

ОСНОВЫ ЯЗЫКА C

Типы данных. Функции. Циклы. Простые программы

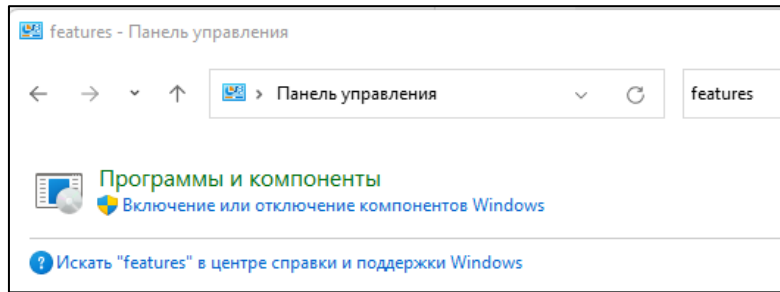
К. Владимиров, Syntacore, 2023
mail-to: konstantin.vladimirov@gmail.com

Как зовут преподавателя?

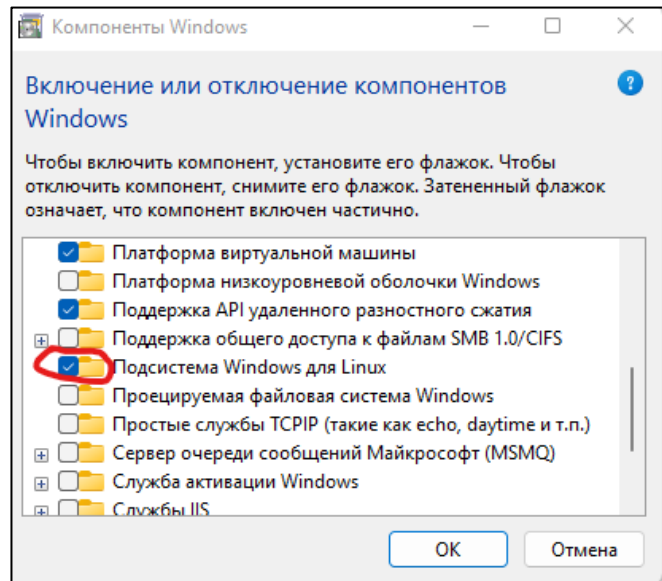
- Владимиров Константин Игоревич
- Email: `konstantin.vladimirov@gmail.com`
- Слайды: http://cs.mipt.ru/wp/?page_id=7775
- Контесты: <http://olymp1.vdi.mipt.ru>
- Исходный код: <https://github.com/tilir/c-graduate>
- Телефон: +7-903-842-27-55 (но лучше писать на email)

Как начать под Windows

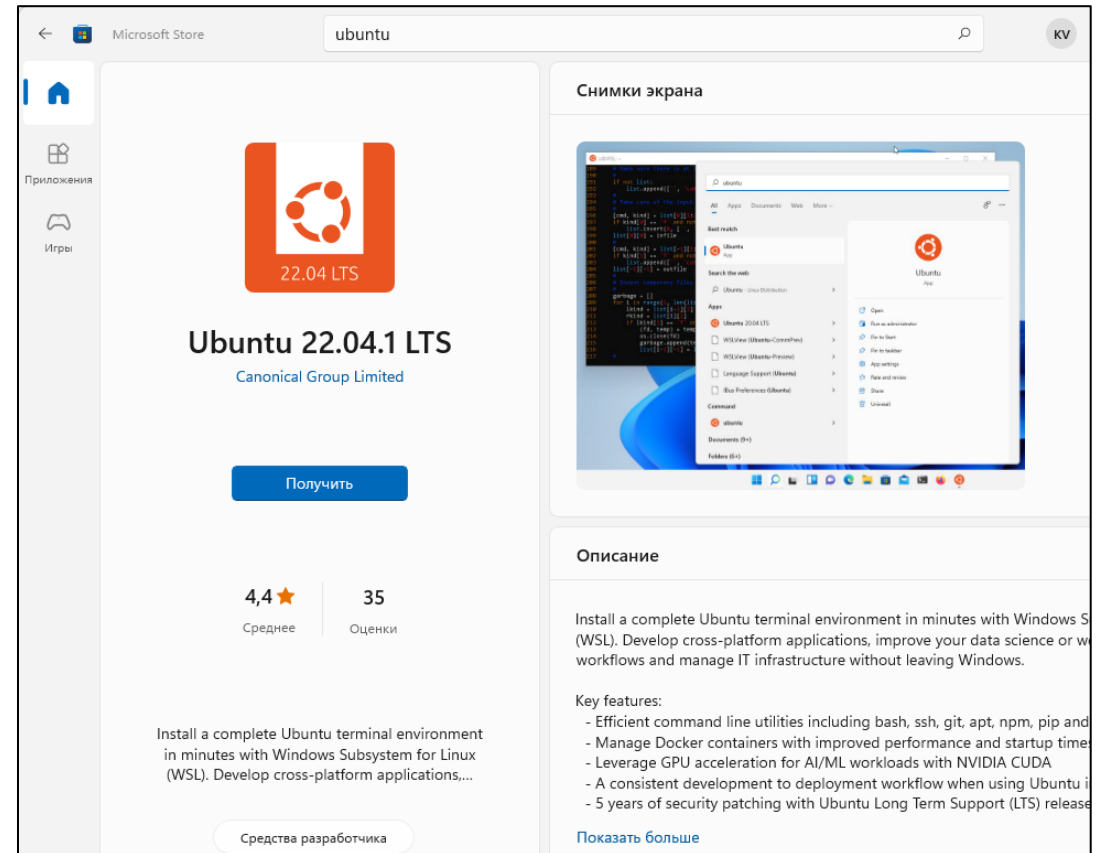
1.



2.



3.



Hello, world!

- Простейшая программа на языке C

```
#include <stdio.h>
```

```
int main() {  
    printf("Hello, world!\n");  
    return 0;  
}
```

- Сохраните её в файл hello.c
- Скомпилируйте и выполните.
- Используйте [K&R] если что-то не ясно.

- Включение заголовочного файла.

```
#include <имя_файла>
```

- Определение функции main.

```
int main() {  
    тело функции  
}
```

- Вызов функции printf.

```
printf(формат, аргументы);  
printf("%s\n", "Hello!");
```

Минимум о функциях

- **Объявление** функции задаёт **сигнатуру** (типы аргументов и тип значения).

```
int foo(int x, double y);
```

```
void bar(); // функция не возвращает значения
```

- Функция может быть вызвана даже если ещё не **определена**.

```
int t = 42; int s; s = foo(t, 1.0);
```

- Где-то в программе (но только один раз) должно найтись **тело** функции.

```
int foo(int x, double y) { int t = x + (int) y; return t; }
```

- Возвращаемое значение можно проигнорировать даже если оно не void.

```
int t = 42; foo(t, 1.0); // ok
```

Базовые стандартные функции

- Функция `main()` это точка входа вашей программы с неё начинается исполнение кода.
 - Она должна быть ровно одна.
 - По конвенции при правильном исполнении программы `main` возвращает ноль.
 - В командной строке `"echo $?"` печатает результат исполнения.
- Функция `printf` (вывод) и `scanf` (ввод) импортируются из [libc](#) и их объявления находятся в заголовочном файле [stdio.h](#)
- Функция `abort()` нужна чтобы прервать программу в произвольной точке, её объявление находится в [stdlib.h](#)

Warmup: частное и остаток

- Для всех a и $b \neq 0$, найдутся такие p и q , что $a = b * p + q$

```
#include <stdio.h>
```

```
int main() {  
    int a, b, p, q;  
    printf("input a and b: ");  
    scanf("%d%d", &a, &b);  
    p = a / b; q = a % b;  
    printf("p = %d, q = %d\n", p, q);  
    return 0; // не обязательно в main  
}
```

- Что за амперсанд в scanf и почему его нет в printf?

Компиляция и запуск

```
> gcc divmod.c -o divmod  
> ./divmod  
input a and b: 10 3  
p = 3, q = 1
```

Минимум об указателях

- Указатель на переменную это способ **косвенно** записать или прочитать её значение.

```
int a = 0;
```

```
int *p = &a; // & означает "взять адрес"
```

```
a = 1; // прямая запись, теперь a == 1 и *pa == 1
```

```
*p = 2; // косвенная запись, теперь a == 2 и *pa == 2
```

- Мы скоро поговорим об указателях гораздо более подробно.
- Чтобы изменить переменную внутри функции в неё передаётся указатель на переменную, поэтому scanf берёт указатели, а printf значения.

Warmup: частное и остаток

- Для всех a и $b \neq 0$, найдутся такие p и q , что $a = b * p + q$

```
#include <stdio.h>
```

```
int main() {  
    int a, b, p, q;  
    printf("input a and b: ");  
    scanf("%d%d", &a, &b);  
    p = a / b; q = a % b;  
    printf("p = %d, q = %d\n", p, q);  
}
```

- Ктонибудь видит проблемы в этом коде?

Компиляция и запуск

```
> gcc divmod.c -o divmod  
> ./divmod  
input a and b: 10 3  
p = 3, q = 1
```

Реальная жизнь

- В реальной жизни в программах бывают проблемы.

```
#include <stdio.h>
```

```
int main() {  
    int a, b, p, q;  
    printf("input a and b: ");  
    scanf("%d%d", &a, &b); // что если введены не два числа?  
    p = a / b; q = a % b; // что если b == 0?  
    printf("p = %d, q = %d\n", p, q);  
}
```

- Варианты решения?

Реальная жизнь: сообщить о проблеме

- Тщательная проверка ввода теперь занимает большую часть программы.

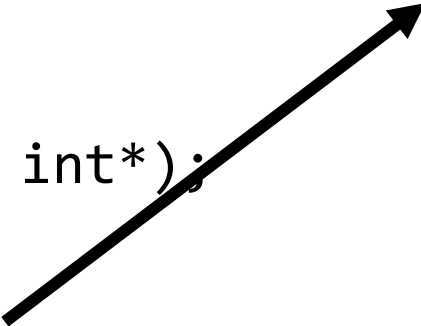
```
int a, b, p, q, nitems;  
printf("input a and b: ");  
  
nitems = scanf("%d%d", &a, &b);  
if (nitems != 2 || b == 0) {  
    printf("Error: input invalid, expect any a and b != 0");  
    abort();  
}  
  
p = a / b; q = a % b;  
printf("p = %d, q = %d\n", p, q);
```

Выносим проверку в функцию

```
#include <stdio.h>
#include <stdlib.h>

void read_input(int*, int*);

int main() {
    int a, b, p, q;
    read_input(&a, &b);
    p = a / b;
    q = a % b;
    printf("p: %d, q: %d\n", p, q);
}
```



```
void read_input(int *pa, int *pb) {
    int nitems;
    printf("input a and b: ");
    nitems = scanf("%d%d", pa, pb);

    if ((nitems != 2) || (*pb == 0))
        printf("Wrong input!\n");
    abort();
}
```

- Теперь мы уверены, что read_input считала аргументы корректно.
- Эту уверенность можно выразить языковыми средствами.

Проверка утверждений: assert

- Для проверки утверждений о программе можно поставить assert.

```
int main() {  
    int a, b, p, q;  
    read_input(&a, &b);  
  
    assert(b != 0); // b != 0 инвариант в этой точке  
    p = a / b;  
    q = a % b;  
    printf("p: %d, q: %d\n", p, q);  
}
```

- Проверка работает только в отладочной сборке, при подаче -DNDEBUG все assertions исчезают из кода.

Наибольший общий делитель

- Говорят, что целое число a делит b если их частное $\frac{b}{a}$ является целым числом.

$$a \mid b \Leftrightarrow \exists c \mid b = ac$$

- Наибольшее среди чисел, которые делят оба числа x и y называется их наибольшим общим делителем (greatest common divisor, gcd).

$$a = \gcd(x, y) \Leftrightarrow a = \max\{n \mid (n \mid x) \wedge (n \mid y)\}$$

- Например наибольшим общим делителем чисел 14 и 8 является число 2.
- Что является наибольшим общим делителем —14 и 8?
- Как найти наибольший общий делитель чисел 698917 и 102089?

Математический инсайт

- $x = y * p + q$
- Пусть k – общий делитель x и y .
- Тогда $q = x - y * p$ тоже делится на k .
- Значит $\gcd(x, y) = \gcd(y, q)$
- Например подсчитаем $\gcd(35, 13)$

$$35 = 13 * 2 + 9 \text{ значит } \gcd(35, 13) = \gcd(13, 9)$$

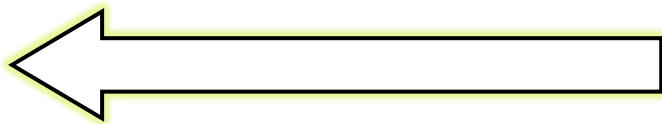
$$13 = 9 * 1 + 4 \text{ значит } \gcd(13, 9) = \gcd(9, 4)$$

- Сделайте последние два шага в уме.

Первый алгоритм: НОД

- $a \mid b \Leftrightarrow \exists c \mid b = ac$
- $a = \gcd(x, y) \Leftrightarrow a = \max\{n \mid (n \mid x) \wedge (n \mid y)\}$
- Задача: найти $\gcd(x, y)$

```
int gcd(int x, int y) {  
    ???  
}
```



$$x = y * p + q$$

Пусть k – общий делитель x и y

Тогда $q = x - y * p$ тоже делится на k

Значит $\gcd(x, y) = \gcd(y, q)$

- Как перевести математический инсайт в программу?
- Очень часто это основной вопрос в программировании.

Первый алгоритм: НОД

- $a \mid b \Leftrightarrow \exists c \mid b = ac$
- $a = \gcd(x, y) \Leftrightarrow a = \max\{n \mid (n \mid x) \wedge (n \mid y)\}$

```
int gcd(int x, int y) {  
    int q;  
    assert(y != 0);  
    q = x % y;  
    if (q == 0) return y;  
    return gcd(y, q);  
}
```

$$x = y * p + q$$

Пусть k – общий делитель x и y

Тогда $q = x - y * p$ тоже делится на k

Значит $\gcd(x, y) = \gcd(y, q)$

- Посчитайте $\gcd(698917, 102089)$.
- Как вы будете тестировать эту функцию?

Внезапная проблема

- Мы считаем $\text{gcd}(14, -8)$ и внезапно оказывается, что это -2 .
- Всё дело в тонкой разнице.
- В математике Евклидово деление определено следующим образом:
$$a = q * b + r, \quad 0 \leq r < |b|$$
- Тогда как в языке C операция `%` ведёт себя немного иначе:
`assert(a == (a / b) * b + (a % b));`
- Чтобы получить настоящий алгоритм Евклида, нам надо реализовать настоящее Евклидово деление.

Алгоритм Е

- Вычисляет $\text{gcd}(x, y)$, при этом числа могут идти в любом порядке.

```
int iabs(int x) { return (x < 0) ? -x : x; }
```

```
int eu_mod(int x, int y) {  
    int r;  
    assert(y != 0);  
    r = x % y; if (r < 0) r += iabs(y);  
    return r;  
}
```

```
int gcd(int x, int y) {  
    int q = eu_mod(x, y);  
    return (q == 0) ? y : gcd(y, q);  
}
```

Минимум о циклах

- В языке C есть три основных типа циклов: for, while и do-while.
- Циклы while: выполняются пока какое-то условие истинно.

```
while (i < 100) { /* тело цикла */ }
```

- Циклы do-while: точно выполняются единожды.

```
do { /* тело цикла */ } while (i < 100);
```

- Циклы for: содержат инициализацию и изменение **индуктивной переменной**.

```
for (i = 0; i < 100; ++i) { /* тело цикла */ }
```

```
i = 0; while(i < 100) { /* тело цикла */ ; ++i; }
```

Проблемы

- Вот мы и добрались до первой задачи, которую вам нужно решить, написав соответствующую программу самостоятельно.
 - Все такие проблемы имеют двух или трех буквенный код для простоты ссылок: Problem RL, Problem EE, ...
- Зайдите на сайт-задачник olymp1.vdi.mipt.ru и зарегистрируйтесь на первый контеcт. Добрый робот поможет вам найти ваши ошибки.
 - Если вы найдёте в интернете код решения на языке C, вы теряете немного практики и вам будет сложнее с теми проблемами, которые вы не сможете найти.
 - Полезно искать в интернете только идеи, а реализацию писать самостоятельно.
- Оценки в баллах за обычные проблемы ничего не значат и на вашу оценку в семестре не влияют.

Problem RL – рекурсия в цикл

- Ключевая часть алгоритма E рекурсивна.

```
int gcd(int x, int y) {  
    int q = eu_mod(x, y);  
    if (q == 0) return y;  
    return gcd(y, q);  
}
```

- Задача: написать то же самое без рекурсии, с явным циклом.
- Также эта задача поможет вам освоиться с системой контекста.

Problem CF: цепные дроби

- Цепная дробь это красивый математический объект, представляющий деление двух чисел как итерированные правильные дроби.

- Например

$$\frac{101}{60} = 1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{6 + \frac{1}{3}}}}$$

- Запишем короче: $\frac{101}{60} = [1; 1, 2, 6, 3]$
- Вам необходимо написать программу, которая берет на вход два числа и выдаёт все разряды цепной дроби в её дробной части.

Проблемы со звёздочкой

- Некоторые проблемы на этих слайдах обозначены звёздочкой: они могут быть существенно сложнее прочих.
- Они нужны если вы всё уже сделали, а вся группа пока не успела: делайте их, так как они интересные. Впрочем в интернете они ищутся не хуже.
- Ещё одно их применение: при оставшемся времени на семинарах или специально на допсеминарах (по желанию) мы можем поразбирать стоящую за ними теорию. Она обычно ещё более интересна.

Problem EE* – расширенный алгоритм E

- Даны два числа m и n
- Необходимо вычислить их общий делитель d и два числа a и b , такие, что $am + bn = d$
- Обратите внимание, числа m и n целые положительные, числа a и b целые, но могут быть отрицательными, например для $m = 1769$ и $n = 551$ имеем:
 $5 * 1769 - 16 * 551 = 29$, то есть $a = 5$ и $b = -16$
- Можно ли внести в алгоритм E простую модификацию, чтобы получить оба этих числа одновременно с общим делителем? Математический инсайт можно найти в [ТАОСР, 1.2.1].

Problem DE* : Диофантовы уравнения

- Линейное Диофантово уравнение имеет вид $ax + by = c$, где для заданных целых a, b, c нужно найти x, y
- Решений может быть много а может не быть вовсе.
- Диофантовы уравнения связаны с алгоритмом проблемы EE
 - В проблеме EE мы по x, y искали a, b, c
 - Можно видеть что это до некоторой степени обратная задача.
- Вам предлагается по заданным параметрам найти и вывести на экран любое решение если оно есть или вывести на экран $0\ 0$ если его нет.

Типы данных

- Пока что на слайдах и в задачах использовался только тип `int`.
- Но что такое `int`?

Типы данных

- Пока что на слайдах выше использовался только тип `int`.
- Но **что такое** `int`?
- `int` это множество всех его допустимых значений (value type).

Пример: число 1073741824 помещается в `int`, а 4294967296 или 1.1 нет.

- `int` это множество всех его допустимых операций (object type).

Пример: операция `x / y` имеет разный смысл для `int` и `double`.
операция `x % y` вообще невозможна для `double`.

```
assert(5.0 / 2.0 == 2.5);  
assert(5 / 2 == 2);
```

Типы данных: целые типы

- char, short, **int**, long, long long
 - $1 = \text{sizeof}(\text{char}) \leq \dots \leq \text{sizeof}(\text{long long})$
 - $\text{CHAR_BIT} \geq 8 \text{ bit}$
 - Диапазон от $-(2^{x-1} - 1)$ до $2^{x-1} - 1$, где $x = \text{sizeof}(T) * \text{CHAR_BIT}$
- Все эти типы имеют беззнаковых двойников:
 - unsigned char, unsigned short, **unsigned int**, unsigned long, unsigned long long
- Обычно $\text{sizeof}(T) == \text{sizeof}(\text{unsigned } T)$
- Диапазон от 0 до $2^x - 1$, где $x = \text{sizeof}(T) * \text{CHAR_BIT}$
- unsigned int принято записывать просто как **unsigned**.

Типы данных: вещественные типы

- Для научных вычислений принято приближать вещественные числа рациональными, используя идею **плавающей точки**.
- Например мы договариваемся, что у нас есть 8 **значащих разрядов**. Тогда возможны числа: 1024561, 10245.61, 10.24561, 1.024561.
- Но число 10245.6124561 может быть не отличимо от 10245.61 из-за чего плавающие числа коварны.
- Основные типы в языке C это **float** (примерно шесть значащих десятичных разрядов) и **double** (примерно пятнадцать).
- В будущем мы поговорим о них более подробно, пока что достаточно знать, что они существуют.

Суффиксы констант

- Разные константы имеют разные типы и это можно указать явно.

Константа	Тип	Константа	Тип
1	int	1.0	double
1u	unsigned	1.0f	float
1ll	long long	'0'	char (также '\x30')
1ull	unsigned long long	"abc"	const char[4]

- Это указание может быть очень важно при смешанных операциях.
- Можно использовать приведение, например `(unsigned int)(1)` это то же самое, что `1u`

Целочисленные продвижения

- Всё, что меньше чем `int`, смешанное с `int`, даёт `int`.

```
assert((unsigned char)(48) - 50 == -2); // uchar * int == int
assert('\x30' * 40 == 1920); // char * int == int
```

- Смешивание знакового и беззнакового целого даёт беззнаковый результат.

```
assert(5 - 10 == -5); // int - int == int
assert(5 - 10u == 4294967291u); // int - unsigned == unsigned
```

- Смешивание плавающего и целого даёт плавающий результат.

```
assert(5.0 / 2 == 2.5); // double / int == double
assert(5 / 2.0 == 2.5); // int / double == double
```


Ввод и вывод разных типов

- Для работы с разными типами, функции `printf` и `scanf` используют **форматные спецификаторы**.

```
int x = 2; unsigned long y = 0x30ff;  
printf("x = %d, y = %lu\n", x, y);
```

- Сводная табличка.

int	%d	long	%ld
unsigned	%u	long long	%lld
float	%f	unsigned long	%lu
char	%c, %hhd	unsigned long long	%llu
short	%hd	double	%lf

Обсуждение: алгоритм Евклида и типы

- Посчитайте $\text{gcd}(879609302220711, 443701051911)$
- Сколько шагов вычисления вы затратили на этот процесс?
- Вопрос для математически настроенной аудитории: сколько максимум шагов займёт алгоритм Евклида для двух 512-битных чисел?
- Возможен ли для огромных чисел алгоритм вычисления НОД более эффективный, чем алгоритм Евклида?*

Внезапная связь с алгоритмом Евклида

$$\frac{101}{60} = 1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{6 + \frac{1}{3}}}}$$

- Отрежем последний этаж дроби

$$1 + \frac{1}{1 + \frac{1}{2 + \frac{1}{6}}} = \frac{32}{19}$$

- Тогда, поскольку $\gcd(101, 60) = 1$, имеем $-101 * 19 + 60 * 32 = 1$, что и являлось искомыми коэффициентами в проблеме ЕЕ.

СЕМИНАР 1.2

Основы систем счисления и вычисления по модулю

Минимум о системах счисления

- Позиционное число $(\dots abc)_n$ эквивалентно выражению $c + b * n^1 + a * n^2 + \dots$
- При этом мы предполагаем $0 < \dots a, b, c < n$
- Например $(1101)_2 = (31)_4 = (15)_8 = (13)_{10} = (D)_{16}$
- В системе с основанием 16 мы придумали цифры A, B, C, D, E, F чтобы сохранить одну цифру на одной позиции.
- Для чисел в системе счисления с основанием 10 мы будем опускать основание.
- В языке C есть специальный способ записи констант: `015` и `0xD`.
- Также для `printf` есть специальный модификатор `"%x"` чтобы напечатать целое число в шестнадцатеричном представлении.

Упражнения с переводом систем

- Поупражняемся с числом $(110010010010)_2$
 - Переведите $(110010010010)_2$ в 8-ричное число.
 - Переведите $(110010010010)_2$ в 16-ричное число.
 - Чуть более сложно: переведите в десятичное число?
- Посмотрите что будет на экране после вывода ниже.

```
printf("%x\n", 302);
```

```
printf("%d\n", 0x12E);
```

- Философский вопрос: произвольное число типа `int` в какой системе счисления на самом деле?

Тренируемся без компьютера

- Переведите в систему счисления по основанию 2

$26 = ?$ $026 = ?$ $0x26 = ?$

- Переведите в систему счисления по основанию 8

$110101 = ?$ $34 = ?$ $0x34 = ?$

- Переведите в систему счисления по основанию 10

$1011 = ?$ $043 = ?$ $0x43 = ?$

- Переведите в систему счисления по основанию 16

$101001001 = ?$ $051 = ?$ $51 = ?$

Перевод делением

- Интуиция: допустим у нас есть число x . Тогда первый разряд x в двоичной системе это $x \bmod 2$ и дальше надо анализировать $\frac{x}{2}$
- Переведём 13 в двоичную систему последовательно вычисляя остатки.

$$13 \bmod 2 = 1$$

$$6 \bmod 2 = 0$$

$$3 \bmod 2 = 1$$

$$1 \bmod 2 = 1$$

- Ответ: 1101 и мы видим что ответ перевёрнутый, то есть нам нужно промежуточное место чтобы его хранить.

Минимум о массивах

- Массив определяется как последовательность элементов.

```
int a[3]; // массив a[0], a[1], a[2]
```

- Далее мы оперируем с его индивидуальными элементами.

```
a[0] = 1;
```

- Можно делать инициализированный массив.

```
int a[3] = {1, 2, 3}; // a[0] == 1, a[1] == 2, a[2] == 3
```

- Часть инициализаторов можно опускать.

```
int a[3] = {1}; // a[0] == 1, a[1] == 0, a[2] == 0
```

Problem NS – системы счисления

- Можно использовать деление с остатком, чтобы напечатать число в системе по основанию $base$. Предполагаем $base \leq 10$.
- $(12623)_{10} = (11000101001111)_2 = (122022112)_3 = (3011033)_4 = (400443)_5$
- Напишите функцию, печатающую число по основанию $base$ на экран.

```
int print_converted(unsigned n, unsigned base) {  
    // TODO: your code here  
}
```

- Вам понадобится вспомогательный массив. Чтобы оценить его размер, вспомните, что число типа `unsigned` имеет не более 32 разрядов в системе по основанию 2.

Problem FS* – система факториалов

- Число может быть единственным образом представлено в факториальной системе счисления.

$$463 = 3 \cdot 5! + 4 \cdot 4! + 1 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! = (34101)_!$$

- Правила довольно просты: у каждого $n!$ может быть коэффициент $0 \leq k \leq n$
- Ваша задача написать программу которая принимает на вход положительное десятичное число и печатает его в факториальной системе счисления.
- Вход: 463, выход: 3.4.1.0.1.

Математика по модулю

- Два числа называются равными по модулю m если m делит их разность.
- $10 \equiv 6 \pmod{2}$ так как $2 \mid (10 - 6)$
- Вообще все чётные числа равны по модулю 2: $10 \equiv 0 \pmod{2}$.
- В некотором смысле по модулю 2 существует всего два числа: 0 и 1.
- Законы арифметики по модулю:

$$(a + b) \bmod m == ((a \bmod m) + (b \bmod m)) \bmod m$$

$$(a * b) \bmod m == ((a \bmod m) * (b \bmod m)) \bmod m$$

- Пример: $29 * 17 \pmod{3} = 2 * 2 \pmod{3} = 1 \pmod{3}$

Пример: в степень по модулю

- Наивная реализация на С это простой цикл:

```
// возвращает  $n^k \pmod m$ 
unsigned pow_mod(unsigned n, unsigned k, unsigned m) {
    unsigned long long mult = n % m;
    unsigned long long prod = mult;
    while (k > 1) {
        prod = (prod * mult) % m;
        k -= 1;
    }
    return prod;
}
```

- Это решение не слишком эффективно. Можем ли мы его улучшить?

Русское крестьянское умножение

- Представим что у нашего компьютера нет встроенной возможности умножать.
- Как бы мы тогда реализовали $(a * b) \bmod m$?

```
// возвращает n * k (m)
unsigned mul_mod(unsigned n, unsigned k, unsigned m) {
    // очень не хочется k раз складывать
}
```

- Подумайте о бинарном представлении k. Сколько на самом деле сложений вам надо сделать?

Алгоритм POWM

- См. также [*TAOCP Algorithm 4.6.3A*]

```
unsigned pow_mod(unsigned n, unsigned k, unsigned m) {  
    unsigned mult = n % m;  
    unsigned prod = 1;  
    while (k > 0) {  
        if ((k % 2) == 1) {  
            prod = (prod * mult) % m; k = k - 1;  
        }  
        mult = (mult * mult) % m; k = k / 2;  
    }  
    return prod;  
}
```

Problem RPS: возводим в сверхстепень

- Сверхстепенью $a \uparrow\uparrow b$ называется итерация возведения в степень $a^{a^{\dots^a}}$.
- Ваша задача посчитать $(a \uparrow\uparrow b) \bmod m$
- Например $2 \uparrow\uparrow 4 = 2^{2^{2^2}} = 2^{16} = 65536$, $2 \uparrow\uparrow 4 \bmod 10 = 6$

```
unsigned spow_mod(unsigned n, unsigned k, unsigned m) {  
    // ваш код здесь  
}
```

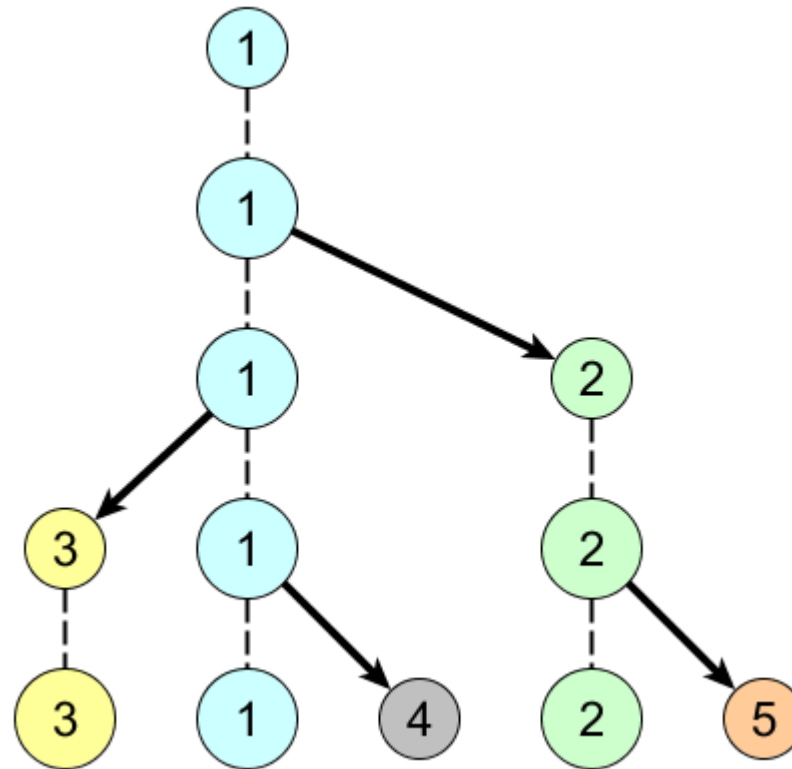

СЕМИНАР 1.3

Числа Фибоначчи

Числа Фибоначчи



- Паре молодых кроликов нужен месяц чтобы вырасти.
- Каждая пара взрослых кроликов приносит пару молодых кроликов каждый месяц.
- **Сколько кроликов будет через год?**



0	F_0
1	F_1
1	F_2
2	F_3
3	F_4
5	F_5

Наивное решение

- Очевидная рекуррентность $F_n = F_{n-1} + F_{n-2}$
- Ведёт к очевидной функции на C.

```
unsigned long long fib(unsigned n) {  
    if (n == 0) return 0ull;  
    if (n <= 2) return 1ull;  
    return fib(n - 1) + fib(n - 2);  
}
```

- Хороша ли эта функция?
- Посчитайте на своей машине `fib(10)`, `fib(20)`, `fib(50)`, `fib(80)`.
- Визуализация: <http://www.cs.usfca.edu/~galles/visualization/DPFib.html>

Алгоритм F

- Заменяем рекурсию на цикл (каждая итерация обновляет два числа).

```
unsigned long long fib(unsigned n) {  
    unsigned long long first = 0ull, second = 1ull; int idx;  
    if (n == 0) return 0ull;  
    for (idx = 2; idx <= n; ++idx) {  
        unsigned long long tmp = second;  
        second = second + first;  
        first = tmp;  
    }  
    return second;  
}
```

- Посчитайте на своей машине `fib(10)`, `fib(20)`, `fib(50)`, `fib(80)`

Соблазн плавающих чисел

- Вообще-то для чисел Фибоначчи легко вывести точную формулу.

$$F_n = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}, \text{ где } \phi = \frac{1+\sqrt{5}}{2}, \hat{\phi} = \frac{1-\sqrt{5}}{2}$$

- Почему мы не воспользовались ей?
- Потому что плавающие числа коварны.
- В языке C есть три вида плавающих чисел: `float`, `double`, `long double`
- Также есть много операций с ними, такие как `pow`, `sqrt`, `round`, etc...
- Давайте посмотрим возможную реализацию Фибоначчи через плавающие числа.

Алгоритм FF

- Наивная реализация на C для плавающих чисел двойной точности.

```
#include <math.h>
```

```
unsigned long long fibd(unsigned n) {  
    double phi = (1.0 + sqrt(5.0)) / 2.0;  
    return round(pow(phi, n) / sqrt(5.0));  
}
```

- Вычислите `fibd(20)`, `fibd(50)`, `fibd(80)`
- Сравните с результатами алгоритма F.
- Мы обязательно займёмся плавающими числами, но позже.

Переполнения в числах

- Подсчитайте `fib(91) ... fib(100)` с помощью алгоритма F.
- Что на экране?

Переполнения

- Подсчитайте `fib(91) ... fib(100)` с помощью алгоритма F.
- Что на экране?

`fib(91) = 4660046610375530309`

`fib(92) = 7540113804746346429`

`fib(93) = 12200160415121876738`

`fib(94) = 1293530146158671551 // oops`

- До 93-го числа всё шло хорошо, но 94-е очевидно неверно, должно быть `19740274219868223167`
- Давайте подсчитаем их в [шестнадцатеричном виде](#)

Переполнения

- Подсчитайте `fib(91) ... fib(100)` с помощью алгоритма F в hex-виде.
- Что на экране?

`fib(91) = 40abcfb3c0325745`

`fib(92) = 68a3dd8e61eccfbdb`

`fib(93) = a94fad42221f2702`

`fib(94) = 11f38ad0840bf6bf // oops`

- Теперь очевидно, что сложение 64-битных чисел идёт по модулю 2^{64} .
- Настоящий результат `111F38AD0840BF6BF` был сокращён по этому модулю.

Разряды чисел Фибоначчи

- Числа Фибоначчи растут **очень** быстро.

`fib(200) == 280571172992510140037611932413038677189525`

- Увы, это не влезет даже в 64 бита.
- Но мы можем подсчитать последний разряд числа `fib(200) % 10 == 5`
- Последние разряды чисел Фибоначчи это числа Фибоначчи **по модулю 10**.

1, 1, 2, 3, 5, 8, 3, 1, 4,

- `assert((8 + 3) % 10 == 1);`
- Напишите программу на C которая подсчитает последний разряд числа `fib(331)`, используя алгоритм F и модульную арифметику.

Problem FM – вычисления по модулю m

- Обобщите функцию из предыдущей задачи до функции вычисления n -го числа Фибоначчи по любому модулю m .

```
int fib_mod(unsigned n, unsigned m);
```

- Выведите на экран последовательности Фибоначчи по модулям 2, 3, 4, 5, 6, 7, 8, 9, 10 хотя бы по 30 элементов каждой последовательности.
- Видите ли вы какие-нибудь закономерности?
- Дополнительный вопрос: как вы обработали случай $m == 0$?

Problem PP – периоды Пизано

- В прошлой задаче были замечены некие закономерности в последовательностях Фибоначчи по модулю m : они периодичны и новый период всегда начинается с 0, 1.
- Напишите функцию, которая ищет длину периода по модулю m (этот период называется периодом Пизано) и обобщите функцию `fib` для гигантских номеров.

```
int get_pisano_period(unsigned m);  
int fib_mod(unsigned long long n, unsigned m);
```

- Посчитайте число $F_{2816213588}$ (пожалуй в этом числе больше цифр, чем символов на этих слайдах) по модулю 30524.

Частичная оптимизация

- Общая формула для периода Пизано до сих пор неизвестна. Но подсчитаны некоторые небольшие периоды.
- Скорее всего в вашем решении предыдущей задачи использовалась редукция типа такой:

```
unsigned fib_mod(unsigned long long n, unsigned m) {  
    n = n % get_pisano_period(m);  
    // ..... и так далее .....
```

- Её можно оптимизировать, предвычислив некоторые небольшие периоды и используя их как таблицу.

Массивы и мемоизация

```
unsigned pisanos[1000] = {0}; // означает {0, 0, 0, .... 0}
```

```
int fib_mod(unsigned long long n, unsigned m) {  
    assert (m > 0);  
    if (m < 1000) {  
        if (pisanos[m] == 0)  
            pisanos[m] = get_pisano_period(m);  
        n = n % pisanos[m];  
    } else {  
        n = n % get_pisano_period(m);  
    }  
}
```

- Глобальный массив по умолчанию инициализирован нулями, 0 маркирует невалидное значение.
- Мы можем частично инициализировать массив.

Массивы и мемоизация

```
unsigned pisanos[1000] = { 0, 1, 3, 8, 6, 20, 24, 16, 12, 24,  
                           60, 10, 24, 28, 48, 40, 24, 36 };
```

```
int fib_mod(unsigned long long n, unsigned m) {  
    assert (m > 0);  
    if (m < 1000) {  
        if (pisanos[m] == 0)  
            pisanos[m] = get_pisano_period(m);  
        n = n % pisanos[m];  
    } else {  
        n = n % get_pisano_period(m);  
    }  
}
```

- Все неинициализированные элементы глобального массива нулевые.

Инициализированность массивов

- Локальный массив по умолчанию не инициализирован и считывать из него данные это серьёзная ошибка.

```
int foo () {  
    int wrong[100];  
    int corr[100] = {0};  
    printf ("%d\n", wrong[3]); // напечатает всё что угодно  
    printf ("%d\n", corr[3]); // напечатает 0  
    arr[3] = 1;  
    printf ("%d\n", wrong[3]); // напечатает 1  
}
```

- Избегайте неинициализированных массивов и неинициализированных переменных в своих программах.

Problem SF – система чисел Фибоначчи

- Любое положительное число единственным образом представимо как сумма Фибоначчи если запретить в представлении две единицы рядом.
- $8 = \{10000\}_F = 5 + 3 = \{1100\}_F = 5 + 2 + 1 = \{1011\}_F$
- $20 = 13 + 5 + 2 = \{101010\}_F$
- $30987 = 28657 + 1597 + 610 + 89 + 34$
- Напишите программу которая будет печатать число типа unsigned int в представлении Фибоначчи
- Возможно вы захотите **мемоизировать** числа Фибоначчи чтобы быстро искать ближайшее число Фибоначчи (как 28657 для 30987)

[Fibonacci coding](#)

[Fibonacci system online](#)

Обсуждение

- Можно ли превзойти по скорости алгоритм для проблемы РР?
- Да, и тут снова поможет математический инсайт

Оказывается $\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$ см. [ТАОСР 1.2.8 (5)]

- Мы ещё не проходили многомерных массивов. Но можно заметить, что матрица 2×2 это всего четыре числа и представить её как массив из четырёх элементов
- Тогда A_{ij} это $A[i * 2 + j]$, где $i, j \in \{0,1\}$

Problem MF* – матрицы Фибоначчи

- Используйте матрицы 2x2, представленные массивами, для быстрого вычисления по гигантскому модулю гигантских чисел Фибоначчи

```
int fib_mod(unsigned long long n, unsigned long long m);
```

- Периоды Пизано тут уже не сработают так как модули гигантские, а вот матрицы дадут быстрый ответ
- Возможно вы захотите адаптировать алгоритм POWM для быстрого возведения в степень уже не числа, а матрицы

Домашние работы

- Некоторые задачи помечены буквой H: HWF, HWE, etc...
- Они больше внутри, чем кажутся снаружи. Над ними положено думать дома и для их решения необходимо, но недостаточно просто пройти контекст.
- После прохождения контекста высылайте их на email преподавателю.
- У каждой такой работы есть до восьми дополнительных уровней с экспоненциально возрастающей сложностью (по желанию). Задачи высоких уровней нигде не публикуются, следующий уровень высылается по прохождении предыдущего.

Домашнее задание HWF

- Два игрока играют в интересную игру (источник: [ТАОСР 1.2.8] задача 37)
 - Изначально дано N спичек.
 - Первый игрок берёт любое количество, но не все сразу спички. Теперь второй может взять не больше, чем вдвое больше чем первый. Далее первый берёт не больше чем вдвое больше второго. И так далее.
 - Выигрывает тот, кто взял последнюю спичку
- Примеры
 - 11 спичек. Первый берёт 4, второй может взять до 8 и берёт 7, победа. Запись партии: 11.4,7!
 - Ещё варианты: 11.3,3,5! 11.3.2.1.1.1.1.2! и т.д.
- Ваша задача: написать программу которая достаточно хороша в этой игре

Литература

- [C11] ISO/IEC, "Information technology – Programming languages – C", ISO/IEC 9899: 2011
- [K&R] Brian W. Kernighan, Dennis Ritchie – The C programming language, 1988
- [KGP] Ronald L. Graham, Donald E. Knuth, Oren Patashnik – Concrete Mathematics: A Foundation for Computer Science, 1994
- [TAOCP] Donald E. Knuth – The Art of Computer Programming, Volume 1, 2011

