Задания по программированию студентам группы 0201 ФИТ НГУ. Первый семестр

Александр Геннадьевич Фенстер

16 февраля 2011 г.

Для получения зачёта по практическим занятиям («информатика») необходимо сдать как минимум двадцать из двадцати пяти перечисленных ниже заданий. Для получения зачёта по семинарам («программирование») необходимо посетить все семинары и написать все «пятиминутки» на оценку не ниже «±». В случае пропуска семинара или получения оценки «−» необходимо попросить у преподавателя или взять на сайте http://0201.fenster.name задание с соответствующего семинара, выполнить его и сдать.

Оба зачёта дифференцированные, т. е. в итоге по каждому из двух предметов будет выставлена оценка. На итоговую оценку по практике влияет количество сданных задач; на итоговую оценку по теории влияют результаты пятиминуток и контрольных работ. Посещайте лекции: на них могут быть проведены потоковые контрольные, результаты которых также будут влиять на оценки.

Перечисленные ниже задания должны быть сданы в виде исходного кода программы на языке C (не C++!), оформленного в соответствии с требованиями. Для компиляции программ можно использовать любой из доступных компиляторов.

Исходные коды задачи при сдаче прогоняются через антиплагиатную систему. В случае обнаружения плагиата преподаватель может не принять задачу или попросить переделать решение самостоятельно. Обнаружение плагиата будет весьма негативно влиять на итоговую оценку и на отношение преподавателя к вам.

Задания по программированию студентам группы 0201 ФИТ НГУ

Содержание

Задания по программированию студентам группы 0201 ФИТ НГУ

Требования к оформлению программ

Пожалуйста, привыкайте соблюдать следующие несложные правила с самых первых занятий.

1. В программе не должно быть глобальных переменных. Передавайте данные через параметры функций.

2. Разбивайте программу на функции. Не делайте функции слишком длинными (не влазящие в экран по вертикали функции — плохо), а инструкции — слишком глубоко вложенными (не влазящие в экран по горизонтали функции — тоже плохо).

3. Желательно (а начиная с сортировок — обязательно) разбивать программу на несколько файлов. Например, для всех сортировок функция `main` с вводом и выводом может быть одной и той же, а реализация каждого метода сортировки может находиться в отдельном файле.

4. Желательно (а начиная со второго семестра — обязательно) проверять результат стандартных функций (особенно функций, ошибка которых может привести к неверной работе или аварийному завершению программы, например, `malloc` и `fopen`).

5. Соблюдайте отступы в коде. Общее правило: закрывающая фигурная скобка должна стоять под началом инструкции, к которой она относится, а весь код внутри блока должен быть сдвинут вправо. Открывающая скобка может стоять в конце строки с инструкцией или в начале следующей строки с тем же отступом, что и инструкция.

6. Пробелы ставятся между операторами, после запятых и в прочих местах, где они радуют глаз, но не ставятся между именем функции и открывающей круглой скобкой. Пустые строки разделяют функции, а также различные фрагменты кода.

7. Не пишите более одной инструкции в одной строке.

8. Не стесняйтесь проверить свою программу при помощи `valgrind`. На факультетском сервере это делается так:

```
/opt/home/valgrind/bin/valgrind --tool=memcheck имя_программы
```

Пример бессмысленной, но допустимо оформленной программы на языке C

```
#include <stdio.h>

int sqr(int x)
{
    return (x * x);
}

int main()
{
    int a = 25;
    int b;

    printf("Enter any number: ");
    scanf("%d", &b);

    if (sqr(b) < a)
    {
        printf("Square of %d is less than %d\n", b, a);
    }
    else
    {
        printf("Square of %d is greater than "
               "or equal to %d\n", b, a);
    }

    return 0;
}
```

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 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: $x = 1$.

Задание 4. Даты. Вычислите количество дней между двумя датами, заданными в формате `DD/MM/YYYY`, включая начальный и конечный день. Даты могут быть заданы в любом порядке (сначала более ранняя, затем более поздняя или наоборот). 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 `01/09/2010` прошло 9858 дней.

Эту задачу можно решить путём написания множества условных инструкций и формул, но намного проще перебрать все дни от одной до другой даты в цикле и подсчитать количество итераций.

Часть 2. Работа с массивами

Здесь и далее фразы типа «дан массив ...», «дана строка ...» и т. п. нужно понимать как «из файла `input.txt` вводится массив, строка, ...». Для ввода массива необходимо использовать цикл, читающий последовательно каждый из его элементов.

Задание 5. Проверка свойств элементов. Дан одномерный массив натуральных чисел размера N. Проверьте, что:

а) все его элементы являются простыми числами;

б) среди его элементов есть хотя бы одно простое число.

Код, проверяющий число на простоту, вынесите в отдельную функцию `int isprime(int a)`. Заметьте, что зачастую ответ на оба вопроса известен до окончания перебора всех элементов массива. В этом случае не следует продолжать перебор.

Задание 6. Бинарный поиск. Дан массив целых чисел, 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 очень хорошим 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 с файлами

Задание 7. Копирование. Реализовать простейший вариант команды `cp` — копирование одного файла в другой:
`./cp source.txt destination.txt`
Параметры, переданные в командной строке можно узнать, описав функцию `main` особым образом: `int main(int argc, char *argv[])` Тогда в `argc` будет содержаться количество параметров, в `argv[0]` — путь к исполняемому файлу программы, а остальные элементы (`argv[1]`, ..., `argv[argc - 1]`) будут содержать параметры командной строки.

Задание 8. Количество символов, слов и строк. Реализовать аналог программы `wc`: подсчёт количества символов, слов и строк в файле. Количество слов не зависит от количества пробелов между ними! Количеством строк считайте количество вхождений символа `'\n'`.

Часть 4. Работа со строками

Задание 9. 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.** Вот пример строки, являющейся палиндромом:

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

Задание 10. Системы счисления — 1. 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 $\overline{a_n a_{n-1} \dots a_1 a_0 . a_{-1} a_{-2} \dots a_{-m}}_b$ (a_i — цифры числа, индекс $_b$ показывает основание системы) вычисляется по следующей формуле:

$$\overline{a_n a_{n-1} \dots a_1 a_0 . a_{-1} a{-}2 \dots a_{-m}}_b = \sum_{i=-m}^{n} a_i b^i.$$

Например, значением числа 14.3_6 является $3 \cdot 6^{-1} + 4 \cdot 6^0 + 1 \cdot 6^1 = 10.5$.

Введите с клавиатуры натуральное число b ($2 \leqslant b \leqslant 16$) и строку, содержащую запись некоторого вещественного числа в системе счисления с основанием b. Вычислите значение этого числа в десятичной системе счисления.

Задание 11. Системы счисления — 2. Вычислите, как будет записываться введённое с клавиатуры десятичное вещественное число в системе счисления с основанием b ($2 \leqslant b \leqslant 16$). Существует несколько способов перевода.

Способ 1. Целая часть переводится отдельно от дробной. Для перевода целой части необходимо разделить её на b. Остаток от деления будет являться последней цифрой целой части результата (помните, что для систем с основанием, большим десяти, необходимо использовать буквы A, B, ..., F). Частное необходимо снова разделить на b, остаток будет являться предпоследней цифрой целой части результата, и так далее. Деление повторять до тех пор, пока частное не будет равняться нулю, остаток в этом случае будет являться первой цифрой результата. Дробную часть необходимо умножать на b, после каждого умножения целая часть результата будет являться очередной цифрой дробной части (возможно зацикливание — например, попробуйте перевести 0.1 в двоичную систему: $0.1 = 0.0(0011)_2$).

Способ 2. Последовательно вычитайте из исходного числа степени числа b, начиная с максимальной, не превосходящей исходное число, и уменьшая степень (при необходимости делая её меньше нуля) до тех пор, пока либо исходное число не превратится в ноль, либо не будет определено, что результат — бесконечная дробь. Пример: из числа 66.5 можно 1 раз вычесть 4^3, из оставшегося числа 2.5 нельзя вычесть 4^2 и 4^1, можно 2 раза вычесть 4^0 и из оставшегося числа 0.5 можно два раза вычесть 4^{-1}; получаем, что $66.5 = 1002.2_4$.

Идеальная программа может быть настолько умной, что будет сама определять период дроби.

Для получения символа, соответствующего данному числу ($0 \to$ `'0'`, $1 \to$ `'1'`, ..., $15 \to$ `'F'`), не следует писать много условных конструкций. Эта операция легко делается путём добавления некоторого числа к символу `'0'` или `'A'`.

Задание 12. 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`.

Задание 13. Разбор формата CSV. CSV (comma-separated values) — популярный (но не самый удобный) формат хранения табличных данных. Файл в формате CSV состоит из строк, в каждой из которых хранятся данные из нескольких ячеек, разделённые запятыми, например, так:

```
Фамилия,Имя,Отчество
Иванов,Иван,Петрович
Петров,Василий,Сергеевич
```

Проблемы возникают, если ячейка таблицы содержит запятую или перевод строки. В таком случае содержимое ячейки заключается в двойные кавычки. Если же ячейка содержит двойную кавычку, эта кавычка дублируется. В следующем примере приведена таблица из трёх строк и трёх столбцов (во второй строке третья ячейка пустая):

```
следующая ячейка содержит перевод строки,"до перевода строки
после перевода строки",а это третий столбец
следующая ячейка содержит кавычку и запятую,"вот они: "","
в этой строке вторая и третья ячейка содержат по одной кавычке,""""","""""
```

При форматировании таблицы в языке разметки HTML таблица начинается тегом `<table>` и заканчивается тегом `</table>`. Строка таблицы начинается тегом `<tr>` и заканчивается тегом `</tr>` (table row). Ячейка таблицы начинается тегом `<td>` и заканчивается тегом `</td>` (table detail). Перевод строки обозначается тегом `<br/>` (break).

Прочитайте из файла таблицу в формате CSV и выведите в файл таблицу в формате HTML. Для простоты считайте, что таблица не содержит символов < и >.

Если бы эти символы допускались, необходимо было бы заменять < на последовательность `<` (less than), а > на `>` (greater than). Символ `&`, который может встретиться в тексте, при этом необходимо заменять на `&`.

Часть 5. Рекурсивные алгоритмы

Задание 14. Перестановки. Напечатайте все перестановки элементов множества $\{1, \ldots, N\}$, используя рекурсивный алгоритм генерации перестановок: на каждом этапе функции передаётся уже построенное начало перестановки и набор ещё не использованных элементов и она поочерёдно добавляет каждый из неиспользованных элементов в конец перестановки и выполняет рекурсивный вызов.

Часть 6. Алгоритмы сортировки

В двух 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. **Пожалуйста, не пытайтесь сдать программу, которая не читает данные из файла или имеет ограничение на размер массива меньше 100000 элементов.**
Пример файла `input.txt`:

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

Пример файла `output.txt`:

```
1 2 3 4 5
```

Задание 15. Простая сортировка. Реализуйте любой простой алгоритм сортировки, кроме пузырьковой (например, простой выбор или простые вставки).

Задание 16. Быстрая сортировка (обменная сортировка с разделением). Реализуйте алгоритм быстрой сортировки массива.

Задание 17. Пирамидальная сортировка (сортировка при помощи кучи). Реализуйте алгоритм пирамидальной сортировки массива.

С подробным описанием алгоритмов быстрой и пирамидальной сортировки можно ознакомиться в отдельном файле.

Часть 7. Динамические структуры данных

Задание 18. 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 19. Количество вхождений слов в текст. 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. Максимальная длина слова равна 10 символам.

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

```
struct item {
        char word[11];
        int count;
        struct item *next;
};
```

Задание 20. Хэш-таблицы. *Хэш-функцией* называется функция

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

сопоставляющая 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 21. 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 item {
        int data;
        struct item *left, *right;
};
struct item *root = NULL;
```

Часть 8. Графы

Задание 22. Обход 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 23. Обход в ширину. Прочитайте из файла список рёбер неориентированного графа и выведите номера вершин, доступных из вершины №1, используя обход в ширину. Считайте, что граф содержит не более 100 вершин (можно использовать матрицу смежности заданного заранее размера).

Задание 24. Алгоритм Дейкстры. Прочитайте из файла список рёбер неориентированного взвешенного графа и номер начальной вершины, выполните алгоритм Дейкстры и выведите для каждой вершины графа длину кратчайшего пути от начальной вершины до неё и вершины, через которые этот путь проходит.

Задание 25. Полный перебор. Дан полный граф из N вершин: вершине i соответствует точка на плоскости с координатами (x_i, y_i), длина ребра (i, j) равна расстоянию между соответствующими точками:

$$d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}.$$

Решите задачу коммивояжёра для этого графа полным перебором всех гамильтоновых циклов: полезно хотя бы раз в жизни написать действительно полный перебор.