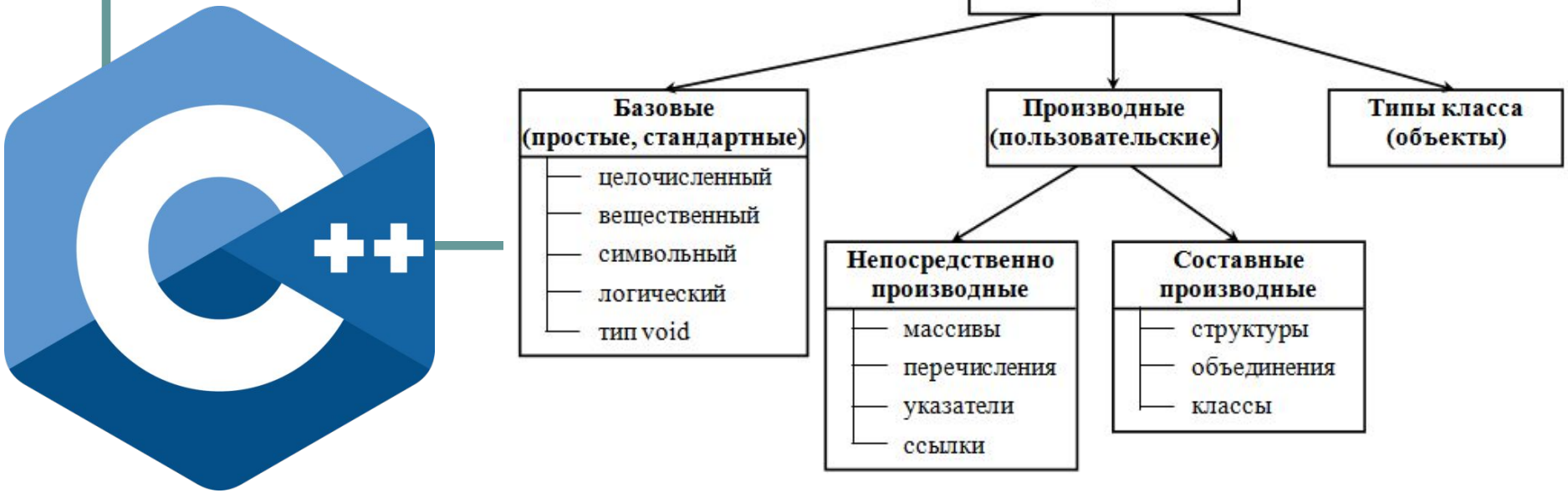


- Этапы разработки ПО
- Что такое алгоритм?
- Что такое #include?
- Что такое main?
- Какая команда печатает текст на экране консоли?
- Что такое escape-последовательности?

- Прописных и строчных букв латинского алфавита (A-Z, a-z). Прописные и строчные буквы в коде различаются, это свойство называется **чувствительностью к регистру символов** (регистрозависимость). Примеры регистрозависимых языков: Java, C++, C#. Примеры регистронезависимых языков: HTML, SQL.

- Цифр от 0 до 9
- Пробельных символов (пробел, горизонтальная табуляция TAB, переход на следующую строку ENTER)
- Специальных символов: **, . ; : _ + - * / % < > = ^ ? ! & | ~ ( ) { } [ ] “ ‘ #**

**Лексема** – это наименьшая неделимая часть языка, которую распознает компилятор. Из лексем составляются все языковые конструкции.

- идентификаторы (identifiers)
- ключевые слова (keywords)
- литералы (literals)
- разделители (separators)
- операторы (operators)

**Идентификаторы** - это имена, которыми обозначаются различные объекты программы, определяемые программистом (переменные, функции, классы и тд.)

Идентификатор обязан быть уникальным. Может состоять из букв латинского алфавита, цифр, символа подчеркивания. Идентификатор не может начинаться с цифры!

Язык C++ регистрозависим – большие и маленькие буквы в лексемах различаются:

Name

name

nAmE

- это совершенно разные лексемы!

**Ключевые слова** (keywords) - это зарезервированные, служебные слова, которые нельзя использовать в своих целях (например, идентификатор не может быть ключевым словом).

**abstract, auto, bool, break, case, catch, char, class, const, continue, decltype, default, delete, do, double, dynamic_cast, else, enum, explicit, export, extern, false, final, float, for, friend, goto, if, inline, int, long, mutable, namespace, new, nullptr, private, protected, public, return, short, signed, sizeof, static, struct, switch, template, this, throw, true, try, typedef, typeid, typename, union, unsigned, using, virtual, void, volatile, while**

https://msdn.microsoft.com/uk-ua/library/2e6a4at9(v=vs.110).aspx

**Литерал** - это лексема, жёстко прописанная в коде программы, которая представляет собой фиксированное значение определённого типа.

Литералы позволяют задать в программе значения для числовых, символьных и строковых выражений.

В C++ определены следующие виды литералов:

- целочисленный (integer)
- дробный (floating-point)
- булевский (boolean)
- символьный (character)
- строковый (string)

**Оператор** – это конструкция языка программирования, которая выполняет определённое действие над аргументами (операндами).

**Операнд** - это аргумент оператора, то есть то значение, над которым оператор выполняет действие.

По количеству операндов операторы делят на:

- **Унарные** – требуют наличия 1 операнда:

-5

level++

- **Бинарные** – требуют 2 операнда:

3 * 6

2 + 2

- **Тернарный** – состоит из трёх операндов:

```
int max = a > b ? a : b;
```

Примеры операторов:

```
+   - *   /   =     ++   -- >>     <=   ==
```

Операторы отличаются:

- Количеством операндов
- Приоритетом
- Ассоциативностью

| Приоритет | Операция | Описание | Порядок выполнения |
|---|---|---|---|
| 1 | ++ | Префиксный инкремент | Слева направо |
| | -- | Префиксный декремент | |
| | ( ) | Подвыражение | |
| 2 | ! | Логическое отрицание | Справа налево |
| | - | Унарный минус(изменение знака | |
| | + | Унарный плюс | |
| | (type) | Преобразование к типу | |
| | sizeof | Определение размера в байтах | |
| 3 | * | Умножение | Слева направо |
| | / | Деление | |
| | % | Остаток от деления | |
| 4 | + | Бинарный плюс (сложение) | Слева направо |
| | - | Бинарный минус (вычитание) | |
| 5 | < | Меньше | Слева направо |
| | > | Меньше или равно | |
| | <= | Больше | |
| | >= | Больше или равно | |
| 6 | = = | Равно | Слева направо |
| | != | Не равно | |
| 7 | && | Логическое И | Слева направо |
| 8 | \|\| | Логическое ИЛИ | Слева направо |
| 9 | ?: | Операция условия | Справа налево |
| 10 | = | Присваивание | Справа налево |
| | += | Составное сложение | |
| | -= | Составное вычитание | |
| | *= | Составное умножение | |
| | /= | Составное деление | |
| | %= | Составное определение остатка от деления | |
| 11 | , | Операция «запятая» | Слева направо |
| | ++ | Постфиксный инкремент | |
| | -- | Постфиксный декремент | |

- условные операторы (if, switch)
- операторы цикла (while, do while, for)
- операторы безусловного перехода (return, break, continue, throw)
- метки (case, default, user labels)
- операторы-выражения (любое выражение, которое заканчивается точкой с запятой, является оператором).
- операторы-операции (арифметические, логические, поразрядные и операции сравнения)
- блоки

Одни операторы ставятся перед операндами и называются префиксными, другие — после, их называют постфиксными операторами.

Большинство же операторов ставят между двумя операндами, такие операторы называются инфиксными бинарными операторами.

Разделители – это специальные символы, используемые в коде:

“()”, “[]”, “{}”, “;” “ ” “ ” “ ”

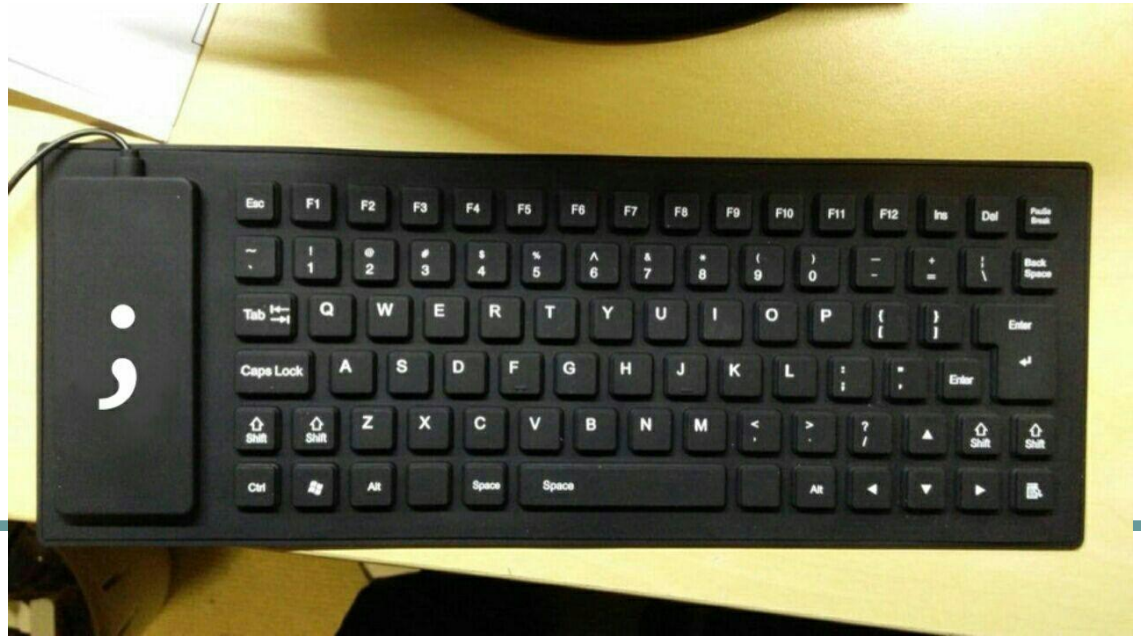
**Выражение** – это корректная комбинация операндов и операторов, которая всегда имеет определённое результирующее значение (результат). Это значение характеризуется типом данных. В качестве операндов могут выступать переменные, константы, литералы, результаты работы функций. Пример выражения:

**5 + x / 8 – (3 * number);**

В результате этого выражения будет **значение** определённого типа данных, которое можно использовать в дальнейшем - например, присвоить переменной.

Почти любая команда (оператор) в языке C++ заканчивается точкой с запятой.

```
cout << "Hello, world!\n";
```

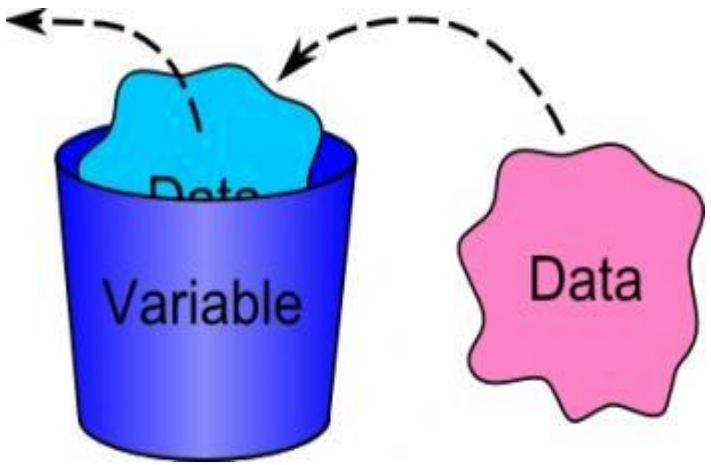


В программе должна существовать функция **main**().

Эта функция вызывается автоматически!

**Переменная** – это именованная область в оперативной памяти, которая может хранить и изменять своё значение на протяжении работы программы. Значение характеризуется типом данных. Для того, чтобы использовать переменную в программе, необходимо её **объявить** (создать). Объявление переменной даст компилятору понять, что для этой переменной необходимо выделить память, что данный идентификатор уже будет использоваться, и как с этой переменной работать в дальнейшем.

Переменные – это **своеобразные контейнеры**, которые могут нести в себе числовые, строковые или логические значения.



**Статическая типизация** – тип данных объекта определяется на этапе компиляции.

Если это происходит на этапе выполнения программы — то **динамическая**.

В C++ используется статическая типизация, а это значит, что программисту придётся выбирать тип для переменной самостоятельно.

тип идентификатор = инициализатор;

**Инициализатор** – это выражение, которое вычисляется в этом месте программы. Им будет инициализирована переменная.

| Тип | байт | Диапазон принимаемых значений |
|---|---|---|
| целочисленный (логический) тип данных | | |
| bool | 1 | 0 / 255 |
| целочисленный (символьный) тип данных | | |
| char | 1 | 0 / 255 |
| целочисленные типы данных | | |
| short int | 2 | -32 768 / 32 767 |
| unsigned short int | 2 | 0 / 65 535 |
| int | 4 | -2 147 483 648 / 2 147 483 647 |
| unsigned int | 4 | 0 / 4 294 967 295 |
| long int | 4 | -2 147 483 648 / 2 147 483 647 |
| unsigned long int | 4 | 0 / 4 294 967 295 |
| типы данных с плавающей точкой | | |
| float | 4 | -2 147 483 648.0 / 2 147 483 647.0 |
| long float | 8 | -9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0 |
| double | 8 | -9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0 |

**bool** (1 байт) – true или false

**char** (1 байт) – от -128 до 127

**short** int (2 байта) – от -32768 до 32767

**int** (машинное слово*) – -+2 147 483 648

**long** int (4 байта) – -+2 147 483 648

**long long** int (8 байт) – от $-9*10^{18}$

**float** (4 байта) - 3.4E-38…3.4E38

**double** (8 байт) - 1.7E-308…1.7E308

*** Размер типа данных int в языке C++ – стремится быть равен машинному слову: не меньше чем short, и не больше чем long!**

Как должен работать душ

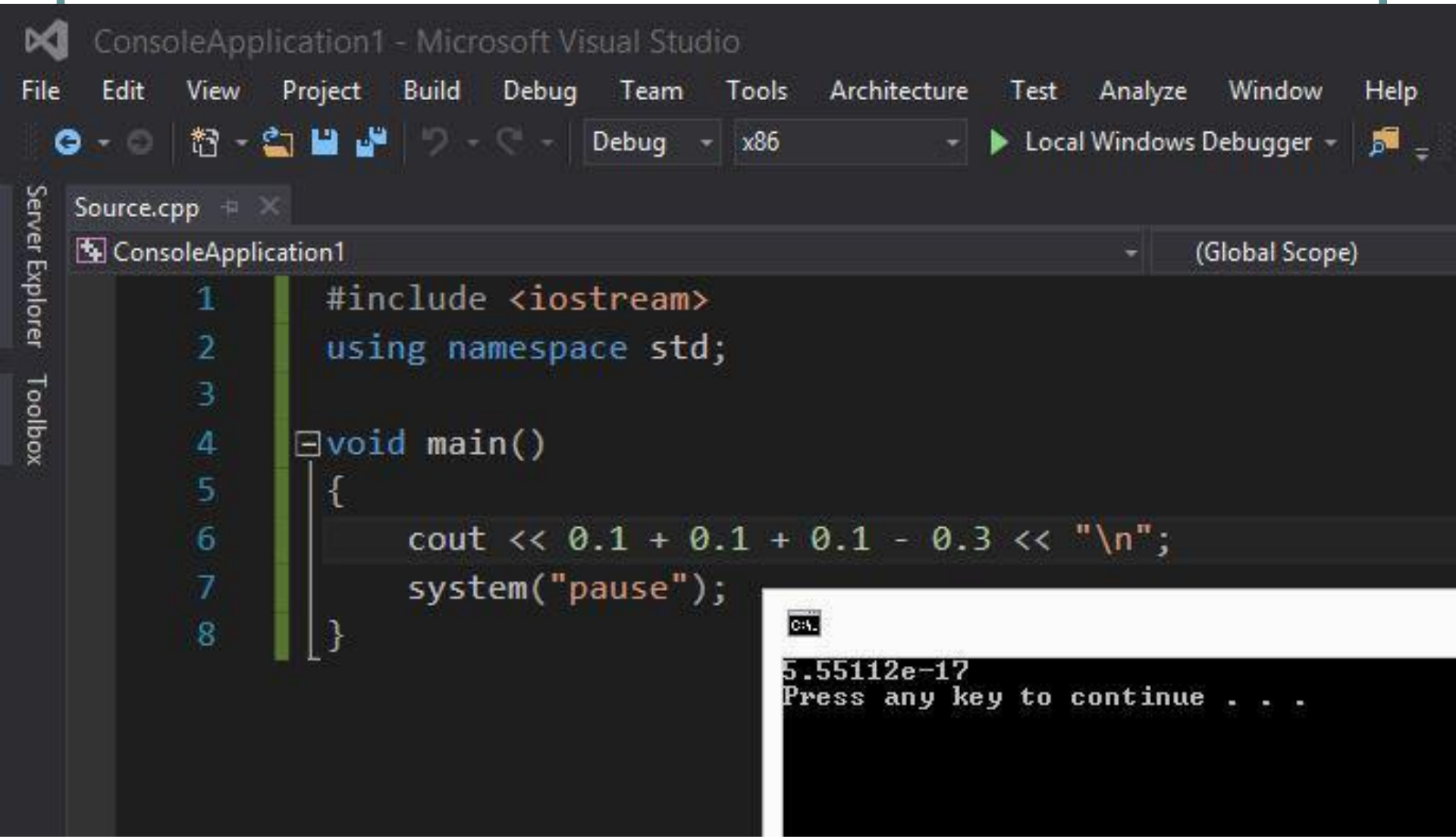
```
int temperature;
```

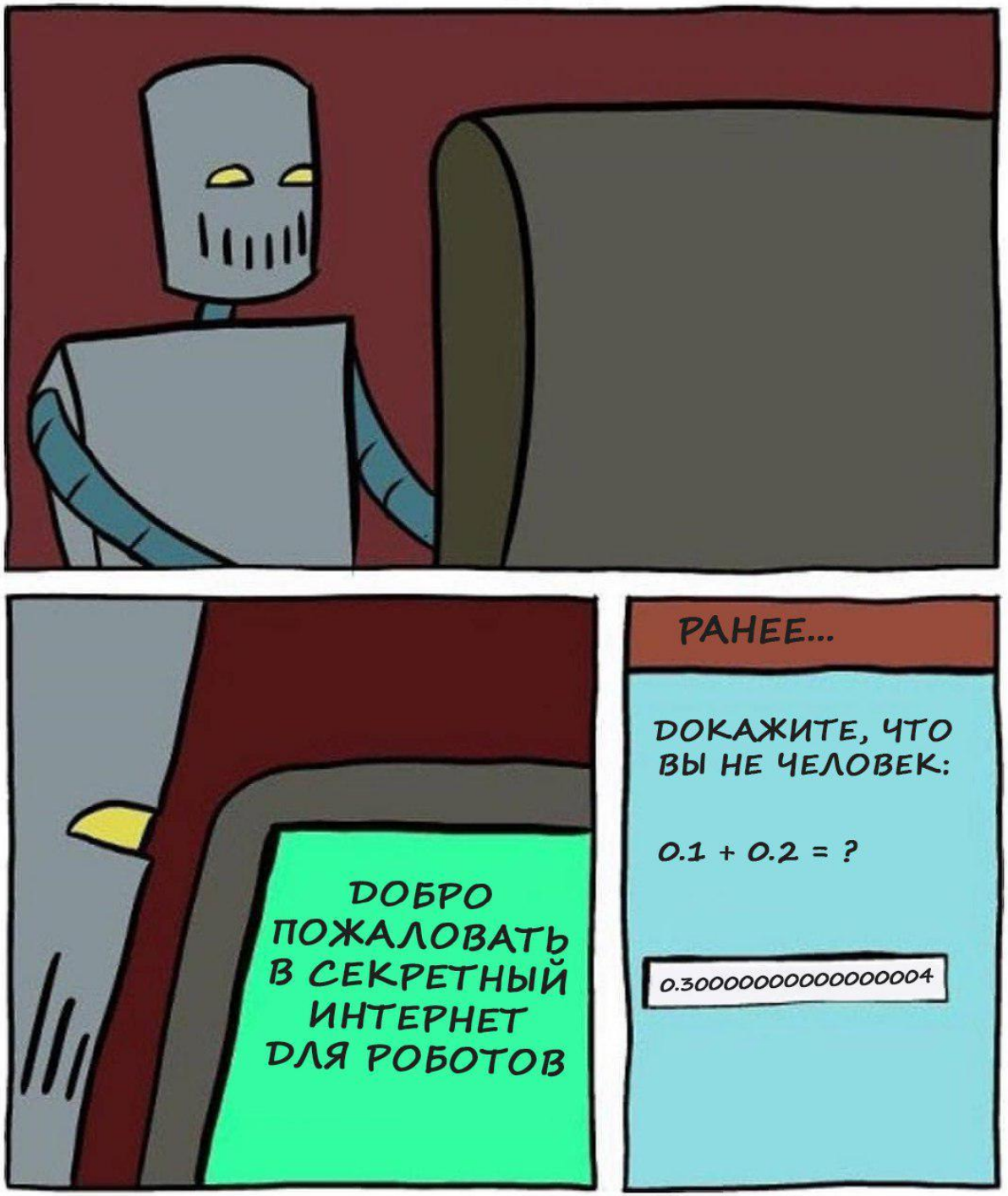
Как душ работает на самом деле

```
bool temperature;
```

По стандарту IEEE 754 представление действительных чисел должно записываться в экспоненциальном виде. Это значит, что часть битов кодирует собой **мантиссу** числа, другая часть — показатель **порядка** (степени), и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное).

https://ru.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE_%D0%B4%D0%B2%D0%BE%D0%B9%D0%BD%D0%BE%D0%B9_%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D1%81%D1%82%D0%B8


ConsoleApplication1 - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Architecture Test Analyze Window Help
Debug
x86
Local Windows Debugger
Server Explorer
Toolbox
Source.cpp
ConsoleApplication1
(Global Scope)
#include <iostream>
using namespace std;
void main()
{
cout << 0.1 + 0.1 + 0.1 - 0.3 << "\n";
system("pause");
}
5.55112e-17
Press any key to continue . . .


ДОБРО ПОЖАЛОВАТЬ В СЕКРЕТНЫЙ ИНТЕРНЕТ ДЛЯ РОБОТОВ
РАНЕЕ...
ДОКАЖИТЕ, ЧТО ВЫ НЕ ЧЕЛОВЕК:
0.1 + 0.2 = ?
0.30000000000000004

- **Размер блока памяти**, выделяемый для хранения данных
- **Структуру** этого блока памяти (как в машине будет сохранено, и как машина будет воспринимать данное значение - наличие или отсутствие знакового бита для целых чисел; наличие или отсутствие в числе битов для мантиссы, порядка и знака дробного числа
- **Диапазон значений**
- **Набор операторов для работы** с этими значениями (например, для строк нельзя использовать оператор «минус», а для дробных чисел нельзя использовать битовые операции)

## Целочисленные литералы

Например:
126

По умолчанию имеют тип данных **int**

Чтобы задать явно тип данных **long** можно использовать суфиксы **l** или **L**

Например:
126**l**
126**L**

## Вещественные (дробные) литералы

Например:
2.14

По умолчанию имеют тип данных **double**

Чтобы задать явно тип данных **float** можно использовать суфиксы **f** или **F**

Например:
2.14**f**
2.14**F**

## Символьные литералы

Например:
'a'

По умолчанию имеют тип данных **char**

## Логические литералы

Например:
**true**

По умолчанию имеют тип данных **boolean**

тип идентификатор;

```
int age;
float price;
short cats, dogs;
char answer, symbol;
bool is_hungry;
```

https://stackoverflow.com/questions/15320756/what-is-the-range-of-the-long-double-c

тип идентификатор = инициализатор;

```
int age = 32;
float price = 500.15f;
short cats = 3, dogs = 1;
char answer = ‘b’, symbol = ‘x’;
bool is_hungry = false;
```

Эти модификаторы изменяют формат представления данных, но не влияют на размеры выделяемых областей памяти.

- Модификатор **signed** указывает, что переменная может принимать как положительные, так и отрицательные значения. Возможно, что при этом самый левый бит области памяти, выделяемой для хранения значения, используется для представления знака. Если этот бит установлен в 0, то значение переменной считается положительным. Если бит установлен в 1, то значение переменной считается отрицательным.
- Модификатор **unsigned** указывает, что переменная принимает неотрицательные значения. При этом самый левый бит области памяти, выделяемой для хранения значения, используется так же, как и все остальные биты области памяти - для представления значения.

```
// компилятор выводит тип из типа значения инициализации
auto x = 100;      // x получает тип int
auto y = 1.5;      // y получает тип double
auto z = 1.3e12L; // z получает тип long double

// попытка создать три переменные типа double
auto a = 0.0; // нормально, a будет с типом double
double b = 0; // нормально, 0 автоматически преобразуется в 0.0
auto c = 0;   // проблема, c получит тип int!
```

**Константа** – это именованная область в оперативной памяти, которая может хранить, НО НЕ МОЖЕТ изменять своё значение на протяжении работы программы. Чтобы объявить константу, добавьте к синтаксису переменной модификатор const. Также под понятие константы можно подвести понятие литерала.

```
const int code = 24;
int x = 24;             // старая-добрая, привычная форма инициализации
int y {24};             // НОВАЯ форма с использованием фигурных скобок
char c1 {1989};         // списковая инициализация не допускает сужение!
char c2 = {24};         // разрешено, char может хранить значение 24
char c3 {code};         // то же самое
char c4 = {x};          // не разрешено, x не является константой!
x = 1989;
char c5 = x;            // разрешено привычной формой инициализации

char M = 1.77e72;       // поведение не определено...
char D = 934737625;     // поведение не определено...
char N = {1.77e72};     // ошибка времени компиляции!
char A = {934737625};   // выход за пределы диапазона!
```

**x = 3;**

**y = x;**

**z = x;**

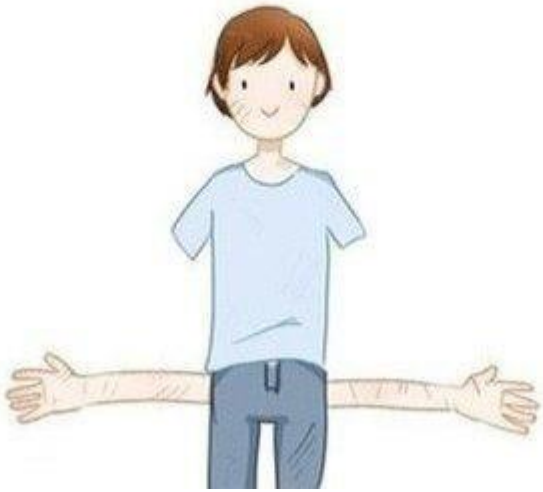
многократное использование операции присваивания в одном выражении:

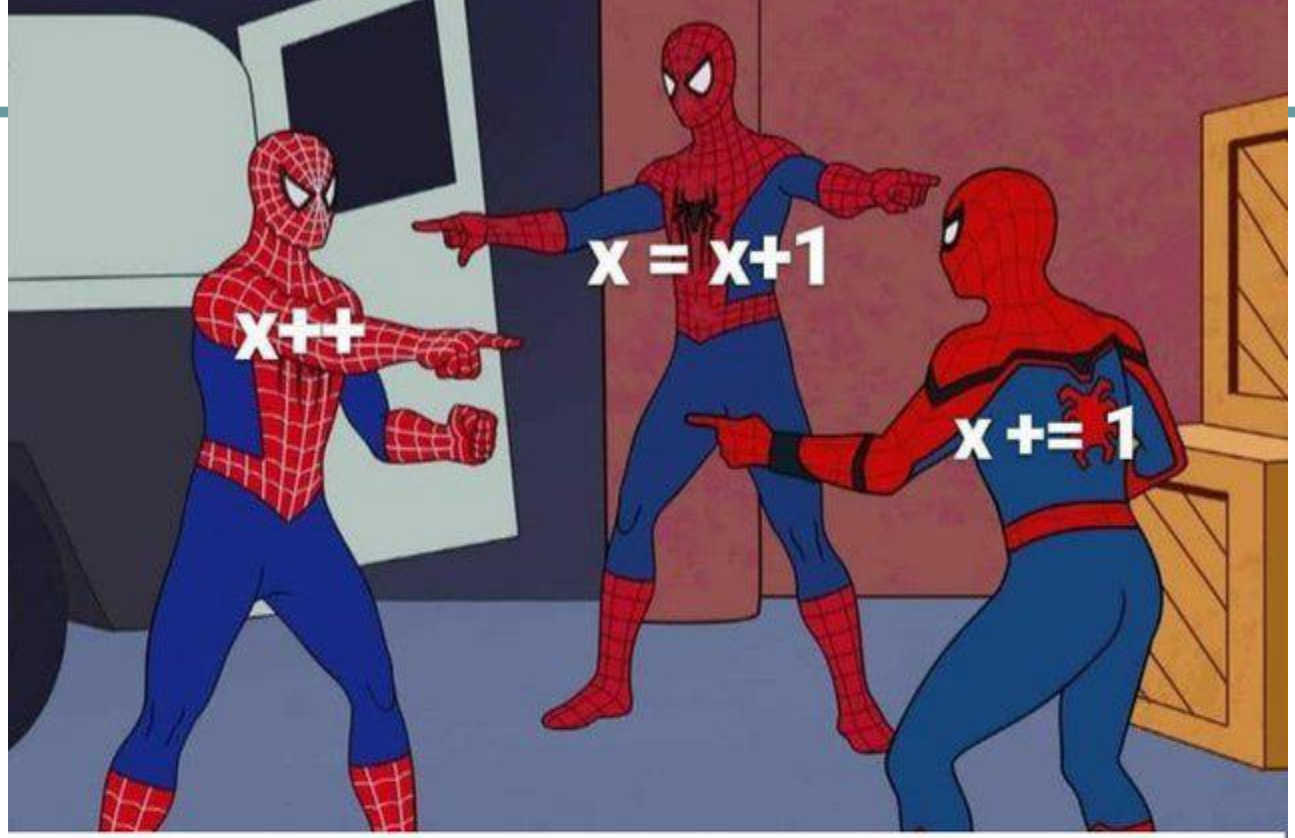
**x = y = z = 0;**

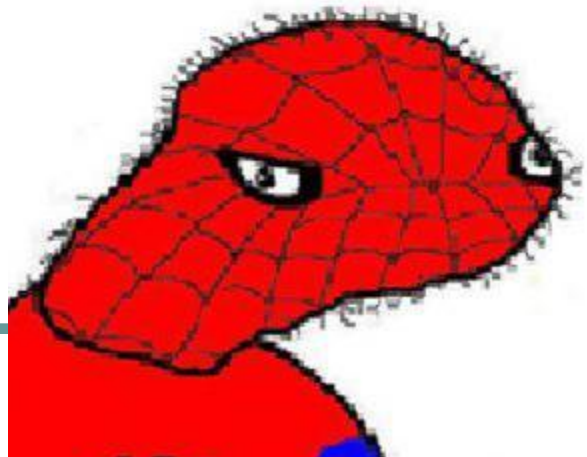
Не рекомендуется так делать!

- **инкремент** "**++**"– увеличивает значение переменной на 1
- **декремент** "**--**" – уменьшает значение переменной на 1

Для этих операторов существует префиксная и постфиксная форма (практика).

| i++ | i=i+1 | i+=1 |
| --- | --- | --- |
|  |  |  |



x-=-1

\+ сложение

\- вычитание

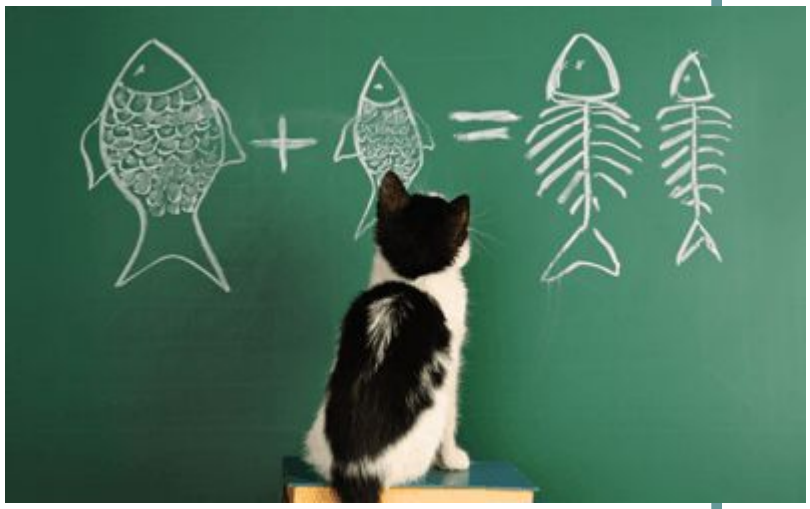
* умножение

/ деление

% остаток от деления

++ инкремент

-- декремент



```
int a = 5 + 5;
a = b * c;
b++;
a = 10 % 2;
```

**Единственность цели каждой переменной**

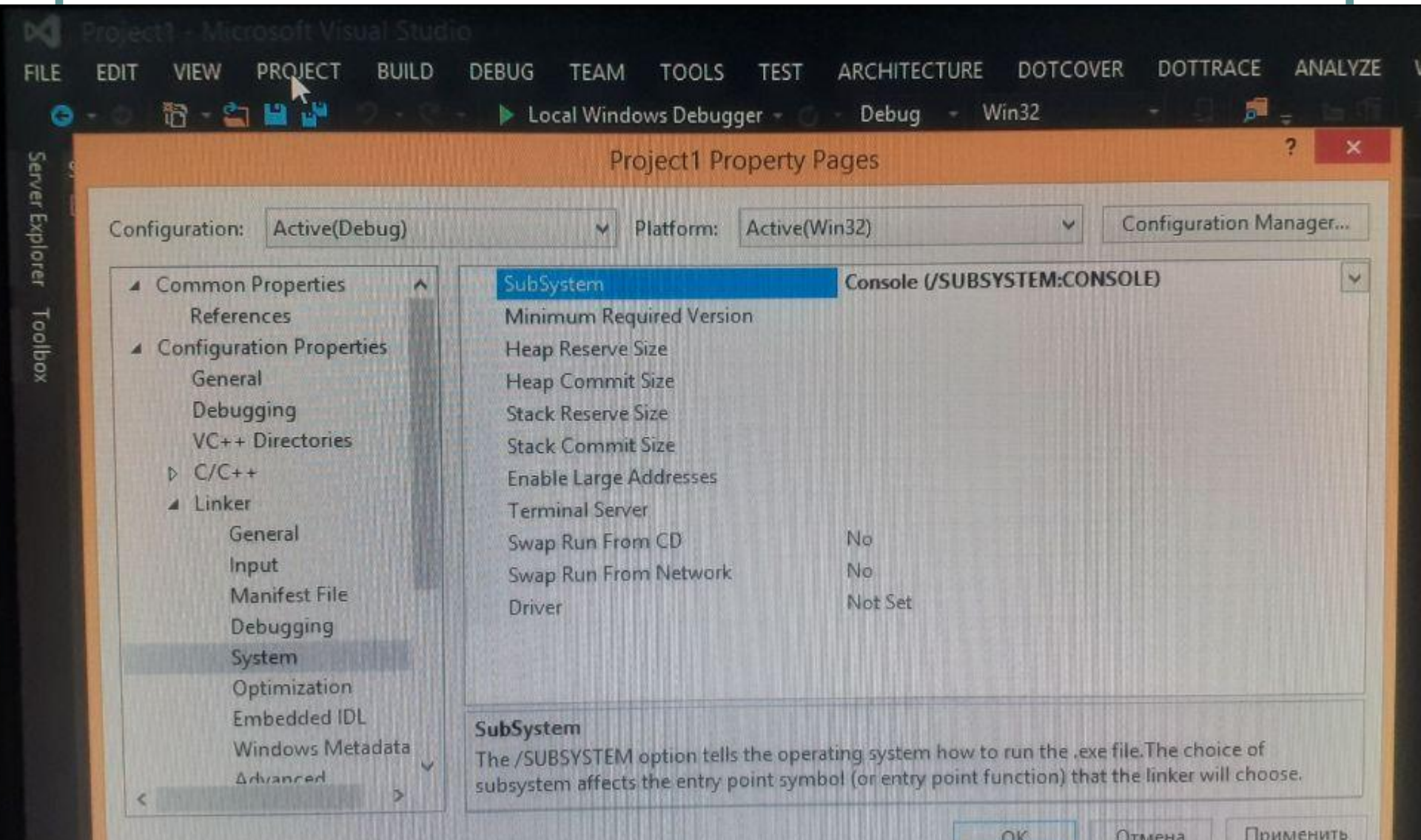
Используйте переменную только с одной целью. Иногда есть соблазн вызвать одну переменную в двух разных местах для решения двух разных задач. Использование "временной" переменной - очень плохая затея.

Из-за использования в нескольких разных ситуациях одной переменной создается впечатление, что задачи связаны, хотя на самом деле это не так. Создавайте уникальные переменные для каждой цели, чтобы сделать код более читабельным и понятным.

У моей соседки **K** кошек. Каждая кошка за день съедает **M** граммов кошачьего корма. Килограмм корма стоит **G** гривен.

Сколько денег уходит на кошачий корм в месяц, и за целый год?

```
//system("cls"); // очистка экрана
//system("color 9F"); cout<<"text"; // применение цвета ко всему окну
//system("title заголовок"); // новый заголовок окна
//system("echo привет"); // аналог cout
//system("echo hello > D:\\1.txt"); // создать файл
//system("cmd"); // запуск командной строки

//system("date /t"); // показ даты на экран
//system("date"); // изменение даты
//system("time"); // показ точного времени и изменение его
//system("time \t"); // показ часов и минут

//system("C:\\2.txt"); // открытие обычного или exe файла
//system("start http://google.com"); // открытые интернет-сайта
//system("type \"C:\\2.txt\""); // показ текста из файла на экран консоли
//system("copy C:\\1.txt C:\\2.txt"); // копирование файлов, 1-что копировать, 2-куда копировать
//system("move C:\\1.txt F:\\2.txt"); // перемещение или переименование файла
//system("md C:\\folder"); // создание папки
//system("rd C:\\folder"); // удалить папку
//system("tree F:"); // графическое отображение дерева каталогов
//system("del C:\\1.txt /f"); // принудительное удаление файлов и папок

//system("label C: X-Files"); // метка диска
//system("systeminfo"); // информация о системе
//system("format F: /q"); // быстрое форматирование диска

//system("taskkill /im notepad.exe"); // завершение всех приложений с указанным именем
//system("taskkill /s alexnote /fi \"username ne nt*\" /im * /f"); // мгновенное завершение всех приложений на удалённом компе

//system("shutdown -s"); // выключение компа
//system("shutdown -a"); // отмена выключения компа
//system("shutdown -l"); // завершение сеанса
//system("shutdown -i"); // окно настроек отключения
//system("shutdown -r"); // перезагрузка компа
//system("shutdown -g"); // перезагрузка компа с перезапуском всех работающих программ
//system("shutdown -p"); // немедленное отключение локального компа, однако виста может предотвратить это, с глюками
//system("shutdown -h"); // гибернация
//system("shutdown -g -m alexnote -t 0 -f"); // мгновенное принудительное выключение удалённого компа с перезапуском программ

//system("help"); // показ справки по всем командам
//system("help shutdown"); // справка по указанной команде
```


Project1 - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE DOTCOVER DOTTRACE ANALYZE
Local Windows Debugger
Debug
Win32
Server Explorer
Toolbox
Project1 Property Pages
Configuration: Active(Debug)
Platform: Active(Win32)
Configuration Manager...
Common Properties
References
Configuration Properties
General
Debugging
VC++ Directories
C/C++
Linker
General
Input
Manifest File
Debugging
System
Optimization
Embedded IDL
Windows Metadata
SubSystem
Console (/SUBSYSTEM:CONSOLE)
Minimum Required Version
Heap Reserve Size
Heap Commit Size
Stack Reserve Size
Stack Commit Size
Enable Large Addresses
Terminal Server
Swap Run From CD
No
Swap Run From Network
No
Driver
Not Set
SubSystem
The /SUBSYSTEM option tells the operating system how to run the .exe file.The choice of subsystem affects the entry point symbol (or entry point function) that the linker will choose.

```
system("pause >> NUL");
```