# Задания по программированию студентам группы 0113 ММФ НГУ

Александр Геннадьевич Фенстер

2011 г.

Для получения зачёта по программированию необходимо и достаточно выполнить следующие действия:

1. На практических занятиях в терминальном классе сдать хотя бы 15 задач из перечисленных ниже.
2. Посетить все семинары и написать все «пятиминутки» на оценку не ниже «±». В случае пропуска семинара необходимо попросить у преподавателя или взять на сайте http://0113.fenster.name задание с соответствующего семинара, выполнить его и сдать.
3. Сдать устный зачёт в конце семестра (несколько задач по основным темам).

Перечисленные ниже задания должны быть сданы в виде исходного кода программы на языке C. Для компиляции программ можно использовать любой из доступных компиляторов. Для удобства список задач разделён на части, при этом задачи имеют сквозную нумерацию.

Попытки сдать чужое решение задачи не приветствуются. В случае появления сомнений в авторстве программы преподаватель может переформулировать задачу или попросить реализовать дополнительную функциональность.

В зависимости от количества сданных задач каждому студенту будет выставлена условная «оценка за семинары», которая будет влиять на итоговую (экзаменационную) оценку; оценку «отлично» нельзя получить, не сдав индивидуальное задание (№20).

# Содержание

## Часть 1. Простые программы с циклами

**Задание 1. Числа Фибоначчи.** Вычислите первые $N$ членов ряда Фибоначчи:

$$f_0 = 0,\ f_1 = 1,\ f_n = f_{n-1} + f_{n-2}.$$

Не используйте массивы и рекурсию.

**Задание 2. Алгоритм Евклида.** Для нахождения наибольшего общего делителя двух чисел удобно использовать следующий метод, называемый *алгоритмом Евклида*:

$$\text{НОД}\,(a, b) = \begin{cases} b, \text{ если } a \,\%\, b = 0; \\ \text{НОД}\,(b,\ a \,\%\, b) \text{ в противном случае,} \end{cases}$$

где $x \,\%\, y$ — остаток от деления $x$ на $y$. Реализуйте вычисление НОД двух чисел по алгоритму Евклида. Не используйте рекурсию: рекурсивная реализация будет разобрана на семинаре.

**Задание 3. Уравнение.** Пусть функция $f : \mathbb{R} \to \mathbb{R}$ и известно, что $f(x)$ строго монотонна на отрезке $[a, b]$. Найдите приближённое решение уравнения $f(x) = 0$ с точностью до $\varepsilon = 10^{-3}$ на этом отрезке или сообщите, что решения нет. Для решения задачи определите в программе функцию `float f(float x)` и переменные `a` и `b`, например, так:

```
float a = 0;
float b = 2;

float f(float x)
{
    return (x * x - 1);
}
```

В этом примере уравнение имеет вид

$$\begin{cases} x^2 - 1 = 0, \\ x \in [0, 2]; \end{cases}$$

его решение: $x = 1$.

**Не нужно делать перебор чисел от $a$ до $b$ с некоторым шагом.** Т. к. функция монотонна, можно найти корень методом деления пополам.

**Задание 4. Даты.** Вычислите количество дней между двумя датами, заданными в формате `DD/MM/YYYY`, **включая начальный и конечный день**. Сначала задана более ранняя, затем более поздняя дата. Учитывайте високосные годы: год является високосным, если его номер делится на 400 или делится на 4, но не на 100.

Пример: между датами `06/09/1983` и `01/02/2011` прошло 10011 дней.

Эту задачу можно решить путём написания формулы для подсчёта количества дней или путём простого перебора всех дней от одной до другой даты в цикле с увеличением счётчика. Для чтения даты в формате `DD/MM/YYYY` удобно написать такой вызов `scanf`:

```
scanf("%d/%d/%d", &day, &month, &year);
```

## Часть 2. Работа с массивами

В следующих задачах все массивы необходимо считывать из файлов (а не задавать в коде программы). Для ввода массива необходимо использовать цикл, читающий последовательно каждый из его элементов. Вы можете открывать файл напрямую, используя функции `fopen` или `freopen`, или считывать данные обычным способом (`scanf`), используя `<` для перенаправления ввода:

```
./program < input.txt
```

**Задание 5. Проверка свойств элементов.** Дан одномерный массив натуральных чисел размера $N$. Проверьте, все ли его элементы являются простыми числами. Код, проверяющий число на простоту, вынесите в отдельную функцию `int isprime(int a)`. Ясно, что если нашлось хотя бы одно составное число, то дальнейшие проверки производить не нужно.

**Задание 6. Умножение матриц.** Даны две матрицы $A$ и $B$ размера $N \times N$. Выведите на экран элементы матрицы $C = A \cdot B$:

$$C_{ij} = \sum_{k=1}^{n} A_{ik} \cdot B_{kj}.$$

**Частая ошибка:** многие забывают инициализировать матрицу $C$ нулями.

**Задание 7. Бинарный поиск.** Дан массив, про который известно, что его элементы упорядочены в порядке неубывания. Введите с клавиатуры несколько чисел и, используя метод деления пополам, определите, встречаются ли эти числа в массиве. Не используйте рекурсию: рекурсивная реализация будет разобрана на семинаре.

Бинарный поиск является, пожалуй, лучшим способом поиска элемента в упорядоченном массиве. В наихудшем случае для поиска элемента (или определения того, что число в массиве отсутствует) необходимо сделать всего $[\log_2 N] + 1$ сравнений.

## Часть 3. Работа со строками

В задачах из этой части можно считать, что длина всех строк ограничена (например, меньше 1000 символов). Строки необходимо считывать из файла или с клавиатуры (например, при помощи функции `fgets` или посимвольно вызовами `getchar`).

**Задание 8. Палиндром.** Проверьте, что данная строка является палиндромом, т.е. читается одинаково слева направо и справа налево (без учёта пробелов и знаков препинания). **Не используйте дополнительную строку.**

Обратите внимание, что в этой задаче нельзя использовать для чтения строки функцию `scanf` с параметром `%s`, поскольку в этом случае будет прочитана строка до первого пробела.

Вот пример строки, являющейся палиндромом:

```
Он дивен, палиндром, и ни морд, ни лап не видно...
```

**Задание 9. Системы счисления.** В системе счисления с основанием $b$ для записи чисел используются цифры 0, 1, ..., $b-1$ (если $b > 10$, используют буквы: $A = 10$, $B = 11$ и т. д.). Значение числа $\overline{a_n a_{n-1} \ldots a_1 a_0}_b$ ($a_i$ — цифры числа, индекс $_b$ показывает основание системы) вычисляется по следующей формуле:

$$\overline{a_n a_{n-1} \ldots a_1 a_0}_b = \sum_{i=0}^{n} a_i b^i.$$

Например, значением числа $14_6$ является $4 \cdot 6^0 + 1 \cdot 6^1 = 10$.

Введите с клавиатуры натуральное число $b$ ($1 < b \leqslant 16$) и строку, содержащую запись некоторого числа в системе счисления с основанием $b$. Вычислите значение этого числа в десятичной системе счисления (по сути, необходимо написать функцию, похожую на стандартную функцию `strtol`).

При вычислении числового значения символов `'0'`, ..., `'9'`, `'A'`, ..., `'F'` ни в коем случае не используйте инструкции `if` или `switch` с 16 ветками. Вычисления числового значения производите по коду символа. При этом желательно не использовать непонятные константы (48, 55 и прочие) — так называемые «magic numbers» в коде, смысл которых зачастую понятен только автору программы. Например, вместо 48 в данном случае правильнее писать `'0'`.

**Задание 10. Имя пользователя.** Информация о пользователях хранится в системе Linux в текстовом файле `/etc/passwd`. Каждая строка этого файла хранит данные об одном пользователе. Вот пример такой строки:

```
fenster:x:1000:1000:Alexander Fenster,Teacher,,:/home/fenster:/bin/bash
```

Легко заметить, что в этой строке хранится несколько полей, разделённых двоеточием. Четвёртое (считая с нуля) поле в свою очередь хранит список полей, разделённых запятыми. Напишите программу, которая запрашивает с клавиатуры логин и определяет имя этого пользователя по файлу `/etc/passwd`. Если пользователя с таким логином в файле нет, необходимо напечатать соответствующее сообщение.

## Часть 4. Алгоритмы сортировки

В трёх следующих заданиях программа должна брать входные данные из файла `input.txt`. В первой строке этого файла записано число $N$ — количество чисел, которые необходимо отсортировать (известно, что $1 \leqslant N \leqslant 100000$). Далее в файле записаны $N$ чисел типа `int`. В файл `output.txt` необходимо выдать эти $N$ чисел в порядке неубывания. Проверяться задача будет в том числе на тесте размером 100000.
Пример файла `input.txt`:

```
5
3 1 5 4 2
```

Пример файла `output.txt`:

```
1 2 3 4 5
```

**Задание 11. Простая сортировка.** Реализуйте один из следующих алгоритмов: сортировка простым выбором, сортировка простыми вставками, сортировка пузырьком.

**Задание 12. Быстрая сортировка** (обменная сортировка с разделением). Реализуйте алгоритм быстрой сортировки массива.

**Задание 13. Пирамидальная сортировка** (сортировка при помощи кучи). Реализуйте алгоритм пирамидальной сортировки массива.

Описание этих алгоритмов находится в отдельном файле.

## Часть 5. Работа со списками и деревьями

**Задание 14. Сортировка вставкой в список.** Отсортируйте последовательность чисел из файла путём вставки их в упорядоченный список (список изначально пуст). Длина последовательности заранее неизвестна.
Пример чтения чисел из файла до конца файла:

```
FILE *f = fopen("input.txt", "r");
int a;
while (fscanf(f, "%d", &a) == 1)
{
        /* do something */
}
fclose(f);
```

Функция `fscanf` в нормальном случае возвращает количество прочитанных аргументов. Если она вернула не 1, делается вывод о том, что либо достигнут конец файла, либо произошла ошибка (например, в файле записано не число), и цикл завершается.
Пример описания списка:

```
struct item
{
        int data;
        struct item *next;
};
struct item *head = NULL;
```

**Задание 15. Количество вхождений слов (со списком).** Подсчитайте, сколько раз каждое слово встречается в тексте. Текст читается из файла `input.txt`, для простоты можно считать, что словом является любая последовательность латинских букв, а всё, что не является латинской буквой, считается разделителем.

Для хранения информации о количестве вхождений каждого слова можно определить следующую структуру:

```
struct item
{
        char *word;
        int count;
        struct item *next;
};
```

**Задание 16. Количество вхождений слов (с хэш-таблицей).** *Хэш-функцией* называют функцию

$$f : K \to \{0, \ldots, N-1\},\ N \in \mathbb{N},$$

сопоставляющую каждому элементу *множества ключей* $K$ натуральное число от 0 до $N-1$. В данном примере множеством $K$ является множество всех слов из букв латинского алфавита.

При помощи хэш-функции каждому слову сопоставляется хэш-код, по которому определяется позиция в массиве из $N$ элементов. Ситуация, при которой два разных слова имеют один и тот же код, называется *коллизией*.

*Хэш-таблицей* называется массив из $N$ элементов, в $i$-м элементе которого хранится указатель на начало списка слов, хэш-код которых равен $i$.

Подсчитайте количество вхождений слов в текст, используя для хранения слов хэш-таблицу.

**Задание 17. Сортировка вставкой в дерево.** Отсортируйте последовательность чисел из файла путём вставки их в двоичное дерево поиска (дерево изначально пусто). Длина последовательности заранее неизвестна.
Пример описания дерева:

```
struct node
{
        int data;
        struct item *left, *right;
};
struct item *root = NULL;
```

**Задание 18. Количество вхождений слов (с деревом поиска).** Подсчитайте количество вхождений слов в текст, используя для хранения слов двоичное дерево поиска.
Пример описания дерева для этой задачи:

```
struct node
{
        char *word;
        int count;
        struct item *left, *right;
};
struct node *root = NULL;
```

## Часть 6. Графы

**Задание 19. Обход графа в глубину.** Прочитайте из файла список рёбер неориентированного графа и выведите номера вершин, доступных из вершины №1, используя обход в глубину. Считайте, что граф содержит не более 100 вершин (можно использовать матрицу смежности заданного заранее размера).

**Задание 20. Индивидуальное задание.** Задания будут распределены позднее (в конце апреля).