Этот документ содержит описание модулей LazExprDraw и LazExprMake.

Модули являются портами для Lazarus модулей ExprDraw и ExprMake для отрисовки математических формул на TCanvas, написанных Антоном Григорьевым (grigorievab@mail.ru) на Delphi 5 и выложенных в 2002 году на delphikingdom.com (http://www.delphikingdom.com/asp/viewitem.asp?catalogID=718). Большая часть описания взята из описания оригинальных ExprDraw и ExprMake с поправкой на изменения и дополнения.

В оригинале использовались функции GDI/Windows API и проприетарный шрифт Times New Roman, что исключало/затрудняло использование кода в Linux. Здесь же используются стандартные возможности TCanvas и шрифта XITS, распространяемого по свободной лицензии SIL Open Font License (https://opensource.org/licenses/OFL-1.1).

Шрифты XITS доступны на github: https://github.com/alif-type/xits. Если нужны только стандартные функции модулей и не требуется особого форматирования текста, то необходимы только два файла с начертаниями Regular и Italic:

XITS-Regular.otf (https://github.com/alif-type/xits/blob/master/XITS-Regular.otf)

XITS-Italic.otf (https://github.com/alif-type/xits/blob/master/XITS-Italic.otf)

Модуль LazExprDraw содержит классы, использующиеся для отображения математических формул.

Модуль LazExprMake используется для создания классов на основании символьной записи формулы.

Описание языка, на котором описываются формулы в LazExprMake, приведено во второй части этого документа, а также вынесено в программу LazExprGuide.

Испытано в Windows 7 и MX Linux 18.2.

Оглавление

1. Модуль LazExprDraw.

Модуль LazExprDraw содержит классы для отображения практически любой формулы. Имена всех классов имеют префикс `TExpr`. Для отображения выражения необходимо построить дерево выражения, узлами и листьями которого являются классы `TExprXXXX`. В простейшем случае (число, переменная) дерево состоит из одного класса. Конкретный класс для каждого узла дерева выбирается исходя из смысловой нагрузки, которую узел несёт в формуле. Так, например, для отображения простой дроби используется класс `TExprRatio`, имеющий две ветви: для числителя и знаменателя. Сами ветви также могут быть составными, глубина дерева ограничивается только ресурсами компьютера. Таким образом, можно построить выражение практически любой сложности.

Данное описание библиотеки LazExprDraw не претендует на полноту, а лишь излагает основные принципы применения. В частности, не описываются многие второстепенные функции. Этого вполне хватает для полноценного использования библиотеки, но её совершенствование потребует более детального знакомства с кодом.

1.1. О разрешениях устройства вывода

Для расчетов размеров отображаемых элементов формул требуются значения горизонтального и вертикального разрешения устройства вывода. В оригинальном ExprDraw для их определения использовалась функция Windows API `GetDeviceCaps`. В LazExprDraw определена процедура:

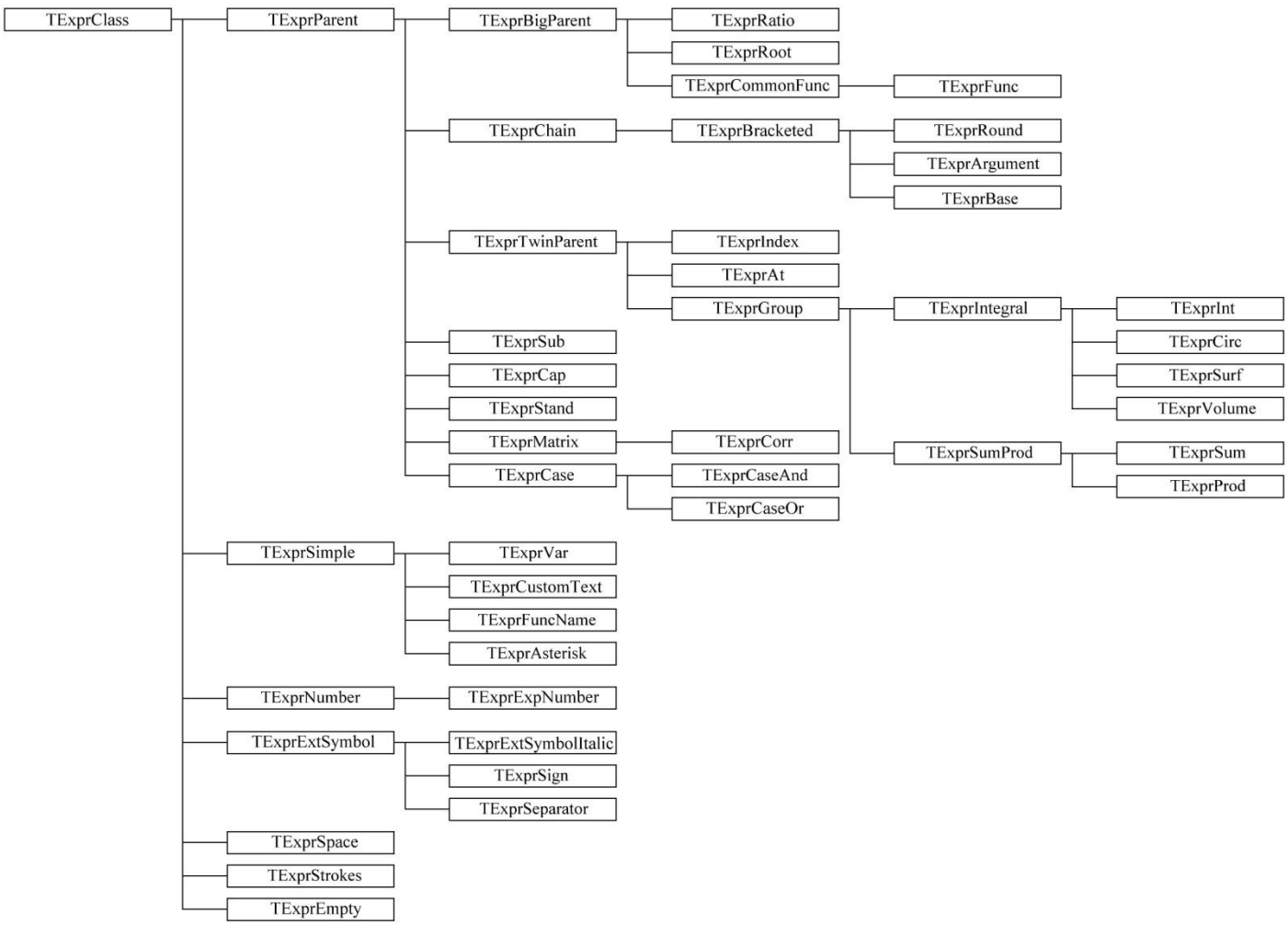
```
SetOutputDPI(const AHorzDPI: Integer = 0; const AVertDPI: Integer = 0);
```

В качестве входных параметров принимает значения вертикального и горизонтального разрешения устройства вывода. Процедура должна быть вызвана до назначения канвы устройства вывода. При вызове без параметров устанавливаются значения разрешений экрана из `Graphics.ScreenInfo.PixelsPerInchX` и `Graphics.ScreenInfo.PixelsPerInchY`.

Для вывода формул на принтер нужно вызывать процедуру с параметрами `Printers.Printer.XDPI` и `Printers.Printer.YDPI`.

При инициализации LazExprDraw устанавливаются значения разрешений для экрана и если предполагается выводить формулы только на экран, то специально вызывать `SetOutputDPI` не нужно.

1.2. Иерархия классов LazExprDraw



1.3. Класс TExprClass

Базовым классом для всех классов модуля LazExprDraw является `TExprClass`. Он поддерживает следующую функциональность: сообщает свои геометрические размеры (свойства `Width` и `Height`), параметры, необходимые для отображения узла (свойства `MidLineUp`, `MidLineDown`, `PowerXPos`, `PowerYPos`, `IndexXPos`, `IndexYPos`, `CapDXRight`, `CapDXLeft`, `CapDY`). Если узел не является корневым, то поле `FParent` содержит указатель на родительский узел. Свойство `Next` используется для построения цепочек классов. Изменение шрифта или канвы передаются далее по цепочке. Свойства `Font` и `Canvas` используются для задания шрифта и канвы отображения. По умолчанию используется шрифт XITS, использование других гарнитур нежелательно, так как все размеры знаков, символов, отступов и т.д. рассчитывались именно для этой гарнитуры. Изменение стилей шрифта не имеет смысла, так как каждый узел сам устанавливает себе стили, определяемые его смысловой нагрузкой. Единственный параметр шрифта, которым имеет смысл управлять - это его размер. Размер задаётся только для корневого узла дерева, далее установки передаются всем ветвям (узлы ветвей не обязательно получают такой же размер шрифта, что и корень: в некоторых случаях предусмотрено изменение размеров шрифта дочерних узлов, например, при добавлении верхнего или нижнего индекса). Канва также назначается корневому узлу и передаётся всем остальным узлам. Геометрические размеры и параметры не могут быть вычислены до того, как дереву назначены шрифт и канва. Параметры имеют следующий смысл:

`MidLineUp, MidLineDown` - расстояния (в пикселях) от средней линии до верхнего и нижнего края выражения. Средняя линия - это линия, на которой должен стоять знак "`-`", если его поставить перед выражением.

`PowerXPos, PowerYPos` - если выражению добавляется верхний индекс, то эти параметры используются при расчёте его положения.

`IndexXPos, IndexYPos` - аналогичная пара для нижнего индекса.

`CapDXLeft, CapDXRight, CapDY` - параметры, использующиеся для позиционирования диакритического знака (вектора, тильды и т.п.), если он ставится над выражением.

Вычисление размеров и параметров выражения производится один раз с помощью функций с префиксом `Calc`, далее оно сохраняется в переменной и повторное вычисление в целях оптимизации не производится. Поле `FToChange` хранит флаги `tcXXXX`, которые показывают, какие параметры уже вычислены, а какие ещё нет. Если происходят изменения, влияющие на значения параметров (например, меняется размер шрифта или канва), соответствующие флаги в `FToChange` взводятся, указывая, что параметры нужно вновь вычислять, а не использовать сохранённые ранее значения.

Отображение выражения осуществляется с помощью функции `Draw`. В качестве параметров ей передаются координаты и флаги выравнивания по горизонтали (`TExprHorAlign`) и вертикали (`TExprVertAlign`). Выравнивать можно по левому краю, правому краю и по центру (по горизонтали) и по верхнему краю, нижнему краю и по средней линии (по вертикали). Отображение выражения реализуется виртуальной функцией `Paint`. Эта функция вызывается функцией `Draw` после соответствующих преобразований координат.

Функция `FTType` возвращает комбинацию флагов `efXXXX`, которые используются в модуле LazExprMake при перемножении выражений с помощью символа "`*`". Данные флаги показывают, может ли выражение быть умножено без знака слева, справа, является ли оно числом и т.д.

Функция `NeedBrackets` возвращает `True`, если появление данного выражения в цепочке выражений вызывает необходимость заключения данной цепочки в скобки при умножении её на другое выражение.

Функция `ArgNeedBrackets` возвращает `True`, если при использовании данного выражения в качестве имени функции (см. класс `TExprCommonFunc`) аргумент должен заключаться в скобки.

Свойство `Color` определяет цвет, которым будет отображаться выражение. Если задан цвет `clNone`, используется цвет родительского узла. Если для корневого узла задан цвет `clNone`, используется чёрный цвет.

Поля `FWX, FWY, FRWX` и `FRWY` используются при вычислении размеров пропусков, линий и т.п. `FWX` примерно равно толщине вертикальной линии знака "`+`", а `FWY` - горизонтальной. В `FRWX` и `FRWY` хранятся величины, линейно зависящие от размера шрифта. Эти величины используются при выводе выражений.

1.4. Класс TExprParent

Класс `TExprParent` является базовым классом для всех компонентов-родителей. Свойство `Son` этого класса хранит указатель на дочерний узел дерева выражения. Создавать экземпляры `TExprParent` не имеет смысла, так как вся функциональность этого класса сводится к управлению шрифтами и канвой дочернего выражения.

Дочернее выражение и выражение, на которое ссылается свойство `Next`, сильно различаются по смыслу. Если класс включен в цепочку, организованную с помощью свойства `Next`, это никак не влияет на его размеры и вид выражения, отображаемого с его помощью. Строго говоря, значение свойства `Next` никак не используется самим классом, если не считать того, что канва, цвет и шрифт передаются далее по цепочке. То, что классы объединены в цепочку, может использоваться родительским узлом первого класса в цепочке. С другой стороны, выражение, задаваемое свойством `Son`, является неотъемлемой частью выражения, определяемого самим классом. В качестве примера можно привести класс, реализующий квадратный корень. `Son` указывает на подкоренное выражение. Когда, например, нужно вычислить размеры выражения, этот класс вызывает `Son.Width` и `Son.Height` и прибавляет к полученным величинам размеры знака извлечения корня. При вызове функции `Draw` этот класс рисует знак квадратного корня, а затем вызывает `Son.Draw` для отображения подкоренного выражения. Таким образом, свойство `Next` используется для организации цепочек классов, причём то, что класс входит в такую цепочку, им самим не используется. А свойство `Son` используется для задания выражения, являющегося частью того выражения, которое задаётся классом.

1.5. Класс TExprBigParent

Класс `TExprBigParent` является наследником класса `TExprParent`. В нём добавлена ссылка на ещё один дочерний узел - свойство `Daughter`. Этот класс является базовым для реализации выражений, в которых есть две составные части. Примером такого выражения может служить простая дробь, в которой одно из дочерних выражений определяет числитель, а другое - знаменатель.

1.6. Класс TExprRatio

Класс `TExprRatio` является наследником класса `TExprBigParent`. Реализует простую дробь. `Son` указывает на числитель, `Daughter` - на знаменатель.

1.7. Класс TExprRoot

Класс `TExprRoot` является наследником класса `TExprBigParent`. Реализует извлечение корня. `Son` указывает на подкоренное выражение, `Daughter` - на показатель степени корня. Для `Daughter` допускается значение `nil`. В этом случае показатель степени перед корнем не ставится (квадратный корень).

1.8. Класс TExprCommonFunc

Класс `TExprCommonFunc` является наследником класса `TExprBigParent`. Он реализует "общую" функцию, в качестве "имени" которой может использоваться любое выражение. `Son` указывает на "имя" функции, `Daughter` - на её аргумент.

1.9. Класс TExprFunc

Класс `TExprFunc` является наследником `TExprCommonFunc`. Он реализует традиционную функцию, имя которой является обычной комбинацией букв. Конструктор сам создаёт класс нужного типа для отображения имени функции. Если имя имеет длину 1 символ, то используется класс `TExprVar`, если оно длиннее – класс `TExprFuncName`. В первом случае имя функции пишется курсивом, а аргумент всегда заключается в скобки. Во втором случае имя функции пишется прямым шрифтом, а аргумент заключается в скобки лишь при необходимости.

1.10. Класс TExprChain

Класс `TExprChain` является наследником `TExprParent`. Этот класс служит для отображения цепочки выражений. Первым в цепочке выводится `Son`, затем `Son.Next`, затем - `Son.Next.Next` и так далее, пока не будет достигнут конец цепочки.

1.11. Класс TExprBracketed

Класс `TExprBracketed` является наследником `TExprChain`. Он выводит цепочку выражений, заключённую в скобки. Виды скобок задаются типом

```
TExprBracketStyle = (ebNone,ebRound,ebSquare,ebFigure,ebModule,ebNorm,ebAngle,ebFloor,ebCeil);
```

Значение	Скобка левая	Скобка правая
ebNone	нет	нет
ebRound	(	)
ebSquare	[	]
ebFigure	{	}
ebModule	∣	∣
ebNorm	‖	‖
ebAngle	⟨	⟩
ebFloor	⌊	⌋
ebCeil	⌈	⌉

Скобки могут быть непарными (например, открывающая скобка круглая, а закрывающая - квадратная). Возможен также вариант, когда скобка стоит только с одной стороны выражения, а с другой `ebNone`. Выражение заключается в скобки только в том случае, если функция `IsBracketed` возвращает `True`.

1.12. Класс TExprRound

Класс `TExprRound` является наследником `TExprBracketed`. Он переопределяет функцию `FTType`. Этот класс используется библиотекой LazExprMake при сокращении лишних скобок.

1.13. Класс TExprArgument

Класс TExprArgument является наследником TExprBracketed. В нём переопределён конструктор таким образом, чтобы скобки всегда были круглыми. Кроме того, переопределена функция IsBracketed таким образом, чтобы она возвращала True только в том случае, если хотя бы один из узлов цепочки, на которую указывает Son, нуждается в скобках (т.е. его функция NeedBrackets возвращает True). В скобках нуждаются, например, узлы, реализующие знаки "+" и "-". Таким образом, если цепочка выражений представляет собой сумму нескольких слагаемых, оно заключается в скобки, если это произведение нескольких множителей - не заключается. Это совпадает с тем, как записываются аргументы функций (например, cos 2x, но cos(x+y)). Этот же класс используется для реализации отдельных множителей: если в составе множителя есть операции сложения или вычитания, он заключается в скобки, если нет, то не заключается.

В класс также добавлена функция SetBrackets, которая заставляет рисовать скобки независимо от того, есть ли в цепочке нуждающиеся в скобках узлы.

Если класс TExprArgument используется как аргумент функции (в этом случае его родительский узел является экземпляром класса TExprCommonFunc), то вызывается соответствующая функция родительского класса для определения, нужны ли скобки аргументу.

При построении дерева вручную для конкретного выражения всегда заранее известно, нужно ли заключать какие-то его части в скобки или нет. В этом случае лучше использовать классы TExprBracketed и TExprChain. Но при автоматизированном построении дерева (например, с помощью модуля LazExprMake) данный класс может быть очень полезным.

1.14. Класс TExprBase

Класс TExprBase является наследником TExprBracketed. Он отображает выражение, заключённое в круглые скобки, если цепочка, на которую указывает Son, содержит более одного узла. В противном случае выражение в скобки не заключается. Это совпадает с тем, как должна отображаться база при возведении в степень (например, x^2, но (2x)^2). Класс TExprBase, как и TExprArgument, предназначен только для автоматизации построения дерева.

1.15. Класс TExprTwinParent

Класс TExprTwinParent является наследником класса TExprParent. В нём добавлены указатели на два дочерних узла-близнеца. Примером выражения, для которого может понадобиться такой класс, является индексированное выражение. Son ссылается на то выражение, которому добавляются индексы, а Twin1 и Twin2 – на нижний и верхний индексы.

1.16. Класс TExprIndex

Класс TExprIndex является наследником TExprTwinParent. Используется для отображения выражений с верхним или нижним индексом. Son указывает на выражение, которому добавляются индексы, Twin1 - нижний индекс, Twin2 - верхний индекс. Twin1 или Twin2 могут быть равны nil - в этом случае выражению добавляется только один индекс. Верхний индекс также используется как показатель степени.

1.17. Класс TExprAt

Класс TExprAt является наследником класса TExprTwinParent. Реализует "значение при условии" или значение "от … до …". Отображается это следующим образом: после выражения, на которое указывает Son, ставится вертикальная черта, справа от неё внизу пишется выражение, на которое указывает Twin1, а сверху Twin2. Значение Twin2 может быть равно nil, тогда оно не рисуется.

1.18. Класс TExprGroup

Класс TExprGroup является наследником класса TExprTwinParent. Это базовый класс для всех выражений типа суммы или интеграла. Все эти выражения строятся одинаково: Son указывает на выражение, стоящее после знака суммы или интеграла, Twin1 - стоящее под знаком, Twin2 - над ним. Twin1 и Twin2 могут быть равны nil одновременно или по одному. В этом случае под знаком и/или над ним ничего не пишется.

1.19. Класс TExprIntegral

Класс TExprIntegral является наследником класса TExprGroup. Это базовый класс для всех типов интегралов.

1.20. Класс TExprInt

Класс TExprInt является наследником TExprIntegral. Реализует однократный или многократный интеграл. Параметр конструктора Mult задаёт кратность интеграла. Если Mult <=0, рисуется символ интеграла неопределённой кратности (два символа интеграла, многоточие, ещё один символ интеграла).

1.21. Класс TExprCirc

Класс TExprCirc является наследником TExprIntegral. Реализует интеграл по замкнутому контуру.

1.22. Класс TExprSurf

Класс TExprSurf является наследником TExprIntegral. Реализует интеграл по замкнутой поверхности.

1.23. Класс TExprVolume

Класс TExprVolume является наследником TExprIntegral. Реализует интеграл по замкнутому объему.

1.24. Класс TExprSumProd

Класс TExprSumProd является наследником класса TExprGroup. Это базовый класс для выражений суммы и произведения.

1.25. Класс TExprSum

Класс TExprSum является наследником TExprSumProd. Реализует сумму (с помощью греческой буквы сигма).

1.26. Класс TExprProd

Класс TExprProd является наследником TExprSumProd. Реализует произведение (с помощью греческой буквы пи).

1.27. Класс TExprSub

Класс TExprSub является наследником TExprParent. Этот класс реализует отображение функций по типу предела (lim): выводит имя функции FuncName а под ним - выражение, на которое указывает Son. Класс предназначен для совместного использования с TExprCommonFunc. TExprCommonFunc.Son указывает на экземпляр TExprSub, а TExprCommonFunc.Daughter - на выражение, для которого вычисляется функция (например, предел).

1.28. Класс TExprCap

Класс TExprCap является наследником TExprParent. Рисует над выражением, на которое указывает Son, диакритический знак. Виды знаков задаются типом

```
TExprCapStyle = (ecPoints,ecVector,ecCap,ecTilde,ecLine);
```

Значение	Диакритический знак
ecPoints	¨
ecVector	→
ecCap	^
ecTilde	~
ecLine	–

Конструктор содержит параметр Count, который задаёт количество точек при стиле ecPoints. При прочих стилях этот параметр игнорируется.

1.29. Класс TExprStand

Класс TExprStand является наследником TExprParent. Предназначен для вывода нескольких выражений в виде столбца. Первым выводится выражение, на которое указывает Son, под ним - Son.Next, ещё ниже - Son.Next.Next и так далее до конца цепочки. Параметр конструктора HorAlign показывает, будут ли выражения выравниваться по левому краю, по правому краю или по центру.

1.30. Класс TExprMatrix

Класс TExprMatrix является наследником TExprParent. Предназначен для вывода матрицы. Размеры матрицы определяются параметрами конструктора HorSize и VertSize. Левый верхний элемент матрицы задаётся выражением Son, второй слева в верхней строке - Son.Next и так далее слева направо сверху вниз. Если цепочка длиннее, чем HorSize*VertSize, при отображении матрицы лишние элементы игнорируются (однако они не игнорируются при подсчёте размеров ячейки матрицы, что может привести к искажению вида матрицы). Если цепочка содержит меньше элементов, чем HorSize*VertSize, ячейки, на которые не хватило узлов, будут пустыми.

1.31. Класс TExprCorr

Класс TExprCorr является наследником TExprMatrix. Предназначен для вывода таблицы соответствия из двух столбцов, разделенных вертикальной чертой. Первый элемент таблицы (слева от черты) задается выражением Son, соответствующее ему значение (справа от черты) - Son.Next, второй элемент - Son.Next.Next, его соответствие - Son.Next.Next.Next и так далее.

1.32. Класс TExprCase

Класс TExprCase является наследником TExprParent. Это базовый класс для отображения вариантной конструкции: несколько выражений обводятся слева скобкой, после каждого ставится условие. Son указывает на первый вариант, Son.Next - на условие его применимости, Son.Next.Next - на второй вариант, Son.Next.Next.Next - на условие его применимости и т.д. Если цепочка содержит нечётное число узлов, последний вариант остаётся без условия.

1.33. Класс TExprCaseAnd

Класс TExprCaseAnd является наследником TExprCase. Предназначен для отображения вариантной конструкции с условием «И» - используется фигурная скобка.

1.34. Класс TExprCaseOr

Класс TExprCaseOr является наследником TExprCase. Предназначен для отображения вариантной конструкции с условием «ИЛИ» - используется квадратная скобка.

1.35. Класс TExprSimple

Класс TExprSimple служит для отображения плоского текста. Текст выводится прямым шрифтом.

1.36. Класс TExprVar

Класс TExprVar является наследником TExprSimple. Главное отличие - текст выводится не прямым шрифтом, а курсивом, как это принято при отображении переменных в выражениях.

1.37. Класс TExprCustomText

Класс TExprCustomText является наследником TExprSimple. С его помощью можно вывести текст с любыми атрибутами и любой гарнитурой.

1.38. Класс TExprFuncName

Класс TExprFuncName является наследником класса TExprSimple. Он реализует имя функции в том случае, когда оно пишется прямым шрифтом. Переопределена функция ArgNeedBrackets таким образом, чтобы аргумент не заключался в скобки, если это не является необходимым. Чтобы аргумент мог воспользоваться результатом, возвращаемым функцией TExprFuncName.ArgNeedBrackets, он должен иметь тип TExprArgument.

1.39. Класс TExprAsterisk

Класс TExprAsterisk является наследником класса TExprSimple. Предназначен для отображения звёздочки *, смещённой вниз по вертикали. Эта звёздочка используется в качестве верхнего индекса (например, для обозначения сопряжённой величины).

1.40. Класс TExprNumber

Класс TExprNumber предназначен для отображения чисел в обычной или экспоненциальной форме. Если параметр конструктора ExpForm равен True, используется экспоненциальная форма, в противном случае - обычная.

1.41. Класс TExprExpNumber

Класс TExprExpNumber является наследником TExprNumber. Служит для отображения чисел с возможностью гибкого управления форматированием числа.

1.42. Класс TExprExtSymbol

Класс TExprExtSymbol используется для отображения одиночных символов, заданных десятичным кодом в кодировке UTF-16BE, выводятся прямым шрифтом.

1.43. Константы кодов символов

Для простоты использования класса TExprExtSymbol и его наследников определены константы символов:

Константа	Символ	Константа	Символ	Константа	Символ
esApproxLess	≲	esLessOrEqual	≤	esMuchLess	≪
esApproxGreater	≳	esGreaterOrEqual	≥	esMuchGreater	≫
esLess	<	esPlus	+	esPlusMinus	±
esGreater	>	esMinus	−	esMinusPlus	∓
esEqual	=	esIdentical	≡	esApproxEqual	≅
esNotEqual	≠	esNotIdentical	≢	esAlmostEqual	≈
esMultiply	·	esSlash	/	esBackSlash	\
esCrossMultiply	×	esDotsDivide	÷	esParallel	∥
esSemicolon	;	esDot	.	esNotParallel	∦
esComma	,	esTilde	~	esPerpendicular	⊥
esExists	∃	esBelongs	∈	esColon	:

esNotExists	∄	esNotBelongs	∉	esIntersection	∩
esSubSet	⊂	esSuperSet	⊃	esUnion	∪
esNotSubSet	⊄	esNotSuperSet	⊅	esAnd	⋀
esProportional	∝	esSum	∑	esOr	⋁
esInfinity	∞	esProd	∏	esNot	¬
esNatural	ℕ	esStroke	'	esXor	⊻
esReal	ℝ	esPartDiff	∂	esForAll	∀
esRational	ℚ	esAngle	∠	esArc	⌒
esEntire	ℤ	esNabla	∇	esEmptySet	∅
esComplex	ℂ	esPlanck	ℏ	esEllipsis	…
esQuaternion	ℍ	esLambdaSpec	ƛ	esEllipsVert	⋮
esProjective	ℙ	esAsterisk	*	esEllipsHoriz	⋯
esArrowLeft	←	esDoubleArrowLeft	⇐	esEllipsDiagUp	⋰
esArrowRight	→	esDoubleArrowRight	⇒	esEllipsDiagDown	⋱
esArrowUp	↑	esDoubleArrowUp	⇑	esLeftAngleBracket	⟨
esArrowDown	↓	esDoubleArrowDown	⇓	esRightAngleBracket	⟩
esArrowLeftRight	↔	esDoubleArrowLeftRight	⇔	esLeftFloorBracket	⌊
esArrowUpDown	↕	esDoubleArrowUpDown	⇕	esRightFloorBracket	⌋
esArrowLeftUp	↖	esDoubleArrowLeftUp	⇖	esLeftCeilBracket	⌈
esArrowRightUp	↗	esDoubleArrowRightUp	⇗	esRightCeilBracket	⌉
esArrowRightDown	↘	esDoubleArrowRightDown	⇘	esDivide	|
esArrowLeftDown	↙	esDoubleArrowLeftDown	⇙	esNotDivide	∤
esArrowLeftNot	↚	esDoubleArrowLeftNot	⇍	esBarArrowLeft	↤
esArrowRightNot	↛	esDoubleArrowRightNot	⇏	esBarArrowRight	↦
esArrowLeftRightNot	↮	esDoubleArrowLeftRightNot	⇎	esBarArrowUp	↥
esInt	∫	esSurf	∯	esBarArrowDown	↧
esCirc	∮	esVolume	∰	esTriangle	△
esAlphaBig	Α	esAlphaSmall	α	esQuadrate	□
esBetaBig	Β	esBetaSmall	β	esRectangle	▭
esGammaBig	Γ	esGammaSmall	γ	esCircle	○
esDeltaBig	Δ	esDeltaSmall	δ	esParallelogram	▱
esEpsilonBig	Ε	esEpsilonSmall	ε	esRhomb	◇
esZetaBig	Ζ	esZetaSmall	ζ	esBegin	◀
esEtaBig	Η	esEtaSmall	η	esEnd	▶
esThetaBig	Θ	esThetaSmall	θ	esSimilar	≈
esIotaBig	Ι	esIotaSmall	ι	esDegree	°
esKappaBig	Κ	esKappaSmall	κ	esPlusCircle	⊕
esLambdaBig	Λ	esLambdaSmall	λ	esMinusCircle	⊖
esMuBig	Μ	esMuSmall	μ	esCrossCircle	⊗
esNuBig	Ν	esNuSmall	ν	esSlashCircle	⊘
esXiBig	Ξ	esXiSmall	ξ	esDotCircle	⊙

esOmicronBig	Ο	esOmicronSmall	ο	esCircleCircle	⊚
esPiBig	Π	esPiSmall	π	esAsteriskCircle	⊛
esRhoBig	Ρ	esRhoSmall	ρ	esEqualCircle	⊜
esSigmaBig	Σ	esSigmaSmall	σ	esHarpArrowsLeftRight	⥊
esTauBig	Τ	esTauSmall	τ	esHarpArrowsRightLeft	⥋
esUpsilonBig	Υ	esUpsilonSmall	υ	esTwoArrowsRightLeft	⇄
esPhiBig	Φ	esPhiSmall	φ	esTwoArrowsLeftRight	⇆
esChiBig	Χ	esChiSmall	χ	esTwoArrowsUpDown	⇅
esPsiBig	Ψ	esPsiSmall	ψ	esTwoArrowsDownUp	⇵
esOmegaBig	Ω	esOmegaSmall	ω	esTwoArrowsLeft	⇇
esThetaOther	ϑ	esSigmaOther	ς	esTwoArrowsRight	⇉
esDotEqual	≐	esDotsEqualLeftRight	≒	esTwoArrowsUp	⇈
esDotsEqualCenter	≑	esDotsEqualRightLeft	≓	esTwoArrowsDown	⇊
esColonEqual	≔	esEqualColon	≕	esThreeArrows	⇶

1.44. Класс TExprExtSymbolItalic

Класс `TExprExtSymbolItalic` является наследником `TExprExtSymbol`. Выводит символы наклонным шрифтом.

1.45. Класс TExprSign

Класс `TExprSign` является наследником `TExprExtSymbol`. Переопределяет функцию `NeedBrackets`, которая помогает правильно расставить скобки, поэтому при автоматическом построении выражений даже для простых знаков (например, "+", "-" и т.д.) рекомендуется пользоваться классом `TExprSign`, а не `TExprExtSymbol` или `TExprSimple`.

1.46. Класс TExprSeparator

Класс `TExprSeparator` является наследником `TExprExtSymbol`. Предназначен для отображения разделителей – запятой и точки с запятой. Переопределена функция `NeedBrackets` (чтобы выражения, содержащие разделитель, заключались при необходимости в скобки).

1.47. Класс TExprSpace

Класс `TExprSpace` предназначен для вставки в формулу пустого пространства. Параметр конструктора `Count` задаёт ширину этого пробела в единицах ширины (одна единица ширины примерно равна ширине вертикальной линии в символе «+»).

1.48. Класс TExprStrokes

Класс `TExprStrokes` предназначен для отображения чёрточек, которыми обозначается производная. Используется совместно с классом `TExprIndex`: функция задаётся в `TExprIndex.Son`, `TExprStrokes` используется как верхний индекс.

1.49. Класс TExprEmpty

Класс `TExprEmpty` используется для отображения выражения с нулевой шириной, но высотой, равной высоте обычного текста. Используется вместе с `TExprIndex` для создания индексов, «висящих в воздухе».

1.50. Примеры использования

Пример 1: $a + b\left(\frac{c}{d} + 1\right)$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr:= TExprRatio.Create(TExprVar.Create('c'),TExprVar.Create('d'));
  // Теперь Expr содержит дробь c/d
  Expr.AddNext(TExprSign.Create(esPlus));
  Expr.AddNext(TExprNumber.Create(1));
  // Теперь Expr является началом цепочки, в которой есть ещё знак "+" и единица
  Expr:= TExprBracketed.Create(Expr,ebRound,ebRound);
  // Теперь Expr содержит выражение, которое заключено в скобки
```

```
  Expr2:= TExprVar.Create('a');
  Expr2.AddNext(TExprSign.Create(esPlus));
  Expr2.AddNext(TExprVar.Create('b'));
  Expr2.AddNext(Expr);
  // Теперь Expr2 содержит всю цепочку выражений, которая нам нужна.
  Expr:= TExprChain.Create(Expr2);
  // Используем класс TExprChain, предназначенный для отображения цепочки как единого целого.
  // Построение дерева выражения закончено.
  Expr.Font.Size:=12; // Теперь размер шрифта установлен
  // Для отображения выражения на форме нужно установить его канву и вызвать Draw
  Expr.Canvas:= Form1.Canvas;
  Expr.Draw(5,5,ehLeft,evTop);
  // Можно вывести это же выражение на принтер
  SetOutputDPI(Printer.XDPI, Printer.YDPI);
  Printer.BeginDoc;
  Expr.Canvas:= Printer.Canvas;
  Expr.Draw(50,50,ehLeft,evTop);
  Printer.EndDoc;
  // Можно поменять шрифт и снова вернуться к канве формы
  SetOutputDPI;
  Expr.Canvas:= Form1.Canvas;
  Expr.Font.Height:=24;
  Expr.Draw(Form1.ClientWidth-5,Form1.ClientHeight-5,ehRight,evBottom);
  Expr.Free; // или FreeAndNil(Expr)
  // После использования нужно очистить дерево. Деструктор автоматически вызывает
  // деструкторы для всех узлов дерева. Expr2.Free в данном случае вызывать не нужно,
  // так как дерево Expr2 стало частью дерева Expr, поэтому при вызове Expr.Free оно
  // также будет очищено.
end;
```

В данном примере, кроме собственно построения дерева выражения, показано, как это дерево потом использовать, однако простейший способ, показанный здесь, не всегда позволяет достичь требуемой производительности. Каждый раз при смене канвы и шрифта приходится заново рассчитывать все размеры выражения, поэтому менять их нужно как можно реже. Кроме того, даже если все размеры уже рассчитаны, вывод формулы также происходит сравнительно медленно. Если формула предназначена для многократного отображения только на экране, лучше построить дерево один раз, вывести формулу на `TBitmap`, затем освободить дерево формулы, а затем каждый раз отображать полученный растр.

Следует также отметить, что после вызова `TExprClass.Draw` обычно меняется перо, кисть и шрифт, выбранные в данной канве. Если отображение формул перемежается выводом графических примитивов, каждый раз после вызова Draw следует заново восстанавливать кисть, перо и шрифт.

Пример 2: $\cos^2\varphi+\sin^2\varphi=1$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr2:= TExprIndex.Create(TExprSimple.Create('cos'),nil,TExprNumber.Create(2,False));
  // Expr2 содержит "косинус квадрат"
  Expr:= TExprCommonFunc.Create(Expr2,TExprExtSymbol.Create(esPhiSmall));
  // Expr содержит общую функцию, в которой "именем" является "косинус квадрат", а аргументом – фи
  Expr.AddNext(TExprSign.Create(esPlus));
  // В цепочку добавлен +
  Expr2:= TExprIndex.Create(TExprSimple.Create('sin'),nil,TExprNumber.Create(2,False));
  Expr.AddNext(TExprCommonFunc.Create(Expr2,TExprExtSymbol.Create(esPhiSmall)));
  // В цепочку добавлен "синус квадрат фи"
  Expr.AddNext(TExprSign.Create(esEqual));
  // В цепочку добавлено =
  Expr.AddNext(TExprNumber.Create(1,False));
  // В цепочку добавлена 1
  Expr:= TExprChain.Create(Expr);
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример 3: $y=e^x$

```
var
  Expr: TExprClass;
begin
  Expr:= TExprVar.Create('y');
  Expr.AddNext(TExprSign.Create(esEqual));
  Expr.AddNext(TExprIndex.Create(TExprVar.Create('e'),nil,TExprVar.Create('x')));
  Expr:= TExprChain.Create(Expr);
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример 4: $y = \sqrt[3]{x-1}$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr:= TExprVar.Create('x');
  Expr.AddNext(TExprSign.Create(esMinus));
  Expr.AddNext(TExprNumber.Create(1,False));
  Expr:= TExprChain.Create(Expr);
  Expr2:= TExprVar.Create('y');
  Expr2.AddNext(TExprSign.Create(esEqual));
  Expr2.AddNext(TExprRoot.Create(Expr,TExprNumber.Create(3,False)));
  Expr:= TExprChain.Create(Expr2);
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример 5: $\lim\limits_{x \to 0} \frac{1}{x} = \pm\infty$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr2:= TExprVar.Create('x');
  Expr2.AddNext(TExprSign.Create(esArrowRight));
  Expr2.AddNext(TExprNumber.Create(0,False));
  Expr2:= TExprChain.Create(Expr2);
  Expr2:= TExprSub.Create(Expr2, 'lim');
  Expr:= TExprRatio.Create(TExprNumber.Create(1,False),TExprVar.Create('x'));
  Expr:= TExprCommonFunc.Create(Expr2,Expr);
  Expr.AddNext(TExprSign.Create(esEqual));
  Expr.AddNext(TExprSign.Create(esPlusMinus));
  Expr.AddNext(TExprExtSymbol.Create(esInfinity));
  Expr:= TExprChain.Create(Expr);
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример 6: $(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr2:= TExprFunc.Create('f',TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound));
  Expr2.AddNext(TExprFunc.Create('g',TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound)));
  Expr2:= TExprBracketed.Create(Expr2,ebRound,ebRound);
  Expr:= TExprIndex.Create(Expr2,nil,TExprStrokes.Create(1));
  Expr.AddNext(TExprSign.Create(esEqual));
  Expr2:= TExprIndex.Create(TExprVar.Create('f'),nil,TExprStrokes.Create(1));
  Expr2:=TExprCommonFunc.Create(Expr2,TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound));
  Expr.AddNext(Expr2);
  Expr.AddNext(TExprFunc.Create('g',TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound)));
  Expr.AddNext(TExprSign.Create(esPlus));
  Expr.AddNext(TExprFunc.Create('f',TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound)));
  Expr2:= TExprIndex.Create(TExprVar.Create('g'),nil,TExprStrokes.Create(1));
  Expr2:=TExprCommonFunc.Create(Expr2,TExprBracketed.Create(TExprVar.Create('x'),ebRound,ebRound));
  Expr.AddNext(Expr2);
  Expr:= TExprChain.Create(Expr);
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример 7: $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} a_{ij}x_i x_j$

```
var
  Expr,Expr2: TExprClass;
begin
  Expr:= TExprIndex.Create(TExprVar.Create('a'),TExprVar.Create('ij'),nil);
  Expr.AddNext(TExprIndex.Create(TExprVar.Create('x'),TExprVar.Create('i'),nil));
  Expr.AddNext(TExprIndex.Create(TExprVar.Create('x'),TExprVar.Create('j'),nil));
  Expr:= TExprChain.Create(Expr);
  Expr2:= TExprVar.Create('j');
  Expr2.AddNext(TExprSign.Create(esEqual));
  Expr2.AddNext(TExprNumber.Create(1,False));
```

```
  Expr2:= TExprChain.Create(Expr2);
  Expr:= TExprSum.Create(Expr,Expr2,TExprVar.Create('n'));
  Expr2:= TExprVar.Create('i');
  Expr2.AddNext(TExprSign.Create(esEqual));
  Expr2.AddNext(TExprNumber.Create(1,False));
  Expr2:= TExprChain.Create(Expr2);
  Expr:= TExprSum.Create(Expr,Expr2,TExprVar.Create('n'));
  // Дерево построено, здесь должен быть код, отображающий его
  Expr.Free;
end;
```

Пример неправильного построения дерева выражения

Если выражение содержит повторяющиеся части, один и тот же узел не должен использоваться в нескольких местах. Иначе возникнут проблемы при изменении шрифта, а при освобождении дерева этот узел программа попытается освободить дважды, и в результате возникнет Access Violation.

Проиллюстрируем это правило на примере выражения: $e^x + x$

Пример неправильного построения дерева:

```
  Expr2:= TExprVar.Create('x');
  Expr:= TExprIndex.Create(TExprVar.Create('e'),nil,Expr2); // Expr2 используется первый раз
  Expr.AddNext(TExprSign.Create(esPlus));
  Expr.AddNext(Expr2); // Expr2 используется во второй раз
  Expr:= TExprChain.Create(Expr);
```

Правильный пример: нужно создать ещё один узел для отображения «x», а не использовать повторно имеющийся

```
  Expr2:= TExprVar.Create('x');
  Expr:= TExprIndex.Create(TExprVar.Create('e'),nil,Expr2);
  Expr.AddNext(TExprSign.Create(esPlus));
  Expr2:= TExprVar.Create('x'); // Чтобы не использовать узел повторно, создаём другой такой же
  Expr.AddNext(Expr2);
  Expr:= TExprChain.Create(Expr);
```

2. Модуль LazExprMake

Модуль LazExprMake предназначен для автоматизации построения формул с помощью классов модуля LazExprDraw.

Модуль содержит класс `TExprBuilder`, который строит дерево выражения по строке, содержащей выражение в текстовом виде. Подробно синтаксис исходного выражения с примерами приведен ниже, а также вынесен в виде справочника в программу LazExprGuide.

2.1. Класс TExprBuilder

Класс `TExprBuilder` имеет две функции для построения выражений: `BuildExpr` и `SafeBuildExpr`. Обе эти функции в качестве параметра принимают строку и возвращают указатель на корень построенного дерева.

Если выражение содержит ошибки, возникает исключение `EincorrectExpr`. Если исключение возникло во время работы функции `BuildExpr`, в памяти остаётся мусор от частично построенного дерева, поэтому функцией `BuildExpr` можно пользоваться только для заведомо корректных строк.

Функция `SafeBuildExpr` сначала осуществляет прогонку алгоритма построения дерева вхолостую. На этом этапе дерево не строится, но исключения при неверном синтаксисе возникают. И только если первый этап пройден успешно, функция приступает к следующему этапу: построению дерева.

Дерево, созданное в результате работы функций `BuildExpr` и `SafeBuidExpr`, после использования надо освободить, вызвав `Free`.

Модуль LazExprMake также содержит глобальные функции `BuildExpr` и `SafeBuildExpr`. Они создают экземпляр класса `TExprBuilder`, вызывают его одноимённую функцию, а затем освобождают созданный `TExprBuilder`. Эти функции можно использовать, если нужно создать только одно дерево, в противном случае оптимальнее создать вручную класс `TExprBuilder`, вызвать необходимое число раз `TExprBuilder.BuildExpr` или `TExprBuilder.SafeBuildExpr`, а затем вручную его удалить.

Примеры создания выражений с использованием LazExprMake (см. п.1.50.) можно переписать так:

Пример 1: $a+b\left(\frac{c}{d}+1\right)$

```
Expr:= BuildExpr('a+b*(c/d+1)');
```

Пример 2: $\cos^2\varphi+\sin^2\varphi=1$

```
Expr:= BuildExpr('cos(phi)^2+sin(phi)^2=1');
```

Пример 3: $y=e^x$

```
Expr:= BuildExpr('y=Pow(e,x)');
```

Пример 4: $y=\sqrt[3]{x-1}$

```
Expr:= BuildExpr('y=Root(3,x-1)');
```

Пример 5: $\lim_{x\to 0}\frac{1}{x}=\pm\infty$

```
Expr:= BuildExpr('lim(x->0,1/x)=+-Inf');
```

Пример 6: $(f(x)g(x))'=f'(x)g(x)+f(x)g'(x)$

```
Expr:= BuildExpr('(f(x)*g(x))`=f(x)`*g(x)+f(x)*g(x)`');
```

Пример 7: $\sum_{i=1}^{n}\sum_{j=1}^{n}a_{ij}x_ix_j$

```
Expr:= BuildExpr('Sum(Sum(a_(i*j)*x_i*x_j,j=1,n),i=1,n)');
```

2.2. Язык описания формул

Строка, описывающая формулу состоит из выражений. К простым выражениям относятся идентификаторы, токены и числовые константы.

Токены – это зарезервированные слова, использующиеся для обозначения специальных символов (пример 1). Регистр символов при написании токенов не важен, за исключением токенов, обозначающих греческие буквы. Если первый символ такого токена имеет верхний регистр, то получается заглавная буква, если нижний – строчная.

Идентификаторы воспроизводятся так, как они написаны в тексте. Идентификаторы могут состоять из английских и русских букв и цифр, причём начинаться идентификатор должен с буквы. Если идентификатор начинается с английской буквы, он выводится наклонным шрифтом (пример 2), если с русской – прямым (пример 3). Вывести английский текст прямым шрифтом можно с помощью функции `String`. Если в конце идентификатора стоят цифры, а

`TExprBuilder.VarAutoIndex:=True` (по умолчанию), то они рассматриваются как нижний индекс (пример 4). Если цифры стоят после токена, то они также рассматриваются как нижний индекс (пример 5).

Числовые константы записываются в обычном для Паскаля синтаксисе (пример 6). Допускается использование английской «e» или «E» для обозначения экспоненциальной части числа. Оптимальный формат вывода числа подбирается автоматически. Если он по каким-то причинам не устраивает, можно воспользоваться символом «#» или функцией `Num`.

№ примера	Строка	Результат
1	`'Alpha, alpha'`	A, α
2	`'EnglEnc'`	*EnglEnc*
3	`'РУС'`	РУС
4	`'x12'`	x_{12}
5	`'beta0'`	β_0
6	`'1.234,0.7e9,1234678'`	1.234, 700000000, 1234678

Сложные выражения состоят из операндов, разделённых знаками операций. Операнды могут быть одного из следующих видов:

1. Простое выражение, т.е. числовая константа (в том числе начинающаяся с символа «#»), идентификатор или токен.
2. Выражение, заключенное в круглые, квадратные, прямые или фигурные скобки.
3. Функция.

№ примера	Строка	Результат
7А	`'B_(2*k)=((-1)^(n-1)**2*(2*k)!/(2*pi)^(2*k))*sum(1/n^(2*k),n=1,inf)'`	$B_{2k}=\frac{(-1)^{n-1}2(2k)!}{(2\pi)^{2k}}\sum_{n=1}^{\infty}\frac{1}{n^{2k}}$
7Б	`'line(u_i^2)=(2*planck^2/M/k/Theta)*((T^2/Theta^2)*int(x*diff(x)/(e^x-1),0,T//Theta)+1/4)'`	$\overline{u_i^2}=\frac{2h^2}{Mk\Theta}\left(\frac{T^2}{\Theta^2}\int_0^{T/\Theta}\frac{x\,dx}{e^x-1}+\frac{1}{4}\right)$

2.2.1. Знаки операций

Знак	Описание	№ примера	Строка	Результат
+	Сложение.	8	`'a+b'`	$a+b$
–	Вычитание.	9	`'a-b'`	$a-b$
*	Умножение. Проверяет, могут ли сомножители быть перемноженными без знака. Если могут, то знак умножения не используется (пример 10). Сомножители могут быть переставлены местами, чтобы обеспечить умножение без символа (пример 11). Если среди сомножителей несколько чисел, то числа объединяются в одно (пример 12). Если перемножение без символа невозможно ни при каком порядке множителей, используется символ точка (пример 13).	10	`'5*x'`	$5x$
		11	`'y*2'`	$2y$
		12	`'2*a*3'`	$6a$
		13	`'sin(x)*cos(x)'`	$\sin x \cdot \cos x$
**	Умножение. Сомножители перемножаются без знака независимо от того, допустимо ли такое перемножение. Перестановка множителей не производится.	14	`'5**x'`	$5x$
		15	`'y**2'`	$y2$
		16	`'2**a**3'`	$2a3$
		17	`'sin(x)**cos(x)'`	$\sin x \cos x$
*.	Умножение. Сомножители перемножаются с помощью знака точки. Перестановка множителей не производится.	18	`'5*.x'`	$5 \cdot x$
		19	`'y*.2'`	$y \cdot 2$
		20	`'2*.a*.3'`	$2 \cdot a \cdot 3$
		21	`'sin(x)*.cos(x)'`	$\sin x \cdot \cos x$
*+	Умножение. Сомножители перемножаются с помощью знака косого креста. Перестановка множителей не	22	`'5*+x'`	$5 \times x$
		23	`'y*+2'`	$y \times 2$
		24	`'2*+a*+3'`	$2 \times a \times 3$

	производится.	25	'sin(x)*+cos(x)'	$\sin x \times \cos x$
/	Деление. Всегда используется деление в виде простой дроби. В сложных выражениях, использующих различные символы умножения и деления в произвольном порядке, распределяет множители между числителем и знаменателем (пример 27). Чтобы вынести множитель за пределы дроби, нужно дробь заключить в скобки (примеры 28, 29 и 30)	26	'a/b'	$\frac{a}{b}$
		27	'(x+1)/(x-1)*(x+2)/(x-2)/(x//y)*4'	$\frac{4(x+1)(x+2)}{(x-1)(x-2)x/y}$
		28	'(1/2)*x'	$\frac{1}{2}x$
		29	'(3/4)*((x+1)/(x-1))'	$\frac{3}{4}\frac{x+1}{x-1}$
		30	'(3/4)*(x+1)/(x-1)'	$\frac{\frac{3}{4}(x+1)}{x-1}$
//	Деление. Используется косая черта. В некоторых случаях неправильно убирает скобки (пример 32). В этом случае рекомендуется пользоваться скобками !() (пример 33)	31	'a//b'	a/b
		32	'x//(2*y)'	$x/2y$
		33	'x//!(2*y)'	$x/(2y)$
/+	Деление.	34	'a/+b'	$a \div b$
:	Деление.	35	'(1/2):(1/3):(1/4)=12/2=6'	$\frac{1}{2}:\frac{1}{3}:\frac{1}{4}=\frac{12}{2}=6$
\	Разность множеств.	36	'A\B'	$A \setminus B$
+-	Плюс-минус.	37	'a+-b'	$a \pm b$
-+	Минус-плюс.	38	'a-+b'	$a \mp b$
=	Равно.	39	'a=b'	$a = b$
==	Тождественно.	40	'a==b'	$a \equiv b$
=~	Равно или порядка.	41	'a=~b'	$a \cong b$
~	Порядка.	42	'a~b'	$a \sim b$
~~	Примерно равно.	43	'a~~b'	$a \approx b$
<>	Не равно.	44	'a<>b'	$a \neq b$
>	Больше.	45	'a>b'	$a > b$
>>	Много больше.	46	'a>>b'	$a \gg b$
>~	Больше или порядка.	47	'a>~b'	$a \gtrsim b$
>=	Больше или равно.	48	'a>=b'	$a \geq b$
<	Меньше.	49	'a<b'	$a < b$
<<	Много меньше.	50	'a<<b'	$a \ll b$
<~	Меньше или порядка.	51	'a<~b'	$a \lesssim b$
<=	Меньше или равно.	52	'a<=b'	$a \leq b$
->	Стремится к.	53	'a->b'	$a \to b$

2.2.2. Символы, стоящие перед операндом

Перед операндом может стоять символ "_", обозначающий вектор, и/или символы "+", "-", "+-", "-+", обозначающие соответствующие унарные операции. Для числовых констант может использоваться символ "#".

Символ	Описание	№ примера	Строка	Результат
"_"	Вектор. В некоторых случаях возможна неправильная расстановка скобок, тогда рекомендуется использовать функцию Vect.	54	'_a'	$\vec{a}$
"+"	Унарный плюс.	55	'+a'	$+a$
"-"	Унарный минус.	56	'-a'	$-a$
"+-"	Унарный плюс-минус.	57	'+-a'	$\pm a$
"-+"	Унарный минус-плюс.	58	'-+a'	$\mp a$
"#"	Ставится перед числовыми константами для указания, что они должны быть выведены в научном формате.	59	'0.03*x'	$0{,}03x$
		60	'#0.03*x'	$3 \cdot 10^{-2}x$

2.2.3. Символы, стоящие после операнда

Если `TExprBuilder.PostSymbols:=True` (по умолчанию), после операнда могут стоять символы "_", "^", "!" или "`".

Символ	Описание	№ примера	Строка	Результат
"_" —	Нижний индекс. В некоторых случаях возможна неправильная расстановка скобок или многоступенчатых индексов (пример 62). В таких случаях вместо символа нижнего подчеркивания рекомендуется использовать функцию `Ind` (пример 63)	61	`'a_b'`	a_b
		62	`'a_x_0'`	a_{x_0}
		63	`'Ind(a,Ind(x,0))'`	a_{x_0}
"^"	Верхний индекс или возведение в степень. В некоторых случаях возможна неправильная расстановка скобок или многоступенчатых индексов. В таких случаях вместо символа "^" рекомендуется использовать функцию `Pow`. Если используется вместе с нижним индексом (символ "_" или функция `Ind`), то сначала указывается нижний индекс, а затем - верхний (пример 65), в противном случае индексы будут отображаться некорректно (пример 66).	64	`'a^b'`	a^b
		65	`'Ind(x,a)^2, x_a^2'`	$x_a^2,\ x_a^2$
		66	`'Ind(x^2,a), x^2_a'`	$\left(x^2\right)_a,\ x^{2_a}$
"!"	Факториал.	67	`'C_n^k=n!/k!/(n-k)!'`	$C_n^k = \frac{n!}{k!(n-k)!}$
"`"	Производная. Допустимо использование нескольких знаков подряд для обозначения производных высших степеней. В некоторых случаях возможна неправильная расстановка скобок. В таких случаях вместо символа обратного апострофа рекомендуется использовать функцию `Strokes`.	68	`'f(x)`'`	$f'(x)$
		69	`'f(x)```'`	$f'''(x)$

2.2.4. Конкатенация выражений

Несколько выражений могут склеиваться в одно при помощи символа "&" (не рисуется в изображении формулы) или разделителей "," и ";" (рисуются в изображении формулы).

Символ	Описание	№ примера	Строка	Результат
"&"	Конкатенация двух выражений. Символ должен либо не отделяться от обоих выражений пробелами (пример 69), либо отделяться от каждого из них одним пробелом (пример 70)	70	`'x&y'`	xy
		71	`'x & y'`	xy
"," ";"	Символ, отделяющий несколько подряд идущих выражений. После символа может быть произвольное число пробелов, однако это не влияет на расстояние между выражениями, которое составляет семь единиц ширины. Перед символом пробелов быть не должно.	72	`'a0,a1, a2, a3'`	$a_0,\ a_1,\ a_2,\ a_3$
		73	`'a0;a1; a2; a3'`	$a_0;\ a_1;\ a_2;\ a_3$

2.2.5. Скобки

Символы	Описание	№ примера	Строка	Результат
( )	Круглые скобки, служат для изменения порядка выполнения действий. Могут быть убраны построителем формулы, если в это не приведёт к искажению смысла выражения. Для принудительной установки скобок используйте скобки !().	74	`'(x+1)*(y-2)'`	$(x+1)(y-2)$
		75	`'(x+1)/(x-1)'`	$\frac{x+1}{x-1}$
		76	`'a+(b+c)=d*.(e*.f)'`	$a+(b+c)=d\cdot e\cdot f$
		77	`'y=(1+1/(1+1/x))'`	$y=\left(1+\frac{1}{1+\frac{1}{x}}\right)$
! ( )	Используются там же, где и обычные круглые скобки, но никогда не убираются построителем формулы.	78	`'!(x+1)*!(y-2)'`	$(x+1)(y-2)$
		79	`'!(x+1)/!(x-1)'`	$\frac{(x+1)}{(x-1)}$
		80	`'a+!(b+c)=d*.!(e*.f)'`	$a+(b+c)=d\cdot(e\cdot f)$

		81	'y=!(1+1/!(1+1/x))'	$y=\left(1+\frac{1}{\left(1+\frac{1}{x}\right)}\right)$
[ ]	Квадратные скобки. Никогда не убираются построителем.	82	'[x+1]*[y-2]'	$[x+1][y-2]$
		83	'[x+1]/[x-1]'	$\frac{[x+1]}{[x-1]}$
		84	'a+[b+c]=d*.[e*.f]'	$a+[b+c]=d\cdot[e\cdot f]$
		85	'y=[1+1/[1+1/x]]'	$y=\left[1+\frac{1}{\left[1+\frac{1}{x}\right]}\right]$
{ }	Фигурные скобки. Никогда не убираются построителем.	86	'{x+1}*{y-2}'	$\{x+1\}\{y-2\}$
		87	'{x+1}/{x-1}'	$\frac{\{x+1\}}{\{x-1\}}$
		88	'a+{b+c}=d*.{e*.f}'	$a+\{b+c\}=d\cdot\{e\cdot f\}$
			'y={1+1/{1+1/x}}	$y=\left\{1+\frac{1}{\left\{1+\frac{1}{x}\right\}}\right\}$
| |	Прямые скобки. Никогда не убираются построителем.	89	'|x+1|*|y-2|'	$\lvert x+1\rVert y-2\rvert$
		90	'|x+1|/|x-1|'	$\frac{\lvert x+1\rvert}{\lvert x-1\rvert}$
		91	'a+|b+c|=d*.|e*.f|'	$a+\lvert b+c\rvert=d\cdot\lvert e\cdot f\rvert$
		92	'y=|1+1/|1+1/x||'	$y=\left\lvert 1+\frac{1}{\left\lvert 1+\frac{1}{x}\right\rvert}\right\rvert$

Больше возможностей вывода скобок предоставляет функция `Brackets`.

2.2.6. Токены символов греческого алфавита

Токен	Символ	Токен	Символ	Токен	Символ	Токен	Символ
Alpha	Α	alpha	α	Nu	Ν	nu	ν
Beta	Β	beta	β	Xi	Ξ	xi	ξ
Gamma	Γ	gamma	γ	Omicron	Ο	omicron	ο
Delta	Δ	delta	δ	Pi	Π	pi	π
Epsilon	Ε	epsilon	ε	Rho	Ρ	rho	ρ
Zeta	Ζ	zeta	ζ	Sigma	Σ	sigma	σ
Eta	Η	eta	η			sigmao	ς
Theta	Θ	theta	θ	Tau	Τ	tau	τ
		thetao	ϑ	Upsilon	Υ	upsilon	υ
Iota	Ι	iota	ι	Phi	Φ	phi	φ
Kappa	Κ	kappa	κ	Chi	Χ	chi	χ
Lambda	Λ	lambda	λ	Psi	Ψ	psi	ψ
Mu	Μ	mu	μ	Omega	Ω	omega	ω

2.2.7. Токены-стрелки

Токен	Символ	Токен	Символ	Токен	Символ
ArrowL	←	DArrowL	⇐	BArrowL	↤
ArrowR	→	DArrowR	⇒	BArrowR	↦
ArrowU	↑	DArrowU	⇑	BArrowU	↥
ArrowD	↓	DArrowD	⇓	BArrowD	↧
ArrowLR	↔	DArrowLR	⇔	TArrowRL	⇄
ArrowUD	↕	DArrowUD	⇕	TArrowLR	⇆
ArrowLU	↖	DArrowLU	⇖	TArrowUD	⇅
ArrowRU	↗	DArrowRU	⇗	TArrowDU	⇵
ArrowRD	↘	DArrowRD	⇘	TArrowL	⇇
ArrowLD	↙	DArrowLD	⇙	TArrowR	⇉
ArrowLN	↚	DArrowLN	⇍	TArrowU	⇈
ArrowRN	↛	DArrowRN	⇏	TwoArrowD	⇊
ArrowLRN	↮	DArrowLRN	⇎	TRArrow	⇶
HArrowLR	⇋	HArrowRL	⇌		

2.2.8. Прочие токены

Токен	Значение	Описание
...	…	Многоточие, располагающееся на уровне знаков пунктуации
And	⋀	Логическое "И"
Angle	∠	Знак угла
Arc	⌒	Знак дуги
Asterisk	∗	Астериск - символ "*", несколько опущенный вниз по сравнению с обычным положением. Предназначен для использования в верхних индексах. Может обозначать, например, сопряженную величину
AsteriskC	⊛	Астериск в круге
Begin	◀	Начало доказательства (расчетов)
Belongs	∈	Принадлежит
BelongsN	∉	Не принадлежит
Circle	○	Круг
CircleC	⊙	Круг в круге
Colon	:	Двоеточие. В большинстве случаев может быть заменен функцией Colon
Comma	,	Запятая. В отличие от символа "," не вставляет пробелы после запятой. В большинстве случаев может быть заменен символом "," или функцией Comma.
Complex	ℂ	Множество комплексных чисел
Const	const	Слово "const", обозначающее произвольную константу. В отличие от обычных идентификаторов пишется прямым шрифтом, а не курсивом
CrossC	⊗	Крест в круге (прямое произведение)
Degree	°	Градус
Divide	|	Делит
DivideN	∤	Не делит
Dot	.	Точка
DotC	⊙	Точка в круге
DotEqual	≐	Приближается к пределу

DotEqualC	≑	Геометрически равный
DotEqualLR	≒	Образ
DotEqualRL	≓	Образ
DotsD	⋱	Диагональное многоточие. Может использоваться, например, для обозначения пропущенных элементов матрицы
DotsH	⋯	Горизонтальное многоточие. Может использоваться, например, для обозначения пропущенных элементов матрицы
DotsU	⋰	Диагональное многоточие. Может использоваться, например, для обозначения пропущенных элементов матрицы
DotsV	⋮	Вертикальное многоточие. Может использоваться, например, для обозначения пропущенных элементов матрицы
Empty		Пустое выражение. В отличие от Nil, имеет только нулевую ширину, а высота равна высоте символов.
EmptySet	∅	Пустое множество
End	▶	Конец доказательства (расчетов). Что и требовалось доказать
Entire	ℤ	Множество целых чисел
EqualC	⊜	Равно в круге
Exists	∃	Существует
ExistsN	∄	Не существует
ForAll	∀	Для всех
Ident	≡	Тождественно. Аналог знака "=="
IdentN	≢	Не тождественно
Inf	∞	Бесконечность
Intersection	∩	Пересечение
Minus	−	Минус. Предназначен для использования преимущественно в индексах. Прочие знаки ("+", "=" и т. п.) можно отобразить с помощью функции String. Но выражение "String(-)" даст не минус, а дефис, который существенно короче
MinusC	⊖	Минус в круге
Nabla	∇	Оператор Гамильтона
Natural	ℕ	Множество натуральных чисел
Nil		Пустое выражение, имеющее нулевые размеры. Предназначен для использования в функциях там, где по синтаксису должно быть выражение, но в конкретном случае требуется, чтобы его не было
Not	¬	Логическое отрицание
Or	⋁	Логическое "ИЛИ"
Parallel	∥	Параллельно
ParallelN	∦	Не параллельно
Parallelogram	▱	Параллелограмм
Perpendicular	⊥	Перпендикулярно
PLambda	ƛ	Лямбда с чертой. Используется, например, в квантовой механике
Planck	ℏ	Постоянная Планка
PlusC	⊕	Плюс в круге (прямая сумма)
Projective	ℙ	Проективное пространство
Prop	∝	Пропорционально
Quadrate	□	Квадрат
Quaternion	ℍ	Множество кватернионов Гамильтона

`Rational`	$\mathbb{Q}$	Множество рациональных чисел
`Real`	$\mathbb{R}$	Множество действительных чисел
`Rectangle`	$\square$	Прямоугольник
`Rhomb`	$\diamond$	Ромб
`Semicolon`	;	Точка с запятой. В отличие от символа "`;`" не вставляет пробелы после точки с запятой. В большинстве случаев может быть заменен символом "`;`" или функцией `Semicolon`
`Similar`	$\sim$	Подобно
`SlashC`	$\oslash$	Косая черта в круге
`SubSet`	$\subset$	Является подмножеством
`SubSetN`	$\not\subset$	Не является подмножеством
`SuperSet`	$\supset$	Является надмножеством
`SuperSetN`	$\not\supset$	Не является надмножеством
`Triangle`	$\triangle$	Треугольник
`Union`	$\cup$	Объединение
`Xor`	$\veebar$	Логическое исключающее "ИЛИ"

2.2.9. Функции

Функцией считается текст, после которого в скобках стоит один или несколько аргументов. Существуют зарезервированные имена функций, которые описаны ниже.

Если имя функции не является зарезервированным словом, результат зависит от длины имени. Имена функций, состоящие из одного символа, выводятся курсивом, а их аргументы всегда заключаются в круглые скобки (пример 93). Более длинные имена функций выводятся прямым шрифтом, а их аргументы заключаются в скобки только при необходимости (пример 94). Можно принудительно заключить аргумент в скобки, используя вместо открывающей скобки комбинацию "`!(`" (пример 95).

При возведении функции в степень или присоединении к ней индекса знаки соответствующих операций должны стоять после аргумента (пример 96).

Цифры, стоящие в конце имени функции, интерпретируются как нижний индекс, если `TExprBuilder.FuncAutoIndex:=True` (по умолчанию; пример 97).

В качестве имён функций можно использовать токены, обозначающие греческие буквы, а также токены `Nabla` и `PLambda`. В этом случае цифры в конце также интерпретируются как нижний индекс (пример 98).

№ примера	Строка	Результат
93	`'f(x,y,z)'`	$f(x, y, z)$
94	`'cos(x), sin(pi/2+x), tg(1/x)'`	$\cos x,\ \sin\left(\frac{\pi}{2}+x\right),\ \mathrm{tg}\,\frac{1}{x}$
95	`'cos!(x)'`	$\cos(x)$
96	`'f(x)_n,cos(x)^2'`	$f_n(x),\ \cos^2 x$
97	`'f0(x)=g1(x)'`	$f_0(x)=g_1(x)$
98	`'gamma0(x)'`	$\gamma_0(x)$

2.2.10. Зарезервированные функции

Имена зарезервированных функций нечувствительны к регистру символов. При описании зарезервированных функций используются следующие условные обозначения:

`E,E1,E2` и т. д. – произвольные обязательные выражения (если требуется, чтобы обязательное выражение было пустым, нужно вместо него вставлять токен `Nil` (примеры 138, 167));

`[,E1]` и т.д. – произвольные необязательные выражения (могут не прописываться в зависимости от ситуации);

`m,n,n1,n2` и т. д. – целочисленные константы;

`R` - вещественная константа.

Единицы ширины, использующиеся в некоторых зарезервированных функциях для указания размера пробела, подобраны так, что одна единица ширины примерно равна толщине вертикальной линии в символе "+"

Функция	Описание	№ примера	Строка	Результат
`Abs(E)`	Модуль E. Аналог скобок "||", добавлена для совместимости с синтаксисом Паскаля	99	`'Abs(x^2)=Abs(x)^2'`	$\left\|x^2\right\| = \left\|x\right\|^2$
`At(E1[,E2[,E3]])`	Значение E1 при условии E2 (пример 100) или Значение E1 от E2 до E3 (пример 101)	100	`'At(DiffRF(f,x),x=0)=1'`	$\left.\frac{df}{dx}\right\|_{x=0} = 1$
		101	`'Int(x*Diff(x),a,b)= At(x^2/2,a,b)= (b^2-a^2)/2'`	$\int_a^b x\,dx = \left.\frac{x^2}{2}\right\|_a^b = \frac{b^2-a^2}{2}$
`Brackets(S1S2,E)`	Заключает E в различные скобки. В качестве S1 может стоять символ "(", "[", "{", "|" или "0", "1", "2", "3", "4", в качестве S2 - ")", "]", "}", "|" или "0", "1", "2", "3", "4", "0" - означает отсутствие скобки с данной стороны, "1" - угловые скобки "⟨" и "⟩" (для обозначения, например, скалярного произведения векторов), "2" – скобки "⌊" и "⌋", обозначающие округление до ближайшего целого в меньшую сторону ("пол" числа), "3" – скобки "⌈" и "⌉", обозначающие округление до ближайшего целого в большую сторону ("потолок" числа), "4" - двойные прямые скобки.	102	`'Brackets(([],0&comma&1)'`	$(0,1]$
		103	`'Brackets(11, _a&comma(5)&_b)'`	$\langle \vec{a}, \vec{b} \rangle$
		104	`'Brackets(22,3.4)=3<> Brackets(33,3.4)=4'`	$\lfloor 3{,}4 \rfloor = 3 \neq \lceil 3{,}4 \rceil = 4$
		105	`'Brackets(23,3.4)=3'`	$\lfloor 3{,}4 \rfloor = 3$
		106	`'Brackets(23,3.6)=4'`	$\lfloor 3{,}6 \rfloor = 4$
		107	`'Brackets(44,_x)'`	$\left\|\vec{x}\right\|$
`Cap(E)`	Знак "^" над E	108	`'Cap(x)'`	$\hat{x}$
`CaseAnd(E[,...])`	Выбор одного из возможных вариантов с условием "И". Выражения в скобках идут в виде пар вариант-условие	109	`'|x|=CaseAnd( -x,x<0,0,x=0,x,x>0)'`	$\left\|x\right\| = \begin{cases} -x & x<0 \\ 0 & x=0 \\ x & x>0 \end{cases}$
`CaseOr(E[,...])`	Выбор одного из возможных вариантов с условием "ИЛИ". Выражения в скобках идут в виде пар вариант-условие	110	`'f(x)=CaseOr( 1//x,x<>0,0,x=0)'`	$f(x) = \begin{cases} 1/x & x \neq 0 \\ 0 & x = 0 \end{cases}$
`Circ(E1[,E2[,E3]])`	Интеграл по замкнутому контуру выражения E1. Под знаком интегрирования ставится E2, над ним - E3	111	`'Circ(F*Diff(L),L)'`	$\oint_L F\,dL$
`Colon(n)`	Вставляет в выражение двоеточие, а после него - пробел шириной в n единиц ширины	112	`'x & colon(15) & y'`	$x:\quad y$
`Comma(n)`	Вставляет в выражение запятую, а после неё - пробел шириной в n единиц ширины	113	`'x & comma(15) & y'`	$x,\quad y$
`Corr(E[,...])`	Выстраивает выражения в таблицу соответствия из двух столбцов, разделенных вертикальной чертой. Выражения в скобках идут в виде пар соответствующих значений	113A	`|Corr(u=x, u`=1, v`=sin(x), v=-cos(x))|`	$\begin{vmatrix} u = x & u' = 1 \\ v' = \sin x & v = -\cos x \end{vmatrix}$
`Diff(E1[,E2])`	Дифференциал выражения E1, возведенного в степень E2	114	`'Diff(x)'`	dx
		115	`'Diff(x,n)'`	dx^n
`DiffN(E1[,E2])`	Дифференциал степени E2 выражения E1	116	`'DiffN(x)'`	dx
		117	`'DiffN(x,n)'`	$d^n x$

`DiffR(E1[,E2])`	Полная производная степени E2 по E1	118	`'DiffR(x)'`	$\frac{d}{dx}$
		119	`'DiffR(x,n)'`	$\frac{d^n}{dx^n}$
		120	`'DiffR(x,2)*f(x)= DiffR(x)*DiffR(x)*f(x)'`	$\frac{d^2}{dx^2}f(x)=\frac{d}{dx}\frac{d}{dx}f(x)$
`DiffRF(E1,E2[,E3])`	Полная производная степени E3 выражения E1 по E2	121	`'DiffRF(f,x)'`	$\frac{df}{dx}$
		122	`'DiffRF(f(x),x,n)'`	$\frac{d^n f(x)}{dx^n}$
`Dot(n)`	Вставляет в выражение точку, а после неё - пробел шириной в n единиц ширины	123	`'x & dot(15) & y'`	$x\,.\;\;y$
`Fact(E)`	Факториал E	124	`'Fact(n)'`	$n!$
		125	`'Fact(k+1)'`	$(k+1)!$
`Func(E1,E2)`	Функция, "именем" которой является E1, а аргументом - E2	126	`'Func(PDiffRF(f,x,3),x)'`	$\frac{\partial^3 f}{\partial x^3}(x)$
`FuncSub("Name", E1,E2)`	Функция с именем "Name" выражения E2 при условии E1	127	`'FuncSub("Res",z=z0,f(z))'`	$\operatorname{Res}_{z=z_0} f(z)$
		128	`'FuncSub("max", x&Belongs&[a,b],f(x))'`	$\max_{x\in[a,\,b]} f(x)$
`Ind(E1,E2)`	Добавление к E1 нижнего индекса в виде E2. В большинстве случаев может быть заменена символом "_". При использовании с функцией `Pow` должна применяться раньше `Pow` (пример 130)	129	`'Ind(a,n)'`	a_n
		130	`'Pow(Ind(x,n),2)'`	x_n^2
`Int(E1[,E2[,E3]])`	Интеграл выражения E1. Под знаком интеграла ставится E2, над ним - E3	131	`'F(x)=Int(f(x)*Diff(x))'`	$F(x)=\int f(x)\,dx$
		132	`'Phi=Int(_H*Diff(_S),S)'`	$\Phi=\int_S \vec{H}\,d\vec{S}$
		133	`'Int(Diff(x),0,1)=1'`	$\int_0^1 dx=1$
		134	`'M=Int(Diff(x)*Int(x*y* Diff(y),0,1-x),0,1)'`	$M=\int_0^1 dx\int_0^{1-x} xy\,dy$
`IntM(n, E1[,E2[,E3]])`	n-кратный интеграл выражения E1. Под знаком интеграла ставится E2, над ним - E3. Если n=0, рисуется интеграл с неизвестной кратностью (используется многоточие)	135	`'IntM(3,f(x,y,z)*Diff(x)* Diff(y)*Diff(z),V)'`	$\iiint_V f(x,y,z)\,dx\,dy\,dz$
		136	`'IntM(0,f(x1,...,x_n)* DiffN(x,n))'`	$\iint\!\dots\!\int f(x_1,\dots,x_n)\,d^n x$
`Lim(E1,E2)`	Предел выражения E2 при условии E1	137	`'Lim(x->0,f(x))'`	$\lim_{x\to 0} f(x)$
		138	`'Lim(Nil,f)'`	$\lim f$
		139	`'Lim(StandC( x->0,x>0),f(x))=1'`	$\lim_{\substack{x\to 0\\ x>0}} f(x)=1$
`Line(E)`	Горизонтальная черта над E	140	`'Line(x)'`	$\overline{x}$
`Log(E1,E2)`	Логарифм E2 по основанию E1	141	`'log(a,x+1)=ln(x+1)/ln(a)'`	$\log_a(x+1)=\frac{\ln(x+1)}{\ln a}$
`Matrix( n,m,E[,...])`	Матрица размером m на n. Выражения E и следующие за ним расставляются по ячейкам матрицы. Выражений может быть меньше, чем m*n - в этом случае последние ячейки остаются пустыми. Матрица не	142	`'Matrix(2,3,x,y,x-y, x+y,z,z+y)'`	$\begin{matrix} x & y \\ x-y & x+y \\ z & z+y \end{matrix}$
		143	`'!(Matrix(2,2,1,2,-3,4))'`	$\begin{pmatrix} 1 & 2 \\ -3 & 4 \end{pmatrix}$

	обрамляется скобками, скобки надо добавлять явно	144	`'det(A)=|Matrix(3,3,a_11, DotsH,a_(1&n),DotsV,DotsD, DotsV,a_m1,DotsH,a_mn)|'`	$\det A = \begin{vmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m_1} & \cdots & a_{mn} \end{vmatrix}$
		145	`'[_a,_b]=|Matrix(3,3,_e_x, _e_y,_e_z,x_a,y_a,z_a,x_b, y_b,z_b)|'`	$[\vec{a}, \vec{b}] = \begin{vmatrix} \vec{e}_x & \vec{e}_y & \vec{e}_z \\ x_a & y_a & z_a \\ x_b & y_b & z_b \end{vmatrix}$
`Num( R[,n1[,n2[,n3]]])`	Позволяет управлять форматом записи числа R. Если порядок числа меньше или равен -n3, используется научная запись с точностью n1, если больше - обычная запись с числом разрядов перед точкой n1 и общим n2. По умолчанию n1=4, n2=4, n3=2	146	`'Num(0.00123456)'`	$1{,}235 \cdot 10^{-3}$
		147	`'Num(0.00123456, 6)'`	$1{,}23456 \cdot 10^{-3}$
		148	`'Num(0.00123456, 4, 4, 3)'`	$0{,}0012$
		149	`'Num(0.00123456, 4, 6, 3)'`	$0{,}001235$
`PDiff(E1[,E2])`	"Частный дифференциал" выражения E1, возведенного в степень E2. С математической точки зрения подобный "дифференциал" не имеет смысла, но функция очень удобна для создания выражений типа примера 152	150	`'PDiff(x)'`	∂x
		151	`'PDiff(x,n)'`	∂x^n
		152	`'PDiffN(f(x,y),3)/PDiff(x)/ PDiff(y,2)'`	$\frac{\partial^3 f(x,y)}{\partial x\, \partial y^2}$
`PDiffN(E1[,E2])`	"Частный дифференциал" спепени E2 выражения E1. С математической точки зрения подобный "дифференциал" не имеет смысла, но функция очень удобна для создания выражений типа примера 152	153	`'PDiffN(x)'`	∂x
		154	`'PDiffN(x,n)'`	$\partial^n x$
`PDiffR(E1[,E2])`	Частная производная степени E2 по E1	155	`'PDiffR(x)'`	$\frac{\partial}{\partial x}$
		156	`'PDiffR(x,n)'`	$\frac{\partial^n}{\partial x^n}$
		157	`'Nabla=PDiffR(x)*_e_x+ PDiffR(y)*_e_y+ PDiffR(z)*_e_z'`	$\nabla = \frac{\partial}{\partial x}\vec{e}_x + \frac{\partial}{\partial y}\vec{e}_y + \frac{\partial}{\partial z}\vec{e}_z$
`PDiffRF( E1,E2[,E3])`	Частная производная степени E3 выражения E1 по E2	158	`'PDiffRF(f,x)'`	$\frac{\partial f}{\partial x}$
		159	`'PDiffRF(f(x,y),x,n)'`	$\frac{\partial^n f(x,y)}{\partial x^n}$
`Points(E[,n])`	Точки над E, обычно означающие производную по времени	160	`'Points(y,2)=y*Points(x)'`	$\ddot{y} = y\dot{x}$
`Pow(E1,E2)`	Возведение E1 в степень E2. При использовании с функцией Ind должна применяться после Ind (пример 162). В большинстве случаев может быть заменена символом "^"	161	`'Pow(x+2,2//3)'`	$(x+2)^{2/3}$
		162	`'Pow(Ind(x,a),3)'`	x_a^3
`Prod(E1[,E2[,E3]])`	Произведение выражений E1. Под знаком произведения ставится E2, над ним - E3	163	`'Prod(a_i)'`	$\prod a_i$
		164	`'Prod(a_i,i<>j)'`	$\prod_{i \neq j} a_i$
		165	`'Prod(a_i,i=0,n)'`	$\prod_{i=0}^{n} a_i$
`Root(E1,E2)`	Извлечение корня степени E1 из выражения E2	166	`'Root(3,x-1)'`	$\sqrt[3]{x-1}$
		167	`'Root(nil, x-1)'`	$\sqrt{x-1}$

`Semicolon(n)`	Вставляет в выражение точку с запятой, а после неё - пробел шириной в n единиц ширины	168	`'x & semicolon(15) & y'`	$x;\ \ y$
`Space(n)`	Пробел размером в n единиц толщины. Используется для разделения выражений	169	`'y=x & space(7) & z=q'`	$y = x\ \ z = q$
`Sqr(E)`	Возведение выражения E в квадрат. Не имеет никаких преимуществ по сравнению с использованием символа "^" или функции Pow. Добавлена для совместимости с синтаксисом Паскаля	170	`'Sqr(a+b)= Sqr(a)+2*a*b+Sqr(b)'`	$(a+b)^2 = a^2 + 2ab + b^2$
`Sqrt(E)`	Извлечение квадратного корня из E	171	`'Sqrt(x^2+y^2)'`	$\sqrt{x^2+y^2}$
`StandC(E[,...])`	Размещает несколько выражений одно под другим, выравнивая по центру	172	`'StandC(0<=i<n,i<>j)'`	$0 \le i < n$ $i \ne j$
`StandL(E[,...])`	Размещает несколько выражений одно под другим, выравнивая по левому краю	173	`'StandL(0<=i<n,i<>j)'`	$0 \le i < n$ $i \ne j$
`StandR(E[,...])`	Размещает несколько выражений одно под другим, выравнивая по правому краю	174	`'StandR(0<=i<n,i<>j)'`	$0 \le i < n$ $i \ne j$
`String(Текст)` `String("Текст")`	Текст, выводящийся прямым шрифтом без изменений. Если в тексте встречаются круглые скобки, он должен быть заключён в двойные кавычки	175	`'String(Произвольный текст)'`	Произвольный текст
		176	`'String("Текст (со скобками)")'`	Текст (со скобками)
`Strokes(E[,n])`	Добавляет к E штрихи, обычно обозначающие производную	177	`'Strokes(f(x))'`	$f'(x)$
		178	`'Strokes(y,3)'`	y'''
`Sum(E1[,E2[,E3]])`	Сумма выражений E1. Под знаком суммы ставится E2, над ним - E3	179	`'Sum(a_i)'`	$\sum a_i$
		180	`'Sum(a_i,i<>j)'`	$\sum_{i \ne j} a_i$
		181	`'Sum(a_i,i=0,n)'`	$\sum_{i=0}^{n} a_i$
`Surf(E1[,E2[,E3]])`	Интеграл по замкнутой поверхности выражения E1. Под знаком интегрирования ставится E2, над ним - E3	182	`'Surf(F*Diff(S),S)'`	$\oint\!\!\!\oint_S F\,dS$
`Symbol(n)`	Вставляет в выражение символ с десятичным кодом n в кодировке UTF-16BE прямым шрифтом	183	`'Symbol(198)=1'`	$Æ = 1$
`SymbolI(n)`	Вставляет в выражение символ с десятичным кодом n в кодировке UTF-16BE наклонным шрифтом	184	`'SymbolI(198)=1'`	$\mathit{Æ} = 1$
`SystemAnd(E[,...])`	Объединяет выражения в систему с фигурной скобкой (с условием "И")	185	`'SystemAnd(x+y=5,x*y=6)'`	$\begin{cases} x+y=5 \\ xy=6 \end{cases}$
		186	`'SystemAnd(x+y=5 & Semicolon,x*y=6 & Dot)'`	$\begin{cases} x+y=5; \\ xy=6. \end{cases}$
`SystemOr(E[,...])`	Объединяет выражения в систему с квадратной скобкой (с условием "ИЛИ")	187	`'x^2=4 & space(7) & DArrowR & space(7) & SystemOr(x=2 & comma, x=-2 & Dot)'`	$x^2 = 4 \ \Rightarrow\ \left[\begin{array}{l} x = 2, \\ x = -2. \end{array}\right.$
`Tilde(E)`	Знак тильды над E	188	`'Tilde(x)'`	$\tilde{x}$
`Vect(E)`	Стрелка (вектор) над E. В большинстве случаев может быть заменена на символ "_" перед E	189	`'Vect(a)'`	$\vec{a}$
`Volume( E1[,E2[,E3]])`	Интеграл по замкнутому объему выражения E1. Под знаком интегрирования ставится E2, над ним - E3	190	`'Volume(F*Diff(V),V)'`	$\oint\!\!\!\oint\!\!\!\oint_V F\,dV$

2.3. Примеры использования

Пример 1

строка

```
'Si!(x)=Int((sin(x)/x)*Diff(x),0,x)=pi/2-Int((sin(x)/x)*Diff(x),x,+Inf)=
x-(1/3!)*(x^3/3)+(1/5!)*(x^5/5)-+...'
```

результат

$$\mathrm{Si}\left(x\right)=\int\limits_{0}^{x}\frac{\sin x}{x}\,dx=\frac{\pi}{2}-\int\limits_{x}^{+\infty}\frac{\sin x}{x}\,dx=x-\frac{1}{3!}\frac{x^3}{3}+\frac{1}{5!}\frac{x^5}{5}\mp\ldots$$

Пример 2

строка

```
'FuncSub("Res",z=a,f(z))=lim(z->a,[(1/(m-1)!)*.DiffR(z,m-1)*[(z-a)^m*.f(z)]])'
```

результат

$$\operatorname*{Res}_{z=a}f\left(z\right)=\lim_{z\to a}\left[\frac{1}{\left(m-1\right)!}\cdot\frac{d^{\,m-1}}{dz^{\,m-1}}\left[\left(z-a\right)^{m}\cdot f\left(z\right)\right]\right]$$

Пример 3

строка

```
'P(H_i&Divide&E)=P(H_i&Intersection&E)/P(E)=
P(H_i)*P(E&Divide&H_i)/Sum(P(H_i)*P(E&Divide&H_i),i)'
```

результат

$$P\left(H_i\,|\,E\right)=\frac{P\left(H_i\cap E\right)}{P\left(E\right)}=\frac{P\left(H_i\right)P\left(E\,|\,H_i\right)}{\sum\limits_{i}P\left(H_i\right)P\left(E\,|\,H_i\right)}$$

Пример 4

строка

```
'{StandC(1,1&space(15)&2)}_S&Ident&{StandC(1,2&space(15)&1)}_S&Ident&
(G*E_thetao-F*G_u)/(2*(E*G-F^2))'
```

результат

$$\left\{\begin{matrix}1\\1&2\end{matrix}\right\}_S\equiv\left\{\begin{matrix}1\\2&1\end{matrix}\right\}_S\equiv\frac{GE_\theta-FG_u}{2\left(EG-F^2\right)}$$

Пример 5

строка

```
'SystemAnd(a11*x1+a12*x2+...+a_(1&n)*x_n=b1 & Comma, a21*x1+a22*x2+...+a_(2&n)*x_n=
b2 & Comma,DotsV,a_(n&1)*x1+a_(n&2)*x2+...+a_nn*x_n=b_n & Semicolon)&
String(     или     ) & Sum(a_(i*k)*b_i,k=1,n),!(i=1,2,...,n)'
```

результат

$$\begin{cases}a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1,\\a_{21}x_1+a_{22}x_2+\ldots+a_{2n}x_n=b_2,\\\vdots\\a_{n1}x_1+a_{n2}x_2+\ldots+a_{nn}x_n=b_n;\end{cases}\quad\text{или}\quad\sum_{k=1}^{n}a_{ik}b_i,\ \left(i=1,\,2,\,\ldots,\,n\right)$$

Пример 6

строка

```
'A*.X=B & String(,     где  )&A=
!(Matrix(4,4,a11,a12,DotsH,a_(1&n),a21,a22,DotsH,a_(2&n),DotsV,DotsV,DotsH,DotsV,a_(n&1),
a_(n&2),DotsH,a_nn))&comma(20)&X=
!(StandC(x1,x2,DotsV,x_n))&comma(20)&B=!(StandC(b1,b2,DotsV,b_n)) & Dot'
```

результат

$$A\cdot X=B,\quad\text{где}\ A=\begin{pmatrix}a_{11}&a_{12}&\cdots&a_{1n}\\a_{21}&a_{22}&\cdots&a_{2n}\\\vdots&\vdots&\cdots&\vdots\\a_{n1}&a_{n2}&\cdots&a_{nn}\end{pmatrix},\quad X=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix},\quad B=\begin{pmatrix}b_1\\b_2\\\vdots\\b_n\end{pmatrix}.$$

Пример 7

строка

```
'Int(Int(x*y*Diff(x)*Diff(y),x^2,sqrt(x)),0,1)=
Int(At([(1/2)*x*y^2],x^2,sqrt(x))**Diff(x),0,1)=
(1/2)*Int(x*(x-x^4)*Diff(x),0,1)=(1/2)*At(x^3/3-x^6/6,0,1)=1/12'
```

результат

$$\int\limits_{0}^{1}\int\limits_{x^2}^{\sqrt{x}}xy\,dx\,dy=\int\limits_{0}^{1}\left[\frac{1}{2}xy^2\right]\Bigg|_{x^2}^{\sqrt{x}}dx=\frac{1}{2}\int\limits_{0}^{1}x\left(x-x^4\right)dx=\frac{1}{2}\left(\frac{x^3}{3}-\frac{x^6}{6}\right)\Bigg|_{0}^{1}=\frac{1}{12}$$