***А.Н. Шилин, А.А. Шилин,***

***П.В. Дикарев***

***Виртуальная   
лаборатория на базе LabVIEW***

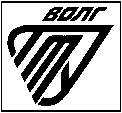
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

А. Н. Шилин, А. А. Шилин, П. В. Дикарев

Виртуальная лаборатория   
на базе *LabVIEW*

*Учебно-методическое пособие*



Волгоград

2020

УДК 004.9(075)

Рецензенты:

Зав. кафедрой радиофизики ФГАОУ ВО   
«Волгоградский государственный университет»,   
*канд. ф.-м. наук, доцент*, *А.Л. Якимец*;

Доцент кафедры Энергетики ФГБОУ ВО «НИУ «Московский энергетический институт», филиал в г. Волжском,  
 *канд. техн. наук, с.н.с., А.В. Стрижаченко*

Печатается по решению редакционно-издательского совета

Волгоградского государственного технического университета

**Шилин А.Н., Шилин А.А., Дикарев П.В.**

Виртуальная лаборатория на базе *LabVIEW* / А. Н. Шилин, А. А. Шилин, П. В. Дикарев; ВолгГТУ. – Волгоград, 2020. – 90 с.

ISBN 978-5-9948-0576-3

Учебно-методическое пособие содержит практикум по основам измерительных технологий, в котором представлены работы по методам обработки и оценки погрешностей результатов измерений, поверке средств измерений и методам и средствам измерения электрических и неэлектрических величин. Все работы практикума выполняются с компьютерными моделями, реализованными в среде *LabVIEW*.

Предназначено для студентов старших курсов физических специальностей технических университетов, а также для начального обучения студентов инженерных специальностей.

Ил. 00. Табл. 00. Библиогр.: 10 назв.

ISBN 978-5-9948-0576-3 © Волгоградский государственный

технический университет, 2020  
 © А. Н. Шилин, А. А. Шилин,   
 П. В. Дикарев, 2020

**Оглавление**

Введение 5

1 Основы программирования в графической среде *LabVIEW* 8

1.1 Основные элементы среды программирования *LabVIEW* 8

1.2 Палитра элементов управления и индикаторов 11

1.3 Палитра функций 13

1.4 Палитра инструментов 14

1.5 Справочная система в *LabVIEW* 16

2 Примеры программ в среде *LabVIEW* 18

2.1 Первая программа 18

2.2 Программа, преобразующая градусы *C* в градусы *F* 19

2.3 Программа, вычисляющая заданную формулу 22

2.4 Циклы 25

2.4.1 Цикл по Условию (*While*) 25

2.4.2 Цикл с фиксированным числом итераций (*For*) 27

2.4.3 Сдвиговые регистры 30

3 Массивы 34

3.1 Создание массива элементов управления и отображения 34

3.2 Двумерные массивы 35

3.3 Создание массива констант 37

3.4 Создание массива с помощью цикла 38

3.5 Создание двумерных (2*D*) массивов с помощью цикла 39

3.6 Пример вычисления чисел Фибоначчи 41

3.7 Функции работы с массивами 42

3.7.1 *Array Size* 42

3.7.2 *Initialize Array* 43

3.7.3 *Build Array* 45

3.7.4 *Array Subset* 46

3.7.5 *Index Array* 47

3.7.6 *Search* 1*D* *Array* 50

3.8 Передача массива данных в цикл 53

3.8.1 Пример ВП с входным индексированным тоннелем 54

3.8.2 Виртуальный прибор поиска максимального значения 56

3.9 Полиморфизм 58

4 Отображение данных в графической среде *LabVIEW* 59

4.1 Использование графиков для отображения данных 59

4.1.1 Однокоординатные графики 60

4.1.2 Двухкоординатные графики 62

4.2 Использование палитры элементов управления графиком 65

5 Функция *Case Structure* 68

5.1 Примеры использования функции *Case Structure* 69

5.1.1 Функция *Case Structure* с логическим элементом 70

5.1.2 Функция *Case Structure* с числовым элементом 73

5.1.3 Функция *Case Structure* со строковым элементом 75

5.2 Определение вариантов функции *Case Strucrure* 77

Индивидуальные задания к лабораторной работе №1 - Вычисление 79

Индивидуальные задания к лабораторной работе №2 - Логика 81

Индивидуальные задания к лабораторной работе №3 - Циклы 83

Индивидуальные задания к лабораторной работе №4 – Массивы 85

Индивидуальные задания к лабораторной работе №5 – Графики 87

Библиографический список 89

# Введение

Учебно-методическое пособие представляет собой *LabVIEW* практикум по основам измерительных технологий. В данном учебно-методическом пособии подробно разобраны примеры создания виртуальных приборов для различных ситуаций, используя полный набор элементов среды программирования *LabVIEW*. В результате изучения этого пособия и выполнения лабораторно-практических заданий студенты должны освоить принципы построения виртуальных приборов в программной среде *LabVIEW*.

Практикум создан на основе компьютерных моделей, благодаря чему лабораторные работы могут выполняться как в учебной лаборатории в рамках традиционно организованного учебного процесса, так и на собственном компьютере во время самостоятельной работы в библиотеке или дома. Основой практикума является комплекс прикладных программ, созданный в инструментальной среде разработки приложений *LabVIEW* .

Лабораторный практикум создан коллективом преподавателей кафедры «Электротехника» Волгоградского государственного технического университета (ВолгГТУ) на основе опыта преподавания дисциплин, связанных с изучением измерений, и практики применения *LabVIEW* программного обеспечения в учебном процессе.

Язык *LabVIEW* (Laboratory Virtual Instrument Engineering Workbench) разработан фирмой «National Instruments». Фирмой выпускаются разнообразные интерфейсные устройства, встраиваемые в компьютер или подключаемые к его портам, устройства генерации и обработки реальных электрических сигналов, датчики, регистрирующие различные физические процессы, и т.п.

Язык *LabVIEW* не похож на другие языки программирования. С его помощью создается не программа, как мы привыкли ее представлять, а виртуальный инструмент, предназначенный не только для моделирования тех или иных процессов, но и для управления аппаратными средствами и исследования реальных физических объектов. Простота образных графических конструкций, наглядность и читаемость готовых программ заставляет отдать предпочтение языку *LabVIEW* перед другими. Он подобен таким системам, как *С*++ или *Basic*, однако в отличие от них оперирует не кодовыми строками, а блоками диаграмм, что делает программирование более простым и понятным.

Преимущества этого языка проявляются при выполнении следующих задач:

• исследование возможности контроля измерительной системой параметров объекта измерений;

• анализ и оптимизация структуры ККИС;

• расчет и анализ статистических и динамических метрологических характеристик (МХ) ККИС и ее компонентов.

Построенные с использованием данного пакета имитационные модели наглядно отражают суть реальных физических процессов, протекающих в объектах исследования. *LabVIEW* включает большой выбор библиотек программ, позволяющих:

• управлять доступом к внешним данным посредством стандартных интерфейсов (КОП, *VXI*, *RS*232, *USB*, и др.);

• анализировать данные измерений;

• осуществлять архивацию и хранение данных.

Сложность выполнения заданий по вариантам возрастает от занятия к занятию. Индивидуальные задания к лабораторным работам построены таким образом, чтобы по окончании их выполнения обучаемые имели достаточный уровень знаний и навыков работы с интерфейсом программной среды *LabVIEW*.

В учебно-методическом пособии отражены все этапы создания виртуального прибора: регистрация сигнала, обработка, отображение. Первый раздел учебного пособия посвящен основам программирования и создания прибора, второй – основным принципам построения программы и обработке сигналов, третий – созданию массивов и систем сбора данных, четвертый – отображению информации на графических элементах, пятый – функции *Case*, используемой для создания и выбора множества вариантов решения в программе. Оставшиеся разделы содержат лабораторный практикум с индивидуальными заданиями для обучающихся.

Данное пособие разработано для методического обеспечения подготовки бакалавров по дисциплине «Информационно-измерительные системы» направления 12.03.01 «Приборостроение» и магистров по дисциплине «Виртуальная лаборатория на базе *LabVIEW*» направления 12.04.01 «Приборостроение».

# 1 Основы программирования в графической среде *LabVIEW*

# 1.1 Основные элементы среды программирования *LabVIEW*

Запустим графическую среду разработчика программного обеспечения *National Instruments LabVIEW 2020.* В появившемся окне (рис. 1.1) выберем раздел *Blank VI* (пустой виртуальный прибор). После чего откроются два окна (рис. 1.2): лицевая панель (*Front Panel*) и блок- диаграмма (*Block Diagram*).

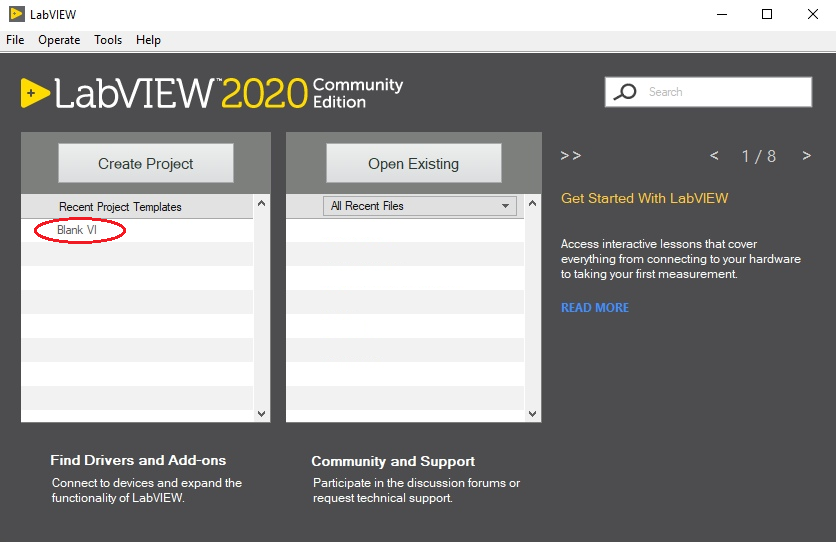


Рис. 1.1 – Стартовое окно *LabVIEW 2020*

Переключаться между окнами можно несколькими способами:

• щелкнув мышкой по соответствующему окну;

• комбинацией клавиш Ctrl+E

Создание программ в графической среде программирования *LabVIEW* производится в двух окнах, называемых:

*Front Panel* (лицевая панель);

*Block Diagram* (блок-диаграмма).

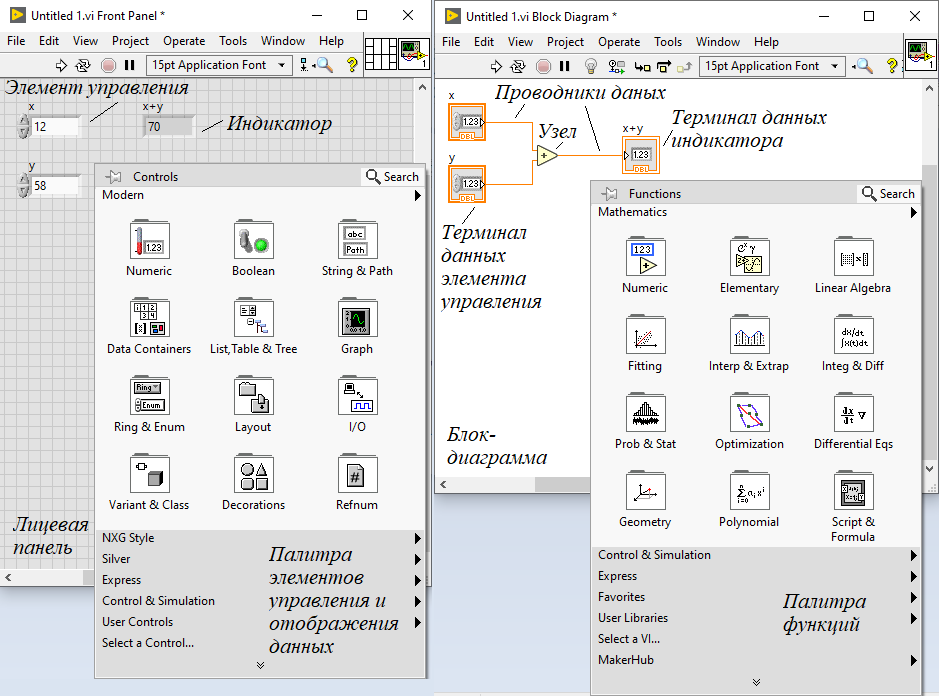


Рис. 1.2 – Графическая среда программирования *LabVIEW*

Ряд кнопок, расположенный под главным меню, называется инструментальной панелью (рис 1.3).

https://sun9-26.userapi.com/chyHMvQ_wKg5Q3wy6CPrKKtgXpOUqHWKEE1CZQ/HTxNfqd6ZEk.jpg

Рис. 1.3 – Инструментальная панель *LabVIEW*

https://sun9-41.userapi.com/ggYb7ojECNFyhfou3ExNQAcIDnBB2H8w_mFCrA/r1cJ-rS0it8.jpg

– Кнопка однократного запуска *Run* - запускает виртуальный прибор.

https://sun9-41.userapi.com/ggYb7ojECNFyhfou3ExNQAcIDnBB2H8w_mFCrA/r1cJ-rS0it8.jpg

– Кнопка непрерывного запуска *Run Continuously* – виртуальный прибор выполняется многократно до момента принудительной остановки

https://sun9-41.userapi.com/ggYb7ojECNFyhfou3ExNQAcIDnBB2H8w_mFCrA/r1cJ-rS0it8.jpg

– Во время выполнения виртуальный прибор активируется кнопка *Abort Execution*. Эта кнопка используется для немедленной остановки выполнения виртуального прибора.

https://sun9-41.userapi.com/ggYb7ojECNFyhfou3ExNQAcIDnBB2H8w_mFCrA/r1cJ-rS0it8.jpg

– Кнопка паузы *Pause* – если нажата, то приостанавливает работу прибора, до момента, пока не будет нажата снова.

На лицевой панели разрабатывается внешний вид будущего виртуального прибора (все программы, разработанные на *LabVIEW*,называются виртуальными приборами (*Virtual Instruments*). На ней создаются элементы управления и отображения, которые являются интерактивными средствами ввода и вывода данных этого виртуального прибора.

*Элементы управления (Controls)* – кнопки и другие устройства ввода данных.

*Элементы отображения (Indicators)* **–** графики, светодиоды и другие индикаторы. Элементы управления моделируют устройства ввода данных и передают данные на блок-

диаграмму ВП. Элементы отображения моделируют устройства вывода и отображения данных, которые получает или генерирует блок-диаграмма (рис. 1.2).

На блок-диаграмме разрабатывается исходный код будущего виртуального прибора. В отличии от классических языков программирования, таких как Си, Паскаль, ФОРТРАН, исходный код *LabVIEW* представляет собой блок-диаграмму, где все команды, операторы циклов и сравнения изображаются графическими обозначениями. Блок-диаграмма состоит из узлов, терминалов и проводников данных (рис. 1.2).

*Узлы (Node)* (рис. 1.2) – это объекты на блок-диаграмме, которые имеют одно или более полей ввода/вывода данных и выполняют алгоритмические операции виртуального прибора. Они аналогичны операторам, функциям и подпрограммам текстовых языков программирования. Узлы включают в себя функции (*functions*), подпрограммы (*SubVI*) и структуры (*Structures*). Подпрограмма - виртуальный прибор (ВПП), который можно использовать на блок-диаграмме другого ВП в качестве подпрограммы. Структуры *(Strucures)* - это элементы управления процессом, такие как структура Последовательности (*Flat sequence*), структура Варианта (*Case*), цикл по условию (*While*) и т.д.

Объекты лицевой панели на блок-диаграмме отображаются в виде *терминалов данных (Data terminals)* (рис. 1.2). *Терминалы данных* обеспечивают обмен данными между лицевой панелью и блок-диаграммой. Различают терминалы данных следующих типов: терминалы управления и отображения данных, терминалы узлов. Терминалы управления и отображения относятся к средствам управления и отображения данных на лицевой панели. Данные, введенные в элементы управления на лицевой панели, поступают на блок-диаграмму через терминалы управления.

Данные между объектами блок-диаграммы передаются по соединительным линиям - по *проводникам данных (Wires)* (рис. 1.2). *Проводник данных* аналогичен переменным в текстовых языках программирования. Каждый проводник данных имеет единственный источник данных, но может передавать их ко многим функциям. Проводники данных различаются цветом, стилем и толщиной линии, в зависимости от типа передаваемых данных. В среде *LabVIEW* объекты соединяются проводниками данных после их помещения на блок-диаграмму.

# 1.2 Палитра элементов управления и индикаторов

Для размещения элементов управления и отображения данных на лицевую панель используется палитра элементов управления и отображения (рис. 1.4). Палитра элементов управления и отображения доступна только с лицевой панели. Для вывода палитры на экран следует щелкнуть правой кнопкой мыши в рабочем пространстве лицевой панели.

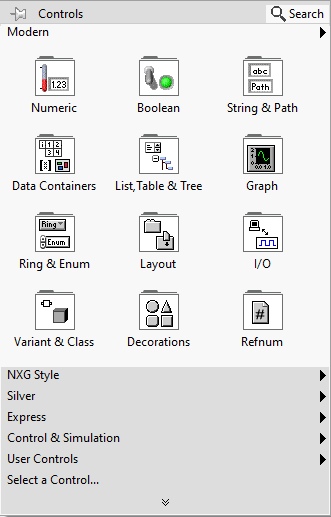


Рис. 1.4 – Палитра элементов управления и отображения

Все элементы управления и отображения на палитре сгруппированы по разделам:

• *Modern* – элементы управления и отображения имеют современный стиль (в данном пособии будет использоваться в основном этот раздел);

• *NXG Style* – элементы управления и отображения имеют стиль версии *LabVIEW NXG*

• *Silver* –элементы управления и отображения имеют серебряный стиль;

• *Express* – распространенные элементы управления и отображения;

• *Control & Simulation* – элементы управления и моделирования;

• *Users Controls* – элементы управления, созданные пользователем и сохраненные в библиотеке *LabVIEW*;

• *Select a Control* – элементы управления созданные пользователем.

Каждый раздел может делиться, в свою очередь, на подразделы (числовые, строковые индикаторы, кнопки и т.д.).

# 1.3 Палитра функций

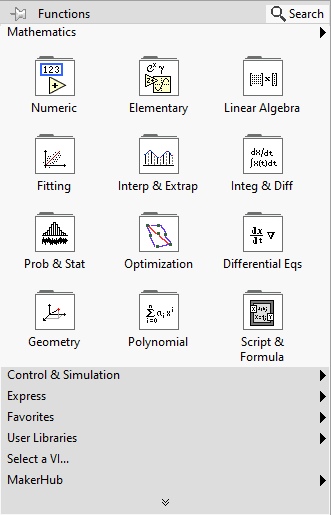


Рис. 1.5 Палитра функций

Как было сказано ранее на блоке - диаграмм разрабатывается исходный текст программы. Для этого используется палитра функций (*Functions pallete*). Для вывода палитры на экран следует щелкнуть правой кнопкой мыши в рабочем пространстве блока-диаграмм.

Все элементы на палитре сгруппированы по разделам:

• *Mathematics* – здесь собраны функции для математических вычислений (решение системы уравнений, решение дифференциальных уравнений и др.);

• *Control* *&* *Simulation* – элементы управления и моделирования;

• *Express* – распространенные функции, связанные как с вычислениями, так и с обработкой данных;

• *Favorites* – избранные функции;

• *User* *Libraries* – функции, созданные пользователем и сохраненные в библиотеке *LabVIEW*;

• *Select* *a* *VI*... – функции, созданные пользователем;

• *MakerHub* – функции, созданные сообществом *LabVIEW*.

# 1.4 Палитра инструментов



Рис. 1.6 Палитра инструментов

Создавать, изменять и отлаживать ВП можно, используя палитру инструментов (*Tools Palette*) (рис. 1.6). Палитра инструментов доступна как на лицевой панели, так и на блок-диаграмме. Термин инструмент подразумевает специальный операционный режим курсора мыши. При выборе определенного инструмента значок курсора изменяется на значок данного инструмента. Палитра инструментов вызывается одновременным нажатием клавиши <*Shift*> и правой клавиши мыши. Палитру Инструментов можно размещать в любой области рабочего пространства блок-диаграммы и лицевой панели.

Если включен автоматический выбор инструмента, то при наведении курсора на объект лицевой панели или блок-диаграммы *LabVIEW* автоматически выбирает соответствующий инструмент из палитры инструментов. Автоматический выбор инструмента включается нажатием на кнопку *Automatic Tool Selection* палитры инструментов (прямоугольник в правом верхнем углу палитры: зеленый – включен, черный – выключен) или нажатием клавиш <*Shift+Tab*>.

Инструмент *УПРАВЛЕНИЕ* (<палец>) используется для изменения значения элементов управления или ввода текста. При наведении курсора на такой элемент как строковый элемент управления, значок инструмента меняется:

Инструмент *ПЕРЕМЕЩЕНИЕ* (<стрелка>) используется для выбора, перемещения или изменения размеров объектов. При наведении инструмента на объект изменяемого размера значок инструмента меняется:

Инструмент *ВВОД ТЕКСТА* (<А>) используется для редактирования текста и создания свободных меток. При создании свободных меток значок инструмента меняется:

Инструмент *СОЕДИНЕНИЕ* (<катушка>) создает проводники данных, соединяя объекты на блок-диаграмме.

Инструмент *ВЫЗОВ КОНТЕКСТНОГО МЕНЮ* (<меню>) вызывает контекстное меню соответствующего объекта по щелчку левой кнопки мыши.

Инструмент *БЫСТРАЯ ПРОКРУТКА ЭКРАНА* (<рука>) используется для просмотра окна без использования полосы прокрутки.

Инструмент *ВВОД КОНТРОЛЬНОЙ ТОЧКИ* (<контрольная точка>) позволяет расставлять контрольные точки на ВП, функциях, узлах, проводниках данных, структурах и приостанавливать в них выполнение программы.

Инструмент *УСТАНОВКА ОТЛАДОЧНЫХ ИНДИКАТОРОВ* (<пробник>) дает возможность исследовать поток данных в проводниках блок-диаграммы

Используется для просмотра промежуточных значений при наличии сомнительных или неожиданных результатов работы ВП.

Инструмент *КОПИРОВАНИЕ ЦВЕТА* (<пипетка>) предназначен для копирования цвета с последующей вставкой с помощью инструмента РАСКРАШИВАНИЕ (<кисть>).

Инструмента *РАСКРАШИВАНИЕ* (<кисть>) позволяет изменить цвет объекта. Он также отображает текущий передний план и параметры настройки цвета фона.

Если автоматический выбор инструмента выключен, можно менять инструменты палитры инструментов с помощью клавиши <*Таb*>. Для переключения между инструментом *ПЕРЕМЕЩЕНИЕ* и *СОЕДИНЕНИЕ* на блок-диаграмме или между инструментом *ПЕРЕМЕЩЕНИЕ* и *УПРАВЛЕНИЕ* на лицевой панели - достаточно нажать пробел.

# 1.5 Справочная система в *LabVIEW*

Окно контекстной справки (*Context Help*) (рис. 1.7) выводится на экран из пункта главного меню Помощь (*Help–Show Context Help*) или вводом <*CtrI-H*> с клавиатуры.

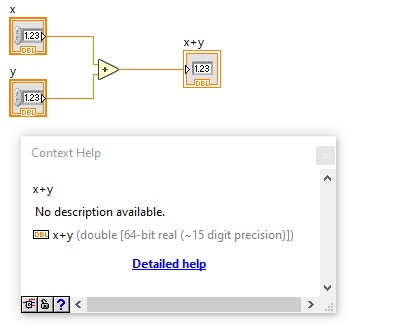


Рис. 1.7 Окно контекстной справки

При наведении курсора на объект лицевой панели или блок-диаграммы в окне контекстной справки (*Context Help*) появляются иконка подпрограммы ВП, функции, константы, элементов управления или отображения данных с указанием всех полей ввода/вывода данных. При наведении курсора на опции диалогового окна в окне контекстной справки (*Context Help*) появляется описание этих опций. При этом поля, обязательные для соединения, выделены жирным шрифтом, рекомендуемые для соединения поля представлены обычным шрифтом, а дополнительные (необязательные) поля - выделены серым или вообще не показаны.

Для отображения встроенной помощи (*LabVIEW Help*) можно нажать кнопку *Detailed Help* в окне контекстной справки (*Context Help*)

Встроенная помощь *LabVIEW* содержит детальные описания большинства палитр, меню, инструментов, виртуальных приборов и функций, включает в себя пошаговую инструкцию использования особенностей *LabVIEW* и связана с руководством пользователя (*LabVIEW Tutorial*), PDF версией учебника *LabVIEW* и технической поддержкой на Web-сайте *National Instruments*.

# 2 Примеры программ в среде *LabVIEW*

# 2.1 Первая программа

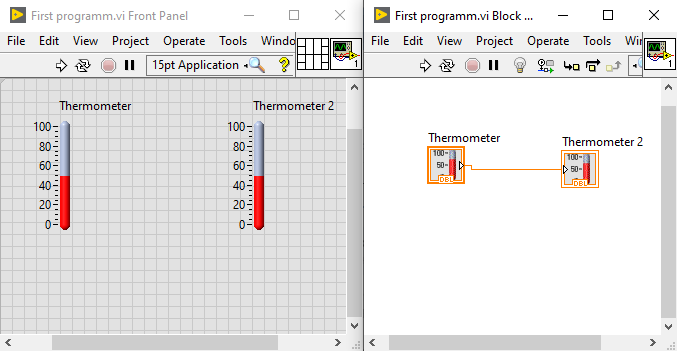


Рис.2.1 – Первая программа

Создадим нашу первую программу, которая выводит на индикатор значения с элемента управления (рис.2.1).

Для этого на лицевой панели разместим два индикатора *Thermometer* (на палитре элементов управления и индикаторов выберем: *Modern*–*Numeric*– *Thermometer*). На блок – диаграмме появились две иконки, называемые терминалами данных, которые соответствуют размещенным на лицевой панели индикаторам.

Один из индикаторов необходимо сделать элементом управления. Для этого, наведем на него курсор мышки (это можно сделать, как на лицевой панели, так и на блок – диаграмме) и, нажав правую клавишу во всплывающем меню, выберем пункт *Change to Control* (сменить на элемент управления). При этом действии на блок – диаграмме у соответствующей иконки треугольник с правой стороны переместился на левую (рис. 2.1). У индикаторов треугольник означает вход, на который необходимо подавать соответствующие значения, а у элементов управления – выход, с которого считываются значения. При подведении курсора мышки к треугольнику (выходу) элемента управления, курсор примет вид катушки. После однократного нажатия левой клавиши мышки, потяните мышку ко входу индикатора *Thermometer 2*. За курсором потянется проводник. Подведем проводник к входу индикатора и, как только курсор превратится снова в катушку, произведем однократное нажатие мышки. После этого блок диаграмма и лицевая панель будут выглядеть как на рис. 2.1.

Работоспособность программы лучше проверять нажатием кнопки *Run Continuously* (циклический запуск). При изменении мышкой значения элемента управления (*Thermometer*) изменяются значения на индикаторе (*Thermometer 2*).

**Задание:**

Остановите программу нажатием клавиши *Stop*, разорвите проводник соединяющий элемент управления и индикатор (для этого выделите его мышкой и нажмите клавишу <*Delete*>). Запустите программу циклически еще раз и сравните результат работы.

# 2.2 Программа, преобразующая градусы C в градусы F

Создадим простую программу, конвертирующую температуру представленную в градусах Цельсия в температуру по Фаренгейту, используя формулу:

, (1)

где *F* – температура по Фаренгейту, *C* – температура в градусах Цельсия.

Выполним следующие действия:

• Создайте новый виртуальный прибор (*Blank* *VI*);

• На лицевой панели разместите два индикатора *Thermometer* (на палитре элементов управления и индикаторов выберем: *Modern*–*Numeric*– *Thermometer*);

• Двойным нажатием левой клавиши мышки по меткам индикаторов выделите их и переименуйте в ― “C” и ― “F”;

• Индикатор с меткой ― “C” сделайте элементом управления как показано на рис. 2.2;

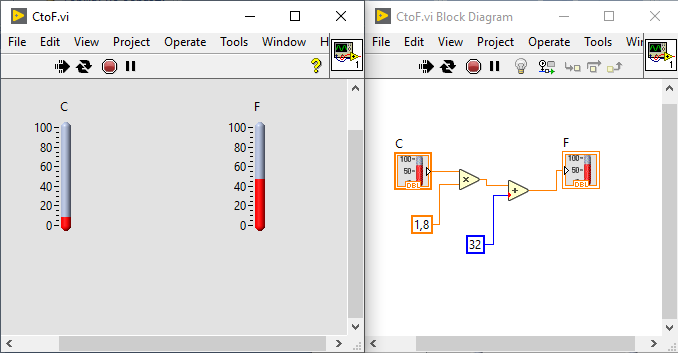


Рис.2.2 – Программа, конвертирующая температуру в градусах Цельсия в температуру по Фаренгейту

• Перейдите на блок-диаграмму.

• Выберите функции Умножение (*Multiply*) и Сложение (*Add*) из палитры функций в разделе Арифметические функции (*Programming*–*Numeric*);

• Поместите выбранные функции на блок-диаграмму.

• Создайте числовую константу у соответствующего входа функции умножения. Для этого подведите курсор мышки к этому входу, чтобы появился инструмент Катушка и однократно нажмите правую клавишу мышки. Во всплывающем меню выберите *Create*–*Constant*. После размещения числовой константы на блок-диаграмме поле ввода ее значений подсвечивается и готово для редактирования. Присвойте ей значение 1,8. Обратите внимание, что контур квадрата станет оранжевым – так отображается дробное число – число с плавающей точкой.

• Точно также создайте вторую константу и присвойте ей значение 32, как показано на рис. 2.2.

• Запустите программу, нажав клавишу *Run Continuously*.

**Задания:**

1. Измените программу так, чтобы она преобразовывала значение температуры по Фаренгейту в градусы Цельсия.

2. Создайте новую программу. На лицевой панели разместите два числовых элемента управления, назовите их ― “*X*” и ― “*Y*” и два числовых индикатора (рис. 2.3).

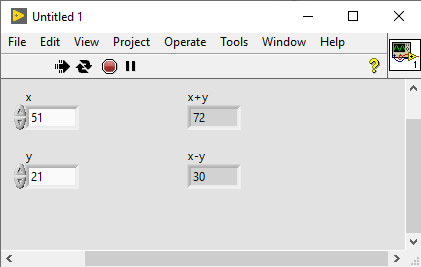


Рис 2.3 – Лицевая панель виртуального прибора

На блок-диаграмме реализуйте алгоритм, такой, чтобы на одном индикаторе выводилась сумма, а на другом разность значений, введенных в элементы управления.

# 2.3 Программа, вычисляющая заданную формулу

Задание: Ввести два числа *A* и *B*, вычислить ответ по заданной формуле и вывести на лицевую панель. Дана формула:

 (2)

Подробно рассмотрим решения данной задачи. Создадим новый прибор и добавим на лицевой панели два элемента управления *Numeric* Control, которые назовем «*A*» и «*B*» из раздела *Modern*– *Numeric* и один элемент отображения *Numeric Indicator* оттуда же, назовем «Ответ».

Перейдем к блок-диаграмме. Для вычисления формулы будем использовать функции из раздела *Mathematic– Numeric* и *Elementary/Trigonometric* и *Exponential.* Функции будем добавлять последовательно:

Соединим терминалы *A* и *B* функцией *Add*. Выход подключаем к функции *exp* из раздела *Mathematic–Elementary–Exponential*. Пока выход этой функции не трогаем.

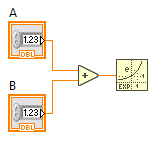


Рис 2.5 – Функция *exp*

Возведем в степень числа *A* и *B*, для этого соединим терминалы *A* и *B* и соответствующие константы 4 и 2 из раздела *Mathematic–Numeric* с функцией *x*^*y* из раздела *Mathematic–Elementary–Exponential*. Для копирования одной и той же функции можно удерживая клавишу *Ctrl* нажать ЛКМ и перенести функцию на свободное место – функция продублируется. Также обратите внимание, что у функции *x*^*y* вход *x* находится снизу, а *y* сверху.

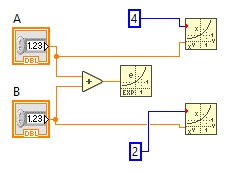


Рис 2.6 – Функция *x*^*y*

Полученные значения складываем функцией *Add*. Далее выход функции соединяем с функцией модуля *Absolute Value*, которая находится в разделе *Mathematic–Numeric.* Пока выход этой функции не трогаем.

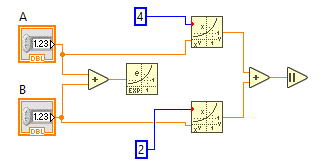


Рис 2.7 – Функция *x*^*y*

Добавим тригонометрическую функцию *atan* из раздела *Mathematic–Elementary–Trigonometric.* На её вход подадим значение с выхода функции *Add*, которая складывает значения терминалов *A* и *B*. И соединяем выход этой функции с входом *x* функции *Divide* из раздела *Mathematic–Numeric*, второй вход *y* соединим с выходом функции *Absolute Value*.

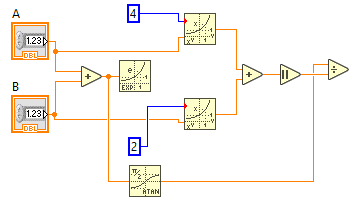


Рис 2.8 – Функция *atan* и *Divide*

Выход функции *Divide* подключим к входу тригонометрической функции *cos* из раздела *Mathematic–Elementary–Trigonometric*. Выход функции *cos* сложим функцией *Add* с математической константой π/2 из раздела *Mathematic–Numeric–Math Constants*.

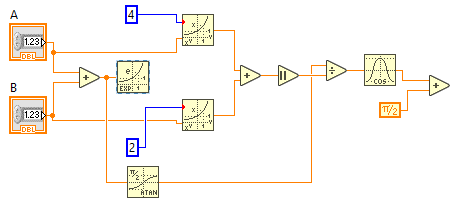


Рис 2.9 – Функция *cos* и *Add*

Выход функции *Add* и выход ранее использованной функцией *exp* перемножим функцией *Multiply*. И подключим к входу терминала *Numeric Indicator*. Проверим работоспособность программы:

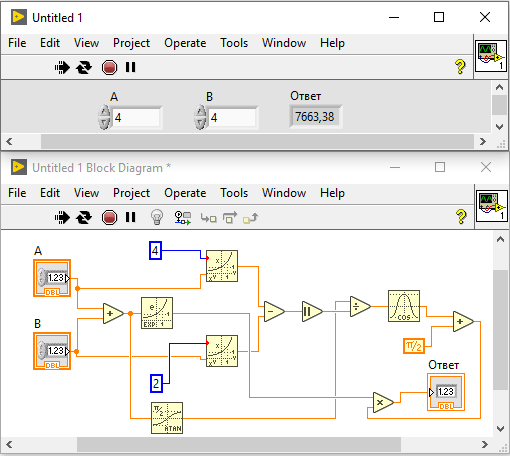


Рис 2.10 – Виртуальный прибор рассчета заданной формулы

# 2.4 Циклы

# 2.4.1 Цикл по Условию (*While*)

*Цикл по условию (While)* аналогичен циклу *While*, используемому в текстовом языке программирования *Си*, выполняет многократное повторение операции над потоком данных, пока не выполнится логическое условие выхода. Цикл *While* расположен на палитре функций в разделе Структуры (*Programming–Structures*)

После того как цикл найден и выбран на палитре функций, следует с помощью курсора изменить промежуточные границы структуры для выделения части блок-диаграммы, которую необходимо поместить в цикл. После отпускания кнопки мыши, выделенная область блок-диаграммы помещается в тело цикла. Добавление объектов блок-диаграммы в тело цикла осуществляется помещением или перетаскиванием объекта.

Блок-диаграмма цикла по условию (*While*) выполняется до тех пор, пока не выполнится условие выхода. По умолчанию, терминал условия выхода указывает, что цикл будет выполняться до поступления на терминал значения *ЛОЖЬ (FALSE*). В этом случае терминал условия выхода называется терминалом *«Продолжение Если Истина (Continue If True)»*.

Терминал счетчика итераций, показанный слева, содержит значение количества выполненных итераций. Начальное значение терминала *<i>* всегда равно нулю.

**Пример цикла *While*:**

Программа примера цикла *While* (рис. 2.11) увеличивает значение индикатора *Numeric* на 1 с интервалом в одну секунду.

Создайте программу, приведенную на рисунке 2.11. Числовой индикатор *Numeric* и кнопку *Stop* удобнее создать на блоке-диаграмм. Для этого подведите курсор мышки к соответствующему терминалу, пока курсор не превратиться в катушку, произведите однократное нажатие на правую клавишу мышки (ПКМ) и выберите *Create–Indicator* для терминала *«i»* и *Create*–*Control* для терминала условия выхода.

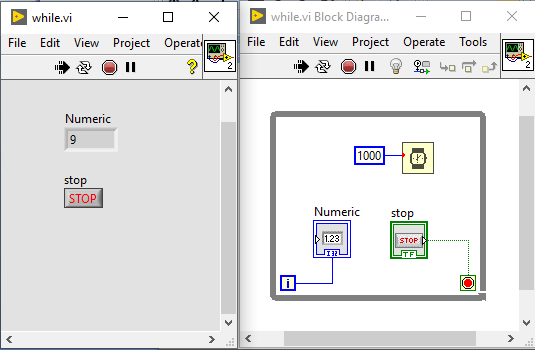


Рис 2.11 – Блок-диаграмма и лицевая панель программы с циклом *While*

Функция *Wait* находится в палитре функций в разделе *Programming*–*Timing*. Входной параметр определяет время задержки в миллисекундах. В нашем примере подключаем к функции *Wait* константу, равную 1000 *Mathematics–Numeric–Numeric Constant* для отображения задержки в секундах.

Запускать программу на выполнение следует кнопкой *Run*, расположенной на инструментальной панели, а останавливать – кнопкой *Stop*, созданной на лицевой панели.

Измените предыдущий пример так, чтобы цикл прекращал выполняться после пяти итераций, если до этого не была нажата кнопка *Stop*. Для этого следует использовать функцию *Equal* (равенство) в разделе *Programming*–*Comparison* и функцию *Or* (логическое ИЛИ) в разделе *Programming*–*Boolean*.

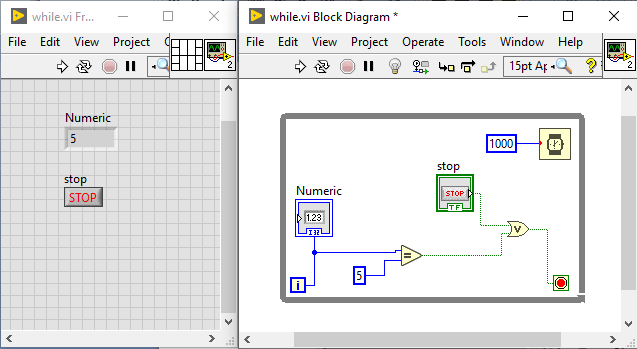


Рис 2.12 – Модификация ВП, представленного на рис. 2.11

# 2.4.2 Цикл с фиксированным числом итераций (*For*)

Цикл с фиксированным числом итераций (*For*) выполняет повторяющиеся операции над потоком данных определенное количество раз. Цикл с фиксированным числом итераций (*For*), расположен на палитре функций в разделе *Programming–Structures*. Значение, присвоенное терминалу максимального числа итераций *«N»* цикла определяет максимальное количество повторений операций над потоком данных. Терминал счетчика итераций *«i»* содержит количество выполненных итераций. Начальное значение счетчика итераций всегда равно нулю.

Цикл с фиксированным числом итераций (*For*) отличается от цикла по условию (*While*) тем, что завершает работу, выполнив заданное максимальное число итераций *«N».* Цикл по Условию (*While*) завершает работу после выполнения заданного условия выхода из цикла.

**Пример цикла for:**

Максимальное число итераций *«N»* цикла ставим 5. Терминалу счетчика итераций *«i»* добавляем индикатор: ПКМ на *«i»–Create Indicator*. Ставим функцию *Wait* с задержкой в секундах как на рис. 2.13.

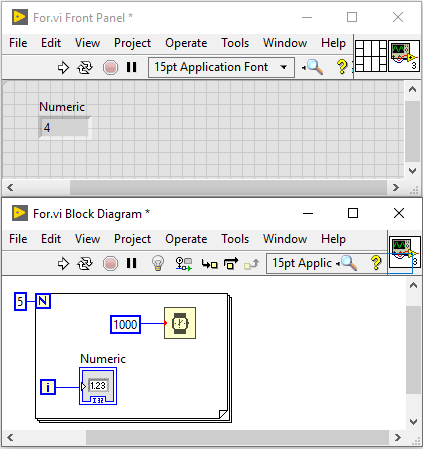


Рис 2.13 – Блок-диаграмма и лицевая панель программы с циклом *For*

Программа (рис. 2.13), также как и предыдущая, увеличивает значение индикатора *Numeric* с 0 до 4 с шагом в 1 и с интервалом в одну секунду. Распишем итерации *N* (максимальное число итераций) = 5, тогда:

• Первая итерация: *i*=0;

• Вторая итерация: *i*=1;

• Третья итерация: *i*=2;

• Четвертая итерация: *i*=3;

• Пятая итерация: *i*=4.

Таким образом, программа останавливается на пятой итерации с цифрой 4 на индикаторе – это нужно учитывать при создании своих приборов.

Изменим предыдущий пример (рис. 2.13) так, чтобы на индикатор *Numeric* выводились числа от 5 до 0 с шагом -1.

Максимальное число итераций *«N»* цикла ставим 6. Добавим функцию разности *Mathematics–Numeric–Subtract* и функцию декремента (прибавление -1 к числу) *Mathematics–Numeric–Decrement* как показано на рисунке 2.14.

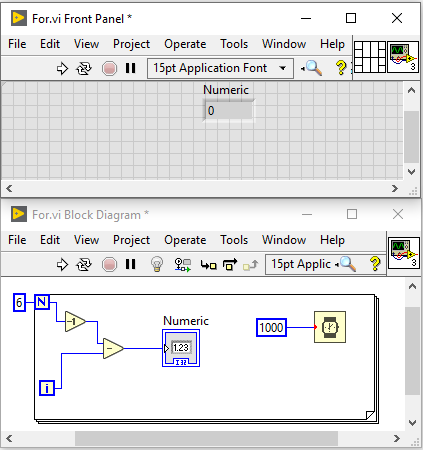


Рис 2.14 – Виртуальный прибор отображающий числа от 5 до 0

Распишем итерации, *N*=6, терминал *Numeric*=6-1-*i*,тогда:

• Первая итерация: *i*=0; *Numeric*=6-1-0=5

• Вторая итерация: *i*=1; *Numeric*=6-1-1=4

• Третья итерация: *i*=2; *Numeric*=6-1-2=3

• Четвертая итерация: *i*=3; *Numeric*=6-1-3=2

• Пятая итерация: *i*=4, *Numeric*=6-1-4=1

• Шестая итерация: *i*=5, *Numeric*=6-1-5=0

# 2.4.3 Сдвиговые регистры

Сдвиговые регистры используются при работе с циклами для передачи значений от текущей итерации цикла к следующей. Сдвиговый регистр создается щелчком правой клавиши мышки на границе цикла и выбором пункта *Add Shift Register* из контекстного меню. Сдвиговый регистр выглядит как пара терминалов; они расположены непосредственно друг против друга на противоположных вертикальных сторонах границы цикла. Правый терминал содержит стрелку <вверх> и сохраняет данные по завершению текущей итерации, *LabVIEW* передает данные с этого регистра к следующей итерации.

Сдвиговый регистр передает любой тип данных и автоматически принимает тип первых же данных, переданных на него. Данные, передаваемые на терминалы сдвигового регистра, должны быть одного типа. Предусмотрена возможность создания нескольких сдвиговых регистров в одной структуре цикла. К тому же сдвиговый регистр может иметь несколько левых терминалов сдвигового регистра для возможности работы с несколькими значениями предыдущих итераций.

Сдвиговые регистры можно использовать для запоминания значений предыдущих итераций, что полезно при создании алгоритмов усреднения. Установка дополнительных терминалов сдвигового регистра для переноса значений на следующую итерацию осуществляется щелчком правой кнопкой мышки на левом терминале и выбором *Add Element* из контекстного меню. Например, если добавить два дополнительных терминала к левому терминалу сдвигового регистра, то значения последних трех итераций поступят на текущую итерацию.

Чтобы инициализировать сдвиговый регистр, необходимо передать на его левый терминал любое значение извне тела цикла. Если не инициализировать регистр, цикл использует значение, записанное в регистр во время последнего выполнения цикла или значение, используемое по умолчанию для данного типа данных, если цикл никогда не выполнялся. Например, если тип данных сдвигового регистра логический (*Boolean*), начальное значение *ЛОЖЬ* *(FALSE).* Точно так же, если тип данных сдвигового регистра числовой, то начальное значение - 0.

Цикл с неинициализированным сдвиговым регистром используется при неоднократном запуске ВП для присвоения выходному значению сдвигового регистра значения, взятого с последнего выполнения ВП. Чтобы сохранить информацию о состоянии между последующими запусками ВП, следует оставить вход левого терминала сдвигового регистра не определенным.

**Пример цикла со сдвиговым регистром:**

Создаем новую программу и добавим в неё цикл с фиксированным числом итераций (*For*). Максимальное число итераций *«N»* цикла ставим 10. Ставим функцию *Wait* с константой = 2000 – таким образом мы добиваемся интервала 2 секунды. Создаем сдвиговый регистр щелчком ПКМ на границе цикла и выбором пункта *Add Shift Register* из контекстного меню. Добавляем индикаторы на терминалы: ПКМ на *i*, правый терминал (стрелка вверх) и левый терминал (стрелка вниз) *–Create Indicator* как показано на рис 2.15*.* Левому терминалу добавляем константу ПКМ на левый терминал (стрелка вниз) – *Create Constant* и ставим ей значение -1.

На индикаторе *Numeric* с интервалом 2 секунды будут появляться числа от 0 до 9, на индикаторе *Numeric3* с тем же интервалом будут появляться числа от -1 до 8, соответствующие числам индикатора *Numeric* на предыдущей итерации (-1 – начальное значение сдвигового регистра), на индикаторе *Numeric2* числовое значение 9 появится после выполнения всего цикла.

На рисунке 2.15 приведен пример, демонстрирующий работу сдвигового регистра.

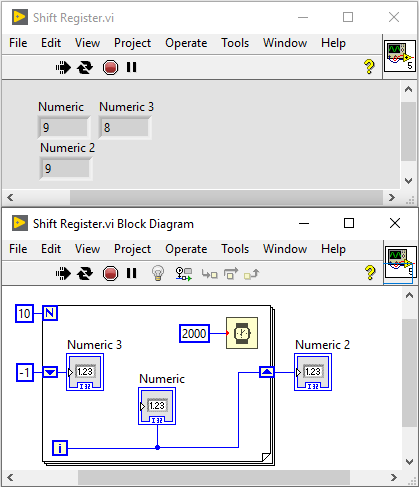


Рис 2.15 – Пример цикла со сдвиговым регистром

**Пример вычисления суммы от 1 до N**

Сделаем программу, которая, используя цикл с фиксированным числом итераций (*For*) и сдвиговые регистры (*Shift Register*), вычисляет сумму от 1 до *N*, где *N* – максимальное число итераций цикла, которое вводит пользователь.

Создаем новую программу и добавим в неё цикл *For*. К максимальному числу итераций *«N»* добавим элемент управления ПКМ на *N*–*Create Control*. Создаем сдвиговый регистр щелчком ПКМ на границе цикла и выбором пункта *Add Shift Register* из контекстного меню. Добавляем математические функции декремента *Mathematics–Numeric–Decrement* и сложения *Mathematics–Numeric–Add* как показано на рис. 2.16.

Добавляем индикатор на правый терминал (стрелка вверх) ПКМ *–Create Indicator*.Левому терминалу добавляем константу ПКМ на левый терминал (стрелка вниз) – *Create Constant* и ставим ей значение 0.

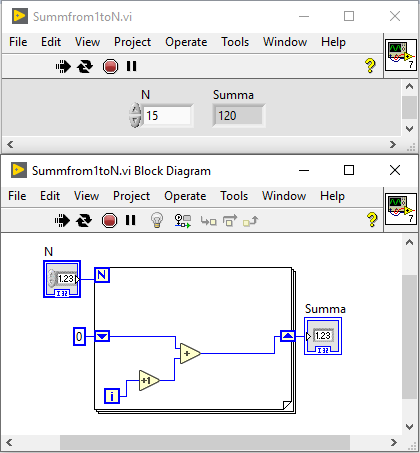


Рис 2.16 – Пример вычисления суммы от 1 до *N*

Для работы программы вводим любое целое число в элемент управления *N* и получаем ответ в индикаторе *Summa*. Например, введем число *N*=15 и получим сумму от 1 до 15, т.е. *Summa*=120.

# 3 Массивы

Массивы объединяют группу элементов одного типа данных. Массивы элементов могут иметь разную размерность. Элементами массива называют группу составляющих его объектов. Размерность массива это совокупность столбцов (длина) и строк (высота). Глубина – общее количество элементов в массиве. Массив может иметь одну и более размерностей, до 231 элементов в каждом направлении, насколько позволяет оперативная память.

Данные, составляющие массив, могут быть любого типа: целочисленного, логического, строкового (*integer*, *boolean*, *string*).

Использование массивов удобно при работе с группами данных одного типа и при накоплении данных после повторяющихся вычислений. Массивы идеально подходят для хранения данных, накопленных во время работы циклов, при этом одна итерация цикла создает один элемент массива.

Все элементы массива упорядочены. Каждому элементу массива присвоен индекс, что обеспечивает легкий к нему доступ. Индекс первого элемента массива всегда 0. Таким образом, индексы массива находятся в диапазоне от 0 до *N*-1, где *N* – число элементов в массиве. Например, для *N*=10, индекс находится в пределах от 0 до 9.

# 3.1 Создание массива элементов управления и отображения

Для создания массива элементов управления или отображения, как показано на примере на рис. 3.1, необходимо выбрать его шаблон из палитры элементов (*Controls*) в разделе «Хранилища данных» (*Modern*–*Data Containers–Array*), поместить его на лицевую панель. Затем поместить в шаблон массива элемент управления *Numeric Control* либо отображения *Numeric Indicator*. Для просмотра и введения значений массива нужно использовать переключатель индексов массива слева. Чтобы посмотреть весь массив целиком нужно подвести курсор к границе массива и растянуть.

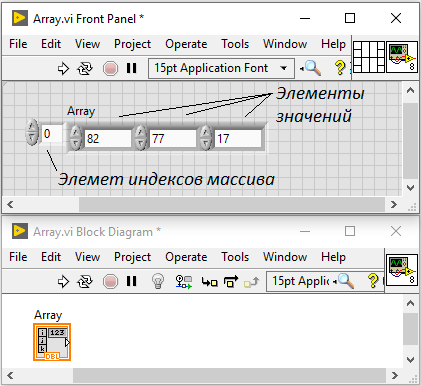


Рис 3.1 – Массив числовых элементов управления

Если производится попытка поместить в шаблон массива такой элемент управления или отображения, как двухкоординатный график осциллограмм (*XY graph*), то разместить такой элемент не удастся.

Поместить объект в шаблон массива следует до того, как он будет использоваться на блок- диаграмме. Иначе на блок-диаграмме шаблон массива не будет инициализирован.

# 3.2 Двумерные массивы

Двумерный (*2D*) массив хранит элементы в виде матрицы. Таким образом, для размещения элемента требуется указание индекса столбца и строки. На иллюстрации показан двумерный массив, состоящий из 2 столбцов (длина) и 3 строк (высота). Количество элементов в массиве (глубина) – .

Для добавления массиву размерности необходимо щелкнуть правой кнопкой мыши на элементе индекса и выбрать из контекстного меню пункт Добавить Размерность (*Add Dimension*). Также для добавления размерности можно использовать инструмент ПЕРЕМЕЩЕНИЕ (<стрелка>), изменив размер элемента индекса.

Для создания двумерного массива, показанного на рис. 3.2, выбираем шаблон массива из палитры элементов (*Controls*) в разделе *Modern*–*Data Containers–Array* и помещаем его на лицевую панель. Добавляем еще одну размерность ПКМ на элементе индекса и выбираем *Add Dimension*. Для увеличения количества элементов массива подводим курсор к границе массива и растягиваем его на нужное количество элементов, например на 6, и заполняем элементы числовыми значениями, как показано на рис. 3.2

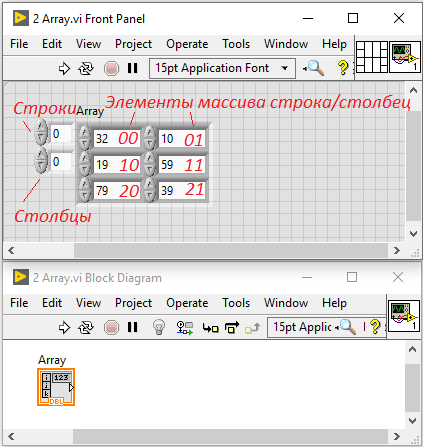


Рис. 3.2 Двумерный массив числовых элементов управления

Распишем получившийся массив. Так как индекс первого элемента массива всегда 0, тогда:

Первая строка, первый столбец, Элемент 00=32

Вторая строка, первый столбец, Элемент 10=19

Третья строка, первый столбец, Элемент 20=79

Первая строка, второй столбец, Элемент 01=10

Вторая строка, второй столбец, Элемент 11=59

Третья строка, второй столбец, Элемент 21=39

# 3.3 Создание массива констант

На блок-диаграмме создать массив констант можно, выбрав в палитре функций в разделе Программирование (*Programming*) – Массив (*Array*) – Шаблон Массива Констант (*Array Constant*) и поместив в него числовую константу *Mathematics–Numeric–Numeric Constant* или другой объект данных (*boolean, string*).

Сделаем новый прибор. Добавим на блок-диаграмме массив констант *Programming–Array–Array Constant.* Затем поместить в шаблон массива константу *Mathematics–Numeric–Numeric Constant*. Для просмотра и введения значений массива нужно использовать переключатель индексов или нужно подвести курсор к границе массива и растянуть как на рисунке 3.3.

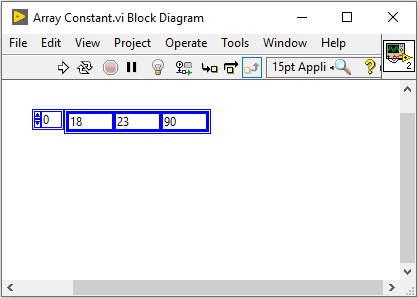


Рис. 3.3 Пример массива констант

# 3.4 Создание массива с помощью цикла

Цикл с фиксированным числом итераций (*For*) и цикл по условию (*While*) могут автоматически накапливать массивы и проводить их индексацию на своих границах. Это свойство называется автоиндексацией, После соединения терминала данных массива с терминалом выхода из цикла каждая итерация цикла создает новый элемент массива.

Сделаем новый прибор. Добавим на лицевой панели массив *Modern*–*Data Containers–Array.* Затем поместим в шаблон массива индикатор *Numeric Indicator.* На блок-диаграмме добавим цикл *For Programming–Structures–For Loop*, ввести значение максимального количества итераций *N*=5 и подключить счетчик итераций *i* к массиву *Array* как показано на рисунке 3.4. Автоиндексация по-умолчанию включена в цикле *For*, поэтому не изменяем этот параметр.

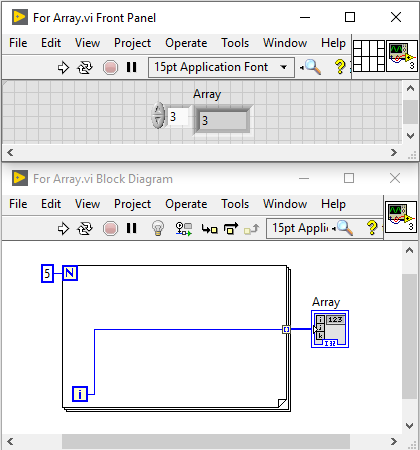


Рис 3.4 – Пример создания массива с помощью цикла

На рис. 3.4 видно, что проводник данных, соединяющий терминал данных массива с терминалом выхода из цикла стал толще, а сам терминал выхода из цикла окрашен в цвет терминала данных массива.

Автоиндексация отключается щелчком правой кнопки мыши на терминале входа/выхода из цикла и выбором Отключить Автоиндексацию (*Disable Indexing*) из контекстного меню. Автоиндексация отключается в случае, когда на элемент отображения необходимо передать только последнее значение.

Примечание. Ввиду того, что цикл с заданным числом итераций (*For*) часто используется для создания циклов, *LabVIEW* осуществляет автоиндексацию автоматически. Автоиндексация для цикла с заданным числом итераций (*For*) включена по умолчанию. Для цикла по условию (*While*) по умолчанию автоиндексация отключена. Чтобы включить автоиндексацию, необходимо щелкнуть правой кнопкой мыши на терминале входа/выхода из цикла и выбрать в контекстном меню Включить Автоиндексацию (*Enable Indexing*).

# 3.5 Создание двумерных (2D) массивов с помощью цикла

Для создания двумерных (2D) массивов необходимо использовать два цикла с заданным числом итераций (*For*), один внутри другого.

Сделаем пример виртуального прибора, который формирует квадратную матрицу – двумерный массив 5x5, заполненный числами от 1 до 25.

Создадим новый прибор и добавим на лицевой панели массив *Modern*–*Data Containers–Array.* Затем поместим в шаблон массива индикатор *Numeric Indicator.* Добавляем еще одну размерность ПКМ на элементе индекса и выбираем *Add Dimension*.На блок-диаграмме добавим два цикла *For Programming–Structures–For Loop*, один внутри другого. Вводим значение максимального количества итераций *N*=5 и подключаем их и счетчики итераций *i* к массиву *Array,* используя функции *Add*, *Multiply* и *Increment* как показано на рисунке 3.5.

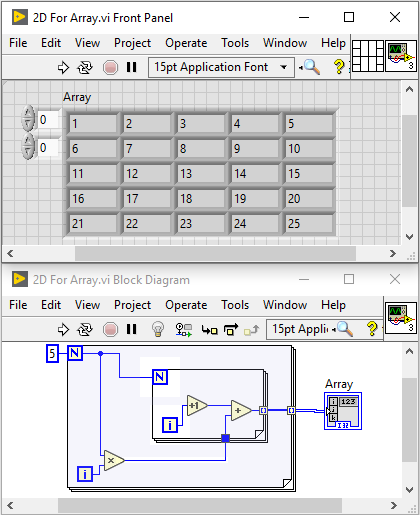


Рис 3.5 – Программа создающая двумерный массив 5x5, заполненный числами от 1 до 25

Примем, что у внешнего цикла *i1* и *N1*, а у внутреннего цикла *i2* и   
*N2*=*N1*. Тогда запишем формулу, используемую в программе:

, (3)

где *i1* – номер строки элемента массива, *i2* – номер столбца элемента массива, *Array*[*i1*, *i2*] – элемент массива, находящийся в *i1* строке и *i2* столбце.

Рассмотрим принцип работы программы по итерациям (*N2*=*N1*=5):

• Первая итерация: *i*1 =0, *i*2 =0: 

• Вторая итерация: *i*1 =0, *i*2 =1: 

• Третья итерация: *i*1 =0, *i*2 =2: 

• Четвертая итерация: *i*1 =0, *i*2 =3: 

• Пятая итерация: *i*1 =0, *i*2 =4: 

• Шестая итерация: *i*1 =1, *i*2 =0: 

• Седьмая итерация: *i*1 =1, *i*2 =1: 

И так далее по аналогии, увеличивая *i1* и *i2* от 0 до 4получаем двумерный массив 5x5 (квадратную матрицу), заполненный числами от 1 до 25.

# 3.6 Пример вычисления чисел Фибоначчи

Числа Фибоначчи – это числа, вычисляемые по следующему алгоритму: каждое новое число вычисляется как сумма двух предыдущих. Первые два числа обычно берут 0 и 1.

Сделаем пример виртуального прибора, который вычисляет 6 первых чисел Фибоначчи.

Создадим новый прибор и добавим на лицевой панели массив *Modern*–*Data Containers–Array.* Затем поместим в шаблон массива индикатор *Numeric Indicator*.На блок-диаграмме добавим цикл *For Programming–Structures–For Loop*. Вводим значение максимального количества итераций *N*=6. Создаем сдвиговый регистр щелчком ПКМ на границе цикла и выбором пункта *Add Shift Register* из контекстного меню. Добавляем еще один левый терминал к сдвиговому регистру, для этого нажимаем на левом терминале (стрелка вниз) и выбираем пункт *Add Element*.После этого, только созданным левым терминалам (стрелки вниз) добавляем константы 0 и 1для этого ПКМ на левый терминал (стрелка вниз) – *Create Constant* и ставим ей значения 0 и 1 соотственно.Соединяем терминалы сдвигового регистра, используя функцию Суммирования *Add* из раздела *Mathematics–Numeric*, с массивом данных *Array*, как показано на рис. 3.6.

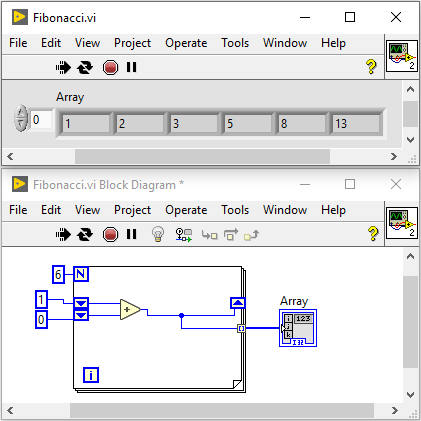


Рис 3.6 – Программа вычисляющая 6 первых чисел Фибоначчи

# 3.7 Функции работы с массивами

# 3.7.1 *Array Size*

Для создания и управления массивами используют функции, расположенные на палитре функций в разделе Массивы (*Programming–Array*). Функции обработки массивов включают в себя:

Размерность массива (*Array Size*) – показывает количество элементов массива в каждой размерности. Если массив *n*-мерный, на выходе функции *Array Size* будет массив из *n* элементов.

Сделаем пример виртуального прибора, который вычисляет размерность (количество элементов) массива, используя функцию *Array Size*.

Создадим новый прибор и добавим на блок-диаграмме массив констант *Programming–Array–Array Constant.* Затем растянем шаблон массива и запишем в первые три элемента значения как показано на рис. 3.7.Затем добавим на лицевую панель элемент отображения (индикатор) *Modern*–*Numeric*–*Numeric Indicator*. На блок-диаграмме добавим функцию *Programming–Array–Array Size* и соединим массив с терминалом индикатора, посредством функции *Array Size*, как показано на рис 3.7. В приведенном примере функция *Array Size* выдаст на индикаторе значение 3, что указывает на количество элементов нашего массива.

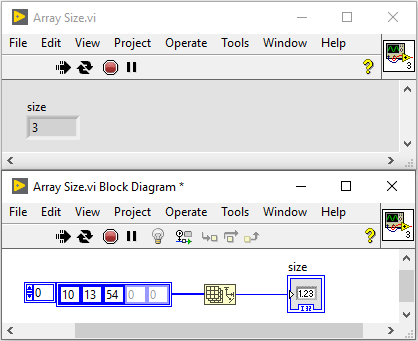


Рис 3.7 – Программа вычисляющая размер массива с функцией *Array Size*

# 3.7.2 *Initialize Array*

Инициализация массива (*Initialize Array*) – создает *n*-мерный массив, в котором каждый элемент инициализирован значением поля ввода данных «*element*». Для увеличения размерности массива достаточно добавить поля ввода данных, растянув функцию.

Например, функция Инициализация массива (*Initialize Array*) с заданными полями ввода данных – в поле «элемент» («*element*») значение 1, в поле размерность (*dimension size*) значение 3 и при наличии одного поля ввода данных размерность (*dimension size*) - выдаст массив, показанный на рис. 3.8.

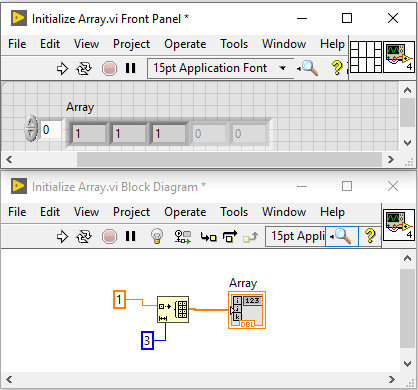


Рис 3.8 – Пример массива с функцией *Initialize Array*

Сделаем пример виртуального прибора, который инициализирует одномерный массив с указанием количества элементов и заполнением их значениями, используя функцию *Initialize Array*.

Создадим новый прибор и добавим на лицевой панели массив *Modern*–*Data Containers–Array* и растянем шаблон массива на 5 элементов.Также на лицевую панель добавим два элемента управления *Modern*–*Numeric*–*Numeric Control*, которые назовем как «*element*» –содержащее значение элементов массива и *«dimension size*» – содержащее количество элементов в массиве.

На блок-диаграмме добавим функцию *Programming–Array– Initialize Array* и соединим терминалы элементов управления с массивом, посредством функции *Initialize Array*, как показано на рис 3.9.

Программу запускаем в цикличном режиме *Run Continuosly*. Вводим нужные нам значения, например 5 и 4, в элементы управления на лицевой панели и получаем одномерный массив с заданным числом элементов и их значением.

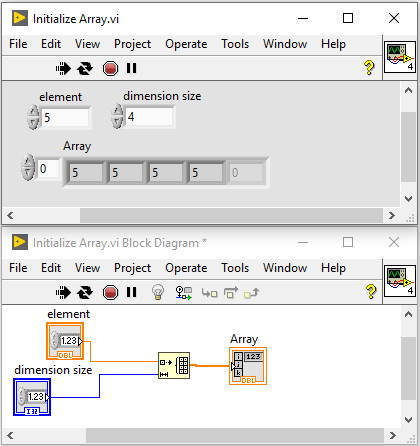


Рис 3.9 – Пример массива с указанием количества элементов и их значениями с функцийей *Initialize Array*

# 3.7.3 *Build Array*

Компоновка массива (*Build Array*) – объединяет несколько массивов или добавляет элемент в *n*-мерный массив. Изменение размера функции увеличивает количество полей ввода данных, что позволяет увеличить количество добавляемых элементов. Например, если объединить два массива, которые изображены на рис. 3.7 и 3.8 то функция Компоновка массива (*Build Array*) выдаст на выходе массив, показанный на рис. 3.10.

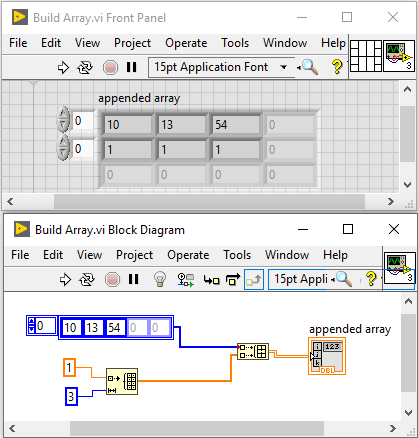


Рис 3.10 – Пример использования функции *Build Array*

Для объединения входных данных в более длинный массив той же размерности, достаточно щелкнуть правой кнопкой мыши на функции и выбрать из контекстного меню пункт Объединить вводы (*Concatenate Inputs*).

# 3.7.4 *Array Subset*

Подмножество массива (*Array Subset*) – выдает часть массива, начиная с индекса, введенного в поле *index*, и длиной, указанной в поле *length*.

Сделаем пример виртуального прибора, который выделяет часть массива, начиная с четвертого элемента длиной два элемента.

Создадим новый прибор и добавим на блок-диаграмме массив констант *Programming–Array–Array Constant.* Затем растянем шаблон массива и запишем значения шести элементов как показано на рис. 3.11.Затем добавим на лицевую панель массив с названием «subarray» *Modern*–*Data Containers–Array* и растянем его на четыре элемента. На блок-диаграмме добавим функцию *Programming–Array–* *Array Subset* и соединим массив с терминалом массива «subarray», посредством функции *Array Subset*, как показано на рис 3.11. У функции *Array Subset* есть еще два входа – *index* (индекс начального элемента) и *length* (длина выделяемого участка массива). Подсоединяем к ним константы 3 и 2 соответственно. Можно воспользоваться палитрой функций *Mathematics–Numeric–Numeric Constant* или подвести курсор к нужному входу функции *Array Subset* и нажать ПКМ–*Create Constant*.

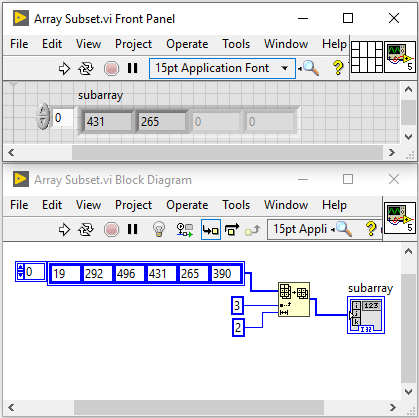


Рис 3.11 – Пример использования функции *Array Subset*

# 3.7.5 *Index Array*

Элемент массива по индексу (*Index Array*) – выдает элемент, соответствующий индексу, номер которого введен в поле *index*. Рассмотрим пример использования функции *Index Array*, при использовании предыдущего массива.

Откроем предыдущий проект *Array Subset*, на блок-диаграмме удалим массив subarray и заменим функцию *Array Subset* на функцию *Index Array: Programming–Array–Index Array*.

На лицевой панели добавим один элемент управления, назовем «*n*», и один элемент отображения, назовем «*n*-элемент». Соединим на блок-диаграмме терминал элемента управления с входом *index* функции *Index Array*, массив из предыдущего примера соединим с другим входом *array* функции *Index Array*. Выход функции *Index Array* подключим к терминалу элемента отображения как показано на рис. 3.12 и проверим работоспособность программы, запустив в цикле *Run Continuously*.

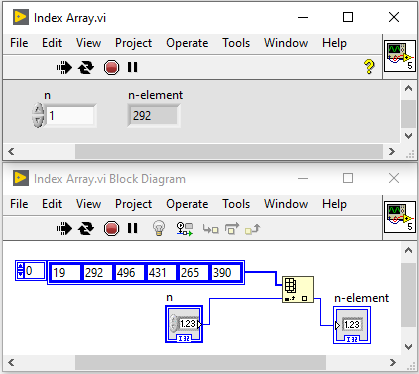


Рис 3.12 – Пример использования функции *Index Array*

Таким образом, меняя число в элементе управления «*n*» на элементе отображения показывается элемент массива, который находится на *n*-ом месте в массиве.

Функцию *Index Array* можно использовать для выделения строки или столбца из двумерного массива и дальнейшего отображения в виде подмассива. Для этого двумерный массив надо подать в поле ввода данных функции. Функция *Index Array* должна иметь два поля *index*. Верхнее поле *index* указывает строку, а нижнее поле *index* указывает столбец. Можно задействовать оба поля *index* для выбора отдельного элемента или только одно, для выбора строки или столбца.

Сделаем пример виртуального прибора, который выделяет вторую строку из двумерного массива.

Создадим новый прибор и добавим на блок-диаграмме массив констант *Programming–Array–Array Constant.* Затем добавляем массиву размерность - щелкаем ПКМ на элементе индекса и выбираем из контекстного меню пункт Добавить Размерность (*Add Dimension*), а затем растянем шаблон массива и запишем значения двенадцати элементов как показано на рис. 3.13.

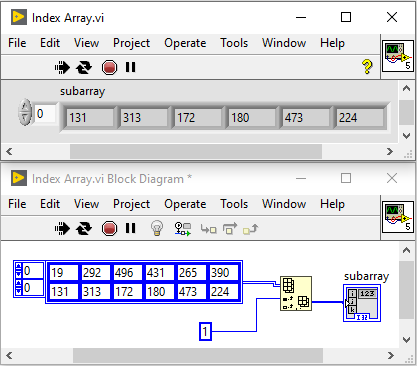


Рис 3.13 – Пример виртуального прибора, который выделяет вторую строку из двумерного массива

Добавим на лицевую панель массив с названием «*subarray*» *Modern*–*Data Containers–Array* и растянем его на шесть элементов. На блок-диаграмме добавим функцию *Programming–Array–* *Index Array* и соединим массив с терминалом массива «*subarray*», посредством функции *Index Array*, как показано на рис 3.13. На второй вход – *index* подсоединяем константу 1, для этого можно воспользоваться палитрой функций *Mathematics–Numeric–Numeric Constant* или подвести курсор к нужному входу функции *Array Subset* и нажать ПКМ–*Create Constant*. Отсчет строк в массивах начинается с 0, поэтому при запуске программы в массиве *subarray* отображается вторая строка.

# 3.7.6 *Search 1D Array*

Поиск в одномерном массиве (*Search 1D Array*). Данная функция выполняет поиск заданного элемента в одномерном массиве начиная с заданного индекса. Входными значениями данной функции являются: верхний вход (*1D array*) – массив, в котором производится поиск, средний вход (*element*) – значение которое необходимо найти, нижний вход (*start index*) – с какого индекса массива начинать поиск. Функция возвращает (*index of element*) индекс элемента массива значение, которого совпадает со значением на входе ― *element*. Если элемент не найден, то возвращается значение -1.

Пример виртуального прибора, использующего данную функцию, приведен на рис. 3.14.

Данный виртуальный прибор создает массив из 1000 целых случайных чисел в диапазоне от 0 до 100, в котором ищется первое вхождение числа 55. Функция Генератор случайных чисел от 0 до 1 (*Random Number* (0-1)) расположена в палитре функций в разделе *Programming*–*Numeric* и функция преобразования в 8-разрядное целое беззнаковое число (*To Unsigned Byte Integer*) расположена в палитре функций в разделе *Programming*–*Numeric*–*Conversion*.

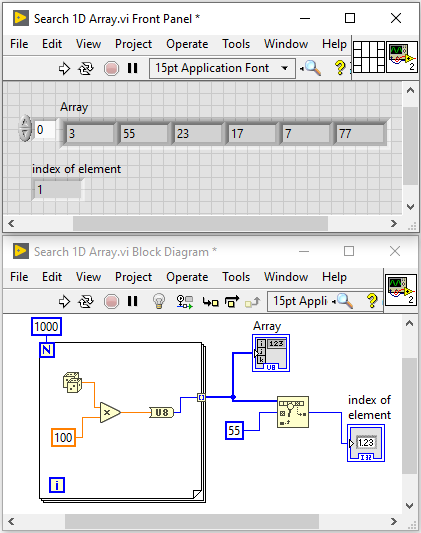


Рис 3.14 – Пример №1 виртуального прибора, использующего *Search 1D Array*

Сделаем пример виртуального прибора, в котором реализован поиск всех индексов массива, значение которых совпадает с заданным значением.

Создадим новый прибор и добавим на блок-диаграмме два цикла *For* и *While* из раздела *Programming–Structures.* Затем в цикле *For* добавляем функцию Генератор случайных чисел от 0 до 1 (*Random Number* (0-1)), из раздела *Programming*–*Numeric.* Там же в цикле добавим константу *Numeric Constant* из раздела *Mathematics–Numeri* и присвоим ей значение 100. Соединим константу и генератор случайных чисел функцией умножения *Multiply* из раздела *Programming*–*Numeric* как показано на рис. 3.15. Далее преобразуем выходные данные с помощью функции преобразования в 8-разрядное целое беззнаковое число (*To Unsigned Byte Integer*) из раздела *Programming*–*Numeric*–*Conversion*. Выводим полученные значения за цикл *For* в терминал массива *Array*.

В цикл *While* добавляем функцию поиска в одномерном массиве *Search 1D Array* из раздела *Programming–Array*, функцию инкремента *Increment* из раздела *Programming*–*Numeric* и функцию сравнения <0 Less Than 0? из раздела *Programming–Comparison*. Добавляем сдвиговый регистр щелчком ПКМ на границе цикла *While* и выбором пункта *Add Shift Register* из контекстного меню. На вход *Element* функции *Search 1D Array* добавляем константу со значением 55. Оставшиеся элементы соединяем, как показано на рисунке 3.15.

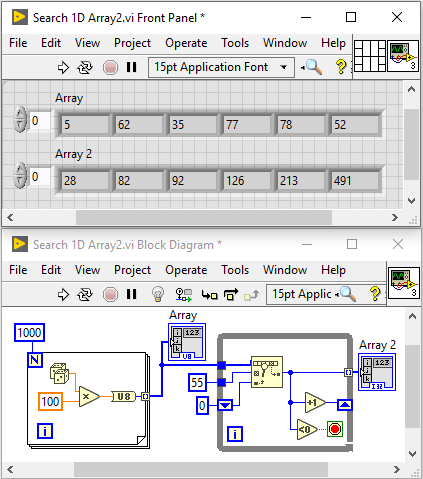


Рис 3.15 – Пример №2 виртуального прибора, использующего *Search 1D Array*

Запускаем программу однократно *Run*. Итого в массиве *Array2* отображаются индексы элементов, значение которых равны 55. Например, в нашем примере в массиве *Array* элементы со значением 55 стоят на позициях с индексом 28, 82, 92, 126, 213, 491 и т.д.

# 3.8 Передача массива данных в цикл

Если протянуть проводник от массива, расположенного на блоке диаграмм вне цикла до границы цикла, то на границе цикла образуется тоннель (рис. 3.16 и рис. 3.17). Тоннель может быть индексированным   
(рис. 3.16) и неиндексированным (рис. 3.17.).

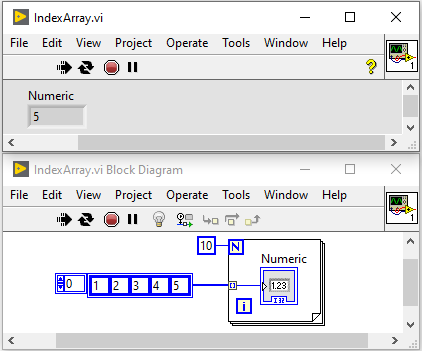


Рис. 3.16 – Пример передачи массива данных в цикл *for* через индексированный тоннель

В случае неиндексированного тоннеля массив передается в цикл целиком, в случае индексированного – поэлементно (на каждой итерации цикла из массива берется одно значение с индексом, совпадающим со значением переменной цикла *i*). Тоннель можно сделать индексированным и неиндексированным. Для этого необходимо подвести к нему курсор мышки и однократно нажать правую клавишу мышки. Во всплывающем меню выбрать пункт *Enable Indexing* (Включить индексацию) или *Disable Indexing* (Выключить индексацию).

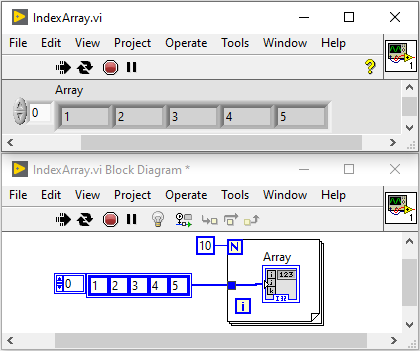


Рис. 3.17 – Пример передачи массива данных в цикл *for* через неиндексированный тоннель

Заметим, что если используется индексированный тоннель, то устанавливать значение *N* необязательно. Количество итераций, которое будет выполнять цикл, будет равно размерности массива. На рис. 3.16 цикл выполнится 5 раз. Если используется несколько индексированных тоннелей и задается значение *N*, то количество итераций цикла будет соответствовать наименьшему массиву или значению *N*, если оно меньше размерности массивов переданных в цикл через индексированные тоннели.

# 3.8.1 Пример ВП с входным индексированным тоннелем

Сделаем виртуальный прибор, в котором создается массив из 1000 целых случайных чисел в диапазоне от 0 до 100, а затем ищется первое вхождение числа 55.

Создадим новый прибор и добавим на лицевой панели массив *Array* из раздела *Data Containers*–*Array* и элемент отображения *Numeric Indicator* из раздела *Modern*–*Numeric*. На блок-диаграмме сделаем два цикла *For* и *While* из раздела *Programming–Structures.* Затем в цикле *For* добавляем функцию Генератор случайных чисел от 0 до 1 (*Random Number* (0-1)), из раздела *Programming*–*Numeric.* Там же в цикле добавим константу *Numeric Constant* из раздела *Mathematics–Numeric* и присвоим ей значение 100. Соединим константу и генератор случайных чисел функцией умножения *Multiply* из раздела *Programming*–*Numeric* как показано на рис. 3.18. Далее преобразуем выходные данные с помощью функции преобразования в 8-разрядное целое беззнаковое число (*To Unsigned Byte Integer*) из раздела *Programming*–*Numeric*–*Conversion*. Выводим полученные значения за цикл *For* в терминал массива *Array*.

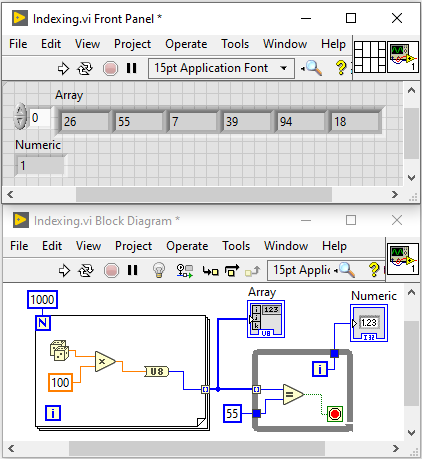


Рис. 3.18 – Пример использования входного индексированного тоннеля

По-умолчанию тоннель на выходе из цикла *for* стоит индексированный (имеет светлый цвет), если по каким-либо причинам этого не произошло, нужно подвести к нему курсор мышки и однократно нажать ПКМ, далее во всплывающем меню выбрать пункт *Enable Indexing* (Включить индексацию).

В цикл *While* добавляем функцию сравнения Equal? из раздела *Programming–Comparison*, на первый вход которой подключаем выход массива, как показано на рис. 3.18, на второй – константу со значением 55, а выход соединим с красной кнопкой – элементом остановки цикла *While*. Номер итерации *i* соединим с терминалом элемента отображения Numeric.

В отличии от виртуального прибора изображенного на рис. 3.14, в данном приборе не используется функция *Search 1D Array*.

# 3.8.2 Виртуальный прибор поиска максимального значения

Сделаем виртуальный прибор, в котором создается массив из 1000 дробных случайных чисел в диапазоне от 0 до 100, а затем ищется максимальное число, которое выводится на элемент отображения.

Создадим новый прибор и добавим на лицевой панели массив *Array* из раздела *Data Containers*–*Array* и элемент отображения *Numeric Indicator* из раздела *Modern*–*Numeric*. На блок-диаграмме сделаем два цикла *For* из раздела *Programming–Structures.* Затем в цикле *For* добавляем функцию Генератор случайных чисел от 0 до 1 (*Random Number* (0-1)), из раздела *Programming*–*Numeric.* Там же в цикле добавим константу *Numeric Constant* из раздела *Mathematics–Numeric* и присвоим ей значение 100. Соединим константу и генератор случайных чисел функцией умножения *Multiply* из раздела *Programming*–*Numeric* как показано на рис. 3.19

Выводим полученные значения из первого цикла *For* в терминал массива *Array*. По-умолчанию терминал массива *Array* имеют размерность DBL, если по каким-либо причинам там стоит другая размерность, тогда подводим курсор к терминалу массива, нажимаем ПКМ, далее во всплывающем меню наводим курсор на пункт *Representation* и выбираем DBL. В свою очередь для целых чисел следует выбирать I8-I64 или U8-U64. В этом первом цикле *For* создается массив из 1000 числовых элементов со значениями расположенными в диапазоне от 0 до 100.

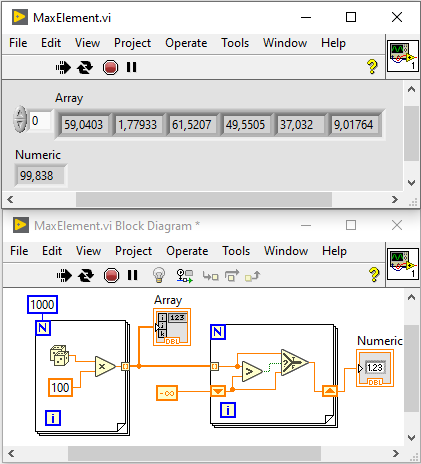


Рис. 3.19 – Виртуальный прибор поиска максимального значения

Во втором цикле *For* реализуем алгоритм поиска. Для этого добавим функцию выбора *Select*, которая возвращает значение, соответствующее верхнему входу (*t*), если на средний вход (*s*) подано значение TRUE или соответствующее нижнему входу (*f*) , если на средний вход (*s*) подано значение FALSE. Функция *Select* расположена в палитре функций в разделе *Programming*–*Comparison*. Для использования функции *Select* добавив функцию сравнения *Greater?*, которая сравнивает значения на входах *x* и *y*, и при условии, что *x*>*y* – выдает на выходе значение *TRUE*, в ином случае – *FALSE*. Функция *Greater?* расположена в разделе *Programming*–*Comparison.*

Создаем сдвиговый регистр на втором цикле *For* щелчком ПКМ на границе цикла и выбором пункта *Add Shift Register* из контекстного меню. Также добавляем числовую константу *Negative Infinity*, соответствующую минимальному числовому значению, которая расположена в палитре функций в разделе *Programming*–*Numeric*. Соединяем элементы, как показано на рисунке 3.19.

# 3.9 Полиморфизм

Арифметические функции, расположенные на палитре функций в разделе Арифметические функции (*Numeric*) полиморфны. Это означает, что поля ввода данных этих функций могут различаться по структуре данных (скалярные величины, массивы). Например, можно использовать функцию Сложение (*Add*) для прибавления скалярной величины к массиву или сложения двух массивов. Если в одно поле ввода данных функции Сложение (*Add*) подать скалярную величину 2, а другое соединить с массивом, то функция прибавит 2 к каждому элементу массива.

Если на вход функции Сложение (*Add*) подать два предыдущих массива, функция сложит каждый элемент первого массива с соответствующим элементом второго и выдаст результат в виде массива.

Если с помощью функции Сложение (*Add*) сложить два массива разной размерности, то функция сложит каждый элемент первого массива с соответствующим элементом второго и выдаст результат в виде массива размерностью меньшей из двух.

# 4 Отображение данных в графической среде *LabVIEW*

# 4.1 Использование графиков для отображения данных

С помощью графиков виртуальный прибор обычно отображает накопленные в массив данные в виде осциллограмм. На рис. 4.1 представлен один из элементов графики: график осциллограмм (*Waveform Graph*).



Рис. 4.1 – Графический элемент отображения *Waveform Graph*

График осциллограмм расположен на палитре элементов управления и отображения в разделе Графики (*Modern*–*Graph*). График осциллограмм (*Waveform Graph*) отображает только однозначные функции, такие как *у*=*f*(*x*), с точками, равномерно распределенными по оси *X*, Двухкоординатный график осциллограмм (*XY Graph*) отображает любой набор точек, будь то равномерно распределенная выборка или нет.

Для отображения множества осциллограмм необходимо изменить размер панели управления свойствами осциллограмм (*Plot legend*). График множества осциллограмм используется с целью экономии пространства на лицевой панели и для сравнения осциллограмм данных между собой. График осциллограмм (*Waveform Graph*) и Двухкоординатный график осциллограмм (*X*-*Y Graph*) автоматически поддерживают режим отображения множества осциллограмм.

# 4.1.1 Однокоординатные графики

*Одиночный график осциллограмм* работает с одномерными массивами и представляет данные массива в виде точек на графике, с приращением по оси *X* равным 1 и началом в *Х*=0.

*График множества осциллограмм* работает с двумерными массивами данных, где каждая строка массива есть одиночная осциллограмма данных и представляет данные массива в виде точек на графике, с приращением по оси *X* равным 1 и началом в *Х*=0.

Для представления каждого столбца двумерного массива данных в виде осциллограммы на графике необходимо соединить терминал данных массива с терминалом приема данных графика, затем щелкнуть правой кнопкой мыши в поле графика и выбрать из контекстного меню пункт *Транспонирование Массива* (*Transpose Array*).

Сделаем виртуальный прибор, в котором создается двумерный массив из 100 целых случайных чисел в диапазоне от 0 до 100, и выведем их в виде графиков в элементе отображении *Waveform Graph.*

Создадим новый прибор и добавим на лицевой панели массив *Array* из раздела *Data Containers*–*Array* и элемент отображения *Waveform Graph* из раздела *Modern*–*Graph*. На блок-диаграмме сделаем два цикла *For* из раздела *Programming–Structures.* Затем во внутреннем цикле *For* добавляем функцию Генератор случайных чисел от 0 до 1 (*Random Number* (0-1)), из раздела *Programming*–*Numeric.* Там же в цикле добавим константу *Numeric Constant* из раздела *Mathematics–Numeric* и присвоим ей значение 100. Соединим константу и генератор случайных чисел функцией умножения *Multiply* из раздела *Programming*–*Numeric* как показано на рис. 4.2. Максимальное количество итераций *N* ставим равное 100.

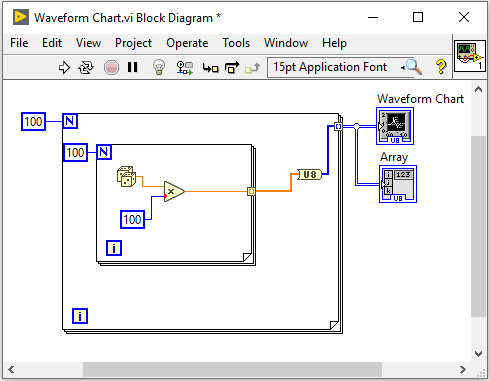


Рис. 4.2 – Блок-диаграмма виртуального прибора с *Waveform Graph*

Выводим данные во внешний цикл *For*, где преобразуем выходные данные с помощью функции преобразования в 8-разрядное целое беззнаковое число (*To Unsigned Byte Integer*) из раздела *Programming*–*Numeric*–*Conversion*.

Выводим полученные значения за внешний цикл *For* в терминал массива *Array* и терминал элемента отображения *Waveform Graph.* На границах циклов не забываем включить индексирование – ПКМ на границу и выбрать пункт в контекстном меню Включить Автоиндексацию (*Enable Indexing*).

Запустим нашу программу однократно *Run* и можем видеть получившийся график значений на элементе отображения *Waveform Graph* как на рис. 4.3. Если запустим программу в цикличном режиме *Run Continuosly*, то можем видеть, как значения меняются в реальном времени.

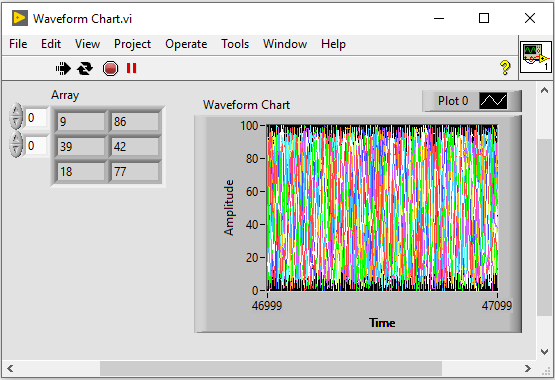


Рис. 4.3 – Лицевая панель виртуального прибора с *Waveform Graph*

# 4.1.2 Двухкоординатные графики

*Двухкоординатные графики* универсальны, они предназначены для отображения многозначных функций в декартовой системе координат (замкнутые кривые, распределение осциллограммы во времени с переменной временной базой).

*Одиночный Двухкоординатный график осциллограмм* (*XY graph*) работает с группами данных, содержащими массивы *х* и *у*. Двухкоординатный график осциллограмм (*XY graph*) также воспринимает массивы точек, где точки – группы данных, содержащие значения по шкалам *х* и *у*.

*Двухкоординатные графики множества осциллограмм* работают с массивами осциллограмм, в которых осциллограмма данных – кластер, содержащий массивы значений *х* и *у*. Двухкоординатные графики множества осциллограмм воспринимают также массивы множества осциллограмм, где каждая осциллограмма представляет собой массив точек. Каждая точка это группа данных, содержащая значения по *х* и *у*.

Сделаем виртуальный прибор, в котором нарисуем куб на элементе отображения *XY graph*.

Создадим новый прибор и добавим на лицевой панели два одномерных массива *Array* из раздела *Data Containers*–*Array* размерностью 11х1, которые назовем «*x*» и «*y*», помимо этого добавим два ползунка – горизонтальный *Horizontal Pointer Slide* и вертикальный *Vertical Pointer Slide* из раздела *Modern*– *Numeric*, назовем из «*Slide x*» и «*Slide y*». Ползунки будем использвать для перемещения нашего будущего куба по плоскостям координат. В качестве элемента отображения используем *XY graph* из раздела *Modern*–*Graph*, в свойствах которого установим максимальное и минимальное значение шкалы *X* и *Y* – от 0 до 10. Расположим элементы как на рис. 4.4.

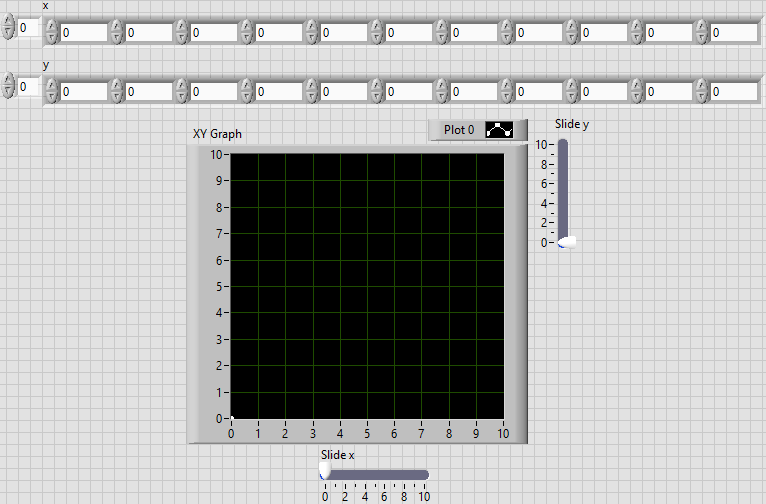


Рис. 4.4 – Лицевая панель виртуального прибора с *XY graph*

Перейдем к блок-диаграмме нашего прибора. На вход *XY graph* нельзя подключить напрямую массивы (не считая функции *Create Constant*, которую мы не будем использовать, т.к. с её помощью можно ввести массив констант только вручную), поэтому нужно использовать функции из раздела *Programming*–*Cluster, Class, & Variant*.

На блок-диаграмме добавим функцию связки *Bundle* из раздела *Programming*–*Cluster, Class, & Variant* и подключим к выходу терминала элемента отображения *XY* *graph*. Затем добавим две функции сложения *Add* из раздела *Mathematics–Numeric* и соединим, как показано на рис. 4.5 это позволит перемещать куб по координатам *X* и *Y*.

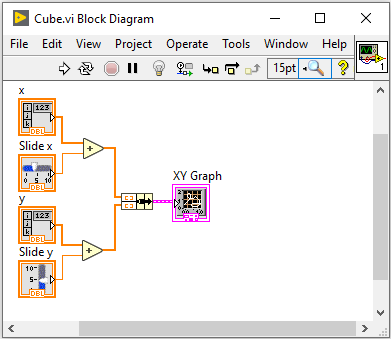


Рис. 4.5 – Блок-диаграмма виртуального прибора с *XY* *graph*

Вернемся к лицевой панели нашего прибора. Введем числа в массивы:

X: 2; 0; 0; 2; 2; 3; 3; 2; 0; 1; 3.

Y: 0; 0; 2; 2; 0; 1; 3; 2; 2; 3; 3.

Запустим нашу программу в цикличном режиме *Run Continuosly* и можем видеть получившийся график куба на элементе отображения *XY* *graph* как на рис. 4.6.

Получившийся график отрисовывается последовательно по точкам из массивов *x* и *y*. Прибор, используя функцию *Add* складывает число на ползунках *Slide x* и *Slide y* с координатам *x* и *y* соответственно. Таким образом реализуется перемещение графика по плоскостям координат.

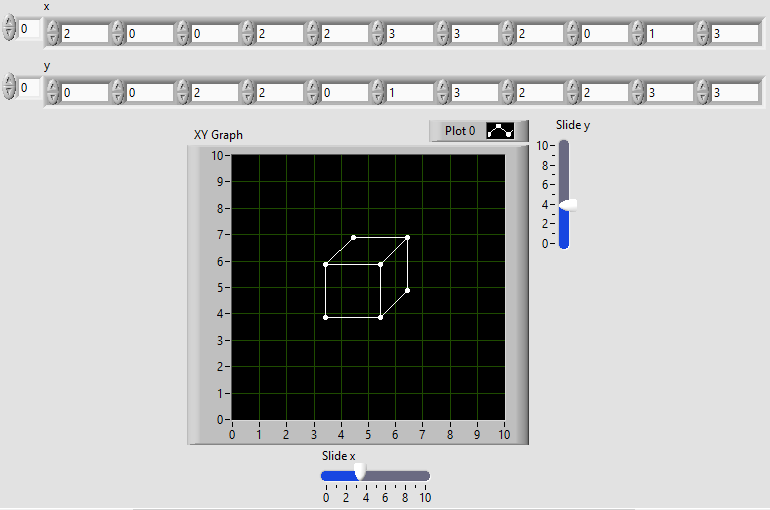


Рис. 4.6 – Лицевая панель виртуального прибора с нарисованным кубом

# 4.2 Использование палитры элементов управления графиком

Создадим новый прибор и добавим на лицевой панели элемент отображения *Waveform Graph* из раздела *Modern*–*Graph*. На блок-диаграмме добавим цикл *For* из раздела *Programming–Structures.* Поставим максимальное число итераций *N* равное 10000. Внутри цикла добавим числовую константу из раздела *Programming*–*Numeric* и присвоим ей значение 100 и математическую константу 2π из раздела *Programming*–*Numeric–Math Constants*.

Умножаем номер итерации *i* на математическую константу 2π, а полученное значение на 100, а после этого делим на *N* равное 10000. Для этого используем такие функции как *Multiply* и *Divide* из раздела *Programming*–*Numeric.*

Добавим на блок-диаграмму функцию *Sine*, вычисляющую синус числового значения, из раздела *Mathematics*–*Elementary–Trigonometric*. Соединим все, как показано на рис. 4.7.

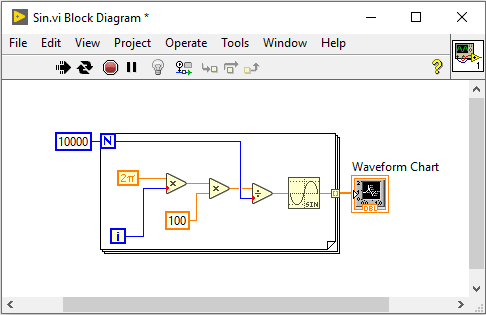


Рис. 4.7 – Блок-диаграмма виртуального прибора с функцией Sin

Запустим нашу программу в цикличном режиме *Run Continuosly* и можем видеть график синусоидальной функции *Sin* как на рис. 4.8.

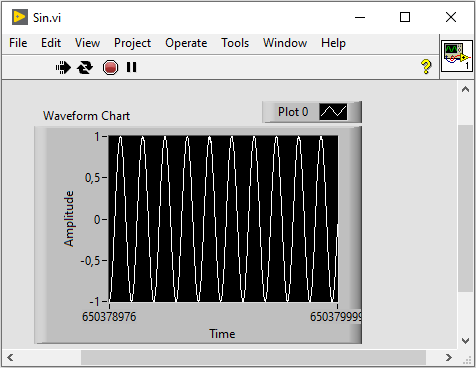


Рис. 4.8 – Лицевая панель виртуального прибора с функцией Sin

Перейдем к лицевой панели прибора. Установим палитру элементов управления графиком. Для этого нажмем ПКМ на графическом индикаторе *Waveform Graph* и во всплывающем меню установим галку в пункте *Visible Items*–*Graph Palette*. С помощью палитры элементов управления графиком выберем способ выделения участка графика, как показано на рис. 4.9.

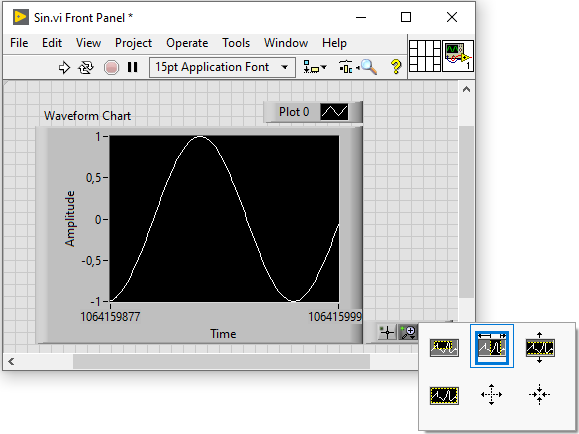


Рис. 4.9 – Выбор способа выделения участка графика с помощью палитру элементов управления графиком

# 5 Функция *Case Structure*

Функция *Case* *Structure* (рис. 5.1) имеет две или более подпрограммы. Только одна подпрограмма видна на терминале функци и только одна подпрограмма работает при выполнении данной функции. Функция *Case Structure* расположена в разделе *Programming*–*Structures*.

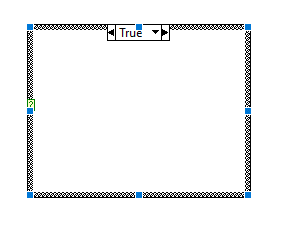


Рис. 5.1 – Функция *Case Structure*

Входное значение терминала селектора функции *Case Structure* определяет, какая подпрограмма будет выполняться в данный момент времени. Функция *Case Structure* аналогична операторам варианта (switch...case) или логическим операторам (if... else) в текстово-ориентированном языке программирования Си.

Селектор функции *Case Structure*, расположенный сверху графического изображения функции, включает в себя указатели: значения варианта подпрограммы в центре и стрелки влево и вправо по бокам. Эти стрелки используются для просмотра возможных вариантов подпрограмм.

Входное значение терминала функции *Case Structure* определяет, какая подпрограмма будет выполняться. Допустимо использовать целочисленный (*integer*), логический (*Boolean*), строковый (*string*) типы, а также тип перечисления в качестве значения терминала функции *Case Structure*, Терминал функции *Case Structure* может располагаться в любом месте левой границы терминала функции *Case Structure*. Если терминал функции *Case Structure* логического типа, то структура состоит из двух логических вариантов ИСТИНА (*TRUE*) и ЛОЖЬ (*FALSE*). Если терминал функции *Case Structure* имеет такой тип данных как целочисленный, строковый или перечисление, то количество вариантов может достигать 231-1 вариантов.

Для использования функции *Case Structure* необходимо отметить вариант по умолчанию (*default case*). Вариант по умолчанию или подпрограмма по умолчанию выполняется, если значение терминала функции *Case Structure* выходит за пределы диапазонов или не существуют варианты для возможных значений терминала функции.

Правый щелчок мыши на границе функции *Case Structure* позволяет добавлять, дублировать, перемещать и удалять варианты (подпрограммы), а также отмечать вариант по умолчанию.

Функция *Case Structure* допускает использование терминалов входных и выходных данных. Терминалы входных данных доступны во всех подпрограммах, но их использование подпрограммой функции необязательно. Создание выходного терминала на одной подпрограмме приводит к его появлению на других подпрограммах в том же самом месте границы структуры. Если хотя бы в одной подпрограмме выходной терминал не определен, поле этого терминала окрашивается в белый цвет, что характеризует ошибку определения структуры. Необходимо определять значения выходных терминалов во всех вариантах (подпрограммах). Кроме того, выходной терминал должен иметь значение одного и того же типа.

Для определения значения выходного терминала следует правым щелчком мыши по терминалу вызвать всплывающее меню и выбрать пункты: *Create*–*Constant* или *Create*–*Control*.

# 5.1 Примеры использования функции *Case Structure*

Следующие примеры показывают, как значения входных терминалов функции *Case Structure* взаимодействуют друг с другом в зависимости от значения терминала функции *Case Structure*.

# 5.1.1 Функция *Case Structure* с логическим элементом

Сделаем виртуальный прибор на основе прибора из пункта 2.2, который преобразует температуру из градусов Цельсия в градусы Фаренгейта.

Создаем новый виртуальный прибор (*Blank VI*). Добавим на лицевую панель логический элемент горизонтальный переключатель *Horizontal Toggle Switch* из раздела *Modern*–*Boolean*. С помощью палитры инструментов *Tools* *Palette*, добавим два текста (иконка А) слева и справа от переключателя *K* и *F* соответственно. Далее на лицевой панели разместим два индикатора *Thermometer* из раздела *Modern*–*Numeric*, которые переименуем как ― “*C*” и ― “*K или F*”. Индикатор с меткой ― “*C*” сделаем элементом управления.

Изменим шкалу у второго термометра-индикатора, отображающего температуру в Кельвинах или градусах Фаренгейта, для этого нажмем на индикатор правой кнопкой мыши и выберем пункт *Properties*. Далее перейдем на вкладку *Scale* и изменим значения *Scale Range* на такие:

*Minimum* – 0;

*Maximum* – 400.

Перейдем на блок-диаграмму. На блок-диаграмму добавим функцию *Case Structure* из раздела *Programming*–*Structures* и будем работать в варианте TRUE.

Внутри структуры добавим функции Умножение (*Multiply*) и Сложение (*Add*) из палитры функций в разделе *Programming*–*Numeric*. Создадим числовую константу у соответствующего входа функции умножения. Для этого подведем курсор мышки к этому входу, чтобы появился инструмент Катушка и однократно нажмем ПКМ. Во всплывающем меню выбираем *Create*–*Constant*. После размещения числовой константы на блок-диаграмме поле ввода ее значений подсвечивается и готово для редактирования. Присвоим ей значение 1,8. Точно также создаем вторую константу и присвоим ей значение 32, как показано на рис. 5.2.

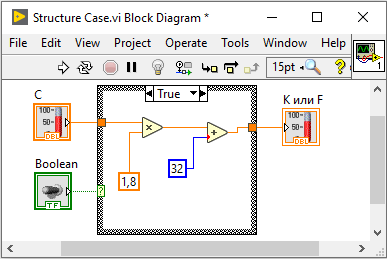


Рис. 5.2 – Блок-диаграмма прибора с функцией *Case Structure, TRUE*

После этого меняем вариант функции *Case Structure* нажатием стрелки вправо на вариант FALSE.

Внутри структуры добавим функцию и Сложение (*Add*) из палитры функций в разделе *Programming*–*Numeric*. Создадим числовую константу у соответствующего входа функции сложения равную 273, как показано на рис. 5.3.

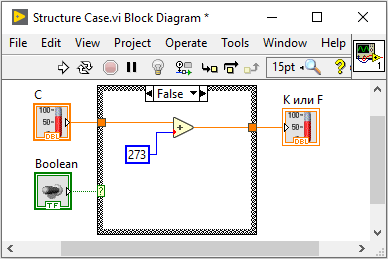


Рис. 5.3 – Блок-диаграмма прибора с функцией *Case Structure, FALSE*

Запустим нашу программу в цикличном режиме *Run Continuously.* Переведем переключатель в правое положение, которое соответствует варианту TRUE и переводу температуры из градусов Цельсия в градусы Фаренгейта, тогда получим на лицевой панели такое изображение как на рис. 5.4.

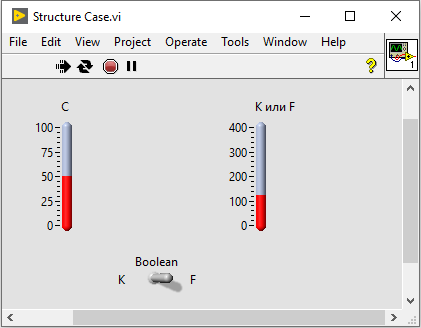


Рис. 5.4 – Лицевая панель прибора с функцией *Case Structure, TRUE*

Переведем переключатель в левое положение, которое соответствует варианту FALSE и переводу температуры из градусов Цельсия в Кельвины, тогда получим на лицевой панели такое изображение как на рис. 5.5

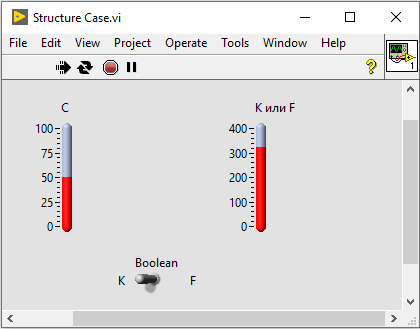


Рис. 5.4 – Лицевая панель прибора с функцией *Case Structure, FALSE*

Таким образом, если терминал логического элемента управления (Boolean), соединенный проводником с терминалом функции *Case Structure*, выдает значение ИСТИНА (TRUE), то выполняется преобразование температуры, заданной в градусах Цельсия, в градусы Фаренгейта (блок-диаграмма (рис.5.2)), если ЛОЖЬ (FALSE), то выполняется – преобразование температуры, заданной в градусах Цельсия, в Кельвины (блок-диаграмма (рис.5.3)).

# 5.1.2 Функция *Case Structure* с числовым элементом

Изменим предыдущий пример. Логический элемент управления *Boolean* заменим целочисленным – ручкой-переключателем как на рис. 5.5.

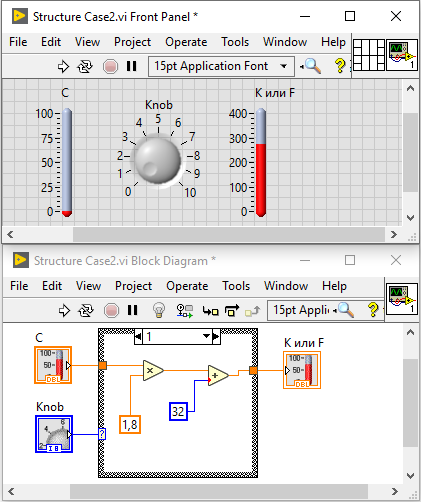


Рис. 5.5 – Виртуальный прибор с функцией *Case Structure,* с *Knob*

Для этого, на лицевой панели наведем на него курсор мышки и выполним однократное нажатие правой клавиши мышки. Во всплывающем меню выберем пункт *Replace*–*Modern*–*Numeric*–*Knob*. Наведем курсор мышки на появившийся элемент *Knob* и снова выполним однократное нажатие правой клавиши мышки. Во всплывающем меню выберем пункт *Representation*– *I8* *Byte* (целые числа от 0 до 255). При этом на блоке диаграмм терминал изменит окраску с оранжевой (соответствующей вещественным числам) на синюю (соответствующую целым числам).

При такой замене на селекторе структуры варианта значение ИСТИНА (TRUE) изменится на числовое значение 1, значение ЛОЖЬ (FALSE) – на значение 0, которое будет значением по умолчанию.

Данный виртуальный прибор будет выполнять преобразование температуры, заданной в градусах Цельсия в температуру по Фаренгейту только, если на элементе управления *Knob* установлено значение 1, во всех остальных случаях будет производиться преобразование в Кельвины.

Чтобы добавить в функцию *Case Structure* еще вариант необходимо во всплывающем меню на селекторе *Case Structure* выбрать пункт *Add Case After* или *Add Case Before*. Например, чтобы при установленном на элементе управления *Knob* значении 9 на индикаторе отображалось значение 90. Для этого новый вариант в функции можно сделать как на рис. 5.6.

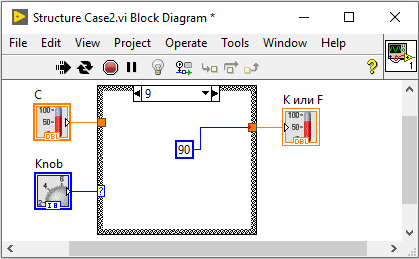


Рис. 5.6 – Модификация виртуального прибора изображенного на рис. 5.5

# 5.1.3 Функция *Case Structure* со строковым элементом

Изменим предыдущий пример. Целочисленный элемент управления *Knob* заменим строковым элементом *Combo Box* как на рис. 5.7. Для этого, на лицевой панели наведем на него курсор мышки и выполним однократное нажатие правой клавиши мышки. Во всплывающем меню выберем пункт *Replace*–*Modern*–*String & Path*–*Combo Box.*

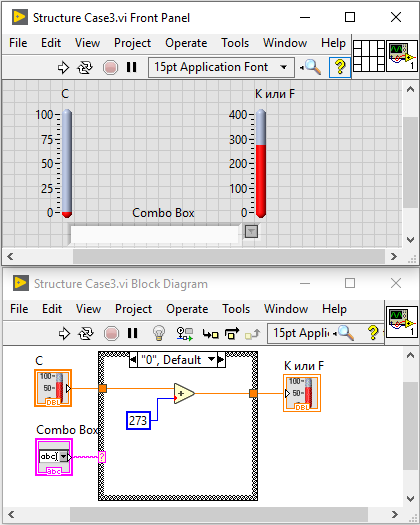


Рис. 5.7 – Виртуальный прибор с функцией *Case Structure,* с *Combo Box*

Наведем курсор мышки на появившийся элемент *Combo Box* и снова выполним однократное нажатие правой клавиши мышки. Во всплывающем меню выберем пункт *Edit Items…*. При этом откроется окно свойств (рис. 5.8) элемента *Combo Box* с вкладкой *Edit Items*, для добавления строк в наш выпадающий список *Combo Box*.

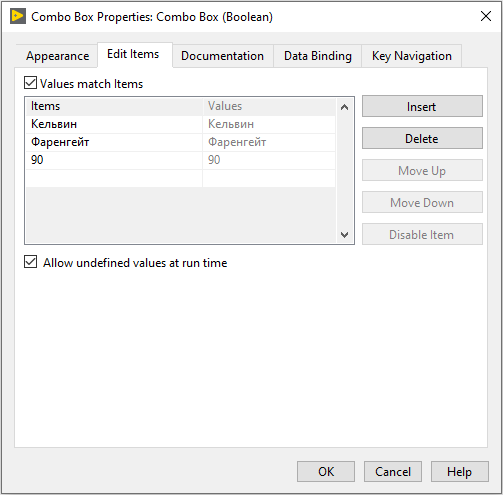


Рис. 5.8 – Окно свойств элемента *Combo Box*

Для добавления строк нажимаем на кнопку в окне *Insert*, пишем «Кельвин» и нажимаем *Enter* на клавиатуре. Также проделаем с добавлением строк с названием «Фаренгейт» и «90». На блок-диаграмме исправим значения в селекторе вверху функции *Combo Box.* Вместо «0» напишем «Кельвин», вместо «1» напишем «Фаренгейт», а вместо «9» напишем «90».

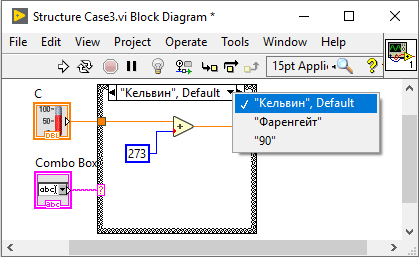


Рис. 5.9 – Блок диаграмма функции *Case Structure,* с *Combo Box*

Запустим нашу программу в цикличном режиме *Run Continuously.*

Теперь с помощью кнопки со стрелкой, направленной вниз, на элементе *Combo Box* мы открываем выпадающий список значений. Выбирая одно из этих значений, мы изменяем работу прибора, который преобразует температуру из градусов Цельсия в Кельвины, градусы Фаренгейта или просто показывая отметку в 90 градусов.

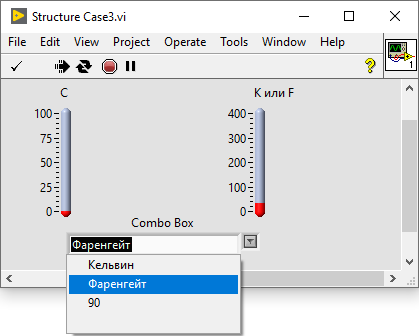


Рис. 5.10 – Лицевая панель прибора с *Case Structure* и *Combo Box*

# 5.2 Определение вариантов функции *Case Strucrure*

Определение вариантов функции *Case Strucrure* осуществляется либо выбором значения на селекторе функции *Case Strucrure*, либо вводом значения с помощью инструмента ВВОД ТЕКСТА (<А>). Это значение селектора функции *Case Strucrure* используется в дальнейшем при выборе вариантов функции. При определении вариантов функции *Case Strucrure* могут использоваться: единственное значение, списки и диапазон значений. Списки значений представляют собой введенные значения через запятую, например, – 1, 4, 6. Диапазон значений определяется как 10..20, т.е. вариант определен значениями в диапазоне от 10 до 20 включительно. При определении вариантов функции *Case Strucrure* также могут использоваться открытые диапазоны. К примеру, диапазон ..100 присваивает тот вариант, значения которого меньше или равны 100. Также можно при определении вариантов использовать комбинацию списков и диапазонов. Например, ..5, 6, 7..10, 12, 13, 14. Если Вы вводите значения селектора, включающие пересекающиеся диапазоны, функция *Case Strucrure* преобразует запись значения в более компактную форму. Предыдущий пример будет преобразован к ..10,12..14.

Значения селектора функции *Case Strucrure* должно быть того же типа, что и тип данных объекта, соединенного с терминалом селектора функции *Case Strucrure*. Значение селектора функции, окрашенное красным цветом, показывает, что необходимо удалить или отредактировать это значение перед запуском функции, иначе виртуальный прибор не будет выполняться. Нельзя использовать вещественные числа в качестве значения селектора функции, так как возможны ошибки округления и возникновения ситуации неопределенности. Если мы соединяем вещественный тип с терминалом селектора функции *Case Strucrure*, *LabVIEW* округляет это значение до ближайшего четного целого. Если вещественное значение введено непосредственно в селектор функции *Case Strucrure*, то оно окрашивается в красный цвет и должно быть удалено или отредактировано.

# Индивидуальные задания к лабораторной работе №1 - Вычисление

Задание: ввести числовые значения *A* и *B* используя элементы контроля, вычислить ответ по формуле, согласно варианту и используя функции из разделов *Mathematic*, и вывести полученный ответ на элемент отображения.

1. 
2. 
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 

**Контрольные вопросы к лабораторной работе №1**

1. Какие языки наиболее подобны *LabVIEW*?
2. Из каких двух окон состоит виртуальный прибор в *LabVIEW*?
3. Назовите основные кнопки на инструментальной панели *LabVIEW*.

# Индивидуальные задания к лабораторной работе №2 - Логика

Примечания к выполнению: для сравнения чисел используем функции из раздела *Programming*–*Comparison*. Для индикации используем элементы отображения из раздела *Modern*–*Boolean.* Для определения остатка от деления используем функцию *Quotient & Remainder* – верхний выход, нижний выход соответствует целой части от деления. Для выбора отображения одного из значений выбираем функцию *Select* из раздела *Programming*–*Comparison*. Для вывода сообщений используем элемент отображения *String Indicator* из раздела *Modern*–*String & Path* и строковые константы из раздела *Programming–String.*

* 1. Ввести два неравных числа *A* и *B*. Выяснить что больше *AB* или *BA*, при этом выведя на экран оба значения.
  2. Вывести сумму чисел , определить четная ли она, если четная, то найти значение произведения.
  3. Определить делится ли на целое число *x* целое число *y* без остатка. Если делится, то разделить *x* на *y* и вывести его.
  4. Проверить на неотрицательность значение выражения . Если выражение неотрицательно, то значение *k* утроить.
  5. Определить имеет ли действительные корни уравнение . Если имеет, то значение *a* возвести в квадрат и вывести на экран.
  6. Ввести целое число *X* , если *X* - четное – получить число *Y* , которое равно половине *X*, если *X* – нечетное – получить число *Y*, которое равно удвоенному значению *X*.
  7. Ввести целые числа *X* и *Y* и действительное число *Z*. Модифицировать число *Z* по формуле , если *X* не делится на *Y* без остатка и в противном случае. (*k* - остаток от деления *X* на *Y*)
  8. Если 2*m* больше чем , то *m* увеличить на 10 и вывести на экран, в противном случае *m* уменьшить в два раза и также вывести на экран.
  9. Определить делится ли на 5 введенное число *x* . Если делится, то другому числу *y* присвоить 100 , в противном случае *y* сделать равным 500.
  10. Ввести два числа *a* и *y*. Проверить равно ли нулю выражение *sin*(*x* + *a*), где *x* - целая часть выражения tg(*y*). Если выражение обращается в ноль, то значение *x* утроить и вывести на экран.
  11. Для двух дробей  и  выяснить равны ли их дробные части. Если части равны, то вывести оба значения, если нет, то ограничиться выводом соответствующего сообщения.
  12. Для двух дробей  и  выяснить равны ли их дробные части. Если части равны, то вывести один раз значение дробной части, если не равны, то вывести обе дробные части.
  13. Для дроби  определить превышает ли дробная часть числа 0,5. Если превышает, то число *m* удвоить и вывести на экран, иначе удвоить число *n* и также вывести.
  14. Определить превышает ли умноженная на 10 дробная часть числа  целую часть . Если превышает, то вывести значение на сколько.
  15. Определить является ли целая часть числа  четной. Если она четная, то *m* возвести в квадрат, иначе *n* утроить.

**Контрольные вопросы к лабораторной работе №2**

1. Назовите разделы палитры управления и отображения.
2. Назовите разделы палитры функций.
3. Как определить четность числа в *LabVIEW*?

# Индивидуальные задания к лабораторной работе №3 - Циклы

Задание: ввести числа *x* и *k* (если требуется) и определить сумму/произведение, согласно формуле по вариантам.

Примечание к выполнению работы: для суммы и произведения чисел в массиве используем сдвиговый регистр.

1. , используя цикл *While*
2. , используя цикл *While*
3. , используя цикл *For*
4. , используя цикл *For*
5. , используя цикл *While*
6. , используя цикл *For*
7. , используя цикл *While*
8. , используя цикл *For*
9. , используя цикл *While*
10. , используя цикл *While*
11. , используя цикл *For*
12. , используя цикл *For*
13. , используя цикл *While*
14. , используя цикл *While*
15. , используя цикл *For*

**Контрольные вопросы к лабораторной работе №3**

1. С какого числа начинает считать счетчик итераций *i*?
2. Цикл *While* еще называют «цикл с постусловием», почему?
3. Для каких целей могут использоваться сдвиговые регистры?

# Индивидуальные задания к лабораторной работе №4 – Массивы

Примечание к выполнению работ: используем циклы со сдвиговыми регистрами, функции работы с массивами из раздела *Programming*–*Array*, математические функции из раздела *Mathematics*–*Numeric* и функции сравнения из раздела *Programming–Comparison*, в т.ч. функцию *Select*.

1. Ввести одномерный массив *X*(13). Найти произведение тех из них, чей квадрат меньше порядкового номера вводимого числа.
2. Ввести два числа *a* и *b*. В программе задается отрезок [*a*, *b*]. Вывести случайное число из этого отрезка.
3. Создать и заполнить массив случайными числами из отрезка [*a*, *b*] и вывести его на экран. Вычислить размер массива и вывести на экран.
4. Создать двумерный массив, заполнить случайными числами из отрезка [*a*, *b*] и вывести его на экран. Вычислить размер массива и вывести на экран.
5. Создать и заполнить массив с *N* элементами, заполнить случайными числами из отрезка [*a*, *b*] и вывести его на экран. Вывести на экран элемент с индексом *N-*1.
6. Ввести одномерный массив *X*(8), вычислить сумму отрицательных элементов этого массива и вывести на экран.
7. Ввести одномерный массив *X*(9), найти максимальный элемент и вывести на экран элемент, стоящий после максимального. Если такого элемента нет – вывести число *Max*-*N.*
8. Ввести одномерный массив *X*(10) и целочисленное число *a.* Вставить три целочисленных случайных числа в массив на позицию *а*-ого элемента и вывести полученный массив.
9. Ввести одномерный массив *X*(11) и целочисленное число *a<*11*.* Удалить из массива два элемента с позицией *a* и *a*+1. Вывести полученный массив.
10. Ввести одномерный массив *X*(7), найти минимальный элемент и вывести на экран элемент, стоящий после минимального. Если такого элемента нет – вывести число *Min*+*N.*
11. Создать и заполнить массив с *N* элементами, заполнить случайными числами из отрезка [*a*, *b*] и вывести его на экран. Найти максимальный и минимальный элемент и вывести на экран их разницу.
12. Ввести одномерный строковый массив *X*, в котором содержаться 4 слова, и вывести его на экран. Выполнить сортировку по алфавиту и удалить последний элемент. Вывести полученный массив.
13. Ввести двумерный строковый массив *X*, в котором содержаться 6 слов. Выполнить сортировку по 1 строке и вывести полученный массив.
14. Ввести одномерный массив *X*(12), вычислить сумму положительных элементов этого массива и вывести на экран.
15. Ввести одномерный массив *X*(6), найти минимальный элемент и вывести на экран. Если Min>0, то умножить все элементы массива на 2, в ином случае на 3. Вывести полученный массив.

**Контрольные вопросы к лабораторной работе №4**

1. Какими способами можно сделать одномерный массив в *LabVIEW*?
2. Какими способами можно сделать двумерный массив в *LabVIEW*?
3. С какого числа начинается индексация в массивах в *LabVIEW*?

# Индивидуальные задания к лабораторной работе №5 – Графики

Задание: Построить график элементарной функции согласно формуле по вариантам. График строим по точкам *x* ϵ[-50;50]. Ввести необходимые коэффициенты, согласно варианту. Вывести массив значений *y*.

Примечание к выполнению работы: Для ввода *x* используем одномерный массив, желательно через цикл *For*, вручную вводить числа от -50 до 50 не рекомендуется.

1. .
2. .
3. .
4. .
5. .
6. .
7. .
8. .
9. .
10. .
11. .
12. .
13. .
14. .
15. .

**Контрольные вопросы к лабораторной работе №5**

1. Какие элементы используются для отображения графиков в *LabVIEW*?
2. Какую функцию необходимо использовать для соединения двух одномерных массивов с элементом *XY Graph*?
3. Как выделить определенный участок на графике?

# Библиографический список

1. Виноградова, Н. А. Разработка прикладного программного обеспечения в среде *LabVIEW*: Учебное пособие / Н. А. Виноградова Я. И. Листратов, Е. В. Свиридов. – М. : Издательство МЭИ, 2005. – 49 с.
2. Дивин, А.Г. Автоматизация измерений, контроля и испытаний. В 2 ч. : лабораторные работы. Ч. 1 : Основы работы в программной среде LabVIEW / авт.-сост. : А.Г. Дивин, В.М. Жилкин, А.Д. Свириденко. Тамбов : Изд-во Тамб. гос. техн. ун-та, 2005. 44 с.
3. Лабораторные работы «Основы программирования в *LabView* 8» по курсу «Проектирование информационно-измерительных систем». / Сост.: Н. Н. Беспалов, М. В. Ильин – Саранск: Ковылк.тип., 2006. – 47 с.
4. Лупов C.Ю., Лупов С.Ю., Муякшин С.И., Шарков В.В. *LabVIEW* в примерах и задачах. Учебно-методические материалы по программе повышения квалификации «Обучение технологиям *National Instruments*». Нижний Новгород, 2007, 101 с.
5. *Getting Started with LabVIEW* [Электронный ресурс] – Режим доступа: https://inside.mines.edu/~lwiencke/elab/labs/321527d.pdf
6. *LabVIEW Basics I Course Manual* [Электронный ресурс] – Режим доступа: https://www.8051projects.net/files/public/1286860497\_9028\_FT32  
   822\_labviewbasic\_1.pdf
7. *LabVIEW User Manual* - *National Instruments* [Электронный ресурс] – Режим доступа: https://www.ni.com/pdf/manuals/320999e.pdf

Учебное издание

Александр Николаевич **Шилин**

Алексей Александрович **Шилин**

Павел Владимирович **Дикарев**

**Виртуальная лаборатория на базе *LabVIEW***

*Учебно-методическое пособие*

Редактор *Л. Н. Рыжих*

Темплан 2020 г. (учебники и учебные пособия). Поз. № 00.

Подписано в печать 00.00.2020. Формат 60x84 1/16. Бумага газетная.

Гарнитура Times. Печать офсетная. Усл. печ. л. 00. Уч.-изд. л. 00.

Тираж 100 экз. Заказ

Волгоградский государственный технический университет.

400005, г. Волгоград, просп. В. И. Ленина, 28, корп. 1.

Отпечатано в типографии ИУНЛ ВолгГТУ.

400005, г. Волгоград, просп. В. И. Ленина, 28, корп. 7.