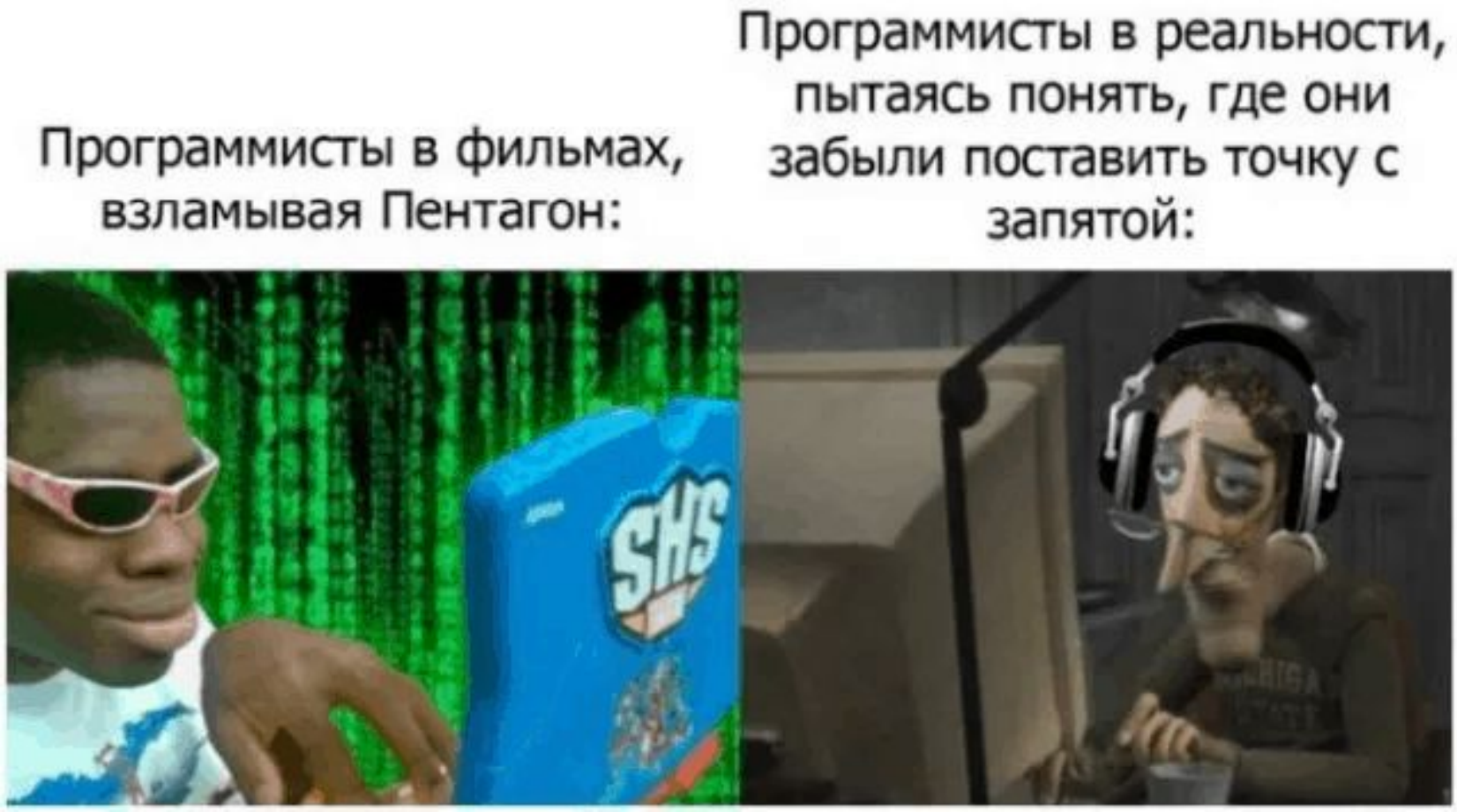
Основы алгоритмизации и программирования

Пашук Александр Владимирович

pashuk@bsuir.by

Мем в начале


Программисты в фильмах, взламывая Пентагон:
Программисты в реальности, пытаясь понять, где они забыли поставить точку с запятой:
SHS

Фидбэк

Почему важно?

- Не тратить время на бесполезные вещи ☺️
- Сделать процесс обучения интереснее

Где оставить?

- Почта: pashuk@bsuir.by
- Портал студентов: https://students.bsuir.by/

(к каждой паре можно оставить анонимный комментарий)

Содержание лекции

1. Программирование
2. Языки программирования
3. Компиляторы/Интерпретаторы
4. Методологии программирования

Программирование

Программирование — процесс создания компьютерных программ.

В узком смысле (так называемое **кодирование**) под программированием понимается написание инструкций (программ) на конкретном языке программирования.

В более широком смысле под программированием понимают весь спектр деятельности, связанный с созданием и поддержанием в рабочем состоянии программного обеспечения ЭВМ.

Программирование

Программа – это логически упорядоченная последовательность команд необходимая для управления компьютером.

Программа, с которой работает процессор, представляет собой последовательность чисел, называемую машинным кодом.

Написать программу в машинном коде достаточно сложно и поэтому для представления алгоритма в виде, понятном компьютеру, служат языки программирования.

Языки программирования

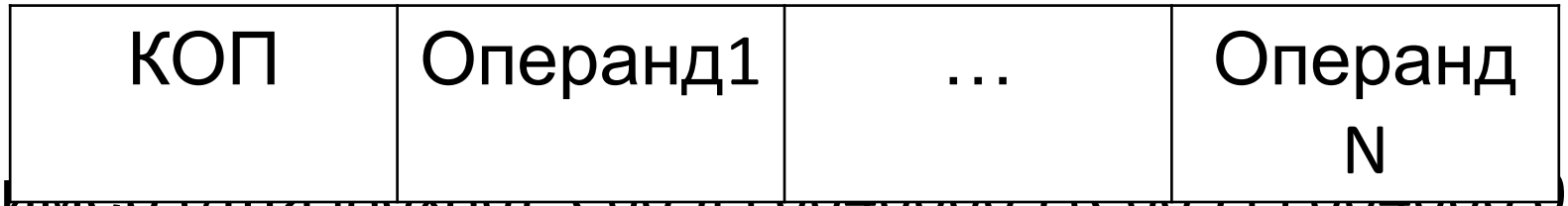
Языки программирования – формальная знаковая система, предназначенная для записи компьютерных программ.

Определяет набор лексических, синтаксических и семантических правил, задающих внешний вид программы и действия, которые выполнит исполнитель (компьютер) под ее управлением.

Машинный код

Первый «язык программирования» - машинный код (native code).

Формат команды:



Пример (для i80x86): C605EF004000 (C605 EF004000 05)

- C605 – opcode для "mov mem, data"
- Операнд1 – адрес [004000EF]
- Операнд2 – константа [05]

BB 11 01 B9 0D 00 B4 0E 8A 07 43 CD 10 E2 F9 CD 20 **48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21**

Языки программирования

- Совокупность требований для записи команд образуют **синтаксис** языка, а смысл каждой команды – **семантику** языка.
- Отличаются от естественных ограниченным, достаточно малым числом слов, значение которых понятно компьютеру (транслятору), и очень строгими правилами записи команд (операторов).
- Процесс поиска ошибок в программе называют **тестированием**, процесс устранения ошибок – **отладкой** программы.

Из чего состоит?

- **Алфавит** – фиксированный для данного языка набор основных символов, допускаемых для составления текста программы на этом языке.
- **Синтаксис** – система правил, определяющих допустимые конструкции языка программирования.
- **Семантика** – система правил однозначного толкования отдельных языковых конструкций, позволяющих воспроизвести процесс обработки данных.

Алфавит ЯП

Алфавит языка программирования состоит из **букв**, **цифр** и **лексем**.

Лексема - это наименьшая единица языка имеющая самостоятельный смысл. К лексемам относятся:

- спецсимволы (==, !=, ++, =, >, < и т.д.);
- ключевые слова (main, switch, if, while и т.д.).

Основные элементы ЯП

- Типы данных, переменные, константы
- Операторы
- Выражения
- Подпрограммы
- Библиотеки

Типы данных

Тип данных – множество значений и множество операций над этими значениями.

Определение множеств значений и операций зависит от языка программирования и его конкретной реализации.

Примеры:

- тип int в C++ – это конечное множество целочисленных значений (4 байта, -2147483648 to 2147483647)

- тип "массив" – индексированная совокупность элементов других (ранее определенных) типов с операцией индексации.

Переменные, константы

Значение – интуитивно понятный термин.

Литерал – последовательность символов, которая в программе непосредственно задает значение (например: 201, 2.01, '0', "Строка", TRUE).

Представление – строка битов для указанного значения (например: представление целого числа 201 есть 11001001).

Переменная – имя ячеек, содержащих представление значения указанного типа, которое может изменяться во время выполнения программы.

Константа – имя ячеек, содержащих представление значения указанного типа, которое не может изменяться во время выполнения программы.

Операторы

Операторы – конструкции языка для управления вычислениями.

- **Оператор присваивания** изменяет значение указанной переменной на указанное новое значение.

```
m[a*i] = r[k+2] * a;
```

- **Оператор вызова подпрограммы** активирует подпрограмму, передавая ей указанные параметры.
- **Управляющие операторы** используются для изменения порядка выполнения вычислений.

Выражения

Выражение – правило для вычисления значения. Значение выражения имеет один и только один тип.

Выражение состоит из операндов (переменные, константы и др.) и операций (отрицание, умножение и др.).

Примеры выражений:

- `-X`
- `A+B`
- `Z/3.14 + 1E-3`
- `5<2`

Подпрограммы/Библиотеки

Подпрограмма – сегмент программы, состоящий из объявлений данных и операторов, которые можно многократно вызывать из различных частей программы.

Библиотека – вспомогательная неисполняемая программная единица, содержащая определения подпрограмм и данных, которые могут быть вызваны из основной программы.

Языки программирования

Для представления алгоритма в виде, понятном компьютеру, служат языки программирования. Алгоритм действий, записывается на одном из таких языков, в итоге получается **текст программы** – полное, законченное и детальное описание алгоритма на языке программирования.

Затем этот текст программы специальными служебными приложениями либо переводится в машинный код, либо исполняется.

Языки программирования

Система программирования — это система для разработки новых программ на конкретном языке программирования. В нее входят:

- транслятор (компилятор или интерпретатор)
- интегрированная среда разработки
- средство создания и редактирования текста программы
- обширные библиотеки стандартных программ и функций
- отладочные программы
- справочная система

ЯП: Проблема

- Код программы пишется на одном или нескольких языках программирования.

- В то же время, все команды и данные процессор получает в виде электрических сигналов, которые можно представить как совокупность 0 и 1.

- Программа, с которой работает процессор, представляет собой последовательность двоичных чисел, которые называются машинным кодом.

ЯП: Транслятор

- **Транслятор** (англ. translator — переводчик) — это программа-переводчик. Она преобразует программу, написанную на одном из языков высокого уровня, в программу, состоящую из машинных команд.

- **Трансляция программы** – преобразование программы, представленной на одном из языков программирования, в машинный код.

ЯП: Компиляторы и интерпретаторы

- **Компилятор** (англ. compiler — составитель, собиратель) читает всю программу целиком, делает ее перевод и **создает** законченный вариант программы на машинном языке, который затем и выполняется.

- **Интерпретатор** (англ. interpreter — истолкователь, устный переводчик) переводит и **выполняет** программу строка за строкой.

ЯП: Компиляторы и интерпретаторы

Достоинства компиляторов:

- Программа, обработанная компилятором, получается компактной и эффективной, работает быстрее программы, выполняемой с помощью интерпретатора.
- При компиляции исходная программа и компилятор больше не нужны.

Недостаток: сложность внесения изменений в программу.

ЯП: Компиляторы и интерпретаторы

Преимущество интерпретатора заключается в том, что программа имеет одно представление – в виде текста. Их проще исправлять и изменять.

Недостаток: интерпретатор для каждой строки выполняет следующие проверки:

- Какой оператор нужно выполнить?
- Это правильный оператор?
- Есть ли у него нужное количество операндов?
- Корректные ли значения операндов

- И **компилятор**, и **интерпретатор** выполняют одну и ту же работу — преобразовывают язык программирования высокого уровня в машинный код. Однако компилятор преобразовывает исходный материал в машинный код **перед запуском программы**. Интерпретатор выполняет эту функцию **при ее запуске**.

Языки программирования

Если язык программирования ориентирован на конкретный тип процессора и учитывает его особенности, то он называется **языком программирования низкого уровня**.

Языком самого низкого уровня является язык Ассемблера, который представляет каждую машинную команду в виде символьных условных обозначений.

Ассемблер

Язык ассемблера представляет собой язык машинных команд, в котором вместо численных кодов используются их символьные синонимы, более понятные программисту. Примеры команд:

- MOV – пересылка данных
- ADD – сложение
- SUB – вычитание
- JMP – переход

Ассемблер – программа, транслирующая синонимы в машинные команды.

Ассемблер

Пример кода:

```
mov AX,1
mov BX,1
@2:
cmp BX,N
jg @1
imul BX
inc BX
jmp @2
@1:
```

Языки программирования

Достоинства:

- Эффективность
- Компактность
- Доступ ко всем возможностям процессора

Недостатки:

- Сложность, высокий порог входа
- Программы, созданные на таком языке, неприменимы для процессоров других типов

Область применения

Низкоуровневое системное программирование

- Разработка драйверов устройств
- Разработка библиотек системных функций
- Разработка модулей операционных систем
- Разработка модулей систем управления базами данных

Языки программирования

Высокоуровневый язык программирования – язык программирования, разработанный для быстроты и удобства использования программистом.

Примеры: C, C++, Java, Python, PHP, Perl, Delphi, Lisp и др.

Пример кода

```
#include <iostream>
using namespace std;

int main() {
    int firstNumber, secondNumber, sumOfTwoNumbers;

    cout << "Enter two integers: ";
    cin >> firstNumber >> secondNumber;

    sumOfTwoNumbers = firstNumber + secondNumber;

    cout << firstNumber << " + " <<  secondNumber << " = " <<
sumOfTwoNumbers;

    return 0;
}
```

Языки программирования

Достоинства:

- Близость к естественному языку
- Высокая скорость разработки ПО
- Большой выбор средств разработки
- Возможность использования программ на процессорах разных типов

Языки программирования

Недостатки:

- Недостаточная компактность и сравнительно меньшая скорость выполнения программ
- Требуется трансляция или компиляция в язык низкого уровня
- Невозможность непосредственного использования аппаратных ресурсов процессора

Область применения

Языки программирования высокого уровня используются для разработки прикладных программ.

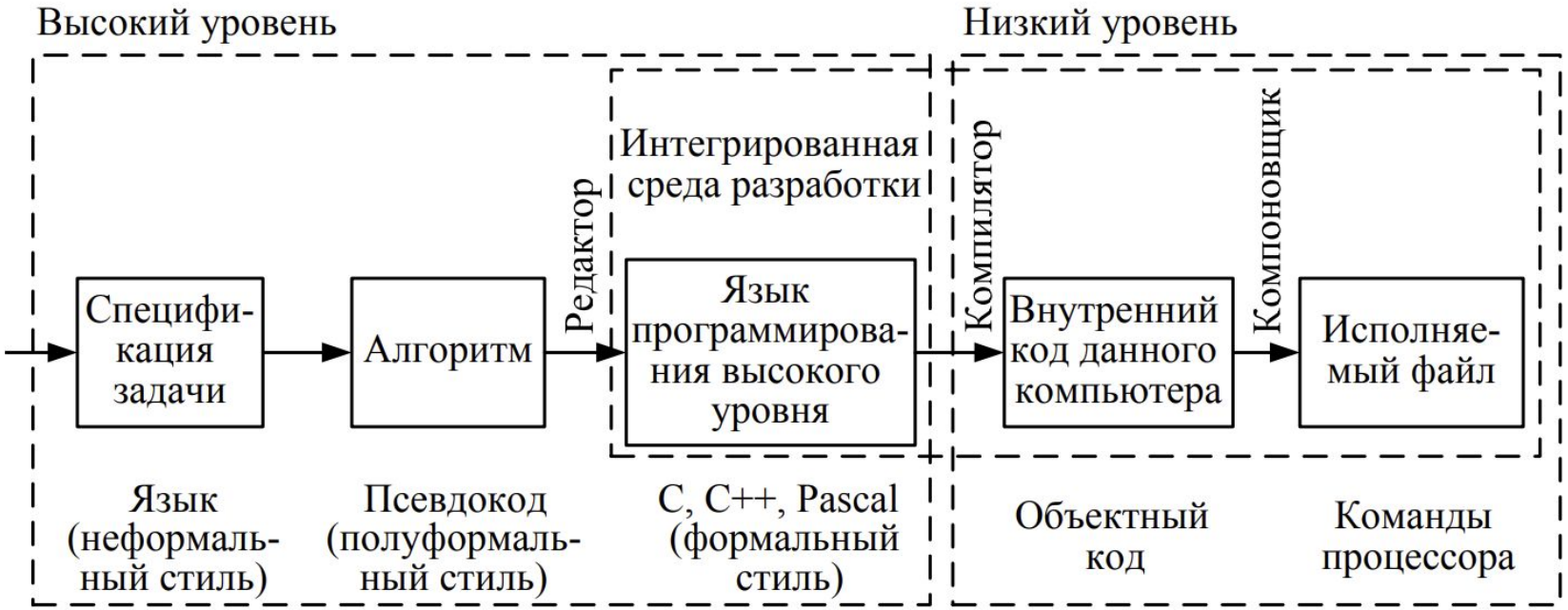
Выбор языка программирования должен обосновываться требованиями решаемой задачи.

Языки программирования

```
double a = st->value[--st->Count];
```

```
mov ecx, DWORD PTR _st$[ebp]
mov edx, DWORD PTR [ecx]
sub edx, 1
mov DWORD PTR tv71[ebp], edx
mov eax, DWORD PTR _st$[ebp]
mov ecx, DWORD PTR tv71[ebp]
mov DWORD PTR [eax], ecx
mov edx, DWORD PTR _st$[ebp]
mov eax, DWORD PTR [edx+4]
mov ecx, DWORD PTR tv71[ebp]
movsd   xmm0, QWORD PTR [eax+ecx*8
movsd   QWORD PTR _a$[ebp], xmm0
```

Этапы подготовки программы



Методологии программирования

Методология (парадигмы) программирования – это совокупность идей, понятий, принципов, способов и средств, определяющая стиль написания, отладки и сопровождения программ.

Основные методологии программирования:

– структурное (директивное) программирование;

– декларативное программирование;

– объектно-ориентированное

Структурное программирование

Основные положения:

1. Сложная задача разбивается на более мелкие.
2. Используются комбинации трех базовых структур: следования, ветвления и цикла.
3. Алгоритмы изображаются в виде блок-схем.
4. Исключается использование безусловного перехода.
5. Применяется нисходящее проектирование программ.

Структурное программирование

При нисходящем проектировании исходная, подлежащая решению задача разбивается на ряд подзадач, подчиненных по своему содержанию главной задаче. Такое разбиение называется **детализацией** или **декомпозицией**.

Модуль – это последовательность логически связанных операций, оформленных как отдельная часть программы. Модули связаны между собой только по входным и выходным данным.

Модули

Все структуры данных и подпрограммы, составляющие разрабатываемую программу, объединяются в модули.

Каждый модуль состоит из:

- открытая часть модуля (interface) – видна другим модулям;
- закрытая часть модуля (implementation) – видна только внутри модуля.

Почему модули?

1. Возможность создания программы несколькими программистами
2. Простота проектирования и последующих модификаций программы
3. Упрощение отладки программы – поиска и устранения в ней ошибок
4. Возможность использования готовых библиотек наиболее употребительных модулей

Декларативное программирование

При декларативном программировании разработчик фокусируется на том, что необходимо сделать, а не на деталях, как это сделать.

Иногда выделяют логическое (LISP) и функциональное (Prolog) программирование, в зависимости от основной единицы кода.

Примеры языков: HTML, CSS, SQL.

Логическое программирование

Pascal/Delphi

```
Program CalcFactorial;
function Fact(N: Integer) : Integer;
var i, F: Integer;
begin
    F := 1;
    for i := 1 to N do
        F := F * i;
    Result := F;
end;
var N: Integer;
begin
    Write('Введите число: '); Read(N);
    Write(N, '! = ', Fact(N));
end.
```

PROLOG

```
/* Описание фактов */
Fact(0, 1).

/* Описание правил */
Fact(N, F) :-
    N > 0,
    prevN is N-1,
    N > 1,
    Fact (prevN, prevF),
    F is N*prevF.

/* Описание запроса */
?- Fact (6, N)
```

Функциональное программирование

Pascal/Delphi

```
Program CalcFactorial;
function Fact(N: Integer) : Integer;
var i, F: Integer;
begin
    F := 1;
    for i := 1 to N do
        F := F * i;
    Result := F;
end;
var N: Integer;
begin
    Write('Введите число: '); Read(N);
    Write(N, '! = ', Fact(N));
end.
```

LISP

```
/* Описание фактов и правил как функции */
(defun fact (N)
    (if (zero N)
        1
            (* N ( fact (1- N) ) )
    )
)

/* Описание запроса */
(format t "6! = ~A~%" (fact 6))
```

ООП

Объектно-ориентированное программирование является логичным продолжением идей структурного программирования.

Объектно-ориентированный подход основан на:

- выделении классов объектов;
- установлении свойств и методов обработки;
- создании иерархии классов, наследовании свойств объектов и методов их обработки.

Популярность языков

Sep 2020	Sep 2019	Change	Programming Language	Ratings	Change
1	2	^	C	15.95%	+0.74%
2	1	v	Java	13.48%	-3.18%
3	3		Python	10.47%	+0.59%
4	4		C++	7.11%	+1.48%
5	5		C#	4.58%	+1.18%
6	6		Visual Basic	4.12%	+0.83%
7	7		JavaScript	2.54%	+0.41%
8	9	^	PHP	2.49%	+0.62%
9	19	^^	R	2.37%	+1.33%
10	8	v	SQL	1.76%	-0.19%

Самый сложный ЯП

Одним из самых сложных языков программирования считается Brainfuck.

Придуман Урбаном Миллером в 1993 году в попытке создать язык с как можно меньшим компилятором.

Алфавит языка: *< > + - . , [ ]*

Самый сложный ЯП

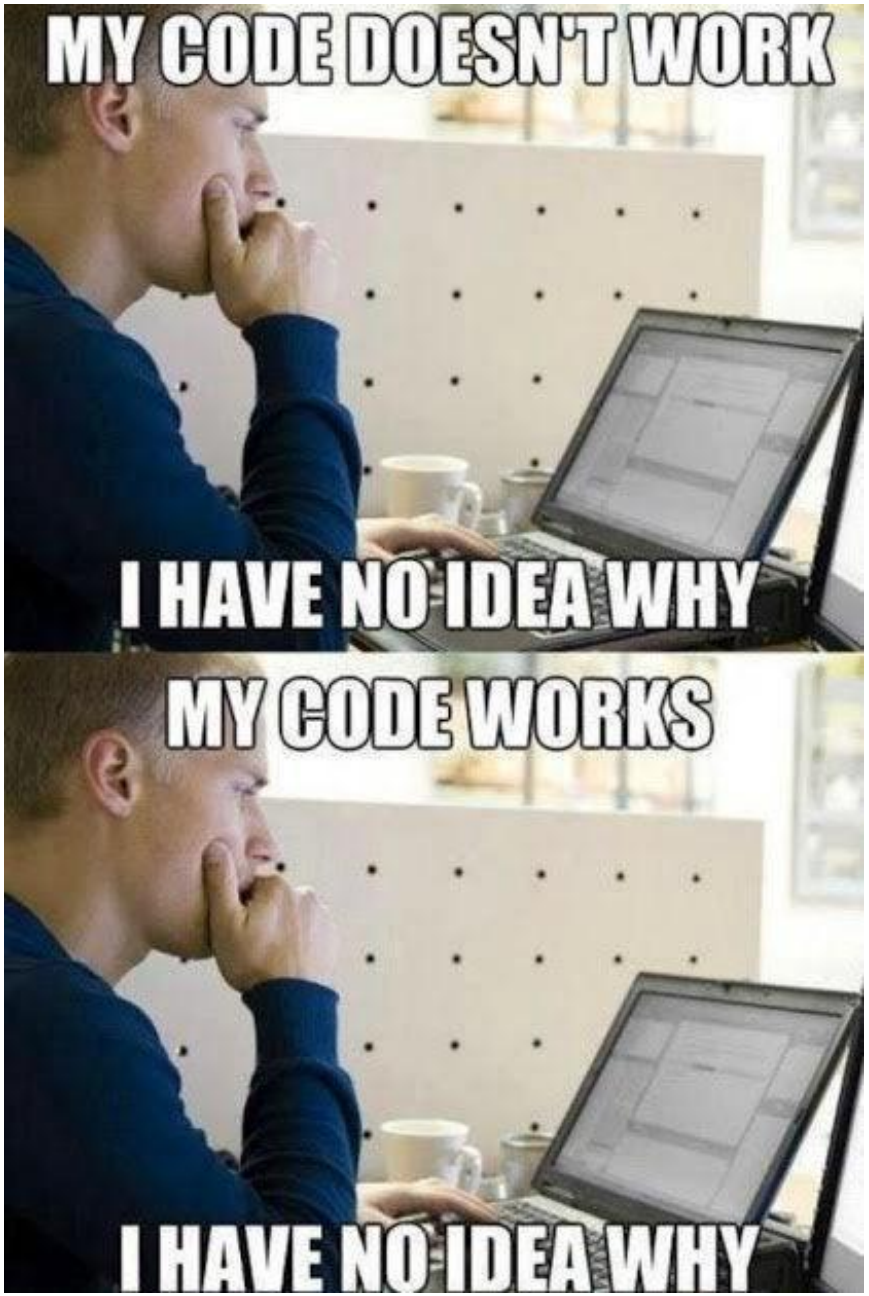
Стандартный «Hello World!» пример:

```
++++++++++[>+++++++>++++++++++>+++>+<<<<-]>++
.>+.+++++++..+++.>++.<<+++++++++++++++.>.+++.
  ------.--------.>+.>.
```

- *>++.* – это вывод «H»
- *>+.* – это вывод «e» и т.п.

Интерпретатор языка можно реализовать в 62 строках кода C++.

Мем в конце


MY CODE DOESN'T WORK
I HAVE NO IDEA WHY
MY CODE WORKS
I HAVE NO IDEA WHY