**Все задания на лабораторные.**

Распределение вариантов: Берём свой порядковый номер в журнале. Делим на количество вариантов. Берём остаток от деления и прибавляем один. Получаем номер варианта.

**Лабораторная 1. Структурное тестирование программного обеспечения.**

Лабораторная состоит из двух частей:

* 1. Необходимо протестировать требования, предъявляемые заказчиком. Требования берём из документа lab1V2.pdf. Для каждого требования необходимо написать уточняющие вопросы если требование сформулировано не корректно или не однозначно.
  2. Провести структурное тестирование алгоритма по методу базового пути. Берём алгоритм из файла Lab32017.pdf пишем алгоритм для решения задачи. Для каждой функции строим потоковый граф. Ищем все возможные пути в графе и пишем для каждого пути свой тест.

Варианты заданий:

**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 А\*. 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

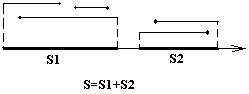
**Лабораторная работа 2. Функциональное тестирование программного обеспечения**

Берём алгоритм из файла: Лабораторная работа 2 Функциональное тестирование. Пишем алгоритм для решения задачи. Для полученной программы по методу причин и следствий строим причинно-следственный граф. На основе графа строим таблицу с комбинациями причин и следствий. Для каждого столбца таблицы пишем тест.

Варианты заданий:

**Вариант 1***.*

Разработать программу определения суммарной длины тени, которую отбрасывают на ось ОХ отрезки, параллельные этой оси и заданные координатами x начала и конца отрезка:



**Вариант 2.**

Разработать программу исследования уравнений второго порядка с двумя неизвестными Ax2+2Bxy+Cy2+2Dx+2Ey+F=0. Программа должна определять вид графика: эллипс, парабола, гипербола, две пересекающиеся прямые, две параллельные прямые, две мнимые прямые.

Примечание. Вид прямой втрого порядка определяется по двум дискриминантам

большому: и малому .

Малый дискиминант для эллипса положителен, для гиперболы отрицателен, для параболы равен нулю. Если большой дискриминант равен нулю, то линия второго порядка распадается на две прямых:

для эллиптического вида - пересекающиеся мнимые прямые (точка), для гиперболического вида - пара пересекающихся действительных прямых, для параболического вида - две параллельные прямые.

**Вариант 3.**

Разработать программу определения вида треугольника, заданного длинами его сторон: равносторонний, равнобедренный, прямоугольный, разносторонний.

**Вариант 4.**

Разработать программу определения вида четырехугольника, заданного координатами вершин на плоскости: квадрат, прямоугольник, параллелограмм, ромб, равнобедренная трапеция, прямоугольная трапеция, трапеция общего вида, четырехугольник общего вида.

**Вариант 5.**

Разработать программу, определяющую взаимное расположение прямых в пространстве: параллельны, пересекаются, скрещиваются и отдельно, расположение каждой прямой (параллельна оси, перпендикулярна плоскости или общего расположения). Прямые задаются координатами двух точек.

Примечание. Две прямые лежат в одной плоскости, если

, прямые параллельны если ,

где *l=x2-x1, m=y2-y1, n=z2-z1* (верхний индекс соответствует номеру прямой)*.*

**Вариант 6.**Определение взаимного положения прямой и окружности. Прямая описывается уравнением Y=kX+b. Окружность с центром в начале координат задается радиусом R. Результат – линии не пересекаются, пересекаются в двух точках, прямая линия является касательной к окружности.

**Вариант 7.**Определение взаимного положения двух окружностей. Окружности задаются координатами центра X,Y и радиусом R. Результат – линии не пересекаются, пересекаются в двух точках, касаются в одной точке, совпадают.

**Лабораторная работа 3. Тестирование пользовательского интерфейса.**

Добавляем графический интерфейс для задания из лабораторной работы 1. Тестируем полученный интерфейс на основе сценариев использования. Тест представляет собой сценарий действий пользователя с элементами интерфейса и ожидаемые реакции интерфейса на них. Подробнее можно посмотреть в лекции в файле: lecture4\_2017.pdf

**Лабораторная работа 4. Автоматизированное тестирование.**

Необходимо решить задачу в соответствии с вариантом. Каждый вариант задачи в своём файле 1 задание …pdf – 13 задание …pdf Для решённой задачи написать модульные тесты полностью покрывающие исходный код. Для написания модульных тестов использовать любую библиотеку модульного тестирования в зависимости от выбранного языка программирования. Пример автоматизированного тестирования в VisualStudio Описан в файле «Модульные тесты в VisualStudio 2012.docx» Для других языков программирования и сред разработки можно использовать аналогичные механизмы. Выполнить модульные тесты и получить результаты.

**Лабораторная работа 5. Разработка собственного тестового драйвера.**

Необходимо разработать программу тестового драйвера для тестирования программных модулей. На вход данная программа получает путь к исходному коду тестируемого модуля. Также драйвер на вход получает файл с описанием тестов, которые нужно выполнить. Описание теста включает в себя название функции, параметры, которые должны быть переданы в функцию и ожидаемый результат. Структура файла с тестами может быть произвольной. Результатом работы тестового драйвера должен быть отчёт о результатах тестирования. Отчёт должен содержать следующую информацию: номер теста, полученный результат, ожидаемый результат, результат прохождения теста (success или faile). Если результат прохождения теста faile, то нужно вывести причину ошибки. Драйвер тестирования должен быть максимально универсальным. Для языков программирования, позволяющих выполнять код в runtime нужно чтобы драйвер работал с разными модулями без перекомпиляции. Если это не возможно, то сделать так, чтобы для тестирования нового программного кода требовалось минимум изменений в коде драйвера тестирования (в идеале в отдельном заголовочном файле изменить путь к файлу и возможно список вызовов методов).

**Лабораторная работа 6 Нагрузочное тестирование.**

Выбрать произвольный ресурс в сети интернет. Выбрать две наиболее часто выполняемые операции на данном ресурсе или наиболее ресурсоёмкие. Определить группы пользователей, которые работают с ресурсом. Определить количество пользователей в каждой группе и операции, которые они выполняют с интенсивностями. Построить профиль нагрузки. Оптимизировать время тестирования и интенсивности нагрузки. Взять программу для автоматизированного нагрузочного тестирования. В качестве такого ПО можно использовать бесплатную программу ApachJmetter. В программе настроить требуемые параметры нагрузки. Провести нагрузочное тестирование. Получить статистику. Увеличить нагруку и посмотреть как себя будет вести система. Проанализировать результаты.