ОСНОВЫ ЯЗЫКА C

Типы данных. Функции. Циклы. Простые программы

К. Владимиров, Intel, 2018
mail-to: konstantin.vladimirov@gmail.com

Как зовут преподавателя?

- Владимиров Константин Игоревич
- Телефон: +7-903-842-27-55
- Email: konstantin.vladimirov@gmail.com
- Слайды: sourceforge.net/projects/cpp-lects-rus и далее files/slides-informatics/2018
- Все домашние работы высылать на email

Настройка рабочего места

- Компилятор g++

```
> cat hello.c

#include <stdio.h>

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

> g++ hello.c -g -o hello

> ./hello

Hello, world!
```

- Отладчик gdb

```
> gdb hello

tui enable

next

refresh == Ctrl+L

Enter == repeat command
```

Warmup: частное и остаток

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

На языке C:

```
#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);
}
```

Запуск

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

- Кто нибудь видит проблемы в этом коде?

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?

Assert, assert, assert

- Первый вариант: проверка утверждений (assertions)

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

int main () {
  int a, b, p, q, nitems;
  printf("input a and b: ");
  nitems = scanf("%d %d", &a, &b);
  assert((nitems == 2) && (b != 0));
  p = a / b;
  q = a % b;
  printf("p = %d, q = %d\n", p, q);
}
```

- Ещё варианты?

Проверить и спросить ещё раз

- Второй вариант: тщательная проверка ввода

```
int main () {
  int a, b, p, q, nitems;
  for (;;) {
    printf("input a and b: ");
    nitems = scanf("%d %d", &a, &b);
    if ((nitems == 2) && (b != 0))
      break;
    printf("Wrong input!\n");
    fpurge(stdin); // or scanf("%*s");
  }
  p = a / b; q = a % b;
  printf("p = %d, q = %d\n", p, q);
}
```

Хочется вынести проверку ввода в отдельную функцию

Минимум о функциях

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

```
int foo(int x, double y);
void bar(); // эта функция ничего не вернёт
```

- 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
```

Выносим проверку в функцию

```c
#include <stdio.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);
}
```



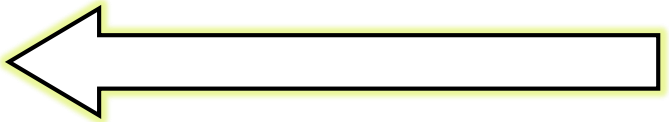
```c
void read_input(int *pa, int *pb) {
 for (;;) {
    int nitems;
    printf("input a and b: ");
    nitems = scanf("%d %d", pa, pb);
    if ((nitems == 2) && (*pb != 0))
      break;
    printf("Wrong input!\n");
    fpurge(stdin);
  }
}
```

- Функция должна быть объявлена до использования и определена где-то в коде
- Часто объявления функций выносят в специальный заголовочный файл
- Например объявления для printf и scanf находятся в stdio.h

Первый алгоритм

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

Алгоритм E

- a \ b ⇔ b = ac
- a = gcd(x, y) ⇔ a \ x и a \ y и a наибольшее из таких чисел
- Задача: найти gcd(x, y)

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

- Проверьте это под gdb
- Посчитайте gcd(1769, 427)

```
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)
```

Типы данных

- Пока что на слайдах выше 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 значений, которые могут быть в нём сохранены (value type)

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

- int это определение всех операций, которые могут быть над ним произведены (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);
```

Обсуждение

- Когда начинается наше время?

Наше время

- 0 секунд было 1 января 1970 года (Unix time)
- Диапазон от $-(2^{31}-1)$ до $(2^{31}-1)$ сек
- То есть от 13 декабря 1901 года до 19 января 2038 года
- Но **почему** это так?

Наше время

- 0 секунд было 1 января 1970 года (Unix time)
- Диапазон от $-(2^{31}-1)$ до $(2^{31}-1)$ сек
- То есть от 13 декабря 1901 года до 19 января 2038 года
- Но **почему** это так?
- Дело в том, что в машине PDP-11 для хранения времени был выбран тип, помещающий в 4 байта

Типы данных

- `char, short, int, long, long long`
- `1 = sizeof(char) <= .... <= sizeof(long long)`
- `CHAR_BIT >= 8 bit`
- Диапазон от $-(2^{x-1}-1)$ до $2^{x-1}-1$, где $x = sizeof(T) * CHAR\_BIT$
- Все эти типы имеют беззнаковых двойников:
  `unsigned char, unsigned short, unsigned int,`
  `unsigned long, unsigned long long`
- Обычно `sizeof(T) == sizeof(unsigned T)`
- Диапазон от 0 до $2^x - 1$, где $x = sizeof(T) * CHAR_{BIT}$
- `unsigned int` принято записывать просто как `unsigned`

Упражнения с двоичными числами

- В языке C используются префиксы `0` и `0x` для 8-битных и 16-битных чисел. Также в этих лекциях будет использован нестандартный суффикс `0b` для двоичных чисел.
- Верно ли следующее (подсчитайте без использования компьютера)

1. `assert (0xA0 == 0b10100000); // ?`
2. `assert (075 == 0b111101);    // ?`
3. `assert (075 == 0x3E);        // ?`
4. `assert (0xA0 == 0240);       // ?`

- Назовите минимальные типы данных для перечисленных чисел?

Перевод в системы счисления

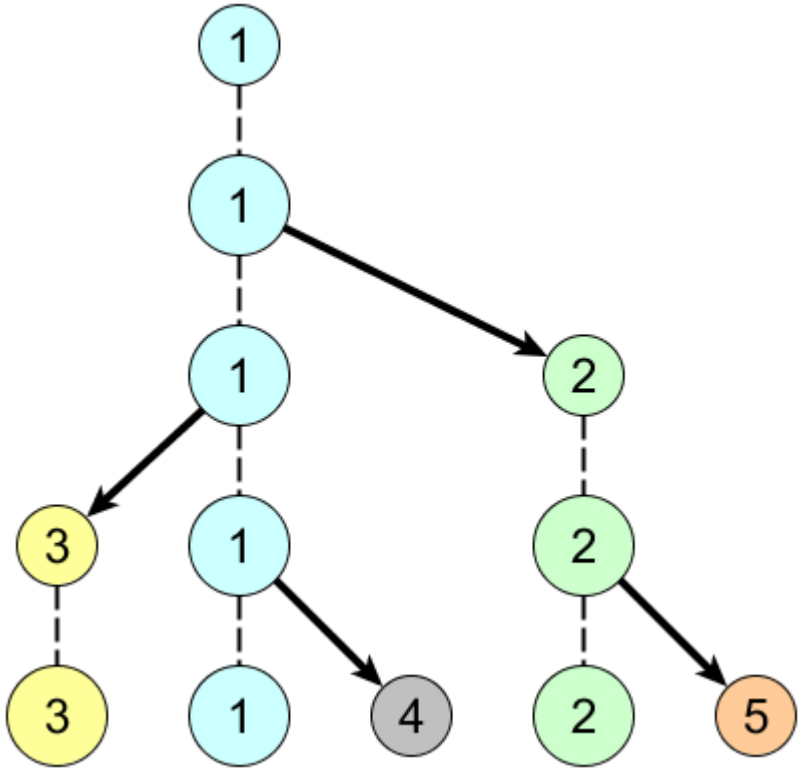
- Можно использовать деление с остатком, чтобы напечатать число в системе по основанию N для небольшого N
- $(12623)_{10} = (11000101001111)_2 = (122022112)_3 = (3011033)_4 = (400443)_5$

```
int print_converted(unsigned n, unsigned base) {
  assert (base > 1);
  while (n > 0) {
    printf("%d", n % base);
    n = n / base;
  }
}
```

- Процедура `print_converted` не вполне хороша: число выводится перевёрнутым

Warmup: 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(5), fib(10), fib(20), fib(50)`
- 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(5)`, `fib(10)`, `fib(20)`, `fib(50)`

Problem RL: рекурсия в цикл

- Ранее мы встречались с рекурсивной реализацией алгоритма Евклида

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

- Задача: написать то же самое без рекурсии, с явным циклом

Переполнения

- Подсчитайте `fib(90)` … `fib(100)` с помощью алгоритма F
- Что на экране?

Переполнения

- Подсчитайте `fib(90)` … `fib(100)` с помощью алгоритма F
- Что на экране?

```
fib(90) = 2880067194370816120
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 19740274219868223167. Давайте подсчитаем их в шестнадцатеричном виде.

Переполнения

- Подсчитайте `fib(90)` … `fib(100)` с помощью алгоритма F в hex-виде
- Что на экране?

```
fib(90) = 27f80ddaa1ba7878
fib(91) = 40abcfb3c0325745
fib(92) = 68a3dd8e61eccfbd
fib(93) = a94fad42221f2702
fib(94) = 11f38ad0840bf6bf // oops
```

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

Математика по модулю

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

```
(a + b) mod m == ((a mod m) + (b mod m)) mod m
(a * b) mod m == ((a mod m) * (b mod m)) mod m
```

- Пример: $29 * 17\ (3) = 1 * 2\ (3) = 2\ (3)$

Пример: в степень по модулю

- Возведение чисел в большие степени по небольшому модулю широко используется в криптографии
- Наивная реализация на C:

```
// returns n^k (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;
}
```

Алгоритм POWM

- См. также [$TAOCP\ Algorithm\ 4.6.3A$]

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

Problem FD: 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);`
- Задание: написать 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 использовав их как таблицу
- Для этого в языке существуют массивы

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 всё равно нули

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 Фибоначчии
- Возможно вы захотите мемоизировать числа Фибоначчи чтобы быстро искать ближайшее число Фибоначчи (как `28657` для `30987`)

https://en.wikipedia.org/wiki/Fibonacci_coding

http://www.maths.surrey.ac.uk/hosted-sites/R.Knott/Fibonacci/fibrep.html

Месть рекурсии

- Рекуррентные решения могут быть довольно красивыми
- Верна формула $F_{n+k} = F_k F_{n+1} + F_{k-1} F_n$ $[KGP\ 6.108]$
- Из неё следуют (при $k = n$ и $k = n + 1$) рекуррентные соотношения

$$F_{2n} = F_n F_{n+1} + F_{n-1} F_n = F_n (2F_{n-1} + F_n)$$

$$F_{2n+1} = {F_{n+1}}^2 + {F_n}^2$$

- Значит на каждом шаге рекурсии в зависимости от чётного или нечётного n, можно уменьшать размер задачи вдвое
- Так реализован алгоритм, приведённый на следующем слайде, который не менее эффективен, чем алгоритм F

Алгоритм FR

// $F_{2n} = F_n(2F_{n-1} + F_n)$

// $F_{2n+1} = {F_{n+1}}^2 + {F_n}^2$

```
unsigned long long fib(unsigned n) {
  unsigned long long fk, fkprev; unsigned k;
  if (n == 0) return 0ull;
  if (n <= 2) return 1ull;
  k = (n + (n % 2)) / 2; fk = fib(k); fkprev = fib(k - 1);
  if ((n % 2) == 0)
    return (2 * fkprev + fk) * fk;
  return fk * fk + fkprev * fkprev;
}
```

- Сравните быстродействие этого алгоритма с алгоритмом F на практике

Обсуждение

- Можно ли превзойти алгоритмы F и FR?

Обсуждение

- Можно ли превзойти алгоритмы F и FR?
- 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. $[TAOCP\ 1.2.8\ (5)]$

Problem MF: matrix Fibonacci (optional)

- Известно соотношение

$$\begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n$$

- Задание: используйте это соотношение для быстрого вычисления числа Фибоначчи
- Возможно вы захотите адаптировать алгоритм POWM для быстрого возведения в степень уже не числа, а матрицы

```
unsigned pow_matr22 (unsigned n[2][2], unsigned k);
```

- Задача не обязательная, так как, строго говоря, многомерных массивов мы ещё не проходили. Но решение легко нагуглить. Главное его при этом понять.

Соблазн плавающих чисел

- Вообще-то для чисел Фибоначчи легко вывести точную формулу

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

- Почему мы изначально не пользовались 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.

Домашнее задание HWF (optional)

- Два игрока играют в интересную игру: 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 (источник: $[TAOCP\ 1.2.8]$ задача 37)
- 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.
- Найдите связь этой игры с числами Фибоначчи и напишите играющую в неё компьютерную программу.
- Программу высылайте на email (см. первый слайд).

Описание игры и дополнительные материалы: https://en.wikipedia.org/wiki/Fibonacci_nim

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, 2011
- UCSanDiegoX: ALGS200x Algorithmic Design and Techniques, EDX course