**Оглавление**

[Типы данных языка С 2](#_Toc166868681)

[Компиляция программы на Linux 3](#_Toc166868682)

[Функция форматного вывода printf 3](#_Toc166868683)

[Функция ввода данных scanf 5](#_Toc166868684)

[Математические функции 6](#_Toc166868685)

[Рандом 6](#_Toc166868686)

[Оператор switch 7](#_Toc166868687)

[Условный оператор if-else 7](#_Toc166868688)

[Цикл FOR 8](#_Toc166868689)

[Цикл с условием WHILE 9](#_Toc166868690)

[Массивы 10](#_Toc166868691)

[Двумерные массивы 10](#_Toc166868692)

[Строки 11](#_Toc166868693)

[Вывод строки на экран 11](#_Toc166868694)

[Ввод строк 11](#_Toc166868695)

[Функции работы со строками 12](#_Toc166868696)

[Функции 16](#_Toc166868697)

[Указатели 17](#_Toc166868698)

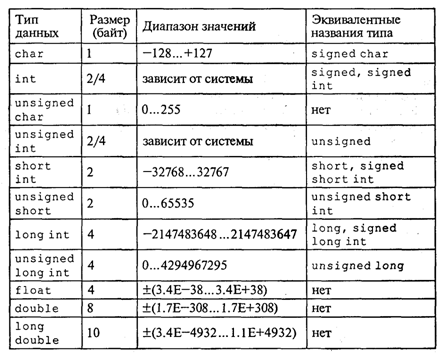
[Адресная арифметика 17](#_Toc166868699)

[Const с указателями 17](#_Toc166868700)

[Передача аргументов в функцию 18](#_Toc166868701)

[Передача массива в функцию 18](#_Toc166868702)

Типы данных языка С



Компиляция программы на Linux

1 Шаг: Установить необходимые компиляторы GCC и G++

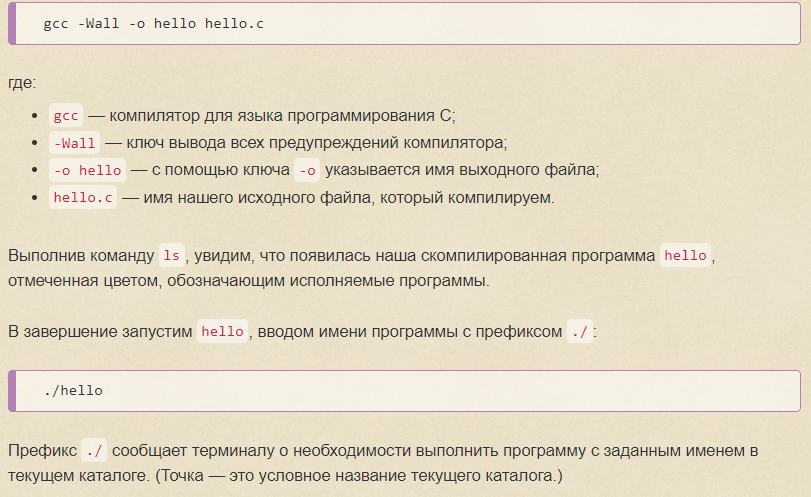
Sudo apt install build-essential

Если ошибки: попробовать обновить пакетный менеджер через команду

Apt-get update

2 Шаг: Написать программу на языке C. Расширение должно быть .c

3 Шаг: Перейти в папку с программой и скомпилировать ее



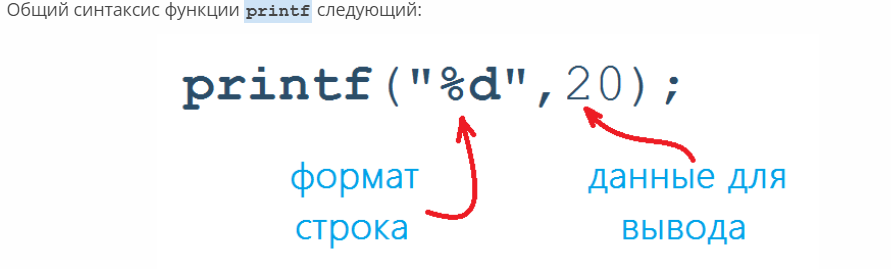
Функция форматного вывода printf

**Полезно : https://microsin.net/programming/arm/secrets-of-printf.html**

Для ввода и вывода информации нужно подключить заголовочный файл

#include <stdio.h>

Он определяет поток ввода (stdin) и поток вывода (stdout)



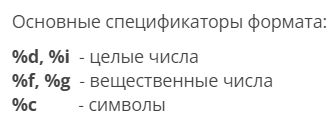
Формат строка может состоять из:

- символы, которые выводятся на экран без изменений

- escape-последовательности (табуляция, перевод каретки)

- спецификаторы формата (для вывода переменных)

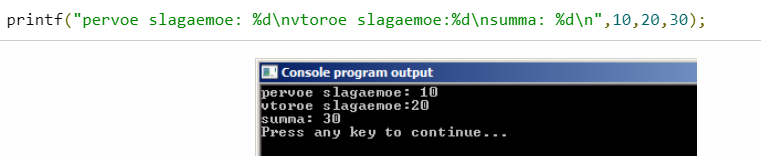
**Спецификаторы формата:**



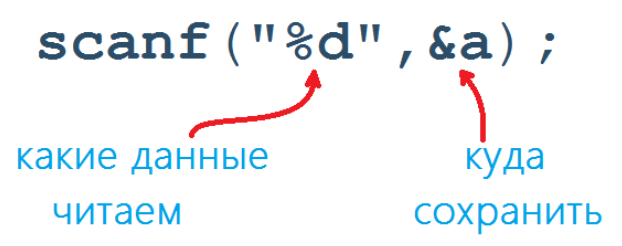
Также есть **модификаторы формата**

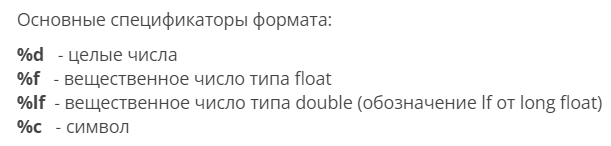
Например %8.3f – вещественное число из 8 символов с точностью 3 знака после запятой.

Пример:



Функция ввода данных scanf



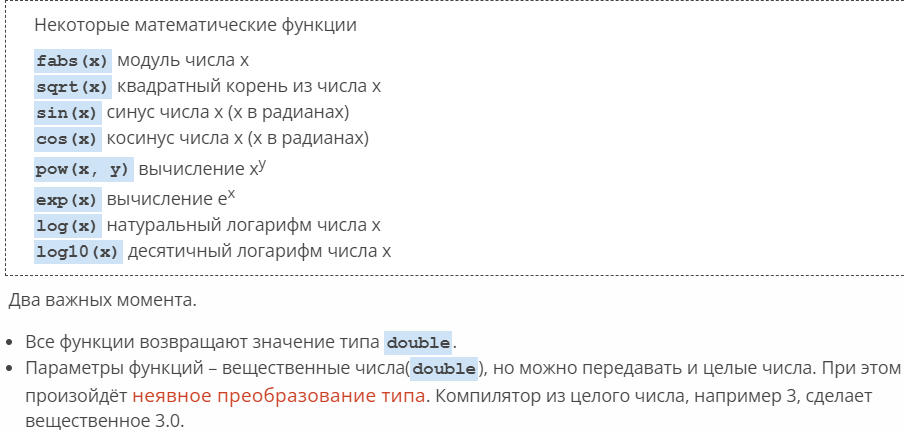




Математические функции

Для добавления математических функций нужно подключить заголовочный файл.

#include <math.h>



Рандом

Нужно подлкючить #include <stdlib.h>

Функция **rand()** – возвращает случайное целое число в диапазоне от 0 до RAND\_MAX

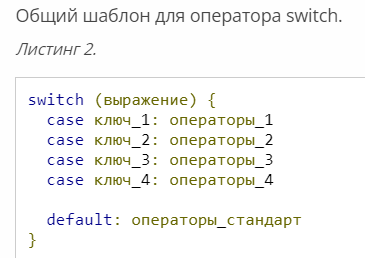
Чтобы каждый раз генерировались новые случайные числа используем

#include <time.h>

Srand(time(NULL))

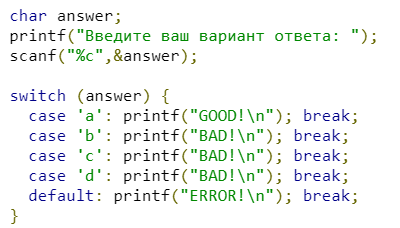
Теперь каждый раз при вызове функции rand() будут генерироваться случайные разные числа.

Оператор switch

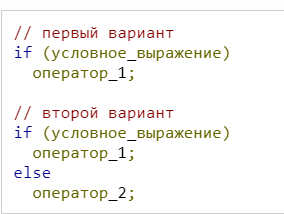


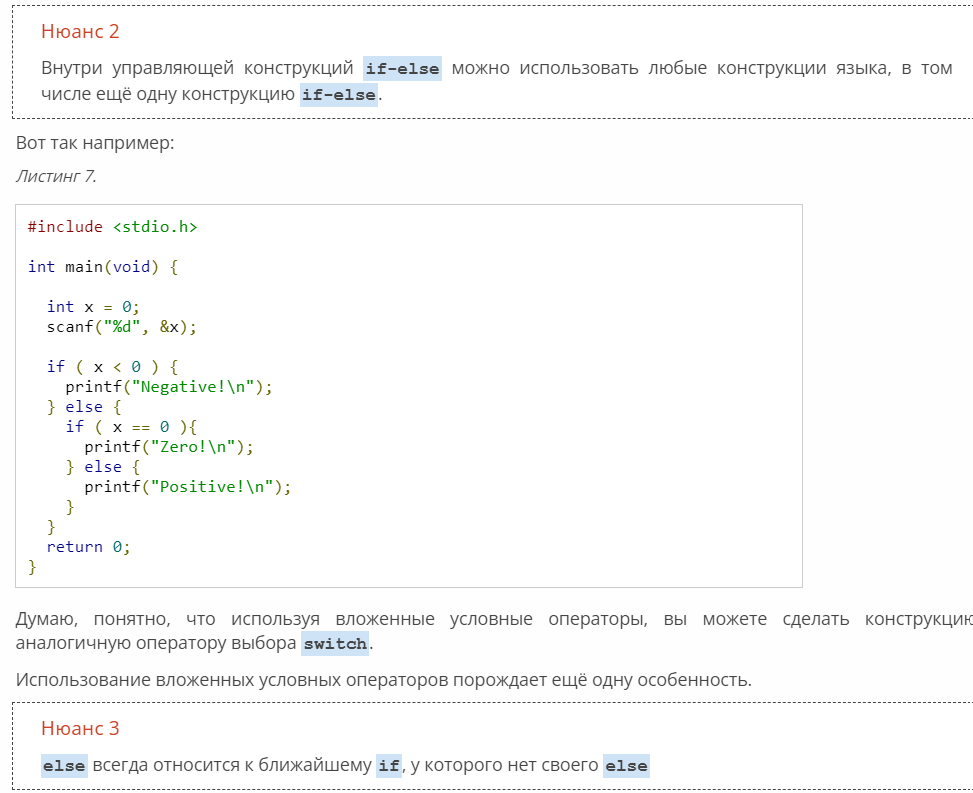
В качестве выражения может использоваться переменная типа int или char

После того, как выражение вычислено, его результат проверяется на соответствие с ключ\_1 и т.д.

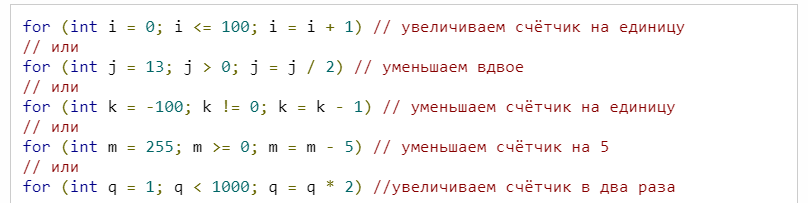


Условный оператор if-else

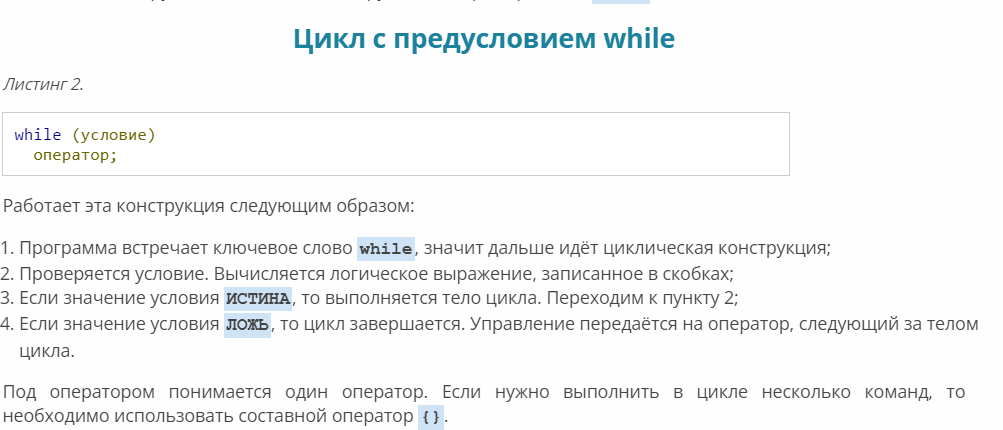


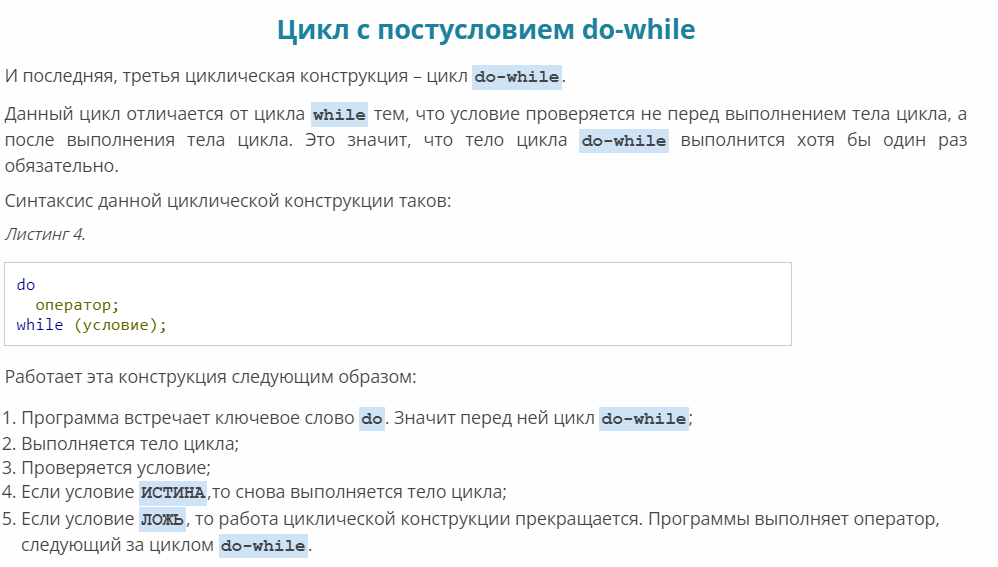


Цикл FOR



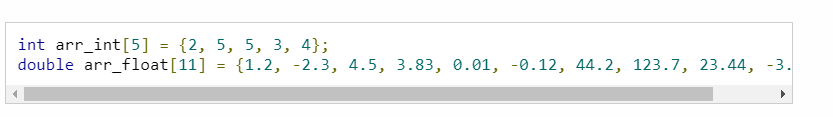
Цикл с условием WHILE

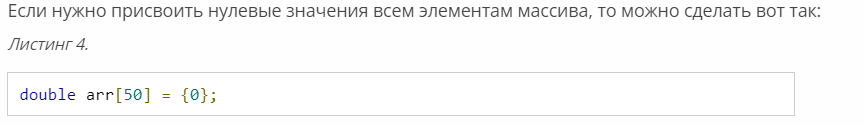




Массивы

**Объявление массива и инициализация**



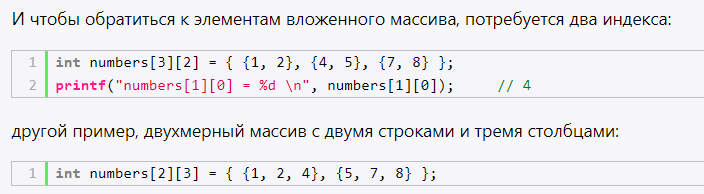


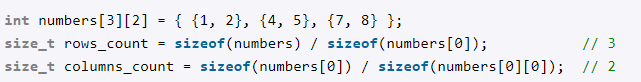
**Как узнать размер массива**

Size\_t size = sizeof(array) – размер массива в байтах

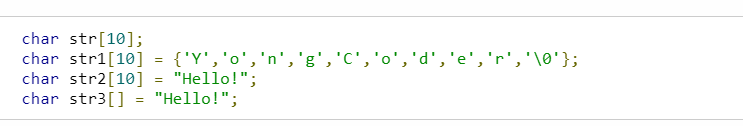
Size\_t count = sizeof(array) / sizeof(int) – количество элементов массива

Двумерные массивы

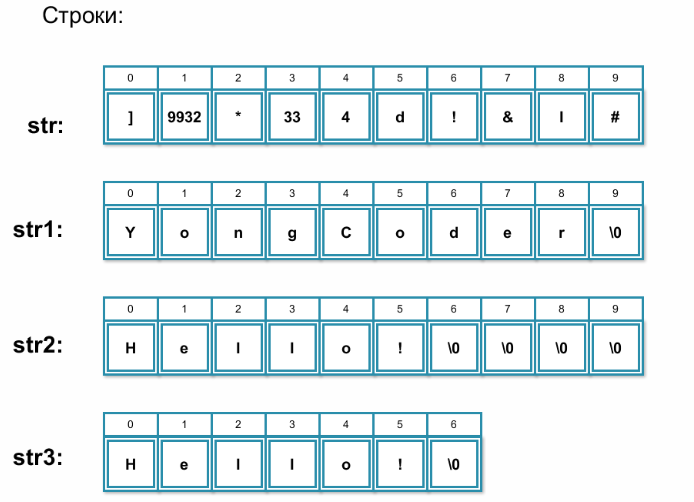




Строки



Си-строка это массив char. **Обязательно последний символ ‘\0’**



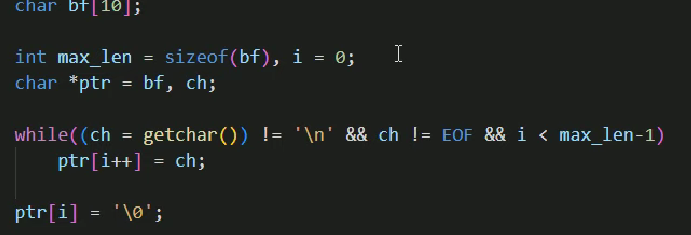
Вывод строки на экран

1. Использовать printf со спецификатором %s
2. Использовать функцию puts(str) – без форматирования
3. Использовать функцию fputs(str,stdout)

Puts переносит вывод на следующую строку, fputs не переносит

Ввод строк

1. Gets(str) – считывает пока не нажмем Enter , поэтому есть опасность выйти за рамки массива **(НЕ РЕКОМЕНДУЕТСЯ ИСПОЛЬЗОВАТЬ)**
2. Scanf(“%10[^n]”, str) – считать не более 10 символов, при этом считать ввод до тех пор пока не встретится символ перехода строки (пока не нажали клавишу Enter). Если символы не влезли в str, они так и останутся в потоке ввода. Нужно очистить поток ввода.
3. While (c = getchar() != ‘\0’) – считать посимвольно в переменную int c; getchar возвращает переменную типа int!! Т.к. помимо символов, хранятся еще служебные значения (EOF = -1)



1. Fgets(str, кол-во\_символов, stdin) – считывает n-1 символов из-за символа переноса строки.

Однако с fgets есть проблема. Символы, которые не влезли в строку останутся в потоке ввода stdin и когда мы будем после считывать что-то из этого потока – мы считаем эти символы. Поэтому нужно очистить поток с помощью fflush(stdin) ИЛИ

while (getchar() != '\n');

**Этот способ является более предпочтительным.**

Функции работы со строками

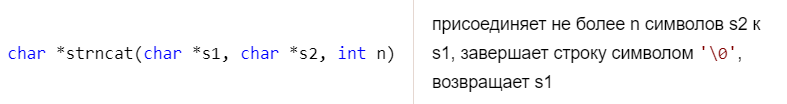
Для использования #include <string.h>

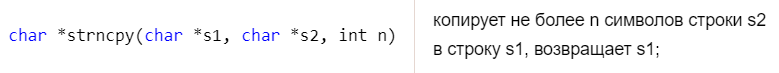
Strlen(str) – возвращает длину строки

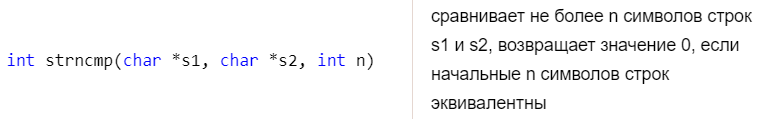
Strcat(str1,str2) – приклеить к концу строки str1 строку str2 (ПРИ ЭТОМ STR1 ДОЛЖНА ВМЕЩАТЬ ДВЕ СТРОКИ И ЗАВЕРШАЮЩИЙ СИМВОЛ)

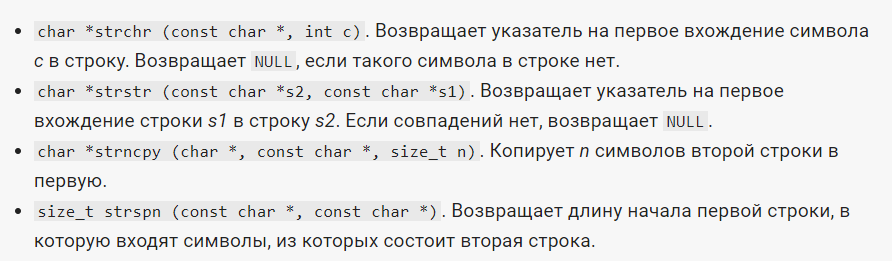
Strcmp(str1,str2) – сравнение двух строк. Если строки одинаковые вернет 0, если первая строка больше – положительное число, если первая строка меньше – отрицательное число.

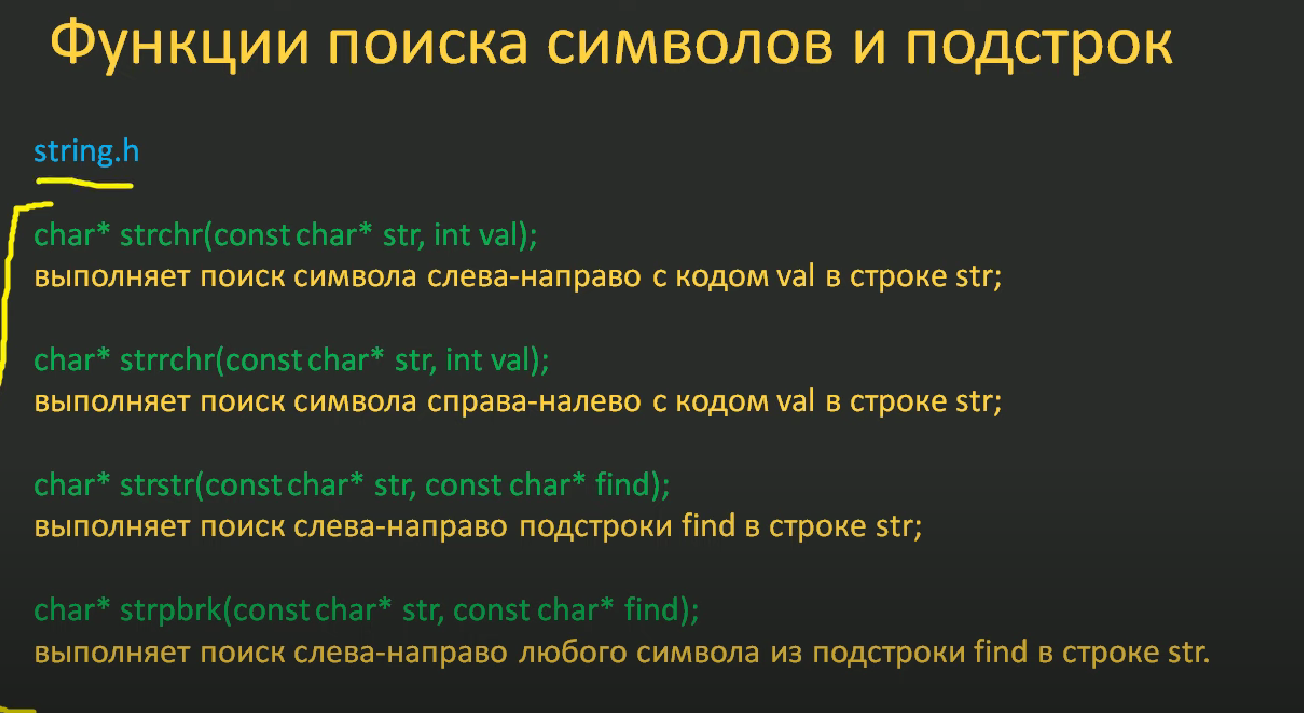
Strcpy(str1,str2) – копирует str2 в str1 (str1 по размеру должна вмещать str2)

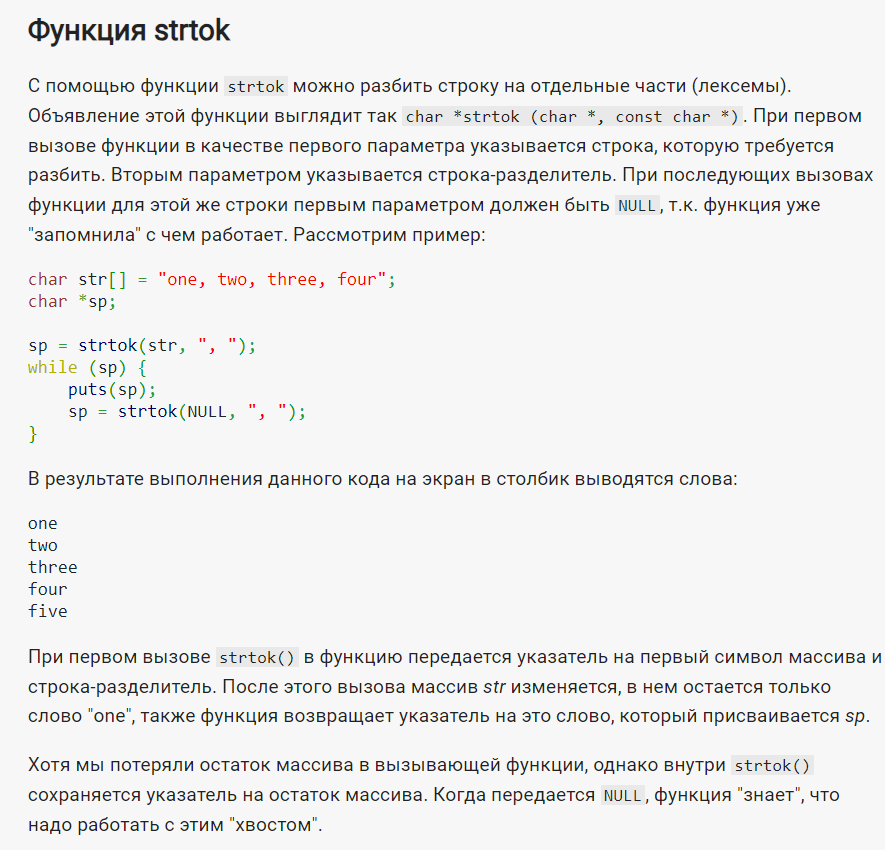




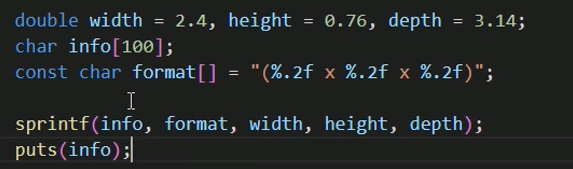






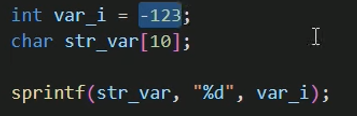


Int sprint(char \*buffer, const char\* format, параметры) – занести в строку данные, учитывая формат

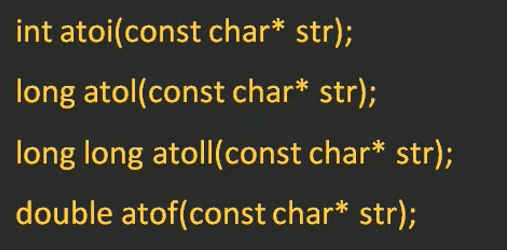


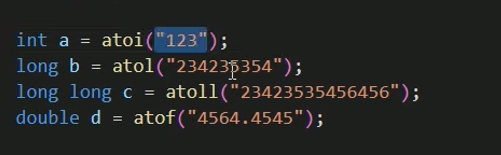
Получится (2.4 х 0.76 х 3.14)

Еще один удобный трюк с этой функцией : представить число в виде строки



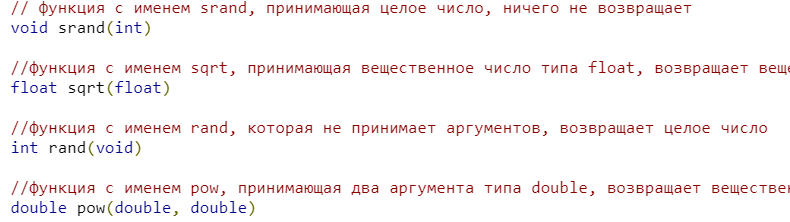
**Обратное действие: перевести число в строковом формате – в числовой тип данных**





Функции

Заголовки функций



Указатели

Переменные, которые хранят адрес в памяти. Нужны для считывания данных из ячейки памяти и записи туда.

Допустим есть ячейка памяти с адресом 34024

Char \*gpt = 34024; - присвоить указателю адрес

Char x = \*gpt; - считать значения из ячейки памяти

\*gpt = 100; - записать значение в ячейку памяти

Адресная арифметика

1. Если увеличить указатель на 1, то его адрес увеличится на размер типа данных (например для int на 4)



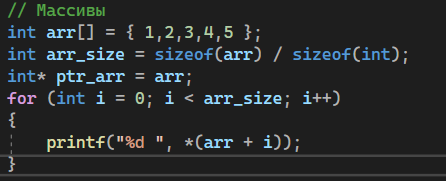
Умножать указатели и делить их нельзя. Также нельзя складывать указатели между собой.

1. Но можно вычесть один указатель из другого. Тогда их разность покажет сколько между ними элементов (например массива).

Если вычесть из указателя на 3 элемент массива указатель на 0 элемент массива, то между ними будет 3 элемента.

1. Название массива = указатель на его первый элемент

С помощью указателей можно итерироваться по массиву и выводить его элементы

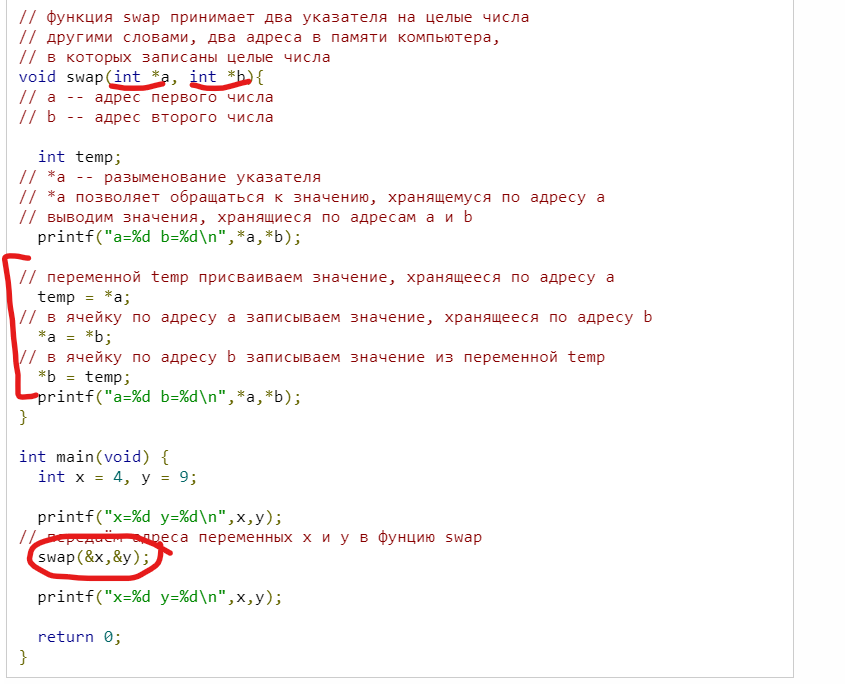


Const с указателями

Const int \*ptr – значит, что можем только считать значение по адресу, но не менять его. ИСПОЛЬЗУЕТСЯ, ЧТОБ НЕ ИСПОГАНИТЬ ДАННЫЕ

Int\*const ptr – не можем менять адрес указателя, после его инициализации.

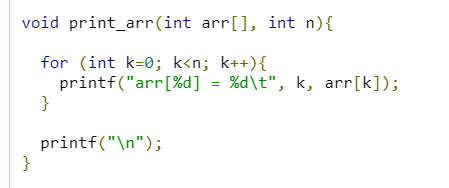
Передача аргументов в функцию

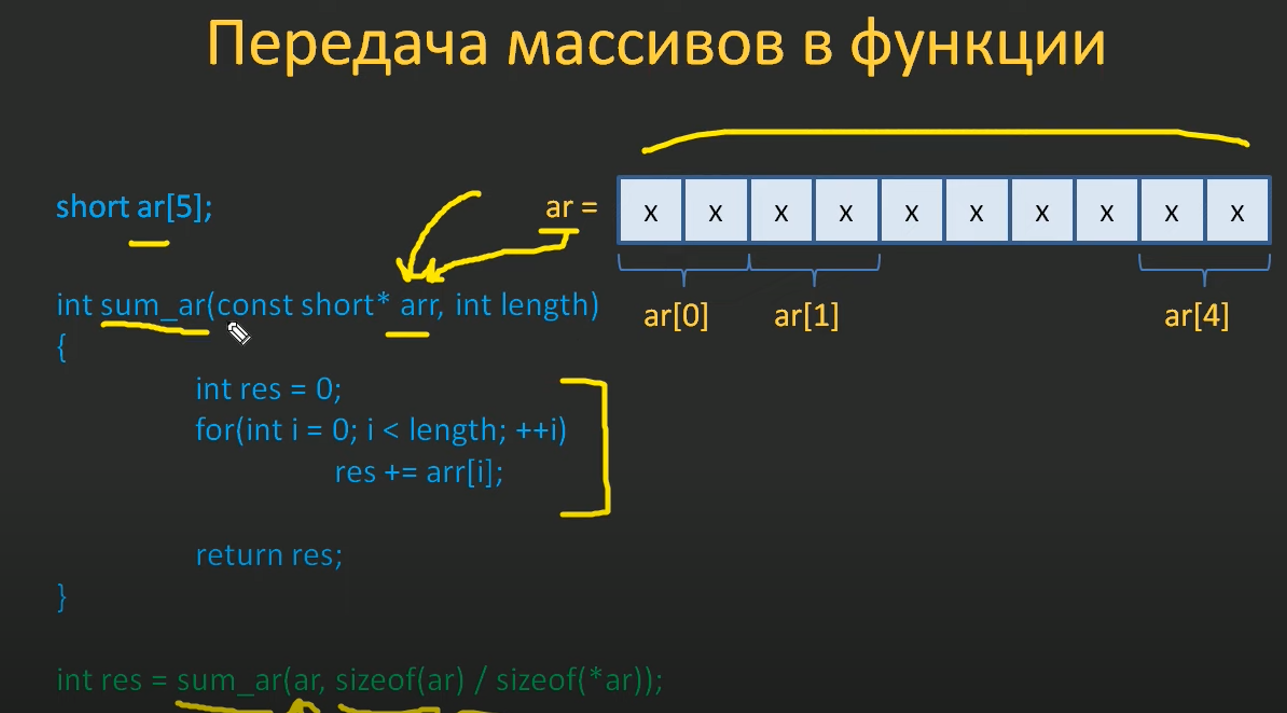


Передача массива в функцию

Чтобы передать массив в функцию нужно написать его имя + квадратные скобки. Также вторым аргументом обязательно идет размер массива.

Массивы всегда передаются по ссылке. Если мы что-то будем менять в функции – изменится и исходный массив. Чтобы не испоганить данные можно передавать массив по константному указателю.





Указатели на функцию

Имя функции = адрес функции, т.е. указатель

Объявление указателя на функцию



<Тип возвращаемого значения> (\*имя\_функции) (типы\_параметров)

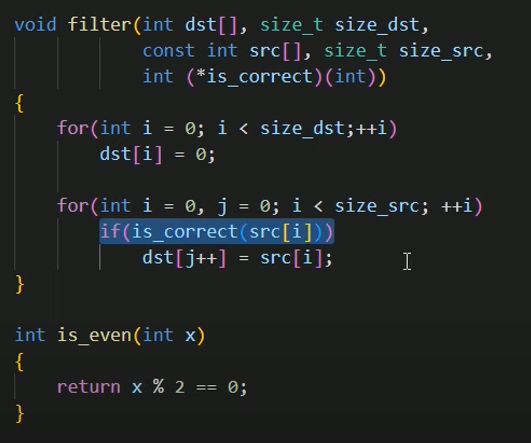
Через указатель на функцию можно вызывать саму функцию также как и обычную.

Ptr\_func = sq\_rect;

Ptr\_func(2,3) – вызов функции через указатель

Пример использования: фильтровать значения массива с помощью определенной функции.

Отобрать только четные значения из старого массива в новый.



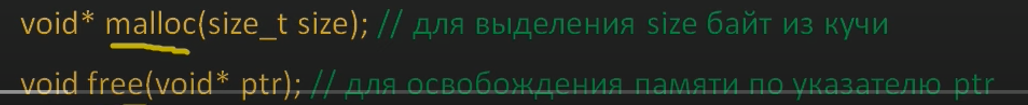
Массив указателей на функцию

Например когда нужно вызвать много функций, можно просто пройтись по массиву этих функций и вызвать их.

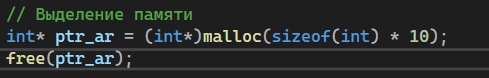


Выделение памяти

Для выделения памяти используются функции из stdlib.h

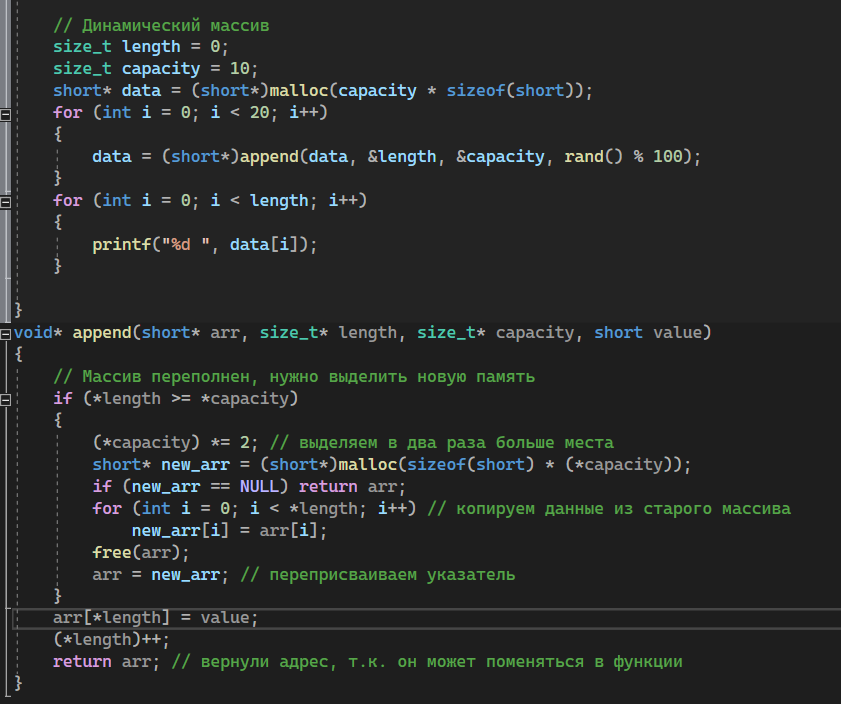


Пример: выделить память на 10 ячеек типа int



Динамический массив

Для создания динамического массива нужно реализовать алгоритм перевыделения памяти на большее количество ячеек, в случае если прошлый массив переполнен.



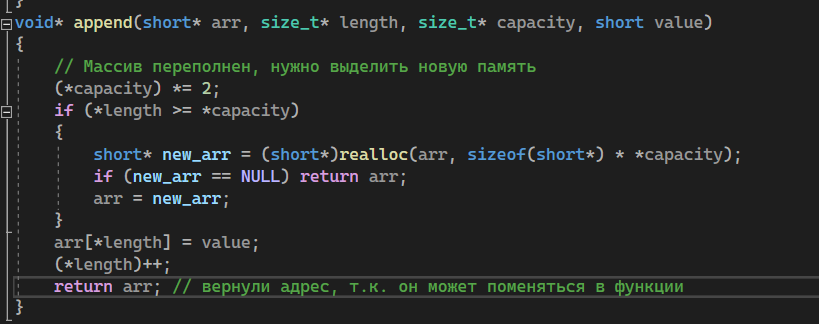
Дополнительные функции выделения памяти

Void\* calloc(size\_t nmemb, size\_t size) – выделяет память под nmemb элементов каждый по size байт + **заполняет эту память нулями**.

Void\* realloc (void \*ptr, size\_t length) – изменяет размер ранее выделенной памяти по указателю ptr, на новый размер length.

Если в realloc поставить length = 0 , то функция будет аналогична free()

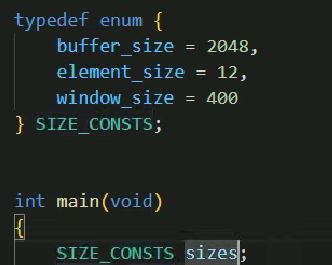
Если в realloc length больше прежнего размера, то функция попытается взять последующие ячейки памяти (расширить массив) , если они будут заняты – realloc выделит новую область памяти и скопирует туда данные.



Перечисления (enum)

Перечисления хранят константы, которые инициализируются во время компиляции. Этим они отличаются от констант определенных глобально, либо с помощью директив #define.

То есть мы можем использовать перечисления в конструкции switch и т.д.

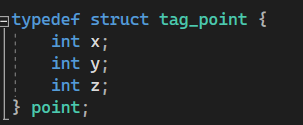


Имена констант в программе заменяются на числа.

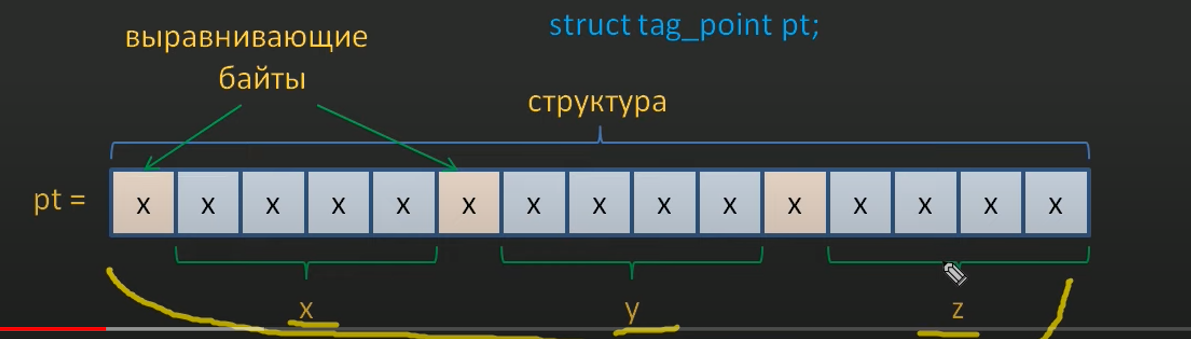
Если мы напишем buffer\_size , то для компилятора это будет = 2048.

Структуры

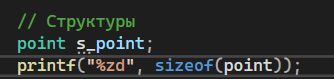
Определение структуры с заменой имени через typedef



Как размещаются переменные структуры в памяти:

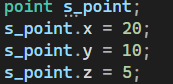


Переменная структуры представляет собой неразрывную последовательную цепочку байт в памяти.



Sizeof(point) выдаст 12, т.к. у нас внутри этой структуры 3 переменные по 4 байта.

Присваивание данных полям структуры происходит как с классами.

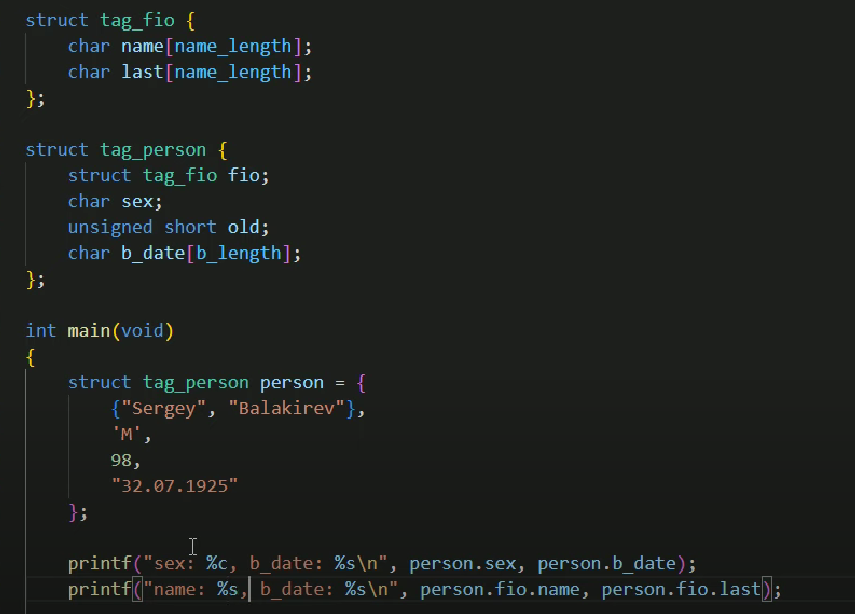


Также можно указать значения полей при инициализации.



Вложенные структуры

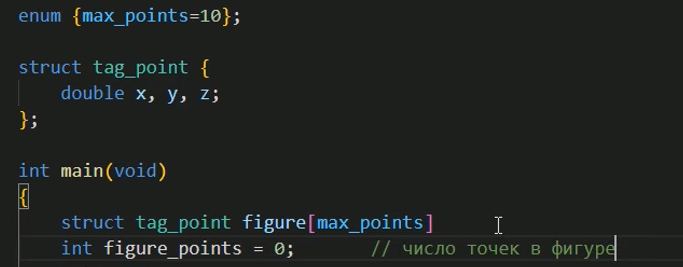
Пример: структура человека с вложенной структурой его ФИО

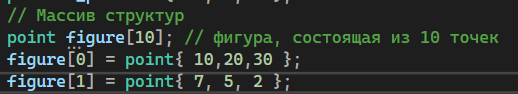


Массивы структур

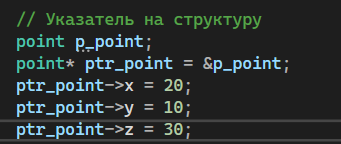
Пример : фигура, состоящая из n количества точек.

Фигуру можно представить в виде массива точек, а точка представляет из себя структуру.



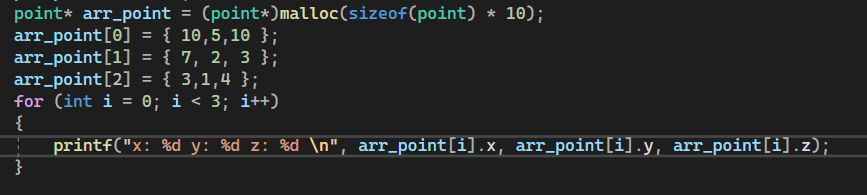


Указатели на структуры



Обращаться к полям класса через указатель можно через оператор «->»

**Указатель на массив структур**

****

Со структурами лучше работать через указатели ТОЛЬКО ЕСЛИ ОНИ ЗАНИМАЮТ МНОГО ПАМЯТИ(иначе велика вероятность че то не заметить и будет утечка памяти).

Реализация стека (LIFO)

LIFO – Last In First Out (типа стопка посуды, последняя положенная тарелка будет сверху и ее легче всего достать)

