# Лабораторная работа номер 2- Функциональное тестирование методом белого ящика

В рамках лабораторной работы необходимо произвести функциональное тестирование кода методом белого ящика (всех ветвей)

Задание на лабораторную работу.

1. Разработать функцию  в соответствии со своим вариантом. **Внимание. Варианты на лабораторную работу раздаются заново**.
2. Разработать функциональные тесты для написанного кода  методом белого  ящика. Добиваться 100% прохождения тестов не нужно. Необходимо описать принципы выбора тестов.

К отчету должна быть приложена спецификация на тесты в следующем формате:

**<начало примера>**

Тестируемая функциональность

| Имя теста | Описание сценария | Входные данные | Выходные данные |
|---|---|---|---|
| | | | |
| | | | |

**<конец примера>**

Например:

Имеется фкнкция sign (x){

If (x>0) return 1;else

If (x<0) return -1;

Else return 0;

}

Функция sign(int)

| Имя теста | Описание сценария | Входные данные | Выходные данные |
|---|---|---|---|
| positiveSign | x>0: true<br>x<0 false | Входной параметр 10 | Результат вызова 1 |

| | | | |
|---|---|---|---|
| negativeSign | x>0:false<br>x<0 true | Входной параметр -10 | Результат вызова -1 |
| zeroSign | x>0: false<br>x<0 false | Входной параметр 0 | Результат вызова 0 |

**Тесты должны быть описаны достаточно недвусмысленно чтобы их содержание было понятно без заглядывания в код!**

Решать задачу можно на языках C++ и Java. Желающим выбрать другой язык необходимо убедиться, что

А) для этого языка существуют необходимые инструменты

Б) он в состоянии объяснить преподавателю, что именно написано в его коде

Отчет должен содержать описание применения всех указанных шагов. Не следует добиваться, чтобы 100% покрытие и полная корректность тестов были достигнуты уже в пункте 2.Значительно более интересным является процесс исправления недостатков тестовой базы.

**Вариант 1.** Компилятор простых арифметических выражений, например 2+(-5)*(7-8). Вход и выход в виде строк

**Вариант 2.** Функция поиска пути в неориентированном графе методом поиска в ширину. На вход подается граф и две вершины. На выходе – путь между этими вершинами.

**Вариант 3.** Функция поиска пути в неориентированном графе методом поиска в глубину. На вход подается граф и две вершины. На выходе – путь между этими вершинами.

**Вариант 4**. Функция поиска пути в неориентированном графе методом A*. На вход подается карта (граф с географическими координатами вершин) и два угла. На выходе – путь между этими узлами.

**Вариант 5**. Функция балансировки двоичного дерева

**Вариант 6**. Функция, рассчитывающая контур пересечения двух треугольников

**Вариант 7.** Хеш-таблица, не перетирающая элементы при вводе значений с совпадающим ключом, а хранящая список таких элементов и, соответственно, возвращающая их методом get. Метод – двойное хеширование

**Вариант 8**. Функция, рассчитывающая следующий ход в игре крестики-нолики на доске заданного размера и для заданной длины выигрышной последовательности путем построения полного дерева решений(например, доска 5 на 5 и длина выигрышной последовательности 4)

**Вариант 9** Функция, производящая поиск заданного набора строк в текстовом файле. Поиск должен уметь находить любую строку из набора, при этом должен правильно обрабатывать переносы текста. Использовать алгоритм Ахо-Корасик.

**Вариант 10** Парсер, использующий простые регулярные выражения, вводимые с клавиатуры, содержащие управляющие конструкции . – любой символ, * - 0 и более символов, + - 1 и более символов (вводится регулярное выражение и строка, результатом является позиция, с которой это выражение встречается в тексте)

**Вариант 11** Молекула ДНК состоит из последовательностей нуклеотидов А, Г, Ц и У. Несколько одинаковых молекул известной длины были нарезаны на фрагменты произвольной длины.

Функция восстанавливает исходную молекулу в том случае, если это возможно сделать единственным образом
Пример: АГЦЦГГУААЦЦ нарезана на фрагменты АГЦЦ,  ЦГГУ , ГГУАА и УААЦЦ.
Пример невосстанавливаемой последовательности: АГЦЦГГУААЦЦ нарезана на фрагменты АГЦЦ, ГГУАА и УААЦЦ.
Вариант алгоритма решения. В памяти строится ориентированный граф, в вершинах которого находятся фрагменты, а связи соединяют  два фрагмента, если фрагмент-источник может быть слева от фрагмента-приемника. Далее в графе ищутся все пути и для каждого проверяется, что он содержит в себе все фрагменты. Если такой путь один, то задача считается решенной
**Вариант 12** В матричной форме задается система линейных уравнений, необходимо ее решить (например, методом Гаусса).
**Вариант 13.** Реализовать структуру «Список с пропусками”, см. https://ru.wikipedia.org/wiki/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D1%81_%D0%BF%D1%80%D0%BE%D0%BF%D1%83%D1%81%D0%BA%D0%B0%D0%BC%D0%B8
Реализовать функции добавления, удаления и поиска.
**Вариант 14.** В заданном произвольном тексте найти все повторяющиеся фрагменты текста длиной не менее трех слов  (без использования стемминга, т.е. слова в различных склонениях и падежах считаются разными, знаки препинания не учитываются). При этом для каждого повторяющегося фрагмента должна указываться максимальная длина, например, для данного текста

- «поиска пути в неориентированном графе методом поиска в” встречается дважды
- «поиска пути в неориентированном графе методом” встречается трижды
- Более короткие части отдельно не встречаются, поэтому не рассматриваются