

# Язык программирования C++



## О языке, основные базовые конструкции.

Преподаватели:

Пысин Максим Дмитриевич, ассистент кафедры ИКТ

Краснов Дмитрий Олегович, аспирант кафедры ИКТ

Лобанов Алексей Владимирович, аспирант кафедры ИКТ

Крашенинников Роман Сергеевич, аспирант кафедры ИКТ

**ИМЕННО ПО ЭТОЙ ПРИЧИНЕ**



**СПАТЬ НА ЛЕКЦИЯХ ОПАСНО**

# Соглашение об именовании



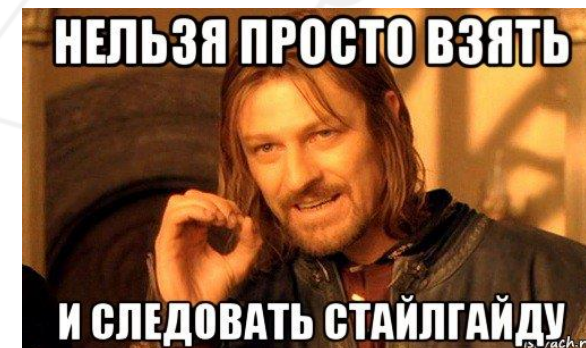
Категория	Стиль	Структура
Глобальные константы	UPPER_CASE	Существительное (иногда с прилагательными)
Переменные и параметры функции	lower_snake_case	Существительное (иногда с прилагательными)
Параметры функций	lower_snake_case	Существительное (иногда с прилагательными)
Функции	lowerCamelCase	Глагол (функция - 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



# 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 кроссплатформенным**, однако программы написанные на нем кроссплатформенными 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.

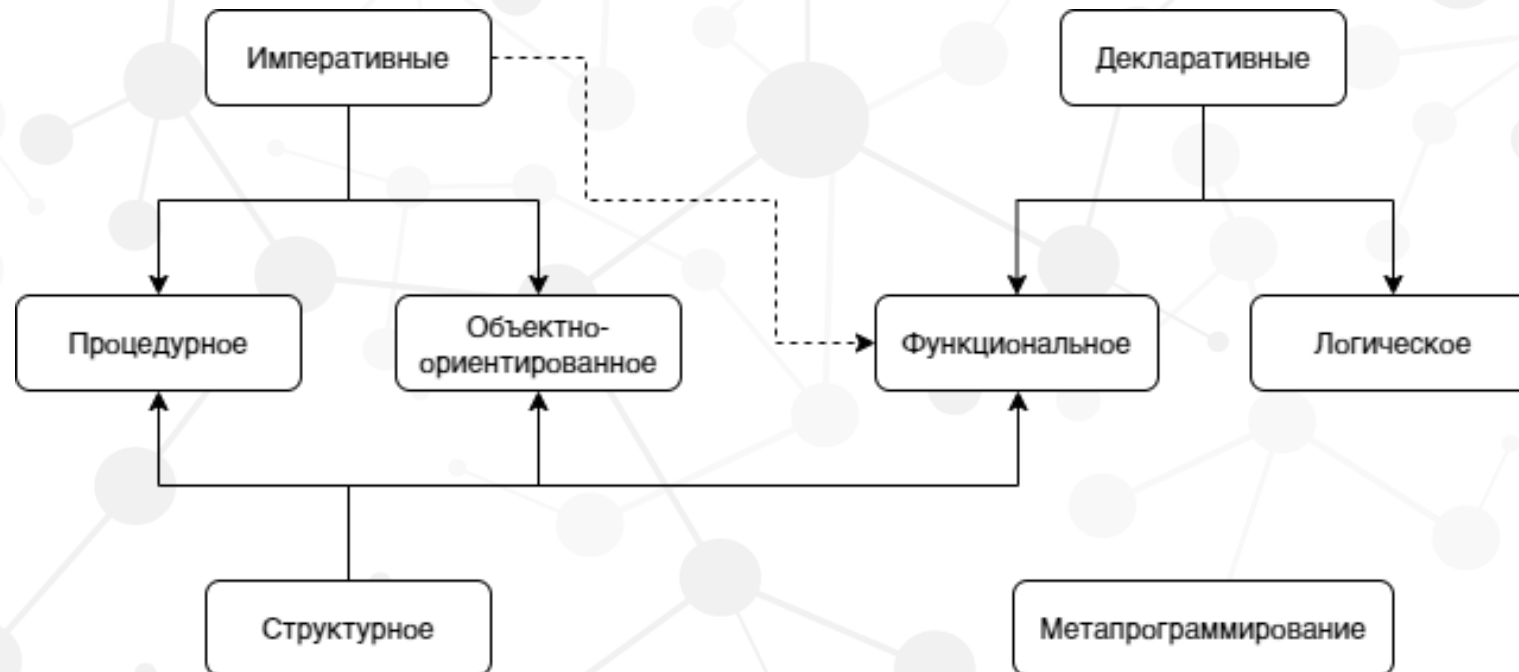


# Парадигмы программирования

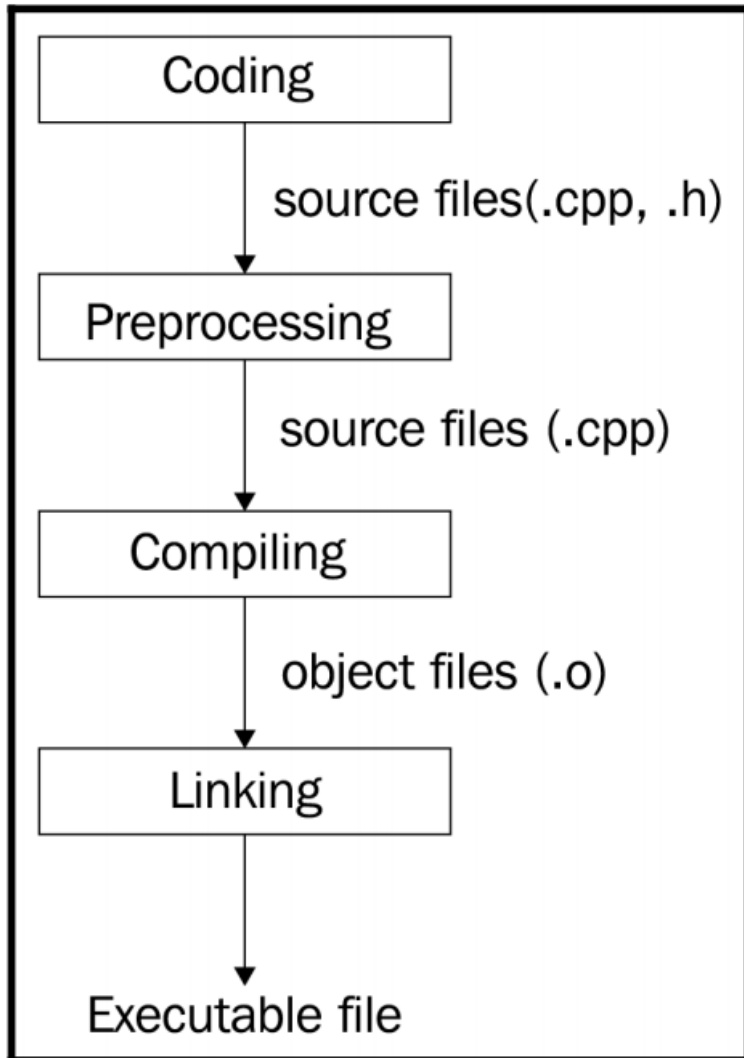


**Парадигма программирования** — это совокупность идей и понятий, определяющих стиль написания компьютерных программ (подход к программированию). Это способ концептуализации, определяющий организацию вычислений и структурирование работы, выполняемой компьютером.

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.

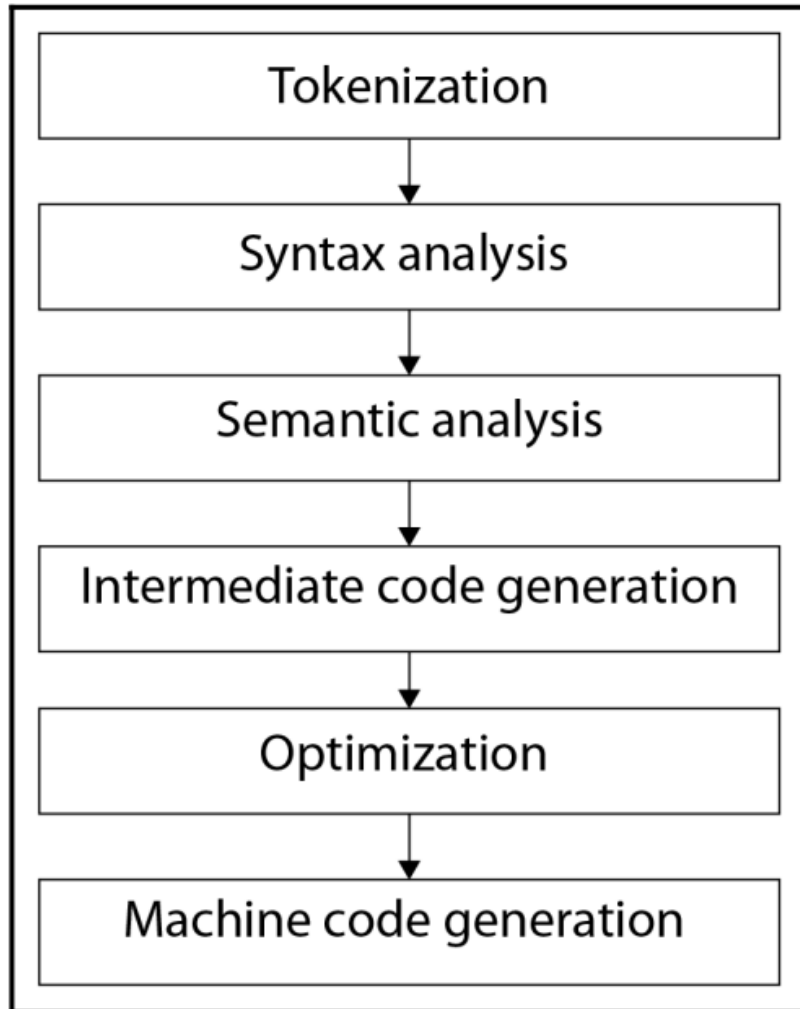


# Создание программы



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, чтобы мы могли разбивать наши программы на большое количество подмодулей и использовать функции из одного подмодуля в другой.

# Соглашение об именовании



1. Токенизация, это процесс разбиения компилируемого кода на минимальные значимые элементы.
2. Синтаксический анализ, проверяет все ли правильно в собираемых частях кода, с точки зрения синтаксиса языка.
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 C++

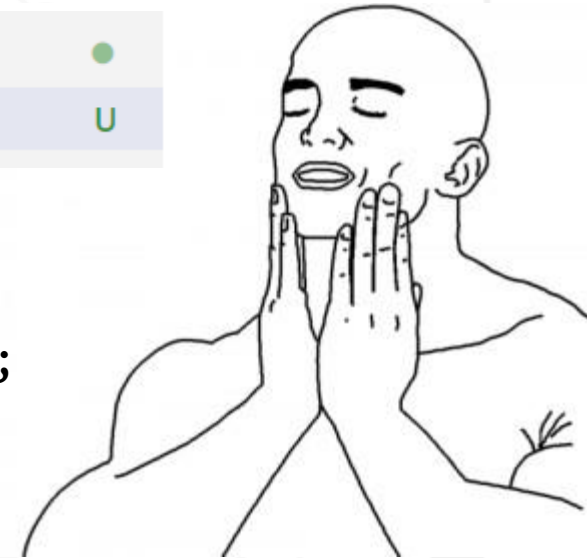


Код наших программ пишется в файлах с расширением .cpp и .h.



Любая программа на C++ начинается с функции:

```
int main (){  
    cout << "Hello 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.



# Структура файла сpp



Основной файл программы на C++ должен содержать блоки:

1. Блок препроцессорных директив. (например define или ifndef)
2. Блок импорта библиотек. (#import <>)
3. Блок определения именованной области видимости. (using namespace std;)
4. Блок декларирования или описания функций (здесь должны быть все функции которые используются в main).
5. Блок описания функции main.
6. Блок описания дополнительных функций (здесь могут быть любые функции, которые ранее не были использованы).

```
#include <iostream>
```

```
using namespace std;
```

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

# Соглашение об демонстрации синтаксиса



Для корректной демонстрации конструкций языка потребуется приводить примеры кода и его структуры, по этой причине, разумно, заранее определить формат и вид описания этой структуры. Такое описание будем называть **“сигнатурой”**.

Символами “<” и “>” будем выделять те блоки, которые программист может варьировать при написании той или иной конструкции.

Тип в конструкции будем обозначать словом **“тип”**.

Наименование чего либо словом **“имя\_”**, после нижнего подчеркивания будем писать имя чего следует написать.

Любое введенное программистом значение будем обозначать словом **“значение”**.

Так же, в случае если существует вариативность в описании какой либо структуры, варианты будут описываться разделенные символом “|”.



# Данные в C++

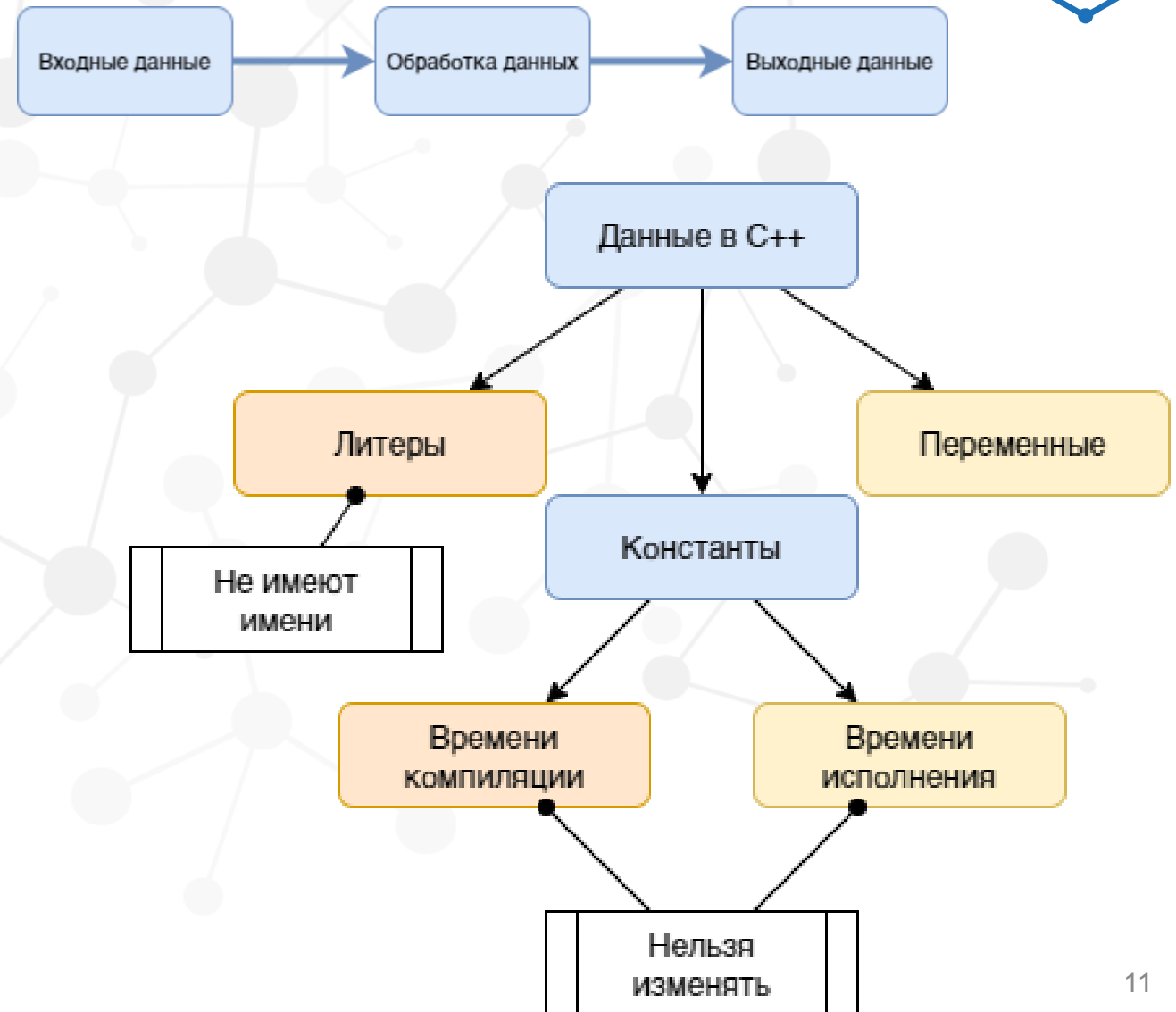


Принципиально программа это черный ящик которые получает данные на вход, проводит с ними манипуляции и выдает какой то результат

Поэтому для корректной работы с программой внутри нее необходимо хранить данные. В целом есть 2 принципиально разных варианта получать данные:

1. Получить данные на вход в программу, в каком либо виде (ввод пользователя или файл).
2. Получить данные в процессе написания программы (ввод разработчика).

В зависимости от того как получены данные, в C++ с ними придется работать разными способами и конструкциями языка.



# 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.

- **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 **signed**, 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**

## Пользовательские типы

Это те типы которые пользователь комбинируя каким либо образом встроенные типы и описывая взаимодействия между ними создает сам

# Переменные в C++



## Описание переменной

**Декларирование** - это процесс резервирования имени под переменную.

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

В каждой приведенной строке происходит декларирования и инициализация переменных.

Каждая строка имеет определённую структуру, которую принято называть сигнатурой, таким образом сигнатура объявления переменной:

```
bool b = false;  
int n = 10;  
double df = 13.24;  
char c = 'c';  
int array[N] {10, 4, 6, 8, 3, 9, 4, 5, 9, 40};
```

<тип> <имя\_переменной> = <значение>;

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



# Литералы 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
```

Для вывода в C++ используются специализированные конструкции. В основе этих конструкций лежит понятие потока ввода вывода, которое мы будем разбирать в дальнейшем, а на данный момент достаточно просто запомнить как их использовать.

При этом важно не забыть импортировать библиотеку.  
iostream

```
cout << "Текст" << "любой" << "цепочкой" << endl;
cout << "Число" << 14 << " " << 1.1f << endl;
```

# Константы в C++

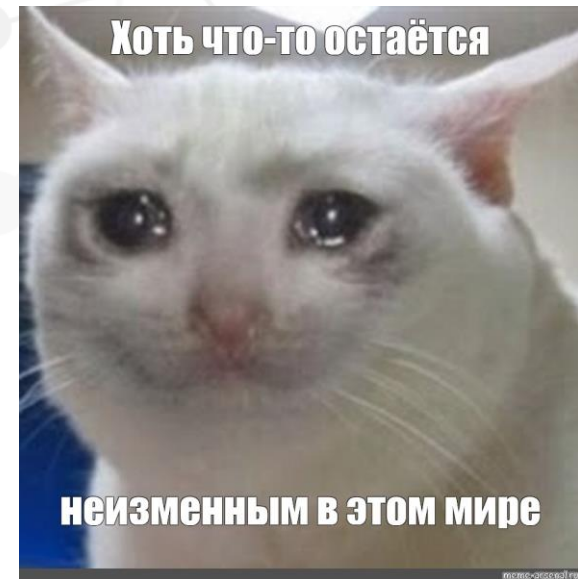


**Константы** — это переменные значение которых нельзя изменять. Константы бывают 2х видов:

**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';
```



# Особенности constexpr



```
int sum (int a, int b)
{
    return a + b;
}
```

```
constexpr int new_sum (int a, int b)
{
    return a + b;
}
```

```
void func()
{
    constexpr int a1 = new_sum (5, 12); // OK: constexpr-переменная
    constexpr int a2 = sum (5, 12); // ошибка: функция sum не является constexpr-
    выражением
    int a3 = new_sum (5, 12); // OK: функция будет вызвана на этапе компиляции
    int a4 = sum (5, 12); // OK
}
```

Особенностью констант времени компиляции является то, что по сути они не являются константами на момент компиляции. Такие константные значения могут вычисляться при помощи функций, помеченных этим же спецификатором и таким образом по сути являются переменные времени компиляции, а их задачей является создание константы времени исполнения.

# Структурный подход и область видимости в C++



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

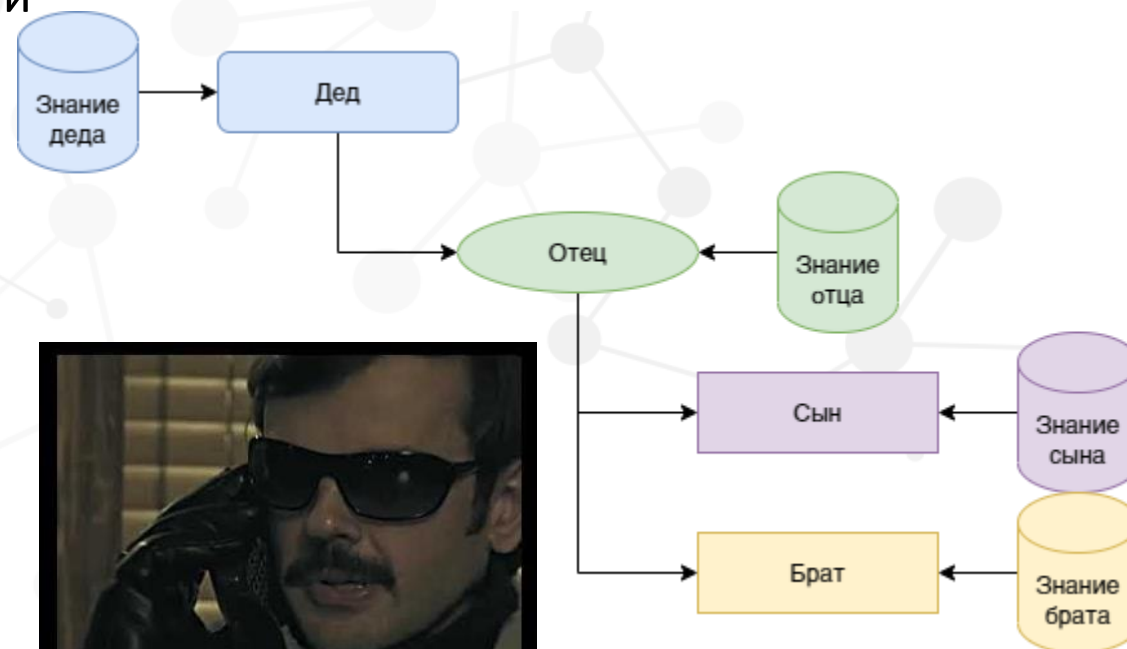
В C++ структурой кода или, что более распространено блоком называют любой код заключенный в фигурные скобки.

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

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.

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



# 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 истинно  
}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 соответствие 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;  
}
```



# Условия в C++



```
bool b = true;
bool c = true;
if(b){
    cout << "in if" << endl;
}else if(c){
    // даже при том, что условие истинно,
    // выведено все равно не будет
    cout << "in else if" << endl;
}else{
    cout << "in else" << endl;
}
```

```
const char top_direction = 't';
char direction = 't';

switch(direction){
    case top_direction:
        cout << "To top direction" << endl;
        break;
    default:
        cout << "Undefined direction" << endl;
        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  
}
```

## 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>;);
```

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

# 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>;
```

```
}
```

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

# Примеры



```
double square(double value){  
    return value * value;  
}
```

```
double a = -10;  
double b = 10;  
double x = a;  
double y = square(x);  
double y_o = 0;  
while (y - y_o > 0.001){  
    y_o = y;  
    x = (a + b) / 2;  
    if(square(a) * square(x) < 0){  
        b = x;  
    }else{  
        a = x;  
    }  
    y = square(x);  
}  
cout << x << endl;
```

```
const int N = 10;  
int array[N] {10, 4, 6, 8, 3, 9, 4, 5, 9, 40};  
int sum {0};  
for(int i = 0; i < N; i++){  
    sum += array[i];  
}  
cout << sum << endl;
```

The background is a solid blue color with a complex, abstract network of white lines and circles. The circles vary in size, and the lines connect them in a non-uniform, organic pattern, resembling a molecular structure or a data network.

**Спасибо за внимание**