# 

[**Вопросы**](#_y5v9iwax98an) **3**

[**1. Понятие алгоритма, определение и правила построения алгоритмов.**](#_5oc7eqsnhkpi) **3**

[**2. Состав языка Си: алфавит, идентификаторы, ключевые слова, знаки операций, константы, комментарии. Структура простейшей программы.**](#_dj6hw3yqly9p) **3**

[**3. Операция присваивания, ее различные формы (инкремент, декремент). Стандартные библиотеки языка C/C++, функции библиотеки math.h.**](#_2swnltmcaour) **4**

[**4. Базовые типы данных. Понятие операции и выражения.**](#_t97uy3q8mk1d) **7**

[**5. Арифметические операции, преобразование типов при выполнении операций, операция явного приведения типа. Понятие переменной.**](#_rre5ucg1p9cc) **10**

[**6. Операции сравнения, логические операции, побитовые операции. Оператор безусловной передачи управления goto. Понятие «блок».**](#_n9utl45b2v0x) **13**

[**7. Оператор условного перехода if-else, оператор альтернативного выбора switch. Условная тернарная операция «?:». Операторы передачи управления**](#_egc5hu17o8l) **15**

[**8. Операция последовательного вычисления «,» (запятая). Оператор цикла с предусловием while.**](#_1uvhavyaf0hj) **18**

[**9. Операторы цикла: оператор цикла с постусловием do-while, оператор цикла с постусловием и коррекцией for. Операторы break, continue.**](#_rlb791od6uf4) **19**

[**10. Декларация статических массивов, размещение данных в памяти, правила обращения к элементам массивов.**](#_kzfte79utmxd) **22**

[**11. Ввод-вывод одномерного и двумерного массивов, заполнение массива случайными равномерно распределенными числами.**](#_y757i5vx6fdk) **24**

[**12. Основные алгоритмы работы с элементами массива: нахождение суммы, произведения, минимального и максимального, среднего.**](#_70g5mwyx1617) **25**

[**13. Декларация указателя. Указатель на объект, указатель типа void. Инициализация указателя, значение NULL. Операции над указателями.**](#_wtzib4g99b1p) **26**

[**14. Связь указателей с массивами. Создание динамических массивов и правила работы с ними.**](#_osbtn0n934ge) **28**

[**15. Динамическое выделение памяти с помощью библиотечных функций (операции new, delete)**](#_5on491f9bb2) **28**

[**16. Строка – массив типа char. Стандартные функции библиотеки string.h**](#_srwojqj170eh) **32**

[**17. Декларация структуры (struct). Создание структурных переменных. Обращение к элементам структуры. Вложенные структуры.**](#_d1ns3gop4y9z) **33**

[**18. Перечисления (enum), объединения (union).**](#_vvpwq9dc1eu0) **37**

[**19. Понятие функции, описание и определение функции. Вызов функции.**](#_cmj9jd3b995o) **38**

[**20. Передача данных в функцию по значению, по указателю, по ссылке.**](#_b5nz2xe8sjqg) **38**

[**21. Параметры функций по умолчанию, функции с переменным числом параметров.**](#_kjij5mny16qp) **39**

[**22. Встраиваемые функции. Перегрузка функций. Передача массивов в функцию. Указатель на функцию.**](#_qa01oxl1xfds) **39**

[**23. Классы памяти. Время жизни и область видимости переменных.**](#_9p92vzogvq27) **41**

[**24. Стандартные библиотечные функции для организации ввода-вывода информации (getc, gets, scanf, putc, puts, printf). Спецификации преобразований для данных различных типов.**](#_ibkte36ou430) **42**

[**25. Директивы препроцессора.**](#_3cvwxr980bmk) **42**

[**Задачи с экза.**](#_6mvk2iiftn94) **45**

# 

**Практика:**

[Ссылочка](https://github.com/AsoTora/labs_2018) на все мои лабы. Там есть комменты и все такое, ошибки конечно же тоже, наверняка. Но основные алгоритмы и примеры задач в общем и и в целом разобраны.

Советую глянуть 5-ю и 6-ю лабы. Перестановка строк массива через переменную и указатель, перестановка столбцов, рекуррентные последовательности.

# 

# Вопросы

# 1. Понятие алгоритма, определение и правила построения алгоритмов.

(в методе нет, в конспекте тоже. Навр - лежать + сосать. **Опер**деление из инета, ваша фантазия приветствуется)

Алгоритм - это последовательность команд. набор инструкций, описывающих порядок действий для достижения результата.

этой хуеты нет в норм виде, сорян

# 2. Состав языка Си: алфавит, идентификаторы, ключевые слова, знаки операций, константы, комментарии. Структура простейшей программы.

* **Алфавит** языка Си состоит из прописных и строчных букв латинского алфавита,

арабских чисел, специальных символов, пробельных и разделительных символов.

Из символов алфавита формируются **лексемы** (элементарные конструк-

ции языка). К лексемам относятся: идентификаторы, зарезервированные слова,

знаки операций, константы, разделители.

* **Идентификатор** – последовательность цифр и букв латинского алфавита, а также специальных символов при условии, что первой стоит буква или знак подчеркивания. (обычное слово короче, *Пр.* ааа, баа, АуЕ). Допустимо использовать любое количество символов, однако значимыми считаются только первые 32.

При выборе идентификатора необходимо:

1. Следить, чтобы идентификатор не совпадал с ключевыми зарезервированными словами и именами библиотечных функций;
2. Использовать с осторожностью символ подчеркивания в качестве первого символа идентификатора и комбинацию «\_t» в конце идентификатора.

* **Ключевые слова** -- зарезервированы стандартом ANSI Си (для компилятора):

auto, double, int, struct, break, long, switch, register, typedef, char, extern, return,

void, case, float, unsigned, default , for, signed, union, do, if, sizeof, else,

while, volatile, continue, enum, short.

* **Комментарии** – текстовая или символьная информация, используемая для

пояснения участков программы. Комментарии не влияют на ход выполнения программы, т. к. не являются лексемами и не включаются в содержимое исполняемого файла.

В С++ комментарии:

1. Начинаются последовательностью «//» и заканчиваются концом

строки.

1. Начинаются последовательностью «/\*» и заканчиваются последова-

тельностью «\*/».

* **Знак операции** – один или несколько символов, определяющих дейст-

вие над операндами. Использование пробелов внутри знака операции не до-

пускается.

* **Структура программы** -- <http://informatics-lesson.ru/c/structure-program.php>

# 3. Операция присваивания, ее различные формы (инкремент, декремент). Стандартные библиотеки языка C/C++, функции библиотеки math.h.

* Формат операции присваивания

оператор\_1=оператор\_2;

В переменную оператор\_1 заносится значение оператор\_2. В качестве оператора\_1 можно использовать только переменную. В качестве оператора\_2 можно использовать константу, переменную, выражение или функцию.

Допустимо использовать: a=b=c=d;

это равнозначно a=b; b=d; d=c;

Часто в программировании используются операции такого типа:

оператор\_1=оператор\_2 знак\_операции оператор\_2;

для сокращения можно и нужно использовать короткую форму записи:

оператор\_1 знак\_операции=оператор\_2;

a=a+2; равносильно записи a+=2;

**Частный случай, когда оператор\_2 равен еденице называется операцией инкремента(++) или декремента(--):**

оператор\_1++; **инкремент(++)** a++; равносильно а=а+1 равносильно а+=1

оператор\_1--; **декремент(--)** a--; равносильно а=а-1 равносильно а-=1

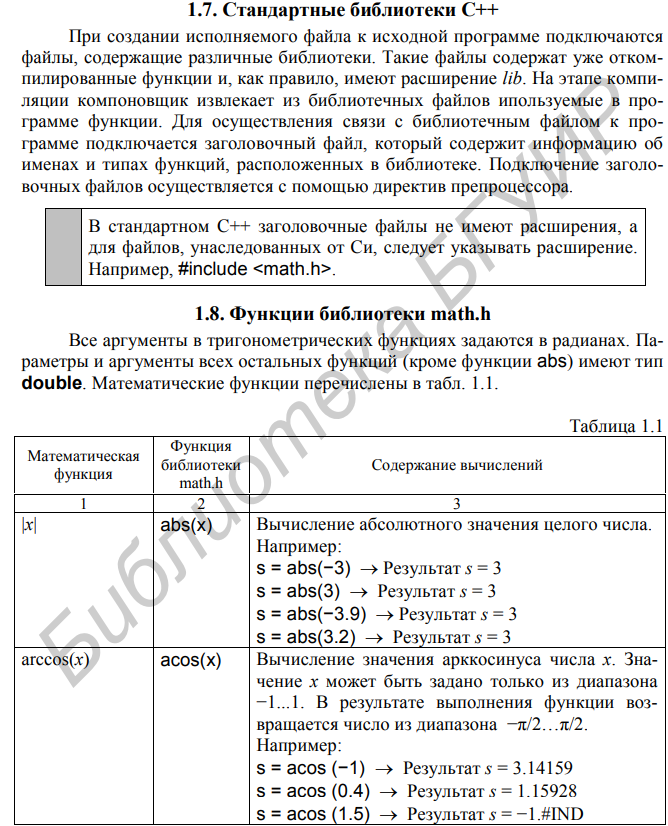
**ЕСТЬ ДВЕ ФОРМЫ (чекайте начало конспекта, Навр много распрягал про это)**

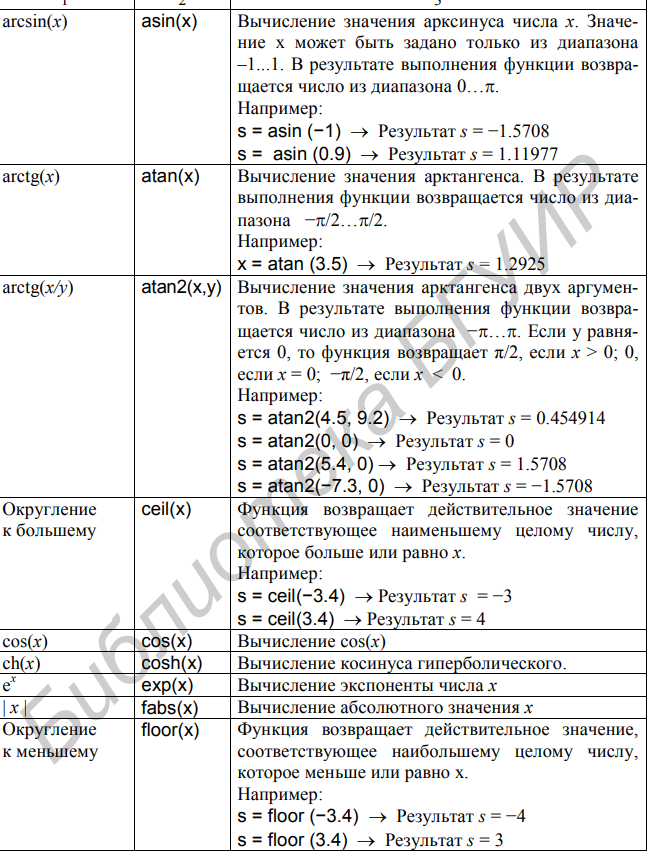
1)префиксная ++a;

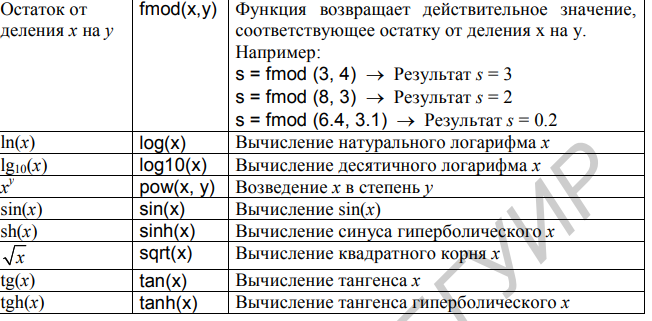
2)постфиксная a++;

при **префиксной** сначала выполняется инкремент/декремент, а затем арифметич. операции.

при **постфиксной** сначала выполняется арифметич. операции, а затем инкремент/декремент.







# 4. Базовые типы данных. Понятие операции и выражения.

* **Тип данных** позволяет определить, какие значения могут принимать пе-

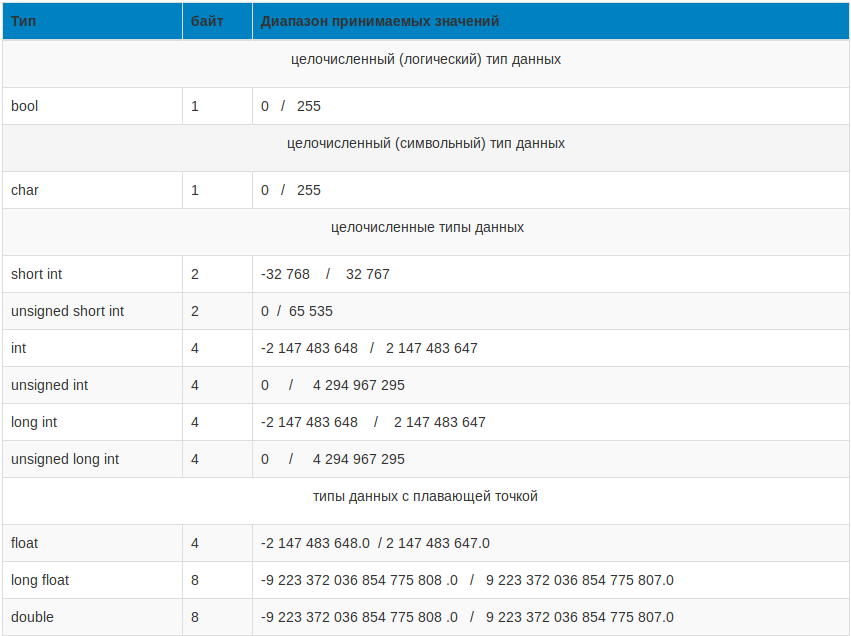
ременные, какая структура и какое количество ячеек используется для их раз-

мещения и какие операции допустимо над ними выполнять.

1. Скалярный тип -- данные, представляемые одним значением (числом, символом) и размещаемые в одной ячейке из нескольких байтов.
2. Структурированные типы определяются пользователем как комбина-

ция скалярных и описанных ранее структурированных типов.

* **Базовыми типами** данных являются: целый, действительный (вещественный) и символьный тип. Могут быть константами и переменными. В отличие от переменных константы не могут изменять свое значение во время выполнения программы.



* **Константы целого типа** – последовательность цифр, начинающаяся со

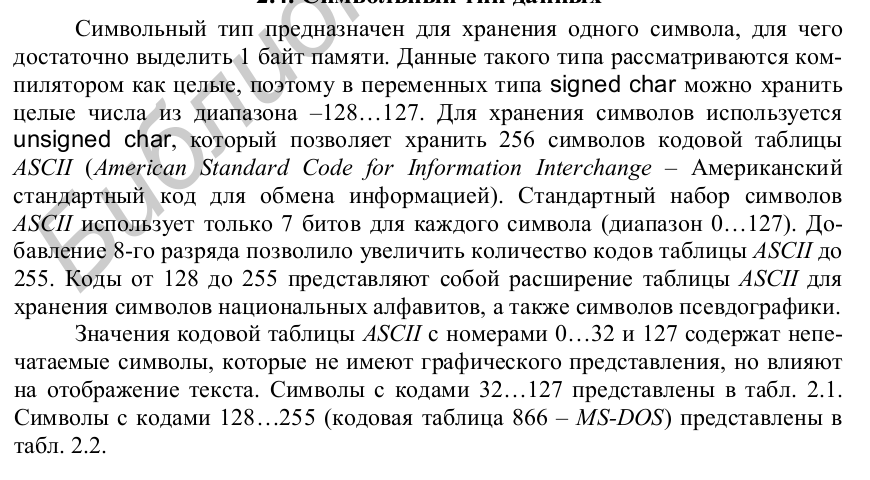
знака минус для отрицательных констант, и со знака плюс или без него для положительных констант. Для обозначения констант типа long после числа ставят букву L или l.

Десятичные константы: последовательность чисел от 0 до 9, начинающаяся не с нуля, например, 334.

Восьмеричные константы: последовательность чисел от 0 до 7, начинающаяся с нуля, например, 045.

Шестнадцатеричные константы: последовательность чисел от 0 до 9 и букв от А до F, начинающаяся с символов 0x, например 0xF5C3.

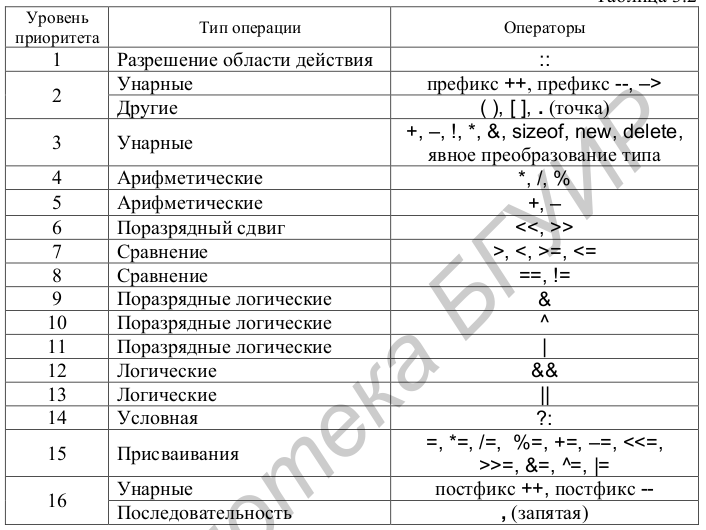
* **Символьный тип данных**

****

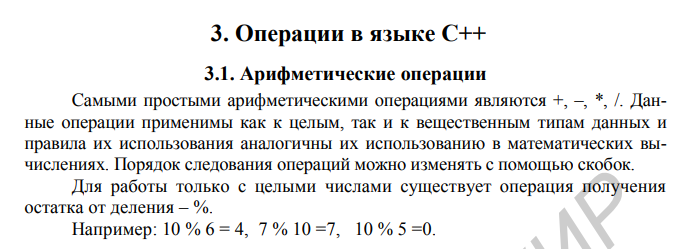
* **Логический тип данных.** 2 значения: true (1) или false (0). Но так как диапазон допустимых значений типа данных bool от 0 до 255, то необходимо было как-то сопоставить данный диапазон с определёнными в языке программирования логическими константами true и false. Таким образом, константе true эквивалентны все числа от 1 до 255 включительно, тогда как константе false эквивалентно только одно целое число — 0.
* **Операции и выражения.** Любое выражение языка состоит из операндов (переменных, констант и др.), соединенных знаками операций.

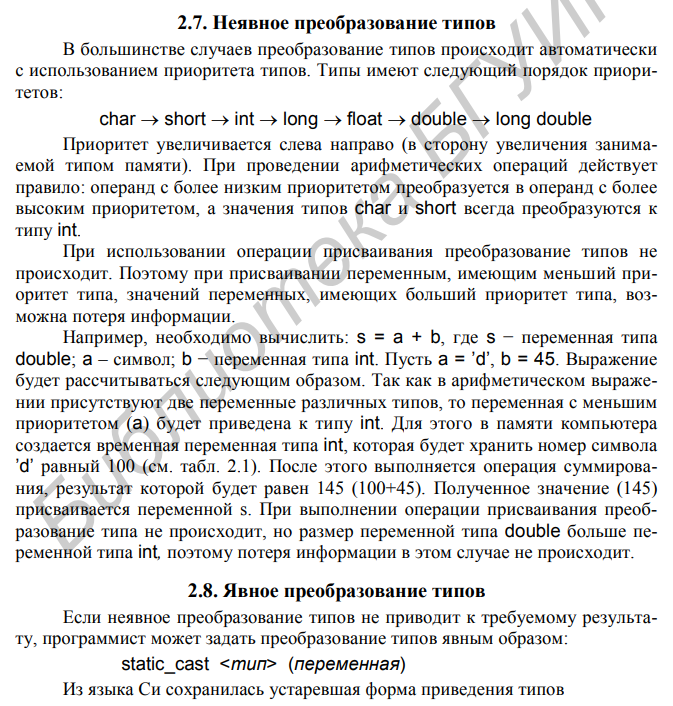
**Знак операции** - это символ или группа символов, которые сообщают компилятору о необходимости выполнения определенных арифметических, логических или других действий.

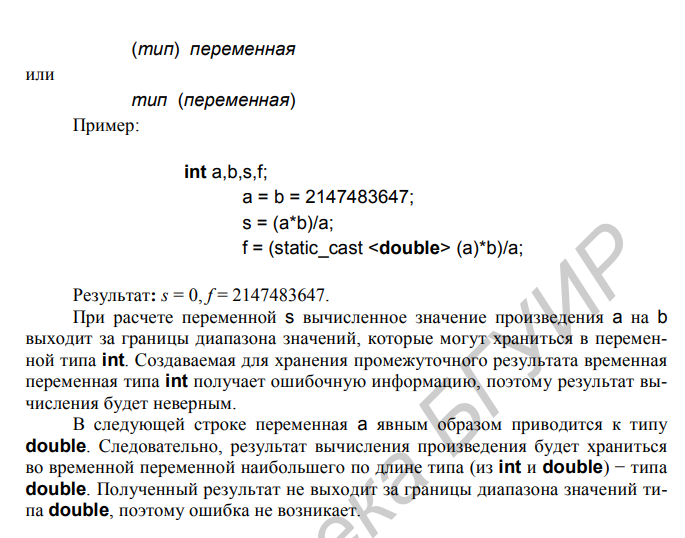
Операции выполняются в строгой последовательности. Величина, определяющая преимущественное право на выполнение той или иной операции, называется **приоритетом**. Порядок выполнения операций может регулироваться с помощью круглых скобок. Таблица приоритетов:



# 5. Арифметические операции, преобразование типов при выполнении операций, операция явного приведения типа. Понятие переменной.

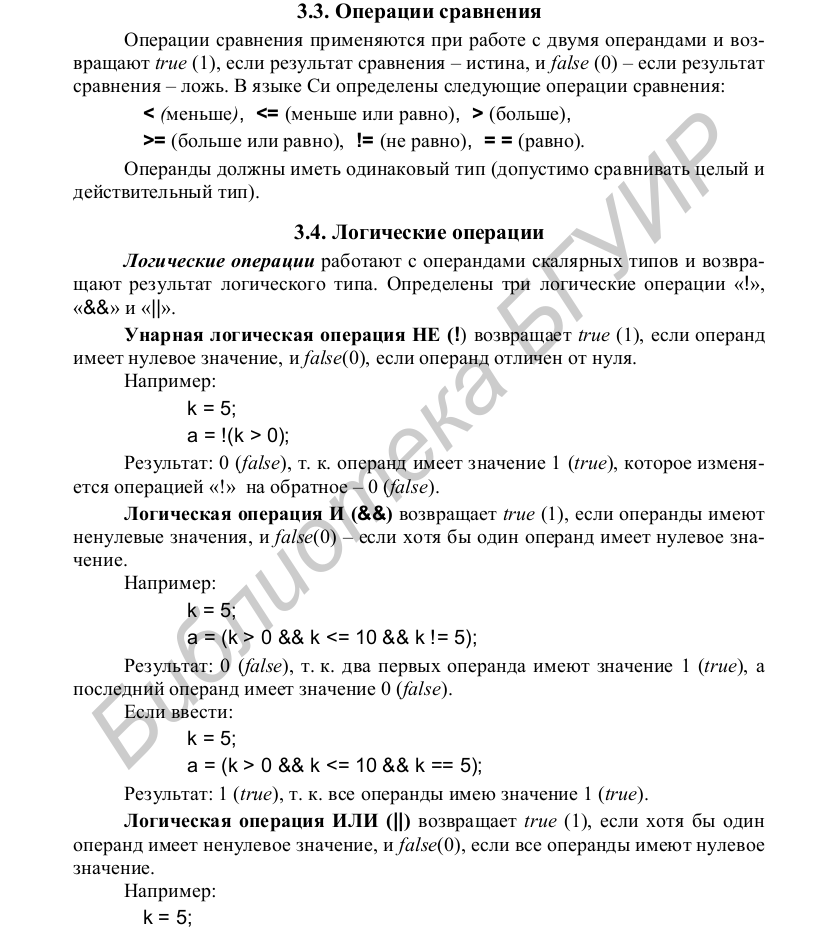
Арифметич операции. Хз че тут рассказывать, просто из таблицы запомните какие есть команды и всё.

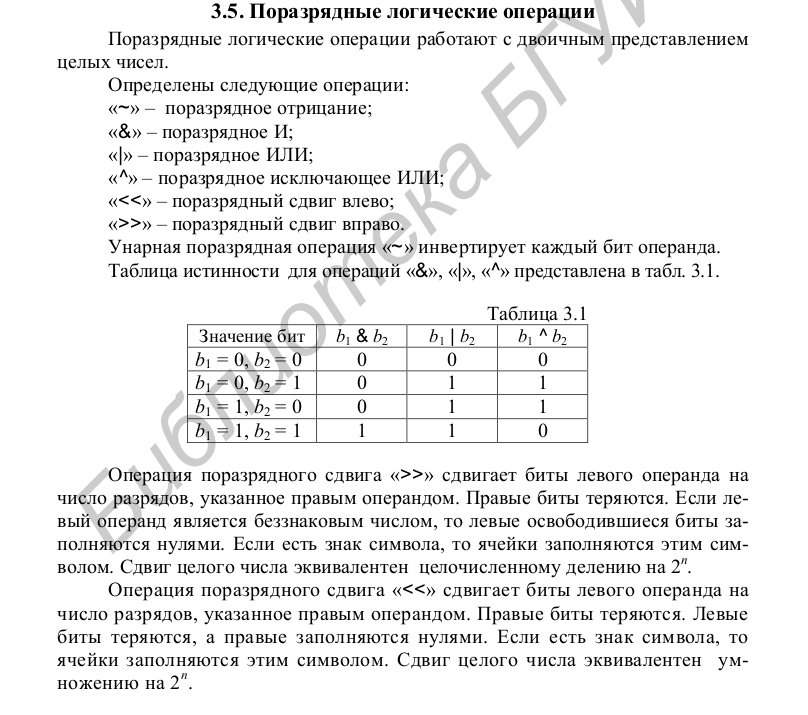




**Переменная** — именованный участок памяти для хранения данных определенного типа

# 6. Операции сравнения, логические операции, побитовые операции. Оператор безусловной передачи управления goto. Понятие «блок».





* **goto.** Операторы безусловной передачи управления изменяют нормальную последовательность программы и показывают, какой оператор должен выполняться следующим в результате некоторого вычисления и сколько раз. Их всего 3: оператор завершения **break**, оператор продолжения **continue**, оператор перехода **goto**.
  + Оператор завершения **break** обеспечивает возможность выхода из оператора цикла или оператора-переключателя switch, внутри которого он был употреблен, в ближайший оператор, охватывающий этот оператор цикла или оператор-переключатель switch.
  + Оператор продолжения **continue** обеспечивает возможность перехода к следующему шагу цикла, внутри которого он был употреблен.
  + Оператор перехода **goto** обеспечивает возможность перехода на метку, по форме совпадающую с идентификатором, внутри одной функции. Переход допускается в любое место из любого. (но есть функции и поэтому он -- **бесполезное говно**)
* **Блок** -- Группа операторов, заключенная в фигурные скобки, рассматривается компилятором как один составной оператор.

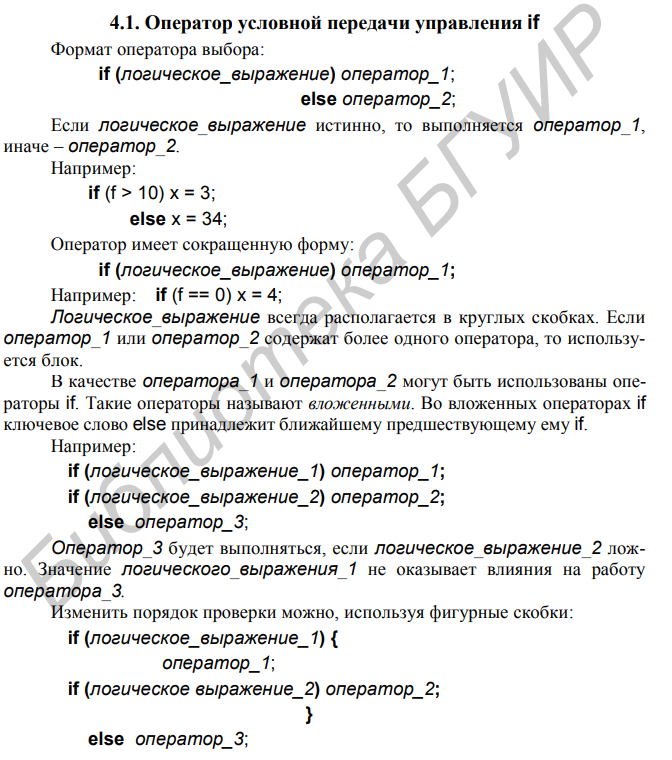
# 7. Оператор условного перехода if-else, оператор альтернативного выбора switch. Условная тернарная операция «?:». Операторы передачи управления

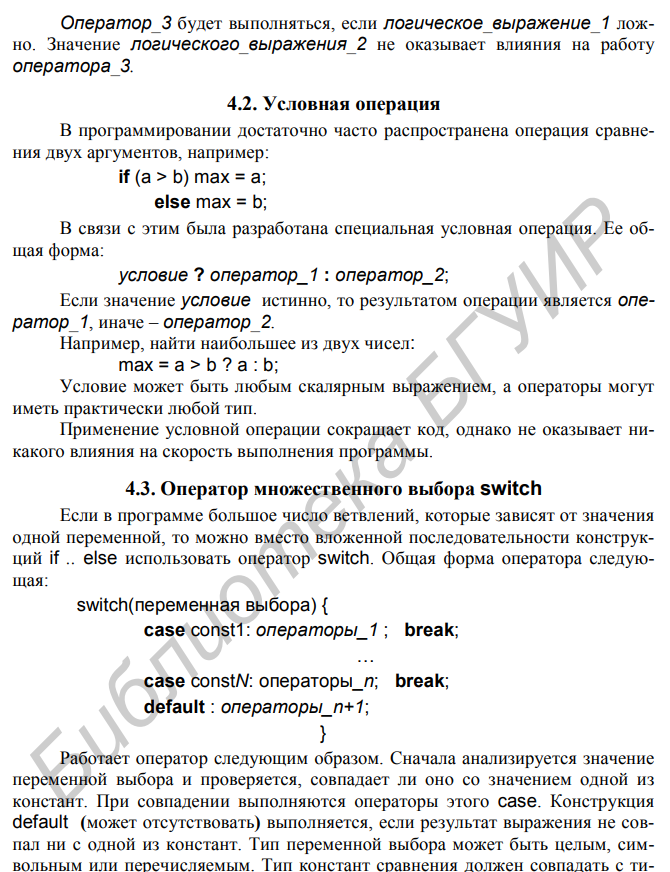
* **тернарная операция «?:»**

Типа пишите чет вроде **a\*b==2 ? cout<<”хуй соси” : cout<<”Губой тряси”;**

Эта херота значит, что  
Если произведение **а** на **b** равно двум, то выводится “хуй соси”, в ином случае выводится “губой тряси”

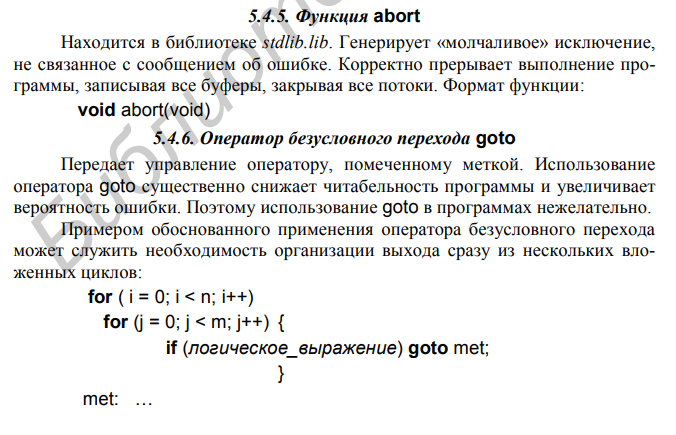
* **Оператор условного перехода if-else**

****

****

* **Операторы передачи управления**

****

****

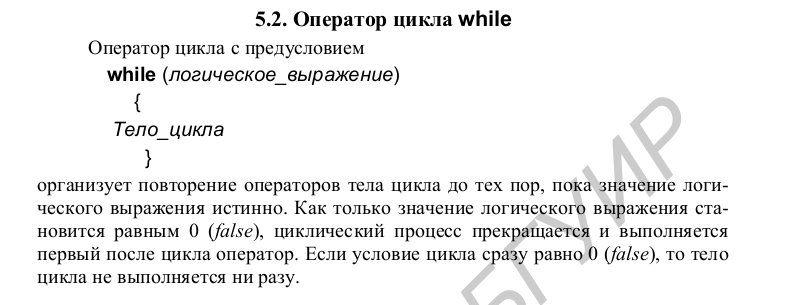
# 8. Операция последовательного вычисления «,» (запятая). Оператор цикла с предусловием while.

* **Операция последовательного вычисления** последовательно вычисляет два своих операнда, сначала первый, затем второй. Оба операнда являются выражениями. Синтаксис операции:

<*выражение1*>, <*выражение2*>

Знак операции - запятая, разделяющая операнды. Результат операции имеет значение и тип второго операнда. Ограничения на типы операндов (т. е. типы результатов выражений) не накладываются, преобразования типов не выполняются.

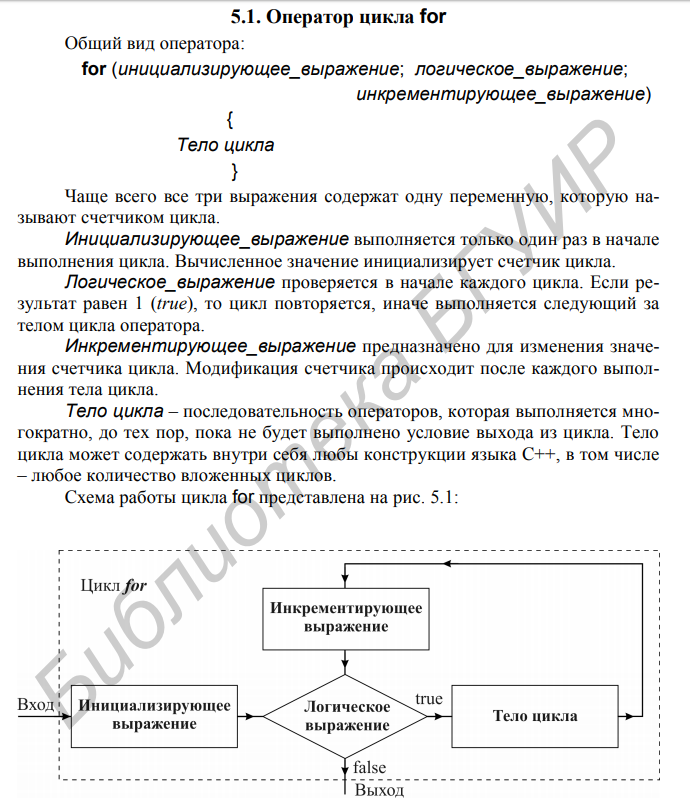
Операция последовательного вычисления обычно используется для вычисления нескольких выражений в ситуациях, где по синтаксису допускается только одно выражение(например цикле for: for(i=j**=**1; i+j**<**20; **i+=i, j--**))

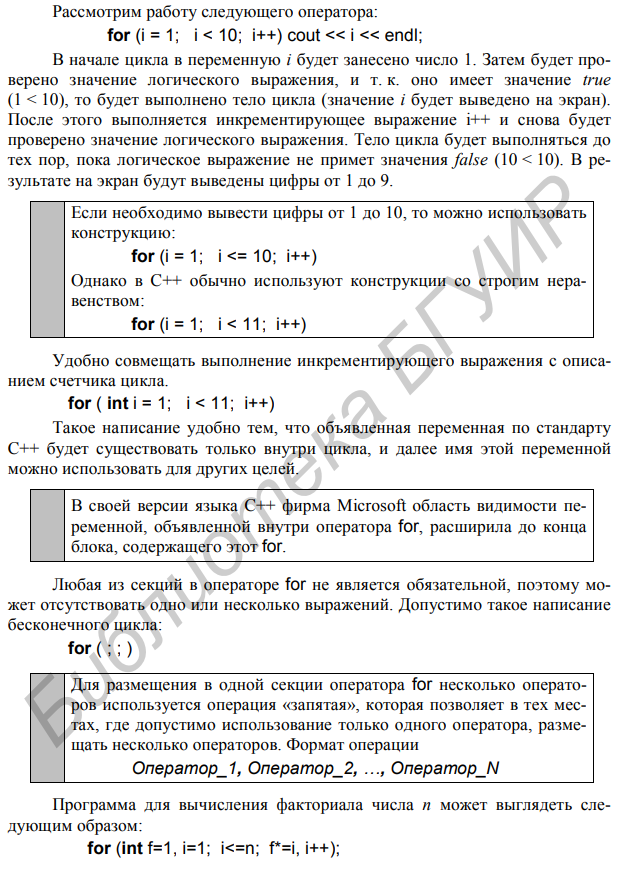


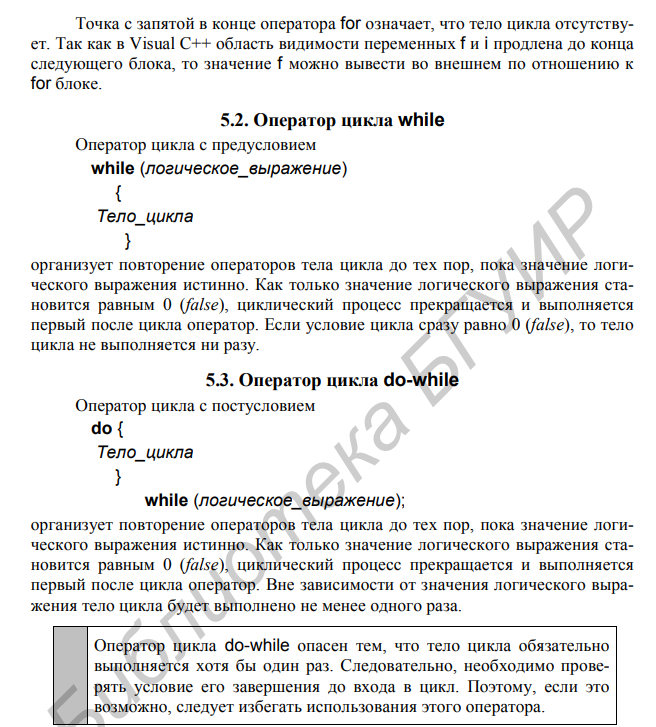
# 9. Операторы цикла: оператор цикла с постусловием do-while, оператор цикла с постусловием и коррекцией for. Операторы break, continue.

Юзайте обычный for и не ебите головы. Но, для общего развития можно и нужно знать другие варианты этой хероты.

**break, continue.** Были в операторах передачи управления







# 10. Декларация статических массивов, размещение данных в памяти, правила обращения к элементам массивов.

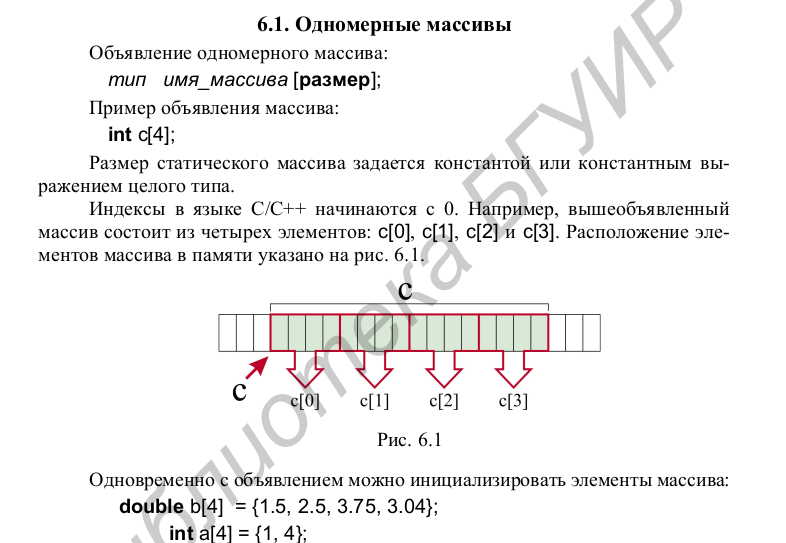
* **Массив --** структура данных, хранящая набор значений (элементов массива),

идентифицируемых по индексу или набору индексов, принимающих целые (или приводимые к целым) значения из некоторого заданного непрерывного диапазона.

Массив характеризуется: именем массива, типом хранимых данных, **размером**

(количеством элементов) и **размерностью** (формой представления элементов массива). Номер ячейки массива называется индексом. Индексы массивов должны иметь целый тип, а элементы массивов могут иметь любой тип.

* Статический массив



Если в группе инициализации не хватает начальных значений, то остав-

шиеся элементы заполняются нулями, например, массив a: a[0] = 1, a[1] = 4,

a[2] = 0 и a[3] = 0. // Зависит от компилятора, но в вижле gcc, который делает именно так.

Памяти занимается столько, сколько размерность умноженная на размер типа данных. **НО** когда объявляется массив в виде int array[25], то этим определяется не только выделение памяти для двадцати пяти элементов массива, но и **для указателя** с именем array, значение которого равно адресу первого по счету (нулевого) элемента массива.

* Для доступа к элементам массива существует два различных способа.
  + Первый способ связан с использованием обычных индексных выражений в квадратных скобках, например, array[16]=3 или array[i+2]=7.
  + Второй способ доступа к элементам массива связан с использованием адресных выражений и операции разадресации в форме \*(array+16)=3 или \*(array+i+2)=7. При таком способе доступа адресное выражение равное адресу шестнадцатого элемента массива тоже может быть записано разными способами \*(array+16) или \*(16+array).
* При реализации на компьютере первый способ приводится ко второму, т.е.

индексное выражение преобразуется к адресному. Для приведенных примеров array[16] и 16[array] преобразуются в \*(array+16). Для доступа к начальному элементу массива (т.е. к элементу с нулевым индексом) можно использовать просто значение указателя array или ptr. Любое из присваиваний

\*array = 2; array[0] = 2; \*(array+0) = 2; \*ptr = 2; ptr[0] = 2; \*(ptr+0) = 2;

присваивает начальному элементу массива значение 2, но быстрее всего выполнятся присваивания \*array=2 и \*ptr=2, так как в них не требуется выполнять операции сложения.

# 11. Ввод-вывод одномерного и двумерного массивов, заполнение массива случайными равномерно распределенными числами.

Ну, лол. Тип воодите всякую вашу дристоту через цикл, границей которого будет размер массива. Помните, что все многомерные массивы можно юзать как адскисукапиздецкак длинный массив (бесполезно, но помнить надо, вроде)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Int mas[n];

For (int i=0; i<n; i++)

cin>>mas[i];

Это для одномерного.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Int mas[n][m];

For (int i=0; i<n; i++)

{

For (int j=0; j<m; i++)

{

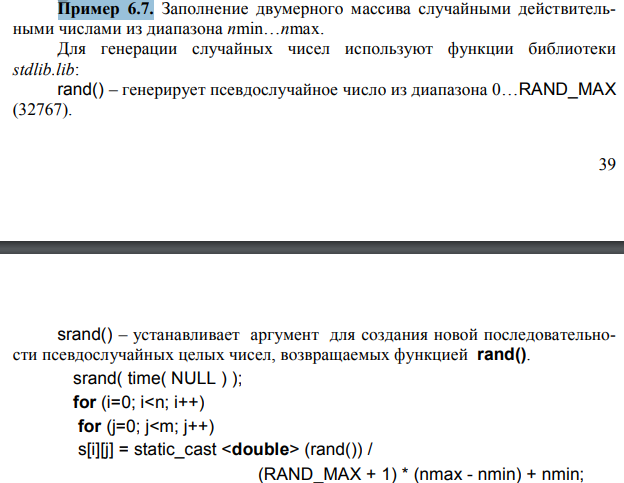
cin>>mas[i][j];

}

}

Это для двумерного.

Заполнять рандомно не надо ваще никак, есть риск отчисления в армию. Но, для проверорк самого себя почему бы и нет



# 12. Основные алгоритмы работы с элементами массива: нахождение суммы, произведения, минимального и максимального, среднего.

|  |
| --- |
| int n; cout << "Size:"; cin >> n; int \*arr = new int[n]; *// объявление массива*  for (int i=0; i<n; i++){ *// вывод эл-тов массива*  cout << " " << arr[i]; }  int pr=1; *// нахождение среднего арифм, суммы и пр-ния эл-тов* double sredn, sum=0; for (int i=0; i<n; i++){  sum += arr[i];  pr \*= arr[i]; } sredn = sum/n;  int min,max; min = max = arr[0]; *// нахождение макс и мин эл-тов* for (int i=1; i<n; i++){  if (arr[i] < min) min = arr[i];  if (arr[i] > max) max = arr[i]; }  for (int i=0; i<n; i++){ *// удаление эл-та из массива*  if (arr[i] < 0)  for (int j=i+1; j<n; j++) *// смещение всех эл-тов на 1 влево*  arr[j-1]=arr[j];  n--; i--; *// чтобы не потерять 1 проверку* }  delete []arr; *// освобождение памяти* |

# 13. Декларация указателя. Указатель на объект, указатель типа void. Инициализация указателя, значение NULL. Операции над указателями.

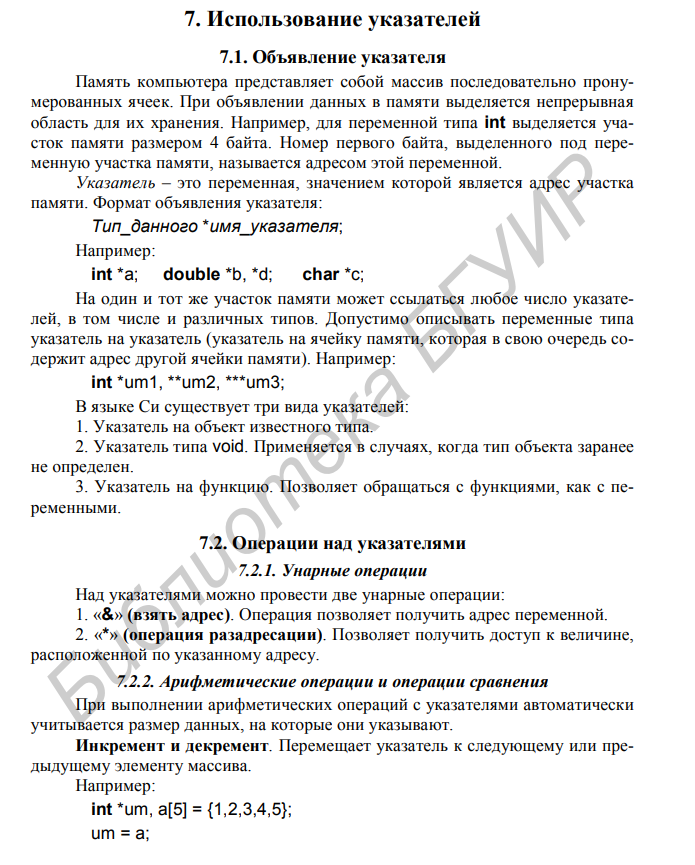
Земля металлом, если ты всё еще плохо шаришь за указатели. Так вот

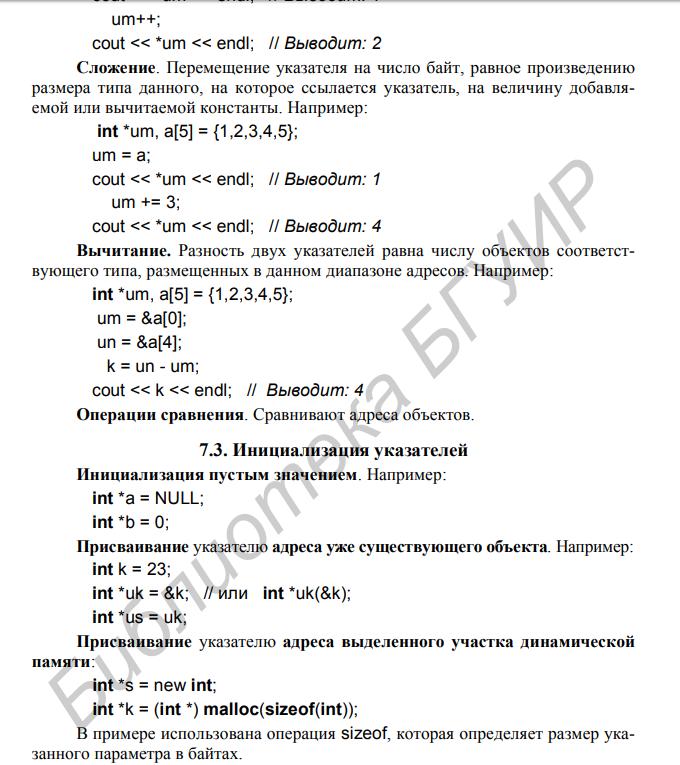
**Inr \*piska, \*ochko;**

**Char \*pidor;**

Звезда-пезда ставится именно перед самой переменной, которая будет указателем, а не после типа. Да, это важно. А еще указатель любого типа занимает 4 байта памяти, тоже вполне себе ловушка от Навра.

Если скрины ниже не помогли понять, то иди лазить в ютабе. Подбери видос на свой вкус и цвет, главное чтоб про указатели.





# 14. Связь указателей с массивами. Создание динамических массивов и правила работы с ними.

* Указатели топ тема при работе со строками матрицы. Пример замены 2х строк местами:

|  |
| --- |
| int \*str1 = arr[0]*;*  int \*str3 = arr[2]*;*   for (int i = 0*; i < N; i++){*  swap(\*(str1+i), \*(str3+i))*;*  } |

# 15. Динамическое выделение памяти с помощью библиотечных функций (операции new, delete)

* **Динамическим** называется массив, размер которого может изменяться во время

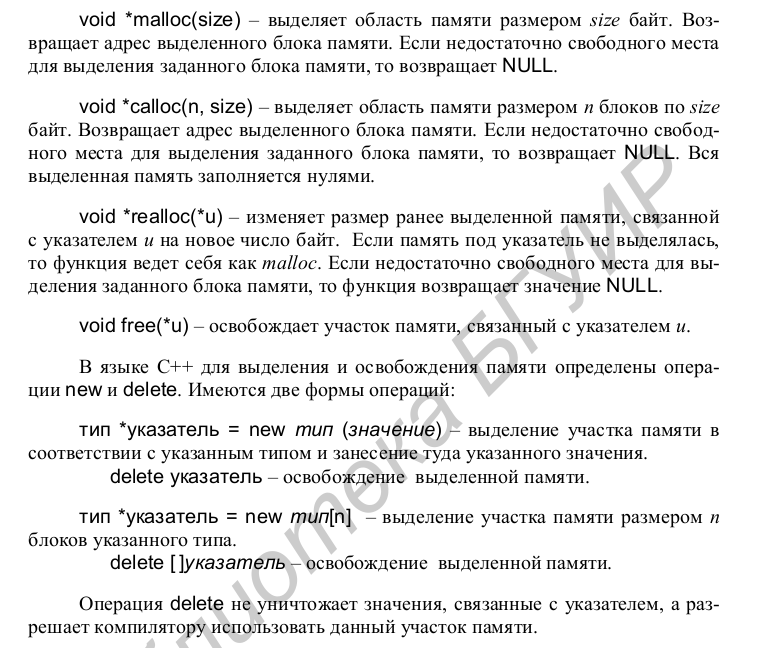
исполнения программы. Возможность изменения размера отличает динамический массив от статического, размер которого задаётся на момент компиляции программы.

Динамические массивы дают возможность более гибкой работы с данными, так как позволяют не прогнозировать хранимые объёмы данных, а регулировать размер массива в соответствии с реально необходимыми объёмами.

* Динамическая память (heap) – специальная область памяти, в которой во

время выполнения программы можно выделять и освобождать место в соответ-

ствии с текущими потребностями. Доступ к выделяемым участкам памяти осуществляется через указатели. Для работы с динамической памятью в языке Си (библиотека malloc.lib) определены следующие функции:



* **Работа с динамическим массивом. (~15 вопрос)**. В С++ операции new и delete

предназначены для динамического распределения памяти компьютера. Операция new выделяет память из области свободной памяти, а операция delete высвобождает выделенную память. Выделяемая память, после её использования должна высвобождаться, поэтому операции **new** и **delete** используются парами. Даже если не высвобождать память явно, то она освободится ресурсами ОС по завершению работы программы. Рекомендую все-таки не забывать про операцию delete.

* **Вкратце**:
  + Одномерный:

|  |
| --- |
| double \*arr = new double[n] arr = new double(n\*sizeof(double)) arr = static\_cast <double\*>(malloc(n\*sizeof(double))); |

* + Двумерный:

|  |
| --- |
| int N = 0, M = 0; *// строки, столбцы*  int \*\*arr;   arr = new int \*[N]; *// Выделение памяти на N строк массива*  for (int i = 0; i < N; i++){  arr[i] = new int [M]; *// Растягивание каждой строки*  }     for(int i=0; i < N; i++)  delete [ ]arr[i];  delete [ ]arr;  arr = nullptr; |

Почитать **подробнее**: <http://cppstudio.com/post/432/>

* **Работа с двумерными массивами.**

|  |
| --- |
| int n, m;   cout << "Rows:";  cin >> n;  cout << "Columnes: ";  cin >> m;  int \*\*arr = new int \*[n]; *// объявление n строк массива*  for (int i=0; i<n;i++){  arr[i] = new int[m]; *// растягивание до m эл-тов*  }   for (int i=0; i<n; i++){ *// заполнение массива*  cout << endl;  for (int j=0; j<m; j++){  cout << "Value of " << "[" << i << "][" << j << "] element: ";  cin >> arr[i][j];  }  }   int s1=0, s2 = 0; *// Нахождение суммы элементов диагоналей.*  for (int i=0; i<n; i++){  s1 += arr[i][n-i-1];  s2 += arr[i][i];  }   int sum = 0; *// Нахождение суммы элементов, лежащих выше главной диаг*  for (int i=0; i<n-1; i++) *// не делать лишних итераций по n + не выйти за размеры в след цикле*  for (int j=i+1; j<m; j++)  sum += arr[i][j];    int k1=1, k2=2, t;  for (int j=0; j<m; j++){ *// Перестановка строк с номерами k1 и k2. Вариант 1.*  t = arr[k1][j];  arr[k1][j] = arr[k2][j];  arr[k2][j] = t;  }    int \*pntr1=arr[1], \*pntr2=arr[2];  for (int j=0; j<m; j++){ *// Перестановка строк с номерами k1 и k2. Вариант 2.*  int t = pntr1[j];  pntr1[j] = pntr2[j];  pntr2[j] = t;  }  */\*Получение из матрицы n-го порядка матрицы порядка n-1 путем удаления из исходной матрицы строк и столбцов, \* на пересечении которых расположен элемент с наименьшим значением \*/*  int imin=0, jmin=0;  for (int i=0; i<n; i++) *// нахождение координат минимального элемента*  for (int j=0; j<m; j++)  if (arr[i][j] < arr[imin][jmin]){  imin = i; jmin = j;  }   *// Работает сдвигом сдвигом эл-тов по строкам и стобцам. Лучше нарисовать.*  for (int i=0; i<n; i++){  for (int j=jmin; j<m-1; j++)  arr[i][j] = arr[i][j+1]; *// смещение всех эл-тов правее минимального в каждой строке на 1 влево.*  }m--; *// это удаляет СТОЛБЕЦ, поэтому в СТРОКЕ становится на 1 элемент меньше*   for (int j=0; j<m; j++){ *// удаляем строку*  for (int i=imin; i<n-1; i++)  arr[i][j] = arr[i+1][j];  }n--; *// количество строк стало на 1 меньше*    for (int i=0; i<n; i++){ *// вывод эл-тов массива*  cout << endl;  for (int j=0; j<m; j++){  cout << " " << arr[i][j];  }  }    for (int i=0; i < n; i++){ *// Очищение памяти*  delete[] arr[i];  }  delete[] arr; |

# 16. Строка – массив типа char. Стандартные функции библиотеки string.h

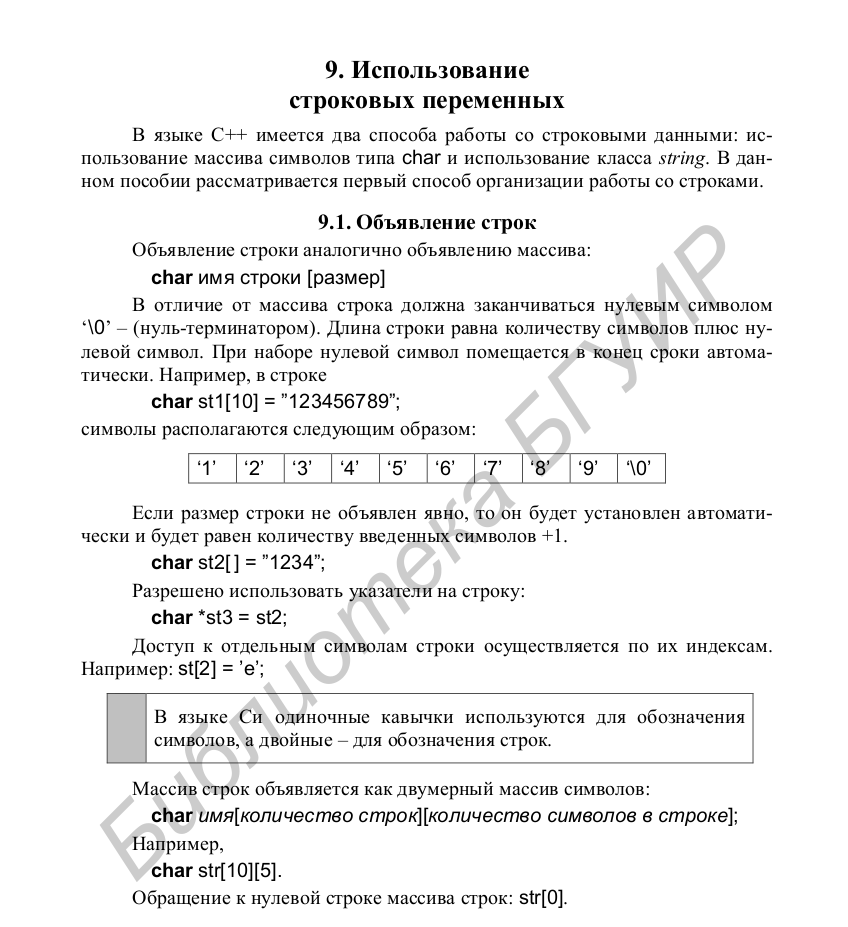
* В методе норм. Функции -- [тут](http://cppstudio.com/post/437/).
* В использовании функций ничего такого нет, но в реальной жизни такой роскоши, как изменение или копирование данных у вас скорее всего не будет (представь бд с тексктом на несколько гб (которые мы кстати проходим на 3м курсе) и что тебе надо обработать текст из нее. Короче, насиловать входные данные -- говнокод.
* Примеры работы:

|  |
| --- |
| int main() {  char str[100];  fgets(str, 100, stdin);  strcat(str, " "); *// пробел в конце, да, так можно*   *// 1. Подсчитать количество слов в строке. Слова отделяются друг от друга одним пробелом. Перед первым словом пробела нет*  int c = 0;  for (int i = 0; str[i] != '\0' ; i++) {  if (str[i] != ' '){  c++;  while (str[i] != ' ') i++;  }  }  cout << endl << c;   *//2. Подсчитать, какое количество букв 'а' в первом слове строки. Слова* *отделяются друг от друга одним пробелом. Перед первым словом пробела нет.*  int j=0, count=0;  while (str[j] != ' '){ *// зачем идти до конца, если мы идем до 1го пробела?*  if (str[j] == 'a') count++; *// TODO: придумайте, как сделать это, если все же есть 1 пробел*  j++;  }  cout << endl << count;  return 0; } |

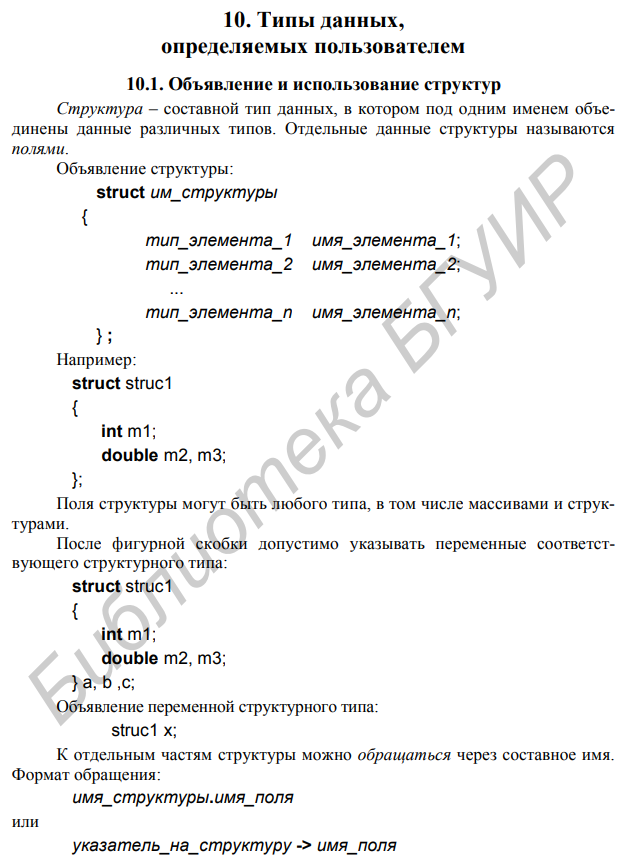
* Полезно также быть тру-джуном и самом писать библиотечные функции:

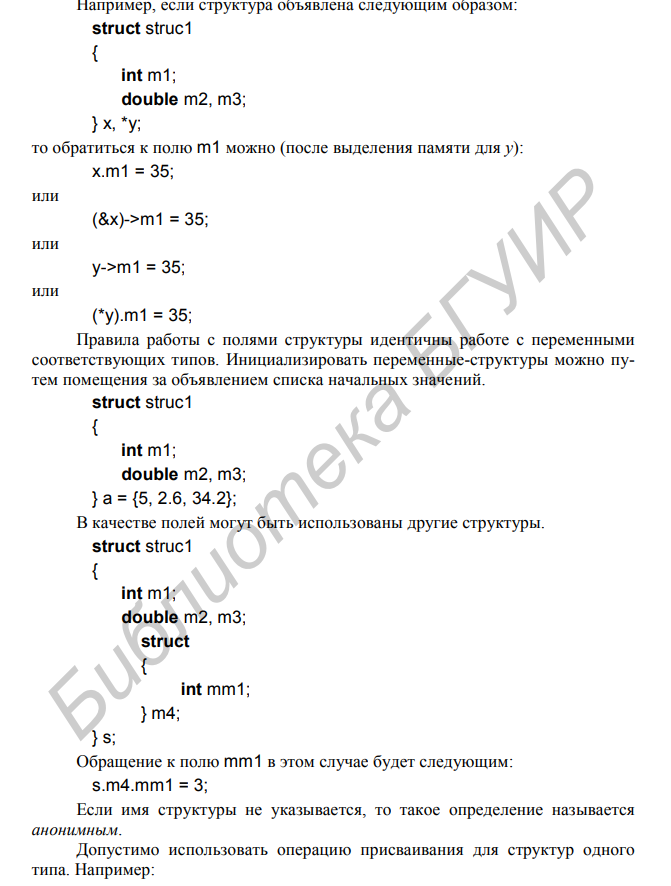
|  |
| --- |
| void strwr(char \*str){  int c = 'a' - 'A'; *// сдвиг по таблице*   for (int i=0; i < strlen(str)-1; i++){  if (str[i] >= 'A' && str[i] <= 'Z')  str[i] += c;  } }  char \*cutstr(char \*str, int n, int k){*// вырезать k символов с n-ной позиции*  static char returnstr[100];  int i = 0  for (int counter=n-1; counter < n - 1 + k; counter ++){  returnstr[i] = str[counter];  i++;  } }  int lengthString(char \*str){  int i = 0;  while (str[i] != '\0'){  i++;  }   return i; }  char \*reverseString(char \*str){  int length = lengthString (str); *// находим длину строки*  char temp; *// временная переменная для хранения переставляемого символа*   for (int counter = 0; counter < (length / 2); counter++){  temp = str[length - 1 - counter]; *// символ с конца строки сохраняем во временную переменную*  str[length - 1 - counter] = str[counter];  str[counter] = temp;  }  return str; } |

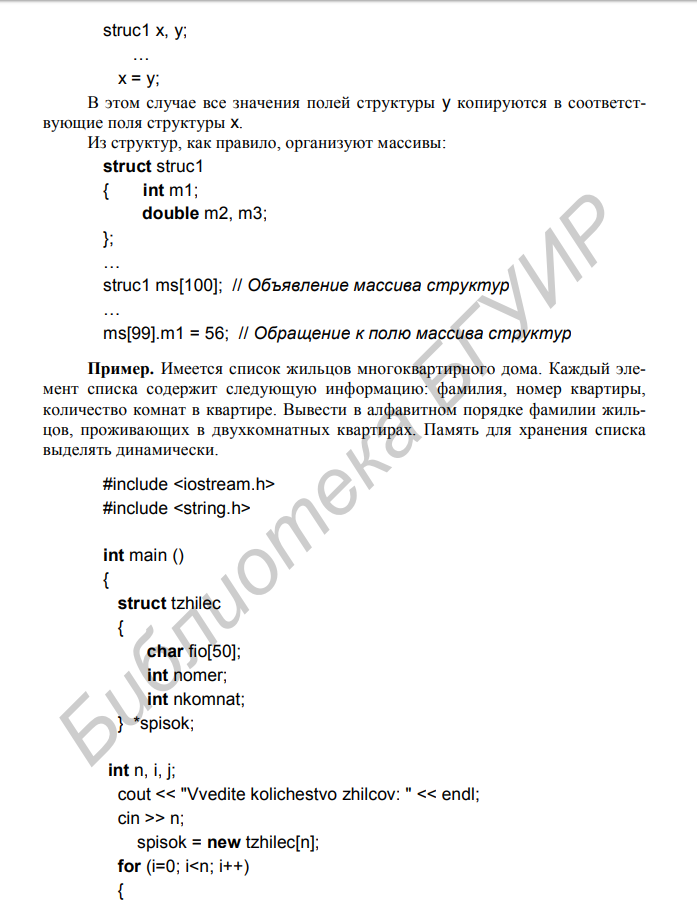
* **Замечание** Для ввода в разных IDE используются разные функции:
  + Старые версии вижлы: gets(str);
  + Новая версия, там немного изменили библиотеку: gets\_s(str);
  + Работает и cin >> str, но могут возникнуть проблемы с буфером.
  + Для линуксоводов или просто нормальных людей: fgets(str, 100, stdin);

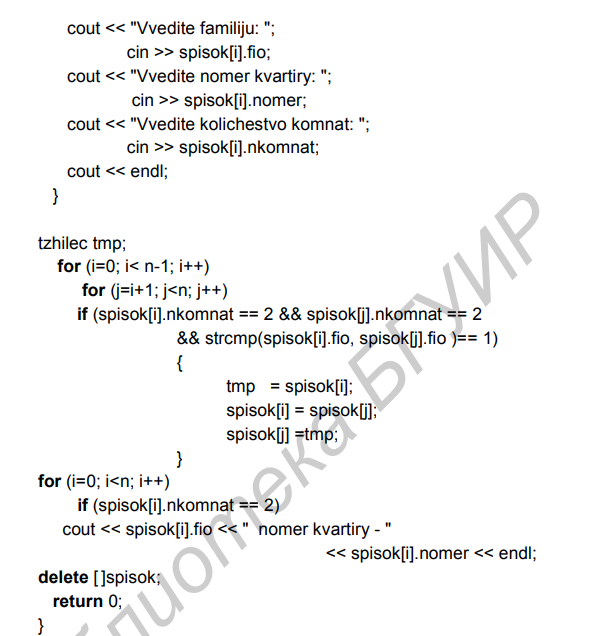


# 17. Декларация структуры (struct). Создание структурных переменных. Обращение к элементам структуры. Вложенные структуры.









# 

# 18. Перечисления (enum), объединения (union).

Дзен и смысл этих двух придет чуть позже, но основы таковы:

* **Объединения (union) -**- это объект, позволяющий нескольким переменным различных типов занимать **один участок памяти**. Объявление объединения похоже на объявление структуры:

|  |
| --- |
| union union\_type {  int a; double b; }; |

Размер памяти, выделяемый под объединение равен **размеру самого большого поля**. Т.е. такая union\_type будет занимать только 8 байт памяти(double), при этом такая же структура занимала бы 8+4=12 байт.

Объединения могут использоваться для: 1) Для экономии памяти (особенно во встроенных системах). 2) Для исследования значений отдельных байтов многобайтных величин. 3) Для интерпретации данных, расположенных в некоторой области памяти.

* **Перечисления (enum)** -- тип данных, который включает множество именованных целочисленных констант. Именованные константы, принадлежащие перечислению, называются перечислимыми константами. Объявление: enum color { r, g, b };

Тут по порядку: r=0, g=1; b=2.

* Остальное -- [тут](http://www.c-cpp.ru/books/perechisleniya). Пример использования:

|  |
| --- |
| enum days\_of\_week { Sun, Mon, Tue, Wed, Thu, Fri, Sat }; int main() {  days\_of\_week day1, day2; *//define variables of type days\_of\_week*  day1 = Mon;  day2 = Sat;   int diff = day2 - day1; *//can do integer arithmetic*  cout << "Days between = " << diff << endl;   if(day1 < day2) *//can do comparisons*  cout << "day1 comes before day2" << endl;  return 0; }  */\*  Days between = 6  day1 comes before day2 \*/* |

# 19. Понятие функции, описание и определение функции. Вызов функции.

* **Функция** - это совокупность объявлений и операторов (именованный блок кода), обычно предназначенная для решения определенной задачи. Каждая функция должна иметь имя, которое используется для ее объявления, определения и вызова.

# 

# 20. Передача данных в функцию по значению, по указателю, по ссылке.

* Прежде всего -- повторить функции и указатели. Если мы передаем данные по значению, то мы передаем их **копию** --> сами данные не изменяться вне функции. Если же по ссылке или указателю (что почти одно и то же), то мы работаем непосредственно с переданными данными и любое их изменение в функции изменит их глобально. В принципе, технически разница между указателем и ссылкой лишь в том, что саму ссылку нельзя изменить (а указатель можно). [[Вот тут](https://ru.stackoverflow.com/q/349/10105) бóльший список отличий.] Кроме этого, разница ещё синтаксическая: с ссылкой вы обращаетесь как будто это переменная, а с указателем нужно его правильно получать/разыменовывать:
* Примеры:

|  |
| --- |
| void Foo1(int a){ *// передача по значению*  a = 1; *// работа с копией* }  void Foo2(int &a){ *// передача по ссылке*  a = 2; *// не нужно разыменовывать* }  void Foo3(int \*a){ *// передача по укзателю*  \*a = 3; *// разыменование* } |

* При работе с этими функциями:

|  |
| --- |
| int value = 5;  cout << "value: " << value << endl;   cout << "Foo1: " << value << endl;  Foo1(value);  cout << "value: " << value << endl << endl;   cout << "Foo2: " << value << endl;  Foo2(value); *// хоть функция и получит адрес, передаем без &*  cout << "value: " << value << endl << endl;   cout << "Foo3: " << value << endl;  Foo3(&value);  cout << "value: " << value << endl << endl;  *// a = 5*  *// a = 5 // т.к. работаем с локальной копией переменной*  *// a = 2*  *// a = 3* |

# 21. Параметры функций по умолчанию, функции с переменным числом параметров.

# 22. Встраиваемые функции. Перегрузка функций. Передача массивов в функцию. Указатель на функцию.

* Встроенные -- [тут](http://cppstudio.com/post/2676/). (Вкратце -- сразу использование кода, быстрее)
* Перегрузка -- [тут](http://cppstudio.com/post/406/). (Вкратце -- одно имя на разные функции, потому что параметры)
* Передача массива. Одномерные -- [тут](https://habr.com/sandbox/111348/) и [тут](https://ru.stackoverflow.com/questions/136769/%d0%9a%d0%b0%d0%ba-%d0%bf%d0%b5%d1%80%d0%b5%d0%b4%d0%b0%d1%82%d1%8c-%d0%bc%d0%b0%d1%81%d1%81%d0%b8%d0%b2-%d0%b2-%d1%84%d1%83%d0%bd%d0%ba%d1%86%d0%b8%d1%8e). Двумерные -- вот [тут](https://stackoverflow.com/questions/8767166/passing-a-2d-array-to-a-c-function) прям хорошо.

Примеры для динамических:

1. Одномерный

|  |
| --- |
| void Foo1(int \*, int);  int main(){  int N;  cin >> N;  int \*arr = new int [N];   for (int i=0; i<N; i++){ *// заполнение массива*  cin >> arr[i];  }  Foo1(arr, N);  delete []arr; *// освобождайте память блять* }  void Foo1(int \*arr, int N){ *// функция вывода*  for (int i=0; i<N; i++){  cout << arr[i];  } } |

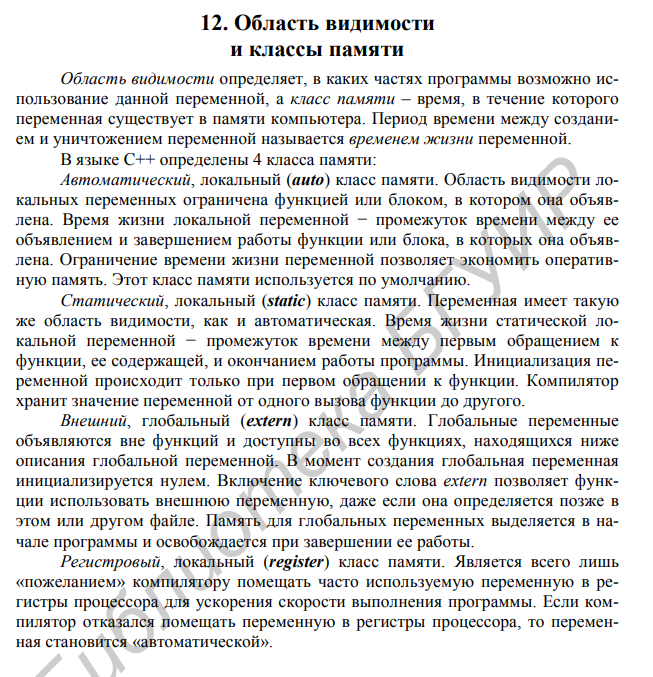
1. Двумерный:

|  |
| --- |
| void Foo1(int \*\*, int, int); int main(){  int N, M;  cin >> N >> M;   int \*\*arr = new int \*[N];  for (int i = 0; i < N; i++) {  arr[i] = new int [M];  }   for (int i=0; i<N; i++){ *// заполнение массива*  cout << endl;  for (int j = 0; j < M; j++) {  cin >> arr[i][j];  }  }  Foo1(arr, N, M);   for(int i=0; i < N; i++) *// удаляйте их блять*  delete [ ]arr[i];  delete [ ]arr; }  void Foo1(int \*\*arr, int N, int M){ *// функция вывода*  for (int i=0; i<N; i++){  cout << endl;  for (int j = 0; j < M; j++) {  cout << arr[i][j];  }  } } |

* **Указатель на функцию.** Сложная и не очень полезная фигня. [Тыц](http://www.c-cpp.ru/books/ukazateli-na-funkcii) и [Тыц](https://purecodecpp.com/archives/2301). (Вкратце -- функция тоже объект в памяти → на него можно ссылаться. Удобно только при создании массива функций. Например, для тестов.)

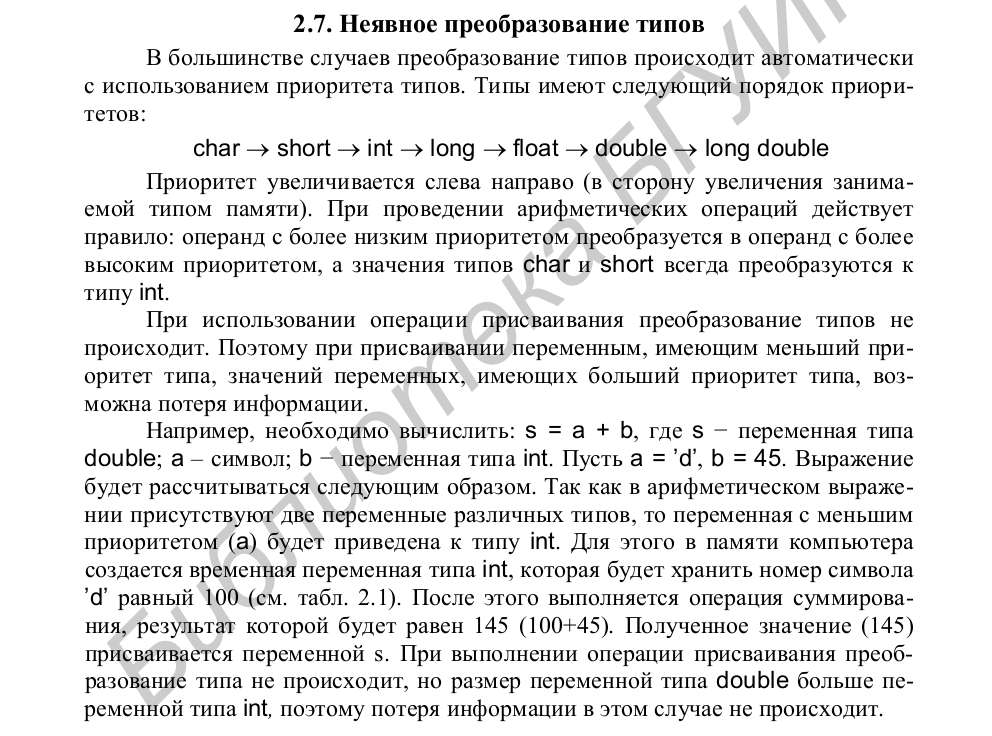
# 23. Классы памяти. Время жизни и область видимости переменных.

Переменная, которую ты объявил глобально, правильно называть **голая жопа в окне** © Рак + Шатилова.



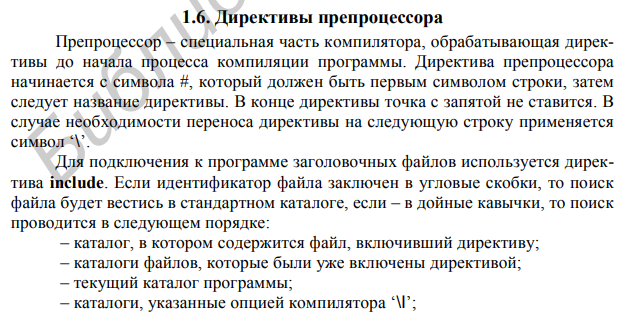
# 24. Стандартные библиотечные функции для организации ввода-вывода информации (getc, gets, scanf, putc, puts, printf). Спецификации преобразований для данных различных типов.

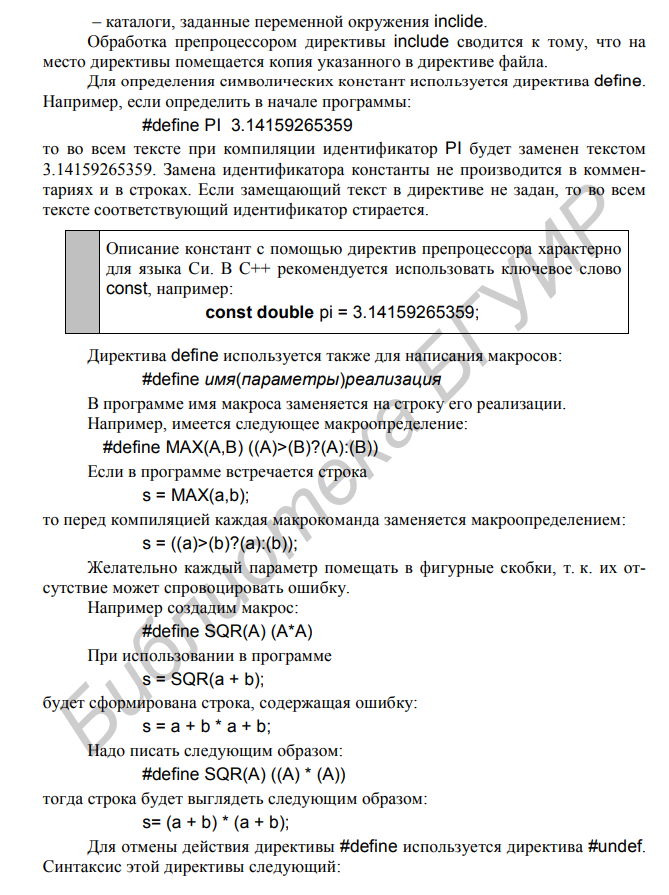
* Функции языка С для работы с файлами. Текстовый ввод-вывод. Почитать подробно -- [тут](http://cpp.com.ru/kr_cbook/ch7kr.html).
* Ссылочки отдельно:
  + [getc](http://cppstudio.com/post/1688/), [gets](http://cppstudio.com/post/1699/)
  + [putc](http://cppstudio.com/post/1696/), [puts](http://cppstudio.com/post/1692/),
  + [scanf, printf](https://fresh2refresh.com/c-programming/c-printf-and-scanf/).
* Спецификации типов -- предмет документации, который определяет язык программирования, чтобы пользователи и разработчики языка могли согласовывать, что означают программы на данном языке [[wiki](https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B5%D1%86%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F_%D1%8F%D0%B7%D1%8B%D0%BA%D0%BE%D0%B2_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)].
* Преобразования типов: явное (ручное) и неявное(компилятор умный). [Тыц](http://cppstudio.com/post/310/) и [тыц](https://purecodecpp.com/archives/2266).

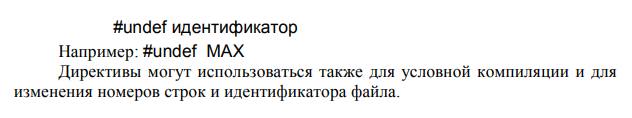


# 25. Директивы препроцессора.

using namespace std -- использование пространства стандартных имен библиотеки.

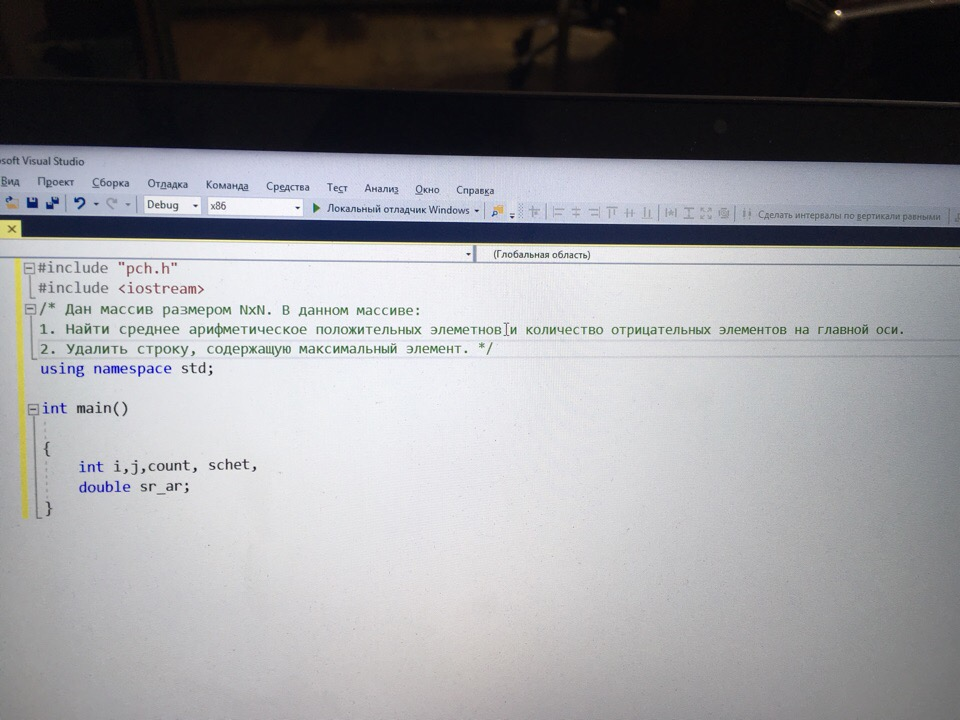






# Задачи с экза.

**1)**



**2)** Динамический массив. Вывести четные элементы массива. Те m[i] четный

Отрицательные элементы массива заменить на нули

Второе задание это элементы равные нулю заменить на полусумму соседних, кроме 1 и последнего элемента

**3)** Билет #20. 1) Динам.массив, вывести сумму его отрицательных элементов, ко всем четным числам массива прибавить 5. 2) Отсортировать так, чтобы сначала шли отриц.элементы, а потом положительные не меняя их взаиморасположения. 3) циклы фор и вайл

**4)** Мат NxN, найти кол-во ненулевых элементов матрицы, произведение положительных

Дан массив B.Если в строке матрицы все положительные, то массиву присваиваю 1, в противном - 0

**5)** У меня было задание с одномерным динамическим

Сначала найти мин и макс значения, потом вывести массив, в котором последние 4 числа заменяются на 9. И надо было найти наиболее повторяющееся число

**6)** Дана строка, слова разделены одним или несколькими пробелами, вывести слова, содержащие “www”. Вывести слова, у которых первый и последний символ одинаковые, а второй и третий разные. Связь указателей с массивами. Создание динамических массивов и правила работы с ними.

**7)** Билет 12

1. Дан динамический одномерный массив (число элементов ввести с клавиатуры). Подсчитать среднее арифметическое положительных элементов. Отрицательные элементы умножить на два. Вывести массив и результаты вычислений. Очистить кучу.

2. Элементам с чётными индексами(кроме последнего) присвоить значение полусуммы соседних элементов. Вывести массив.

3. Арифметические операции. Преобразование типов(явное и неявное)

**8)**

1. Вывести значение функции y=(e^x-e^(-x))/2 для икса от a=0.3 до b=0.9 с шагом h=0.07.

2. Посчитать рекуррентно сумму s=...(не помню, соре, мб в лабах есть, прочекай) при помощи функции s(x, N) для N=1200 слагаемых

3. Первый вопрос из списка теории

**9)**

