**Лабораторная работа №2**

Любой рефакторинг подразумевает покрытие исходного кода тестами, чтобы исключить возможную потерю функциональности в процессе. Если переписываемая программа не содержит ветвлений (операторы switch и if) или не принимает входные данные, то тестировать ее не нужно, любая же вариативность исполнения, по закону Мерфи, подразумевает, что что-то пойдет не так.

Видов тестирования существует великое множество, как их классификаций. На само высоком уровне тестирование можно разделить по уровню проведения. Модульное тестирование (unit-тестирование) имеет своей целью проверить работу отдельных модулей программы перед их встраиванием в проект. Интеграционное тестирование проводится для выявления нарушений при взаимодействии модулей внутри программы после их прогонки через предыдущий этап. Системное тестирование выявляет не только функциональные ошибки, но также и несовместимость с окружением, предоставление прав доступа пользователям, не имеющим соответствующий доступ, и т.д. Операционное тестирование предусматривает проверку выполнения системой своей функции в целом и наличия конфликтов с другими программами, участвующими в бизнес-процессе. Наконец, приемочное тестирование проводится заказчиком для установления соответствия между требованиями, прописанными в ТЗ, и итоговым функционалом.

Основное внимание в данной лабораторной будет уделено модульному тестированию, так как оно является основой для принятия решения о качестве проводимого рефакторинга. Тесты, написанные для исходной системы, должны обрабатываться результирующей. Если тестов для исходной системы не существует (<sarcasm>разве такое когда-нибудь бывает? <\sarcasm>), то их необходимо написать перед проведением рефакторинга.

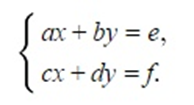
Основным критерием оценки тестов для системы является процент покрытия программного кода. Стремиться необходимо к полному покрытию, чтобы каждой строке Вашей программы соответствовал тест, проверяющий ее работу. Но это справедливо для «сферических цыплят в вакууме». В коде существуют как нефункциональные строки, так и строки, работа которых не зависит от вводимых пользователем данных. Кроме того, к тестам обычно предъявляется требование модульности, т.е. один тест должен проверять одно конкретное состояние системы.

Что должно покрываться тестами:

* ввод данных пользователем (ввод символа вместо цифры, ввод дробного числа вместо целого и т.д.);
* ветвления (каждая ветвь каждого оператора ветвления должна иметь собственный тест, проверяющий ветвь на достижимость и правильную обработку);
* циклы (тестировать следует несколько итераций. Обычно берут нулевую и первую, так как если они проходят нормально, то считается, что в дальнейшем проблем возникнуть не должно. Кроме того, следует тестировать условия выхода из цикла).

**Задача, решаемая в коде:**

Даны вещественные числа a, b, c, d, e, f. Решите систему линейных уравнений



**Входные данные:**

Шесть чисел - коэффициенты уравнений системы.

**Выходные данные:**

Если система не имеет решений, то программа должна вывести единственное число 0.

Если система имеет бесконечно много решений, каждое из которых имеет вид y=kx+n, то программа должна вывести число 1, а затем значения k и n.

Если система имеет единственное решение (x0, y0), то программа должна вывести число 2, а затем значения x0 и y0.

Если система имеет бесконечно много решений вида x=x0, y — любое, то программа должна вывести число 3, а затем значение x0.

Если система имеет бесконечно много решений вида y=y0, x — любое, то программа должна вывести число 4, а затем значение y0.

Если любая пара чисел (x, y) является решением, то программа должна вывести число 5.

**Код программы:**

#include <iostream>

#include <cmath>

using namespace std;

int main()

{

double a, b, c, d, e, f;

cin >> a >> b >> c >> d >> e >> f;

if ((a == 0) && (b == 0) && (c == 0) && (d == 0) && (e == 0) && (f == 0))

{

cout << '5';

}

else if ((a\*d - c \* b != 0) && ((e\*d - b \* f != 0) || (a\*f - c \* e != 0)))

{

double y = (a \* f - c \* e) / (a \* d - c \* b);

double x = (d \* e - b \* f) / (d \* a - b \* c);

cout << "2 " << x << ' ' << y;

}

else if (((a\*d - c \* b == 0) && ((e\*d - b \* f != 0) || (a\*f - c \* e != 0))) ||

(a == 0 && c == 0 && e / b != f / d) ||

(b == 0 && d == 0 && e / a != f / c) ||

(a == 0 && b == 0 && c == 0 && d == 0 && (e / f > 0)))

{

if (((a == 0 && b == 0 && e == 0 && d != 0 && c == 0) ||

(c == 0 && d == 0 && f == 0 && b != 0 && a == 0)))

{

double y;

if (b == 0)

y = f / d;

else if (d == 0)

y = e / b;

else if (e == 0 || f == 0)

y = 0;

cout << '4' << ' ' << y;

}

else if (((a == 0 && b == 0 && e == 0 && c != 0 && d == 0) ||

(c == 0 && d == 0 && f == 0 && a != 0 && b == 0)))

{

double x;

if (a == 0)

x = f / c;

else if (c == 0)

x = e / a;

else if (e == 0 || f == 0)

x = 0;

cout << '3' << ' ' << x;

}

else

cout << '0';

}

else if (a == 0 && c == 0)

{

double y;

if (e == 0)

y = f / d;

else if (f == 0)

y = e / b;

else

y = e / b;

cout << '4' << ' ' << y;

}

else if (b == 0 && d == 0)

{

double x;

if (e == 0)

x = f / c;

else if (f == 0)

x = e / a;

else

x = e / a;

cout << '3' << ' ' << x;

}

else if (b == 0 && e == 0)

{

double k, n;

k = -c / d;

n = f / d;

cout << '1' << ' ' << k << ' ' << n;

}

else if (d == 0 && f == 0)

{

double k, n;

k = -a / b;

n = e / b;

cout << '1' << ' ' << k << ' ' << n;

}

else if (a == 0 && e == 0)

{

double k, n;

k = -d / c;

n = f / c;

cout << '1' << ' ' << k << ' ' << n;

}

else if (c == 0 && f == 0)

{

double k, n;

k = -b / a;

n = e / a;

cout << '1' << ' ' << k << ' ' << n;

}

else if ((a / b == c / d))

{

double k, n;

k = -c / d;

n = f / d;

cout << '1' << ' ' << k << ' ' << n;

}

else

{

cout << "Are you kidding me?";

}

return 0;

}

**Задание:**

Написать тесты для приведенной программы, которые обеспечат полное покрытие ветвлений.