Программирование

Лекция 1

Зачем изучать программирование?

- Во-первых, это интересно.
- Во-вторых, программирование здорово облегчает жизнь во многих профессиях.
- В-третьих, можно хорошо зарабатывать и заниматься в тёплых уютных офисах современными технологиями.

Язык программирования

- В нашем курсе мы будем изучать язык программирования **C++**, так как на сегодняшний момент **C++** один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. На нем пишутся самые различные приложения: от небольших десктопных программ до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

Структура дисциплины

- 2 семестр. Экзамен
- 3 семестр. Диф.зачет
- 4 семестр. Экзамен

Основные понятия

Алгоритм – это последовательность действий для достижения поставленной цели

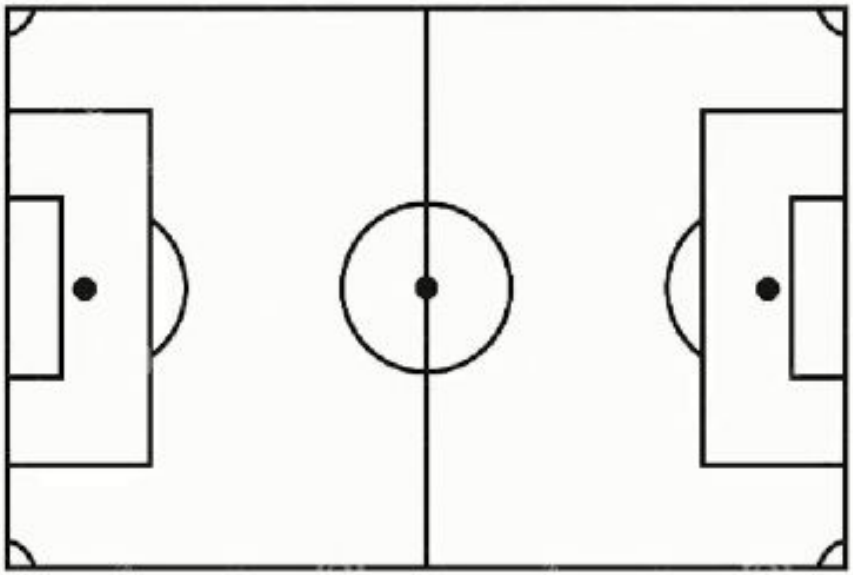
Язык программирования – некий язык, который понимает компьютер

Программа – это **алгоритм**, записанный на некотором **языке программирования**

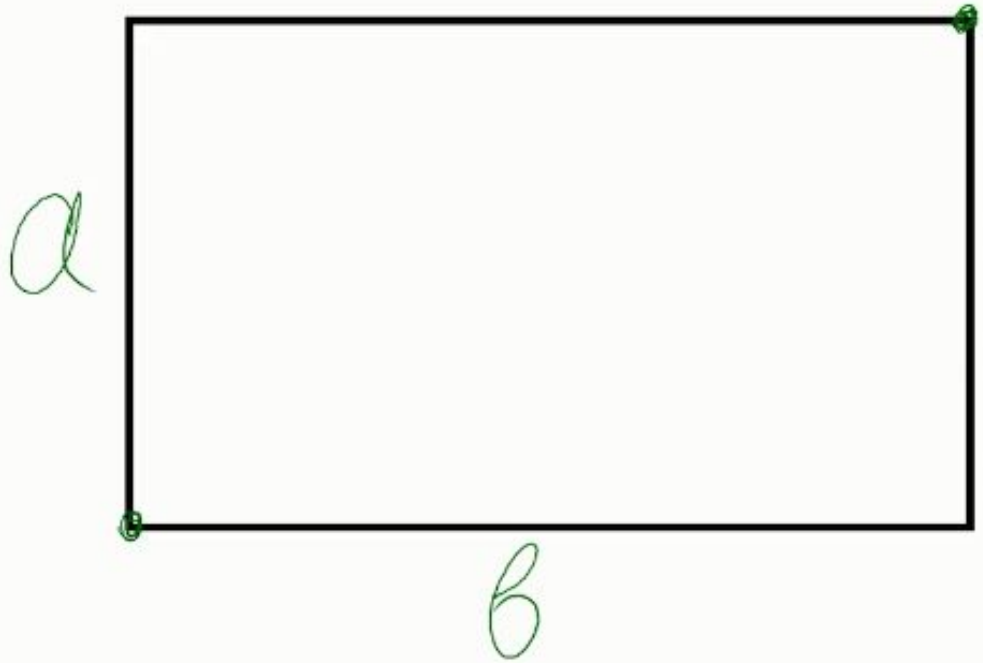
Этапы решения задач

1. Постановка задачи – описание абстрактной задачи
2. Формализация – перевод на математический язык
3. Алгоритмизация – составление алгоритма, который решает задачу
4. Программирование – реализация алгоритма на языке программирования
5. Тестирование – тестирование программы

Постановка задачи

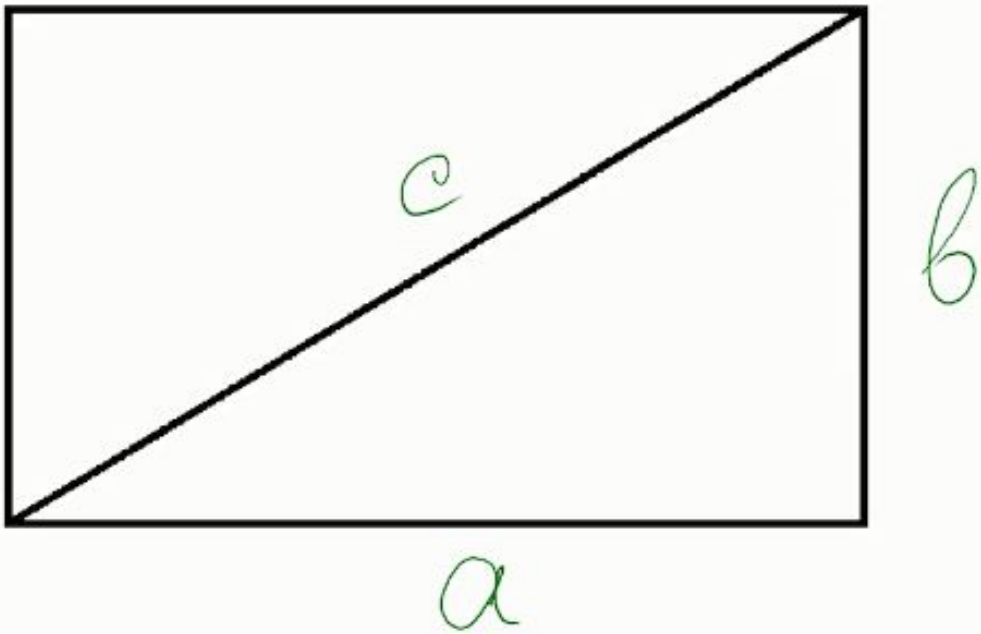


Формализация



Алгоритмизация

$$c = \sqrt{a^2 + b^2}$$



Тестирование

a	b	c
3	4	5
4	3	5
1	12	?
3	2	?

Среда разработки

Перейдите по ссылке https://www.visualstudio.com/ru/thank-you-downloading-visual-studio/?sku=Community&rel=15 и скачивание начнется автоматически.

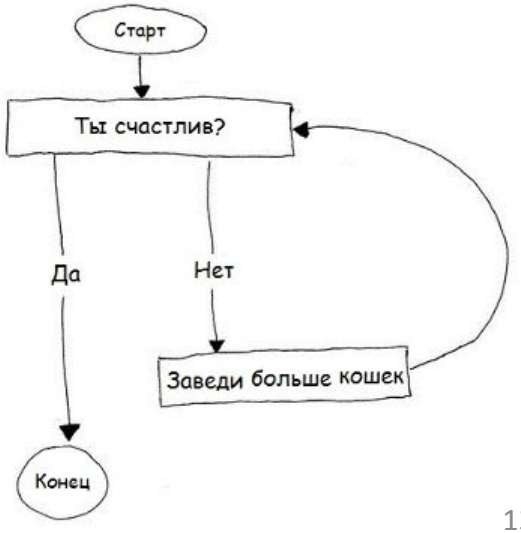
C++

- Произошел из C (процедурный язык)
- Объектно-ориентированный язык программирования
- C++ является очень мощным языком программирования, но вместе с тем на его изучение нужно потратить довольно много времени.

Истоки языка C++: немного истории

- Развитие компьютерных технологий в течение последних нескольких десятков лет происходило удивительно быстрыми темпами.
- В 70х годах прошлого столетия такие языки программирования, как C и Pascal, способствовали зарождению эры структурного программирования.

<u>Структурное программирование</u> — методология разработки программного обеспечения, в основе которой лежит представление программы в виде иерархической структуры блоков. Базовые управляющие структуры: последовательность, ветвление, цикл.

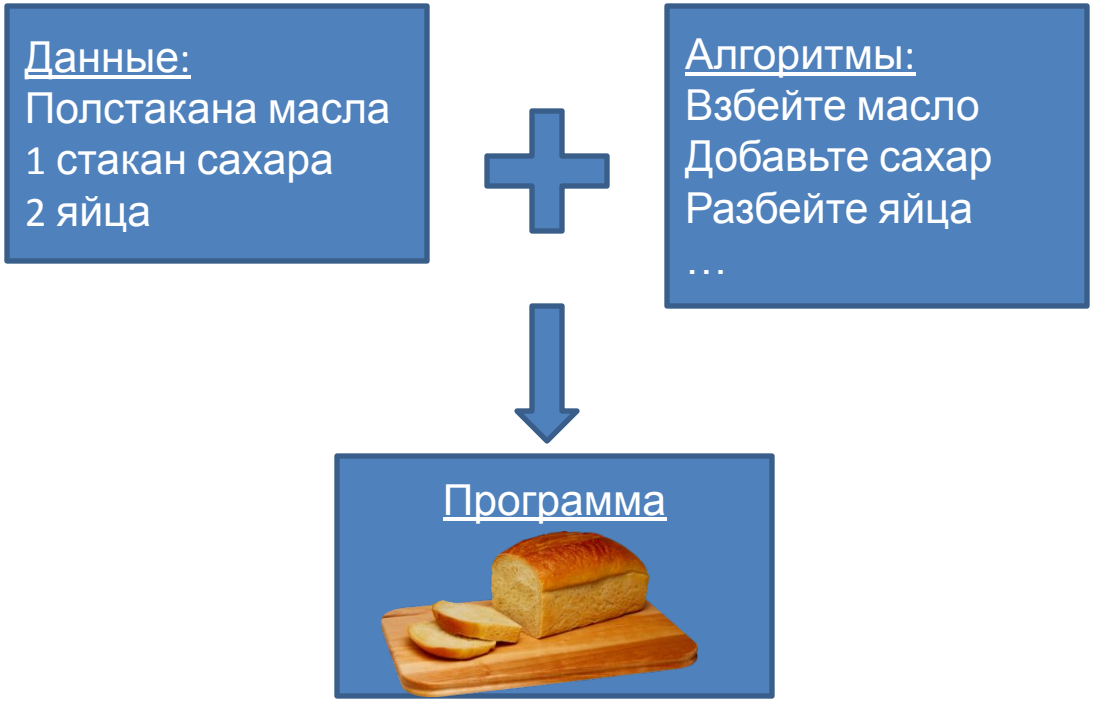


Язык программирования С

- В начале семидесятых годов прошлого столетия Деннис Ритчи, сотрудник компании Bell Laboratories, участвовал в проекте по разработке операционной системы Unix.
- Ритчи необходим был язык, который сочетал бы в себе эффективность языка низкого уровня и возможность доступа к аппаратным средствам с универсальностью и переносимостью языка высокого уровня.
- Поэтому он создал язык С.

Философия программирования на языке С

- В общем случае компьютерные языки имеют дело с двумя концепциями — данные и алгоритмы. Данные — это информация, которую использует и обрабатывает программа. Алгоритмы — это методы, используемые программой.



Философия программирования на языке С

- Спагетти-код (spaghetti code) — плохо спроектированная, слабо структурированная, запутанная и трудная для понимания программа, содержащая много операторов goto (особенно переходов назад), исключений и других конструкций, ухудшающих структурированность.
- Спагетти-код назван так потому, что ход выполнения п а миску спагетт

Переход к C++: объектно-ориентированное программирование (ООП)

Класс Руководитель компании
Фамилия
Должность
Годовой доход
Необычные способности
…

Объект
Иванов
Вице-президент компании
$ 900 000
Умеет восстанавливать системный реестр Windows)
…

Процесс перехода с нижнего уровня организации, например, с классов, до верхнего уровня — проектирования программы, называется ***восходящим программированием***.

Происхождение языка программирования C++

- Как и C, язык C++ был создан в начале восьмидесятых годов прошлого столетия в Bell Laboratories, где работал Бьярне Страуструп (Bjarne Stroustrup).
- «C++ был создан главным образом потому, что мои друзья, да и я сам, не имели никакого желания писать программы на ассемблере, C или каком-нибудь языке программирования высокого уровня, существовавшем в то время. Задача заключалась в том, чтобы сделать процесс написания хороших программ простым и более приятным для каждого программиста» - Страуструп.
- http://www.stroustrup.com/

Переносимость

- Если программу можно перекомпилировать, ничего в ней не меняя, и без помех запустить, то такая программа называется *переносимой*.
- Кроссплатформенность — способность программного обеспечения работать более чем на одной аппаратной платформе и (или) операционной системе.
- C++ является кроссплатформенным ЯП.

Характеристики языка C++

Характеристики C++:

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

Байки о сложности C++

- Есть интересная фраза Бьёрна Страуструпа о языках C и 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.)

  Таким образом, несмотря на сложность C++, выстрелить себе в ногу в C гораздо проще, впрочем, последствия “удачного” самострела в C++ могут быть действительно печальными. Такова цена мощности языка — программист получает больше возможностей, но вместе с тем и ответственность становится больше.

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.

C++ -> .exe

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.

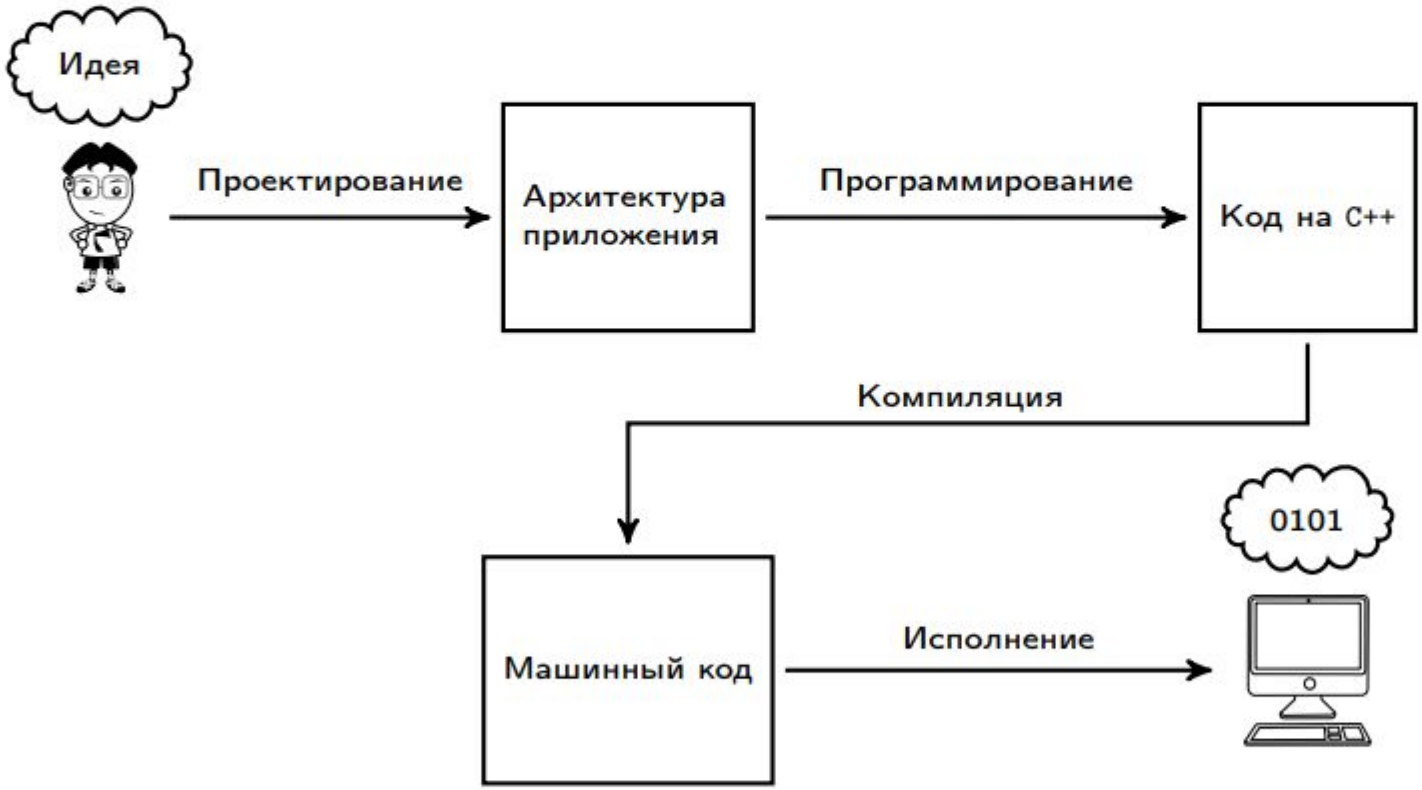
Всё это определяется на момент компиляции и "зашивается" в скомпилированную программу.
В машинном коде никаких типов уже нет — там идёт работа с последовательностями байт.

Выберите все верные утверждения из списка

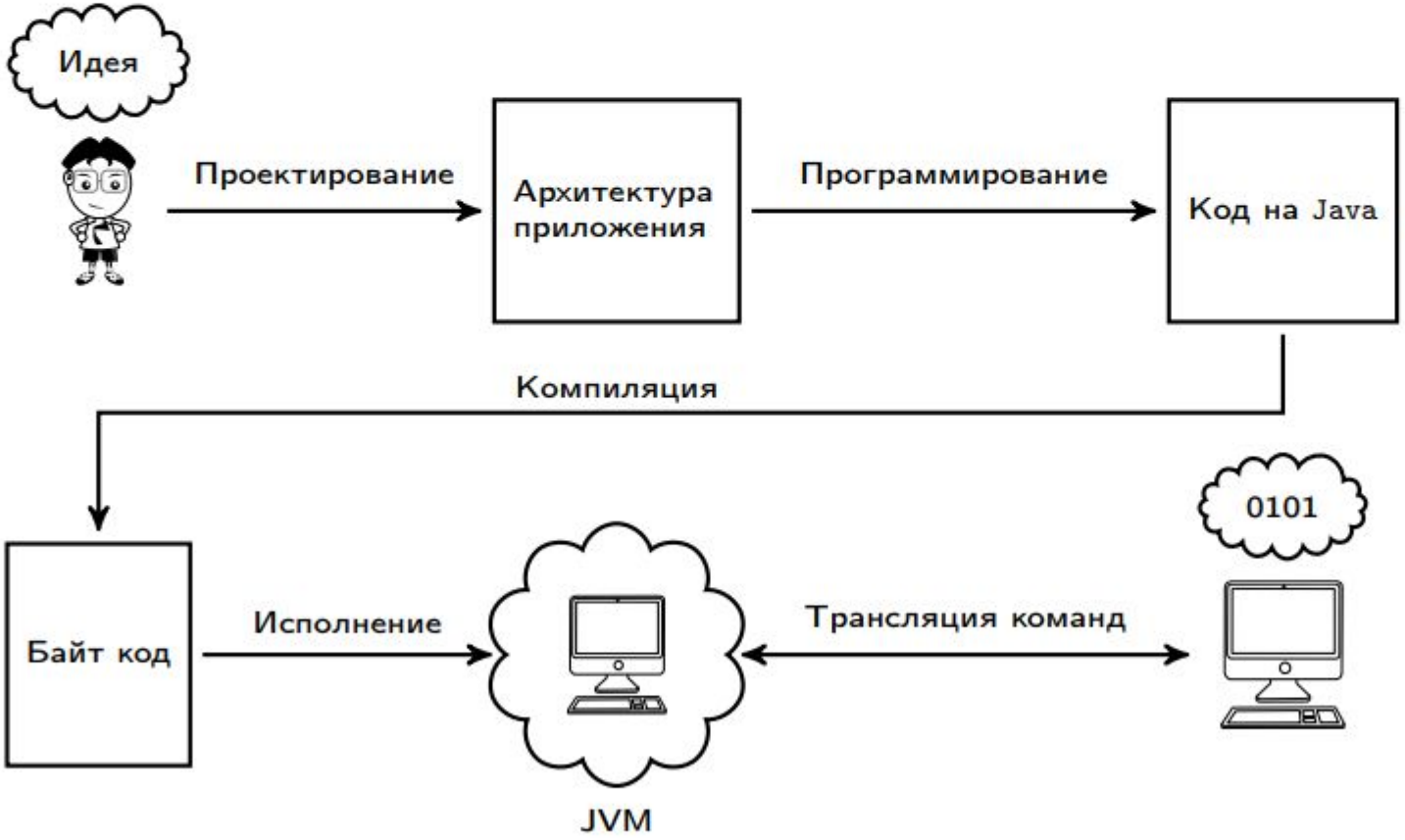
1. C++ не поддерживает объектно-ориентированное программирование.
2. C++ поддерживает процедурное программирование.
3. C++ компилируемый язык программирования.
4. C++ интерпретируемый язык программирования.
5. C++ язык со статической типизацией.
6. C++ ориентирован на безопасность работы с памятью.

- [ ] C++ не поддерживает объектно-ориентированное программирование.
- [x] C++ поддерживает процедурное программирование.
- [x] C++ компилируемый язык программирования.
- [ ] C++ интерпретируемый язык программирования.
- [x] C++ язык со статической типизацией.
- [ ] C++ ориентирован на безопасность работы с памятью.

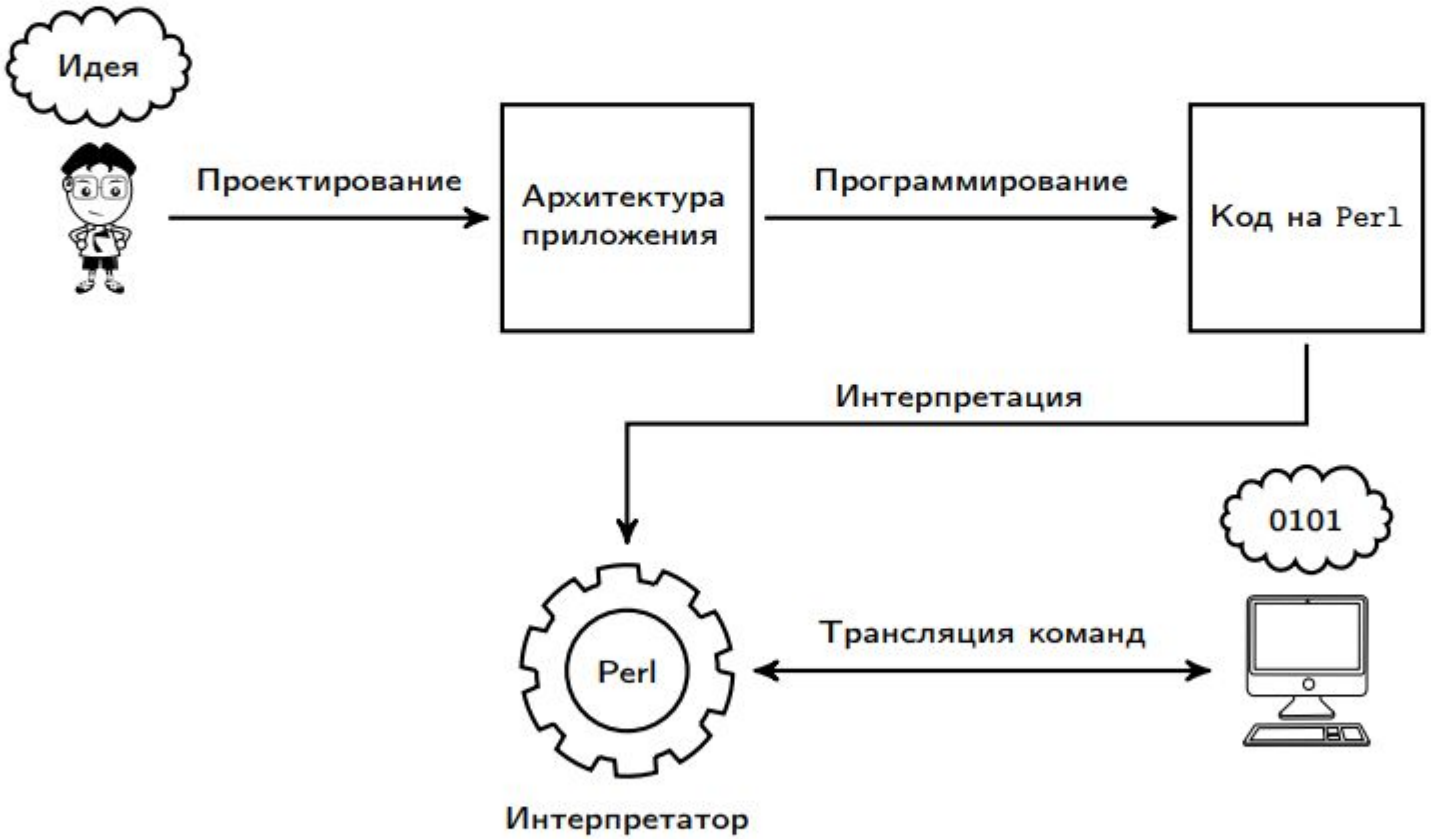
Что такое компиляция?



Что такое компиляция?



Что такое интерпретация?



PHP,
Python,
Ruby

(одновременно и компилятор и виртуальная машина)

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();
}
```

Допустим, что функция bar() используется не в одном файле, а в нескольких. Тогда придется исправлять объявление во всех файлах.

- Файл bar.cpp:

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

- Файл bar.hpp:

```
int bar();
```

Заголовочный файл. Его подключаем с помощью include

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.

Структура кода на C++

- **Объявление (declaration)** — вводит имя, возможно, не определяя деталей. Например, ниже перечислены объявления:
- int a; — объявление переменной типа int,
- void foo(); — объявление функции с именем foo,
- void bar() { foo(); } — объявление функции с именем bar.

- **Определение (definition)** — это объявление, дополнительно определяющее детали, необходимые компилятору. Из перечисленных выше объявлений, определениями являются только два:
  int a; — объявление переменной типа int,
- void bar() { foo(); } — объявление функции вместе с телом является определением.

- 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;

  Ключевое слово **extern** как раз и позволяет сказать компилятору, что переменную нужно только объявить, при этом не нужно выделять под нее память — память под нее должна быть выделена в другом месте (возможно даже в другом файле)

Структура кода на C++

- Кроме указанных в лекции, в C++ также используются следующие расширения:
- .cxx, .cc — для файлов с исходным кодом,
- .hxx, .hh — для заголовочных файлов.
- 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 не рекомендуется так делать — использование общепринятых правил именования файлов упростит жизнь вам и вашим коллегам.

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

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

Ответ

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

Порядок создания программы

Первые шаги в C++

```
// myfirst.cpp -- выводит сообщение на экран
#include <iostream>                          // директива препроцессора
int main()                                   // заголовок функции
{                                            // начало тела функции
   using namespace std;                      // делает видимыми определения
   cout << "Come up and C++ me some time."; // сообщение
   cout << endl;                             // начало новой строки
   cout << "You won't regret it!" << endl;   // дополнительный вывод
   return 0;                                 // завершение функции main()
}                                            // конец тела функции
```


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

Комментарии в языке C++

В C++ комментарий обозначается двумя косыми чертами (//). Комментарий— это примечание, написанное программистом для пользователя программы, которое обычно идентифицирует ее раздел или содержит пояснения к определенному коду. Компилятор игнорирует комментарии.

```
// myfirst.cpp -- выводит сообщение на экран
#include <iostream>  /* комментарий в стиле C */
```

Совет
Используйте комментарии для документирования своих программ. Чем сложнее программа, тем более ценными будут ваши комментарии. Они помогут не только другим пользователям разобраться с вашим кодом, но и вы сами сможете вспомнить, что он делает, по прошествии некоторого времени.

Препроцессор C++ и файл iostream

Препроцессор — это программа, которая выполняет обработку файла исходного кода перед началом собственно компиляции.

```
#include <iostream> // директива препроцессора

using namespace std;
```

Директиву using можно опустить и записать код следующим образом:

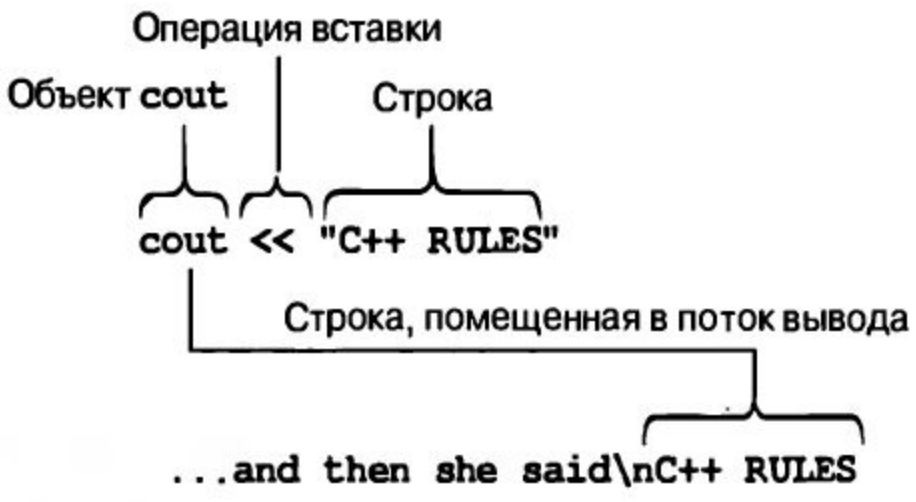
```
std::cout << "Come up and C++ me some time.";
std::cout << std::endl;
```

Вывод в C++ с помощью cout

```
cout << "Come up and C++ me some time.";
```

Часть, заключенная в двойные кавычки — это сообщение, которое необходимо вывести на экран. Запись << означает, что оператор отправляет строку в cout; символы указывают на направление передачи информации.

```
cout << string;
```



Манипулятор endl и символ новой строки

```
cout << endl;
```

endl — это специальное обозначение в C++, которое представляет понятие начала новой строки.
если опустить манипулятор endl, результат будет таким

```
Come up and C++ me some time.You won't regret it!
```

Рассмотрим код:

```
cout << "The Good, the";
cout << "Bad, ";
cout << "and the Ukulele";
cout << endl;
```

В результате его выполнения будет выведена следующая строка:

```
The Good, theBad, and the Ukulele
```

```
cout << "Jupiter is a large planet.\n";       // отображает текст,
                                               // переходит на следующую строку
cout << "Jupiter is a large planet." << endl; // отображает текст,
                                               // переходит на следующую строку
cout << "\n";    // начинает новую строку
cout << endl;    // начинает новую строку
```

Лексемы и пробельные символы в исходном коде

- *Лексемами* называются неделимые элементы в строке кода.
- Как правило, для разделения лексем друг от друга используется пробел, табуляция или возврат каретки, которые все вместе называются *пробельными символами*.

```
return0;          // НЕПРАВИЛЬНО, должно быть return 0;
return(0);        // ПРАВИЛЬНО, пробельный символ опущен
return (0);       // ПРАВИЛЬНО, используется пробельный символ
intmain();        // НЕПРАВИЛЬНО, пробельный символ опущен
int main()        // ПРАВИЛЬНО, пробельный символ опущен в скобках
int main ( )      // ПРАВИЛЬНО, пробельный символ используется в скобках
```

Операторы в языке C++

Программа, написанная на языке C++, представляет собой коллекцию функций, каждая из которых, в свою очередь, является коллекцией *операторов*.

```
// carrots.cpp -- программа по технологии производства пищевых продуктов
// использует и отображает переменную

#include <iostream>

int main()
{
     using namespace std;

     int carrots;              // объявление переменной целочисленного типа

     carrots = 25;             // присваивание значения переменной
     cout << "I have ";
     cout << carrots;          // отображение значения переменной
     cout << " carrots.";
     cout << endl;
     carrots = carrots - 1;    // изменение переменной
     cout << "Crunch, crunch. Now I have " << carrots << " carrots." << endl;
     return 0;
}
```

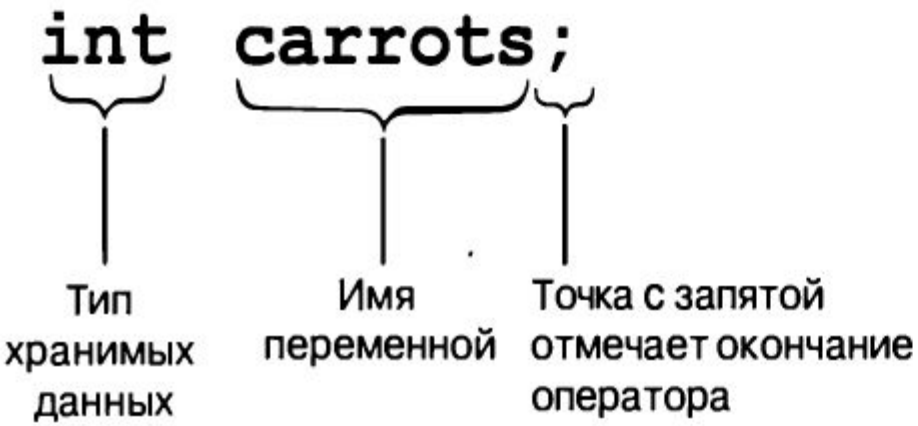
Результат работы программы:

```
I have 25 carrots.
Crunch, crunch. Now I have 24 carrots.
```

Операторы объявления и переменные

Оператор объявления идентифицирует тип памяти и предоставляет метку для ячейки.

```
int carrots;
```



<u>Совет</u>

В языке C++ принято объявлять переменную как можно ближе к той строке, в которой она впервые используется.

Операторы присваивания

Оператор присваивания присваивает значение ячейке памяти.

```
carrots = 25;
```

Символ = называется операцией присваивания.

```
int steinway;
int baldwin;
int yamaha;
yamaha = baldwin = steinway = 88;
```

Второй оператор присваивания демонстрирует возможность изменения значения переменной:

```
carrots = carrots - 1; // изменяет значение переменной
```

Объект cout

Объект cout может принимать переменную целочисленного типа:

```
cout << carrots;
```

Использование функции вывода printf() из языка C:

```
printf("Printing a string: %s\n", "25");
printf("Printing an integer: %d\n", 25);
```

Использование cin

```
// getinfo.cpp -- ввод и вывод
#include <iostream>
int main()
{
        using namespace std;
        int carrots;
        cout << "How many carrots do you have?" << endl;
        cin >> carrots; // ввод C++
        cout << "Here are two more. ";
        carrots = carrots + 2;
     // следующая строка выполняет конкатенацию вывода
        cout << "Now you have" << carrots << " carrots." << endl;
        return 0;
}
```

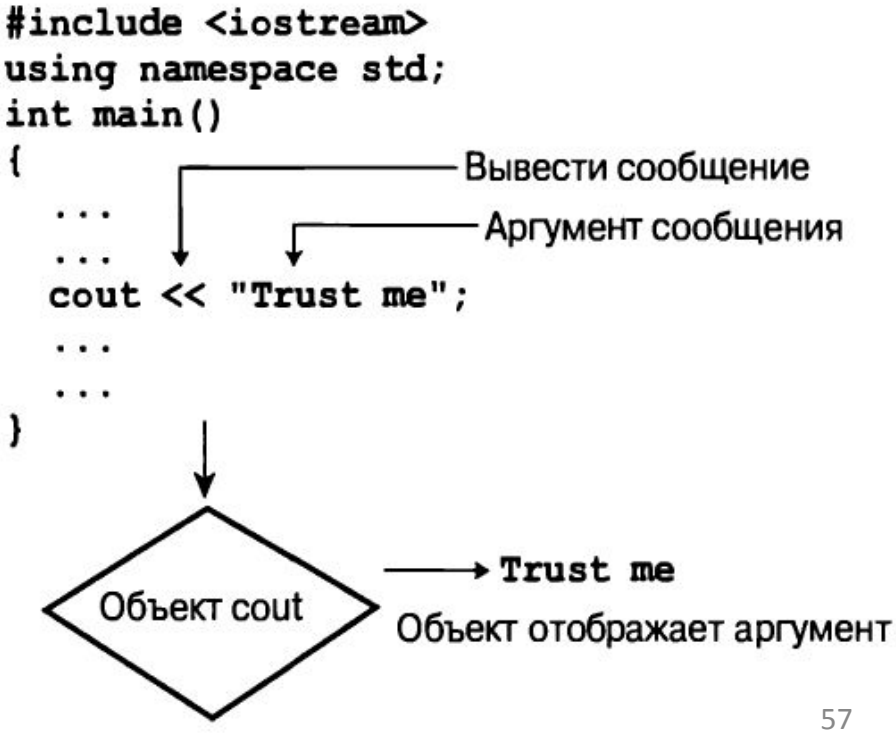
Результат работы программы:

```
How many carrots do you have?
12
Here are two more. Now you have 14 carrots.
```

cin и cout: признак класса

Класс — это тип данных, определяемый пользователем. Чтобы определить класс, вы описываете, какую разновидность информации он может хранить, и какие действия разрешает выполнять над этими данными.

- Объект cout создан для представления свойств класса ostream
- cin — это объект, созданный со свойствами класса istrearn, который тоже определен в iostream.
- Класс описывает все свойства типа данных, включая действия, которые могут над ним выполняться, а объект является сущностью, созданной в соответствии с этим описанием.



Как компилируются программы на C++?

Можно выделить 3 основных этапа

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.
- 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`. 3 – длина названия функции, i – int, d - double
- Аналогично функциям в линковке нуждаются глобальные переменные.

Этап 3: линковка (компоновка)

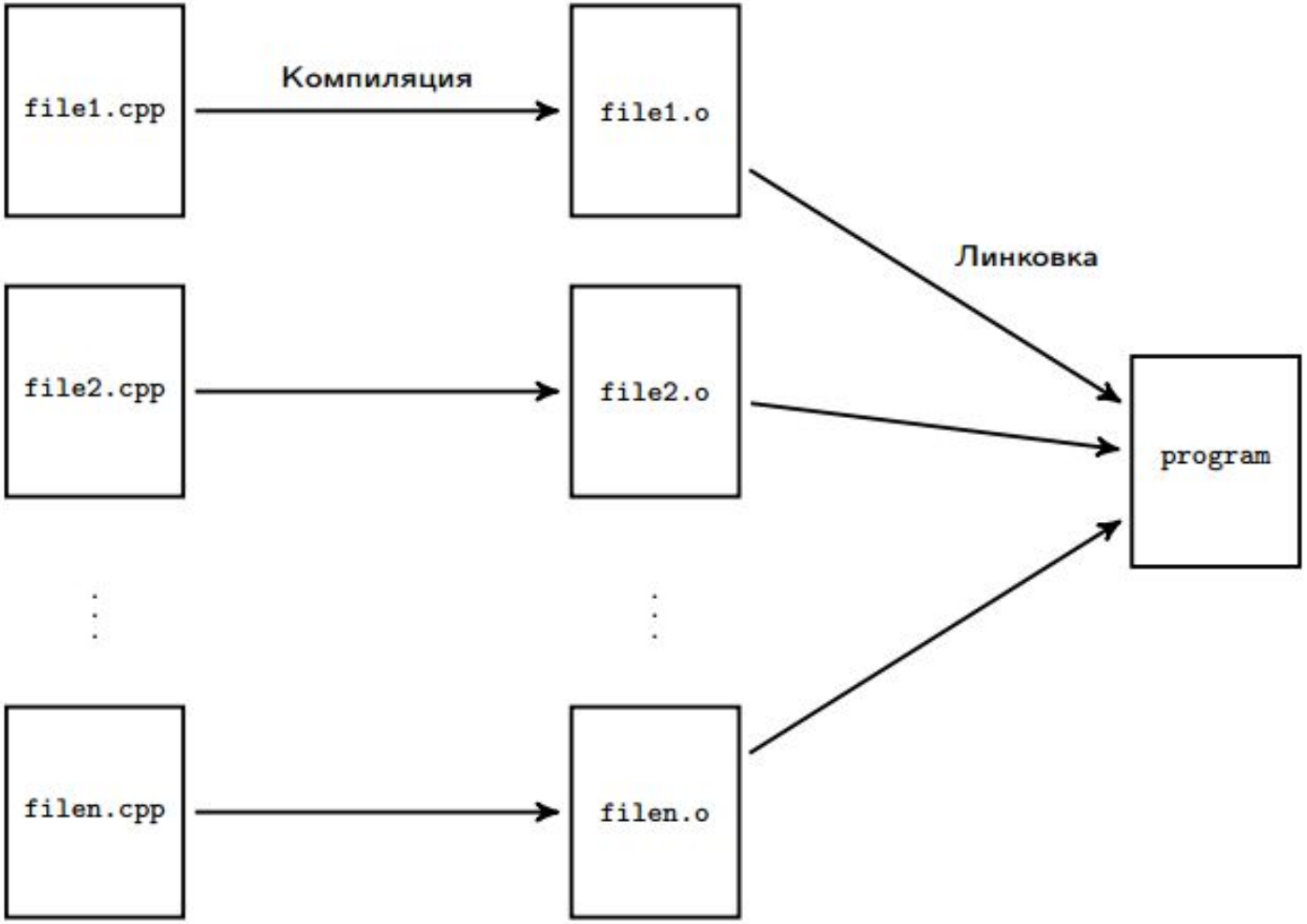
- *Точка входа* — функция, вызываемая при запуске программы. По умолчанию — это функция `main`:

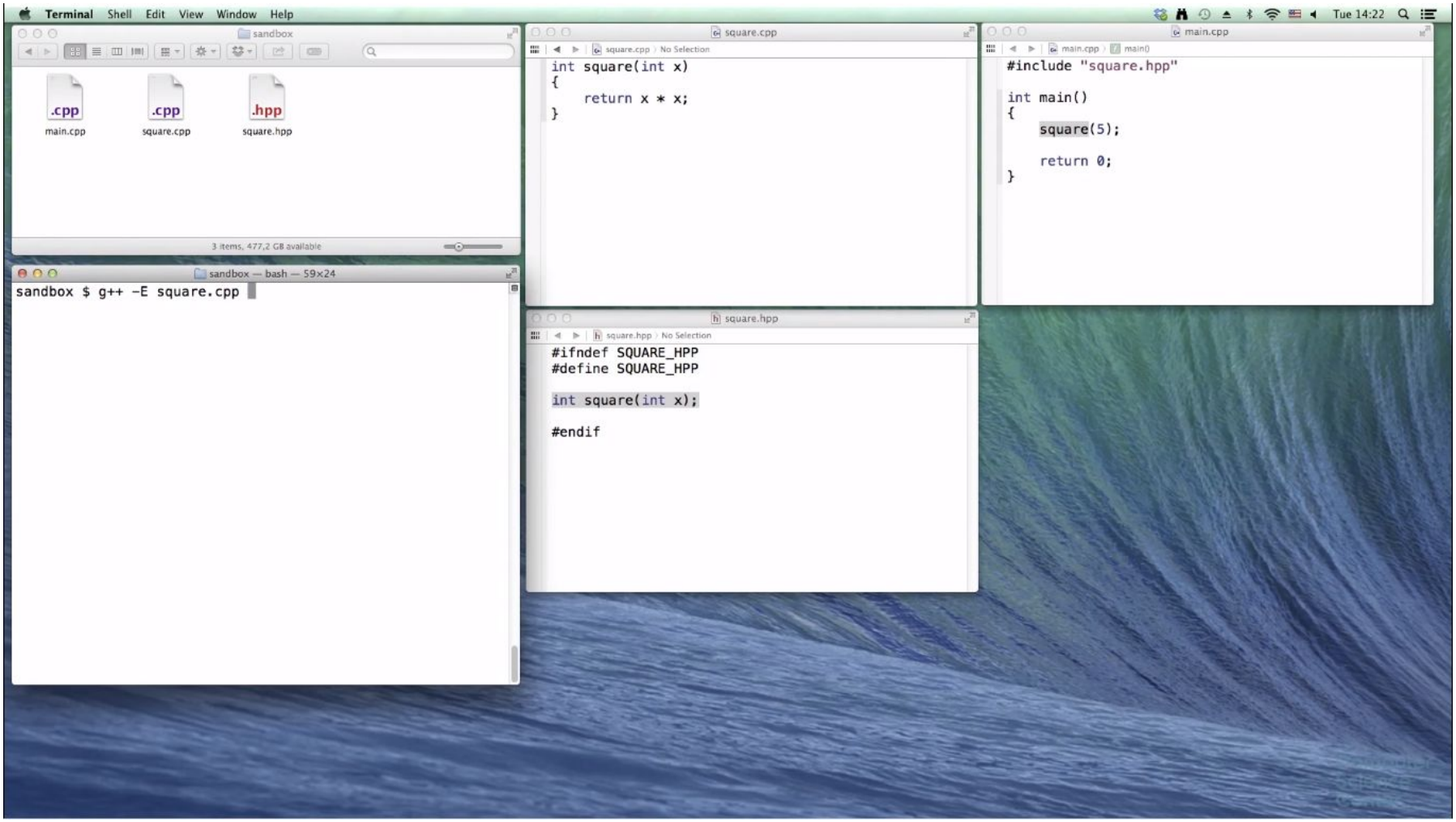
```
int main()
{
    return 0;
}
```

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


Terminal Shell Edit View Window Help
Tue 14:22
sandbox
.cpp
main.cpp
.cpp
square.cpp
.hpp
square.hpp
3 items, 477,2 GB available
sandbox — bash — 59×24
sandbox $ g++ -E square.cpp
square.cpp
square.cpp › No Selection
int square(int x)
{
return x * x;
}
main.cpp
main.cpp › main()
#include "square.hpp"
int main()
{
square(5);
return 0;
}
square.hpp
square.hpp › No Selection
#ifndef SQUARE_HPP
#define SQUARE_HPP
int square(int x);
#endif

Простейшая программа на C++

```
#include <iostream>

int main()
{
   std::cout << "Hello, World!\n";
   return 0;
}
```

Рекомендуемая литература

- Б. Керниган, Д. Ритчи "Язык программирования C". Понимание языка C
- Стенли Б. Липпман, Жози Лажойе, Барбара Э. Му "Язык программирования C++. Вводный курс".
- Стивен Прата "Язык программирования C++. Лекции и упражнения"
- Герберт Шилдт "Язык программирования C++. Вводный курс".
- Скотт Майерс "Эффективное использование C++".
- Герб Саттер Андрей Александреску, "Стандарты программирования на C++".
- Герб Саттер "Решение сложных задач на C++".
- Герб Саттер "Новые сложные задачи на C++".
- А. Александреску "Современное проектирование на C++".
- Бьерн Страуструп "Дизайн и эволюция языка C++".