Конспект лекций Мещерина

Содержание

1. Введение

Лекция 1

О лекторе

Лектора зовут Мещерин Илья Семирович.

VK: vk.com/mesyarik

TG: @mesyarik

1.1. Общие слова о языке и исторические заметки

Почему изучается именно этот язык программирования и что он даст? На сайте www.tiobe.com в котором собрана информация о популярности языков программирования C++ занимает достаточно высокое место.

Существуют языки **общего** и **специального** назначений, например на C++, C#, Python, Java можно писать почти все что угодно, а на JavaScript — нет.

По этому курсу будет довольно много лекций, потому что C++ довольно большой язык. Вместе с изучением C++ будет затронут и его прородитель — C.

Почти всегда то, что написано на C++ будет работать и на C. Также, изучая C++ будет намного проще изучать Java и C#, потому

что эти языки синтаксически похожи.

Язык C++ — очень сложен, освоив его изучать остальное будет проще. Сложность освоения C++ связана со многими вещами, как минимум он довольно низкоуровневый. Придется следить за памятью, есть возможность общаться с ОС на прямую и т.д.

В Java и C# мы защищены от низкоуровневых инструкций, что делают их проще и безопаснее. Однако в этом есть и плюсы и минусы. Из-за незащищенности и низкоуровневости код C++ может работать в разы быстрее своих аналогов. Исходя из этого C++ используется там, где нужен высокопроизводительный код.

Примеров известных сервисов, которые написаны с использованием кода на C++ довольно много.

Яндекс: Поиск, Карты, Такси, Почта, Диск.

Google: Search, Youtube.

Телеграмм.

Также на C++ написано множество игр, например WoW.

Не стоит и забывать про ОС, даже Windows написан с использованием C++.

C++ используется и в науке.

C++ создал Бьерн Страуструп (Bjarne Stroustrup). В курсе его будем называть **создателем**.

У C++ много версий. Мы будем пользоваться стандартом C++ 17.

- C++98
- C++03

  ————
- C++11 (C++0x)
- C++14 (C++1y)
- C++17 (C++1z)

Между версиями языка не зря проведена черта. Они очень сильно отличаются. У каждой версии есть **стандарт**. Тонны страниц текста с мелким шрифтом, исчерпывающе описывающим версию языка. Из-за его объема и формальности не рекомендуется читать и вникать в него полностью, а лишь обращаться к нему и читать по диагнали в неоднозначных моментах.

Рекомендуется пользоваться сайтом www.cppreference.com. Если требуется что-то узнать о языке, то первым делом лучше лезть туда.

Не стоит брезговать сайтом www.stackoverflow.com, на котором почти всегда будут ответы на возникающие вопросы.

Рекомендованная литература, автора Скота Мейерса:

- Эффективное использование C++.
- Наиболее эффективное использование C++.
- Эффективный и современный C++.

1.2. Знакомство с компилятором и первая программа

Добро пожаловать в терминал Linux! В курсе рекомендуется пользоваться именно этой операционной системой.

Напишем первую программму.

```
#include <iostream>

int main() {
    int x;
    std::cin >> x;
    std::cout << x + 5 << '\n';
}
```

В программе на C++ обязательно должна быть такая функция как **main**. Здесь мы пишем программу от которой ожидаем простые действия, а именно ввести число, увеличить его на пять и вывести.

Чтобы делать ввод-вывод необходимо подключить **заголовочный файл** — iostream.

Написав строку

```
#include <iostream>
```

мы подключаем заголовочный файл и получаем возможность работать со стандартными потоками ввода-вывода.

Для того чтобы ввести переменную, необходимо ее **объявить**.

```
    int x;
```

Чтобы ввести переменную из стандартного потока ввода пишем

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

а чтобы вывести число в стандартный поток вывода из этой переменной увеличенное на 5 пишем

```
    std::cout << x + 5 << '\n';
```

Здесь мы также добавляем к выводу **символ перевода строки** — '\n'.

Чтобы перевести строку можно написать и

```
    std::cout << x + 5 << std::endl;
```

однако есть разница. Во втором случае мы также отчищаем буффер вывода.

Файлы исходного кода обычно выглядят как *.cpp.

Перейдем к компиляции. Для того чтобы получить исполняемую программу необходимо перевести исходный код в машинный, этот процесс называется **компиляцией**, а наличие этого процесса отличает компилируемые языки программирования от интерпретируемых.

Для того чтобы скомпилировать исходный код необходимо вызвать специальную программу — **компилятор**. У C++ есть несколько компиляторов. Наиболее известным из них является gcc — Gnu Compiler Collection. Для того чтобы его вызвать именно для языка C++ необходимо писать g++.

Также есть такие компиляторы, как clang++ и msvc. Компилятор msvc разрабатывается компанией Microsoft и не рекомендован к использованию, поскольку многие вещи от туда не всегда соответсвуют стандарту.

Для отладки программы можно пользоваться программой дебагером, например gdb.

Лекция 2

1.3. Виды ошибок и неопределенное поведение (UB)

C++ — компилируемый язык. Поговорим о том, какие ошибки бывают ошибки при компиляции и после нее.

Программа бывает некорректна по многим причинам. Рассмотрим ошибку компиляции **CE** Compilation Error.

Ошибки компиляции бывают разные. Например, если где-то забыть поставить точку с запятой, то это будет **синтаксическая ошибка**. Она случается когда компилятор не смог распарсить то что написано из-за синтаксиса. Ошибкой компиляции также считается использование переменной или функции, которые не были объявлены.

При написании

```
#include <iostream>
```

Мы вставляем в исходный код содержание заголовочного файла, в данном случае iostream, в котором содержатся объявления функций std::cin и std::cout.

И если забыть подключить заголовочный файл и воспользоваться функцией, то это будет CE.

Ошибкой будет и повторное объявление переменной в одной и той же области видимости.

```
int main() {
    int x;
    int x;
}
```

Поэтому подобный код тоже будет выдавать ошибку CE при попытке компиляции.

Также при обычных обстоятельствах недопустимо пользоваться переменной/функцией до объявления.

```
#include <iostream>

int main() {
    std::cout << x << '\n';
    int x;
}
```

Выдаст CE при попытке компиляции.

В целом ошибки компиляции можно разбить на три группы:

- Лексические ошибки
- Синтаксические ошибки
- Семантические ошибки

Компиляция это очень сложный процесс, который состоит из разных стадий, поговорим о трех. Так, в начале происходит лексический разбор, потом синтаксический, а потом компилятор разбирается в семантике того что написано.

Разберем поподробнее. При лексическом разборе компилятор пытается разбить код на осмысленные последовательности символов — **токены**. Пример:

```
    std::cout << x;
```

Этот код будет разбиваться на токены следущим образом:

$$(std)(::)(cout)(<<)(x)(;).$$

Здесь круглыми скобками обозначены токены, на которые разберется код при лексическом разборе.

Возникает вопрос, почему компилятор определил («) а не например как два знака меньше (<)(<)? На самом деле действует следущее правило. Компилятор будет идти слева направо и пока это осмысленно он будет добавлять символы к токену. Именно поэтому не получится разбиения (std:), поскольку это неосмыслленно, а вот (std) — осмысленно.

Если компилятору не удалось разбить код на токены на этапе лексического разбора, то такая ошибка будет называться **лексической**.

```
#include <iostream>

int main() {

    \\\\\;

    int x;
    std::cin >> x;
    std::cout << x + 5;
}
```

При компиляции кода можно увидеть следущее, рисунок 1.

Здесь stray можно перевести как ”заблудший” или ”заплутавший”. Компилятор не смог это как-то интерпетировать.

Приведем более подробные примеры синтаксических ошибок.

```
#include <iostream>

int main() {
    int x
    std::cin >> x;
    std::cout << x + 5;
}
```

На рисунке 2 написано, что компилятор ожидал инициализацию перед std. Иначе говоря он парсит строки 5 и 6 как одно выражение, так как в строке пять нет ’;’, однако после объявления переменной компилятор ожидает либо ничего, либо ее инициализацию, а так как дальнейшее инициализацией никак не является, то это ошибка. Так как эта ошибка возникла из-за отсуствия ’;’ и была обнаружена на этапе синтаксического разбора она считается синтаксической.

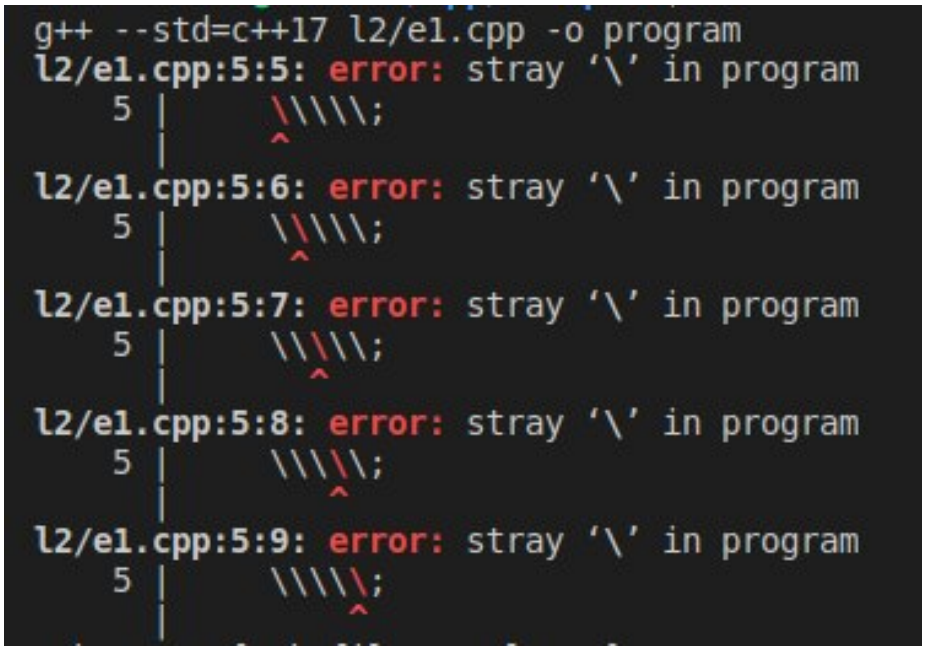

Рис. 1: Пример лексической ошибки.

Далее компиляция не прервалась и было обнаружено, что переменная x не была объявлена, поскольку предыдущее выражение ошибочно. Стоит отметить, что компилятор не сказал об использовании необъявленной переменной в 5-ой строке после std::cin, так как уже отловил в этом выражении синтаксическую ошибку. Выражения не всегда рассматриваются на предмет ошибок до конца, да и здесь это не требуется.

```
#include <iostream>

int main() {
    int x;
    std::cin >> x;
    std::cout << x + 5;
    x + 5 + ;
}
```

Поскольку '+' здесь используется как бинарный оператор, а после стоит ';', то перед ';' ожидается некое primary-expression, о котором будет рассказано позже.

Компилятор старается не завершать компеляцию, чтобы выдать как можно больше ошибок сразу, однако у него не всегда это

```
g++ -std=c++17 l2/e2.cpp -o program
l2/e2.cpp: In function ‘int main()’:
l2/e2.cpp:5:5: error: expected initializer before ‘std’
    5 |     std::cin >> x;
      |     ^~~
l2/e2.cpp:6:18: error: ‘x’ was not declared in this scope
    6 |     std::cout << x + 5;
      |                  ^
```

Рис. 2: Пример синтаксической ошибки 1.

```
g++ -std=c++17 l2/e3.cpp -o program
l2/e3.cpp: In function ‘int main()’:
l2/e3.cpp:7:13: error: expected primary-expression before ‘;’ token
    7 |     x + 5 + ;
      |             ^
```

Рис. 3: Пример синтаксической ошибки 2.

получается. Ошибки, при которых компиляция останавливается, называются **фатальными**.

```
#include <iostream>
#include <aaaaaaaa>

int main() {

}
```

```
g++ -std=c++17 l2/e4.cpp -o program
l2/e4.cpp:2:10: fatal error: aaaaaaaa: No such file or directory
    2 | #include <aaaaaaaa>
      |          ^~~~~~~~~~
compilation terminated.
```

Рис. 4: Пример синтаксической ошибки 3. Фатальная ошибка.

Примером фатальной ошибки будет являться подключение файла, которого не существует.

Можно привести такой неформальный пример синтаксической ошибки

Я пошел лес грибы.

Вроде и все понятно, но синтаксически неверно.

А вот какой пример можно привести для семантической ошибки

Я ем стол.

С точки зрения синтаксиса здесь все верно, однако есть стол это как-то странно. Иначе говоря мы понимаем, какую пищу может есть человек, а какую нет. Человек не может есть стол. Подобная ошибка называется семантической.

Другими словами если мы пытаемся совершить что-то невозможное, то это **семантическая ошибка**.

```cpp
#include <iostream>

int main() {
    int x;
    std::cin >> x;
    std::cout << x.size();
}
```

```
g++ -std=c++17 l2/e5.cpp -o program
l2/e5.cpp: In function 'int main()':
l2/e5.cpp:6:20: error: request for member 'size' in 'x', which is of non-class type 'int'
    6 |     std::cout << x.size();
      |                    ^~~~
```

Рис. 5: Пример семантической ошибки 1.

Мы пытаемся применить метод .size() к переменной целочисленного типа, однако это невозможно, поскольку переменная целочисленного типа не обладает методом .size(). О методах мы узнаем позже.

```cpp
int main() {
    int x;
    int a;
    int b;
    x++ = a + b;
}
```

```
g++ -std=c++17 l2/e6.cpp -o program
l2/e6.cpp: In function 'int main()':
l2/e6.cpp:5:6: error: lvalue required as left operand of assignment
    5 |     x++ = a + b;
      |     ~^~
```

Рис. 6: Пример семантической ошибки 2.

Это тоже семантическая ошибка, поскольку тому, что стоит слева от '=' невозможно присвоить что-либо, так как то что слева не

является lvalue. О том что такое lvalue мы тоже поговорим позднее.

Заметим, что на рисунке 2 также показана семантическая ошибка, так как мы пытаемся использовать необъявленную переменную.

```
int main() {
    int x;
    int x;
}
```

```
g++ -std=c++17 l2/e7.cpp -o program
l2/e7.cpp: In function 'int main()':
l2/e7.cpp:3:9: error: redeclaration of 'int x'
    3 |     int x;
      |         ^
l2/e7.cpp:2:9: note: 'int x' previously declared here
    2 |     int x;
      |         ^
```

Рис. 7: Пример семантической ошибки 3.

Вспомним о повторном объявлении переменной. Это тоже семантическая ошибка, рисунок 7.

Также иногда семантическая ошибка возникает при неоднозначном обращении, рисунок 8.

```
#include <iostream>

namespace N {
    int x;
}

using namespace N;

int x = 0;

int main() {
    std::cin >> x;
    std::cout << x + 5;
}
```

Здесь переменная 'x' объявлена в пространстве имен 'N', а далее, строкой

```
using namespace N;
```

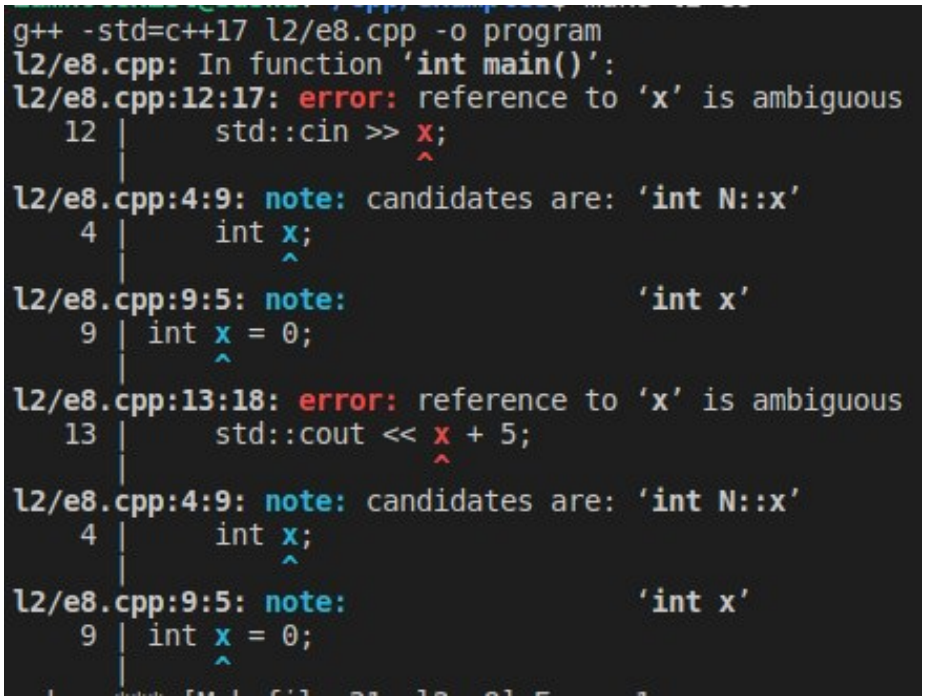

Рис. 8: Пример семантической ошибки 4.

сливается с глобальной областью видимости.

Так как переменные объявлены в разных областях видимости здесь нет ошибки повторного объявления, значит это две разные переменные. Однако после сливания не понятно, к какой именно переменной идет обращение.

Помимо ошибок компиляции существуют ошибки, которые возникают во время исполнения программы. Они называются **Runtime error** (**RE**).