1. // это комментарий
2. Для того чтобы пользоваться функциями , необходимо подключить библиотеки;

#UNCLUDE <>



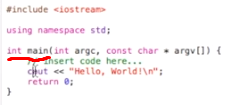
1. Чтобы подключить функции стандартной библиотеки std, и не писать каждый раз так :



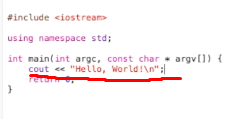
Нужно написать USING NAMESPACE STD



1. Int main – основная функция(тело программы) тут мы пишем наш код



1. Оператор вывода на экран cout<<



1. Оператор ввода у пользователя cin>>



1. Типы переменных:

Int – целый

Char –



Bool – булевый тип



Float – плавающая точка

Unsigned int – целые числа больше 0



1. Арифметические операции:

+,-,\*,/

% остаток от деления



(Тип один раз включает в себя 6, а остаток уже не может поделиться на 6)

-= у переменной отнимаем то, что после =

+=

\*=

/=



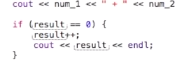
++ прибавляем единицу

--

9. Условия IF ()

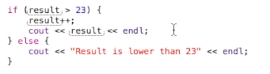
{

}

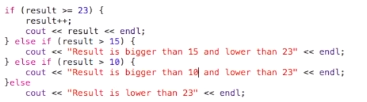


Else {

}



Так же можно использовать ELSE IF для проверки другого условия



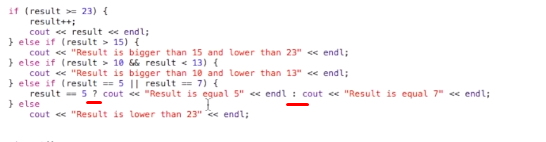
В условии можно использовать:

&&- логическое И

|| - логическое ИЛИ

? – если да

: - иначе



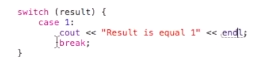
1. Проверка на соответствие switch (переменная){

Case x-то чему равна:

Что то делаем

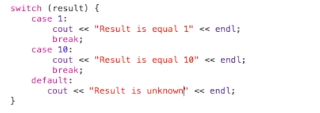
Break;

}

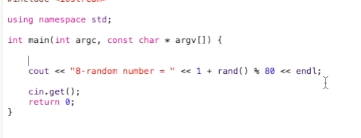


И так можно много кейсов

Так же используется default, и если все кейсы не выполнились то выполнится дефолт

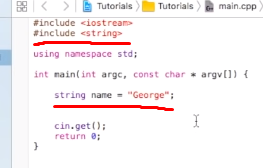


1. Генератор чисел rand():

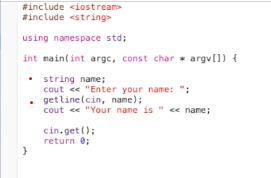


В данном случае остаток от деления используется для задания ограничения интервала случайных чисел

1. Чтобы использовать тип переменных string и хранить в ней строки, подключаем библиотеку string



1. Чтобы записать в переменную то, что вводит пользователь используем функцию getline(cin, переменная)

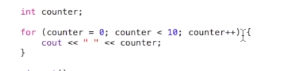


1. Проверить длинну строки переменной можно с помощью length



1. Циклы FOR, WHILE, Do WHILE:

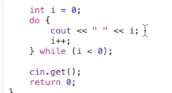
* For(условие ) {тело}



- WHILE (условие) {тело}



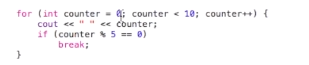
- DO {тело} WHILE(условие)



Этот цикл выполнится минимум 1 раз в любом случае, даже если не удовлетворяет условию

В этих циклах используется BREAK и CONTINUED

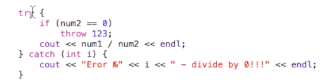
BREAK – если условие срабатывает, выходит из цикла



CONTINUE будет пропускать шаг



1. Исключения (Нужны чтобы поймать ошибку, которая может возникнуть)



TRY { тело и тут пишем throw (бросаем то, что потом ловим)}

Catch(переменная которую ловим) {тело}

1. Массивы (задается [])



Если мы сразу определяем содержимое массива, то число элементов в [] не нужно

- Перебрать массив удобно через цикл



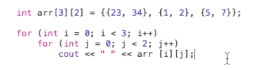
- Данные в массиве легко меняются



1. Можно создать двумерные (и еще больше) массивы:



* Перебрать его можно двумя циклами



1. Константы const (нужны для того чтобы переменную нельзя было изменить в дальнейшем)

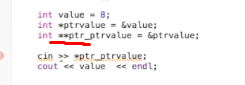


1. Указатели нужны для оптимизации кода и дальнейшего передавания переменной в функцию ( по соглашению называем указатели \*ptrназвание переменной)



Так же мы можем менять переменную используя указатель

Можно делать указатель на указатель



1. Чтобы узнать адрес переменной, перед ней нужно поставить &, а у указателя убрать\*



1. Ссылки (скрытый указатель)



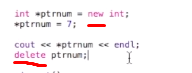
Суть та же что и с указателями с небольшими отличиями, указатели работаю с динамическими объектами которые изменяются в функциях, а ссылки дает прямой доступ к объекту

1. Динамические массивы

Выделить динамическую память под объект:

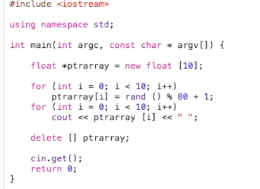


И если у нас будет использоваться динамическая память, то ее в конце выполнения необходимо закрывать иначе по мере накопления кода будет много памяти использоваться



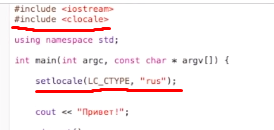
Динамическое выделение памяти в основном применяют к массивам



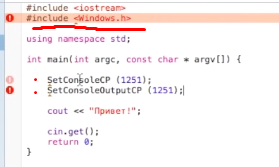


1. Символы и строки

Для вывода кирилицы необходимо использовать setlocale(), заранее подключив библиотеку:



Для того чтобы вводить кириллицу на виндовс, можно сделать так



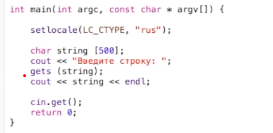
Для строк модно использовать char как массив символов:

и стоит помнить что один символ нужно зарезервировать под нуль терминатор \0

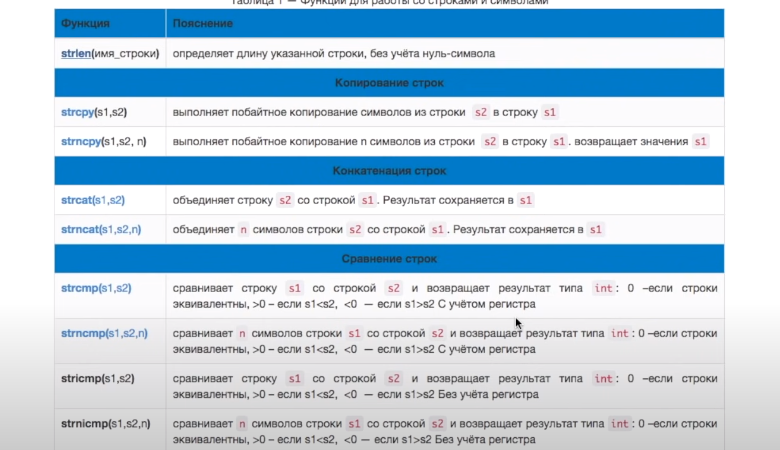
Но правильно так:

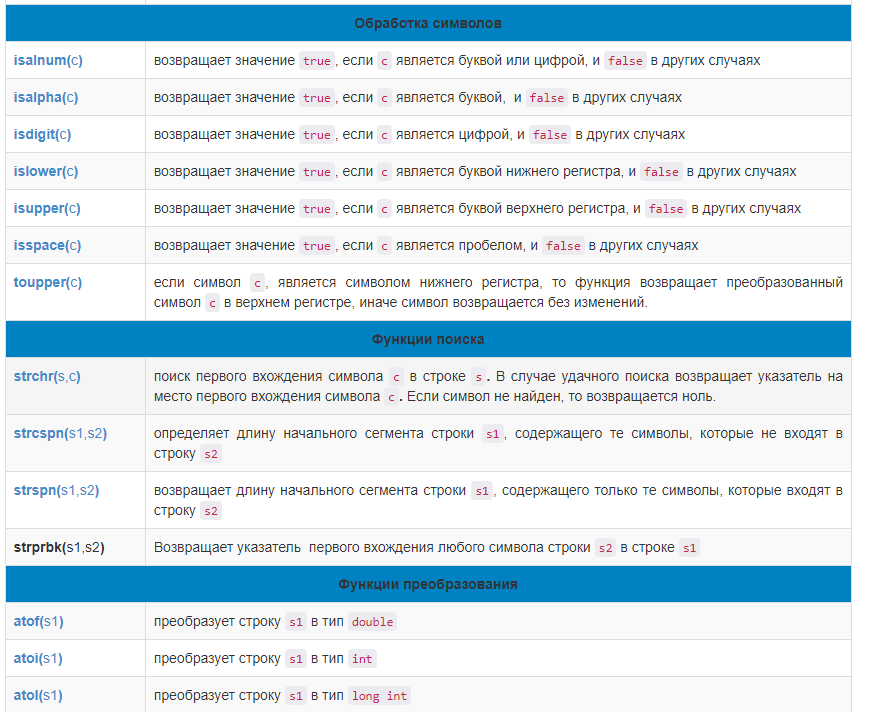


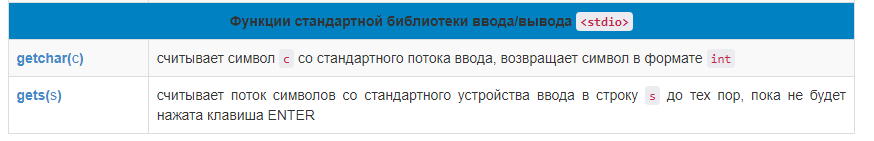
1. Ввести данные можно с помощью функции gets, но это не очень безопасно



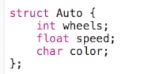
1. Другие функции для строк:



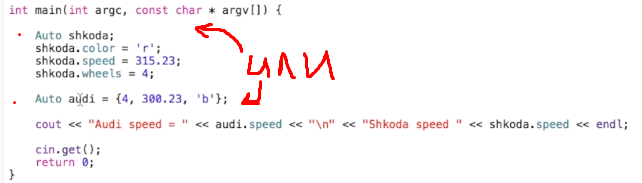




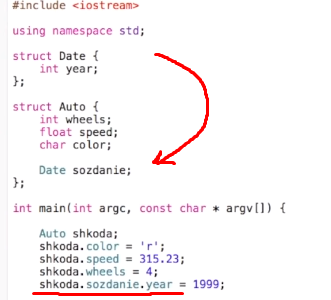
1. Структуры struct (создаются вне функции main) очень похожи на классы



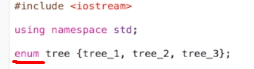
А использовать можно так:



Можно использовать вложенные структуры:



1. Enum (Перечисление)



Прописывается перед main

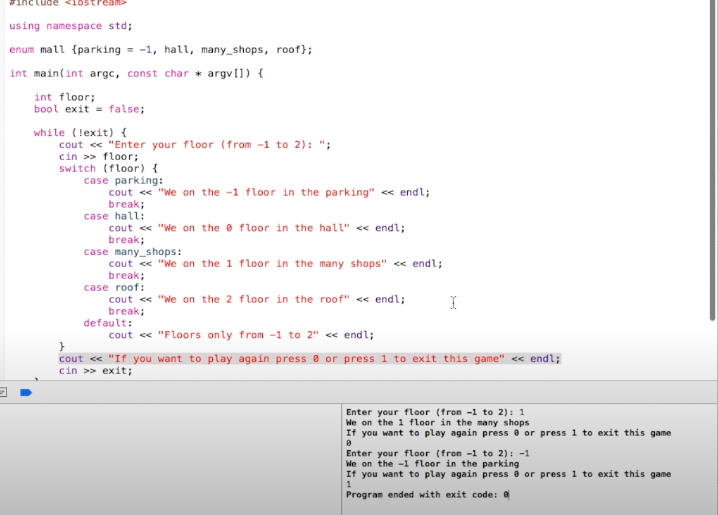
И при выводе он эти элементы пересчитывает



Им можно присваивать значения:



Пример программы:

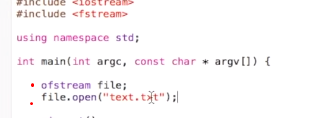


1. Работа с файлами

Необходимо подключить библиотеку



Для того чтобы начать работать с файлом мы используем ofstream(создавать и записывать файл)



Причем если файла такого нет, он будет создан

Для записи используем аналогию с cout<<



После работы с файлом его необходимо закрыть

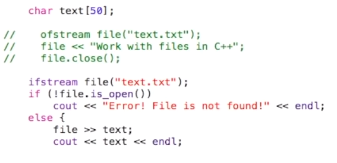
Часто строчку open опускают:



Для считывания из файла используем ifstream и для того чтобы начать считывать с файла, его нужно открыть

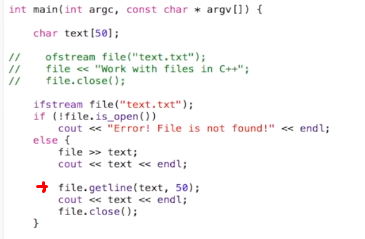


Для пользования можно предварительно сделать проверку на открытие файла, если он не открыт то получим ошибку, а если открыт в блоке else пишем то что нам нужно

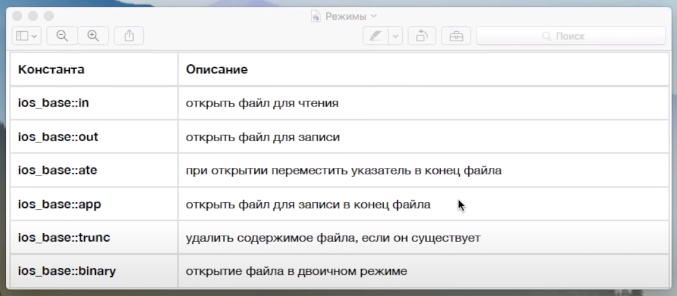


Причем с так считается только первое слово из файла

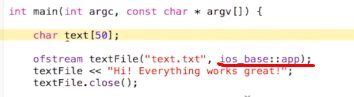
А так мы запишем в переменную 50 символов, с помощью функции getline(переменная, число символов)



Существует много констант для работы с файлами



Пример :



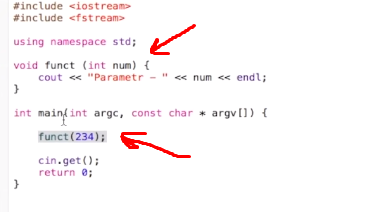
1. Функции

Если у нас встречается два раза один и тот же код, то нужно задуматься о создании функции

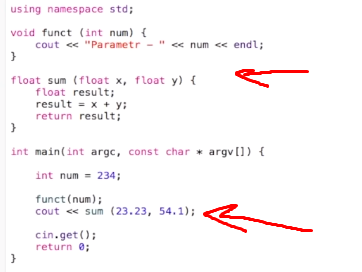
Функцию необходимо создавать не в другой функции, то одна и та же функция может использовать и вызывать саму себя

Для создания функции, которая ничего не возвращает используем VOID, а если возвращает то используем по ситуации int, float и тд

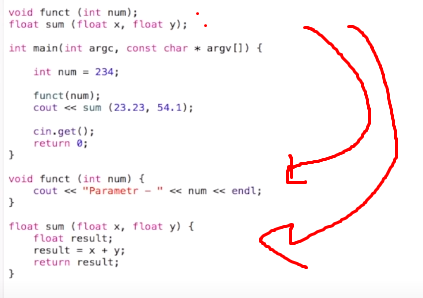
В круглых скобках у функции мы передаем параметры



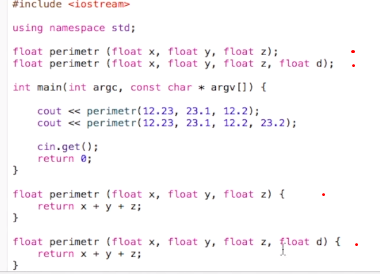
Другой пример:



Если же мы прописали функции ниже чем main, то для работы нам необходимо сделать прототип на наши функции перед main( и это правила хорошего тона)

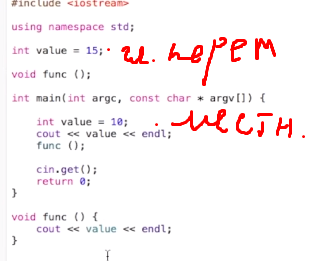


1. Перегрузка функций( одну и ту же функцию можно переопределять(в данном случае добавить еще одну переменную) и все будет работать, компилятор сам решает что мы хотим использовать)



1. Локальные и глобальные переменные

Чтобы использовать переменные в функциях, они должны быть определены глобально, а не в других функциях



Чтобы использовать глобальную переменную можно использовать ::



1. Математические операции

Для использования мат. Операций нужно подключить библиотеку



Pow – возведение в степень

ABS – число по модулю

Sin, Cos

Sqrt – квадратный корень

Ceil – округление до целого

Floor – округление до наименьшего целого

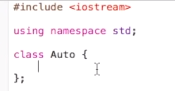
Exp – экспонента

Log – логарифм

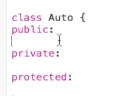
Log10 – десятичный логарифм

1. ООП (классы и объекты)

Создать класс :



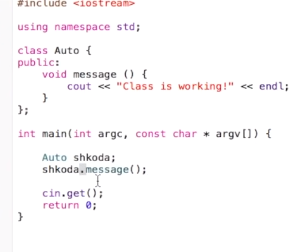
В нем используются модификаторы



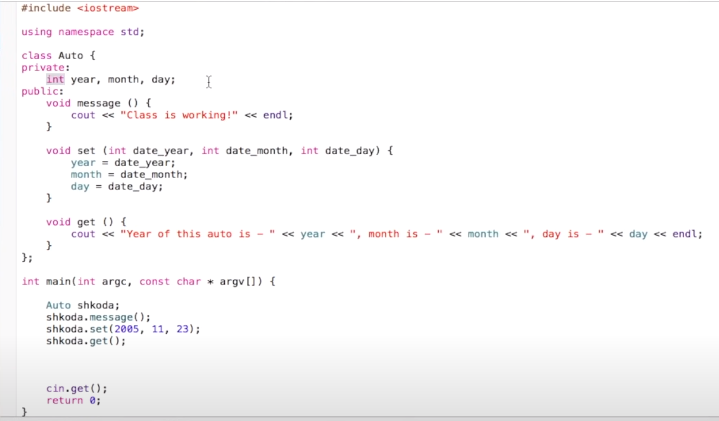
**В public мы пишем переменные или функции, к которым хотим дать доступ при обращении к классу**

**В private то что используется только в классе и больше нигде**

**В protected то, что будет использоваться только наследниками класса и больше нигде**



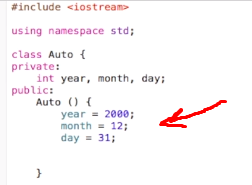
Пример программы:



1. Конструкторы и деструкторы

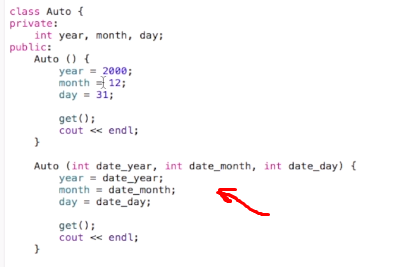
Конструктор необходимо указывать в public (а необходим он для инициализации всех необходимых переменных)

Конструктор должен называться как класс



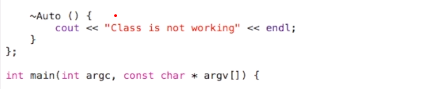
Суть в том, что при создании класса будут уже с параметрами

Можно использовать перегрузку функций



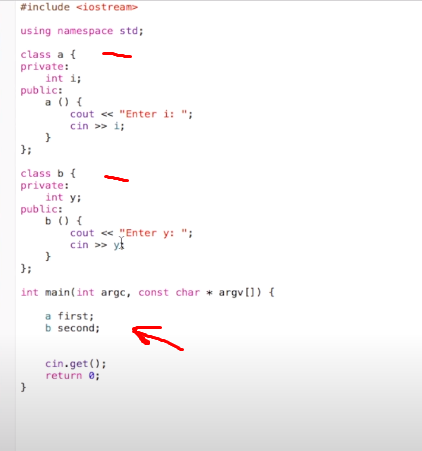
В этом примере мы используем 3 параметра и программа сама понимает когда мы хотим использовать эту функцию

Деструткор ( для удаления конструктора) ~



1. Дружественные функции

Для примера создадим два класса

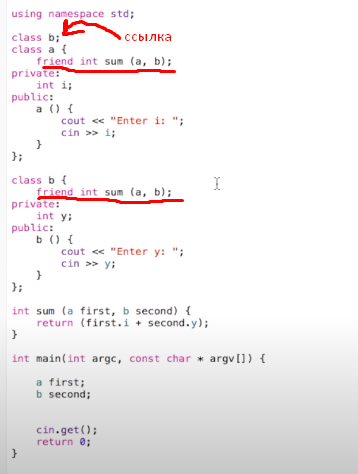


Потом создаем функцию которая бы складывала эти два числа:

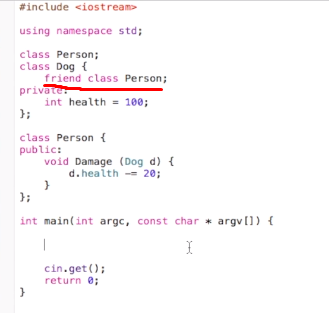


Но пока работать она не будет, так как переменные в классах в привате.

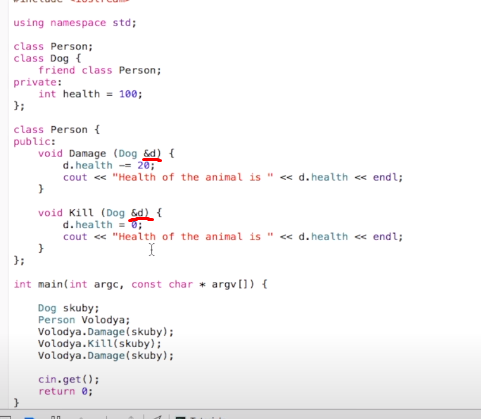
Для решения этого вопроса мы используем дружественные функции (указываем их в классе)



1. Дружественные классы ( по аналогии с функциями)

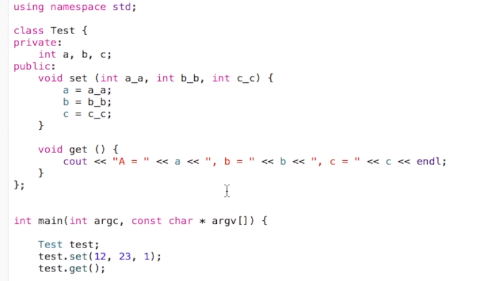


Но если сделать так, то результат урона не будет сохраняться и для этого нам и нужны ссылки на объект

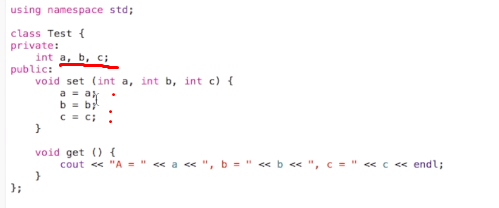


1. Указатель THIS

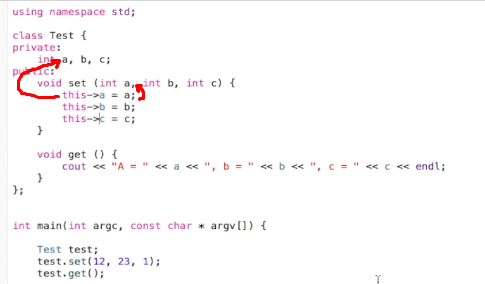
Создаем программу



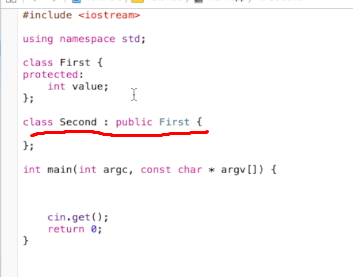
Но нам не хочется придумывать имена переменных и хочется написать проще



Но компилятор не поймет эту запись и чтобы решить проблему (this->)

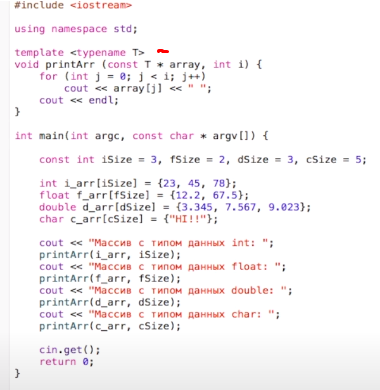


1. Наследование классов



Унаследует все, кроме того что находится в private

1. Шаблоны функций ( template< > )



Нужен чтобы не писать много функций, а сделать одну на все

1. Шаблоны классов

