовочные файлы (расширение `.hpp`, `hxx`, `hh` или `.h`)

Заголовочные файлы

Определение функции

Файл `foo.cpp`:

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

Файл `bar.cpp`:

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

Компиляция этих файлов вызовет ошибку.

Заголовочные файлы

Объявление и определение

Файл `foo.cpp`:

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

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

Файл `bar.cpp`:

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

Заголовочные файлы

Неопределенное поведение

Файл `foo.cpp`:

```
void bar();

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

Файл `bar.cpp`:

```
// определение (defenition) функции bar
int bar(){ return 1; }
```

Данный код некорректен – объявление отличается от определения. (Неопределенное поведение.)

Заголовочные файлы

Подключение заголовочных файлов

Файл `foo.cpp`:

```
#include "bar.hpp"

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

Файл `bar.cpp`:

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

Файл `bar.hpp`:

```
int bar();
```

Заголовочные файлы

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"

int foo();
```

Заголовочные файлы

Защита от двойного включения

Предыдущую ситуацию можно исправить двумя способами:

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

```
#ifndef BAR_HPP
#define BAR_HPP

int bar();

#endif //BAR_HPP
```

- (наиболее просто) Файл `bar.hpp`:

```
#pragma once

int bar();
```

Объявление и определение

Объявление (declaration) — вводит имя, возможно, не определяя деталей. Например, ниже перечислены объявления:

- `int a;`
- `void foo();`
- `void bar() { foo(); }`

Определение (definition) — это объявление, дополнительно определяющее детали, необходимые компилятору. Из перечисленных выше объявлений, определениями являются только два:

- `int a;`
- `void bar() { foo(); }`

Для определения переменной достаточно указать ее тип, а для определения функций, кроме имени, типов параметров и возвращаемого значения, нужно указать еще тело функции. Проще говоря, определение содержит всю информацию, необходимую компилятору, чтобы выделить память для хранения объекта.
В C++ есть также возможность объявить переменную, не определяя ее:
`extern int a;`

Объявление и определение

- `extern int a;` – объявление переменной типа `int`,
- `int a;` – объявление и **определение** переменной типа `int`,
- `void foo();` – объявление функции с именем `foo`
- `void bar(){ foo(); }` – объявление и **определение** функции с именем `bar`

Интересно отметить, что файлы стандартной библиотеки C++ не используют расширение вовсе, например:

- iostream,
- algorithm,
- vector.

Разделение на файлы с исходным кодом и заголовочные файлы чисто условное, нет правил, запрещающих использовать .cpp файл как заголовочный, однако мы не рекомендуем так делать — использование общепринятых правил именования файлов упростит жизнь вам и вашим коллегам.
Не стоит помещать *определения* в заголовочные файлы без явной необходимости. В C++ есть способы, позволяющие поместить определение в заголовочный файл, не вызвав при этом ошибки компоновщика, но, как правило, это приводит к увеличению объектного файла и программы в целом

Выберите из списка объявления, которые **не** стоит помещать в заголовочные файлы.

- ☐ `void foo() { std::cout << "Hello, World!\n "; }`
- ☐ `void foo();`
- ☐ `int a;`
- ☐ `extern int a;`
- ☐ `void bar() { foo(); }`

Выберите из списка объявления, которые **не** стоит помещать в заголовочные файлы.

- ☒ `void foo() { std::cout << "Hello, World!\n "; }`
- ☐ `void foo();`
- ☒ `int a;`
- ☐ `extern int a;`
- ☒ `void bar() { foo(); }`

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.
- Пр этом происходит подстановка адресов функций в места их вызова.

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

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

- По каждому объектному файлу строится таблица всех функций, которые в нем 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 a, double b) {}
```

в `_Z3fooid`

Этап №3: Линковка (компоновка)

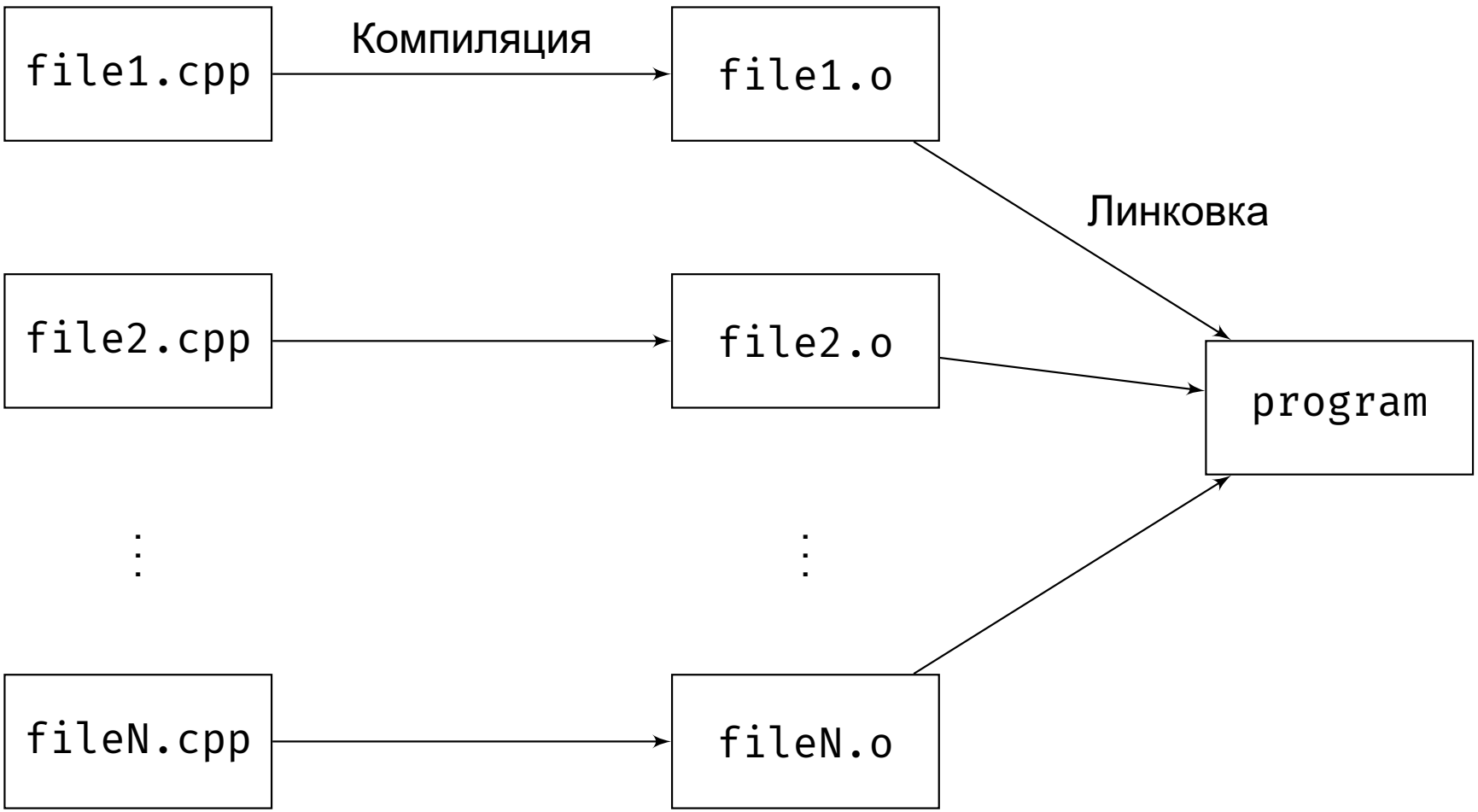
- *Точка входа* – функция, вызываемая при запуске программы. По умолчанию – это функция `main`:

```
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

Выберите все верные утверждения из списка.

- ☐ Даже для программы состоящей из одной пустой функции `int main() { return 0; }` все равно требуется линковка.
- ☐ Если в коде C++ вы вызываете необъявленную функцию, то это не ошибка, при условии, что функция была где-то определена.
- ☐ Если в коде C++ вы вызываете функцию, которая была объявлена, но не была определена, то это ошибка этапа компиляции.
- ☐ Для программы, состоящей всего из одного файла, не требуется линковка.
- ☐ Если в коде C++ вы вызываете необъявленную функцию, то это ошибка этапа компиляции.
- ☐ Если в коде C++ вы вызываете функцию, которая была объявлена, но не была определена, то это ошибка этапа линковки.

Выберите все верные утверждения из списка.

- ☒ Даже для программы состоящей из одной пустой функции `int main() { return 0; }` все равно требуется линковка.
- ☐ Если в коде C++ вы вызываете необъявленную функцию, то это не ошибка, при условии, что функция была где-то определена.
- ☐ Если в коде C++ вы вызываете функцию, которая была объявлена, но не была определена, то это ошибка этапа компиляции.
- ☐ Для программы, состоящей всего из одного файла, не требуется линковка.
- ☒ Если в коде C++ вы вызываете необъявленную функцию, то это ошибка этапа компиляции.
- ☒ Если в коде C++ вы вызываете функцию, которая была объявлена, но не была определена, то это ошибка этапа линковки.