Курс лекций:

Язык программирования

C++Лекция 1: О языке C++, основные базовые конструкции.

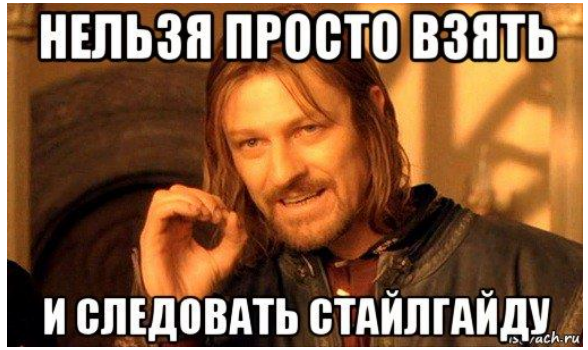
Преподаватель: Пысин Максим Дмитриевич, Краснов Дмитрий Олегович,
аспиранты кафедры ИКТ.


ИМЕННО ПО ЭТОЙ ПРИЧИНЕ
СПАТЬ НА ЛЕКЦИЯХ ОПАСНО

Соглашение об именовании

Категория	Стиль	Структура
Глобальные константы	UPPER_CASE	Существительное (иногда с прилагательными)
Переменные и параметры фунции	lower_snake_case	Существительное (иногда с прилагательными)
Параметры функций	lower_snake_case	Существительное (иногда с прилагательными)
Функции	lowerCamelCase	Глагол (функция - это действие)
Имена типов (структур и т.д.)	UpperCamelCase	Существительное (иногда с прилагательными)

Можно использовать глобальные константы, нельзя использовать глобальные переменные.
Не используйте для объявления констант директивы препроцессора.
Не нужно объявлять переменную в начале функции, если её можно объявить при первом вычислении.
Но иногда объявлять заранее всё-таки приходится. В этом случае инициализируйте переменную
Не стоит писать в коде числа непонятного назначения, чтобы читающий код не задавался вопросами “почему 4?”, “почему 37?”, “что будет, если 36 заменить на 50 в этой строке?”. Используйте именованные константы



https://habr.com/ru/post/172091/ https://github.com/MUCTR-IKT-CPP/lab_readmes/blob/master/NAMING_CONVENTION.md

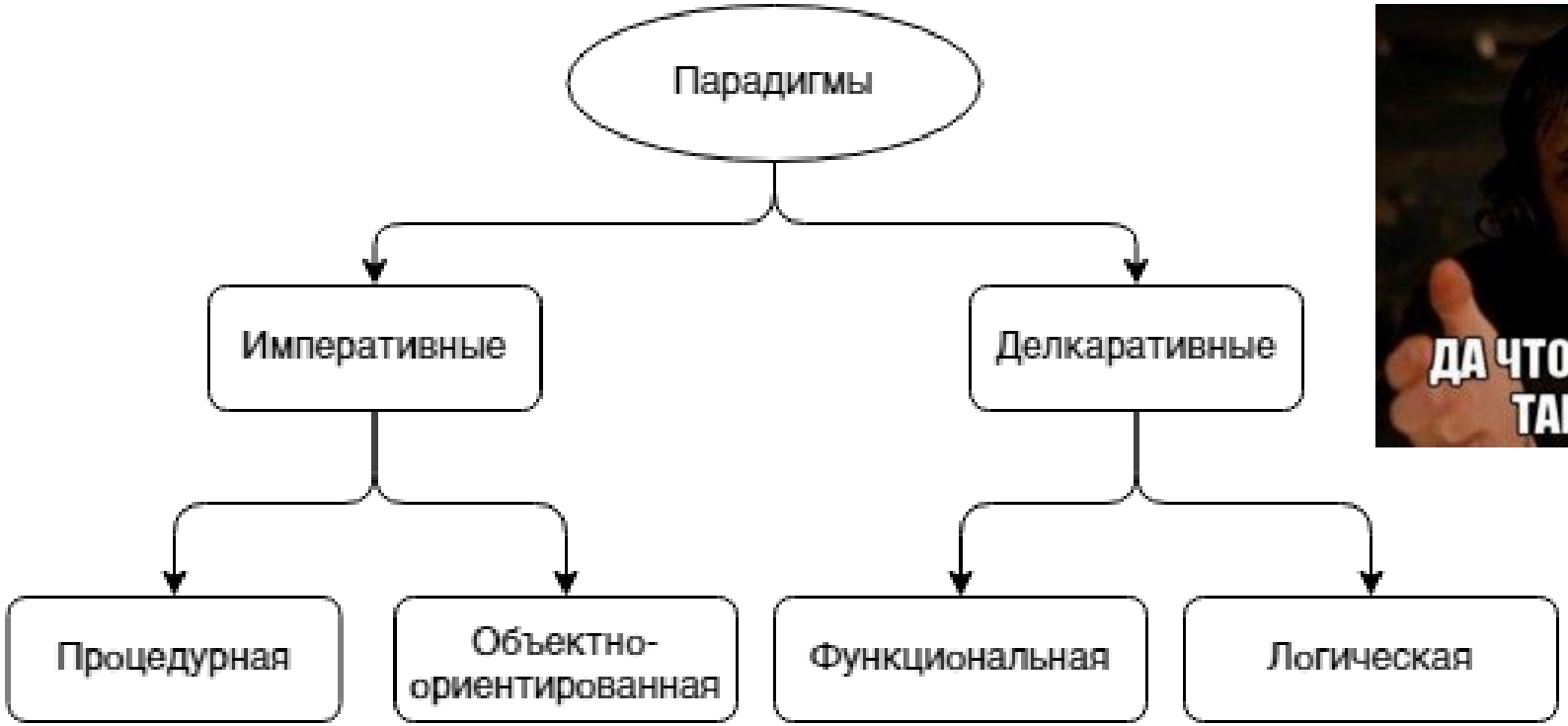
О языке

- **Язык C++ является языком высокого уровня,** язык программирования, разработанный для быстроты и удобства использования программистом.
- **Язык C++ постоянно развивается**. На данный момент в ходу имеются следующие стандарты: 98, 11, 14, 17, 20
- **Язык C++ является кросплатформенным**, однако программы написанные на нем кросплатформенными не являются.
- **Язык имеет обратную совместимость с C**, так как он позиционировал себя наследником и продолжателем идей языка C, т.е. программы написанные на языке C и в его стиле будут прекрасно работать на C++
- **Язык C++ является компилируемым языком**.
- **Язык поддерживает несколько парадигм программирования**, но основная Объектно-ориентированное программирование.

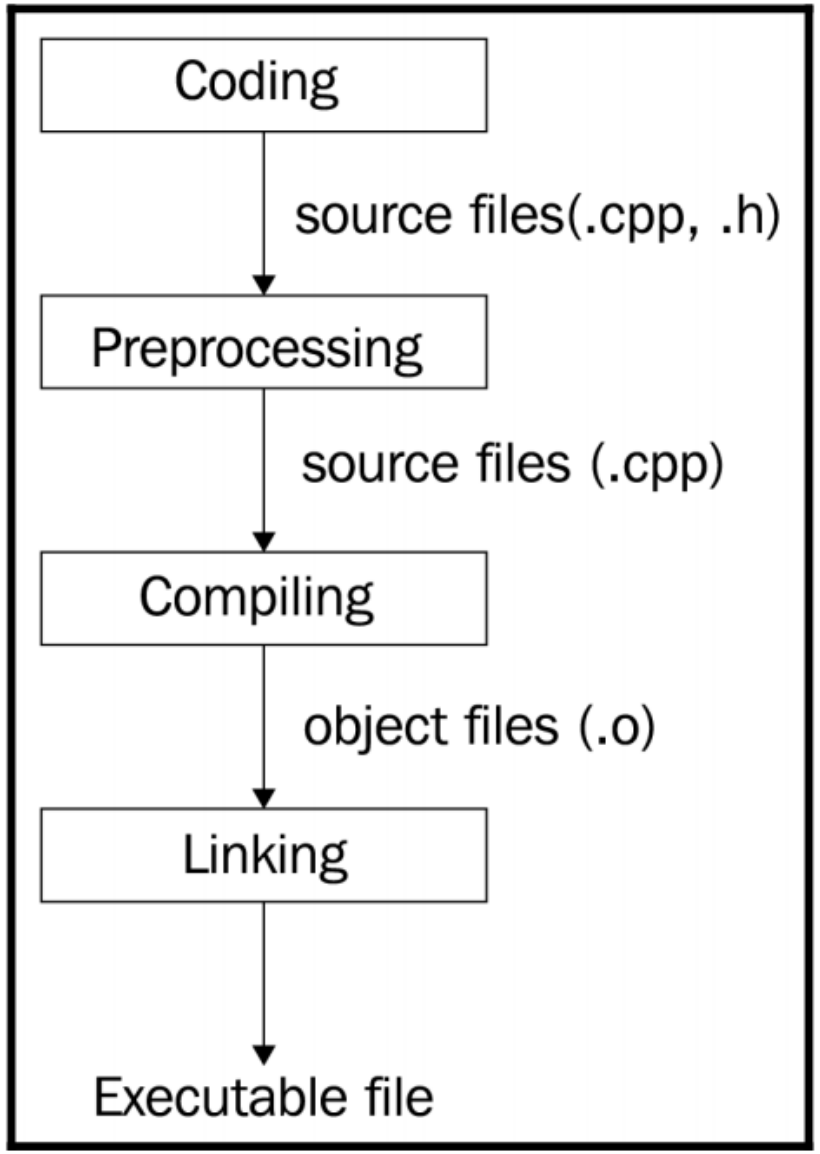
О языке: парадигмы программирования

Парадигма программирования — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером

Парадигма программирования как исходная концептуальная схема постановки проблем и их решения является инструментом грамматического описания фактов, событий, явлений и процессов, возможно, не существующих одновременно, но интуитивно объединяемых в общее понятие.

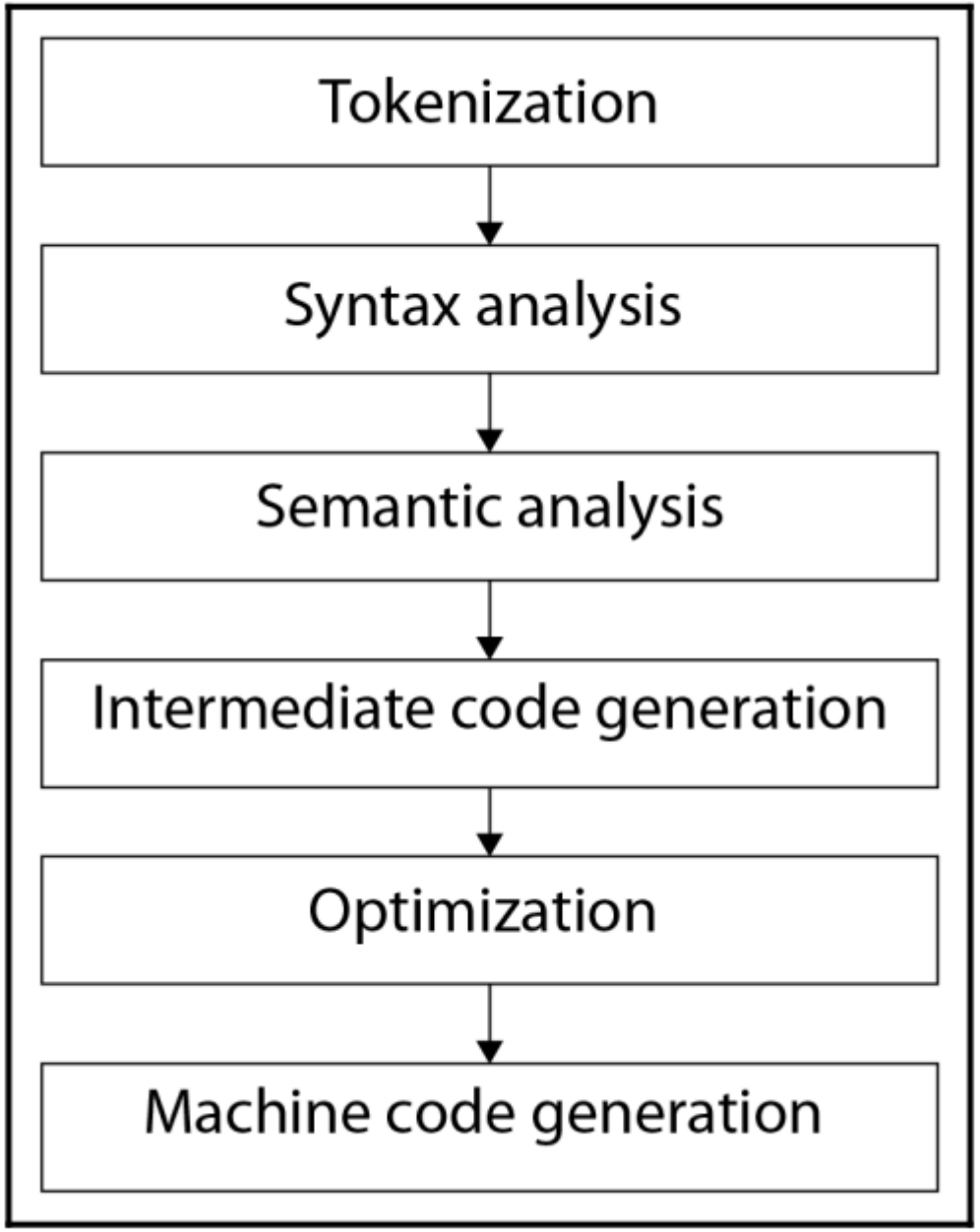


Процесс появления программы



1. Процесс разработки
2. Процесс препроцессора, именно в этот момент обрабатываются все директивы препроцессора
3. Это этап компиляции, именно в этот момент наша программа из написанного для людьми и для людей превращается в то что уже может интерпретировать компьютер.
4. Процесс связывания, он нужен для того, что бы мы могли разбивать наши программы на большое количество подмодулей и использовать функции из одного подмодуля в другой.

Процесс компиляции программы



1. Токенизация, этоп процесс разбиения компилируемоего кода на минимальные значимые элементы
2. Синтаксический анализ, проверяет все ли правильно в тех токенах что мы написали с точки зрения синтаксиса
3. Семантический анализ, в свою очередь занимается абсолютно другим, он проверяет смысл того что мы написали
4. Генерация промежуточного кода, наш код написанный на **C++** перестает быть **C++**. Он превращается в некий промежуточный вариант, условно похожий на **C**.
5. Оптимизация, на этом этапе компилятор пытается оптимизировать код который мы написали
6. Генерация машинного кода, именно в этот момент мы и получаем наш машинный код.

! Важно, компилируется за раз не вся программа а лишь одна единица трансляции

С чего начинаются программы на С++

Код наших программ пишется в файлах с расширением .cpp и .h.

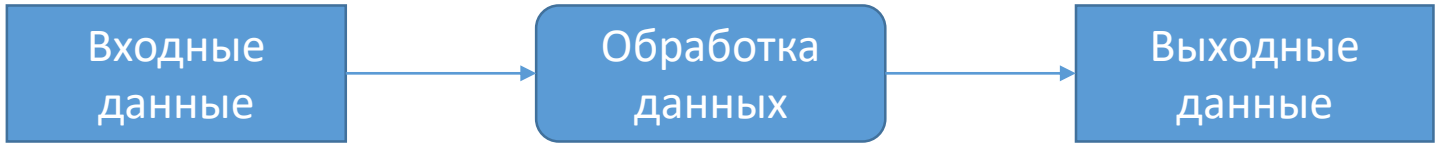
Любая программа на С++ начинается с функции:

```
int main (){
    cout << "Hellow World" << endl;
    return 0;
}
```

Функция должна именно называться словом main и ни как по другому, все следующие действия которые мы хотим получить от нашей программы должны быть написаны или вызваны внутри функции main, а сама функция должна обязательно завершаться ключевым словом return которое в случае успешного исполнения нашей программы должно возвращать 0, любой результат отличный от 0 будет восприниматься операционной системой как ошибка и будет взят как ее код.

Переменные в C++

Суть работы любой программы в упрощенном виде сводиться к схеме:



Как видно из схемы, в процессе работы фигурируют какие либо банные, а соответственно программе нужно их где то хранить, для этого и нужны переменные.

Декларирование - это процесс резервирования имени под переменную.

```
bool b = false;
int n = 10;
double df = 13.24;
char c = 'c';
```

Инициализация - это процесс присвоения переменной своего первого, начального значения.

В каждой приведенной строке происходит объявление и инициализация переменных
Каждая строка имеет определённую структуру, которую принято называть сигнатурой, таким образом сигнатура объявления переменной:

<type> <name> = <value>;

Наименование переменной может быть практически любым, не допускается совпадение с ключевыми словами языка и оно не может начинаться с цифр.

Типы данных в C++

- Встроенные типы
  к ним относятся все те типы что есть в языке сами по себе, и по сути, существуют шире, в самом оборудовании.
- Пользовательские типы
  Это те типы которые пользователь комбинируя каким либо образом встроенные типы и описывая взаимодействия между ними создает сам

- **bool**: логический тип. **true/false**
- **char**: представляет один символ в кодировке ASCII. Занимает в памяти 1 байт (8 бит). **wchar_t**: UTF-8. **char16_t**: UTF-16. **char32_t**: UTF-32.
- **int**: представляет целое число. Занимает от 2 до 4 байт.
- **float**: представляет вещественное число ординарной точности с плавающей точкой в диапазоне +/- 3.4E-38 до 3.4E+38. В памяти занимает 4 байта (32 бита)
- **double**: представляет вещественное число двойной точности с плавающей точкой в диапазоне +/- 1.7E-308 до 1.7E+308. В памяти занимает 8 байт (64 бита)

Типы char, и int имеют приставки **unsigned** означает что тип не хранит отрицательные числа и **singed**, означает что тип хранит отрицательные числа.
Тип **int** имеет приставки **short** уменьшает диапазон и память в 2 раза и **long** увеличивает диапазон и память в 2 раза, так же может применяться к **double**

Литералы и Констарны в C++

Литерал — это элемент программы, который непосредственно представляет значение. Литералы можно использовать во многих контекстах, но наиболее часто они используются для инициализации именованных переменных и для передачи аргументов в функции.

```
42; // Целочисленный типа int
108.87; // С плавающей запятой типа double
12.; // С плавающей запятой типа double
14.2f; // С плавающей запятой типа float
15.1L; // С плавающей запятой типа long double
16u; // Целочисленный типа unsigned int
15l; // Целочисленный типа long int
true; // Логический
's'; // Символьный
"str"; // Строковый
```

Константы в C++

Константы — это переменные значение которых нельзя изменять. Константы бывают 2х видов:
const – это обычные константы, они же константы времени исполнения, их значение создается и присваивается в процессе выполнения, а поэтому их значение можно рассчитывать из переменных
constexpr – это более новый вид констант, появившийся в стандарте C++11 и их называют константы времени компиляции, их значения вычисляются и присваиваются еще при компиляции программы компилятором, а поэтому в них нельзя записывать переменные.

```
int n = 5;
const bool B = false;
const int N = 10;
const double D = N / 2.1;
constexpr int N_2 = N * 2;
const char C = 'c';
```

Структурный подход и область видимости в C++

Структурное программирование — парадигма программирования, в основе которой лежит представление программы в виде иерархической структуры блоков

В C++ структурой кода или, что более распространено блоком кода называют любой код заключенный в фигурные скобки.

К структурам кода в C++ относятся условия, циклы, функции, классы.

```
{
    // Блок кода
    {
        // Вложенный блок кода
    }
    {
        // Вложенный паралельный блок
        кода
    }
}
```

Структурный подход тесно связан с понятием области видимости переменных, это понятие говорит о том как блоки кода обмениваются информацией между друг другом. Концептуально это описывается так:

- Переменные объявленные в одном блоке кода всегда видны внутри него, во всех вложенных в него структурах.
- С другой стороны все что объявлено в параллельных блоках кода не будет видно в этих блоках.
- Так же то что объявлено в дочернем блоке не видно в родительском блоке кода.

```
Дед
    \
    Отец <- отец знает то же что знает дед, но не знает что
    знают дети
        \
        Сын <- сын знает то что знает отец и дед, но не знает
        того что знает брат
        \
        Брат
```

Условия в C++

В C++ есть два разные условные конструкции, это конструкция if else и конструкция switch.

Конструкция if else имеет следующий вид:

```
if(<условие>){
    // Блок кода если условие истинно
}else if(<условие 2>){
    // Блок кода если первое условие ложно а второе условие
    итенно
}else{
    // Блок кода если все условия ложны
}
```

Отличаются конструкции между собой тем, что в конструкции if else в рамках проверки проверяется логическое выражение на истинность или ложность, а в конструкции switch происходит проверка не логического выражения, а значения той или иной переменной на соответчике значениям указываемых в блоках case

Конструкция switch case имеет следующий вид:

```
switch(<проверяемая переменная>){
    case <первое значение>:
        // блок кода если переменная имеет значение 1
        break;
    case <второе значение>:
        // блок кода если переменная имеет значение 2
        break;
    case <третье значение>:
        // блок кода если переменная имеет значение 3, не
        завершается и выполняет следующий по порядку блок, до
        тех пор пока не встретит break
    case <четвертое значение>:
        // блок кода если переменная имеет значение 4
        break;
    default:
        // блок кода если переменная не имеет ни одного из
        представленных значений.
        break;
}
```

Циклы в C++

В C++ есть два разные конструкции циклов, это конструкция for и конструкция while/do while.

Конструкция for

```
for(
    <тип> <название счетчика> = <значение>;
    <условие на сравнение счетчика>;
    <изменение счетчика>
){
    // Блок кода
}
```

```
for(int i = 0; i < 10; i++){
    cout << i << endl;
}
```

Конструкция while/do while

```
while(<условие>){
    //код, пока условие истинно будет повторно выполняться
}
do{
    // код, важно помнить что в таком цикле обязательно
    выполниться хотя бы 1 раз код
}while(<условие>);
```

```
while(p < 10){
    p += h;
    cout << p << endl;
}
do{
    p -= h;
    cout << p << endl;
}while(p > 0);
```

Разница в конструкциях обусловлена тем, что цикл for обычно выполняется заданное количество раз, и его следует использовать тогда когда известно колличество необходиых итераций(можеть быть вычисляемым), циклы while/do while предназначены для тех случаев когда количество итераций неизвестно, и цикл выполняется до тех пор пока условие выполянется.

Функции в C++

Функция это набор операций с данными которые нам необходимо выполнять множество раз(не обязательно)

- Любая функция имеет тип, также, как и любая переменная.
- Функция может возвращать значение, тип которого в большинстве случаев аналогично типу самой функции.
- Если функция не возвращает никакого значения, то она должна иметь тип **void** (такие функции иногда называют процедурами)
- При объявлении функции, после ее типа должно находиться имя функции и две круглые скобки — открывающая и закрывающая, внутри которых могут находиться один или несколько аргументов функции, которых также может не быть вообще.
- после списка аргументов функции ставится открывающая фигурная скобка, после которой находится само тело функции.
- В конце тела функции обязательно ставится закрывающая фигурная скобка.

```
<тип> <название функции>(<тип> <название параметра>, ...){
    // код функции
    return <возвращаемое значение тип которого соответствует
    типу написанному перед функцией>;
}
```

```
<тип> <название функции>(<параметры функции>);
```


СПАСИБО
ЗА ВНИМАНИЕ
© Sony Pictures
risovach.ru