

ОСНОВЫ ПРОГРАММНОГО КОНСТРУИРОВАНИЯ

Лекция № 5
3 октября 2016 г.



ПОИСК ПОДСТРОКИ В СТРОКЕ

- Задача: дана строка — «стог сена» длины N (haystack). Определить, встречается ли в ней строка-«иголка» длины M (needle) и если да, то на какой позиции.
- Метод грубой силы: подставляем «иголку» к каждой возможной позиции в «стоге сена», покуда не совпадет или «стог» не кончится.
Количество сравнений: $O((N-M) \cdot M)$.

АЛГОРИТМ БОУЭРА-МУРА (МОДИФИЦИРОВАННЫЙ)

«стог сена»

b c **a** **b** c **b** c b a a ...

«иголка»

a a a a a

- Явно нет смысла сдвигать «иголку» на одну позицию вправо.
- Хорошо бы как-то учесть частичное совпадение строк и не проверять все символы заново каждый раз.

СУТЬ АЛГОРИТМА

- Подставляем «иголку» к началу «стога» ($i = M - l$).
- Сравниваем символы с конца «иголки»: $h[k]$ и $n[j]$, уменьшая k и j (изначально $k = i, j=M - l$).
- Если j дошло до начала «иголки», подстрока найдена!
- Если на каком-то шаге $h[k] \neq n[j]$, то сдвигаем «иголку» на $d[h[i]]$ вправо ($i += d[h[i]]$).

МАССИВ СДИГОВ

d — хитрый массив, индексируемый символами **x** из конечного алфавита. Для каждого символа:

- если **x** отсутствует в **n**, то **d[x]** равно **M**;
- если **x** – не последний в **n**, то **d[x]** равно расстоянию от последнего вхождения **x** в **n** до конца **n**;
- если **x** – последний в **n**, то **d[x]** равно расстоянию от предпоследнего вхождения **x** в **n** до конца **n**.

$n = "abcabc"$
$d['b'] = 1$
$d['c'] = 2$
$d['a'] = 4$
$d[\dots] = 5$

АНАЛИЗ АЛГОРИТМА БОУЭРА-МУРА

- На практике работает очень хорошо, вплоть до:
 $O(N/M)$.
- В худшем случае (поиск "abbbb" в "bbbbbbbbbb"):
 $O((N-M) \cdot M)$.

АЛГОРИТМ РАБИНА-КАРПА

Вместо сравнения подстрок будем использовать сравнение их хешей.

- Хеш-функция быстро преобразует произвольную строку в некоторое численное значение, причем равные строки преобразуются в равные значения.

Основание исчисления R

- Например:

$$\text{hash}("abc") = (97 \cdot 256^2 + 98 \cdot 256^1 + 99 \cdot 256^0) \% 997$$

Некое большое простое число Q

СУТЬ АЛГОРИТМА

- T_n — хеш «иголки».
- $T_{h,k}$ — хеш M символов «стога», начиная с позиции k .
- Перебираем $k = 0 \dots (N-M)$ и сравниваем T_n и $T_{h,k}$.
 - Если совпали, то проверяем посимвольно, и в случае совпадения — успех.
 - Иначе идем дальше.

ХИТРОСТЬ ВЫЧИСЛЕНИЯ ХЕШЕЙ В «СТОГЕ СЕНА»

- $T_{h,k} = R^{M-1} \cdot h[k] + R^{M-2} \cdot h[k+1] + \dots + R^0 \cdot h[k+M-1]$
- $T_{h,k+1} = R^{M-1} \cdot h[k+1] + R^{M-2} \cdot h[k+2] + \dots + R^0 \cdot h[k+M]$
- Если мы знаем $T_{h,k}$, то за константное время можно вычислить $T_{h,k+1}$:
$$T_{h,k+1} = (T_{h,k} - R^{M-1} \cdot h[k]) \cdot R + h[k+M].$$

АНАЛИЗ АЛГОРИТМА РАБИНА-КАРПА

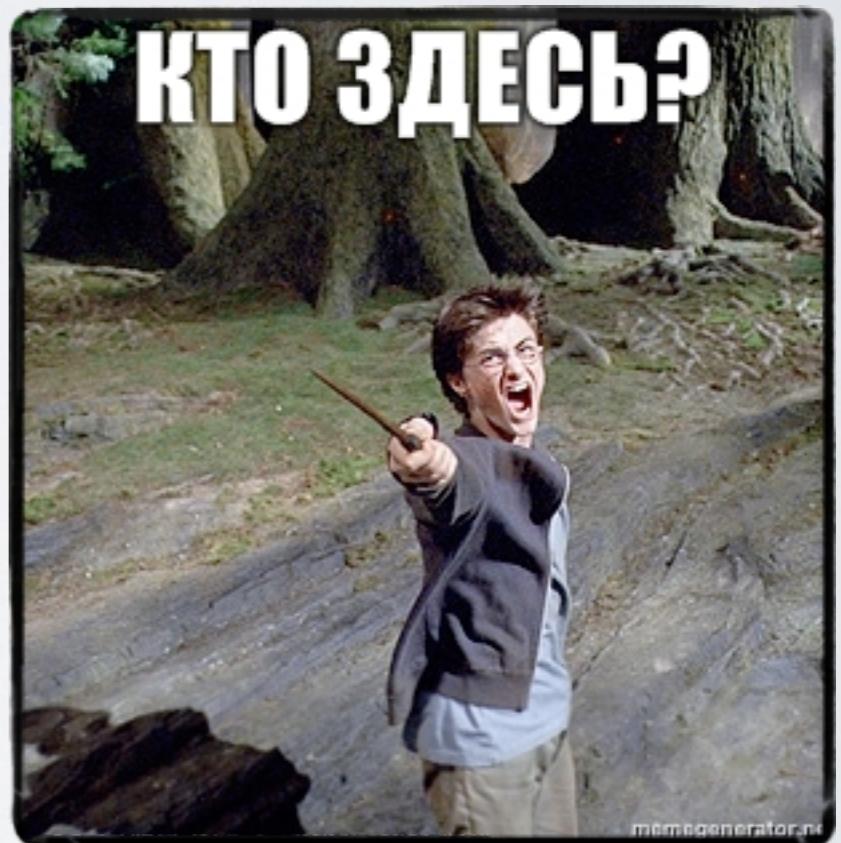
- В среднем алгоритм работает за $O(N-M)$.
 - При достаточно большом простом Q , вероятность ложных совпадений ($1/Q$).
- В худшем случае $O((N-M) \cdot M)$.
- Отлично подходит для поиска нескольких «иголок» в одном «стоге».

АЛГОРИТМ КНУТА- МОРРИСА-ПРАТТА

- В худшем случае работает за $O(M+N)$.
- Предварительно строит по «иголке» двумерный массив сдвигов.
- Во время прохода по «стогу», никогда не идет назад.
- Оптимальный алгоритм, но непростой.

ПРОФЕССИЯ: ПРОГРАММИСТ

- Пожарные борются с огнем.
- Педагоги борются с детьми.
- Полиция борется с преступностью.
- Программисты борются ...



... со сложностью!

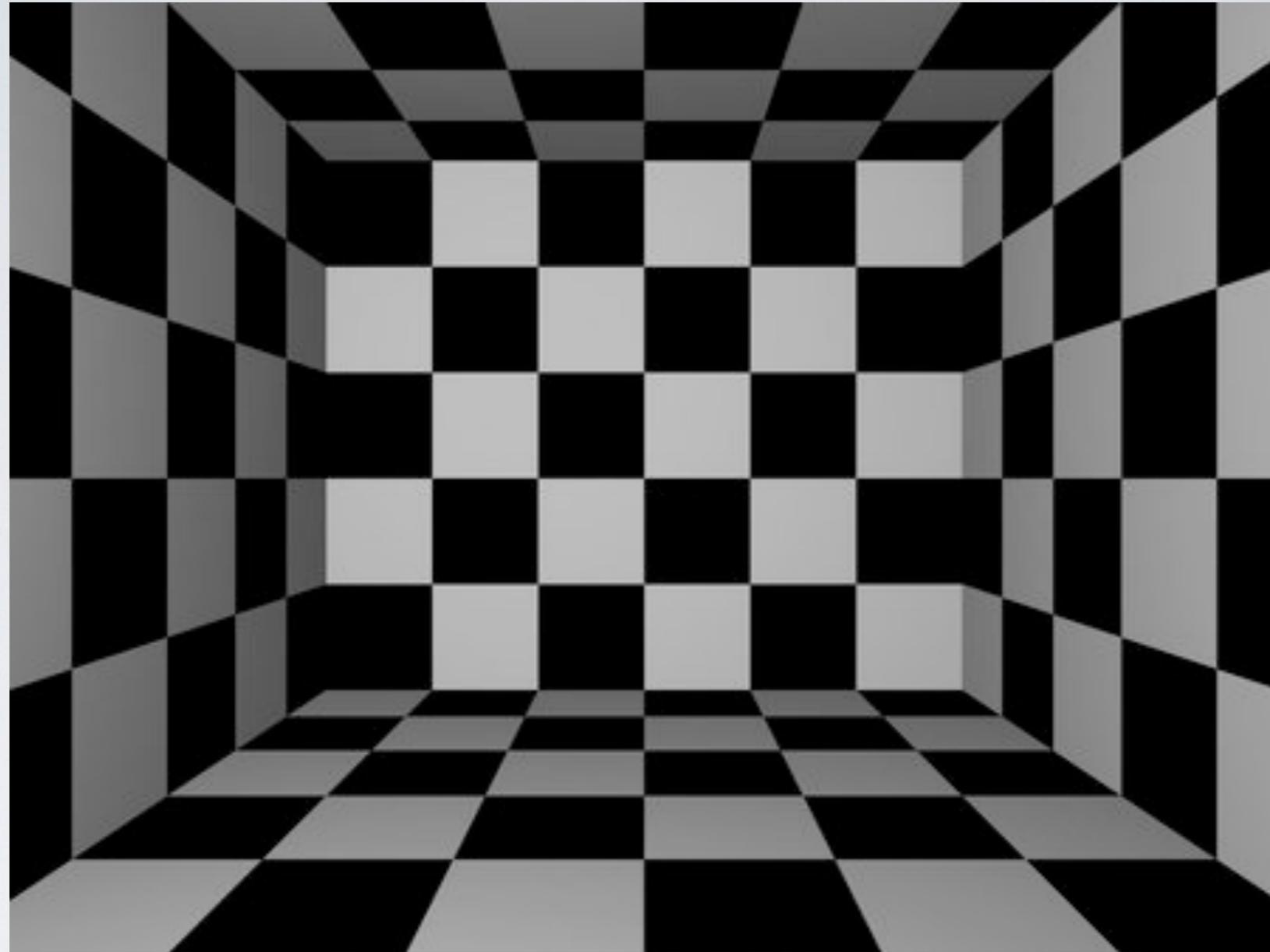
ЧТО СЛОЖНЕЕ?



РАБОТА С ЧЕРНЫМИ ЯЩИКАМИ

```
#include "blackbox.h"

int main() {
    blackbox_prepare();
    return blackbox_work_hard();
}
```



ЕСЛИ НЕТ ЧЕРНОГО ЯЩИКА?

Сделать его с помощью других ящиков!

Динамическая память:
malloc и free

Низкоуровневые функции
работы с сетью
(connect, recv, send, close)

Динамический массив

Скачивание файла по
адресу

Очередь

Распознавание и чтение
форматов (jpg, png, ...)

Скачивалка
изображений

Алгоритм resize для
изображений

ДЕКОМПОЗИЦИЯ



ПЕРВОЕ ПРЕДСТАВЛЕНИЕ МОДУЛЯ



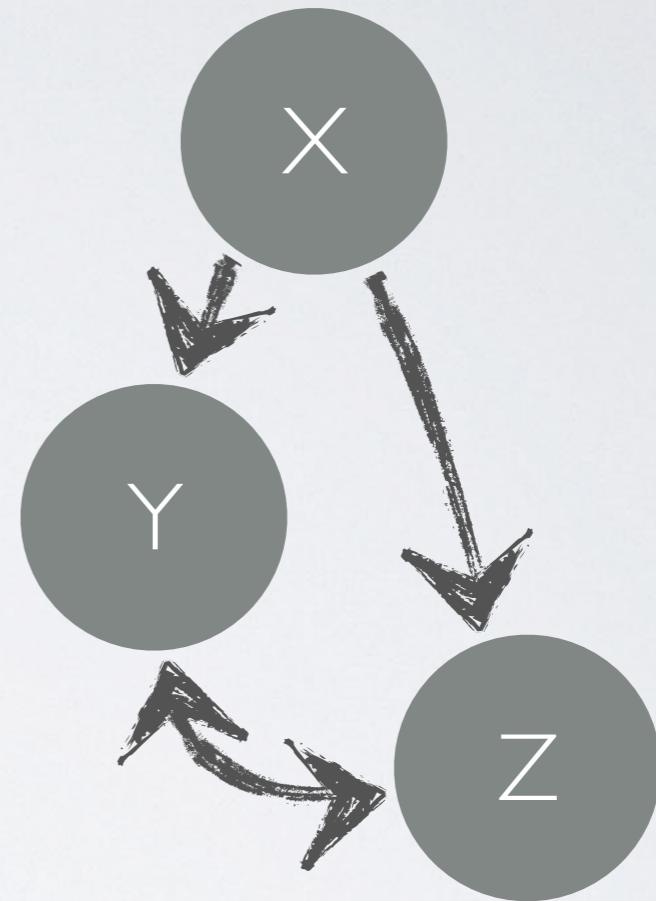
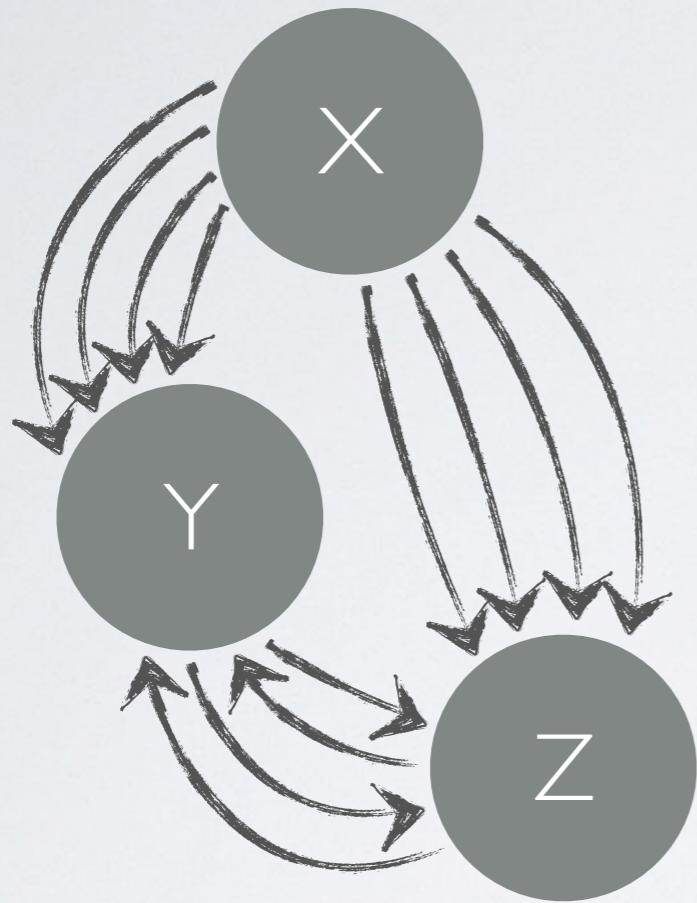
ПРОСТЕЙШИЕ СПОСОБЫ ДЕКОМПОЗИЦИИ

- Взаимодействие с пользователем (**User Interface**) – отдельный модуль.
- Обработка разных структур данных – в разных модулях (**list.c**, **stack.c**, **queue.c**, ...).
- Элементы функциональности, близкие по смыслу – в один модуль (**io.c**, **parse.c**, **errors.c**).
- Если процесс естественным образом разбивается на отдельные шаги, то каждый шаг – в свой модуль (**prepare.c**, **get_data.c**, **process.c**, **output.c**).

УТОЧНЕНИЕ ПРЕДСТАВЛЕНИЯ

- Модули состоят из функций.
- Функции вызывают другие функции, из своего модуля и из других модулей.
- Вызовы функций – связи между модулями.

КАРТИНКИ СВЯЗЕЙ



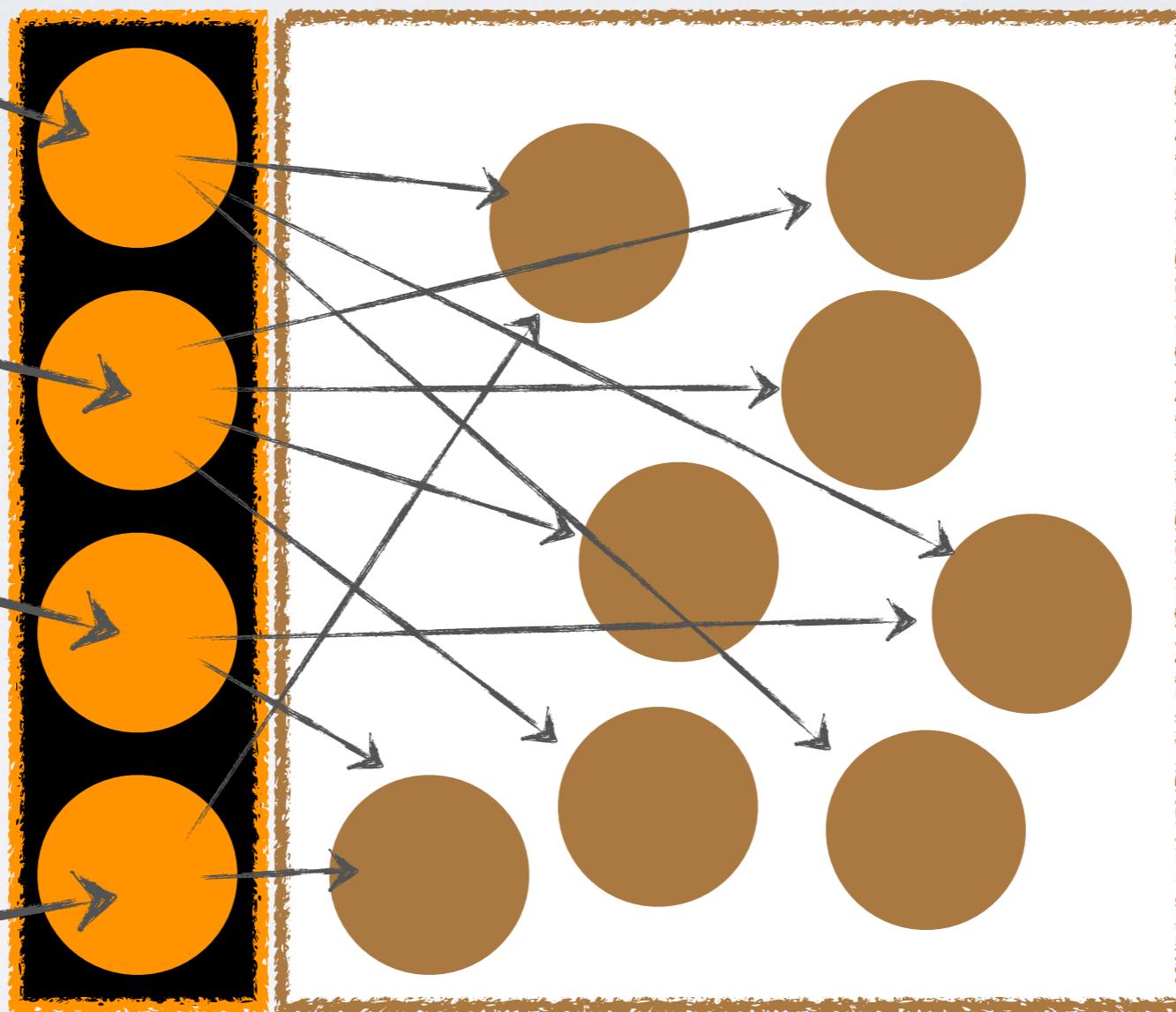
Что проще?

Мера сложности системы – количество связей.

МЕРЫ ПРОТИВ СЛОЖНОСТИ

Черный ящик

Прозрачный ящик



Интерфейс

Реализация

ИНТЕРФЕЙС И РЕАЛИЗАЦИЯ

- Адресная книга:
- **Интерфейс**: операции (добавить запись, найти по имени).
- **Реализация** операций: код функций, основанный на конкретном представлении хранилища и выбранных алгоритмах, а также вспомогательные функции.



УСТРОЙСТВО МОДУЛЕЙ В С

- Каждый модуль состоит как минимум из двух файлов:
 - **module_name.h** – заголовочный файл, содержит определение констант (#define, enum), типов данных (struct, typedef) и прототипы функций интерфейса.
 - **module_name.c** – файл с кодом функций интерфейса и вспомогательных функций.
- Использование модуля:
 - `#include "module_name.h"`

RATIONAL.H

```
#ifndef RATIONAL_H
#define RATIONAL_H

#include <stdio.h>

typedef struct _Rational {
    int numer;
    int denom;
} Rational;

void rat_create(Rational *res, int a, int b);

void rat_add(Rational *result, Rational *a, Rational *b);
void rat_sub(Rational *result, Rational *a, Rational *b);
void rat_mul(Rational *result, Rational *a, Rational *b);
void rat_div(Rational *result, Rational *a, Rational *b);

void rat_power(Rational *result, Rational *r, int power);

int rat_to_i(Rational *a);
double rat_to_f(Rational *a);

int rat_compare(Rational *a, Rational *b);
void rat_print(Rational *a, FILE *fp);

#endif
```

RATIONAL.C

```
#include "rational.h"

static void normalize(Rational *rat);

void rat_mul(Rational *result, Rational *a, Rational *b) {
    result->numer = a->numer * b->numer;
    result->denom = a->denom * b->denom;

    normalize(result);
}

/* ... */

static int gcd(int p, int q) {
    /* ... */
}

static void normalize(Rational *rat) {
    int g = gcd(rat->numer, rat->denom);
    rat->numer /= g;
    rat->denom /= g;
}
```

MAIN.C

```
#include "rational.h"

int main(void) {
    Rational p, q, r;

    rat_create(&p, 1, 2);
    rat_create(&q, 1, 3);
    rat_add(&r, &p, &q);
    rat_print(&r, stdout);
    return 0;
}
```

СТАНДАРТНАЯ БИБЛИОТЕКА С

- Набор интерфейсов!
- **stdio.h** – ввод/вывод.
- **string.h** – работа со строками.
- **math.h** – математические функции.
- **time.h** – функции работы со временем и датой.
- **stdlib.h** – прочие важные функции.



ПОЧЕМУ ИНТЕРФЕЙС ВАЖНЕЕ РЕАЛИЗАЦИИ?

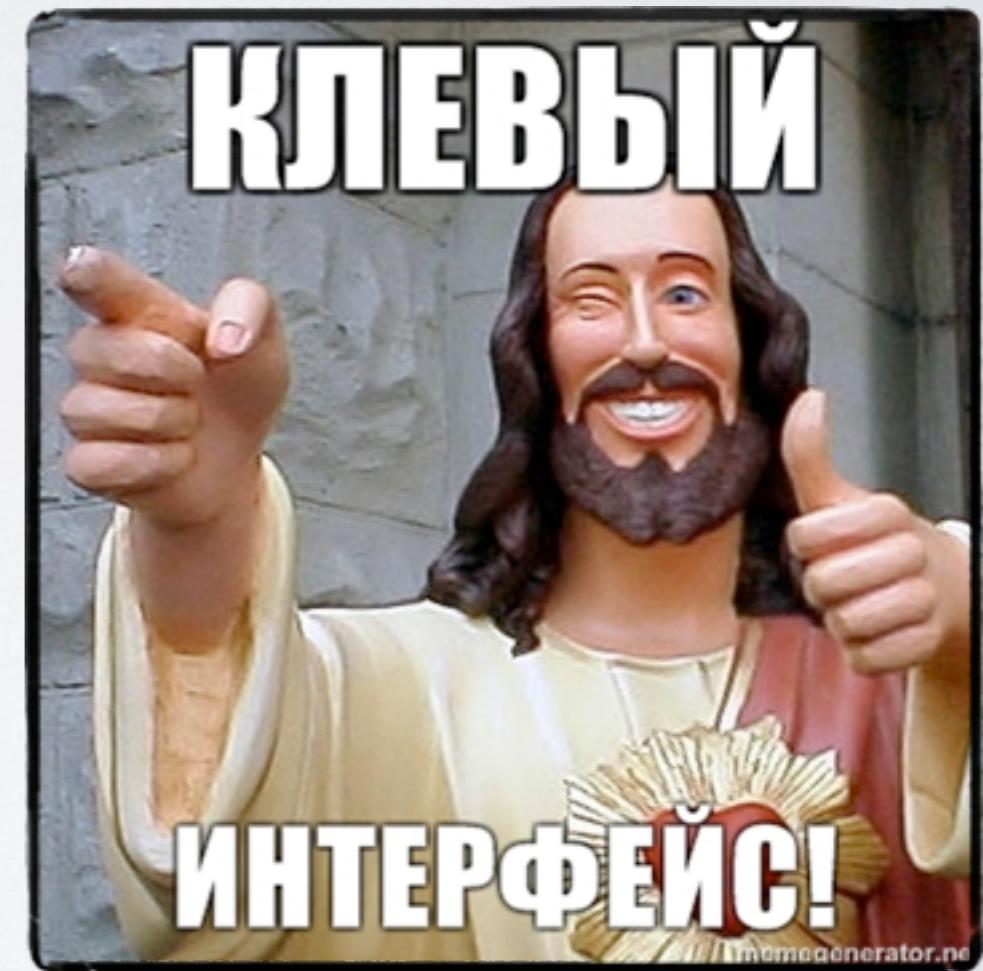
- На один интерфейс может быть несколько реализаций (но не наоборот!).
- Если интерфейс удачный, модуль легче повторно использовать.
- Если интерфейс выпущен в широкую публику, его гораздо сложнее изменить.
- Интерфейс — это то, что увидят другие, реализацию могут и не увидеть.

ОПЕРАЦИОННАЯ СИСТЕМА КАК ИНТЕРФЕЙС

- ▶ What would you do if you had to do it over again?
- ▶ Ken Thompson (*the creator of UNIX*): I'd spell creat with e.

ХОРОШИЙ ИНТЕРФЕЙС

- Легко изучить.
- Легко использовать, даже без документации.
- Сложно использовать неправильно.
- Код, написанный с помощью этого интерфейса, легко читать и поддерживать.
- Достаточная сила (количество возможностей).
- Легкая расширяемость.
- Подходит пользователям, которые будут его использовать.





КОНЕЦ ПЯТОЙ ЛЕКЦИИ

Больше интерфейсов, хороших и разных!