Лабораторные работы по курсу “Тестирование и Отладка ПО (программа “Второе высшее”)

Лабораторная Работа №1

Тема: Unit или Модульное тестирование

Задание:

1. Написать набор unit-тестов для лабораторных работ по курсу “Типы и структуры данных”
2. Можно использовать любой язык программирования знакомый студенту, если нет предпочтений, то использовать тот же язык программирования, что для курса “Типы и структуры данных”

Требования:

1. Настроить запуск прогона тестов из командной строки
2. При прогоне отчет должен быть сгенерен в виде файла, например в формате .csv с заголовком (название теста, ожидаемый результата, полученный результат, описание ошибки если есть, время на выполнение теста)
3. Если язык программирования поддерживается в https://github.com/allure-framework, то отчет генерить в нем (c/c++ не поддерживается)
4. Требуется придерживаться паттерна AAA (Arrange, Act, Assert) при создании тестов
5. Рекомендации по структуре: например если есть файл с набором функций для реализации какой-то структуры или алгоритма, то должен быть соответствующий отдельный файл с тестами
6. Должны быть представлены как позитивные (ожидаемый результат не является ошибкой или исключением) так и негативные сценарии (входные данные приводят к ожидаемой ошибке или исключению); негативные тесты обычно выносят в отдельные файлы от позитивных
7. Unit-тесты должны выполняться в любом порядке и не влиять на результаты выполнения друг друга – в идеале каждый unit тест является “чистой функцией”

Лабораторная Работа №2

Тема: Нефункциональное тестирование на примере benchmark-теста (сравнительное тестирование производительности) разных алгоритмов

Задание:

1. Реализовать две функции для определения является ли введенное число простым; простое число – это натуральное (положительное целое) число, которое делится без остатка только на себя и единицу
2. Написать набор тестов, который позволяет сравнить два алгоритма по времени выполнения (также можно оценить потребляемую память и загрузку процессора процессом, в котором запущен тест, хотя это и не будет столь показательно как время выполнения)

Требования:

1. Максимальное анализируемое число: 2^{64} -1 (не является простым, число из мифической задачи про “зерна на шахматной доске”, точное значение 18446744073709551615, максимальное целое в uint64 из стандартной библиотеки C, можно проверить в limits.h, простые делители – 3, 5, 17, 257, 641, 65537, 6700417)
2. Первый алгоритм: последовательный перебор делителей (самый не эффективный)
3. Второй алгоритм: любой оптимизированный алгоритм не для частных категорий чисел, например реализовать тест Миллера
4. Структура тестов разрабатываемого benchmark:
    a. проверить на простоту все числа от 1 до 65537 (2^{16}+1)
    b. проверить на простоту все числа вида (2^{N}-1) и (2^{N}+1) где N принадлежит отрезку [16; 32]
    c. проверить любое случайное число от 2^{32} до 2^{64}-1 (требуется функция генерации достаточно большого псевдослучайного числа)
5. Для первых двух тестов составить графики зависимости времени выполнения от возрастания проверяемого числа

Лабораторная Работа №3

Тема: Интеграционное тестирование и разработка на основе тестирования (Test Driven Development, TDD)

Задание

1. Реализовать простую программу для игры в “Морской бой”
2. Написать интеграционный сквозной (E2E) тест для проверки работоспособности игры
3. Опционально: могут быть реализованы отдельные unit-тесты, если студент считает, что они упростят его работу
4. Можно использовать любой язык программирования, который знаком студенту

Требования к минимальной реализации игры

1. Предусмотрен только один формат игры: поле 10×10 клеток, корабли 1×4 клеток + 2×3 клеток + 3×2 клеток + 4×1 клеток, в игре нет мин, корабли не могут быть расположены вплотную друг к другу
2. Консольное приложение с двумя пунктами меню на начальном экране: “Начать новую игру”, “Продолжить игру”
3. При выборе “Начать новую игру”:
    a. происходит загрузка двух файлов из папки с исходным кодом программы или любой на усмотрение студента, каждый файл содержит начальное расположение кораблей одного из игроков (1 – ячейка корабля, 0 – пустая ячейка игрового поля), например так (подкраска для удобства отображения в задании):
    1111000000
    0000011101
    1110000000
    0000010110
    0101000000
    0101000010
    0000000000
    0000000000
    0000000000
    0000010000
    для удобства файлы можно называть просто newgame_player_1.cfg и newgame_player_2.cfg
    b. Пишется чей ход, например “player1:”, выводится статус кораблей – слева своих, справа чужих – двумя матрицами, но с выводом “координат” – буквы латиницы от “a” до “j” и числа от 1 до 10, ожидается ввод этого игрока
    c. В новой игре всегда первым ходит player1
    d. Игрок имеет два вариант ввода: exit и координаты “удара”, например a1, k10, буквы английского алфавита от “a” до “j” и числа от 1 до 10
    e. Если попал по кораблю противника – отрисовать новый статус и можно атаковать ещё раз

    f. Если попытка стрельбы туда, куда уже был выстрел – попросить игрока сделать ввод заново
    g. Если не попал – переход хода другому игроку
    h. Если попал по последней ячейке последнего корабля противника – вывести сообщение о том, что текущий игрок победил
    i. Если неправильный ввод – попросить повторить ввод, не перерисовывать статус кораблей
    j. Если вводится exit – сохраняется три файла: lastgame_player_1.cfg, lastgame_player_2.cfg, lastgame_cfg; *_player_*-файлы хранят расположение кораблей и информацию о том, куда уже стрелял противник, отметки о выстреле пишутся как “X” вместо 0 или 1, lastgame.cfg хранит две строчки – в первой номер текущего хода, во второй – чей ход сейчас
4. При выборе “Продолжить игру”:
    a. Попытка чтения lastgame_player_1.cfg, lastgame_player_2.cfg, lastgame_cfg – если файлы прочитаны успешно, то продолжить игру – вывести статус и приглашение ввода актуального игрока и корректно прогрузить информацию о статусе кораблей; если файлы не прочитаны – вывести основное меню снова

Требования к интеграционному тесту:

1. Основной сценарий:
    a. заводятся два тестовых игрока-бота
    b. если нет “поврежденных”, но “не затопленных” кораблей, то каждый тестовый игрок стреляет в рандомную точку игрового поля, в которую не стрелял ранее
    c. если уже есть попадание в “не затопленный” корабль – выстрел производится по соседним точкам, где могут быть оставшиеся ячейки этого корабля
    d. если промах – передача хода
    e. игрок-бот не ошибается (не стреляет дважды в одну точку, не стреляет туда, где кораблей быть не может)
    f. по затоплении половины кораблей одного из игроков – произвести выход из игры сразу после передачи хода, запустить продолжение этой игры
    g. дождаться завершения игры