**II – Video:** [**C in 42 min**](https://youtu.be/0NPVivMBRsU)

Привет всем и добро пожаловать на этот курс языка C за 42 минуты или около того.

**Введение**

Прежде всего, небольшое уточнение: этот курс не предназначен для обучения программированию на языке C, но он дает вам полное описание языка. Целью курса является просто показать вам элементы, которые вам нужны, чтобы решить задачи во время бассейна. Это подраздел C, позволит вам добиться выполнение всего, чего от вас требуют во время обучения.

1. **Что такое компьютер?**

Прежде чем мы начнем программировать, первое, что нам нужно понять: Что такое компьютер? Что внутри него? И как это работает?

* Первый элемент - процессор «CPU». Он делает все вычисления и сравнения.
* Второй элемент - оперативная память «ОЗУ» / «RAM», которая хранит все, что происходит на компьютере (данные обрабатываемые процессором и машинный код (программы)). Это очень важный элемент.
* Третий элемент - периферийные устройства, которые обеспечивают связь между компьютером и внешним миром.

Эти три элемента связаны между собой «[bus](https://ru.wikipedia.org/wiki/Шина_(компьютер))» системой (компьютерной шиной), которая позволяет информации перемещаться. Также есть система синхронизации, которая позволяет элементам работать вместе. Также есть «[clock](https://ru.wikipedia.org/wiki/Генератор_тактовых_импульсов)» сиcтема синхронизации (генератор тактовых импульсов), которая позволяет элементам работать вместе.

Конечно, компьютер гораздо сложнее, чем в этом описании, но именно такое представление нам будет нужно во время бассейна, а также на протяжении первого года учебы в школе. Мы будем придерживаться этого очень краткого описания.

Подведем итоги: процессор CPU, то есть блок обработки данных, система памяти, которую мы называем RAM и перефирийные устройства, система взаимодействия с остальным миром.

1. **Что такое программирование?**

Программирование (в контексте предыдущего описания, то есть в компьютерной парадигме) - означает давать инструкции процессору, то есть список основных действий, которые будут систематически иметь эффект манипулирования данными.

Мы сможем выполнять чрезвычайно простые операции, такие как сравнение значений, сбор, вычитание, деление и другую базовую арифметику. Мы сможем отправить значение на новый адрес, то есть мы копируем информацию с одного адреса на другой. Сможем отправлять значения периферийным устройствам, например, отображать их на экране или получить обратно это значение.

Очень важно понять, это то, что мы делаем элементарные вещи, которые будут сделаны очень быстро и в очень большом количестве, что позволит нам, с помощью этиз основных инструкций симулировать чрезвычайно сложное поведение, может быть, даже симулировать интеллект. В любом случае, на данный момент система программирования сможет выполнять только самые базовые вещи.

1. **Так что насчет C ?**

Так что насчет языка С во всей этой истории? В конце концов, ваш компьютер будет работать с языком, называемый «машинный язык», который является элементарным языком, составленным непосредственно из машинного кода, что довольно сложен для людей.

Язык C - это довольно простой язык, предназначенный для компиляции, будучи последовательным, он будет иметь в основном прямой перевод в элементарные операции, поэтому не очень сложный, но довольно легко читабельный для людей с. С позволяет нам быть эффективными в работе с машинным кодом, то есть мы преобразуем код C, который мы пишем, в машинный язык, который затем будет выполняться процессором.

Еще один уровень, который мы должны рассмотреть, а именно операционную систему (OS).

OS - это программное обеспечение, которое заставит компьютер работать. Она будет запущена при запуске компьюетра и позволяет нам иметь интерфейс. Позволяет взаимодействовать между пользователем и компьютером.

Мы увидим позже более полные определения для операционной системы, потому что мы говорим об очень сложной программе, для которой было проделано много работ с точки зрения объема и времени. OS управляет всем функционированием компьютера, и ваша программа будет использовать интерфейс операционной системой.

Например, чтобы уточнить, когда вы хотите отобразить символ, ваша программа, написанная на языке C, спросит операционную систему, через “системные вызовы” отобразить символ, и операционная система это делает. Таким образом, в вашей программе на C вам не нужно программировать всю часть системы, которая требуется для отображения символов.

Язык C позволяет вам написать ряд довольно простых инструкций, которые позволяют вам взаимодействовать с операционной системой, которая, в свою очередь, будет использовать интерфейс для отображения или для других операций, которые вам понадобятся.

1. **Инструкции and обрабатываемая цифровая информация**

Как мы видели ранее, у компьютера есть память RAM, в которой он хранит данные обрабатываемые процессором, и следуя полученным инструкциям он может выполнять операции с этими данными.

Программа на С является одновременно представлением (описание) данных, которыми мы будем манипулировать и также включает в себя инструкции, по которым мы выполняем операции с этими данными.

В оперативной памяти (двоичной(binary)) данные представлены 8 bit'ной группой (1 bit - 1 ед. хранения и обработка цифр. информации, 8 bit = 1 byte).

|  |  |  |
| --- | --- | --- |
| Length | Name | Example |
| 1 | Bit | 0 |
| 4 | Nibble | 1011 |
| 8 | Byte | 10110101 |

8bit’ная группа из 0 и 1 называются вместе октетом, и это позволяет нам хранить числа от 0 до 255, в зависимости от их представления. В русском языке октет обычно называют [byte](https://ru.wikipedia.org/wiki/Байт). Октет(=1byte=8bit) может принимать 256 возможных состояний (кодов, значений, комбинаций битов (нулей и единиц)).

Каждому байту присваивается адрес, который позволяет получить к нему доступ.

Язык Си позволяет нам выполнять операции с этими байтами, поэтому манипулирует этой адресной системой.

1. **Система** [**типов**](https://metanit.com/cpp/tutorial/2.3.php) **данных C**

|  |  |  |
| --- | --- | --- |
| Тип | Число байтов | Диапазон |
| char | 1 – 8 bit | -128 / +127 |
| unsigned char | 1 – 8 bit | 0 / 255 |
| short | 2 – 16 bit | -32768 / +32767 |
| unsigned short | 2 – 16 bit | 0 / 65536 |
| Int | 4 – 32 bit | −2147483648 / 2147483647 |
| unsigned int | 4 – 32 bit | 0 / 4 294 967 295 |
| long long | 8 – 64 bit | −9223372036854775808 / +922… |
| unsigned long long | 8 – 64 bit | 0 / 18446744073709551615 |
| \* | 8 – 64 bit | адрес |

Язык C использует типы для представления данных, рассмотрим несколько типов.

1. Тип “char”, который состоит из 8 bit’ов, что является октетом / bayt’ом. Может представлять любое значение данных из диапазона от -128 до +127, связан он с классической арифметикой.
2. Тип “unsigned char” также состоящий из 8 bit’ов, но с другим представлением, имеющим значения от 0 до 255.
3. Тип “shart” состоящий из 2 bayt’ов по 16 bit, который позволяет представлять значения от -32768 до +32767.
4. Тип “unsigned short”, но я пожалуй пока его пропущу.
5. Тип “int” на 4 байта, который обычно является типом, который мы используем чаще всего, потому что он покрывает большинство потребностей.
6. Также тип "unsigned int", "long" и "long long", которые позволяют представлять числовые значения до 64 бит. У нас конечно также есть "unsigned long long".
7. Существует также последний тип, называемый «\*» (звездочка) или “pointer”, который позволяет иметь не только тип данных, но и адрес этих данных. Мы можем считать, что адрес также является числовым значением, но у нас есть специальная арифметика для этих значений, которую мы увидем позже.
8. **Структура программ**

Теперь, когда мы знаем, как представлять данные, давайте поговорим о структуре программы. Программа это набор функций.

Структура программы состоит из набора функций. То, что мы называем «функцией», - это закрытый объект, потому что переменные внутри недоступны извне, объект который получает информацию через параметры и которая обеспечивает результат, а между ними у нас есть обработка данных. Скоро мы увидим, как это работает.

Каждая функция состоит из двух частей: инструкции и локальные переменные.

1. **Первые элементы синтаксиса**

Объявление функции.

Здесь вы можете увидеть первый пример синтаксиса объявления функции.

type name(type1 param1, type2 param2, ...)

BLOCK

Функция объявлена довольно просто.

Мы устанавливаем ее тип, то есть тип результата (возвращаемое значение), предоставляемого функцией. Ранее мы видели, что функция получает значения через параметры и выдает результат. Таким образом, тип с самого начала является типом значения, предоставленного функцией.

Затем у нас идет имя функции, и затем у нас есть список параметров, указывающих для каждого типа и их имени, разделяя их запятой. У нас может быть любые параметры, и у нас их может не быть, и в этом случае мы пишем открытые( и закрытые скобки). Так мы определяем, что предоставляет функция и какие параметры она получает.

Сразу следует блок («block»), который содержит саму функцию.

Функция всегда разбивается на две части, как вы видите ниже:

BLOCK:

{

local variables declaratin // объявление локальных переменных

instructions // инструкции

}

между двумя скобками { } у нас есть часть объявления локальных переменных, за которой следуют инструкции. У нас могут быть любые инструкции. У нас может и не быть локальных переменных или инструкций. Блок всегда заканчивается закрытой скобкой}.

1. **Пример 1**

Я быстро приведу пример простой функции:

Int sum(int value1, int value2)

{

Int total;

Instrctions

}

Здесь мы объявляем функцию sum («сумма»), которая приводит к целому числу («int») и получает два параметра. Один параметр называется «value1» типа «int», а второй также имеет тип «int», называемый «value2». После этого мы открываем блок с открытой скобой{.

С помощью «total int» мы объявляем локальную переменную целого числа (целочисленную). Когда мы говорим о «локальном», это потому, что она находится только внутри функции, то есть эта локальная переменная не существует вне функции. Далее я не привел пример инструкций, но, по крайней мере, вы знаете, что они находятся здесь. Написав } закрытую фигурную скобку, мы заканчиваем блок функции.

1. **Составные типы (type)**

На языке С мы можем объявлять составные типы. Это структуры данных, состоящие из элементарных типов из списка, представленного в разделе 5. Их существует несколько, но мы будем использовать во время бассейна только два типа основных структур.

1. Arrays / Массив

type name[numeric value]; int tab[42];

2. Structures / Структура

struct struct\_name struct s\_fortytwo

{ {

type name; int a;

type name; char b[21]

}; };

Declaration / Объявление

struct struct\_name name; struct s\_fortytwo a;

1. type name[numeric value]; int tab[42];

Первый тип – array / массив, массив – последовательность (серия элементов одного типа) объектов одного типа в памяти. Это простой тип объявления: тип, имя, открытая квадратная скобка[, количество элементов в массиве и закрытая квадратная скобка].

Справа у вас есть пример: «int tab [42]», Массив tab может содержать 42х целых числа. На уровне индексации он изменяется от 0 до 41, потому что мы всегда начинаем с нуля. Мы зарезервировали 42 целых числа в памяти, к которым мы можем получить доступ к каждому из них. Чтобы использовать их, мы пишем имя массива, а в квадратных скобках номер целого числа, которое мы хотим получить. Например, если мы хотим получить номер 13 массива tab, то мы напишем «tab [12]».

2. struct struct\_name struct s\_fortytwo

{ {

type name; int a;

type name; char b[21]

}; };

struct struct\_name name; struct s\_fortytwo a;

Следующий комбинированный тип немного сложнее, мы будем использовать его позже

Это структуры данных, структуры позволяют нам определить представление объекта в памяти, который будет составлен в свою очередь из более простых элементов, которые могут быть тоже структурами, массивами и т.д.

Структура определяется следующим образом: ключевое слово "struct", имя структуры, открытая скобка, объявление компонентов структуры, для которых мы указываем их тип и имя, закрытые скобки и заканчиваются точкой с запятой. Смотрите пример справа, где мы объявляем структуру с именем "s\_fortytwo". Он состоит из целого числа «a» и массива «b» из 21 символа.

Затем ниже мы объявляем эту структуру простым способом, мы пишем struct struct\_name(имя структуры) name (имя переменной, компонент), а затем точку с запятой;.

Мы резервируем в памяти необходимое пространство, и когда мы используем это имя, программа узнает, что это структура.

1. **The Norm (Синтаксические нормы)**

Компилятор языка C достаточно гибок на уровне синтаксиса, что означает, что мы можем написать выбор несколькими различными способами, но чтобы сделать написание программ максимально читабельным и обеспечить некоторую однородность, мы определили стандард, Норму (Norm). Эта норма будет представлена чуть позже, у вас будет курс, посвященный этой теме.

Важно понимать, что сам стандард, норма имеет много плюсов:

1. Первый плюс, это возможность вам легко читать чужой код и наоборот.

2. Во-вторых, это помогает вам научиться структурированному программированию. Если вы уже до этого программировали, то вы увидите, что Норма написания кода, это будет всего лишь корректировка.

3. Самый большой плюс, это то, что Норма дает нам возможность увидеть, почувствовать и быть полезной при чтении чужого кода. \*от себя: да это уже было, но думаю на этом и идет акцент нормы)\*

Норма позволит читать быстрее чужой код, который написан по ней.

Поэтому я рекомендую вам неукоснительно соблюдать Норму, тем более что у нас есть система валидации / программа, которая будет проверять соответствие кода на Норму.

1. **Отправная точка**

Откуда начать?:

Мы видели, что программа на C - это набор функций, которых может быть очень много. Так откуда нам начать?

Обязательная функция, с определенным именем **main**:

В синтаксисе языка С заведено так, что первая выполняемая функция, которая запускает выполнение программы, является "main" функцией.

Где-нибудь в твоем коде должно быть:

int main()

{

}

Поэтому при написании программы необходимо, чтобы где-то в ней вы объявили функцию «main».

Эта функция, которая берет определенное число параметров, позже ты увидишь, как они работают и как их использовать. “main" возвращает int, вскоре ты увидишь, как это работает. Таким образом, мы имеем определнное объявление с именем «int main ()» (и возможными параметрами в скобках) и {блок функций.}

1. **3 вида инструкций**

Теперь, когда мы увидели общую структуру программы и то, как мы можем представлять, объявлять данные, все, что осталось, так это понять, как инструкции работают, как работать с этими данными.

1. Blocks:

{

declarations;

instructions;

}

Все очень просто, у нас есть три типа инструкций, первый - это «блок», который как мы уже видели, всегда состоит из: открытой скобки, объявлении локальных переменных для этого блока, серий инструкций, выполненных в блоке и закрытой скобке.

2. Expressions / Выражения:

expressions;

Выражение - это вычисление, которое всегда вычисляет числовое значение или числовую оценку. За выражением всегда следует точка с запятой. Это мы разберем чуть позже.

3. Control structures / Структуры управления

Третьим и последним являются «управляющие структуры», которые дают возможность управлять производством серий инструкций внутри блока.

1. **Выражения**

Начнем с выражений. Выражения, как следует из названия, представляют собой серию вычислений, которые дают результат.

1. Непосредственные числовые значения:

0 42 0x1A 010

Самая простое вычисление - это числовое значение. Например, число 0 или число 42, или вы можете написать значение в шестнадцатеричном (основании 16) - 0x1A, или в восьмеричном (основание 8) - "010". Они вычисляются непосредственно по их значению, поэтому компилятор представит значение, как результат.

2. Variables

a // пример: int a.

b[18]

stf\_var.a // пример: a structure

Второй тип выражений – переменные.

Когда мы напишем имя переменной, она вернет (вычислит) ее значение. Например целое число (int) переменной ("a"), оно будет вычисленно по непрерывному значению в переменной "a".

Что касается массивов, если вы введете «b [18]», оно будет вычислено по значению 19-го элемента в самом массиве.

Структуры, переменная с именем «sft\_var.a» будет вычисляться со значением элемента «a» в этой структуре.

Короче говоря, переменная вычисляется непосредственно по значению, содержащемуся в ней.

3. Function calls:

sum(18, a)

Третий тип выражений - это вызов функции. Для вызова функции о мы пишем имя функции и параметры. Параметры будут отправлены в эту функцию, далее компилятор вызовет эту самую функцию и произойдет вычисление возвращаемого значения этой функции. Потому что мы видели ранее, что функция получает некоторые значения в качестве параметров, выполняет вычисления внутри и предоставляет значение.

4. Computations: + - \* / % ()

1 + 3 – ( a \* 42 + b[0]) / sum(21, 34 – b[10])

Конечно, мы можем комбинировать все эти типы выражений, используя вычислительные правила.

Компилятор выполняет арифметические действия и как мы видим, мы можем написать очень сложные выражения. Если вы введете «1 + 3», соответствующая сумма будет вычислена. Компилятор обрабатывает операции в скобках в соответсвии с приоритетами.

У нас происходит: вычисление 0 значения в массиве “b”, далее значение переменной a помноженное на 42, далее мы получим значение суммы двух элементов деленное на возвращаемое значение функции и т.д.

Компилятор вычисляет результат вычислений этих арифметических выражений и выполняет он их в правильном порядке.

5. Assignations:

a = 42

b[2 +a] = sum(12, 30)

sft\_var.a = 0

Последний тип выражения - «присваивание», это когда мы пишем, например, «а = 42». Мы говорим компилятору поместить значение 42 в переменную «a», и эта операция вернет результат «42». Таким образом происходит изменение значения переменной “a” на 42, а также систематичекое вычисление.

К примеру у нас есть «b [2 + a]», компилятор сложит 2 со значением переменной a (a = 42), и мы получим желаемый инек массива b, которому присваиваем возвр. значение функции справа, Таким образом, все это является присваиванием, которое изменит данные в переменной.

1. **Выражения (опять)**

1. Компораторы / [Операторы](https://ru.wikipedia.org/wiki/Операторы_в_C_и_C%2B%2B) сравнения. == != < > <= >= || && !

a < 2 + b[23]

a – 2 == 35 || b[11] + 3 >= 17

У нас также есть другие выражения, которые мы называем компораторы, комператоры в С это операторы, чья оценка приведет (как для всех выражений) к значению. Мы называем «true» возвращение значениея 1 и «false» когда возвращаемое значение 0. К примеру, когда вы ставите «a == b», он будет проверять, равно ли содержимое переменной «a» содержимому переменной «b», и если содержимое равно, он вернет 1, если нет, он вернет 0. Все операторы сравнения работают одинаково, если вы наберете «a < b», аналогично, он будет смотреть на свое значение «a», а затем на его значение «b», если «а» меньше, чем «b», он вернет 1, если нет, он вернет 0.

Таким образом, в С есть операторы сравнения: "==" (равенство), "!="(неравенство), "<" (меньше), ">" (больше), ">=" (больше или равно), "<=" (меньше или равно).

Далее логические операторы: "||" (или), «&&» (и), "!" (не). Если мы напишем «! a», если «a» это 0, он вернет 1, если нет, он вернет 0.

2. Бинарные (троичные) операторы: >> << | & ^

У нас также есть то, что называется бинарным оператором. Бинарные операторы будут работать непосредственно с двоичными данными переменной. Мы рассмотрим это позже, это не очень полезно для старта, но вы увидите, что они позволяют обрабатывать каждый бит переменной.

3. Составное присваивание:

a += 10 <> a = a + 10

a++ <> a += 1 <> a = a + 1

b[50] /= 2 <> b[50] = b[50] / 2

Также есть составное присваивание, например сложение, совмещенное с присваиванием используя оператор “+=” :

«a += 10» тоже самое, что и «a = a + 10»;

«a ++» тоже самое, что и “a += 1”, тоже самое, что и «a = a + 1».

Есть и деление совмещенное с присваиванием "/=". Все это способы упростить наш код..

4. Особенности

‘A’ <> 65

У нас есть несколько специальных выражений, например, 'A' возвращает [ASCII](https://www.ascii-code.com/)-код символа между апострофами 'A'. В этом примере ASCII-код 'A' равен 65. Коды ASCII очень важны, потому что они позволяют нам отображать символы.

Итак, у нас есть таблица ASCII, которая позволяет нам напрямую использовать символы, о ней мы поговорим позже.

sizeof(type\_of\_var)

У нас есть оператор "sizeof". Мы пишем sizeof(тип переменной или непосредственно переменную) и он возвращает размер этой переменной в памяти.

&a (адрес переменной в памяти RAM)

Мы можем напиать «&» для переменной, и он вернет адрес этой переменной в памяти.

exp1 ? exp2 : exp3

У нас также есть тернарная условная операция, которую мы используем довольно редко,ее алгоритм следующий:

1. Вычисляется exp1

2. Если exp1 является истинным, то вычисляется значение выражения exp2, в противном случае — значение выражения exp3

3. Вычисленное значение возвращается

5. Strings / Символьные строки

Последний пункт – символьная строка “hello forty-two” - она начинается и заканчивается с кавычеками (" "), внутри кавычек следует ряд символов. В массиве char будет автоматически храниться последовательность ASCII-кодов. Массив заканчивается на \0 (терминирующий ноль), этот ноль вычисляет адрес в памяти первого символа.

1. **Указатели**

1. Когда мы перечисляли типы в С, то там мы видели определенный тип – указатели (pointers).

Указатель — переменная, содержащая адрес объекта. Указатель не несет информации о содержимом объекта, а содержит сведения о том, где размещен объект.

Указатели широко используются в программировании на языке Си.

Указатели часто используются при работе с массивами.

Память компьютера можно представить в виде последовательности пронумерованных однобайтовых ячеек, с которыми можно работать по отдельности или блоками.

Каждая переменная в памяти имеет свой адрес — номер первой ячейки, где она расположена, а также свое значение. Указатель — это тоже переменная, которая размещается в памяти. Она тоже имеет адрес, а ее значение является адресом некоторой другой переменной.

2. Объявление:

Указатель, как и любая переменная, должен быть объявлен. Общая форма объявления указателя:

тип \*ИмяОбъекта;

int a; // объявляем переменную a типа int

int \*p; // объявляем указатель p типа int

p = &a; // указатель p получает адрес переменной a

Тип указателя - это тип переменной, адрес которой он содержит.

Указатель **p** содержит адрес переменной **a** типа **int**.

3. Применение:

Как применяются указатели? Довольно просто.

Если **p** содержит адрес **a**, тогда **\*p** это другое имя области памяти переменной **a**:

int a; // объявляем переменную a типа int

int \*p; // объявляем указатель p типа int

p = &a; // вводим в указатель p = адрес переменной a

\*p = 1 // присваиваем указателю p – 1, исходя из этого переменная a = 1 (\*p = 1; <> a =1;)

Это очень важная вещь, вам нужно часто возвращаться к этому определению.

4. Арифметическая система для указателей:

У нас также есть арифметическая система для указателей. Возьмем массив tab[42]

int tab[42];

int \*p;

p = &(tab[0]); // вводим в указатель p – первый индекс массива tab

\*(p + 5) // получаем доступ к значению 5 массива tab, тоже самое, если напишу

// tab[5] <> \*(p + 5)

1. **Управление**

1. if: условный оператор, рассмотрим подробнее структуру алгоритма «развилка». Разветвляющимся называется такой алгоритм, в котором выбирается один из нескольких возможных вариантов вычислительного процесса. Каждый подобный путь называется ветвью алгоритма.

Признаком разветвляющегося алгоритма является наличие операций проверки условия. Чаще всего для проверки условия используется условный оператор if.

Условный оператор if может использоваться в форме полной или неполной развилки.

|  |  |
| --- | --- |
| Неполная развилка | Полная развилка |
| if (Условие)  {  БлокОпераций1;  } | if (Условие)  {  БлокОпераций1;  }  Else  {  БлокОпераций2;  } |
| Ð½ÐµÐ¿Ð¾Ð»Ð½Ð°Ñ ÑÐ°Ð·Ð²Ð¸Ð»ÐºÐ° | Ð¿Ð¾Ð»Ð½Ð°Ñ ÑÐ°Ð·Ð²Ð¸Ð»ÐºÐ° |
| В случае неполной развилки если **Условие** истинно, то **БлокОпераций1** выполняется, если **Условие** ложно, то **БлокОпераций1** не выполняется. | В случае полной развилки если **Условие** истинно, то выполняется **БлокОпераций1**, иначе выполняется **БлокОпераций2**. |

**БлокОпераций** может состоять из одной операции. В этом случае наличие фигурных скобок, ограничивающих блок, необязательно.

2. while: цикл с предусловием while, циклом называется блок кода, который для решения задачи требуется повторить несколько раз.

Цикл выполняется до тех пор, пока блок проверки условия возвращает истинное значение.  
Тело цикла содержит последовательность операций, которая выполняется в случае истинного условия повторения цикла. После выполнения последней операции тела цикла снова выполняется операция проверки условия повторения цикла. Если это условие не выполняется, то будет выполнена операция, стоящая непосредственно после цикла в коде программы.

Общая форма записи:

|  |  |
| --- | --- |
| while (Условие)  {  БлокОпераций;  } |  |

Если **Условие** выполняется (выражение, проверяющее **Условие**, не равно нулю), то выполняется **БлокОпераций**, заключенный в фигурные скобки, затем **Условие** проверяется снова.

Последовательность действий, состоящая из проверки **Условия** и выполнения **БлокаОпераций**, повторяется до тех пор, пока выражение, проверяющее **Условие**, не станет ложным (равным нулю). При этом происходит выход из цикла, и производится выполнение операции, стоящей после оператора цикла.

3. [returt](https://prog-cpp.ru/c-functions/): по окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. В зависимости от того, возвращает функция значение или нет, синтаксис оператора return может быть одним из следующих:

return(значение); // в функции, которая возвращает значение

return; // в функции, которая не возвращает значение

Оператор return также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор **return** не обязательно должен находиться в конце тела функции. Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения **void**, а оператор **return** может либо отсутствовать, либо не возвращать никакого значения: