

Программа на языке Си состоит из набора директив препроцессора, определений функций и глобальных объектов. Директивы препроцессора управляют преобразованием текста до его компиляции. Глобальные объекты определяют используемые данные или состояние программы. А функции определяют поведение или действия программы. Точкой входа для выполнения программы является функция с фиксированным идентификатором main. Язык Си является блочно-структурированным. Каждый блок заключается в фигурные скобки. Каждое действие в языке Си заканчивается символом «точка с запятой» — ;. В качестве действия может выступать вызов функции или осуществление некоторых операций.

Константа — это величина, которая при выполнении программы остаётся неизменной. Переменная — это ячейка памяти для временного хранения данных. int a - переменная, `c` - символьная константа. Константу можно задать с помощью квалификатора const, макроса #define и перечисления enum.

Компиляция – это процесс превращения исходного кода в машинный код. Этапы компиляции:

1) Препроцессинг

Препроцессор — это макро процессор, который преобразовывает вашу программу для дальнейшего компилирования. На данной стадии происходит происходит работа с препроцессорными директивами. Например, препроцессор добавляет хэдеры в код (#include), убирает комментирования, заменяет макросы (#define) их значениями, выбирает нужные куски кода в соответствии с условиями #if, #ifdef и #ifndef.

2) Компиляция

На данном шаге gcc выполняет свою главную задачу — компилирует, то есть преобразует полученный на прошлом шаге код без директив в ассемблерный код. Это промежуточный шаг между высокоуровневым языком и машинным (бинарным) кодом. Ассемблерный код — это доступное для понимания человеком представление машинного кода.

3) Ассемблирование

Ассемблер преобразовывает ассемблерный код в машинный код, сохраняя его в объектном файле.

Объектный файл — это созданный ассемблером промежуточный файл, хранящий кусок машинного кода.

4) Компоновка

Компоновщик (линкер) связывает все объектные файлы и статические библиотеки в единый исполняемый файл, который мы и сможем запустить в дальнейшем. Для того, чтобы понять как происходит связка, следует рассказать о таблице символов.

Таблица символов — это структура данных, создаваемая самим компилятором и хранящаяся в самих объектных файлах. Таблица символов хранит имена переменных, функций, классов, объектов и т.д., где каждому идентификатору (символу) соотносится его тип, область видимости. Также таблица символов хранит адреса ссылок на данные и процедуры в других объектных файлах.



Язык Си предоставляет множество базовых типов. Большинство из них формируется с помощью одного из четырёх арифметических спецификаторов типа, (char, int, float и double), и опциональных спецификаторов (signed, unsigned, short и long).

В большинстве систем char - 1 байт, int - 4 байта, float - 8 байт, double - 16 байт.

char - символ, int - целое число, float - число с плавающей точкой, double - float с удвоенной точностью.

signed/unsigned - знаковый/беззнаковый, short/long - короткий/длинный (в 2 раза меньше/больше байт занимает в памяти).

void - тип без значения, \_Bool - логический тип.

Структура (struct) - сложный тип данных, состоящий из нескольких полей возможно разных типов.

Арифметические операторы: + – сложение; - – вычитание; \* – умножение; / – деление; % – остаток от деления.

Постфиксная/префиксная инкрементация/декрементация: x++,++x,x--,--x.

int x = 0;

x++ = 0, x = 1

++x = 1, x = 1

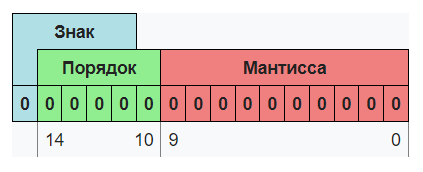
**Целые числа:**

Прямой код представляет собой одинаковое представление значимой части числа для положительных и отрицательных чисел и отличается только знаковым битом. В прямом коде число 0 имеет два представления «+0» и «–0».

Обратный код для положительных чисел имеет тот же вид, что и прямой код, а для отрицательных чисел образуется из прямого кода положительного числа путем инвертирования всех значащих разрядов прямого кода. В обратном коде число 0 также имеет два представления «+0» и «–0».

Дополнительный код для положительных чисел имеет тот же вид, что и прямой код, а для отрицательных чисел образуется путем прибавления 1 к обратному коду. Добавление 1 к обратному коду числа 0 дает единое представление числа 0 в дополнительном коде. Однако это приводит к асимметрии диапазонов представления чисел относительно нуля.

**Числа с плавающей запятой:**



Знак — один бит, указывающий знак всего числа с плавающей точкой. Порядок и мантисса — целые числа, которые вместе со знаком дают представление числа с плавающей запятой в следующем виде:

(−1)S×M×BE, где S — знак, B — основание, E — порядок, а M

— мантисса. Десятичное число, записываемое как ReE, где R

— число в полуинтервале [1;10), E

— степень, в которой стоит множитель 10 ; в нормализированной форме модуль R будет являться мантиссой, а E

— порядком, а S будет равно 1

тогда и только тогда, когда R принимает отрицательное значение. Например, в числе −2435e9

S = 1

B = 10

M = 2435

E = 9



Результатом логической операции является либо 0, либо 1. Результат имеет тип int. && - логическое и, || - логическое или.

Результатом реляционного выражения является 1, если проверенная связь имеет значение true, и 0, если значение false. Результат имеет тип int.

< Первый операнд меньше второго операнда

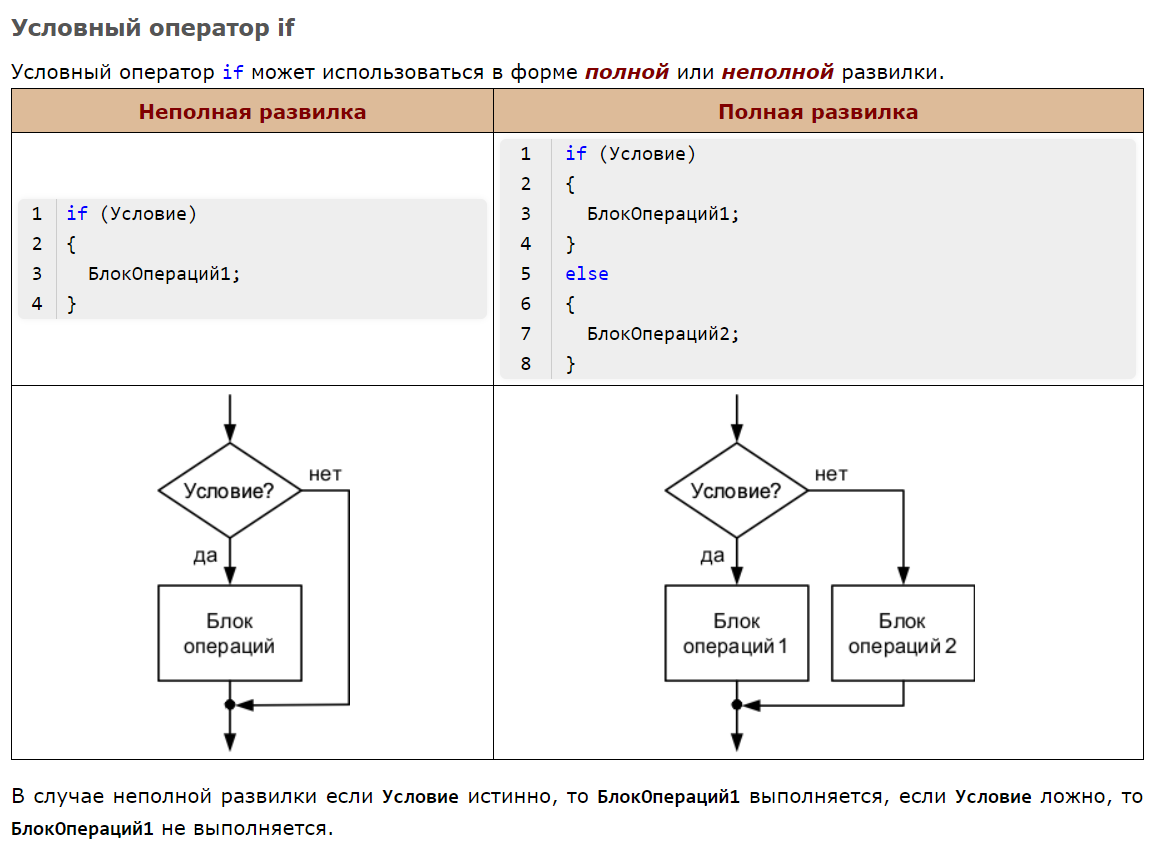
> Первый операнд больше второго операнда

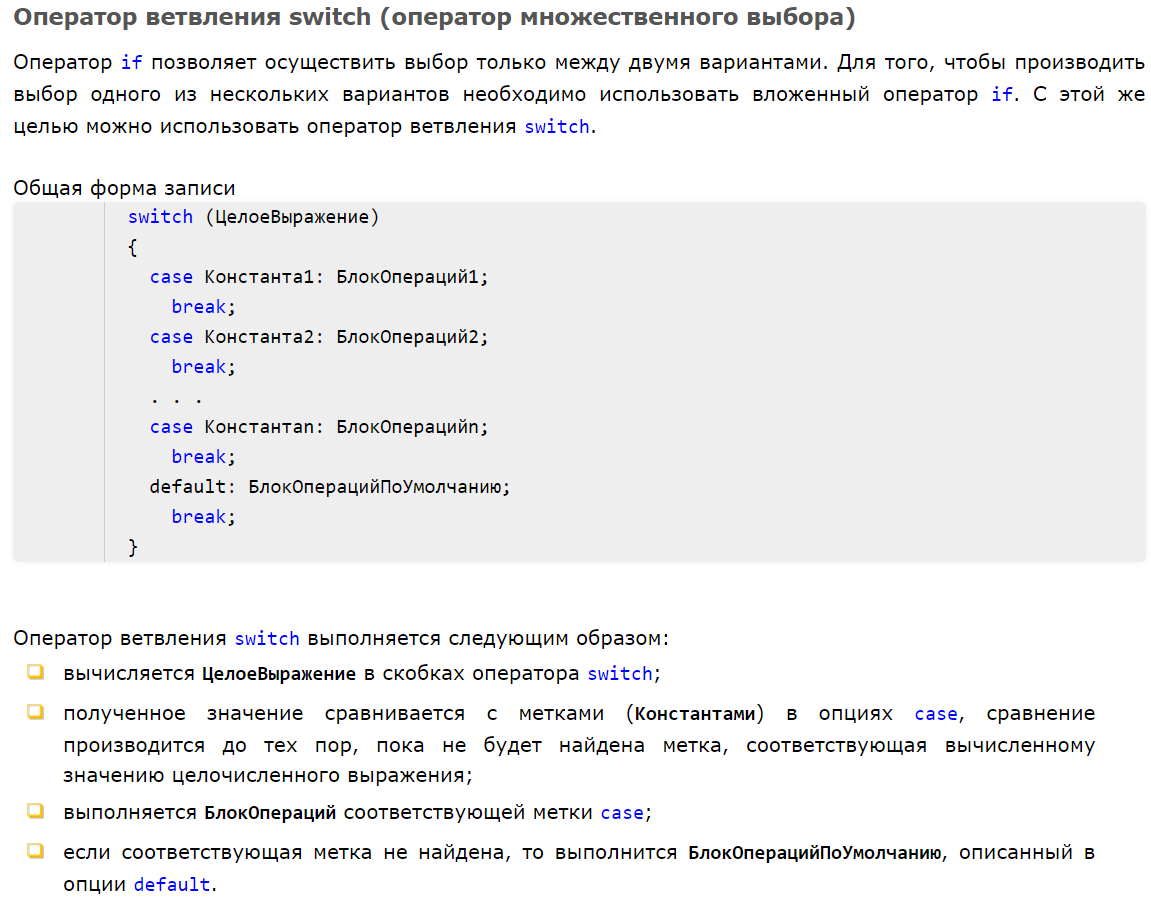
<= Первый операнд меньше или равен второму операнду

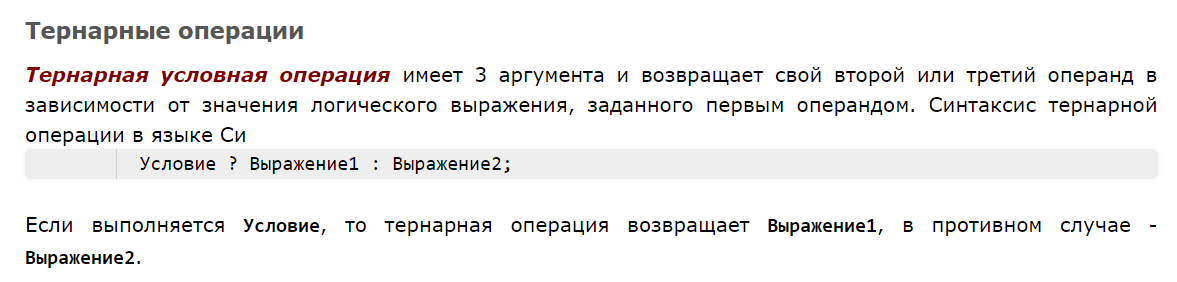
>= Первый операнд больше или равен второму операнду

== Первый операнд равен второму операнду

!= Первый операнд не равен второму операнду

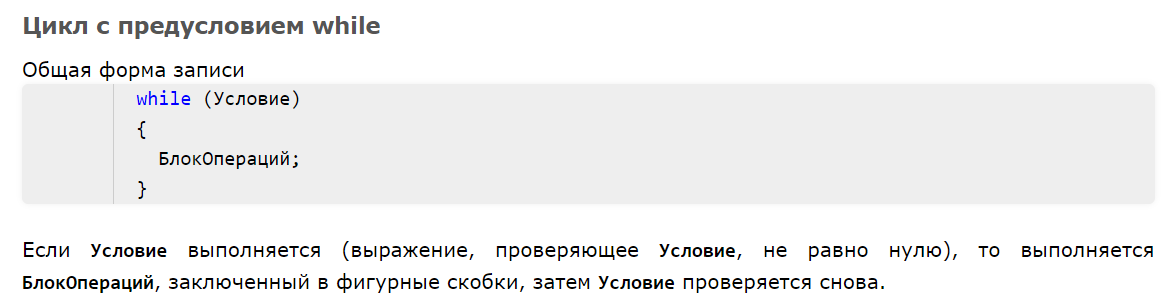


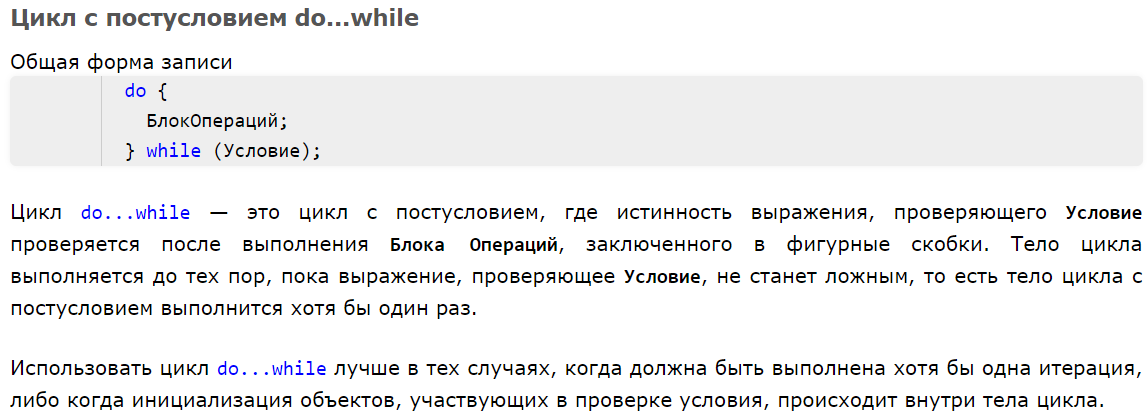


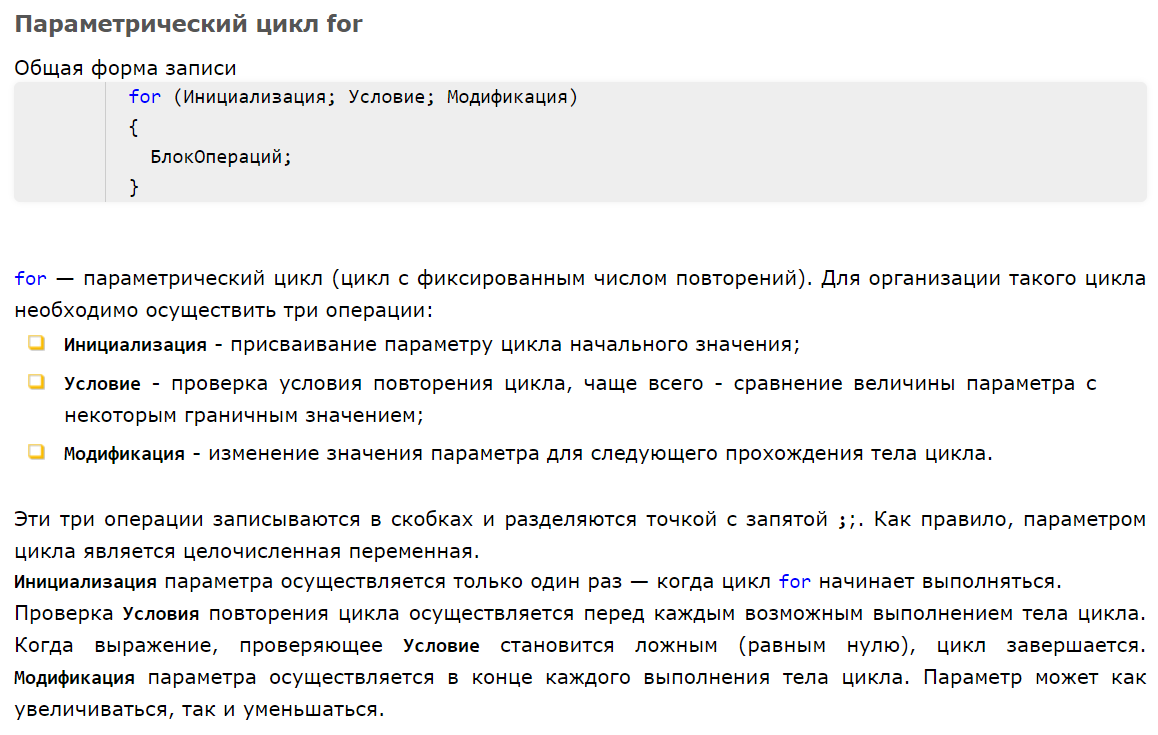


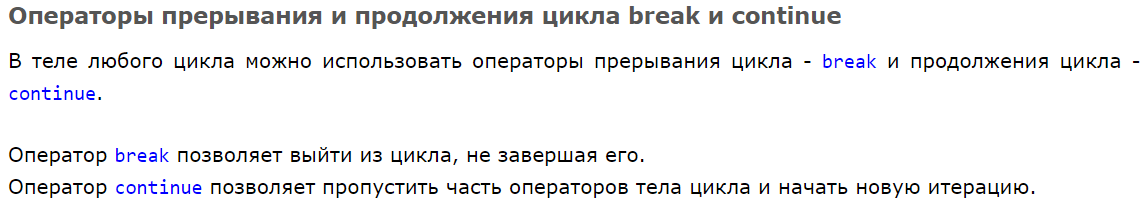


Циклом называется блок кода, который для решения задачи требуется повторить несколько раз.  
Каждый цикл состоит из блока проверки условия повторения цикла и тела цикла.



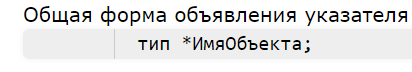








Указатель — переменная, содержащая адрес объекта.



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

Для работы с указателями в Си определены две операции:

операция \* (звездочка) — позволяет получить значение объекта по его адресу — определяет значение переменной, которое содержится по адресу, содержащемуся в указателе;

операция & (амперсанд) — позволяет определить адрес переменной.

Арифметика указателей: увеличение на единицу означает, что мы хотим перейти к следующему объекту в памяти, который находится за текущим и на который указывает указатель. А уменьшение на единицу означает переход назад к предыдущему объекту в памяти. Аналогично указатель будет изменяться при прибавлении/вычитании не единицы, а какого-то другого числа. В отличие от сложения операция вычитание может применяться не только к указателю и целому числу, но и к двум указателям одного типа.



Массив — это непрерывный участок памяти, содержащий последовательность объектов одинакового типа, обозначаемый одним именем.

Массив характеризуется следующими основными понятиями:

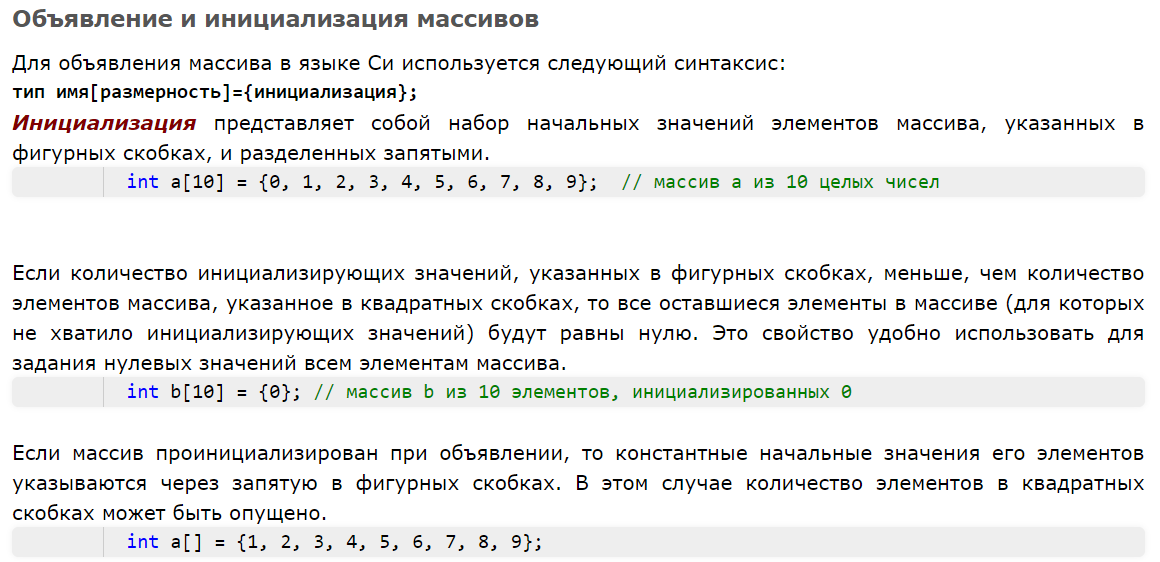
Элемент массива (значение элемента массива) – значение, хранящееся в определенной ячейке памяти, расположенной в пределах массива, а также адрес этой ячейки памяти.

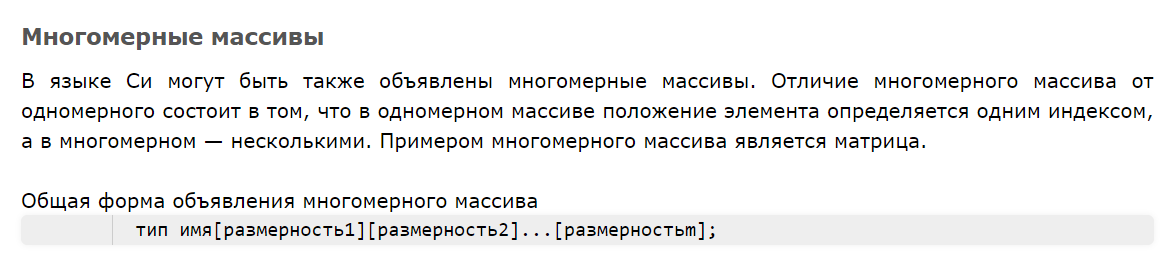
Каждый элемент массива характеризуется тремя величинами:

адресом элемента — адресом начальной ячейки памяти, в которой расположен этот элемент; индексом элемента (порядковым номером элемента в массиве);

значением элемента.

Адрес массива – адрес начального элемента массива.





Передача массива в функцию

Обработку массивов удобно организовывать с помощью специальных функций. Для обработки массива в качестве аргументов функции необходимо передать адрес массива и размер массива.

Исключение составляют функции обработки строк, в которые достаточно передать только адрес.

При передаче переменные в качестве аргументов функции данные передаются как копии. Это означает, что если внутри функции произойдёт изменение значения параметра, то это никак не повлияет на его значение внутри вызывающей функции.



Все функции, в том числе и те, которые пишет пользователь, устроены сходным образом. У них имеется две основных составных части: заголовок функции и тело функции. Заголовок состоит из трёх обязательных частей:

тип возвращаемого значения;

имя функции;

параметры функции.

Параметры - это переменные, которые используются при создании функции.

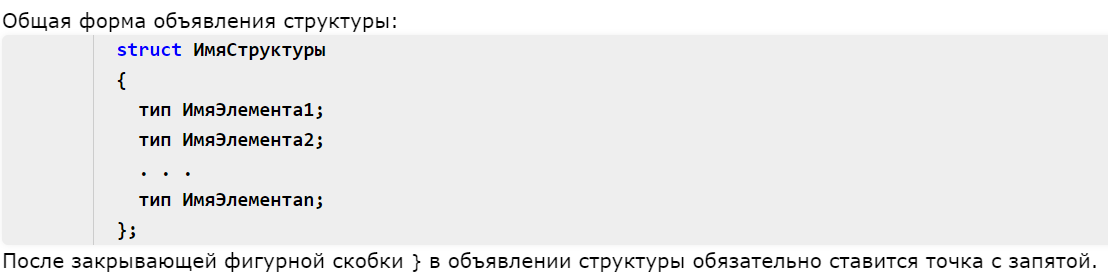
Аргументы - это фактические значения (данные), которые передаются функции при вызове.

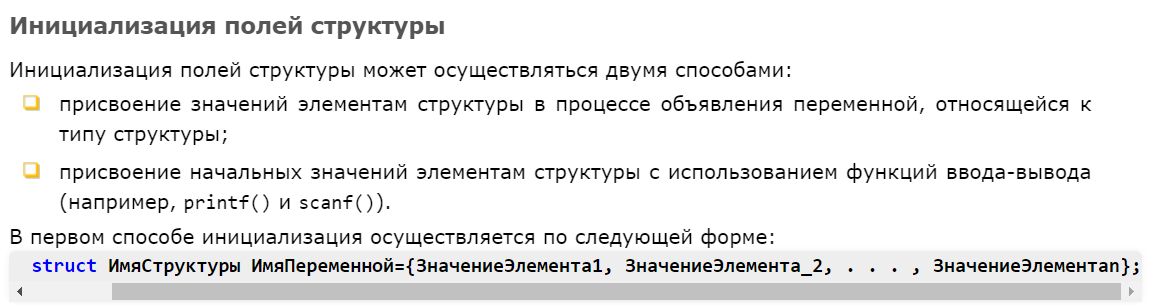
Функция, вызывающая саму себя называется рекурсивной.

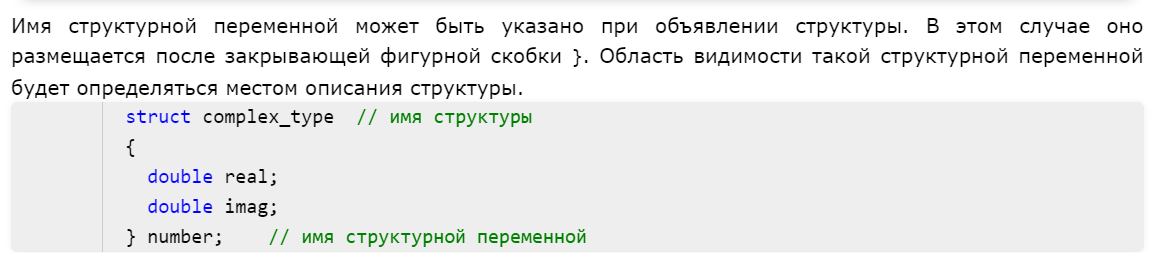
Перед вызовом функции, необходимо объявить её прототип.



Структура — это объединение нескольких объектов, возможно, различного типа под одним именем, которое является типом структуры. В качестве объектов могут выступать переменные, массивы, указатели и другие структуры.



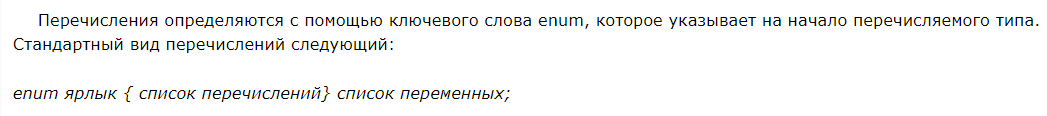




Обратиться к полю структуры можно с помощью оперетора. Если нужно обратиться к полю указателя на структуру есть 2 варианта:



Перечисления - это набор именованных целочисленных констант, определяющий все допустимые значения, которые может принимать переменная.



По умолчанию, первое поле структуры принимает численное значение 0, следующее 1, следующее 2 и т.д. Можно задать нулевое значение явно. Перечисления используются для большей типобезопасности и ограничения возможных значений переменной.



"Идентификаторы" или "символы" — это имена, задаваемые в программе для переменных, типов, функций и меток. Написание и регистр символов в именах идентификаторов должны отличаться от всех ключевых слов. Вы не можете использовать ключевые слова (C или Microsoft) в качестве идентификаторов; они зарезервированы для специального использования.

Область видимости описывает участок или участки программы, где можно обращаться к идентификатору. Переменная в С имеет одну из следующих областей видимости:

в пределах блока, в пределах функции, в пределах прототипа функциии в пределах файла.

В пределах блока - идентификаторы, объявленные в {}.

В пределах функции - только метки go to.

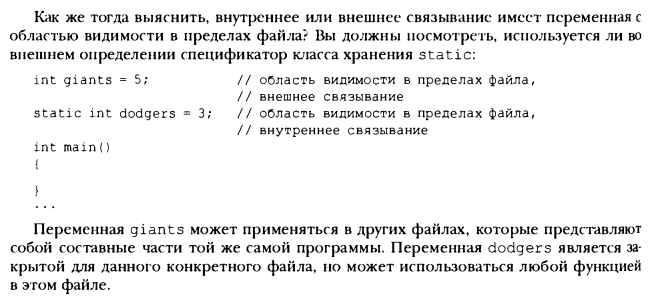
В пределах прототипа функции - объявленные в прототипе функции.

В пределах файла - объявленные вне функций (глобальные переменные).

Переменная в С имеет одно из следующих связываний: внешнее связывание, внутреннее связывание или отсутствие связывания.

Переменные с областью видимости в пределах блока, функции или прототипа функции не имеют связывания. Это означает, что они являются закрытыми для блока, функции или прототипа, в котором определены. Переменная с областью видимости в

пределах файла может иметь либо внутреннее, либо внешнее связывание. Переменная с внешним связыванием может применяться в любом месте многофайловой программы, а переменная с внутренним связыванием — где угодно в единице трансляции.





Продолжительность хранения характеризует постоянство объектов, доступных через идентификаторы. Объект в С имеет одну из следующих четырёх продолжительностей хранения: статическую, потоковую, автоматическую или выделенную.

Если объект имеет статическую продолжительность хранения, он существует на протяжении времени выполнения программы. Переменные с областью видимости в пределах файла имеют статическую продолжительность хранения.

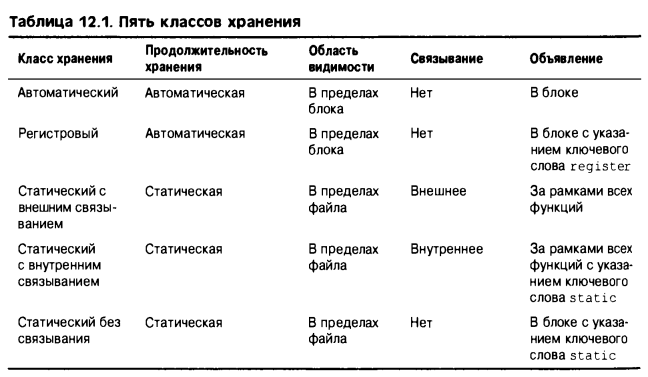
Переменные с областью видимости в пределах блока обычно имеют автоматическую продолжительность хранения. Память для этих переменных выделяется, когда поток управления входит в блок, где они определены, и освобождается, когда поток управления покидает этот блок. Идея заключается в том, что память, используемая для автоматических переменных, является рабочим пространством или временной памятью, которая может применяться многократно. Например, после завершения вызова функции память, которую функция использовала для своих переменных, может

быть задействована при вызове следующей функции.

Массивы переменной длины демонстрируют небольшое исключение в том, что они существуют от места своего объявления и до конца блока, а не от начала блока и до его конца.

Переменная может иметь область видимости в пределах блока, но

статическую продолжительность хранения. Чтобы создать такую переменную, объявите её внутри блока и добавьте в объявление ключевое слово static.





Переносимые объектные файлы создаются на этапе компиляции в машинный код в ходе сборки проекта. Они считаются промежуточными и используются в качестве ингредиентов для создания дальнейших и конечных продуктов.

В переносимом объектном файле, полученном из скомпилированной единицы трансляции, можно найти следующие элементы:

- инструкции машинного уровня, сгенерированные из функций, найденных в единице трансляции (код);

- значения инициализированных глобальных переменных, объявленных в единице трансляции (данные);

- таблицу символов, содержащую все символы, которые были объявлены и на которые ссылается единица трансляции.

ELF формат - Executable and Linkable Format - состоит из секций.

Секция .text содержит все машинные инструкции для единицы трансляции. В секциях .data и .bss находятся соответственно значения для инициализированных глобальных переменных и количество байтов, необходимых для неинициализированных

глобальных переменных. Секция .symtab хранит таблицу символов.

символы в переносимых объектных файлах не обладают итоговыми, абсолютными адресами; данная информация определяется на этапе компоновки.

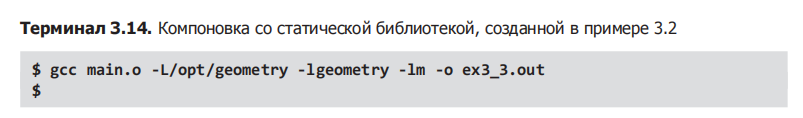
Исполняемые объектные файлы содержат больше секций, содержащих данные, необходимые для загрузки и выполнения программы.

Статическая библиотека в Unix — обычный архив с переносимыми объектными файлами.

В системах семейства Unix применительно к статическим библиотекам действует общепринятое соглашение об именовании. Имя файла должно начинаться с lib и иметь расширение .a.

Команда для создания стат. библиотеки.

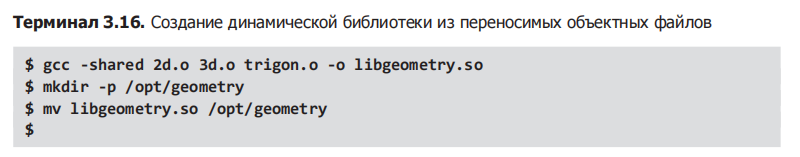
ar crs libexample.a aa.o bb.o ... zz.o



Параметр -L/opt/geometry сообщает компилятору gcc, что каталог /opt/geometry входит в число тех мест, в которых можно найти статические и разделяемые библиотеки. По умолчанию компоновщик ищет библиотечные файлы в традиционных каталогах, таких как /usr/lib или /usr/local/lib.

Параметр -lgeometry говорит компилятору gcc, что ему нужно искать файл libgeometry.a или libgeometry.so.

Динамические (они же разделяемые) библиотеки — еще один продукт компиляции с возможностью повторного использования. Как можно догадаться по названию, от статических библиотек они отличаются тем, что не входят в состав итогового исполняемого файла. Вместе этого их необходимо загружать и подключать во время запуска процесса.

Объектный файл содержит инструкции машинного уровня, эквивалентные единице трансляции. Однако они хранятся не в произвольном порядке, а сгруппированы в так называемые символы.

Каждый переносимый объектный файл содержит только часть символов функций, необходимых для сборки полноценной исполняемой программы. Компоновщик собирает все симво-

лы из разных переносимых объектных файлов в один большой объектный файл, формируя тем самым исполняемую программу.



