Лабораторная работа 4

Функции

Цель этой лабораторной работы — изучить понятие функции в C++ и научиться объявлять, определять и вызывать функции, а также научиться хорошо тестировать свои собственные программы!

(!) Обязательно скачиваете папку с cpp-файлами из Яндекс-диска курса! Там уже лежат мои тесты для задач, а также места для ваших собственных тестов.

(!!) Когда делаете свои unit-тесты, вы должны придумать их независимо от своей программы. Нельзя генерировать программой тесты для проверки самой же программы!

Задания A

1. Напишите функцию возведения целого числа в целую неотрицательную степень

   `long long power(long long x, unsigned k)`, которая вычисляет x^k.

   Надо unit-тестировать функцию `power`.

2. С помощью написанной в предыдущей задаче функции `power` (ее надо скопировать в вашу программу), напишите функцию

   `long long sum_p(int p, int n)`, которая вычисляет сумму p-х степеней чисел $1, 2, ..., n$, т.е. вычисляет $1^p + 2^p + ... + n^p$.

   Надо unit-тестировать функцию `sum_p`.

3. Напишите функцию

   `double dist(double x1, double y1, double x2, double y2)`,

   которая принимает вещественные декартовы координаты двух точек на плоскости и возвращает расстояние между ними.

   Напишите с ее помощью программу, которая вводит вещественные координаты x_1, y_1, x_2, y_2, x_3, y_3 трех точек на плоскости и выводит длину самой длинной стороны треугольника с вершинами в этих точках, или –1, если такого треугольника не существует.

   Пример.

   Ввод: `0 0 6 0 3 2` Вывод: `6`

   Сделайте функцию `solve`, которая принимает шесть аргументов (координаты) и возвращает одно вещественное число — ответ на задачу. Функция `main` отвечает только за ввод-вывод и вызов функции `solve`.

   Надо unit-тестировать обе функции `dist` и `solve`.

4. Напишите функцию `void sort_by_last(int &a, int &b, int &c)`, которая сортирует *три* натуральных числа по их последним цифрам.

   Надо unit-тестировать только функцию `sort_by_last`.

   Пример.

   Ввод: `138 2647 36971` Вывод: `36971 2647 138`

5. Напишите функцию `bool perfect(int n)`, которая принимает целое число n и проверяет, является ли оно *совершенным числом*. Совершенное число — число, равное сумме всех своих делителей.

   Напишите с ее помощью программу, которая вводит целые числа $0 < M \leqslant N$ и выводит все совершенные числа на диапазоне $[M, N]$ в порядке возрастания.

   Пример.

   Ввод: `3 500` Вывод: `6 28 496`

   Надо unit-тестировать только функцию `perfect`.

Задания B

1. Напишите функцию `bool hamming(int n)`, которая принимает целое число n и проверяет, является ли оно *числом Хэмминга*. Число Хэмминга — число, не имеющее других простых делителей, кроме 2, 3 или 5.

   Напишите с ее помощью программу, которая вводит целые числа $0 < M \leqslant N$ и выводит все числа Хэмминга на диапазоне $[M, N]$ в порядке возрастания.

   Пример.

   Ввод: `9 20` Вывод: `9 10 12 15 16 18 20`

   Надо unit-тестировать только функцию `hamming`.

2. Возьмем любое натуральное число. Если оно четное — разделим его пополам, если нечетное — умножим на 3, прибавим 1 и разделим пополам. Повторим эти действия с вновь полученным числом. *Гипотеза Сиракуз* гласит, что независимо от выбора первого числа рано или поздно мы получим 1.

   Напишите функцию `int Syracuse(int n)`, которая проверяет, за сколько шагов число n превратится в единичку вышеуказанным алгоритмом (одним шагом считайте одно полное превращение числа: разделение пополам ИЛИ (умножение на 3 + прибавление 1 + разделение пополам).

   Напишите с ее помощью программу, которая вводит целые числа $0 < M \leqslant N$ и выводит количество шагов Гипотезы Сиракуз для каждого числа на диапазоне $[M, N]$.

   Надо unit-тестировать только функцию `Syracuse`.

3. *Цифровой корень* натурального числа получается следующим образом: складываем все цифры данного числа — получаем новое число. Повторяем процесс, пока в результате не будет получено однозначное число, которое и называется цифровым корнем числа.

   Напишите функцию `void digit_root(long long &n)`, которое превращает n в свой цифровой корень.

   Надо unit-тестировать функцию `digit_root`.

4. Напишите функцию `int my_gcd(int a, int b)`, которая находит НОД (наибольший общий делитель) чисел `a` и `b`. Используйте реализацию алгоритма Евклида через цикл `while`.

   Напишите функцию `void simplify(int &num, int &denom)`, которая сокращает дробь $\frac{num}{denom}$, используя написанную вами функцию `my_gcd`. Также напишите функцию `main`, чтобы она делала ввод числителя и знаменателя, и выводила числитель и знаменатель сокращенной дроби. А само сокращение должно делаться через вашу функцию.

   Надо unit-тестировать обе функции `my_gcd` и `simplify`.

5. Напишите функцию `void intersect(int a, int b, int c, int d, int &l, int &r)`, которая находит пересечение $[l, r]$ отрезков $[a, b]$ и $[c, d]$. Если эти отрезки не пересекаются, то ответом сделайте $l = 0, r = -1$. Гарантируется, что $a \leq b$ и $c \leq d$.

   Надо unit-тестировать функцию `intersect`. Все случаи взаимного расположения отрезков $[a, b]$ и $[c, d]$ надо тщательно тестировать!