# История создания языка Си, стандарты.

***Стандарты***

Язык программирования Си разрабатывался в период с 1969 по 1973 годы в лабораториях Bell Labs. Название языка стало логическим продолжением старого языка "Би", многие особенности которого были положены в основу.

В 1978 году Брайан Керниган и Деннис Ритчи опубликовали первую редакцию книги "Язык программирования Си". Эта книга, известная среди программистов как "K&R", служила многие годы неформальной спецификацией языка. Версию языка Си, описанную в ней, часто называют "K&R C".

По мере развития язык сначала стандартизировали как ANSI C (также C89), а затем этот стандарт был принят комитетом по международной стандартизации ISO как ISO C (также C90). В стандарте С99 язык получил новые возможности, такие как новые типы данных, массивы переменной длины и встраиваемые функции, а в стандарте C11 в язык добавили улучшенную поддержку Юникода, реализацию потоков и поддержку атомарных типов. Однако с тех пор язык развивается медленно, и в стандарт С18 попали лишь исправления ошибок стандарта C11.

Язык Си разрабатывался как язык системного программирования, для которого можно создать однопроходный компилятор. Стандартная библиотека также невелика. Как следствие данных факторов - компиляторы разрабатываются сравнительно легко. Поэтому данный язык доступен на самых различных платформах. К тому же, несмотря на свою низкоуровневую природу, язык ориентирован на переносимость. Программы, соответствующие стандарту языка, могут компилироваться под различные архитектуры компьютеров.

Целью языка было облегчение написания больших программ с минимизацией ошибок по сравнению с ассемблером, следуя принципам процедурного программирования, но избегая всего, что может привести к дополнительным накладным расходам, специфичным для языков высокого уровня.

***Отличия***

Основные концепции языка:

* сравнительно низкий уровень абстракции;
* простота реализации компилятора;
* ответственность за действия программиста лежит на программисте.

# Области использования языка Си.

* разработка операционных, встраиваемых систем;
* разработка высокопроизводительных приложений;
* компиляторы, интерпретаторы;
* ПО с открытым исходным кодом.

# Основные черты языка Си.

* простая языковая база, из которой в стандартную библиотеку вынесены многие существенные возможности, вроде математических функций или функций работы с файлами;
* ориентация на процедурное программирование;
* статическая система типов, предохраняющая от бессмысленных операций;
* использование препроцессора для абстрагирования однотипных операций;
* доступ к памяти через использование указателей;
* небольшое число ключевых слов;
* передача параметров в функцию по значению, а не по ссылке (передача по ссылке эмулируется с помощью указателей);
* наличие указателей на функции и статические переменные;
* области видимости имён;
* структуры и объединения - определяемые пользователем собирательные типы данных, которыми можно манипулировать как одним целым.

В то же время в Си отсутствуют:

* вложенные функции;
* прямое возвращение нескольких значений из функций;
* сопрограммы;
* средства автоматического управления памятью;
* встроенные средства объектно-ориентированного программирования;
* средства функционального программирования.

# Сравнение языка Си и языка Python.

Python - интерпретируемый язык, Си - компилируемый. Как следствие, программы на Python более переносимые, но исполняются гораздо медленнее.

Python - язык с динамической типизацией, Си - со статической. Динамически типизированный код быстрее писать, но сложнее отлаживать и сопровождать, так как проверка корректности типов происходит во время исполнения. Корректность типов в статически типизированном коде происходит до исполнения, что позволяет выявлять многие ошибки и, в случае компиляции, генерировать более оптимальный машинный код.

За управление памятью в Python отвечает сборщик мусора. В Си же ответственность за это лежит на программисте. Такой подход усложняет написание кода, но позволяет его оптимизировать.

# Структура программы на языке Си.

# Дайте определение алфавита языка. Алфавит языка Си.

*Алфавит* - набор символов, допустимых для записи программ.

Алфавит языка Си состоит из

* латинских букв (A-Z, a-z);
* цифр (0-9);
* 29 графических символов (! " # % & ' ( ) \* + , - и т.д.);
* пробельных символов.

# Дайте определение лексемы. Правило выделения лексем в языке Си.

Из символов алфавита образуются лексемы.

***Лексема*** - это минимальная смысловая единица языка.

К лексемам в языке Си относятся

* идентификаторы;
* ключевые слова;
* константы;
* строковые литералы;
* пунктуаторы.

**Правило выделения лексем в языке Си:** Каждый очередной непробельный символ должен добавляться к считываемой лексеме до тех пор, пока она остаётся корректной.

Таким образом, например, можно точно сказать, что выражение a+++b тождественно a++ + b.

# Что такое идентификатор? Правила формирования идентификаторов в языке Си.

***Идентификатор*** - строка символов, используемая для идентификации некоторой сущности в программе.

В языке Си идентификатор может содержать буквы, цифры и символ подчеркивания. Идентификатор не может начинаться с цифры. Регистр символов имеет значение.

# Что такое ключевое слово?

***Ключевое слово*** — это идентификатор, смысл которого зафиксирован правилами языка программирования и который используется для распознавания предложений в программе.

(break, char, while, if, int, sizeof…)

# Что такое предопределенное имя?

***Предопределённый идентификатор*** - идентификатор, определённый в стандартной библиотеке и потому, как и зарезервированное слово, запрещённый к использованию в качестве идентификатора, определяемого программистом.

Такими идентификаторами являются константы ***EOF*** и ***NULL***, типы ***size\_t*** и ***ptrdiff\_t***, файловые переменные ***stdin***, ***stdout***, ***stderr*** и прочие сущности стандартной библиотеки.

# Что такое константа?

***Константа*** - величина, значение которой остаётся неизменным во время выполнения программы.

# Что такое синтаксис языка программирования? Способы описания синтаксиса.

*Синтаксис* - набор правил, описывающий комбинации символов алфавита, считающиеся правильно структурированной программой (документом) или её фрагментом.

Способы описания синтаксиса:

* формальные грамматики;

(Способ описания формального языка, то есть выделения некоторого подмножества из множества всех слов некоторого конечного алфавита.)

* формы Бэкуса-Наура и их расширения;

(Формальная система определения синтаксиса, в которой одни синтаксические категории последовательно определяются через другие.)

* диаграммы Вирта.

(Методика синтаксических диаграмм Вирта основана на графическом описании синтаксиса языка. Диаграммы, которые для этого используются, состоят из нескольких графических примитивов.)

# Что такое семантика языка программирования? Способы описания семантики.

***Семантика*** - система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

* естественный язык: описание естественным человеческим языком;
* формальная семантика: математическое описание;
* эталонная реализация: описание посредством компьютерной программы;
* набор тестов: описание при помощи примеров программ и их ожидаемого поведения. Хотя лишь несколько спецификаций языка начинались с этой формы спецификации, семантика набора тестов влияет на эволюцию некоторых спецификаций.

# Что такое переменная?

*Переменная* - абстракция ячейки памяти компьютера или совокупности таких ячеек.

# Атрибуты переменной.

Каждую переменную характеризует следующий набор атрибутов:

* идентификатор;
* адрес;
* тип;
* значение.

*Идентификатор* - строка символов, используемая для идентификации переменной в программе.

*Адрес* - это первая ячейка памяти, с которой связана данная переменная.

*Тип* определяет набор значений, которые может принимать переменная, и набор допустимых над ней операций.

*Значение* - это содержимое ячеек, связанных с переменной.

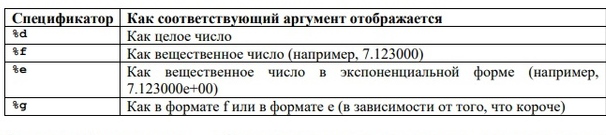
# Описание переменной на языке Си.

int a;

int a, b;

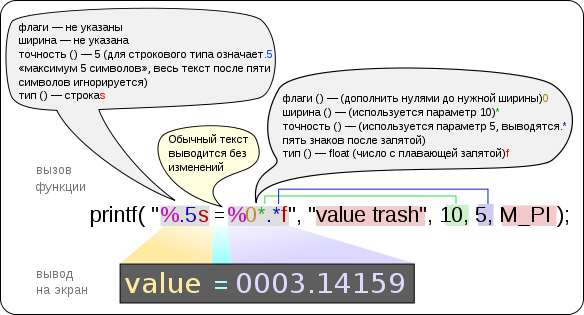
После определения переменной в ней хранится мусор

# Функция printf. Строка форматирования. Ошибки при использовании функции printf.



Спецификатор формата имеет вид:

%[флаги][ширина][.точность][размер]тип



Основные ошибки при использовании функции printf связаны либо с несоответствием количества спецификаторов и количества переменных, либо с несоответствием спецификатора и типа переменной.

# Функция scanf. Строка форматирования. Ошибки при использовании функции scanf.

В случае ошибки функция scanf() возвращает системную константу EOF. Обычно эта константа определена как –1.

При использовании функции scanf, как правило, допускаются те же самые ошибки, что и при использовании функции printf. Новая разновидность ошибок, специфичная именно для функции scanf, связана с тем, что аргументами функции scanf являются адреса переменных, а не сами переменные.

# Что такое выражение? Что такое операция?

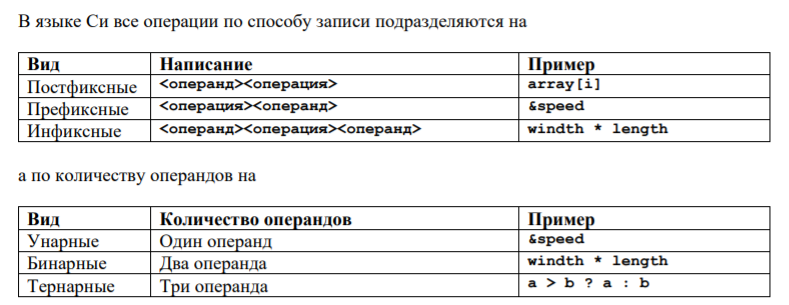
***Операция*** - конструкция в языках программирования, аналогичная по записи математическим операциям, то есть специальный способ записи некоторых действий. Элементы данных, к которым применяется операция, называют операндами.

***Выражение*** – простейшее средство описания действий. Состоит из констант, переменных, а также знаков операций и скобок.

# Что такое побочный эффект операции?

Некоторые операции имеют ***побочный эффект***. При выполнении этих операций, кроме основного эффекта - вычисления значения - происходят изменения объектов или файлов. Таковы, например, операции присваивания. Значением выражения a = b будет значение переменной b и в качестве побочного эффекта это значение будет присвоено переменной a.

# Классификация операций в языке Си по способу записи операции. Классификация операций в языке Си по количеству операндов.



# Что такое приоритет операции?

***Приоритет операций*** — очерёдность выполнения операций в выражении, при условии, что в выражении нет явного указания порядка следования выполнения операций (с помощью круглых скобок).

# Что такое ассоциативность операции?

***Ассоциативность*** - свойство операций, позволяющее восстановить последовательность их выполнения при отсутствии явных указаний на очерёдность при равном приоритете; при этом различается левая ассоциативность, при которой вычисление выражения происходит слева направо, и правая ассоциативность — справа налево. Соответствующие операторы называют левоассоциативными и правоассоциативными.

# Порядок вычисления подвыражений в языке Си.

Как уже говорилось выше, язык Си в общем не определяет порядок, в котором вычисляются подвыражения. Например, для выражения (a + b) \* (c - d) нельзя сказать какое из подвыражение (a + b) или (c – d) будет вычислено раньше.

# Порядок вычисления логических выражений в языке Си.

Логические выражения в языке Си всегда вычисляются слева направо (с99 6.5.13, 6.5.14). При

этом используется так называемая краткая схема вычисления логического выражения:

вычисления прекращаются как только становится понятным значение всего выражения.

# Полная и сокращенная схемы вычисления логических выражений.

# Операции присваивания в языке Си.

Формально алгоритм работы этой операции можно описать следующим образом:

1. Вычислить left-значение первого операнда (v) выражения.
2. Вычислить right-значение второго операнда (e) выражения.
3. Присвоить вычисленное right-значение вычисленному left-значению объекта данных;
4. Возвратить вычисленное right-значение как результат выполнения операции.

Если v и e имеют разный тип, значение e преобразуется к типу v (неявное преобразование).

# Особенности операции присваивания в языке Си.

Правоассоциативна



# Арифметические операции в языке Си.

Операнды арифметических операций (за исключением операции %) могут быть как целыми, так и вещественными. В бинарных операциях возможно смешение типов операций.

Операция «унарный плюс» в качестве результата возвращает значение своего операнда (т.е. ничего не делает). Эта операция была добавлена, исходя из соображений симметрии (есть операция «унарные минус», следовательно, должна быть операция «унарный плюс»).

В операции “%” оба операнда должны быть целочисленными (в противном случае возникнет ошибка компиляции).

Использование 0 в качестве правого операнда операций “/” или “%” приведет к непредсказуемому результату (так называемое неопределенное поведение).

Если оба операнда операций “/” и “%” являются целочисленными и один из них отрицательный, результат зависит от используемого стандарта. В C99 результат деления округляется большую сторону (по направлению к нулю).

-9 / 7 === -1

-9 % 7 === 2

# Операции инкремента и декремента в языке Си.

«инкремент» (т.е. увеличение на единицу)

«декремент» (т.е. уменьшение на единицу)

Постфиксная (i++) форма сначала возвращает значение переменной, а затем производится арифметическая операция. При записи в префиксной форме (++i) происходит тоже самое, но она сначала совершает арифметическую операцию, а после возвращает значение.

# Операции сравнения в языке Си.

Приоритет операций отношения меньше приоритета арифметических операций, поэтому выражение i + j < k – 1 ⬄ (i + j) < (k - 1)

Выражение i < j < k <=> (i < j) < k

Другими словами, в выражении сначала проверяется меньше ли значение переменной i значения переменной j, а затем 0 или 1 как результат этого сравнения будут сравниваться со 6 значением k. Т.е. выражение не проверяет, что величина j лежит между величинами i и k.

# Логические операции в языке Си.

Логические операции «И» (&&), «ИЛИ» (||) и «НЕТ» (!)

\_Bool – это целочисленный тип (точнее беззнаковый целочисленный тип). В отличие от целочисленного типа переменные типа \_Bool могут принимать только значения 0 и 1. Попытка присвоить отличное от нуля значение переменной типа \_Bool приведет к тому, что значение этой переменной станет равным 1.

Переменные типа \_Bool могут использоваться в арифметических выражениях (но это не желательно).

Стандарт c99 предоставляет заголовочный файл stdbool.h, который облегчает использование «нового» логического типа. Это заголовочный файл содержит макрос bool для обозначения \_Bool, а также макросы true и false.

**Побитовые операции**

Сдвиг влево(<<), вправо (>>), отрицание(~), и(&), или(|), исключающее или(^)

Битовые операции применимы только к целочисленным переменным. Обычно эти операции стараются выполнять над беззнаковыми целыми, чтобы

• повысить переносимость программы;

• не было путаницы с битом, который отвечает за знак.

При написании «сложных» выражений, в которых участвуют операции сдвига, нужно помнить, что они имеют меньший приоритет по сравнению с арифметическими операциями, поэтому, например, выражение a << 2 + 1 означает a << (2 + 1), но не (a << 2) + 1 .

Приоритет побитовых операций меньше приоритета операций сравнения и равенства. Поэтому, например, if (status & 0x4000 != 0) означает if (status & (0x4000 != 0)).

# Какие операции в языке Си обладают побочным эффектом?

Некоторые операции имеют **побочный эффект** (side effect). При выполнении этих операций, кроме основного эффекта - вычисления значения - происходят изменения объектов или файлов. Таковы, например, операции присваивания. Значением выражения a = b будет значение переменной b и в качестве побочного эффекта это значение будет присвоено переменной a.

# Что такое оператор?

В языке Си любое выражение можно «превратить» в оператор, добавив к этому выражению точку с запятой:

++i;

В языке Си точка с запятой является элементом оператора и его завершающей частью, а не разделителем операторов.

# Пустой оператор.

Пустой оператор — это оператор, состоящий только из точки с запятой. Он может появиться в любом месте программы, где по правилам синтаксиса требуется оператор. Выполнение пустого оператора не меняет состояния программы.

# Оператор-выражение.

Выполнение оператора выражение заключается в вычислении выражения.

# Составной оператор.

Составной оператор (или блок) группирует несколько операторов и объявлений путем их размещения в фигурных скобках в единый оператор

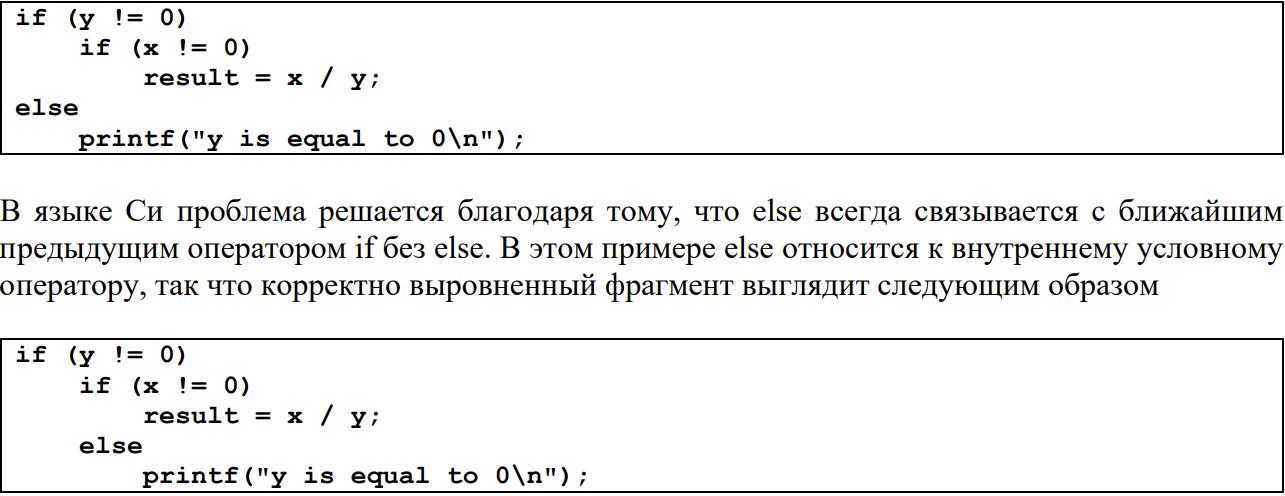
{

[список объявлений и операторов]

}

В отличие от оператора выражения составные операторы не завершаются точками с запятой. Составной оператор используется везде, где синтаксис предусматривает несколько операторов, но назначение программы требует их несколько

# Условный оператор.



Каскадный условный оператор – else if есть

# Условная операция.

|  |
| --- |
| выражение\_1 ? выражение\_2 : выражение\_3 |

выражение\_1, выражение\_2 и выражение\_3 могут быть выражениями любого типа. Условная операция требует трех операндов. Сначала вычисляется выражение выражение\_1. Если оно отлично от нуля, то вычисляется выражение выражение\_2, и его значение становится значением условной операции. Если значение выражение выражение\_1 равно нулю, то значением условной операции становится значение выражения выражение\_3.

# Оператор выбора.

Язык Си предоставляет оператор switch как альтернативу такому каскадному условному оператору

switch (mark)

{

case 5: printf("Excellent\n");

break;

case 4: printf("Good\n");

break;

case 3: printf("Average\n");

break;

case 2: printf("Poor\n");

break;

default: printf("Illegal mark\n");

break;

}

Считается, что оператор switch легче читается. Кроме того, switch часто быстрее, чем каскадный условный оператор.

Особенности оператора switch

* Управляющее выражение, которое располагается за ключевым словом switch, обязательно должно быть целочисленным (не вещественным, не строкой).
* Константное выражение – это обычное выражение, но оно не может содержать переменных и вызовов функций
* После каждой case-метки может располагаться любое число операторов. Никакие скобки не требуются. Последним оператором в группе таких операторов обычно бывает break.
* case-метки не могут быть одинаковыми.
* Порядок саse-меток (даже метки default) не важен.
* Только одно константное выражение может располагаться в case-метке, но несколько caseметок могут предшествовать одной и той же группе операторов
* case-метка default не является обязательной

Выполнение оператора break «внутри» оператора switch передает управление за оператор switch, т.е. на оператор, который располагается за оператором switch. Если бы оператор break отсутствовал, то стали бы выполняться операторы, расположенные в следующих case-метках

Пропуск оператора break является частой ошибкой при использовании оператора switch.

# Оператор цикла while.

while (выражение) оператор

# Оператор цикла for.

for (выражение\_1; выражение\_2; выражение\_3) оператор

Любое из трех выражений можно опустить, но точки с запятой должны остаться на своих местах. Если опустить выражение\_1 или выражение\_3, то соответствующие действия выполняться не будут.

Если же опустить проверку условия выражение\_2, то по умолчанию считается, что условие продолжения цикла всегда истинно.

# Операция запятая.

Иногда бывает необходимо написать оператор for с двумя или более выражениями инициализации или изменить несколько переменных в конце цикла.

**выражение\_1, выражение\_2**

Эта операция выполняется следующим образом: сначала вычисляется выражение\_1 и его значение отбрасывается, затем вычисляется выражение\_2. Значение этого выражения является результатом операции всей операции.выражение\_1 всегда должно содержать побочный эффект. В противном случае от этого выражения не будет никакого толка.

# Оператор цикла do-while.

do оператор while (выражение)

Оператор break.

Выход выполняется из ближайшего цикла или оператора switch.

# Оператор continue.

Если оператор break внутри цикла передает управление за оператор цикла, то оператор continue передает управление в конец цикла. В циклах while и do-while это означает проверку управляющего выражения, а в цикле for – выполнение выражение\_3 и последующую проверку выражение\_2.

# Оператор goto.

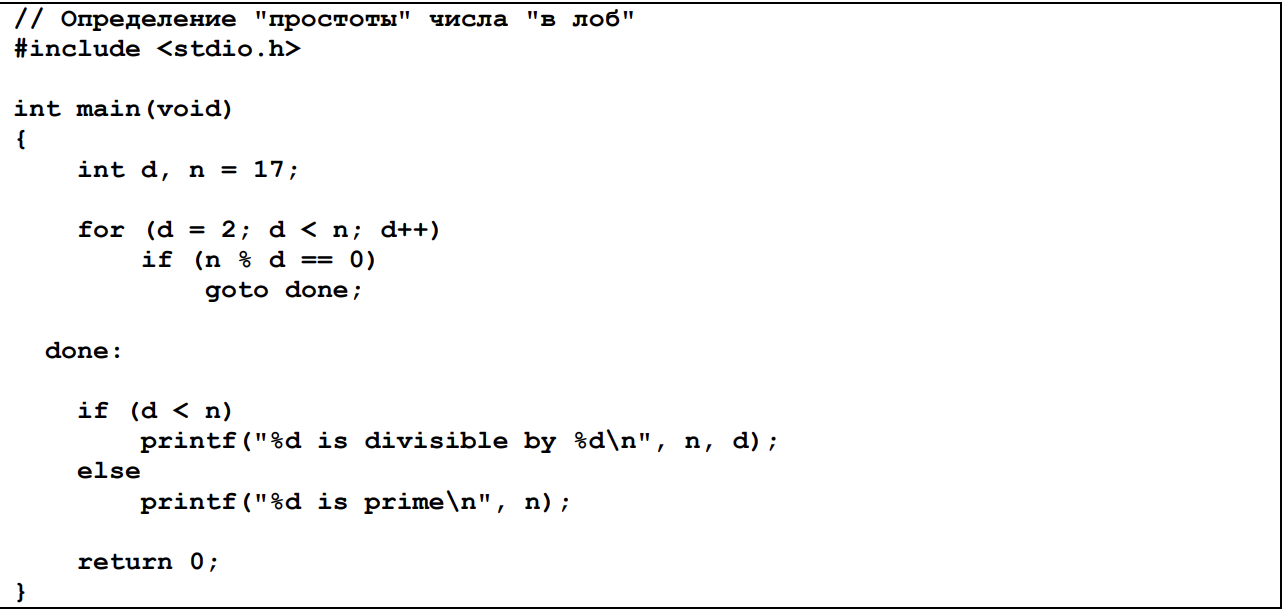
Оператор goto способен передать управление на любой оператор функции, помеченный меткой. Метка – это идентификатор, расположенный вначале оператора:

|  |
| --- |
| идентификатор : оператор |

Оператор может иметь более одной метки. Сам оператор goto записывается в форме

|  |
| --- |
| goto индетификатор; |

Выполнение оператора goto L; передает управление на оператор, который следует за меткой L. Метка должна располагаться в той же функции, что и оператор goto.



Считается, что оператор goto источник потенциальных неприятностей. Этот оператор на

практике практически никогда не бывает необходим и почти всегда легче обходится без него.

Есть несколько ситуаций, в которых без goto удобно использовать. Например, когда

необходимо сразу выйти из двух и более вложенных циклов.

# Что такое тип данных?

# Что такое статическая типизация? Что такое динамическая типизация?

При ***статической типизации*** конечные типы данных должны быть известны уже перед этапом компиляции, ***при динамической*** - тип данных определяется на этапе исполнения программы.

# Сравните статическую и динамическую типизации.

Динамически типизированный код быстрее писать, но сложнее отлаживать и сопровождать, так как проверка корректности типов происходит во время исполнения. Корректность типов в статически типизированном коде происходит до исполнения, что позволяет выявлять многие ошибки и, в случае компиляции, генерировать более оптимальный машинный код.

# Классификация типов данных в языке Си.

Обычно различают *простые* и *составные* типы. Переменные простого типа могут содержать

только одно значение, а переменные составного типа – несколько однотипных или

разнотипных значений.

Простые типы

В языке Си выделяются следующие простые типы:

1) целые \*int\*

2) вещественные \*float\*и др.

3) символьные \*char\*

4) перечислимый тип

5) void

6) указатели

Составными типы

В языке Си выделяются следующие простые типы:

1) массив

2) структура

3) объединение

# Целый тип в языке Си.

Модификаторов типа:

• signed

• unsigned

• short

• long

Диапазон значений, который представляет каждый из этих восьми типов, может изменяться

от компилятора к компилятору. Но в стандарте существуют два правила, которых должны

придерживаться все компиляторы:

• пункт 5.2.4.2 определяет минимальный диапазон значений (limits.h)

• пункт 6.3.1.1 определяет относительные размеры типов

sizeof(short int) <= sizeof(int) <= sizeof(long int) <= sizeof(long long int)

# Целочисленные константы в языке Си.

Целочисленные константы могут записываться в десятичной, восьмеричной и

шестнадцатеричной системах счисления.

Восьмеричные константы (обязательно начинаются с нуля)

Шестнадцатеричные константы (обязательно начинаются с 0x)

Тип целочисленных констант обычно int. Если значение константы слишком велико, чтобы

поместится в int, компилятор (который поддерживает стандарт C99) попытается определить

наименьший подходящий тип, в который оно уместится.

Чтобы заставить компилятор интерпретировать константу как long int необходимо добавить

суффикс L или l к значению (в случае типа long long int – LL или ll)

Чтобы показать, что константа беззнаковая, необходимо добавить суффикс U и u к значению

1ULL

# Целочисленные типы заданного размера.

Заголовочный файл **stdint.h** объявляет целочисленные типы, которые имеют заданный размер, имеют заданный минимальный размер и которые являются наиболее быстрыми при использовании

# Ввод/вывод целочисленных переменных.

Unsigned - u, o, x вместе d (10я, 8я, 16я с/с)

Short - h + (d, o, u, x)

Long – (l, ll) \_ (d, o, u, x)

# Вещественный тип в языке Си. Особенности сравнения вещественных переменных.

Стандарт С99 предоставляет следующие вещественные типы

float

double

long double

Константы, определяющие характеристики вещественных типов, могут быть найдены в

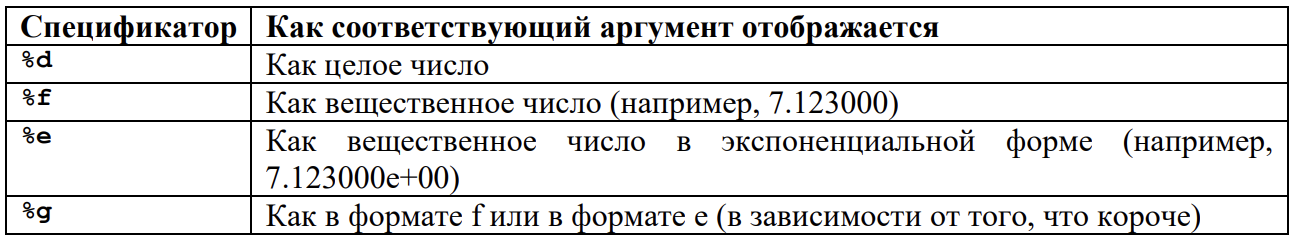
заголовочном файле float.h

# Вещественные константы в языке Си.

По умолчанию вещественные константы имеют тип \*double\*. Чтобы заставить компилятор интерпретировать константу как \*float\* необходимо добавить F (или f), а тобы как \*long double\* - L (или l)

# Ввод/вывод вещественных переменных.

Float



Double – lf

Long double – Lf

Для printf не обязателен l, L т к не имеет знаечния

# Символьный тип в языке Си.

Переменной типа char может быть присвоено значение любого ASCII символа

Под значение типа char отводится один байт.

Язык Си интерпретирует этот тип как «маленькое целое». В переменную типа char помещается

не сам символ, а его код из таблицы ASCII.

# Символьные константы в языке Си.

# Ввод/вывод символьных переменных. Функции для обработки отдельных символов.

%с

Стандартные функции для обработки отдельных символов объявляются в заголовочном файлу ctype.h.

#include *<ctype.h>*

int isalnum(int c); *//Если аргумент функции является либо буквой, либо цифрой, она возвращает ненулевое значение.*

int isalpha(int c); *//Возвращает ненулевое значение, если её аргумент является буквой, в противном случае возвращается нуль.*

int isblank(int c); *//Возвращает true, если с - пробел или горизонтальная табуляция (С99).*

int iscntrl(int c); *//Возвращает true, если с - управляющий символ, такой как <Ctrl+B>.*

int isdigit(int c); *//Возвращает ненулевое значение, если её аргумент является десятичной цифрой, в противном случае возвращается нуль.*

int isgraph(int c); *//Возвращает true, если с - печатаемый символ, отличный от пробела.*

int islower(int c); *//Возвращает true, если с - символ нижнего регистра.*

int isprint(int c); *//Возвращает true, если с — печатаемый символ.*

int ispunct(int c); *//Возвращает true, если с - знак препинания (любой печатаемый символ, отличный от пробела или алфавитно-цифрового символа).*

int isspace(int c); *//Возвращает true, если с — пробельный символ: пробел, новая строка,перевод страницы, возврат каретки, вертикальная табуляция, горизонтальная табуляция или, возможно, другой символ, определяемый реализацией*

int isupper(int c); *//Возвращает true, если с - символ верхнего регистра.*

int isxdigit(int c); *//Возвращает true, если с — шестнадцатеричная цифра.*

#include *<ctype.h>*

int toupper(int c); *// переводит буквы нижнего регистра в верхний регистр*

int tolower(int c); *// переводит буквы верхнего регистра в нижний регистр*

# Перечисляемый тип в языке Си.

В языке Си предусмотрел специальный тип, разработанный для переменных, которые

принимают небольшое количество значений. Это перечисляемый тип, значения которого

перечисляются программистом, который должен придумать имя для каждого значения.

|  |
| --- |
| enum season\_e {WINTER, SPRING, SUMMER, AUTUMN};  season\_e season; |

Язык Си интерпретирует перечисляемый тип и его «константы» как целые. По умолчанию

компилятор назначает значения 0, 1, 2, ... «константам» в каждом конкретном перечислении.

В нашем случае будут равны 0, 1, 2 и 3 соответственно.

Вы можете выбрать произвольные значения для констант.

|  |
| --- |
| enum season\_e {WINTER, SPRING, SUMMER, AUTUMN}; |

Поскольку значения перечисляемого типа — это целые числа, язык Си позволяет их

перемешивать с обычными целыми.

# Логический тип в языке Си (до и после стандарта C99).

\_Bool – это целочисленный тип (точнее беззнаковый целочисленный тип). В отличие от целочисленного типа переменные типа \_Bool могут принимать только значения 0 и 1. Попытка присвоить отличное от нуля значение переменной типа \_Bool приведет к тому, что значение этой переменной станет равным 1.

Переменные типа \_Bool могут использоваться в арифметических выражениях (но это не желательно).

Стандарт c99 предоставляет заголовочный файл stdbool.h, который облегчает использование «нового» логического типа. Это заголовочный файл содержит макрос bool для обозначения \_Bool, а также макросы true и false.

# Определение пользовательского типа (typedef). typedef vs define

Конструкция typedef позволяет задавать собственное имя для типа данных

|  |
| --- |
| typedef тип имя; |

Здесь "тип" - любой разрешенный тип данных и "имя" - любой разрешенный идентификатор.

Использование конструкции typedef позволяет

• улучшить читаемость программы

• улучшить сопровождение программы

|  |  |
| --- | --- |
| unsigned int n\_pens, n\_copybooks; | typedef unsigned int quantity\_t;  quantity\_t n\_pens, n\_copybooks; |

# Операция sizeof. Когда выполняется эта операция?

Унарная операция sizeof позволяет программе определить какое количество памяти требуется

для хранения величины определенного типа.

Возвращает беззнаковое целое, представляющее собой количество байт необходимых для

хранения величины типа type\_name

# Что такое неявное преобразование типа?

Машинные команды, как правило, обрабатывают операнды одного размера и одного типа, компилятор вынужден преобразовывать типы операндов так, чтобы подобрать подходящую машинную команду. Поскольку компилятор выполняет такие преобразования автоматически без участия программиста они называются неявными

# Когда происходит неявное преобразование типа? Примеры.

Неявное преобразование выполняется в следующих ситуациях:

* когда операнды в арифметическом или логическом выражении имеют разные типы (это

преобразование даже получило специальное название – обычное арифметическое

преобразование (С99 6.3.1.8));

Стратегия, которая лежит в основе арифметического преобразования, очень проста –

преобразовать операнды к самому «широкому» типу, который может содержать значения

обоих операндов. Такое преобразование иногда называют расширением типа.

Float – double – long double

Int – unsigned int – long int – unsigned long int

* когда тип выражения с правой стороны операции присваивания не совпадает с типом

выражения с левой стороны; ( приводит к типу с левой стороны)

* когда тип аргументов в вызове функции не соответствует типу параметра в ее

заголовке;

1. Прототип известен. Значение каждого аргумента неявно преобразуется к типу

соответствующего параметра как при присваивании.

2. Прототип неизвестен. Компилятор выполняет расширение по умолчанию: 1) float

преобразуется в double, 2) выполняется целочисленное расширение.

* когда тип выражения в операторе return не соответствует типу функции. (преобразовано к типу функции)

# Явное преобразование типа. Операция приведения типа. Примеры.

|  |
| --- |
| (имя типа) выражение |

# Что такое подпрограмма?

***Подпрограмма*** – именованная часть программы, содержащая описание определённого набора действий. Подпрограмма может быть многократно вызвана из разных частей программы

# Преимущества использования подпрограмм.

* Уменьшение сложности программирования за счет декомпозиции большой задачи на несколько подзадач меньшего размера, которые легче понимать.
* Сокрытие деталей реализации от пользователей подпрограммы.
* Уменьшение дублированного кода.
* Возможность повторного использования кода в других программах.

# Виды подпрограмм.

**Функция** – это подпрограмма специального вида, которая всегда должна возвращать результат. Вызов функции является, с точки зрения языка программирования, выражением.

**Процедура** – это независимая именованная часть программы, которую после однократного описания можно многократно вызвать по имени из последующих частей программы для выполнения определенных действий.

# Что такое заголовок подпрограммы?

Заголовок подпрограммы описывает её имя и параметры, т.е. информацию, необходимую для вызова подпрограммы.

# Что такое тело подпрограммы?

Тело подпрограммы – набор операторов, который будет выполнен всякий раз, когда подпрограмма будет вызвана.

# Вызов подпрограммы. Формальные и фактические параметры.

Вызов подпрограммы выполняется с помощью команды вызова, включающей в себя имя подпрограммы, за которым следуют параметры.

Чтобы отличать параметры подпрограммы, описанные в её заголовке и теле, от параметров, указываемых при вызове подпрограммы, используются формальные и фактические параметры. **Формальные параметры** указываются при описании подпрограммы, а **фактические** – непосредственно при её вызове. Фактические параметры часто называют **аргументами**.

# Способы передачи параметров в подпрограмму. Реализация этих способов применительно к языку Си.

Существуют две концептуальные модели передачи данных при передаче параметров:

- либо фактическое значение физически перемещается (**передача по значению**, англ. pass by value),

- либо передается путь доступа к нему (**передача по ссылке**, англ. pass by reference).

# Описание функции на языке Си.

В языках высокого уровня описание подпрограммы обычно состоит из двух частей: заголовка и тела.

# Заголовок функции языке Си.

Заголовок подпрограммы описывает её имя, параметры и возвращаемое значение т.е. информацию, необходимую для вызова подпрограммы.

# Тело функции на языке Си.

* У каждой функции есть исполнимая часть, которая называется телом функции и заключена в фигурные скобки.
* Тело функции может содержать как описания переменных, так и операторы. Переменные, описанные в теле функции, «принадлежат» только этой функции и не могут быть ни получены, ни изменены другой функцией.
* Тело функции не может содержать в себе определения других функций (т.е. в языке Си нет вложенных функций).
* Если функция ничего не возвращает, ее тело может быть пустым.

# Область видимости локальных переменных и параметров функции.

Только внутри функции

# Оператор return: назначение, использование.

* Оператор return прерывает выполнение функции и возвращает вычисленное значение и управление в ту часть программы, из которой эта функция была вызвана.
* Оператор return может использоваться в функциях, которые ничего не возвращают. В этом случае он записывается без указания выражения.
* Помещать оператор return в конец функции, которая ничего не возвращает можно, но не имеет никакого смысла.

# Вызов функции на языке Си.

# Что такое объявление? Сколько объявлений «объекта» может быть в программе?

Объявление (англ. declaration) – это инструкция компилятору, как использовать указанное имя (описывает свойства переменной или функции). Объявлений одного и того же имени может быть сколько угодно, главное чтобы они все были согласованы (т.е. одинаковы).

# Что такое определение? Сколько определений «объекта» может быть в программе?

Определение (англ. definition) осуществляет привязку имени к сущности (к памяти для данных или к коду для функций), т.е. специфицирует код или данные, которые стоят за этим именем. В языке Си существует правило единственного определения. (?) Это означает, что определение может быть только одно.

# Взаимное расположение объявлений, определения и использования «объекта».

# Объявление функции на языке Си.

Объявление функции состоит из заголовка функции

|  |
| --- |
| тип-результата имя-функции(список формальных параметров с их типами); |

Объявление функции должно соответствовать ее определению.

Объявление функции может не содержать имен параметров. Однако их обычно оставляют для большей наглядности.

# Определение функции на языке Си.

тип имя\_функции(параметры)

{

    инструкции

}

# Особенности описания функций без параметров в языке Си.

Тип имя\_функции(void){}

# Способ передачи параметров в функцию на языке Си.

бфыло

# Выполнение вызова функции.

1. Выделяется область памяти доступная только вызываемой функции (power).
2. В этой области памяти создаются переменные-параметры (base, n) и локальные переменные (res).
3. Переменным-параметрам присваиваются начальные значения, в качестве которых выступают аргументы из точки вызова (base = 2, n = 5).
4. Инициализируются локальные переменные (res = 1). В случае отсутствия такой инициализации локальные переменные содержат «мусор».
5. Выполняется тело функции.
6. Вычисленное значение возвращается из функции (в нашем случае попадает в переменную a).
7. Выделенная область памяти разрушается.

# Что такое чистая функция?

**Чистая функция** - это функция, которая 1) является детерминированной, 2) не обладает побочными эффектами.

Функция является **детерминированной**, если для одного и того же набора входных значений она возвращает одинаковый результат.

Функции с **побочными эффектами** – это функции, которые в процессе выполнения своих вычислений могут модифицировать значения глобальных переменных, осуществлять операции ввода/вывода. Другим видом побочных эффектов является модификация переданных в функцию параметров.

# Способы возврата значения из функции на языке Си.

* Функция может вернуть значение любого типа кроме массива.
* Если функция ничего не возвращает, то в качестве возвращаемого значения следует указать void.
* Если тип возвращаемого значение не указана, то согласно стандарту C89, компилятор предполагает, что возвращается значение целого типа. Согласно стандарту С99, тип возвращаемого значения опускать нельзя.
* Для возврата значения используется оператор return.

# Возвращение нескольких значений из функции на языке Си.

Указатели можно использовать, чтобы вернуть несколько значений из функции. В таком случае в функцию передаётся несколько указателей на переменные, а в самой функции меняется их значение.

# Какая функция называется рекурсивной?

Функция называется рекурсивной, если она вызывает саму себя.

# Преимущества и недостатки рекурсивных функций.

Преимущества

* Рекурсивная форма может быть структурно проще и нагляднее, в особенности, когда сам реализуемый алгоритм рекурсивен.

Недостатки

* Рекурсивный вызов использует больше памяти, поскольку создает свой набор переменных.
* Рекурсия выполняется медленней, поскольку на каждый вызов функции требуется определенное время.

# Что такое «хвостовая рекурсия»? Особенности этого вида рекурсии.

Хвостовая рекурсия - частный случай рекурсии, при котором любой рекурсивный вызов является последней операцией перед возвратом из функции.

Подобный вид рекурсии может быть легко заменён на итерацию путём формальной и гарантированно корректной перестройки кода функции. Оптимизация хвостовой рекурсии путём преобразования её в итерацию реализована во многих оптимизирующих компиляторах.

# Что такое массив? Основные свойства массива.

Массив – последовательность элементов одного и того же типа, расположенных в памяти друг за другом.

# Особенности описания статических массивов на языке Си.

* Тип элемента может быть любым.
* Количество элементов указывается целочисленным константным выражением.
* Количество элементов не может быть изменено в ходе выполнения программы.

# Инициализация статических массивов на языке Си (в том числе и в стандарте C99).

Инициализация представляет собой набор начальных значений элементов массива, указанных в фигурных скобках, и разделенных запятыми.

Если количество инициализирующих значений, указанных в фигурных скобках, меньше, чем количество элементов массива, указанное в квадратных скобках, то все оставшиеся элементы в массиве (для которых не хватило инициализирующих значений) будут равны нулю. Это свойство удобно использовать для задания нулевых значений всем элементам массива.

Если массив проинициализирован при объявлении, то константные начальные значения его элементов указываются через запятую в фигурных скобках. В этом случае количество элементов в квадратных скобках может быть опущено.

|  |
| --- |
| int e[10] = {0, 0, 5, 0, 0, 10, 0, 0, 15, 0};  // C99  int e[10] = {[2] = 5, [5] = 10, [8] = 15}; |

# Операция индексации.

Адрес элемента

* &a[i]
* pa + i

Значение элемента

* a[i]
* \*(pa + i)

&a[i] = = = a + i

Выражение a[i] компилятор заменяет на \*(a + i)

# Особенности передачи массива в функцию на языке Си.

Любое похожее на массив объявление параметра функции рассматривается компилятором как указатель.

Int f(int a[N]); int f(int a[]); int f(int \*a);

Функция не может узнать размер массива

Функция может изменять значения элементов массива

# Организация оперативной памяти с точки зрения прикладного программиста.!!!!!!!!!!!!!!!!

# Дайте определение минимальной единице адресации?

# Что такое машинное слово?

# Что такое little endian/big endian?

# Что такое указатель?

Указатель – это объект, содержащий адрес объекта или функции, либо выражение, обозначающее адрес объекта или функции.

# Разновидности указателей в языке Си.

* типизированный указатель на данные (тип\* имя);
* бестиповой указатель (void\* имя);
* указатель на функцию.

# Использование указателей в языке Си.

* Передача параметров в функцию
  + Изменяемые параметры
  + «Объемные» параметры
* Обработка областей памяти
  + Динамическое выделение памяти
  + Ссылочные структуры данных

# Базовые операции для работы с указателями.

Определение int \*p;

Разыменование \*p;

Взятие адреса у обычной переменной &a;

# Инициализация указателей.

.

# Константа NULL.

Нулевой указатель

# Модификатор const и указатели.

Указатель на константу – const int \*p; (error: \*p = b)

Константный указатель – int \* const p; (error: p = &b)

Константный указатель на константу – const int \* const p;

# Выражение из имени массива. Исключения из этого правила.

Результат выражения, состоящего из имени массива, представляет собой адрес области памяти, выделенной под этот массив

Int a[10];

a, &a[0] – указатель на первый элемент массива (int \*)

&a – указатель на массив (int (\*)[10])

Исключения:

* Sizeof(a) – вычисляет размер массива
* a, &a – возвращают один и тот же адрес, но том указателя разный
* массив не распадается в указатель, когда является строковым литералом char a[] = “qwer”

# Можно ли отождествлять массивы и указатели?

Нет

Отличия

* При определении массива и указателя под соответствующие переменные выделяется разное количество памяти
* Выражению из имени массива нельзя присвоить другое значение

# Сложение указателя с целым числом.

тип \*p = …;

p += n;

новый адрес в p = старый адрес из p + n \* sizeof(тип)

p -= m;

новый адрес в p = старый адрес из p - m \* sizeof(тип)

# Сравнение указателей.

Указатели допускается использовать в операциях сравнения.

При сравнении указателей сравниваются адреса.

При этом можно

* сравнивать указатель с NULL;
* сравнивать два однотипных указателя.

# Вычитание указателей.

pb – pa = (адрес из pb – адрес из pa) / sizeof(тип)

**ptrdiff\_t** – знаковый целый тип, создан для хранения разности между двумя указателями. Разрядность типа ptrdiff\_t зависит от платформы и реализации компилятора.

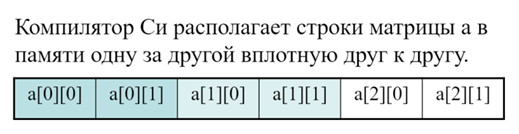
Тип определен в заголовочном файле stddef.h.

# «Концепция» многомерного массива в языке Си.

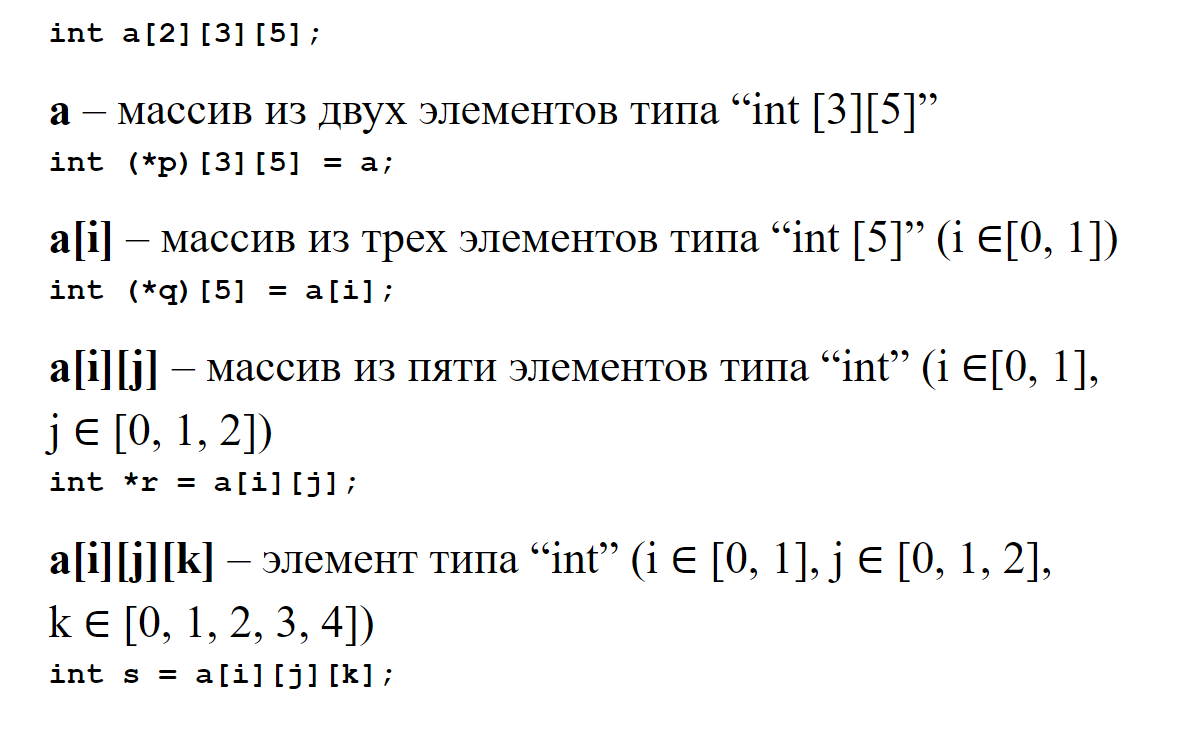
В языке Си нет специального типа для описания многомерных массивов. Массив размерности “n” рассматривается как одномерный массив, элементами которого являются массивы размерности “n – 1”. Количество размерностей многомерного массива практически не ограничено (оно определяется реализацией компилятора).

# Описание многомерного массива на языке Си. Особенности расположения в памяти.

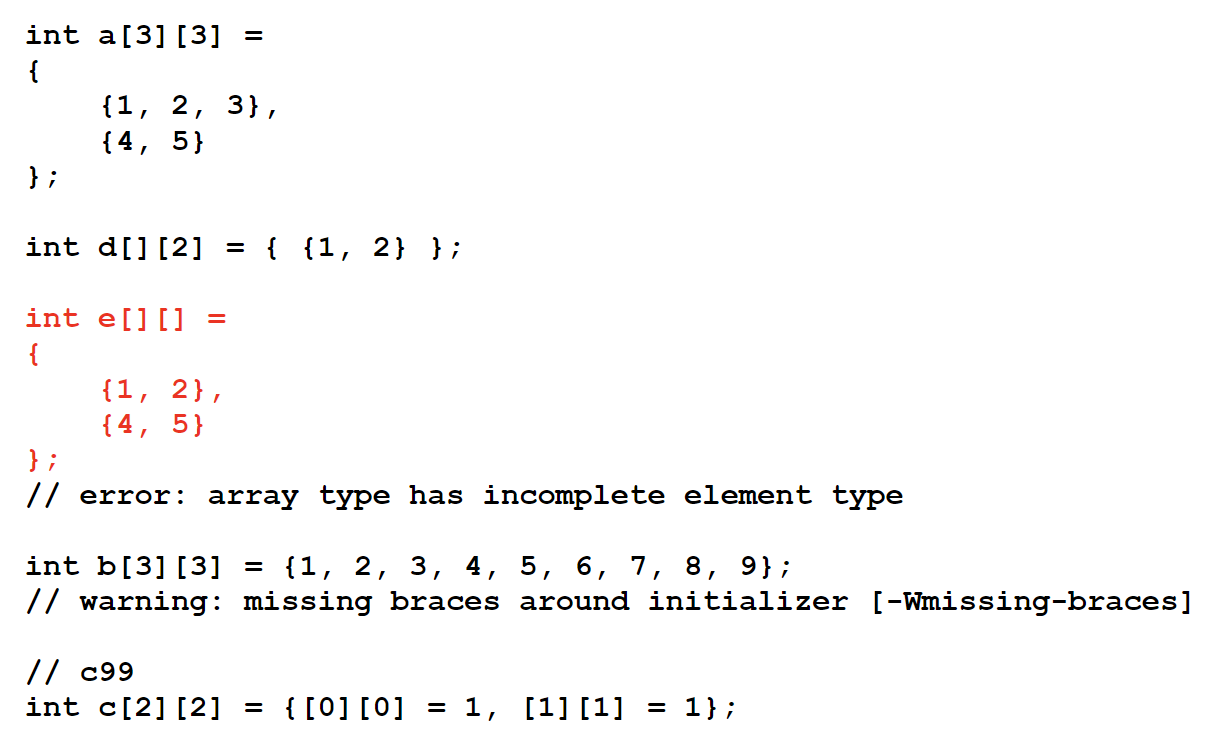
матрица – это одномерный массив, элементами которого являются одномерные массивы (строки);



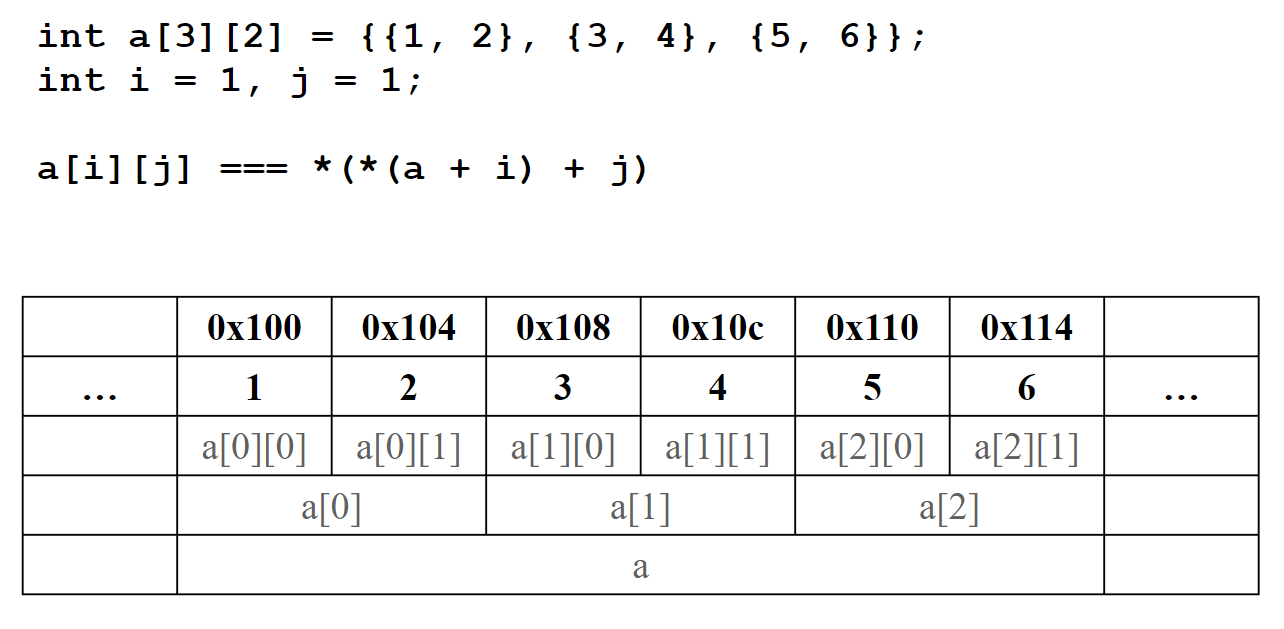
# «Слои» многомерного массива в языке Си.

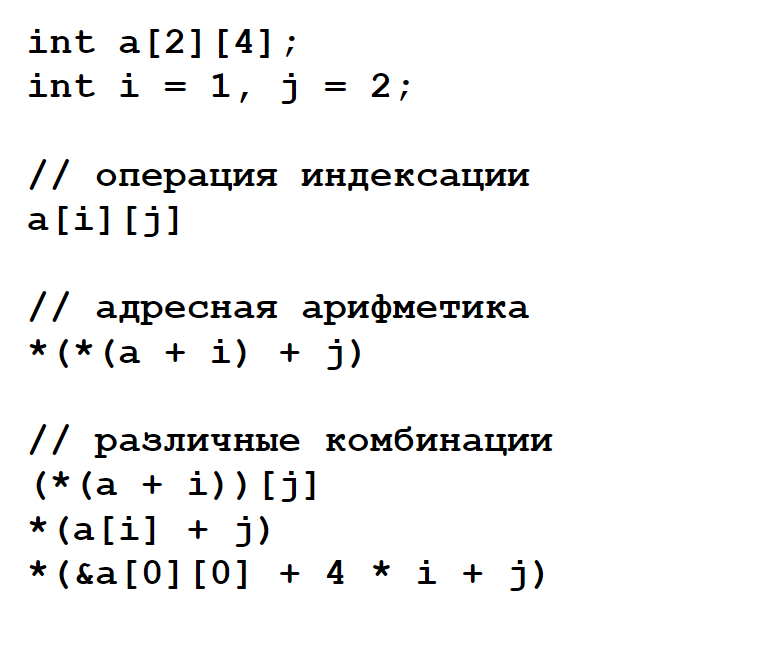


# Инициализация многомерных массивов на языке Си.



# Доступ к элементу многомерного массива в языке Си.

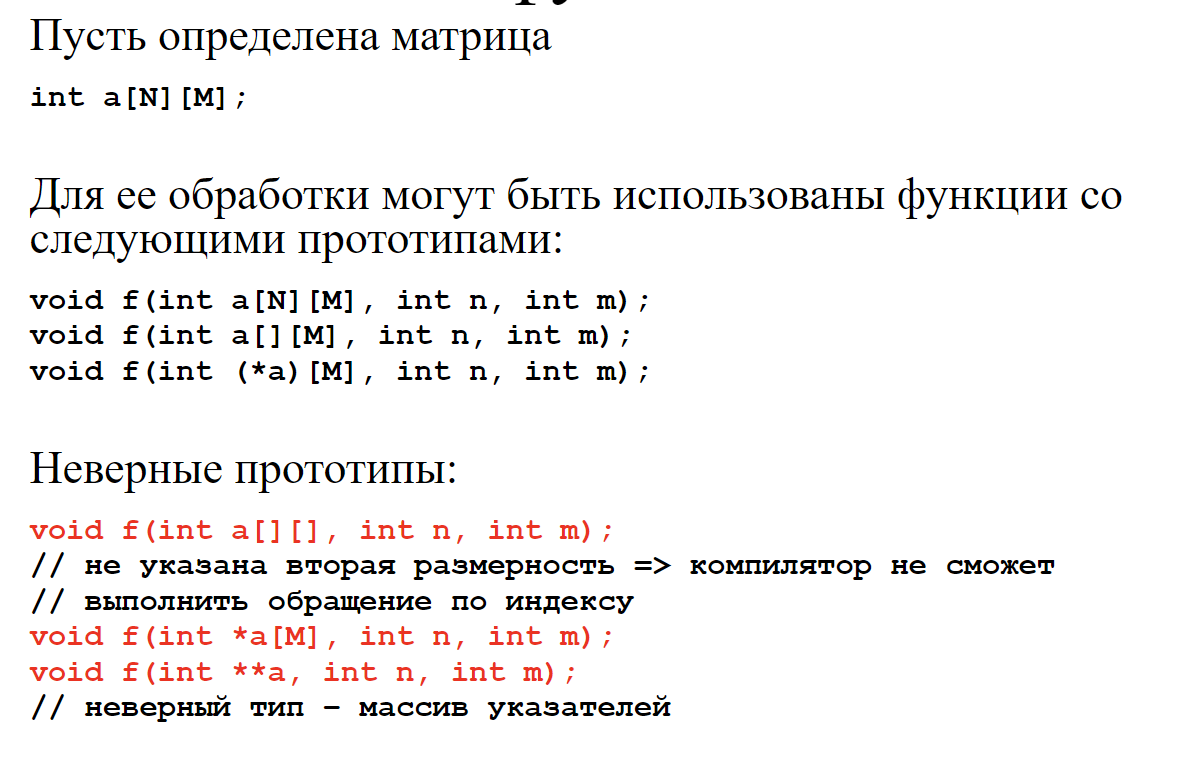




# Обработка многомерных массивов с помощью указателей.

.

# Передача многомерного массива в функцию.



# Особенности использования const и многомерных массивов в языке Си.

Компилятор не может неявным образом преобразовать указатель на массива в указатель на константный массив

Способы борьбы

* не использовать const;
* использовать явное преобразование типа

print((const int (\*)[M]) a, 2, 3);

# Что такое строка в языке Си? Преимущества и недостатки такого представления.

Строка – это последовательность символов, заканчивающаяся и включающая первый нулевой символ (англ., null character ‘\0’ (символ с кодом 0)).

строка – массив символов, для доступа к элементу строки может использоваться операция индексации:

Преимущества подхода

* Простота.

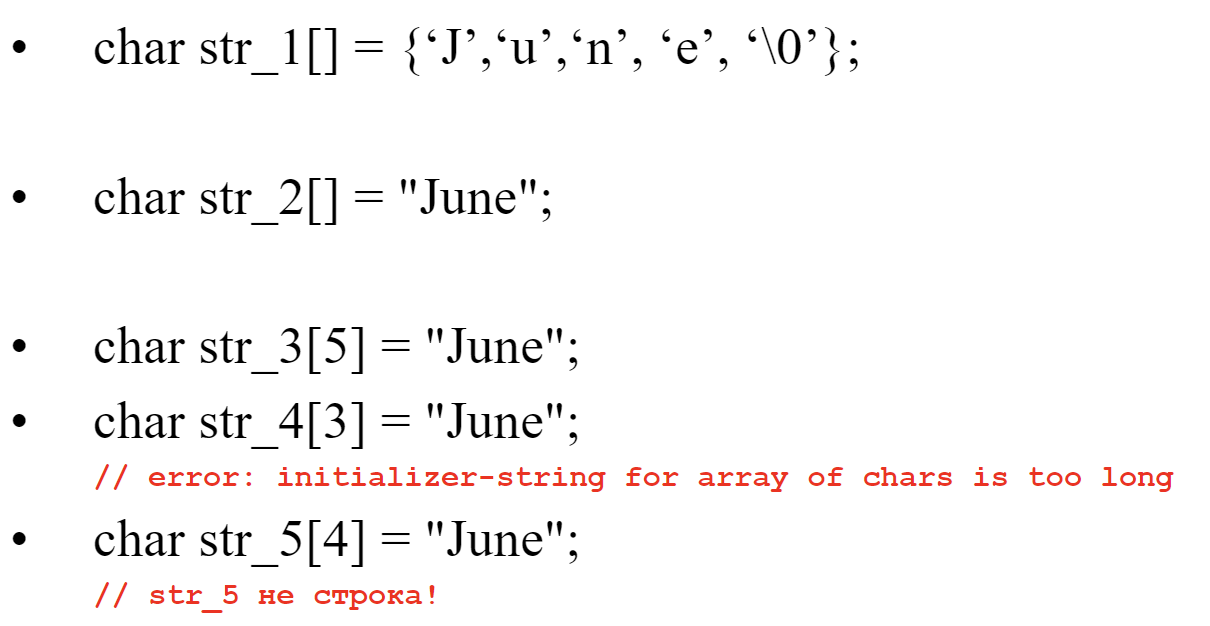
Недостатки подхода

* Отсутствие быстрого способа определения длины строки.
* Тщательность при работе с нулевым символом.

# Описание строки на языке Си.

# Особенности передачи строк в функцию в языке Си.

# Особенности инициализации строковых переменных.



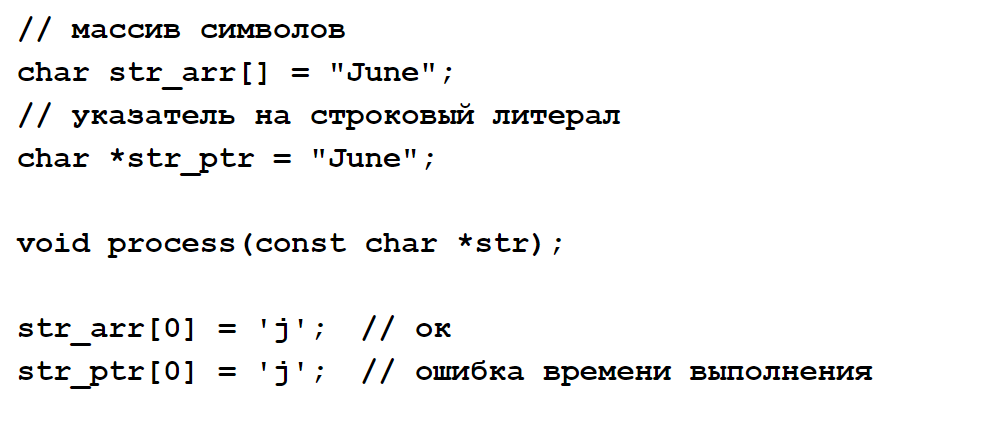
# Что такое строковый литерал?

Строковый литерал – последовательность символов, заключенных в двойные кавычки.

Строковый литерал рассматривается компилятором как массив элементов типа char. Когда компилятор встречает строковый литерал из n символов, он выделяет n+1 байт памяти, которые заполняет символами строкового литерала и завершает нулевым символом.

В стандарте сказано, что поведение программы не определено при попытке изменить строковый литерал.

# Указатель на строковый литерал и на строку.



Указатель на строковый литерал – это указатель на константу

# Способы описания «массива строк» на языке Си.

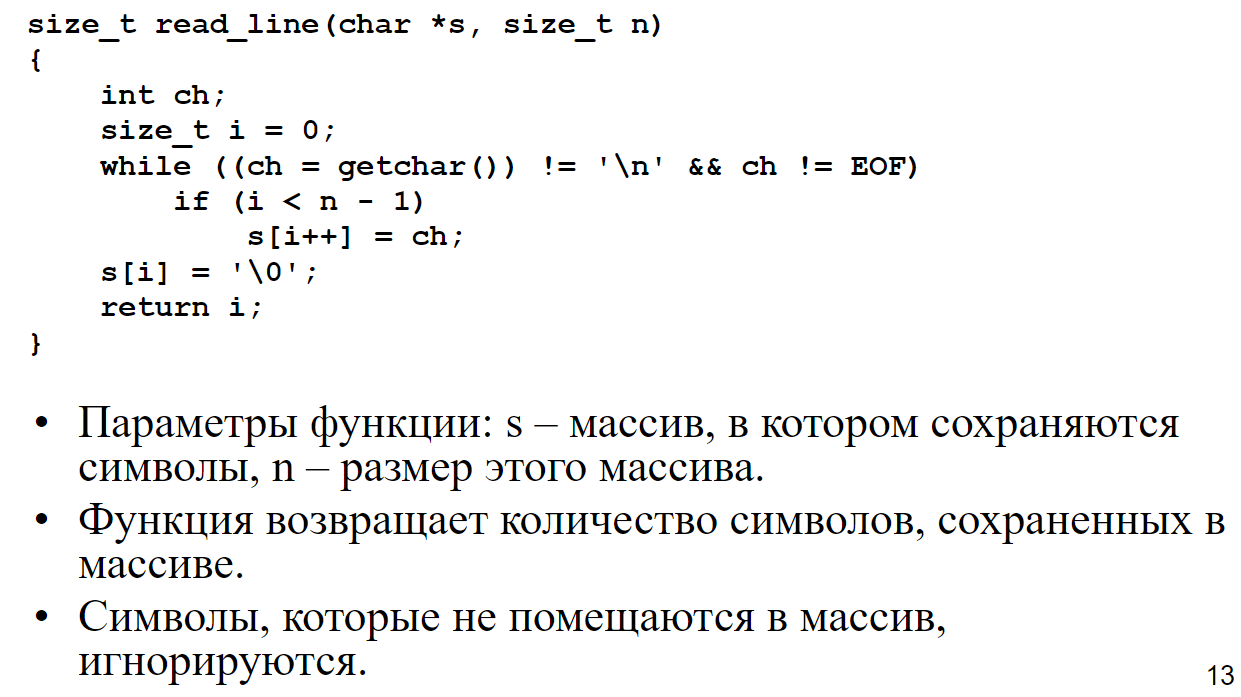


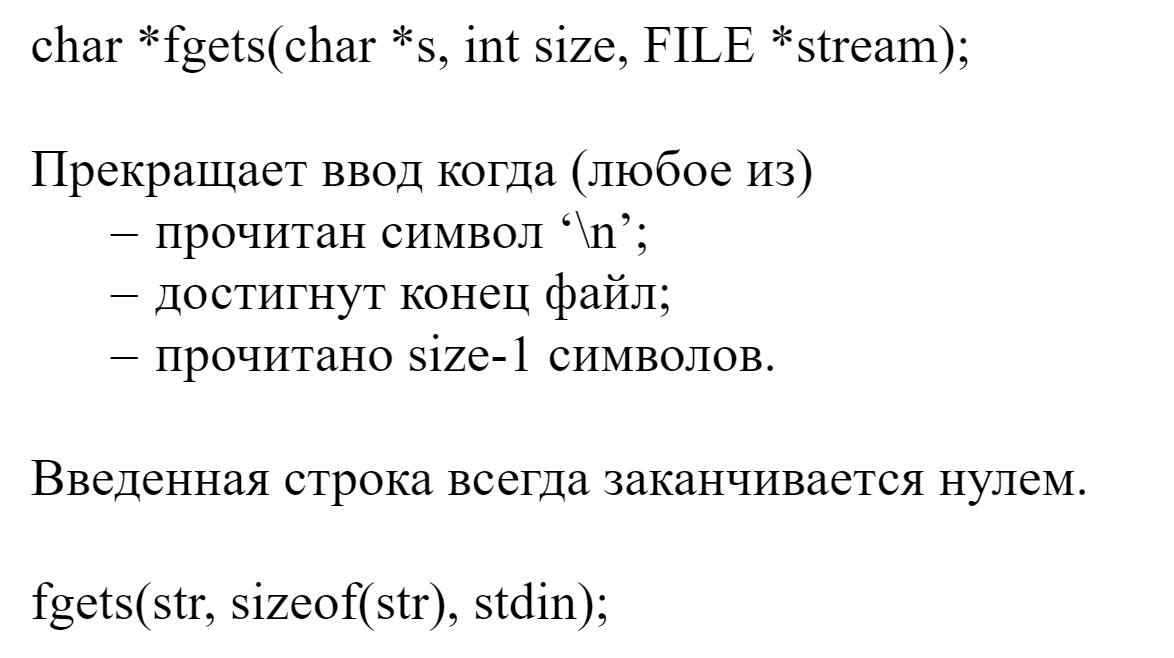
Способ 2: доп траты на указатели (памыти)

# Ввод/вывод строк.

Вывод %s

Собственный вод через getchar()





scanf("%s", str);

// Через scanf нельзя ввести строку с пробелами!

# Обработка строк (strcpy, strcat, strlen, strcmp, snprintf, strtok, перевод строки в число).

**strcpy – копирование строк.**

**Синтаксис:**

#include < string.h >  
char \*strcpy (char \*destination, const char \*source);

**Аргументы:**

destination – указатель на строку, в которую будут скопированы данные.  
source – указатель на строку источник копируемых данных.

**Возвращаемое значение:**

Функция возвращает указатель на строку, в которую скопированы данные.

**Описание:**

Функция strcpy копирует данные из строки, на которую указывает аргумент source, в строку, на которую указывает аргумент destination, пока не встретится символ конца строки (нулевой символ). Копирование производится вместе с символом конца строки.  
  
Если строки перекрываются, результат копирования будет не определен.

**strncpy – копирование строк c ограничением длины.**

**Синтаксис:**

#include < string.h >  
char \*strncpy (char \*destination, const char \*source, size\_t n);

**Аргументы:**

destination – указатель на строку в которую будут скопированы данные.  
source – указатель на строку источник копируемых данных.  
n – ограничение длинны копирования

**Возвращаемое значение:**

Функция возвращает указатель на строку, в которую скопированы данные.

**Описание:**

Функция strncpy копирует данные из строки, на которую указывает аргумент source, в строку на который указывает аргумент destination, пока не встретится символ конца строки (нулевой символ) или количество скопированных символов не достигнет величины указанной в аргументе n. При этом символ конца строки не копируется.  
  
Если в аргументе n указана величина большая, чем длинна строки, т.е. символ конца строки '\000' встретился раньше, чем было скопировано n символов, то в массив, на который указывает аргумент destination, дозаписывается столько нулевых символов, чтобы общая длина записанной строки достигла величины указанной в аргументе n.  
  
Если строки перекрываются, результат копирования будет не определен.

**strcat – объединение строк.**

**Синтаксис:**

#include < string.h >  
char \*strcat (char \*destination, const char \*append);

**Аргументы:**

destination – указатель на массив в который будет добавлена строка.  
append – указатель на массив из которого будет скопирована строка.

**Возвращаемое значение:**

Функция возвращает указатель на массив, в который добавлена строка (destination).

**Описание:**

Функция strcat добавляет в строку, на которую указывает аргумент destination, строку, на которую указывает аргумент append. Символ конца строки помещается в конце объединенных строк.  
  
Если строки перекрываются, результат объединения будет не определен.

**strncat – объединение строк c ограничением длины добавляемой строки.**

**Синтаксис:**

#include < string.h >  
char \*strncat (char \*destination, const char \*append, size\_t n);

**Аргументы:**

destination – указатель на массив в который будет добавлена строка.  
append – указатель на массив из которого будет скопирована строка.  
n – максимальная длина добавляемой строки.

**Возвращаемое значение:**

Функция возвращает указатель на массив, в который добавлена строка (destination).

**Описание:**

Функция strncat добавляет в строку, на которую указывает аргумент destination, строку, на которую указывает аргумент append, пока не встретится символ конца строки или пока не будет добавлено n символов.  
  
Символ конца строки помещается в конце объединенных строк.  
  
Если строки перекрываются, результат объединения будет не определен.

**strlen – определение длины строки.**

**Синтаксис:**

#include < string.h >  
size\_t \*strlen (const char \*str);

**Аргументы:**

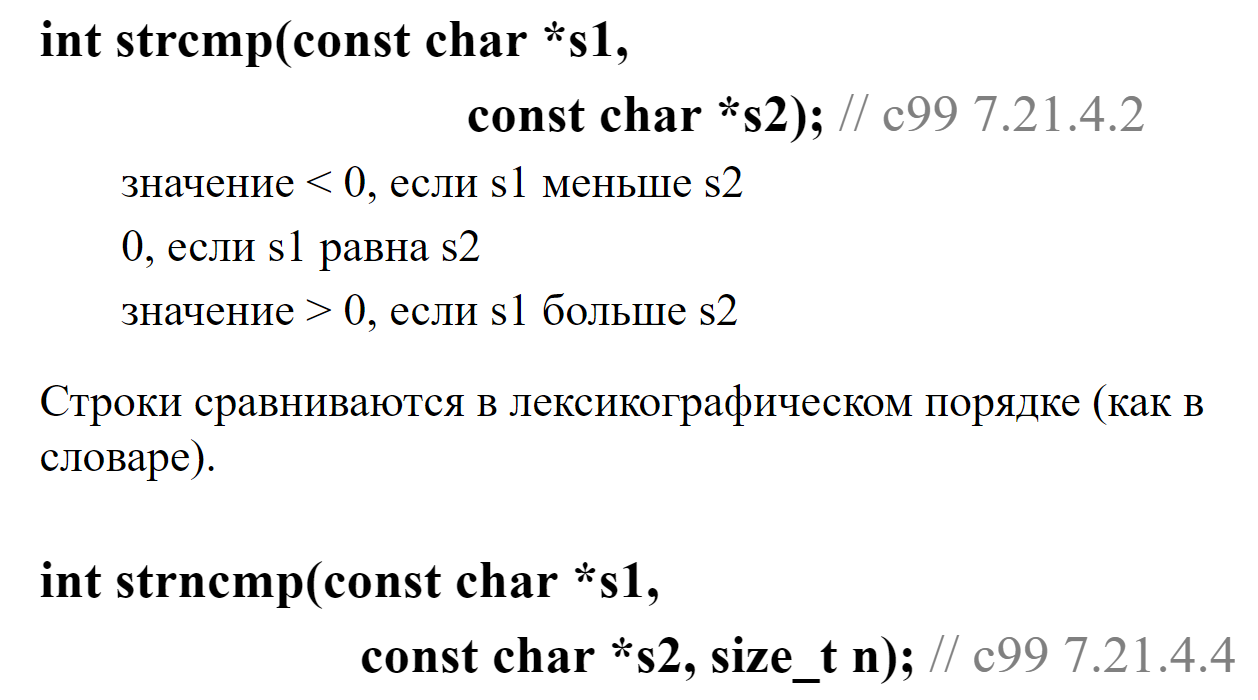
str – указатель на строку.

**Возвращаемое значение:**

Количество символов в строке до первого вхождения символа конца строки.

**Описание:**

Функция strlen вычисляет количество символов в строке до первого вхождения символа конца строки. При этом символ конца строки не входит в подсчитанное количество символов.



Функция strcmp сравнивает символы, сравнивая значения кодов, которые представляют эти символы.

Функция возвращает нечеткие значение, как особенность реализации. Т к в одном из первых изданий Kernighan, Ritchie, она возвращает разность непохожих символов

int sprintf(char \*str, const char \*format, ...) отправляет форматированный вывод в строку, на которую указывает str.

int snprintf(char \*s, size\_t n, const char \*format, ...);

**strtok – разбиение строки на части по указанному разделителю.**

**Синтаксис:**

#include < string.h >  
char \*strtok(char \*str, const char \*sep);

**Аргументы:**

str – указатель на разбиваемую строку.  
sep – указатель на строку, содержащую набор символов разделителей.

**Возвращаемое значение:**

NULL – если строку str невозможно разделить на части.  
Указатель на первый символ выделенной части строки.

**Описание:**

Функция strtok выделяет очередную часть строки, на которую указывает аргумент str, отделенную одним из символов разделителей указанных в строке, на которую указывает аргумент sep. Последовательный вызов функции strtok приводит к разбиению строки str на части (лексемы).  
  
«При первом вызове функции strtok указывается начало разделяемой строки (str) и начало строки, содержащей разделители (sep). В начале функция strtok поочередно просматривает символы строки str и ищет символ, не содержащейся в строке разделителей sep. Если в строке str символ конца строки встречен раньше, чем был найден символ не входящий в строку sep, то разделить строку str на части нельзя и возвращается нулевой указатель (NULL). Если такой символ найден, он считается началом первой части строки str.»  
  
Далее функция strtok ищет разделитель, то есть символ, входящий в строку sep. Если такой символ не найден, то считается, что строка str состоит из одной части и последующее разделения строки str будут возвращать нулевой указатель. Если такой символ найден. то он заменяется нулевым символом (символом конца строки). Далее функция strtok запоминает текущую позицию (указатель на символ, с которого будет начинаться поиск следующей части строки) и возвращает указатель на начало первой выделенной части строки.  
  
Если функция strtok вернула не нулевой указатель, можно продолжить разбиение строки str на части. Для продолжения разбиения строки, повторно вызывается функция strtok, но вместо указателя на разбиваемую строку в качестве первого аугмента указывается NULL. В этом случае функция strtok продолжит разбиение с запомненного адреса. Алгоритм разбиения при этом останется тот же.



# Что такое лексикографический порядок слов?

Алфавитный порядок

# Что такое структура в языке Си?

# Описание структуры в языке Си.

# Что такое тег структуры? Для чего он используется?

# Что такое поле структуры? Типы полей структуры. Описание полей структуры.

# Особенности именования тегов и полей структуры.

# Расположение полей структуры в памяти. Выравнивание. Упаковка.

# Способы описания переменных структурного типа.

# Инициализация переменных структурного типа.

# Операции над структурами.

# Что такое объединение?

# Расположение полей объединения в памяти.

# Инициализация объединений.

# Использование объединений.

# Сравните структуру и объединение.

Что такое файл?

Перечислите основные свойства файла.

Что такое файловая система?

Назначение файловой системы?

Классификация файлов.

Текстовые файлы.

Двоичные файлы.

Сравните текстовые и двоичные файлы.

Последовательность действий при работе с файлами (fopen, fclose, feof, ferror).

Основные функции для обработки текстовых файлов (fprintf, fscanf, fgets, rewind).

Основные функции для обработки двоичных файлов (fread, fwrite, fseek, ftell).

Предопределенные файловые переменные.

Организация работы с ресурсами.

Что такое транслятор?

Что такое интерпретатор?

Что такое препроцессор?

Основные шаги получения исполняемого файла.

Что такое объектный файл?

Что такое таблица символов?

Что такое таблица релокации?

Что такое исполняемый файл?

Преимущества многофайловой организации проекта.

Для чего нужны заголовочные файлы?

Что такое include guard? Для чего нужна эта конструкция? Как она работает?

Ошибки компиляции.

Ошибки компоновки.

Предупреждения компилятора.

Что такое связывание?

Какие виды связывания есть в языке Си?

Как связывание влияет на "свойства"объектного/исполняемого файла? Что это за

"свойства"?

Какими характеристиками (область видимости, время жизни, связывание) обладает

переменная в зависимости от места своего определения?

Какими характеристиками (область видимости, время жизни, связывание) обладает

функция в зависимости от места своего определения?

Какие классы памяти есть в языке Си?

Для чего нужны классы памяти?

Какие классы памяти можно использовать с переменными? С функциями?

Сколько классов памяти может быть у переменной? У функции?

Какие классы памяти по умолчанию есть у переменной? У функции?

Расскажите о классе памяти auto.

Расскажите о классе памяти static.

Расскажите о классе памяти extern.

Расскажите о классе памяти register.

Для чего используется ключевое слово extern?

Особенности совместного использования ключевых слов static и extern.

Какими недостатки есть у использования глобальных переменных?

Журналирование, подходы к реализации

Битовые операции

Битовые поля

Битовые операции vs битовые поля

Библиотеки (определение, классификация, статические библиотеки vs динамические библиотеки)

Процесс запуска исполняемого файла в ОС

Абстрактное адресное пространство программы

Аппаратный стек и его использование в программе на Си