Структура программы.

Программа состоит из следующих компонентов:

1. Form1.cs

2. Parser.cs

3. Memessage.cs

4. ZedGraph.dll

1. Компонент Form1.cs является корневым, использующим все остальные компоненты. Он содержит обработку событий ввода данных (значений) в текстовые окна (textBox), нажатия на кнопки и многие другие события.

Структурно Form1.cs состоит из следующего списка объектов, переменных и методов:

*List<List<double>> listDlt = new List<List<double>>();*

Список списков listDlt предназначен для записи в него значений окрестностей и точек, содержащихся в этих окрестностях, образуемых пересечением эпсило-трубки с графиком функции.

*List<double> listFirstElem = new List<double>();*

В списокlistFirstElem записываются точки входа в эпсилон-трубку (левые крайние значения дельта-трубки). Он необходим для подсчета количества обнаруженных дельта-трубок и нахождения наименьшей из найденных ширин дельта-трубок.

Примечание: в этот список записываются левые крайние значения дельта-трубок не зависимо от того, находится ли дельта трубка в положительной части оси оХ или с отрицательной её части. Это связано с особенностями библиотеки ZedGraph, о которой я расскажу чуть позже.

*List<double> listWidth = new List<double>();*

Список для значений ширин дельта-трубок.

*public Form1()*

Данный метод (точнее конструктор) инициализирует значения по умолчанию окон-ввода, определяет форматирование текста, устанавливает разделитель целой и дробной части числа в значение «,», обнуляет счетчик ошибок и выключает кнопку «Очистить».

*private void textBoxDbeg\_Click(object sender, EventArgs e)*

*private void textBoxDend\_Click(object sender, EventArgs e)*

*private void textBoxEpsStep\_Click(object sender, EventArgs e)*

*private void textBoxY0\_Click(object sender, EventArgs e)*

Данные четыре метода обрабатывают событие нажатие на окна-ввода «От» (textBoxDbeg), «До» (textBoxDend), «r» (textBoxEpsStep), «y0» (textBoxY0) соответственно.

*private void textBoxDbeg\_KeyPress(object sender, KeyPressEventArgs e)*

*private void textBoxDend\_KeyPress(object sender, KeyPressEventArgs e)*

В этих методах обрабатывается событие нажатие клавиш клавиатуры при условии, что фокус ввода находится либо в окне ввода «От» (textBoxDbeg), «До» (textBoxDend).Пользователю разрешено вводить в эти окна:

- цифры (в том числе и константы Pi и Eps в буквенном виде);

- знаки математических операций (в частности: сложение «+», вычитание «-», умножение «\*», деление «/», возведение в степень «^», скобки «(» и «)»).

- разделитель: запятая «,» (причем лишь один раз, что естественно).

*private void textBoxEpsStep\_KeyPress(object sender, KeyPressEventArgs e)*

*private void textBoxY0\_KeyPress(object sender, KeyPressEventArgs e)*

Эти методы аналогичны предыдущим двум, различие заключается лишь в том, что в окна ввода «r» (textBoxEpsStep) и «y0» (textBoxY0) нельзя вводить знаки математических операций.

*private void textBox\_Function\_Click(object sender, EventArgs e)*

Обработка события нажатия на окно ввода функции «f(x)» (textBox\_Function). При нажатии на это окно выделяется функция.

*public double Dbeg*

*public double Dend*

*public double y0*

*public double eps\_step*

Метод Dbeg возвращает значение левого края отрезка, задаваемого пользователем. Если значение не было введено и окно останется «пустым», то в метод вернёт 0.

Метод Dend возвращает значение правого края отрезка, задаваемого пользователем. Если значение не было введено, то в метод вернёт 0.

Метод y0 возвращает значение точки, из которой строится эпсилон-трубка. Если значение не было введено и окно останется «пустым», то в метод вернёт 0.

Метод eps\_step возвращает значение эпсилон, введённое пользователем или заданное по умолчанию. Если значение не было введено, то в метод вернёт 0.00.

*public string func*

Метод func возвращает строку, содержащую текст введённой пользователем функции в окно «f(x)» (textBox\_Function). Если значение не было введено, то в метод вернёт пустую строку.

*private void Build\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Построить» (Build\_button). После нажатия на кнопку последовательно выполняются следующие действия:

- обнуляется счетчик ошибок (для случая, когда пользователь строит несколько графиков);

- вычисляются значения начала и конца отрезка построения (для случая, когда пользователь вводит константы Pi и Eps или же какое-нибудь выражение);

- проверяется наличие введённой функции в поле «f(x)»(textBox\_Function), если в этом окне текста не содержится (содержится пустая строка), возникнет сообщение «Функция не была введена» и пока пользователь не введёт функцию построения её графика не произойдёт;

- сравниваются значения границ отрезка построения, в случае если левый конец больше правого, возникнет сообщение «Ошибка: Введитекорректноконцыинтервалапостроения! Замечание: левый конец интервала должен быть меньше правого», пока пользователь не изменит значения границ отрезка построения на допустимые график функции строится не будет;

- если значения полей ввода функции и границ отрезка построения её графика удовлетворили всем условиям, то вызывается метод CreateGraph(zedGraphControl1, MyGraphColor, eps\_step), который в качестве входных значений получает zedGraphControl1 - экземпляр класса ZedGraphControl (холст, на котором происходит построение графика), MyGraphColor - цвет построения графика, eps\_step – значение поля «r» (textBoxEpsStep), необходимое для «пошагового» построения графика с шагом, равным eps\_step. Значение булевских величин IsGraphCreate (необходимо для определения, построен ли график функции), Clear\_button.Enabled (необходимо для активации кнопки «Очистить» (Clear\_button)), Okrestnost\_button.Enabled(необходимо для активации кнопки «Показать» (Okrestnost\_button))устанавливаются в значение истина (true).

- в случае возникновения ошибки в ходе выполнения вышеперечисленных действий возникнет сообщение «Ошибка функционирования программы» и программа прекратит построение графика (в данном случае необходимо связаться с разработчиком для выяснения и утсранения неполадки).

*private void Update\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Обновить» (Update\_button). Данная кнопка была добавлена для случая более детальной прорисовки «сложных» графиков (например, y = sin(1/x)): происходит уменьшение значения эпсилон, введённого пользователем, в такое количество раз, которое осуществила программа при вычислении ширин дельта трубок в методе BuildArrDelt (о нём ниже) и, исходя уже из измененного значения перерисовывается график (Прим. 1). Алгоритм перерисовки абсолютно аналогичен алгоритму построения графика в перый раз (см. обработку события нажатия на кнопку «Построить» (Build\_button)). Данная кнопка становится активной только в случает последовательного нажатия кнопок «Построить» и «Показать».

*private void Okrestnost\_button\_Click(object sender, EventArgs e)*

Обработка события нажатия на кнопку «Показать» (Okrestnost\_button). После нажатия начинается построение эпсилон- и дельта-трубок, исходя из следующего алгоритма:

- в начале осуществляется проверка построен ли график функции (именно здесь используется булевская величина IsGraphCreate, упоминавшаяся выше);

- обнуляются (очищаются) все значения списков listDlt, listFirstElem, listWidth, окон listBox\_delta, listBox\_Width, textBox\_MinWidth, textBox\_CurentWidth;

- вызывается метод BuildEpsTrub, стоящий эпсилон-трубку;

- вызывается метод BuildArrDelt, строящий дельта-трубку(-и);

- свойство активации кнопки Update\_button устанавливается в значение истина (true);

- свойство активации кнопки Okrestnost\_button устанавливается в значение ложь (false);

- в случае невыполнения условия IsGraphCreate, проверяющего построен ли график, возникнет сообщение «Ошибка: Сначала постройте график! Замечание: Нажмите на кнопку «Построить»»;

- в случае возникновения ошибки в ходе выполнения вышеперечисленных действий возникнет сообщение «Ошибка функционирования программы» и программа прекратит построение графика (в данном случае необходимо связаться с разработчиком для выяснения и утсранения неполадки).

*private void Form1\_Resize( object sender, EventArgs e )*

Обработчик события изменения размера окна программы (формы(Form1)). В случае изменения размера формы все элементы, содержащиеся на форме

*private void SetSize()*

Метод обрабатывает событие изменения размера формы: изменяет размер и положение графика.

*private void CreateGraph(ZedGraphControl zgc, Color MyGraphColor, double eps)*

Метод предназначен для построения графика, используя следующие три входных значения: zgc – экземпляр класса ZedGraphControl (холст, на котором изображен график), MyGraphColor – экземпляр класса Color (цвет, которым будет рисоваться график), eps – переменная типа double (величина эпсилон). Метод реализует следующий алгоритм:

- объявление переменных x и y типа double.

- построение первой точки графика функции для выяления ошибок ввода функции. Ошибки могут быть следующих типов: DIVBYZERO (деление на нуль), NOEXP (отсутствие вычисляемого выражения), SYNTAX (синтаксическая ошибка (некорректный ввод аргументов, неправильный ввод операций, неверное название функций и т. п.)), UNBALPARENTS (количество открывающих скобок не совпадает с количеством закрывающих), OTHER (ошибка алгоритма), NOERR (нет ошибки).

- если экземпляр p.Errors.NOERR класса Parser.Errors имеет истинное значение (true), то последует построение функции: цикл накопления значения переменной х. Он начинается со значения х=Dbeg (начальному значению – левой границе отрезка) и заканчивая значением х=Dend (конечное значение – правая граница отрезка), происходит выполнение метода Evaluate(func, x) класса Parser, использующего в качестве входных значений переменную типа string, в которой должно содержаться текстовое значение функции, вводимое пользователем, и переменную х типа double, точку в которой будет вычисляться значение функции func. Вычисленное значение будет записано в переменную y. После вызывается метод Add(x, y) объекта метода list1 класса ZedGraphControl библиотеки ZedGraph, который добавляет на холст точку с вычисленными координатами x и y, и соединяет её с предыдущей нанесённой точкой (если така имеется). При шибке DIVBYZERO точка просто не будет записана в у и не будет нанесена на холст.

- далее выполняется следующий код: LineItem myCurve = myPane.AddCurve(sName, list1, MyGraphColor, SymbolType.Circle), который наносит на холст название функции (func), инструмент, которым будет строиться график (list1), цвет построения графика (MyGraphColor), геометрическое изображение точек (SymbolType.Circle).

- После выполняется метод zgc.AxisChange(), перестраивающий оси координат (для того, чтобы после построения графика пользователь смог увидеть весь построенный график, а не его часть).

*Color MyGraphColor*

Метод случайным образом выбирает значение типа Color из следующих: Blue, Navy, BlueViolet, CadetBlue, CornflowerBlue, Chartreuse, DarkBlue, DarkSlateBlue, DodgerBlue, Indigo, MediumBlue, MidnightBlue, RoyalBlue. Этим цветом будет строиться график функции.

*Color MyEpsColor*

Метод случайным образом выбирает значение типа Color из следующих: Gainsboro, Lavender, LightSalmon, Lime, LimeGreen, MediumAquamarine, MediumSpringGreen, Moccasin, PaleGreen, PeachPuff, Red, YellowGreen, Yellow, DarkRed. Этим цветом будет строиться эпсилон-трубка.

*Color MyDeltColor*

Метод случайным образом выбирает значение типа Color из следующих: DarkOrange, DarkViolet, DarkOrchid, DarkSlateBlue, DarkGoldenrod, Tomato, DimGray, DarkCyan, Teal, Thistle, DeepPink, DarkTurquoise, Crimson, Fuchsia. Этим цветом будет строиться дельта-трубка.

*private void TextBoxEnter(object sender, EventArgs e)*

Метод обрабатывает событие нажатия на окно. При нажатии на окна «f(x)», «От», «До», «r», «y0» текст в них выделяется.

*void BuildEpsTrub(ZedGraphControl zgc, double Y, Color MyColor)*

Строит эпсилон-трубку, в точке Y, цвета MyColor, на холсте zgc. Цикл начиная с x = Dbeg, заканчивая x = Dend и аналогично методу CrateGraph строится эпсилон-трубка.

*void BuildDeltTrub(ZedGraphControl zgc, double Xbeg, double MinCountEps, double Y, Color MyColor)*

Строит дельта-трубку, радиуса MinCountEps, в точке Xbeg со значения, равного 0, до значения Y, цвета MyColor, на холсте zgc. Цикл начиная с x = 0, заканчивая x = Y и аналогично методу CrateGraph строится дельта-трубка.

*bool yInTbk(double y)*

Метод возвращает истинное значение (true) если переменная y содержится в эпсилон-трубке.

*void BuildArrDelt(ZedGraphControl zgc)*

Метод формирует массив массивов абсцисс точек, у которых ордината удовлетворяет условию yInTbk. Для этой задачи потребуется ловить события входа и выхода из эпсилон-трубки, т.е. для каждой итерации цикла мы будем определять удовлетворяет ли ордината, вычисленная методом Evaluate, условию yInTbk. Итак, у нас имеется цикл, начинающий перебор со значения x=Dbeg до значения x=Dend с шагом итерации, равным значению eps = eps\_step/10 (перебирать ординаты с шагом eps\_step было бы несовсем рационально, поскольку через слишком много дельта-трубок программа перескакивала, а так количество перескоков мы сокращаем в 10 раз), т.е., начиная с x=Dbeg мы последовательно, прибавляя к x значение eps каждую итерацию цикла, вычисляем в этой точке ординату точки. Также для работы нам небходимо знать не только значение ординаты в точке с абсциссой х, но и следующее значение ординаты (значение в точке с абциссой, равной x+eps), чтобы можно было ловить события входа/выхода в эпсилон трубку.

Цикл по переменной х выполняет следующий алгоритм:

- вычисляет значения y (для x) и yNext (для x+eps).

- проверяет, удовлетворяет ли условию *yInTbk(y) == false && yInTbk(yNext)* («y не в трубке, а yNext в трубке») значения обоих вычисленных переменных. Если они удовлетворяют, то булевский параметр Ent ставится в значение истина (Ent = true), в список listDlt объявляется новый подсписок.

- в случае невыполнения предыдущего условия проверяет, удовлетворяет ли условию *yInTbk(y) && yInTbk(yNext) == false* («у в трубке, а yNext не в трубке»), и выполняте в случае истинности выражения присвоение к булевскому параметру Ext значение true.

- в иных случаях (в случаях, когда и у и yNext находятся вне эпсилон-трубки) обоим булевским переменным присваиваются значение ложь (false).

Далее требуется обработать случай нахождения y в эпсилон-трубке, т.е проверяется условие *yInTbk(y).* Причем:

- сначала необходимо проверить первое ли вхождение в эпсилон-трубку сейчас или нет. Эта проверка необходима для случая, когда ордината левой границы отрезка построения графика уже находится в эпсилон-трубке и «поймать» событие входа в трубку не представляется возможным. В случае выполнения этого условия выполняется алгоритм, аналогичный для условия входа в эпсилон-трубку.

- далее (в ином случае) необходимо проверить первый ли элемент будет добавляться в список значений listDlt, если да, то инкрементируем счетчик индексов для этого списка.

- иначе проверяем условие пустоты списка первых точек вхождений дельта-трубок в эпсилон-трубку (listFirstElem), если он пуст, то просто добавляем в него первый элемент иначе проверяем условие *Math.Abs(listFirstElem[listFirstElem.Count - 1] - x) > eps* (если абсолютная величина разности последнего значения в списке listFirstElem и х больше значения шага итерации eps). При значении true этого условия в список первых элементов добавляется элемент х.

- после выполнения вышеперечисленных проверок выполняется присвоение булевскому счетчику firstEl, определяющему первый «пойманный» элемент дельта-трубки, значения false и добавление актуального значения х в список listDlt.

В случае, когда условие yInTbk(y) ложно, будет проверяться условие *!yInTbk(y) && ! yInTbk(yNext) && (y < y0 - eps\_step) && (yNext > y0 + eps\_step) || !yInTbk(y) && ! yInTbk(yNext) && (y > y0 - eps\_step) && (yNext < y0 + eps\_step)* («у не в трубке И yNext не в трубке И у меньше разности у0 (ордината расположения эпсилон-трубки) и eps\_step (первоначального значения эпсилон) И yNext больше суммы у0 и eps\_step» ИЛИ («у не в трубке И yNext не в трубке И у больше разности у0 и eps\_step И yNext меньше суммы у0 и eps\_step»). Условие ИЛИ разделяет здесь случаи возрастания и убывания графика функции, ведь в первом случае значение следующей ординаты будет больше значения предыдущей, а во втором – меньше, что необходимо учитывать при анализе состояния нахождения в/вне эпсилон-трубки. В случае истинности этого условия происходит уменьшение шага эпсилон в 100 раз и выполняется тот же алгорим, что представлен выше, только для измененного значения эпсилон. Программное уменьшение шага итерации эпсилон необходимо для случаев быстрого возрастания графика функции, когда шаг итерации может просто перескочить через эпсилон-трубку. В данном случае вероятность перескока сокращается ещё в 100 раз.

Следующим шагом необходимо вывести все обнаруженные дельта-трубки (им соответствуют первые элементы каждой из дельта-трубок), поэтому пользователь в окне «Дельта-трубки» (listBox\_delta) увидит самые первые попавшие точки этих трубок (по одной точке от каждой трубки).

Далее необходимо построить дельта-трубки минимальной ширины. Для этого происходит вызов методов GetWidth (о нём ниже) и BuildDeltTrub (о нём рассказывалось выше). Причем если ни одна ширина дельта-трубок не является меньше заданного пользователем эпсилон, то за ширину этой трубки будет приниматься величина эпсилон, уменьшенная в 10 раз. Это уменьшение необходимо для случая, когда ордината точки экстремума графика функции находится в эпсилон-трубке, и для некоторых других случаев.

После выводится в окно «Минимальная» (MinWidth) значение минимальной ширины дельта-трубки.

*double GetWidth()*

Метод вычисляет наименьшую ширину дельта-трубки, вычисляя сначала значние каждой, а в последствии сравнивая его со следующим значением. В начале работы метода обнуляются все необходимые перерменные, счетчик выхода из трубки ставится в положение false.

Далее выполняется проверка на пустоту списка точек пересечения эпсилон-трубки с графиком функции listDlt, если список пуст, то возникнет сообщение «Эпсилон-трубка не пересекает график функции у = <текст функции>».

Иначе переменной х1 присвоится значение первого элемента первого подсписка списка listDlt (*x1 = listDlt[0][0]*) и начнёт выполняться цикл по всем индексам i, пока значение i меньше количества подсписков в списке listDlt (*while (i < listDlt.Count)*). Если значение i равно нулю, то переменной firstX присвоится значение первого элемента первого подсписка списка listDlt (*firstX = listDlt[0][0]*). Далее последует цикл по элементам подсписков списка listDlt, начиная с нуля и заканчивая количеством элементов актуального подсписка (*for (int j = 0; j < listDlt[i].Count; j++*)). в котором переменной x2 присваивается значение j элемента i подсписка (*x2 = listDlt[i][j]*) и выполнится проверка условия *Math.Abs(x1 - x2) < eps\_step* (абсолютная величина разности х1 и х2 меньше эпсилон). В случае значения истины этой проверки произойдёт переприсваение переменной х1 значения х2 (*х1=х2*) и проверится значение булевской переменной ExitTbk, в случае true (истины) переменной lastX присвоится значение переменной х2 и в переменную TbkWidth запишется длина отрезка [firstX,lastX] (*TbkWidth = Math.Abs(firstX - lastX)*), значение переменной ExitTbk сменится на false.

В случае, противном случаю *Math.Abs(x1 - x2) < eps\_step*, переменная ExitTbk примет значение истина, переменные х1, firstX получат значения j элемента i подсписка списка listDlt (*x1 = firstX = listDlt[i][j]*).

После выполнения цикла *for (int j = 0; j < listDlt[i].Count; j++*) инкрементируется i.

Далее с помощью метода GetCurWidth(indx), который в качестве входного параметра принимает индекс первых элементов списка дельта-трубок listFirstElem, выполняется цикл перебора всех значений ширин дельта-трубок, записи этих значений в окно «Все значения» (listBox\_Width), причем в случае, когда ширина дельта-трубки больше эпсилон-шага, значению этой ширины присваивается значение эпсилон-шага, уменьшенного в 10 раз. Также записываются значения в списко ширин дельта-трубок listWidth.

После выполняется поиск минимальной ширины путём обычной «пузырьковой» сортировкой (наименьшие элемент помещается в начало списка и всё это делается в двойном цикле).

*privatevoid listBox\_delta\_SelectedIndexChanged(object sender, EventArgs e)*

Обработка события нажатия на окно «Дельта-трубки» (listBox\_delta). Данный метод возвращает ширину конкретной дельта-трубки, которую выделил пользователь, в окно «Выделенной» (textBox\_CurentWidth).

*double GetCurWidth(int idx)*

Метод, получая на вход индекс дельта-трубки, возвращает её ширину, сравнивая разницу между каждой парой элементов списка listDlt с величиной эпсилон. В случае, если разница меньше эпсилон, то в переменную lastX записывается последний взятый для сравнения элемент списка listDlt и разница между этим элементом и актуальным элементом списка listFirstElem будет являться актуальной шириной дельта-трубки (актуальность определяется индексом, который является входным значением для этого метода).

privatevoid Clear\_button\_Click(object sender, EventArgs e)

Обработка события нажатия на кнопку «Очистить». Программа запросит подтверждение очистки холста и после согласия пользователя на продолжение очистит панель отображения графика (холст), обнулит все списки (точек, лежащих между точкой входа и точкой выхода из эпсилон-трубки; точек входа в эпсилон трубку; ширин дельта-трубок), приведёт форму в первоначальный вид.

*privatevoid Form1\_FormClosing(object sender, FormClosingEventArgs e)*

Обработка события нажатия на кнопку «Закрыть» (кнопка «Х»). Во избежание случайного закрытия программы пользователем программа запросит подтверждение выхода.

*privatevoid button\_help\_Click(object sender, EventArgs e)*

При нажатии на кнопку «О программе» открывается окно-сообщение, содержащее сведения о возможностях программы, о её разработчике, месте и времени создания.

2. Компонент Parser.cs обрабатывает вводимую пользователем функцию.

Для обработки текста функции нам понадобится разбивать его на единицы (лексемы), которые впоследствии и будем обрабатывать. Лексема может быть следующих типов: NONE (пустая), DELIMITER (разделитель), VARIABLE (переменная), FUNCTION (функция), NUMBER (число). Также необходимо будет обрабатывать ошибки (иначе – некорректно введённые пользователем значения, функции и др.). Для этого разобьём ошибки на следующие типы: SYNTAX (синтаксическая), UNBALPARENTS (дисбаланс скобок), NOEXP (отсутствие выражения), DIVBYZERO (деление на нуль), OTHER (другая), NOERR (нет ошибки). Понадобятся переменные exp (ссылка на строку выражения), expIdx (текущий индекс в выражении), token (текущая лексема), tokType (тип лексемы), varX (переменная х), err (тип ошибки).

*public double Evaluate(string expstr, double var)*

Этот метод - входная точка анализатора, который на вход требует строку с функцией и значение переменной вещественного типа. После инициализации переменных и обнуления текущего индекса expIdx вызывается метод GetToken, которые возвращает текущую лексему. Далее следует проверка на пустоту этой лексемы, если она пуста, то переменной err присваивается NOEXP и анализатор возвращает 0. Далее следует цепь последовательных рекурсивных вызовов процедур, проверяющих совпадение текущей лексемы с различными операциями или функциями: вызывается метод EvalExp2 (о нём и о рекурсивном вызове чуть позже), после (рекурсивный вызов «посчитает» всю строку) проверяется текущая лексема, которая должна быть нуль-значением (пустой строкой), если это не так, то переменной err присвоится значение SYNTAX, иначе метод Evaluate возвратит значение функции в точке var.

*void EvalExp2(out double result)*

Метод EvalExp2 сначала инициализирует локальную переменную ор, вызывает метод EvalExp3, который «идет» дальше по рекурсивному вызову и возвращает вычисленный результат в переменной result, после сравнивает лексему с «-» или с «+», причем сравнение зацикленно: (переменной op присвоили значение лексемы) пока ор равна одной из этих операций будет выполняться запрос следующей лексемы, вызов метода EvalExp3. После прохождения рекурсивного вызова уже по методу EvalExp3, который вернет результат выражения, записанный в переменную partialResult, в зависимости от значения ор: если ор = «-», то переменной result присваиваем результат разности result и partialResult и выходим из цикла, если ор = «+», то – их сумму и также выходим из цикла.

*void EvalExp3(out double result)*

Метод EvalExp3 сначала инициализирует локальную переменную ор, обнуляет переменную partialResult, вызывает метод EvalExp4, который также как и в предыдущем случае «опускается» ниже по рекурсивному вызову и возвращает результат вычисленного в переменной result. После также, как и в предыдущем методе, зацикливается сравнение лексемы, записанной в переменную ор со следующими значениями: «\*», «/»; запрашивается следующая лексема и вызывается метод EvalExp4, с результатом в переменной partialResult. В результате совпадения ор с одним из них выполняется соотвественно перемножение partialResult и result или деление result на partialResult (причем проверяется деление на нуль).

*void EvalExp4(out double result)*

Метод EvalExp4 инициализирует локальную переменную ор, обнуляет значение partialResult, вызывает метод EvalExp5, возвращающий в переменной result значение посчитанного выражения, сравнивает ор с «^» и выполняет запрос следующей лексемы, вызов метода EvalExp4, возвращающего результат в partialResult, записывает в переменную ex значение result. В случае, когда partialResult равен 0, то переменной result присваеваем 1 и выходим из метода, иначе входим в цикл и начинаем перебор с t = partialResult – 1 до t = 0 с декрементированием индекса t: в result присваеваем значение произведения result и ex. Тем самым получаем возведение в степень.

*void EvalExp5(out double result)*

Метод сначала инициализирует локальную переменную ор и присваивает её пустое значение. Следует сравнение

*if (((tokType == Types.DELIMITER) && token == "+" || token == "-" ) ||*

*((tokType == Types.FUNCTION) && (token == "sin" || token == "cos" || token == "tg" || token == "sqrt" || token == "arcsin" ||token == "arccos" || token == "arctg" || token == "cosh" || token == "abs" || token == "sinh" || token == "tgh" || token == "log" || token == "ln"))*

Если значение сравнения истинно, то переменной ор присваиваем значение текущей лексемы, получаем следующую лексему. В случае, если тип следующей лексемы не равен VARIABLE, то вызываем метод EvalExp6, возвращающий нам результат в переменной result и далее в зависимости от значения ор производим вычисления синуса, косинуса, тангенса, квадратного корня, арксинуса, арккосинуса, арктангенса, гиперболических синуса, косинуса и тангенса, модуля, десятичного и обычного логарифмов от значения, полученного методом EvalExp6. Также метод EvalExp5 обрабатывает значения констант пи и эпсилон, и переменной х.

*void EvalExp6(outdouble result)*

Метод обрабатывает выражение в круглых скобках. Вначале происходит сравнение лексемы с открывающей круглой скобкой, далее запрашивается следующая лексема и вызывается метод EvalExp2 (поэтому и называется рекурсивный вызов). После происходит сравнение текущей лексемы с закрывающей круглой скобкой и в случае несоответствия переменной err присвоится UNBALPARENTS, запросится следующая лексема. В противном случае вызовется метод Atom.

*void Atom(outdouble result)*

Метод возвращает значение числа, сравнивая тип текущей лексемы с типом NUMBER, в случае соответствия вернётся само число, иначе в переменную err запишется ошибка SYNTAX и метод вернёт 0. После сравнения запрашивается следующая лексема.

*void SyntaxErr(Errors error)*

Метод обрабатывает сообщения об ошибках.

*void GetToken()*

Метод записывает в переменную token следующую лексему. Сначала происходит присвоение переменным tokType и token значений NONE и "" соответственно. После текущий индекс сравнивается с длинной строки функции и в случае равенства (вся строка пройдена) происходит выход их метода. Далее, пропуская пробелы в тексте функции, проверяем (с помощью метода IsDelim), является ли следующая лексема оператором: присваиваем в переменную token символ строки функции с индексом expIdx, инкриментируем этот индекс и присваиваем tokType тип DELIMITER. Если следующая лексема не является оператором, то проверяем, является ли она переменной или функцией: в цикле с условием пока лексема не является разделителем (знаком какой-либо математической операции) накапливает значение переменной token. Далее token сравнивается со списком определённых в программе функций (см. выше) и если значение этой переменной не совпадает ни с одной строкой-названием этой функции (например, token не равен «sin» И не равен «cos» и т. д.), то tokType присваивается VARIABLE, иначе – FUNCTION.

После проверяем, является ли лексема числом: также, как и в сравнении с функциями, накапливаем значение token в цикле до тех пор, пока лексема не разделитель и присваиваем tokType тип NUMBER.

*bool IsDelim(char c)*

Метод определяет, является ли одним из следующих символов  *+-/\*%^=()* входной символ c. В случае совпадения возвращает истину (true), иначе ложь (false).

3. Компонент MyMessage предназначен для вывода пользователю сообщений разного рода (ошибок, пояснений). Состоит из следующих методов:

*public MyMessage()*

Инициализация компонента.

public static void Show(string msg)

Обработка события появления окна. На вход принимает строку, в которой тегами выделен текст, который необходимо выделить жирным, курсивом или подчеркнуть, таким образом метод обрабатывает форматирование текста.

*static string MyReplase (int StartIdx, string SubStr, string Str)*

Метод заменяет в строке Str подстроку SubStr начиная с индекса StartIdx.

*static string GetLeftRightSelect(string TegStr1, string TegStr2, string str, List<int> leftL, List<int> rightL)*

Метод удаляет из строки теги и заполняет списки leftL и rightL индексами мест в строке, где раньше были теги, что позволяет определить с какого и до какого места выделять тем или иным способом.

*public string FindWort(string wort, string str)*

Метод ищет слово wort в строке str.