Есть ли у вас вопросы?

Краткое содержание предыдущей серии

- Как в ассемблере происходит сравнение?
- Как используется результат сравнения?
- Что такое «условное исполнение»?
- Как в ассемблере осуществляется ветвление?

Краткое содержание этой серии

- О магии
- Операции в языке С (продолжение)
- О-нотация

Что такое «магия»?

В широком смысле – это «что-то непонятное».

Строгой классификации не существует.
Условно:

- белая магия – результат действия полностью понятен, но не понятен механизм

- черная магия – результат действия понятен не полностью или вообще ничего непонятно

Пример «белой магии»

Функция sin. Что возвращает sin?

Синус угла.

А как она его вычисляет?

Правильно.

Если вас устраивает результат «белой магии» (точность, скорость и т.д.), то понимать ее механизм не обязательно.

Пример «черной магии»

```
#include <stdio.h>
int l;int main(int o,char **O,
int I){char c,*D=O[1];if(o>0){
for(l=0;D[l                 ];D[l
++]-=10){D   [l++]-=120;D[l]-=
110;while   (!main(0,O,l))D[l]
+=   20;   putchar((D[l]+1032)
/20   )   ;}putchar(10);}else{
c=o+     (D[I]+82)%10-(I>l/2)*
(D[I-l+I]+72)/10-9;D[I]+=I<0?0
:!(o=main(c/10,O,I-1))*((c+999
)%10-(D[I]+92)%10);}return o;}
```

Очень сложно понять, что делает эта программа и как она это делает.

Причины «магии»

- «Индуизм»
- Ручная оптимизация
- Магические числа
- Обфускация (намеренное ухудшение читаемости кода)
- Недокументированные и малоизвестные особенности чего-либо
- Соревнования волшебников

«Индуизм» («индусский код»)

Стремление писать код кривым, неочевидным, неестественным способом (жаргонное обозначение).

Не индуизм:

```
if( i > 0 && i < 100 )
```

Индуизм:

```
if( i.toString().length() == 2 )
```

«Магические числа»

Это численные константы, смысл которых не ясен.

```
void setSpeed(int speed)
{
  if(speed > 59)
    return;
  ...
}
```

```
#define SPEED_MAX 59

void setSpeed(int speed)
{
  if(speed > SPEED_MAX)
    return;
  ...
}
```

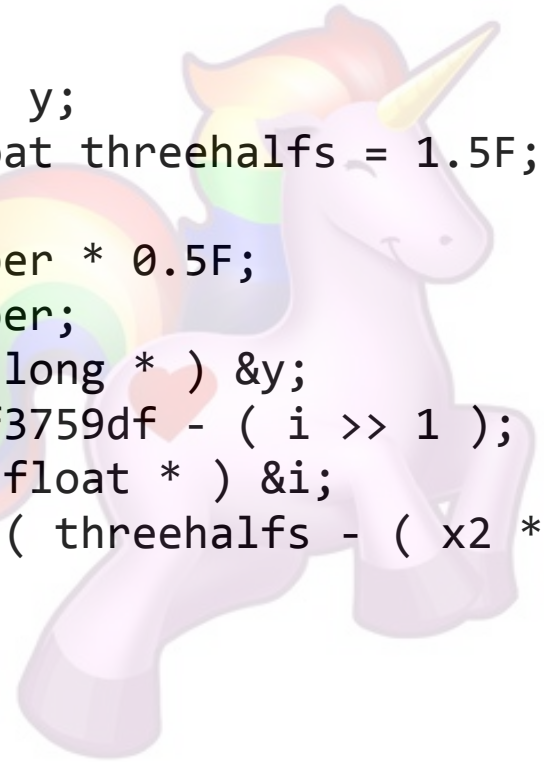
Как сделать черную магию белой?

```
// Быстрый вариант функции 1/sqrt.
// Быстр при аппаратной поддержке плавающей арифметики

float fastInverseSqrt( float number )
{
    long i;
    float x2, y;
    const float threehalfs = 1.5F;

    x2 = number * 0.5F;
    y  = number;
    i  = * ( long * ) &y;
    i  = 0x5f3759df - ( i >> 1 );
    y  = * ( float * ) &i;
    y  = y * ( threehalfs - ( x2 * y * y ) );

    return y;
}
```



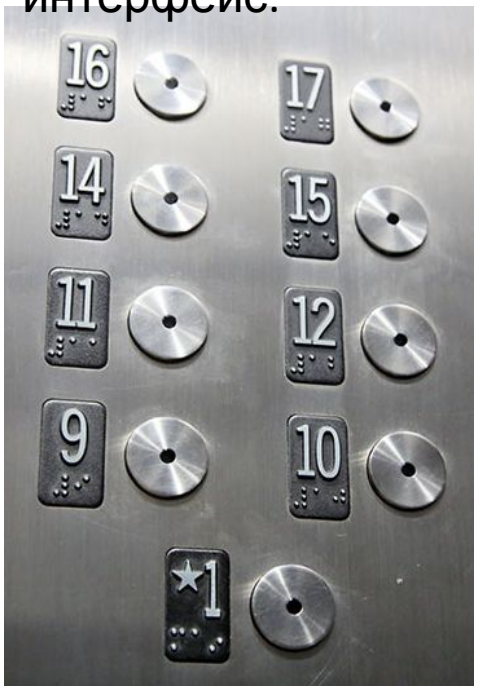
Что такое интерфейс?

Интерфейс – набор входов и выходов черного ящика; их свойства, возможные диапазоны и т.д.

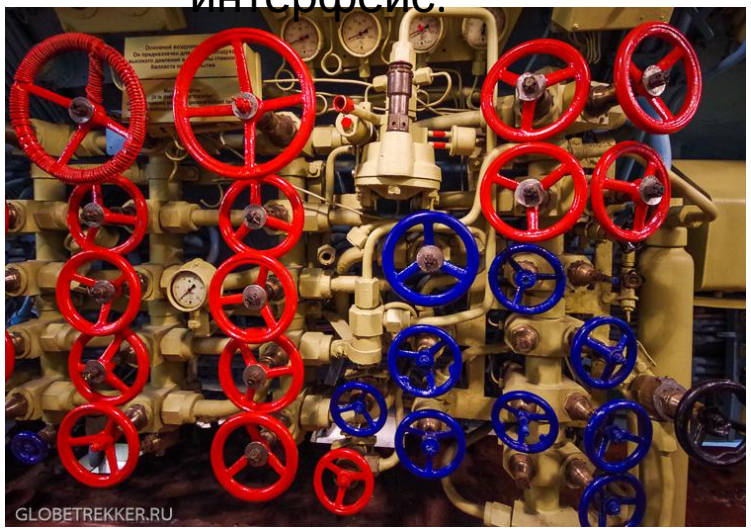
В зависимости от области интерфейс может быть разным.

Интерфейсы

Хороший интерфейс:



Плохой интерфейс:




```
double sin( double angleInRadians );
```

```
int work( int a, int b, int * c );
// зависит от трех глобальных переменных
// возвращает значение -1, если нет ошибок
// переменная c не нужна уже второй год,
```

Хороший интерфейс делает черную магию белой!

Операции в языке С (продолжение)

- Логические
- Битовые

Логические операции

- ! – логическое отрицание
- && - логическое И
- || - логическое ИЛИ

Т.к. тип bool был введен только в C99, все логические операции используют тип int.

Поэтому для них 0 – это ложь, а *любое другое число* – истина.

Логическое отрицание - !

Результат выражения !A равен нулю, если A не равно нулю и равен единице, если A равно нулю.

Выражение !A эквивалентно выражению 0==A.

Выражение	**Результат**
!6	0
!-50	0
!8.495	0
!1	0
!0	1

Логическое ИЛИ - ||

Результат выражения A || B равен нулю, только если оба аргумента равны нулю, во всех остальных случаях результат равен единице.

Выражение	**Результат**
`0 || 1`	`1`
`15 || 3`	`1`
`1.59 || 0.1`	`1`
`-18 || -3`	`1`
`0 || 0`	`0`

Логическое И - &&

Результат выражения A && B равен единице, только если оба аргумента не равны нулю, во всех остальных случаях результат равен нулю.

Выражение	**Результат**
`0  && 1`	`0`
`15 && 0`	`0`
`0 && 0.1`	`0`
`-18 && -3`	`1`
`0.1 && -80`	`1`

Логические операции в ассемблере

Их нет! Есть только битовые.

Все операции, которые называются «logical» в тех. описании, являются битовыми.

Логические операции языка C превращаются в несколько ассемблерных команд.

Битовые операции языка С

- ~ - битовая инверсия
- | - битовое ИЛИ
- & - битовое И
- ^ - битовое исключающее или (XOR)
- >> - сдвиг вправо
- << - сдвиг влево

Все битовые операции выполняются над **двоичными** представлениями чисел.

В языке С битовые операции определены **только для целых** чисел!

Битовая инверсия - ~

При битовой инверсии каждый бит двоичного представления аргумента меняется на противоположный (инвертируется).

Размер (в байтах) результата операции равен размеру аргумента, поэтому результат зависит от типа!

Примеры:

```
uint8_t A = 5; //   A = 0000 0101₂ = 5₁₀
A = ~A;        //   A = 1111 1010₂ = 250₁₀

uint16_t B = 5; // B = 0000 0000 0000 0101₂ = 5₁₀
B = ~B;         // B = 1111 1111 1111 1010₂ = 65530₁₀
```

Битовое ИЛИ - |

Результатом битового ИЛИ будет число, каждый бит которого является результатом булевой операции ИЛИ между соответствующими битами аргументов.

Коротко: если бит равен 1 в любом из аргументов – он равен 1 в результате.

N бита	7	6	5	4	3	2	1	0
a	1	0	1	0	0	1	1	0
b	0	0	0	1	0	1	0	1
a | b	1	0	1	1	0	1	1	1

A |= B эквивалентно A = A | B.

Битовое ИЛИ удобно использовать для **установки отдельных битов в единицу: a |= 1;**

Битовое И - &

Результатом битового И будет число, каждый бит которого является результатом булевой операции И между соответствующими битами аргументов.

Коротко: если бит равен 0 в любом из аргументов – он равен 0 в результате.

N бита	7	6	5	4	3	2	1	0
a	1	0	1	0	0	1	1	0
b	0	0	0	1	0	1	0	1
a & b	0	0	0	0	0	1	0	0

A &= B эквивалентно A = A & B.

Битовое И удобно использовать для **обнуления отдельных битов:**

a &= ~1;

Битовое исключающее ИЛИ (XOR)

_ ^

Если значение одного бита у аргументов разное – то результат равен 1.

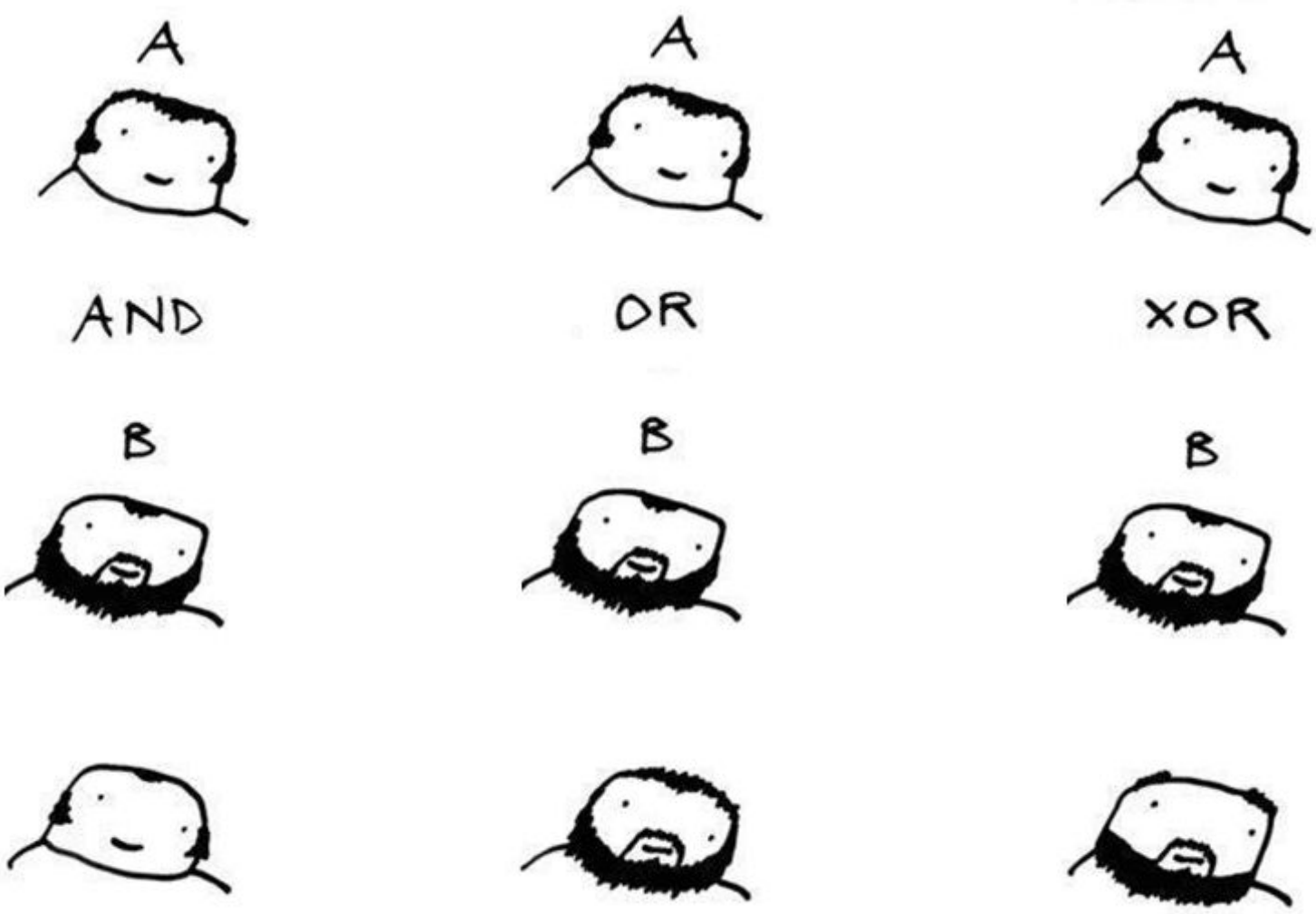
N бита	7	6	5	4	3	2	1	0
a	1	0	1	0	0	1	1	0
b	0	0	0	1	0	1	0	1
a ^ b	1	0	1	1	0	0	1	1

A ^= B эквивалентно A = A ^ B.

Битовое исключающее ИЛИ удобно использовать для **инверсии отдельных битов:**

a ^= 1;

Логические операции с волосами



Сдвиги

Сдвиги бывают:

- «просто» сдвиги – они же «логические» (без учета знака)
- арифметические (с учетом знака)
- циклические

Какие же сдвиги в языке С?
Не циклические.

Сдвиг влево - <<

- A << B эквивалентно $A \times 2^B$
- Пустые биты справа заполняются нулями
- a <<= 3 эквивалентно a = a << 3

N бита	7	6	5	4	3	2	1	0
uint8_t a = 19;	0	0	1	0	0	1	1	0
uint8_t b = a << 2;	1	0	0	1	1	0	0	0

Сюрпризы:

- Сдвиг на отрицательное число – undefined behavior
- Сдвиг отрицательного числа влево (начиная со стандартов C99 и C++11) – undefined behavior
- Если сдвигается положительное знаковое число, то результат сдвига должен помещаться в переменную-приемник, иначе – undefined behavior.

Сдвиг вправо - >>

- A >> B эквивалентно $A/2^B$
- a >>= 4 эквивалентно a = a >> 4

N бита	7	6	5	4	3	2	1	0
uint8_t a = 166;	1	0	1	0	0	1	1	0
uint8_t b = a >> 3;	0	0	0	1	0	1	0	0

Сюрпризы:

- Сдвиг вправо отрицательных чисел – implementation defined
- Сдвиг на отрицательное число – undefined behavior

Поэтому сдвиг вправо вроде бы арифметический, но вроде бы и нет.

Примеры сюрпризов

- `int a = -1 << 1;` - undefined behaviour
- `int a = 1 << 100;` - undefined behaviour
- `int a = 1 << -2;` - undefined behaviour
- `int a = 1 >> -3;` - undefined behaviour
- `int a = -1 >> 4;` - implementation defined

Сюрприз в сюрпризе

Описания сдвигов отрицательных чисел появились в стандарте слишком поздно.

Программисты успели написать достаточно кода, где такие сдвиги используются!

Зачем?

Для получения битовых масок с заданным числом нулей удобно сдвигать -1 (если он в доп. коде)

Но так делать не надо! Сдвигайте лучше `~0u`

Рассмотрим два алгоритма сортировки

```
int array[N] = ....

for (int i=0 ; i < N-1; i++)
{
    for (int j=0 ; j < N-i-1; j++)
    {
      if (array[j] > array[j+1])
      {
        int swap   = array[j];
        array[j]   = array[j+1];
        array[j+1] = swap;
      }
    }
}
```

Сортировка пузырьком

```
int array[N] = ....
int minimum;

for (int i=0; i < N-1; i++)
{
    minimum = i;

    for (int j=i+1; j < N; j++)
    {
        if (array[j] < array[minimum])
        {
            minimum = j;
        }
    }

    if(minimum != i)
    {
        int swap   = array[i];
        array[i]   = array[minimum];
        array[minimum] = swap;
    }
}
```

Сортировка выбором

Какой из них быстрее?

- А если взять другой массив?
- А если взять другой компьютер?
- А если массив будет гораздо, гораздо больше?

Как узнать, какой алгоритм быстрее?

Написать программу и запустить?

- но ее можно написать с ошибками
- на разных компьютерах скорость будет разной
- на разных исходных данных скорость будет разной
- на *разных запусках* скорость *может быть* разной!

Теоретический анализ?

- Какая самая долгая операция в алгоритме?
- Какая операция выполняется наибольшее количество раз?
- От чего зависит это количество?

Теоретический анализ?

```
int array[N] = ....

for (int i=0 ; i < N-1; i++)
{
    for (int j=0 ; j < N-i-1; j++)
    {
      if (array[j] > array[j+1])
      {
        int swap   = array[j];
        array[j]   = array[j+1];
        array[j+1] = swap;
      }
    }
}
```

Теоретический анализ?

```
int array[N] = ....
int minimum;

for (int i=0; i < N-1; i++)
{
    minimum = i;

    for (int j=i+1; j < N; j++)
    {
        if (array[j] < array[minimum])
        {
            minimum = j;
        }
    }

    if(minimum != i)
    {
        int swap    = array[i];
        array[i]    = array[minimum];
        array[minimum] = swap;
    }
}
```

Теоретический анализ?

- Абстрагироваться от «железа»
- Абстрагироваться от входных данных

Получается т.н. «О-нотация» (Big-Oh notation):

- Время работы алгоритма выражается как функция от размера входных данных N
- Игнорируются константные коэффициенты
- Остается только старший порядок

Очень грубое объяснение! Подробнее см. «алгоритмическая сложность», «теория алгоритмов»

А можно сортировать быстрее?

- Если ничего не известно о входных данных
  - Быстрая сортировка – в среднем $O(N * \log(N))$
  - Сортировка слиянием - $O(N * \log(N))$

- Если известно
  - Поразрядная сортировка ($O(k * N)$)
  - Сортировка подсчетом ($O(k + N)$)