Задача 2. Арифметика по модулю

Источник:	базовая II
Имя входного файла:	--
Имя выходного файла:	--
Ограничение по времени:	5 секунд
Ограничение по памяти:	разумное

Внимание: эта задача проверяется на **emailtester**.

В этой задаче нужно реализовать арифметику в поле вычетов (т.е. арифметику по простому модулю). Она должна быть объявлена в файле `modular.h` и реализована в файле `modular.c`. Кроме того, в файле `main.c` нужно написать код для тестирования этой арифметики.

Вот простейший пример использования арифметики:

```
#include "modular.h"     //в этом хедере должны быть объявления
#include <assert.h>

int main() {
    MOD = 13;            //устанавливаем глобальный модуль
    int a = 45;
    a = pnorm(a);
    assert(a == 6);
    int x = pmul(padd(7, psub(2, 3)), 5);
    assert(x == 4);
    int y = pdiv(7, x);
    assert(pmul(x, y) == 7);
    MOD = 2;             //меняем модуль на другой
    assert(pnorm(5) == 1);
    return 0;
}
```

Модуль должен храниться в глобальной переменной `MOD`, которая должна иметь тип `int`. В эту переменную записывается модуль в самом начале работы, кроме того, его можно переприсваивать сколько угодно раз в дальнейшем. Модуль должен быть простым числом не более 10^9 — другие значения устанавливать нельзя. Переменная `MOD` должна быть объявлена в `modular.h` и определена в `modular.c`.

В модульной арифметике должно быть 5 функций:

1. Функция `pnorm`: принимает одно значение типа `int`, возвращает `int`. Функция возвращает остаток от деления переданного аргумента по текущему модулю. Входное число по абсолютной величине не превышает 10^9, может быть отрицательным. Выходное значение должно быть в диапазоне от 0 до `MOD` - 1.

2. Функции `padd`, `psub`, `pmul`, `pdiv`: каждая принимает два параметра типа `int`, возвращает `int`. Они реализуют сложение, вычитание, умножение и деление соответственно в поле вычетов по модулю текущего `MOD`. Значения аргументов и выходное значение должны быть в диапазоне от 0 до `MOD` - 1.

Все эти функции должны быть объявлены в `modular.h` и определены в `modular.c`.

Для тестирования написанного кода следует использовать файл `main.c`, где нужно определить точку входа `main`. В функции `main` следует написать какой-то код, чтобы убедиться, что написанный в `modular` код работает правильно. В этом коде для проверки условий используйте встроенную функцию `assert` (в системе тестирования они **не** удаляются). По сути, приведённый выше кусок кода является хорошим примером содержимого `main.c`, только проверок лучше добавить побольше.

При сборке воедино программы из `main.c`, `modular.h` и `modular.c` должен получаться исполняемый файл, который при выполнении запускает ваши тесты, т.е. проверяет правильность кода `modular.c`. Вам нужно отправить в систему тестирования все эти три файла. В системе часть файлов будет заменяться на файлы жюри, в частности:

1. Будет проверяться ваша реализация `modular` с помощью файла `main.c` от «жюри».
2. Будет проверяться, что ваши тесты в `main.c` отлавливают простейшие ошибки в `modular` — то есть файлы `modular` будут подменяться на неправильно работающие.

Гарантируется, что тестирующий код жюри использует ваши функции и переменные корректно, согласно описанным выше условиям и соглашениям. Аналогично, ваш тестирующий код должен также соблюдать все эти условия.