Задание 5.
Статический анализ кода + написание тестов (10 баллов).

Формулировка задания

1. Статический анализ кода

Дано:

Дан текст 2 программ на языке C. Необходимо провести статический анализ данных программ с помощью средств статического анализа кода.

Требуется:
1. **(2 балла)** Ознакомиться с возможностями средства Microsoft Visual Studio:
- создать проект консольного приложения с данным файлом;
- запуск анализа выполняется из меню Analyze->Run Code Analysis.

Выполнить анализ первого примера, проверить сообщения о найденных дефектах, определить какие из дефектов являются ложными:
- для одного двух ложных дефектов сделать предположения о причине их обнаружения;
- подсчитать число ложных дефектов, определить точность анализа.

Выполнить анализ второго примера, проанализировать исходный код примера вручную и найти дефекты, необнаруженные средством:
- сделать предположение о причине необнаружения этих дефектов;
- оценить полноту полученных результатов.

2. **(2 балла)** Выполнить шаги пункта 1 для любого из свободных статических анализаторов (BLAST, lint, PVS-Studio, CPP-Checker, …)

3. **(1 балл)** Выполнить анализ второго примера с помощью средства Aegis(http://aegis-demo.digiteklabs.ru/s2a.webserver/). Сравнить полученные результаты, с результатами Microsoft Visual Studio и свободного анализатора в виде небольшого эссе.

2. Регрессионное тестирование

Дано:

Дан текст программы на языке C, содержащий функции f, g, k, и t.

Требуется:

4. **(2 балла)** Написать набор тестов, которые будут проверять отсутствие внесенных ошибок (регрессии). Данные реализации функций считаются правильными, то есть они должны проходить тесты. Тесты будут проверяться на закрытой базе из версий программы с искусственно внесенными ошибками. Требуется, чтобы тесты давали 100% покрытие кода (покрытие проверяется с помощью утилиты *gcov*), а также обнаруживали все внесенные ошибки. Формат

5. **(1 балл)** Научиться пользоваться средством анализа покрытия кода *gcov*. Убедиться на своей локальной машине, что написанные тесты дают полное покрытие кода.

6. **(2 балла)** Написать набор, состоящий из не более, чем **14** тестов, который будет обнаруживать все внесенные ошибки и давать полное покрытие кода.

Формат файла с тестами:

1. Предлагается два варианта написания тестов (на выбор студента): на языке Си или на языке Python.

2. Тесты на языке Си должны представлять собой программу на языке Си, которая успешно (с кодом завершения 0) завершается при успешном прохождении всех тестов, и не успешно иначе. Тесты будут собраны вместе с тестируемой программой, таким образом в коде тестов могут использоваться тестируемые функции и должны быть указаны их прототипы, но не должно быть их реализаций. Другими словами, для запуска тесты будут скомпилированы командой, аналогичной следующей:
   ```
   $ gcc prog.c tests.c -o testing
   ```

3. Тесты на я/п Python должны представлять собой модуль Python, содержащий набор тестов, написанных с использованием фреймворка `unittest`. Для вызова функций, написанных на Си, предлагается использовать модуль `cffi`. При этом для функций из первого задания, принимающих на вход два целых числа и возвращающих целое число, переход от оригинальной функции к используемой питоновской обертке полностью прозрачен (питоновские целые числа могут быть использованы как аргументы и получены в качестве возвращаемых значений). При работе с другими типами данных (требующей работы с участками динамической памяти) потребуется использование интерфейсов модуля `cffi` для перевода соотвествующих значений из питоновских типов в представление, используемое программой на Си. При написании тестов можно считать, что существует модуль prog1, содержащий для каждой тестируемой функции одноименную cffi-обертку. Тестирование будет считаться проваленным как при провале одной из тестовых проверок, так и (аналогично тестам на Си) при падении (аварийном завершений) тестирующей программы;

4. Примеры тестовых программ обоих типов могут быть найдены ниже;

5. **Программа должна быть написана самостоятельно.** При заимствовании фрагментов кода из open-source проектов должен быть указан автор кода. Заимствовать код из решений предыдущих лет не допускается. Если будет обнаружено, что несколько присланных решений написаны одним человеком, оценка будет снижена для всех похожих работ. Для анализа схожести кода будут использованы соответствующие инструментальные средства.

Документация:

- Инструмент для оценки покрытия кода gcov: https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
- Фреймворк для написания юнит-тестов на Python unittest: https://docs.python.org/2/library/unittest.html
- Модуль для вызова внешних функций C Foreign Function Interface (CFFI): http://cffi.readthedocs.io/en/latest/

Требования к файлам решения:

Решение задачи должно включать в себя 9 файлов:

1. Файл исходных кодов анализируемых программ year_group_surname_false_positives_sample.c
2. Файл исходных кодов анализируемых программ year_group_surname_undetect_defect_sample.c

3. Текстовый файл year_group_surname_false_positives.txt с описанием найденных дефектов первого примера, предположений о причине обнаружения ложных дефектов и оценкой точности анализа в Microsoft Visual Studio и выбранном анализаторе.
4. Текстовый файл year_group_surname_undetect_defect.txt с описанием дефектов второго примера необнаруженных средствами, предположений о причине необнаружения дефектов и оценкой полноты анализа.
5. Текстовый файл year_group_surname_zcomparision.txt с описанием выводов о сравнении возможностей средства Microsoft Visual Studio, выбранного анализатора и сравнением полученных результатов анализа второго примера со средством Aegis.
6. Файл исходной программы year_group_surname_prog.c.
7. Файл year_group_surname_ prog_tests.c или year_group_surname_ prog_tests.py с тестами (его формат описан выше)
8. Файл year_group_surname_prog_tests.c.gcov лога gcov
9. Файл с текстом эссе на тему “Как я провёл Практикум в 7м семестре”. (файл year_group_surname_esse.txt)

Методические указания по выполнению задания:

1. **Пример тестов для первого задания на языке Си**

```
#include "stdio.h"
#include "assert.h"

int f(int a, int b);
int g(int a, int b);
int k(int a, int b);
int t(int a, int b);

void test 1() {
    assert(f(5, 6) == 9);
}

void test 2() {
    assert(g(5, -10) == 4);
}

void test() {
    test 1();
    test 2();

}

int main() {
    test();
    return 0;
}
```

2. **Пример тестов для первого задания на языке Python**

```
from unittest import TestCase

from prog1 import f, g, k, t

class TestProg1(TestCase):
    def test 0(self):
        self.assertEqual(f(5, 6), 9)

    def test 1(self):
        self.assertEqual(g(5, -10), 4)
```