**Задание 12.** В пакете прикладных программ *National Instruments LabView* создать виртуальные приборы для загрузки данных из файла и сохранения данных в файл.

Для чётных вариантов необходимо реализовать два виртуальных прибора: один для загрузки данных, другой – для сохранения данных.

Для нечётных вариантов необходимо реализовать один виртуальный прибор, сочетающий в себе как загрузку данных из файла, так и сохранение данных в файл.

Использовать следующие схемы загрузки/сохранения:

а) Расчёт значений функции выполняется в *Microsoft Office Excel* и записывается в файл *fromExcel.txt* (в текстовый файл копируются два столбца: в одном содержатся значения абсцисс, в другом – значения ординат). Между столбцами вручную должны быть удалены элементы табуляции, разделяющие значения, и на их место вставлены разделители в соответствии с таблицей вариантов (Таблица 2). В пакете прикладных программ *National Instruments LabView* выполняется построчное считывание данных из файла *fromExcel.txt* и конвертация их для построения графика функциональной зависимости.

**Внимание:** учесть наличие разрывов в функциональных зависимостях. Наличие соединения точек, расположенных на границах разрыва будет считаться ошибкой.

б) Расчёт значений функции выполняется в пакете прикладных программ *National Instruments LabView* и записывается в файл *fromLabView.txt*. В *Microsoft Office Excel* из файла *fromLabView.txt* копируются рассчитанные значения, и далее по ним проводится построение графика функциональной зависимости.

**Внимание:** учесть наличие разрывов. Все выявленные разрывы отобразить асимптотами.

Для вариантов, делящихся на 5 без остатка, реализовать точность вычислений до одного знака после запятой.

Для вариантов, дающих в остатке единицу при делении на 5, – точность до двух знаков после запятой.

Для вариантов, дающих в остатке двойку при делении на 5, – точность до трёх знаков после запятой.

Для вариантов, дающих в остатке тройку при делении на 5, – точность до четырёх знаков после запятой.

Для вариантов, дающих в остатке четвёрку при делении на 5, – точность до пяти знаков после запятой.

На блок-диаграмме все функциональные, то есть значащие пересечения линий связи (узлы) обозначить точками. Для этого в меню перейти к настройкам «*Tools > Options…*», далее в выпадающем списке перейти к настройкам блок-диаграммы (*Block Diagram*) и выставить галочку напротив пункта «*Show dots at wire junctions*».

По итогам выполнения работы сдаются следующие файлы:

- отчёт, выполненный в текстовом редакторе *Microsoft Office Word* (*\*.doc* или *\*.docx*);

- файл виртуального прибора *National Instruments LabView* (*\*.vi*) по индивидуальной части работы (нечётные варианты);

- файлы виртуальных приборов *National Instruments LabView* (*\*.vi*) по индивидуальной части работы (чётные варианты);

- электронные таблицы *Microsoft Office Excel* (*\*.xlsx*), содержащие график функциональной зависимости и данные для построения графика функциональной зависимости;

- текстовые файлы с данными, получаемыми из ВП, и данными, необходимыми для загрузки в ВП.

Отправленные поодиночке файлы проверке не подлежат. При отсутствии одного из упомянутых файлов зачёт по заданию не выставляется. Рекомендуется упаковка перечня направляемых на проверку файлов в архив.

**Требования к именам файлов:**

**Общий вид формата имени файла:** «*Дата. Задание. Фамилия.mcdx*»

**Формат записи даты:** «*ГГГГММДД*», где *ГГГГ* – четыре цифры текущего года, *ММ* – две цифры текущего месяца, *ДД* – две цифры текущего дня.

**Формат записи задания:** «Задание *NNk*», где *NN* – две цифры номера задания, *k* – обозначение «о», если файл содержит общую часть; обозначение «и», если файл содержит индивидуальную часть; обозначение «ои», если файл содержит как общую, так и индивидуальную части.

**Если устранить замечания по работе удаётся в тот же день:** после фамилии ставится пробел и в круглых скобках записывается номер попытки исправления.

**Примеры правильных имён файлов, которые сдаются на проверку впервые:**

«*20190405. Задание 12ои. Иванов.docx*»

«*20190405. Задание 12ои. Иванов.vi*»

**Примеры правильных имён файлов, которые сдаются на проверку повторно в тот же день:**

«*20190405. Задание 12ои. Иванов (1).docx*»

«*20190405. Задание 12ои. Иванов (1).vi*»

**Внимание!** Не забудьте выполнить автоматическую нумерацию страниц в отчёте.

Отчёт по выполненной работе должен содержать:

0. Титульный лист.

1. Формулировку цели работы.

2. Описание задачи согласно выданному варианту.

3. Составление блок-схемы алгоритма программы.

4. Подбор и расчёт тестовых примеров.

5. Листинг кода составленного программного обеспечения (блок-диаграммы *LabView*).

6. Графический пользовательский интерфейс программного обеспечения (передняя панель виртуального прибора *LabView*) и его описание.

7. Расчёт тестовых примеров с использованием составленного программного обеспечения.

8. Формулировку вывода о проделанной работе (обезличено – исключить из вывода местоимения, такие как «я», «мы» и другие).

Рекомендации к отчёту, доказывающие самостоятельность выполнения работы и упрощающие процедуру проверки отчёта преподавателем:

1. Выполнение дополнительных скриншотов для случаев, когда текстовое описание проделанных действий становится громоздким или трудным к восприятию.

2. Нумерация рисунков (если есть) с подписями, содержащими названия рисунков, например, «Рисунок 1 – Пользовательский интерфейс *Microsoft Office Excel*».

**Цель работы (одна из возможных формулировок)**: освоение навыков работы с текстовыми файлами как протоколами передачи информации между средами программирования и пакетами прикладных программ; приобретение знаний, касающихся настройки диалогов и событий; знакомство с механизмом обработки событий в среде с графическим языком программирования.

**Настройка диалога с проверкой на нулевой шаг при построении графика функциональной зависимости:**

Далее рассмотрен пример графического пользовательского интерфейса виртуального прибора с ожиданием возникновения ошибки при запуске с введённым нулевым шагом. Предполагается, что к настоящему моменту известна технология подключения и настройки *XY Graph* (Рисунок 1).

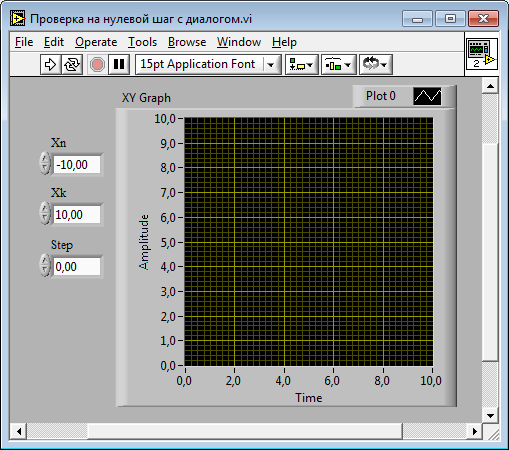


Рисунок 1 – Стандартный графический пользовательский интерфейс для построения графиков функциональных зависимостей

Рассматриваются две ситуации:

- запуск ВП с нулевым шагом (Рисунок 2);

- запуск ВП с ненулевым шагом (нормальная работа ВП).

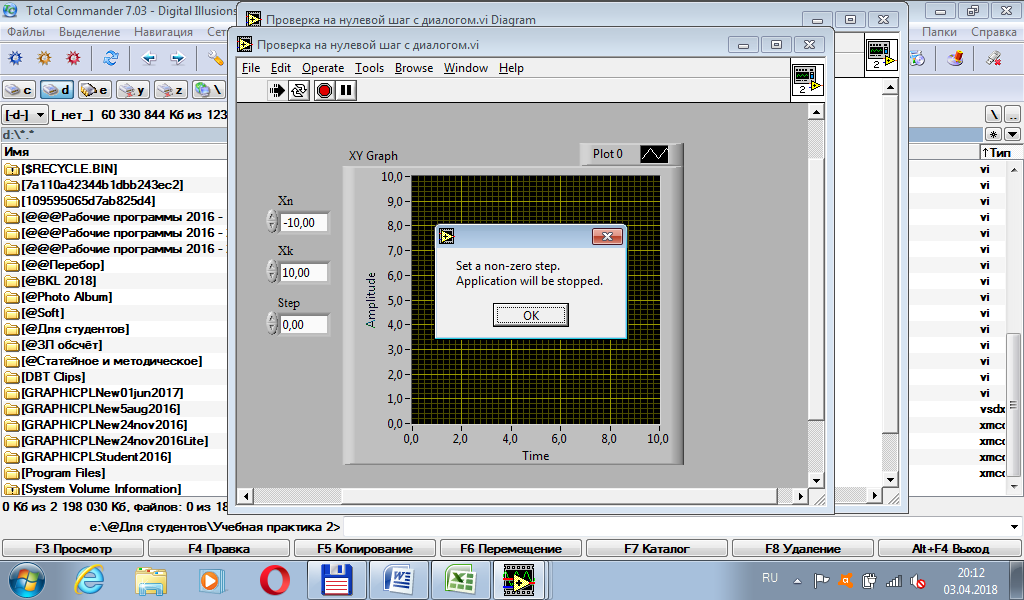


Рисунок 2 – Побуждающее сообщение об ошибке при нулевом шаге

Для отлова подобных ошибок в исходных данных организован разветвляющийся вычислительный процесс с использованием *CASE*-структуры. Шаг проверяется на равенство нулю и в случае, если это так, выводится побуждающее сообщение и работа ВП приостанавливается (Рисунок 3).

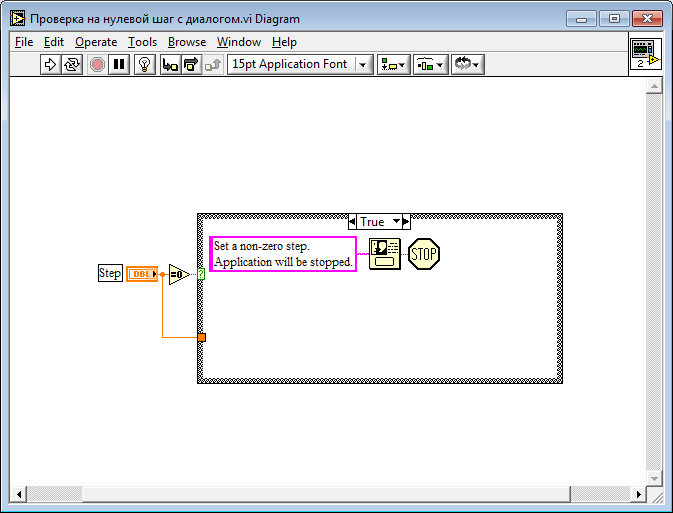


Рисунок 3 – Фрагмент блок-диаграммы с проверкой шага на нулевое значение. В случае нулевого шага – выдача побуждающего сообщения и прерывание работы программы

В ином случае (Рисунок 4) происходит нормальная работа ВП с ненулевым шагом, приводящая в результате к построению графика функциональной зависимости (Рисунок 5).

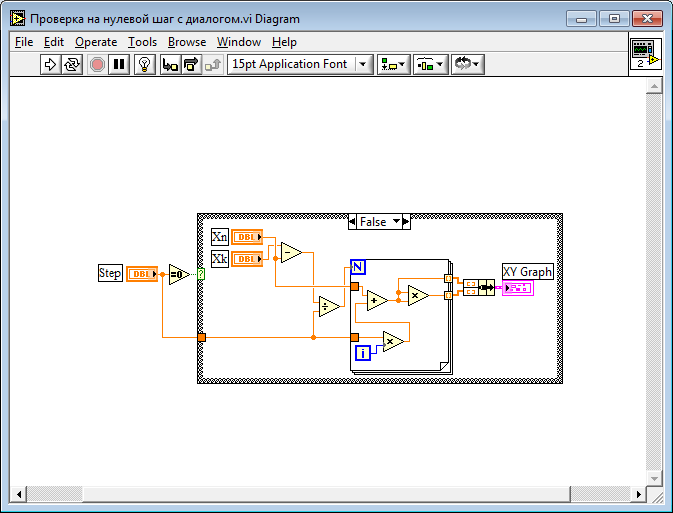


Рисунок 4 – Фрагмент блок-диаграммы для случая, когда задан ненулевой шаг

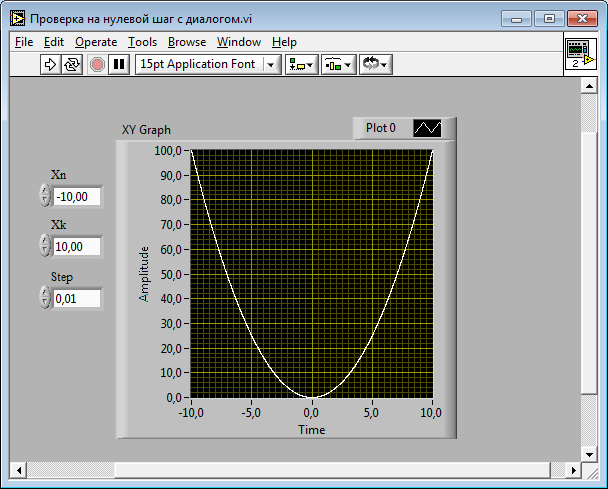


Рисунок 5 – При ненулевом шаге выполняется построение графика функциональной зависимости

**Памятка программисту:**

Для выполнения работы потребуются следующие функции, которые к настоящему моменту не рассматривались в заданиях по Учебной практике:

- однокнопочный диалог (Рисунок 6);

- останов (Рисунок 7).

Однокнопочный диалог принимает в качестве входной информации строку, а на выход передаёт истинный результат в случае, если нажата кнопка «*ОК*» (вообще говоря, истинный результат здесь всегда).

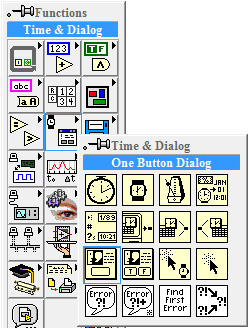


Рисунок 6 – Расположение элемента «Диалог с одной единственной кнопкой» («*ОК»*)

Сигнал с однокнопочного диалога передаётся на останов, принимающий на вход логический сигнал/результат: логическую переменную или логическую константу.

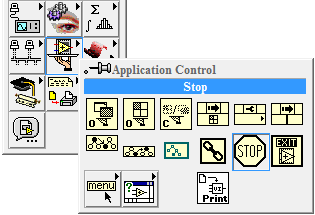


Рисунок 7 – Расположение элемента «Прерывание работы программы» (*STOP*)

**Формирование пути к файлу:**

Файлы, относящиеся к проекту, рекомендуется хранить в непосредственной близости с этим проектом. Таким образом, отталкиваться желательно от пути запуска приложения на исполнение – этот путь гарантированно существует. Для понимания обучающимися идеи формирования путей к файлам в среде *National Instruments* *LabView* разработан следующий учебный пример (Рисунок 8).

Учебный ВП напрямую не создаёт какой-либо файл, но позволяет проследить этапы формирования пути к файлу. Отдельно выводятся строковые и путевые переменные:

- путь к исполняемому приложению;

- строка, содержащая имя исполняемого приложения;

- путь к директории, содержащей исполняемое приложение;

- путь к желаемому файлу.

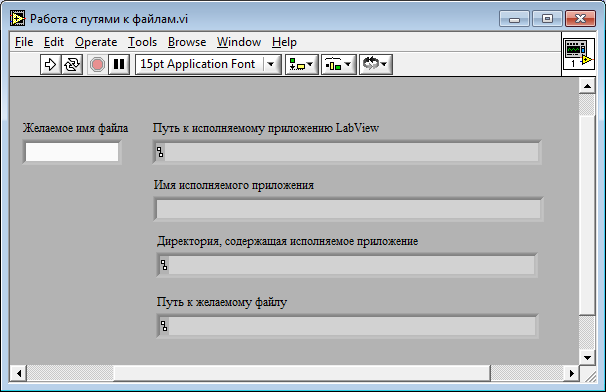
****

Рисунок 8 – Графический пользовательский интерфейс виртуального прибора, обучающего формированию пути к файлу

На вход виртуального прибора пользователь подаёт строку, содержащую желаемое наименование файла (Рисунок 8). Вместе с тем на блок-диаграмму выносится компонент «Путь текущего ВП» («*Current VI’s Path*»), выводимый параллельно на интерфейсный элемент и на вход функции «Декомпозиция Пути» («*Strip Path*»), раскладывающей путь к текущему ВП на его имя и его директорию. Для просмотра имени файла оно выведено на интерфейсный элемент управления. Полученный путь к директории передаётся в параллель на интерфейсный элемент управления для просмотра и на вход функции «Построение Пути» («Build Path») совместно с желаемым именем файла. На выходе получается путь к формируемому файлу. Описанный механизм изображён на блок-диаграмме Рисунок 9. Результат работы такого ВП показан на Рисунке 10.

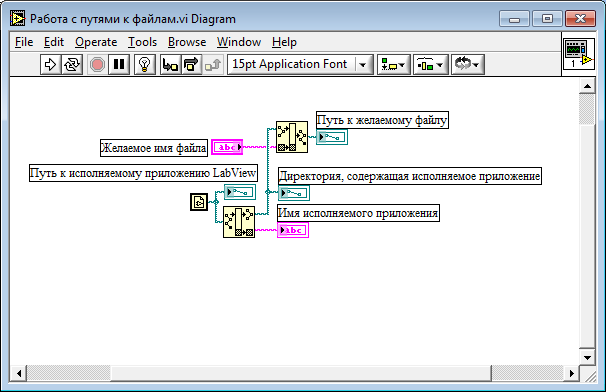
****

Рисунок 9 – Блок-диаграмма виртуального прибора, обучающего формированию пути к файлу

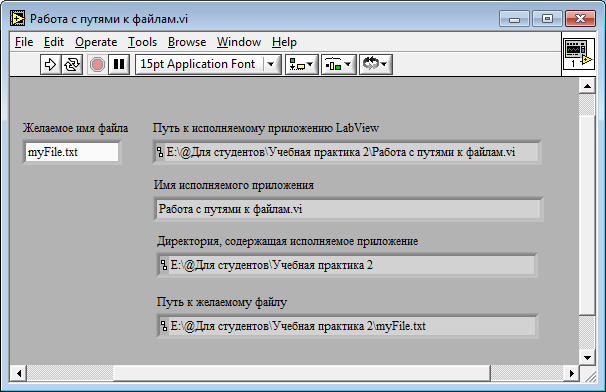
****

Рисунок 10 – Результат запуска виртуального прибора, обучающего формированию пути к файлу «*myFile.txt*»

**Памятка программисту:**

Ниже представлен перечень мест расположения функциональных элементов, необходимых для построения представленного выше виртуального прибора. Среди файловых констант необходимо найти путь к текущему файлу ВП (Рисунок 11).

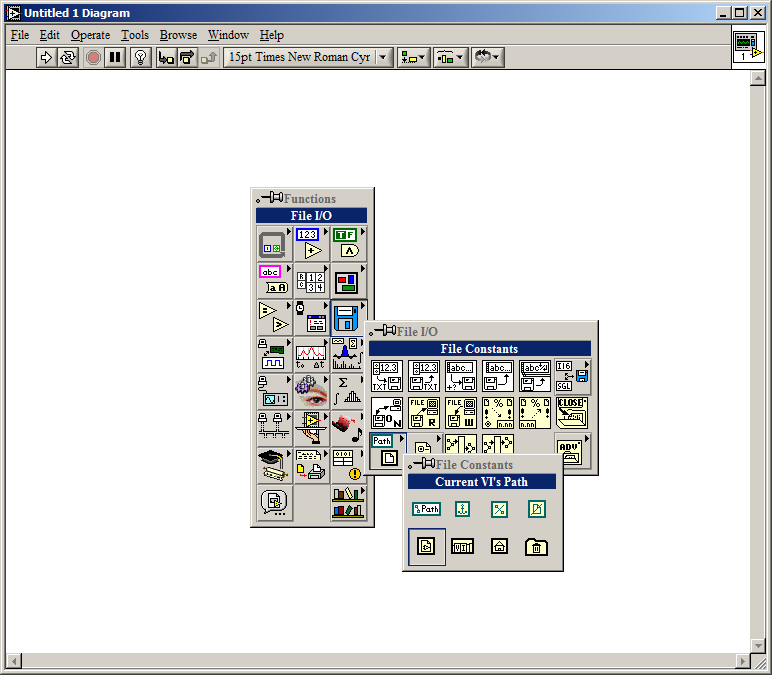


Рисунок 11 – Размещение элемента «Текущий путь к виртуальному прибору»

На Рисунке 12 показано место расположения путевой константы, участвующей в составлении пути к файлу.

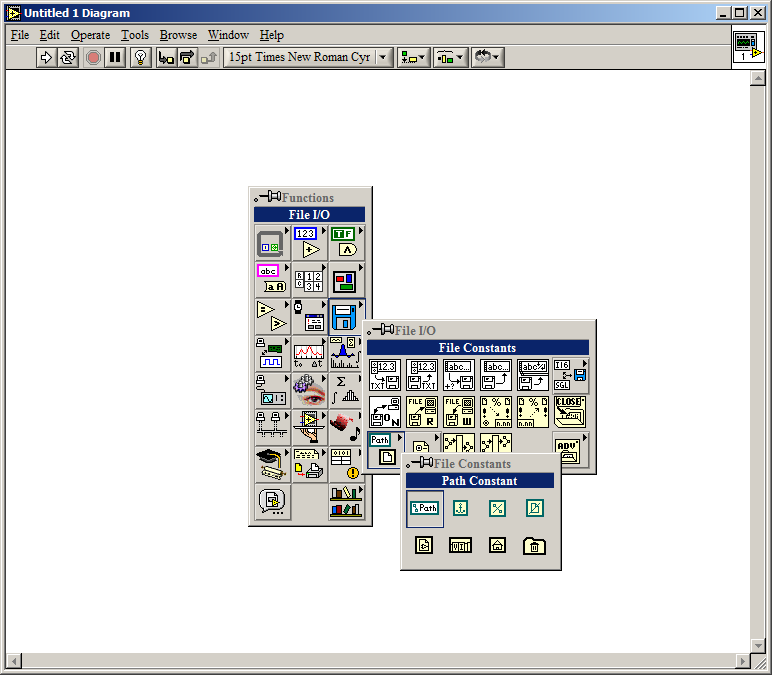


Рисунок 12 – Размещение элемента «Путевая константа»

На Рисунке 13 показана функция для построения пути к файлу, которая из комбинации переменных двух типов «путь» и «строка» составляет новый, дополненный путь (путь дополняется новым звеном, записанным через «\»).

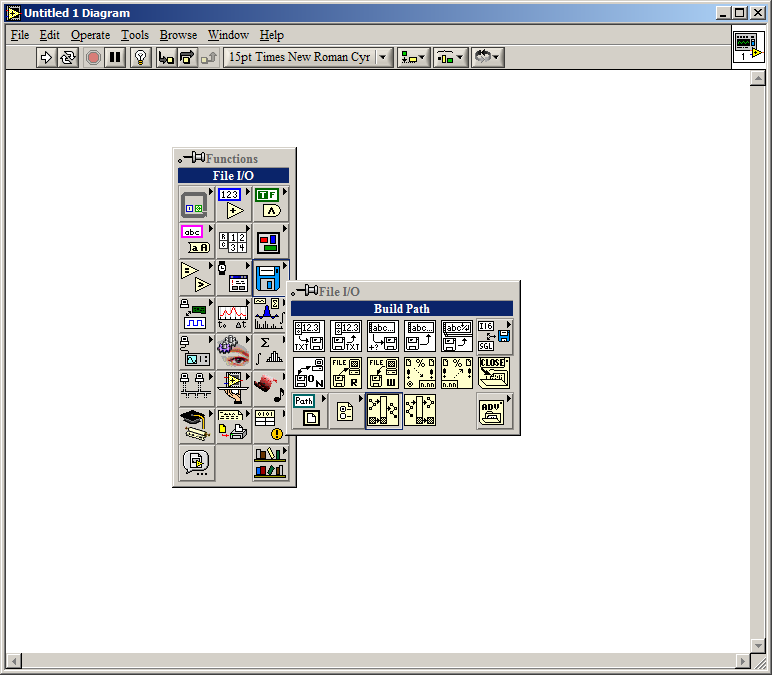


Рисунок 13 – Размещение элемента «Построение пути»

На Рисунке 14 показана функция для декомпозиции пути к файлу, которая разбирает путь к файлу на составляющие двух типов «путь» и «строка». Новый, сокращённый путь урезается до последнего символа «\», а сегмент пути, записанный после последнего символа «\», преобразуется в строку.

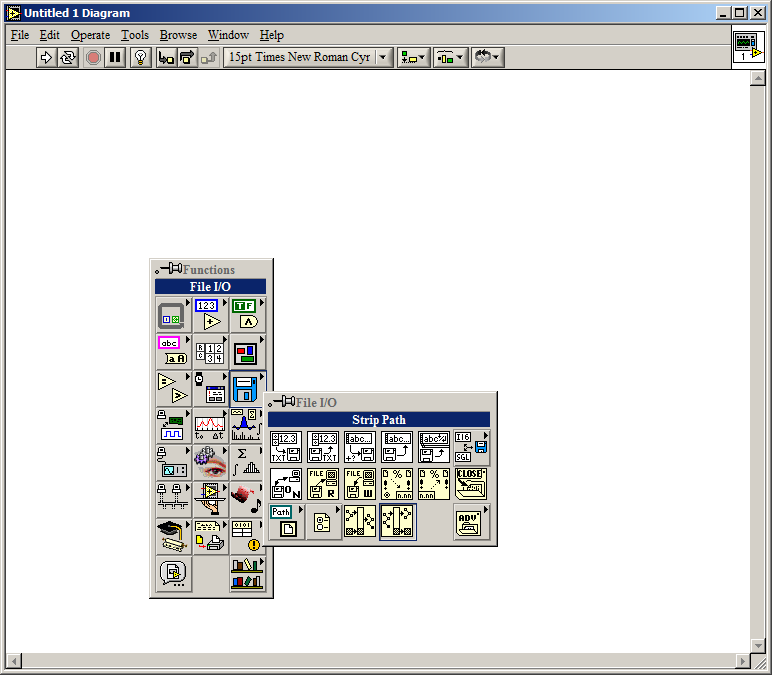


Рисунок 14 – Размещение элемента «Разбор пути»

**Реализация виртуального прибора, записывающего данные в файл:**

В этом разделе рассматривается решение задачи увязки логического интерфейсного элемента управления со структурой «Событие» («*Event*»). На Рисунке 15 показан графический пользовательский интерфейс ВП, записывающего данные в файл. Помимо логического элемента управления на передней панели размещены параметры диапазона (начало, конец и шаг). Работа ВП сводится к расчёту точек функциональной зависимости, относящихся к указанному диапазону и записи значений в файл. Построение графика функциональной зависимости в этом случае передаётся сторонней программе (в рассматриваемой задаче таковой является *Microsoft Office Excel*).

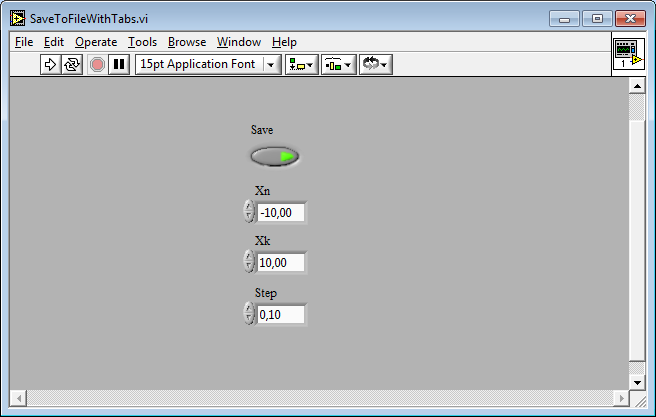
****

Рисунок 15 – Графический пользовательский интерфейс виртуального прибора, реализующего запись в файл

На Рисунке 16 представлен пример реализации ВП, нацеленного на запись данных в текстовый файл.

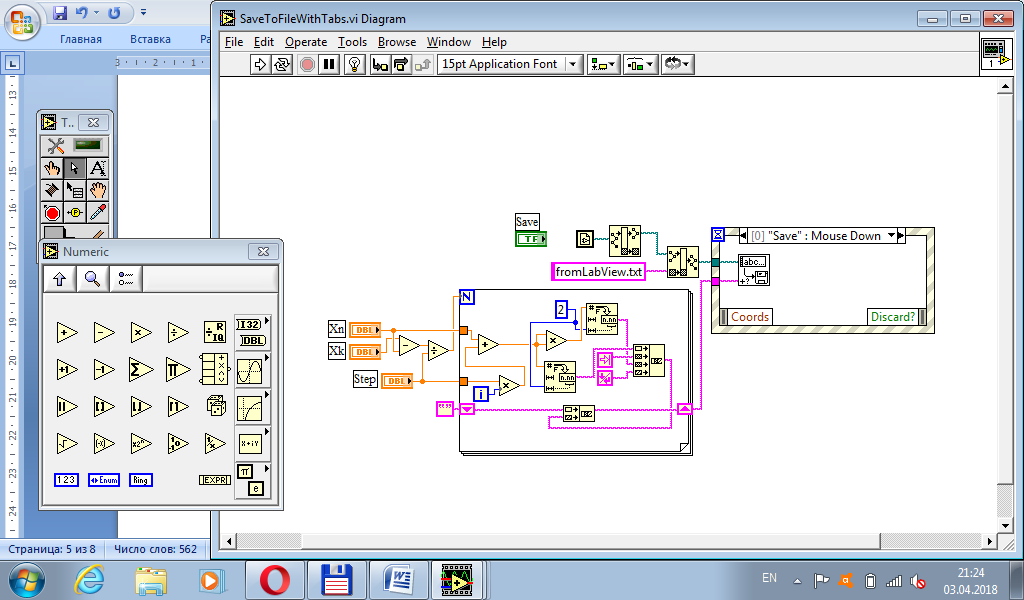
****

Рисунок 16 – Блок-диаграмма виртуального прибора, реализующего запись в текстовый файл. Кнопка записи не связана ни с чем и принципиально не должна быть связанной

**Памятка программисту:**

Далее рассмотрены частные особенности структуры события, необходимые для решения поставленной задачи. На Рисунке 17 показано место расположения структуры «Событие» в перечне функциональных элементов.

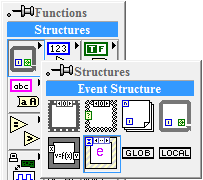


Рисунок 17 – Размещение структуры «Событие» в меню

Размещённая на блок-диаграмме структура «Событие» по умолчанию выглядит нижеследующим образом, как показано на Рисунке 18.

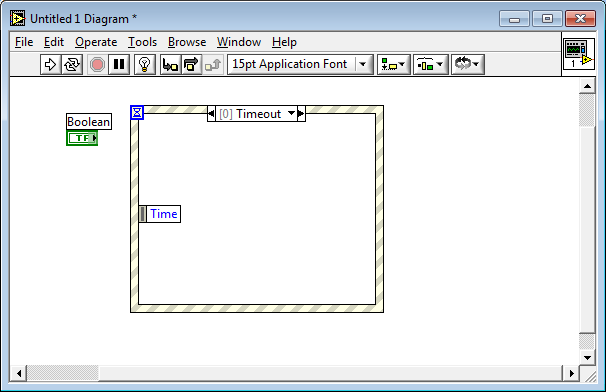
****

Рисунок 18 – Позиционирование структуры «Событие» на блок-диаграмме виртуального прибора

Имеющийся блок работы по «таймауту» (*Timeout*), срабатывающий по истечении заданного времени, использовать в рассматриваемом ВП не требуется, потому рациональным решением является его удаление. Однако, из структуры невозможно удалить блок, если тот является единственным его компонентом. Потому сначала необходимо создать дополнительный блок нового события. Это можно сделать через контекстное меню и пункт «Добавление событийной ситуации» («*Add Event Case*»). Это показано на Рисунке 19.

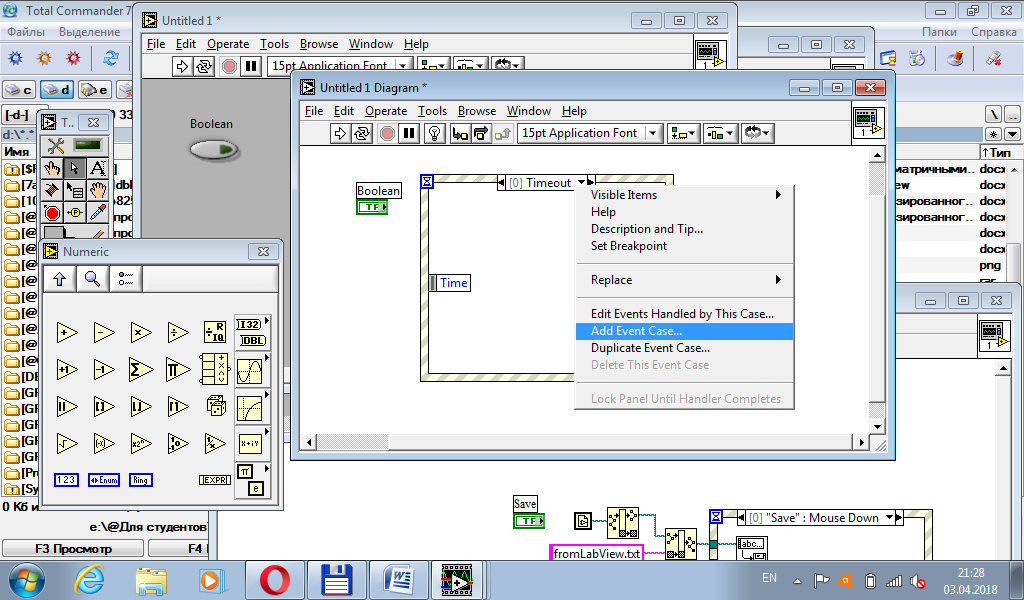
****

Рисунок 19 – Добавление нового события в структуру. Элемент, с которым должно быть связано событие, уже должен находиться на графическом пользовательском интерфейсе

В этом случае очень важно, чтобы элемент, с которым планируется организовать связь через событие, был обязательно размещён на передней панели ВП и блок-диаграмме, соответственно. Добавление новой событийной ситуации реализуется через диалог, представленный на Рисунке 20. В нижнем левом списке представлены источники событий, в частности, элементы управления (*Controls*), в числе которых можно найти интересующую кнопку «*Boolean*».

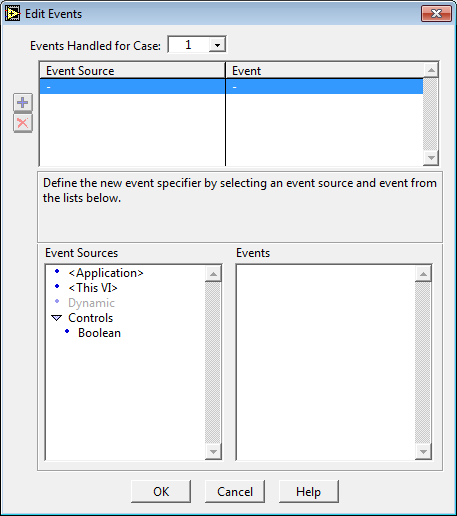
****

Рисунок 20 – Меню настройки нового события. Слева снизу указывается перечень возможных источников событий

После выбора источника события в нижнем правом списке появляется перечень тех событий, по которым можно организовать связь. Наиболее эффективной для рассматриваемой задачи видится событие «Отпускание кнопки мыши» (*Mouse Up*). Его и необходимо добавить, как показано на Рисунке 21. Событие должно отобразиться в верхнем списке, состоящем из двух колонок.

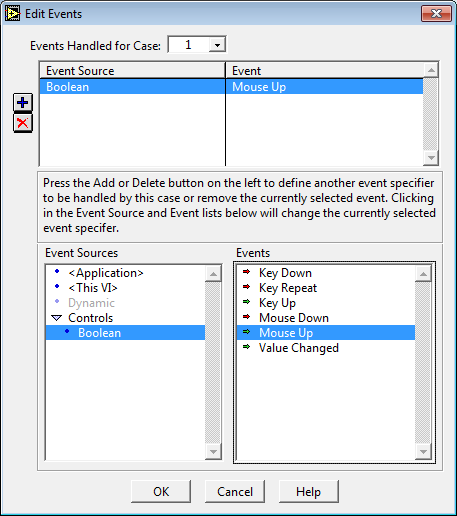
****

Рисунок 21 – Меню настройки события для компонента *Boolean*. Справа снизу указывается вид события, которое необходимо отслеживать (отпускание кнопки мыши после нажатия)

После выбора и подтверждения нажатием на кнопку «*ОК*» блок будет добавлен в структуру «Событие» вместе со стандартным набором параметров. Поскольку задействование ни одного из элементов перечня в ВП не планируется, их необходимо удалить один за другим последовательно через контекстное меню (Рисунок 22).

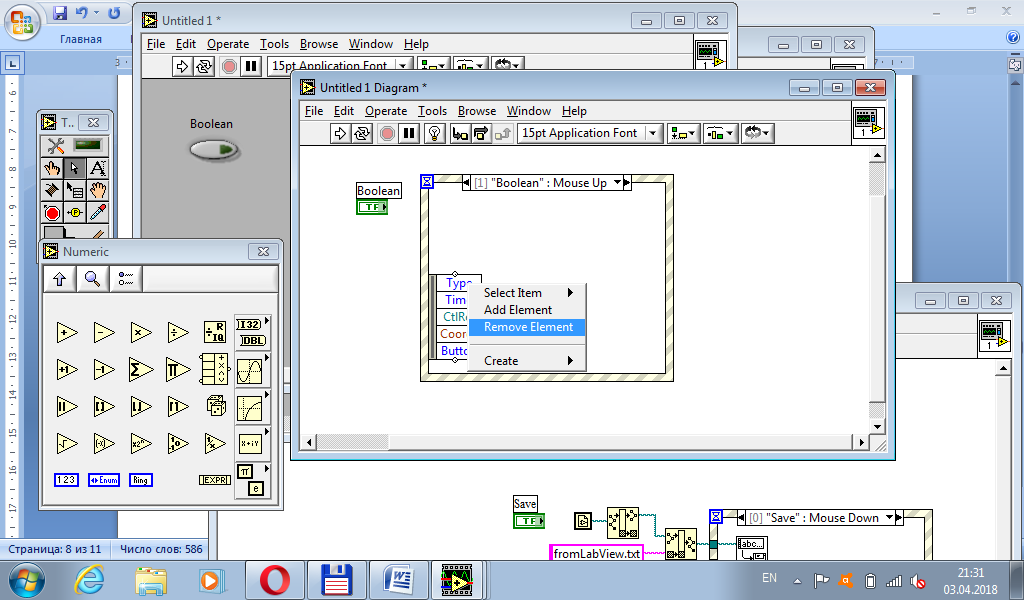
****

Рисунок 22 – Удаление избыточных параметров во вновь созданном событии

По итогам выполнения указанных выше манипуляций параметр в перечне блока останется всего один. Последний параметр удалить невозможно (это принципиально), да это и не требуется (Рисунок 23).

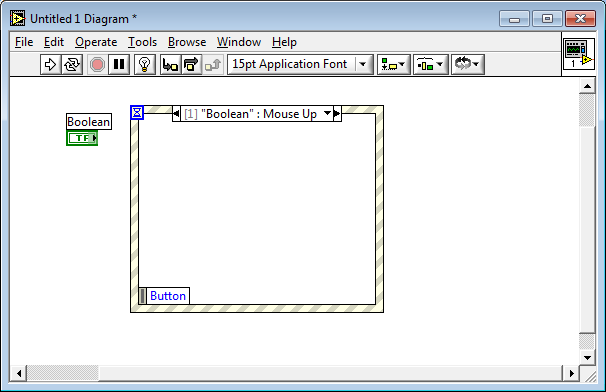
****

Рисунок 23 – Конечный вид события, отслеживающего отпускание кнопки мыши после нажатия на кнопку *Boolean*

Далее не стоит забывать про оставшееся событие, происходящее по таймауту. Необходимо переключиться на его блок и удалить его через контекстное меню (Рисунок 24).

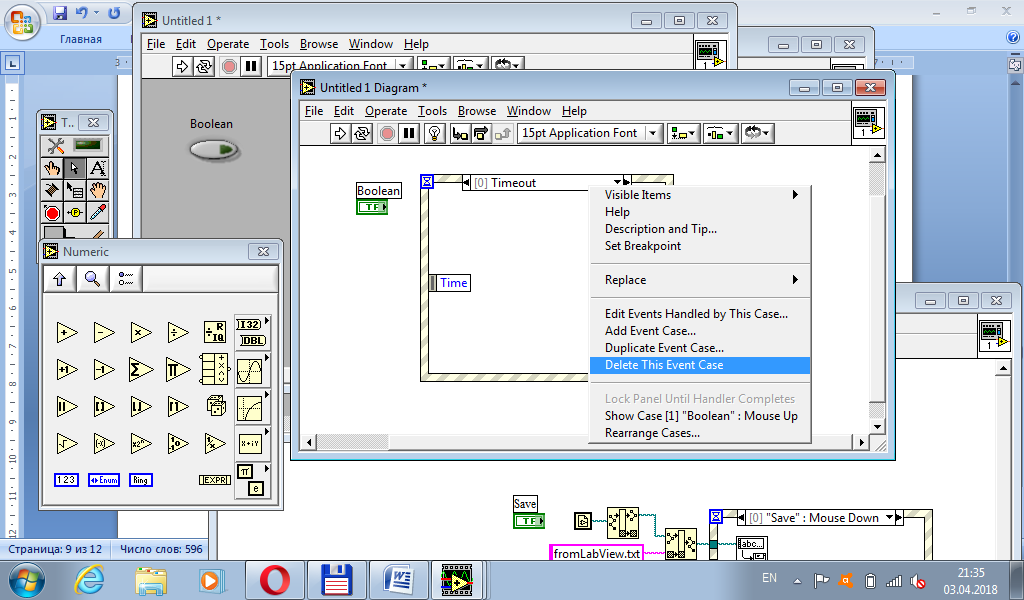
****

Рисунок 24 – Возврат к исходной вкладке, создаваемой по умолчанию в структуре события, для её удаления через контекстное меню

К настоящему моменту среди нерассмотренных ранее в рамках заданий Учебной практики функциональных элементов, но применённых при реализации ВП, является запись символов файл. Место расположения данного функционального элемента показано на Рисунке 25.

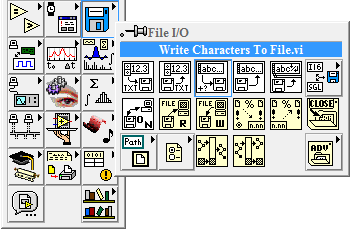


Рисунок 25 – Размещение в меню элемента «Запись символов в файл»

**Реализация виртуального прибора, считывающего данные из файла:**

Обратной/зеркальной для записи данных в файл является операция чтения данных из файла. Далее рассматривается проектирование виртуального прибора, предназначенного для чтения данных из текстового файла. На Рисунке 26 представлен возможный графический пользовательский интерфейс ВП, считывающего сведения из текстового файла.

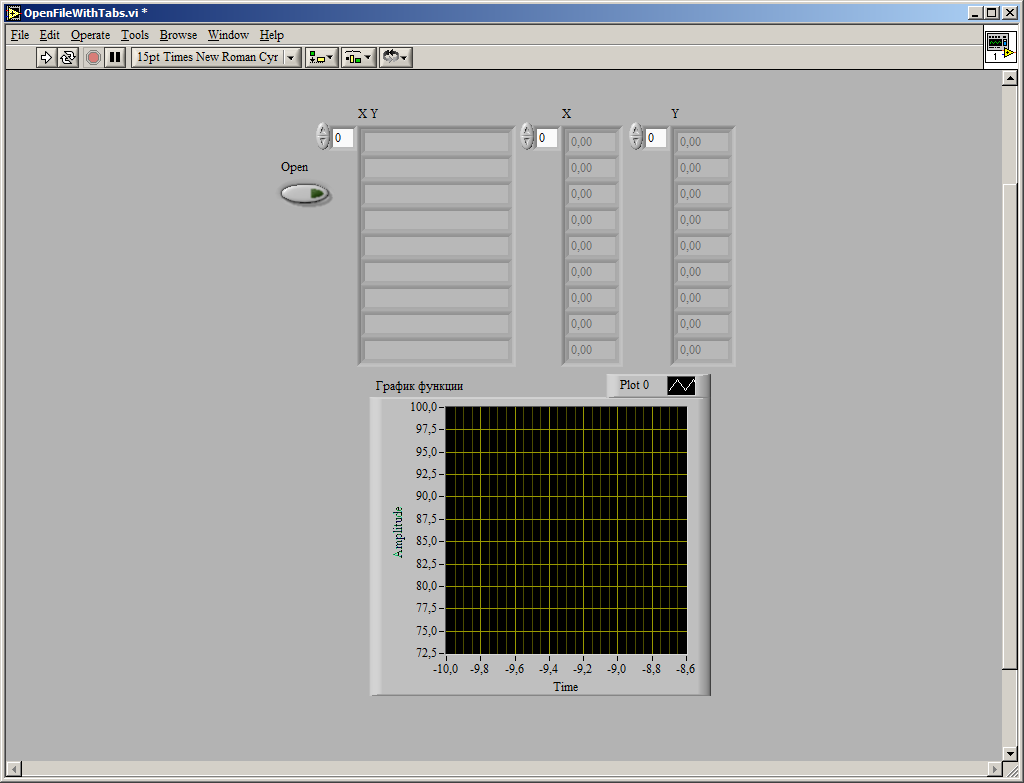


Рисунок 26 – Графический пользовательский интерфейс виртуального прибора, реализующего чтение из файла

Программное наполнение виртуального прибора представлено на блок-диаграмме, изображённой на Рисунке 27.

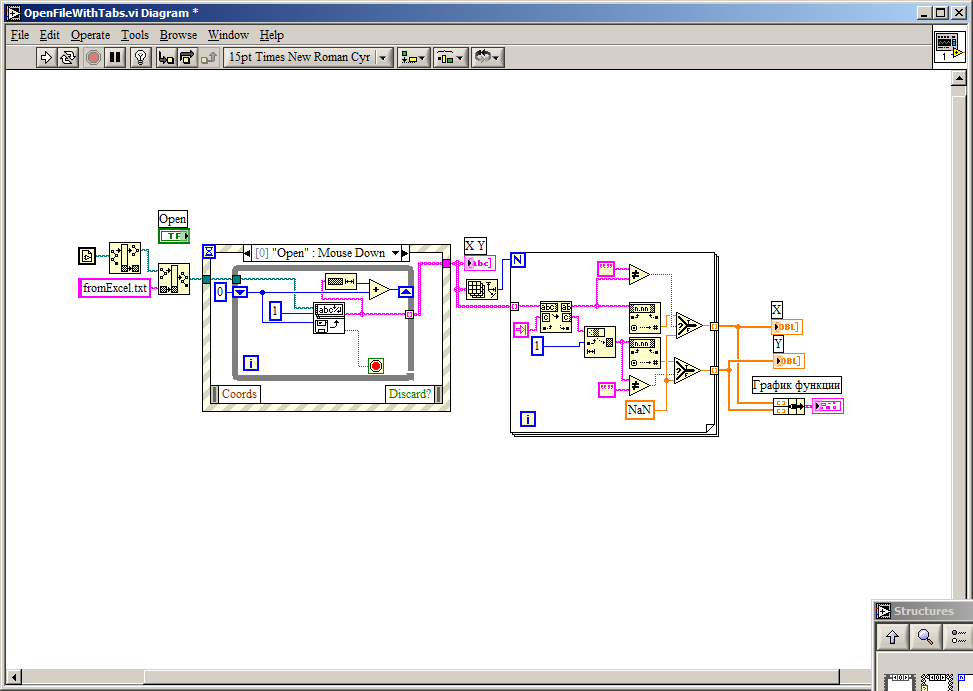


Рисунок 27 – Блок-диаграмма виртуального прибора, реализующего чтение из файла

Рассмотрим идею чтения данных из файла, заключённую (идею) в цикле *While*. На один из входов посимвольного чтения файла «*number of lines*» подаётся константа, равная «1» для чтения одной единственной строки. На другой вход «*start of read offset*» (смещение от начала для чтения) через сдвиговый регистр передаётся длина сумма длин (в символах) ранее считанных строк. На каждом витке/итерации цикла формируется один элемент строкового массива, содержащий:

- значение по абсциссе;

- табуляция (**важно:** это не пробел);

- значение по ординате;

- перевод курсора на следующую строку (*Enter/*↵).

Так, например, строковый массив «*X Y*» в элементе с нулевым индексом содержит значение: -10,00 100,00↵. В соответствии с Рисунком 28.

Цикл *While* переведён в режим останова по истине. К маркеру останова по истине подключён логический выход «*EOF?*» элемента связи с файлом. Аббревиатура «*EOF*» расшифровывается как «*end of file*», что означает «конец файла». По достижении конца файла маркер «*EOF*» выставляется в значение «истина» и работа цикла завершается.

Рассмотрим идею разбора считанных из файла данных, заключённую в цикле *For*. Полученный массив «*X Y*» обрабатывается в цикле поэлементно. То есть каждый элемент массива гарантированно соответствует строке в файле «*fromExcel.txt*». Первый блок в цикле определяет позицию символа «*Tab*». Часть, предшествующая «*Tab*» (выход элемента, визуально обозначенный как «*ab*», именуемый «*substring before match*» – подстрока, предшествующая совпадению), направляется на конвертер из строки в вещественное значение. Это абсциссы. Часть, начинающаяся с «*Tab*» (выход элемента, визуально обозначенный как «*c*», именуемый «*match + rest of string*» – совпадение и остаток строки), направляется на элемент выделения подстроки из строки с первого индекса (тем самым из строки исключается «*Tab*») и направляется на другой конвертер из строки в вещественное значение. Это ординаты.

Далее результаты обёрнуты логикой, которая предварительно отлавливает пустые строки. Пустые строки, которые были получены в *Excel* на месте разрывов, важно сохранить при переносе в текстовый файл. Пустые строки являются сигналом для вставки не числовой точки (*Not A Number – NaN*). Не числовая точка является признаком для *XY Graph* в *National Instruments LabView* для того, чтобы обусловить разрыв – две соседние точки (*x1, y1*) и (*x2, y2*), расположенные на границах разрыва, не будут соединены, поскольку между ними находится точка (*NaN, NaN*).

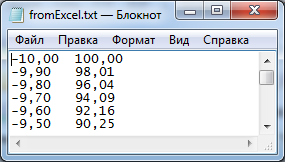


Рисунок 28 – Содержимое файла «*fromExcel.txt*»

На Рисунке 29 отмечено место расположения функции для чтения символов из текстового файла.

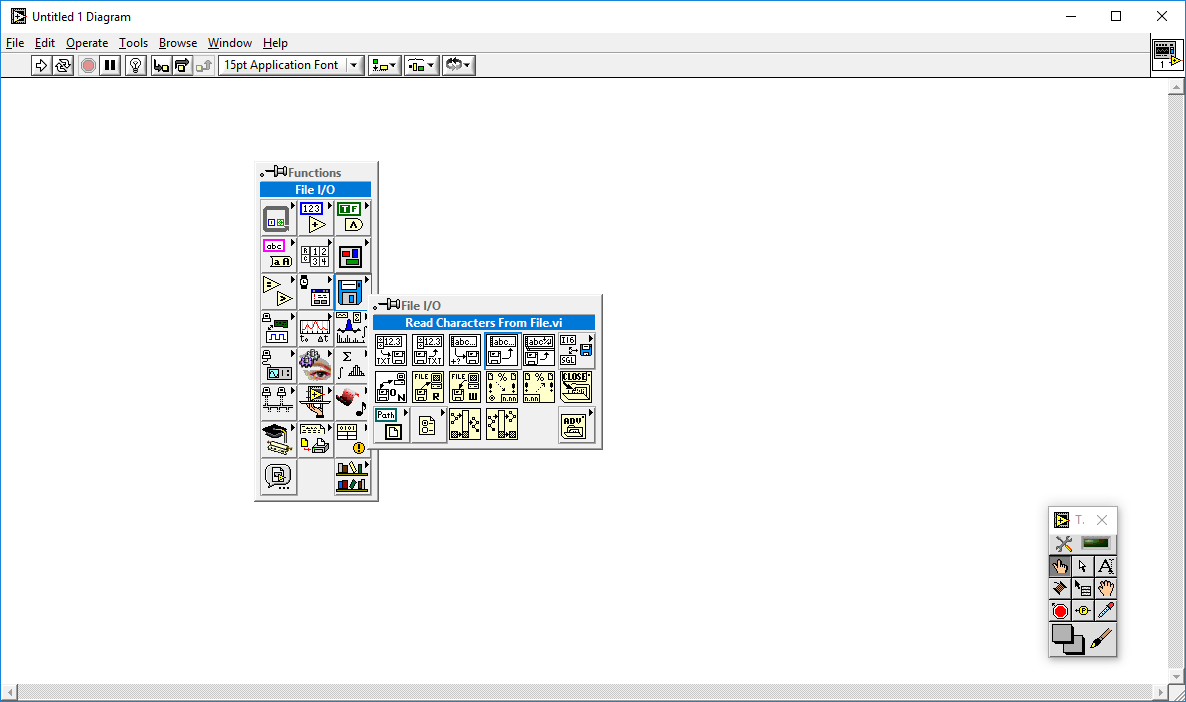


Рисунок 29 – Место расположения функции для чтения из текстового файла

**Варианты исследуемых функций:**

Таблица 1 – Варианты исследуемых функций

|  |
| --- |
| 1. **а)** x4+2·x2+x+1; **б)** x/(1+1/x); |
| 2. **а)** tg(2·x)·cos(2·x)·lg(2·x); **б)** x·sin(x3); |
| 3. **а)** sinh(x/e)+ex+1 **б)** 1/(x·lg(x)); |
| 4. **а)** sin(1/(a·x))·cos(b·x); **б)** sin(x)·tg(1/x); |
| 5. **а)** sin(x2)· ctg(1/x); **б)** sin(1/(2·x))·cos(1/(3·x)); |
| 6. **а)** x3·cos(x)+x+1; **б)** x/(lg(x) + 1); |
| 7. **а)** x4·sin(x); **б)** tg(x)·sin(3·x); |
| 8. **а)** ctg(5·x); **б)** tg(x)3; |
| 9. **а)** sin(2·x)·cos(4·x); **б)** sin(2·x)· ctg(x); |
| 10. **а)** log2(x); **б)** cos(x)·x2; |
| 11. **а)** x2+2·x+1; **б)** 1/sin(x); |
| 12. **а)** x2·sin(2·x); **б)** tg(2·x)·cos(2·x); |
| 13. **а)** sin(x) ·x; **б)** x·sin(2·x)+2·x·cos(x); |
| 14. **а)** log2(x)·cos(x); **б)** x2/sin(x)+cos(x)/x; |
| 15. **а)** x2·sin(x); **б)** 1/sin(x)+1/cos(x); |
| 16. **а)** tg(2·x)4; **б)** ctg(3·x)·x; |
| 17. **а)** x·lg(x); **б)** sinh(2·x)+ex·sin(x); |
| 18. **а)** lg(x); **б)** x2·cos(x)+ x·sin(x)+x; |
| 19. **а)** sin(2·x)·tg(x); **б)** sin(a·x)·cos(b·x); |
| 20. **а)** sin(a·x)·cos(b·x); **б)** sin(2·x)·tg(x); |
| 21. **а)** x2·cos(x)+ x·sin(x)+x; **б)** lg(x); |
| 22. **а)** sinh(2·x)+ex·sin(x); **б)** x·lg(x); |
| 23. **а)** ctg(3·x)·x; **б)** tg(2·x)4; |
| 24. **а)** 1/sin(x)+1/cos(x); **б)** x2·sin(x); |
| 25. **а)** x2/sin(x)+cos(x)/x; **б)** log2(x)·cos(x); |
| 26. **а)** x·sin(2·x)+2·x·cos(x); **б)** sin(x) ·x; |
| 27. **а)** tg(2·x)·cos(2·x); **б)** x2·sin(2·x); |
| 28. **а)** 1/sin(x); **б)** x2+2·x+1; |
| 29. **а)** cos(x)·x2; **б)** log2(x); |
| 30. **а)** sin(2·x)·ctg(x); **б)** sin(2·x)·cos(4·x); |

Таблица 2 – Варианты разделителей данных

|  |  |  |
| --- | --- | --- |
| 1. **##** | 11. **[** | 21. **>** |
| 2. **Space** | 12. **~** | 22. **|** |
| 3. ++ | 13. **\** | 23. **^** |
| 4. **]** | 14. **<** | 24. **/** |
| 5. \_ | 15. **+** | 25. **&** |
| 6. **//** | 16. **{** | 26. \* |
| 7. **}** | 17. **#** | 27. **@** |
| 8. **№** | 18. **(** | 28. **!** |
| 9. **=** | 19. **)** | 29. **%** |
| 10. **?** | 20. **-** | 30. **;** |