УДК 004.422.83

Встраиваемый рекурсивный интерпретатор символьных математических выражений с расширенным функционалом

Аносов Ю.В.[1*], Курдюмов Н.Н.[2]**

[1]Московский государственный областной гуманитарный институт, ул. Зелёная, 22, Орехово-Зуево, Московская область, 142611, Россия

[2]Московский авиационный институт (национальный исследовательский университет), МАИ, Волоколамское шоссе, 4, Москва, А-80, ГСП-3, 125993, Россия

**e-mail: anosoff-yurij@yandex.ru*

***e-mail: nick.n.kurdyumov@gmail.com*

Аннотация

В статье рассматривается проблема разработки прикладных программных приложений, одной из функций которых является распознавание и интерпретация тех или иных математических выражений, заданных в символьной форме, а также представляется разработанный авторами статьи встраиваемый интерпретатор подобных математических выражений, основанный на информационных таблицах и сложноподчинённых рекурсивных вычислениях. Представляемый интерпретатор обладает небольшими размерами, поддерживает широкий набор математических функций (включая гиперболические), скобочную и функциональную запись любой степени вложенности. Реализованная в интерпретаторе многоуровневая система анализа символьных математических выражений позволяет отслеживать множество лексических, синтаксических, семантических и математических ошибок (включая ошибки периода исполнения).

Представляемый интерпретатор может быть использован при разработке различных программных приложений математического характера путём простого включения модуля интерпретатора в разрабатываемый проект директивами инструментальной среды разработки «Microsoft Visual Studio» и соответствующего оформления вызовов средствами языка «Си» [1-3].

Ключевые слова: интерпретатор, лексический анализатор, синтаксический анализатор, семантический анализатор, анализатор математических ошибок периода исполнения, промежуточная форма, программное приложение, программный модуль, рекурсия.

Введение

Одной из наиболее часто встречающихся проблем, возникающих при проектировании тех или иных прикладных программных приложений математического и инженерного характера является проблема разработки встроенных в подобного рода приложения специализированных интеллектуальных распознавателей и интерпретаторов математических выражений, заданных в символьной форме, обладающих свойством выявления и нейтрализации максимального числа лексических, синтаксических, семантических и математических ошибок (включая ошибки периода исполнения). Как правило, готовых решений данной задачи в стандартных инструментальных средах программирования по умолчанию не предлагается. В результате, в процессе реализации каждого из подобных приложений, приходится осуществлять разработку

очередного, локально-ориентированного интерпретатора, определяемого спецификой входного математического языка, соответствующего поставленным заказчиком задачам.

Тот факт, что подобные разработки обычно выполняются под конкретный заказ и под решение конкретных задач (зачастую с нарушениями стандартных правил записи математических выражений, изменённых в угоду устоявшейся у заказчика практике оформления), приводил к тому, что разработанное приложение, конечно, в максимальной степени обеспечивало выполнение требований данного конкретного заказчика, однако, именно это исключало возможность использования разработанных программных модулей в других проектах по причине их весьма узкой спецификации. Таким образом, для каждого нового проекта приходилось разрабатывать очередной «специфический» интерпретатор, ориентированный на задачи, поставленные заказчиком. Идея разработки прикладных программных приложений такого рода возникла при решении практических задач [4-8].

Постановка задачи

В соответствии с обозначенными во введении проблемами была поставлена задача о проектировании и разработке специализированного программного модуля (библиотеки исходных кодов), реализующего функции универсального расширенного интерпретатора сложных математических выражений, заданных в символьной форме.

Предполагаемый интерпретатор должен охватывать потребности интерпретации тех или иных математических выражений в приложении к максимальному числу возможных прикладных задач.

Основными характеристиками разрабатываемого интерпретатора должны являться:

- изначальная поддержка максимального количества математических функций (включая нестандартные и редко используемые);
- многоуровневая вложенность скобочной и функциональной записи;
- использование в выражениях не только констант, но также переменных и параметров;
- идентификация максимального числа лексических, синтаксических и семантических ошибок;
- идентификация наиболее вероятных математических ошибок;
- идентификация ошибок периода исполнения.

При этом, проектируемый интерпретатор должен обладать свойством простого встраивания в любое разрабатываемое программное приложение при помощи средств препроцессора инструментальной среды разработки «Microsoft Visual Studio» и соответствующего оформления вызовов интерпретатора средствами языка Си [1].

При необходимости (с учётом некоторой переработки исходного кода) было бы желательно обеспечить возможность встраивания данного интерпретатора и в проекты, разрабатываемые на уровне API в среде «Delphi» [2].

Дополнительным свойством разрабатываемого интерпретатора должна являться возможность относительно несложной модификации исходного программного кода, позволяющей легко и быстро расширить функционал интерпретатора на поддержку дополнительных математических функций [3].

Выбор технологии реализации

Из общей теории трансляторов известно, что интерпретаторы представляют собой специфическую разновидность трансляторов, которые в процессе получения, обработки и преобразования входного потока данных не генерируют выходной объектный код, а осуществляют прямое непосредственное выполнение (интерпретацию) распознанных во входном потоке инструкций и команд. Все существующие интерпретаторы могут быть разделены на две группы: интерпретаторы простого типа и интерпретаторы компилирующего типа. Отличие между интерпретаторами указанных типов состоит в том, что:

- *интерпретаторы первого (простого) типа* являются однопроходными и осуществляют немедленное исполнение каждой из распознанных команд (независимо от вычислимости и логической сущности вычисляемых значений);
- *интерпретаторы второго (компилирующего) типа* являются многопроходными, т.е. осуществляют более интеллектуальную обработку входного потока данных, с многоуровневым контролем ошибок различной степени и уровня влияния на каждом из проходов интерпретатора.

На этапе первого прохода интерпретаторы компилирующего типа осуществляют разбор входного потока на лексемы и выполняют предварительный лексический, синтаксический и семантический анализ данных входного потока (включая генерацию сообщений об обнаружении возможных лексических, синтаксических и семантических ошибок), генерируют ряд специализированных информационных таблиц (например, расширенных таблиц распознанных лексем входного потока) и формируют некоторую, подготовленную к выполнению промежуточную форму последовательности распознанных инструкций.

Если генерация информационных таблиц и промежуточной формы завершена успешно, то сгенерированная последовательность команд отправляется на второй этап обработки, собственно интерпретатор (вычислитель), на котором и осуществляется непосредственное выполнение инструкций.

Существует несколько *классических* видов записи промежуточных форм последовательности распознанных инструкций: польская инверсная запись, триадная запись, тетрадная запись и некоторые другие формы записи. На практике же, при разработке конкретных интерпретаторов, ориентированных на решение специфических прикладных задач, применяются те или иные модификации указанных видов записи промежуточных форм, а также вариант организации интерпретации, основанный исключительно на базе информационных таблиц без формирования промежуточных форм.

Каждый из рассмотренных вариантов имеет как свои преимущества, так и недостатки.

Решение о выборе технологии

В связи с тем, что в поставленной задаче требовалось реализовать интерпретацию сложных математических выражений, содержащих многоуровневую скобочную и функциональную запись, с дополнительным анализом входного потока данных не только на лексические, синтаксические и семантические ошибки, но и на выявление ряда математических ошибок (включая ошибки периода выполнения), было принято однозначное решение о том, что разрабатываемый интерпретатор должен быть интерпретатором компилирующего типа.

В целях уменьшения объёмов программного кода и упрощения алгоритмов обработки (в частности при обработке многократно вложенных скобочных и функциональных записей), на некотором этапе разработки было решено отказаться от формирования полных промежуточных форм потока инструкций (в классическом смысле) и осуществлять интерпретацию на основе последовательного исполнения инструкций, извлекаемых напрямую из расширенных информационных таблиц, сформированных усовершенствованным лексико-семантическим анализатором входного потока. Вычисление же многоуровневых скобочных и функциональных паттернов было решено организовать за счёт сложноподчинённых рекурсивных вызовов конечного «вычислителя».

Лексико-семантический анализатор

К функциям лексико-семантического анализатора были отнесены такие стандартные и дополнительные действия, как:

- последовательное вычленение лексем из входного потока с автоматической нейтрализацией лишних пробелов;
- присвоение выявленным лексемам специальных кодов с занесением их в специализированную последовательную информационную таблицу лексем;
- формирование информационной таблицы констант;
- выявление максимального числа лексических, синтаксических и семантических ошибок в записи распознаваемых выражений;
- формирование сообщений об обнаруживаемых ошибках;
- в целях простой локализации обнаруживаемых ошибок и обеспечения (для вызывающего приложения) возможности их наглядной визуализации с позиционированием месторасположения во входном потоке – расположение команд и констант в информационных таблицах должно соответствовать естественному расположению математических объектов в исходной записи интерпретируемого математического выражения.

В том случае, когда в процессе распознавания и лексико-семантического анализа входного потока данных возникают события, связанные с обнаружением тех или иных ошибок в записи распознаваемого выражения, дальнейшее распознавание и анализ должны быть прерваны, а вызывающему модулю выставлено соответствующее сообщение об обнаруженных во входном потоке ошибках.

В противном случае, т.е. в случае отсутствия лексических, синтаксических и семантических ошибок во входном потоке, сформированная последовательность кодов инструкций (в виде специализированной информационной таблицы в совокупности с таблицей констант) передаётся на непосредственное исполнение интерпретатору команд (конечному вычислителю).

Организация конечного вычислителя

В процессе разработки было опробовано несколько вариантов организации конечных вычислений последовательности распознанных команд. В результате, в итоговом варианте реализации интерпретатора была принята следующая схема организации вычислений, основанная на сложноподчинённых рекурсивных вызовах:

- на каждом шаге вычислений формируется короткий четырёхпозиционный паттерн команд, содержащий 4 математических объекта: операнд_1, операция_1, операнд_2, операция_2 (т.е. вариант с преданализом последующей операции);
- операциями являются унарные и бинарные математические операции;
- в качестве операндов рассматриваются как числа и константы, явно указанные во входном потоке, так и скобочные записи, и математические функции;
- индексы записей информационной таблицы, соответствующие объектам текущего паттерна необязательно являются последовательными индексами и могут изменяться в результате вложенных рекурсивных

вызовов, т.е. для каждого из элементов текущего паттерна осуществляется независимая индексация;

- в случае, когда приоритет первой операции текущего паттерна равен или выше приоритета второй операции, производится вычисление этой операции, с последующим сдвигом паттерна на две позиции вправо и присвоение результата произведённого вычисления первому операнду нового паттерна;
- в случае, когда приоритет первой операции текущего паттерна ниже приоритета второй операции, осуществляется рекурсивный вызов интерпретатора от нового паттерна (смещённого на две позиции вправо относительно текущего паттерна, в котором первая операция является второй операцией текущего паттерна), с последующим возвращением к вычислению текущего паттерна, с заменой значения второго операнда текущего паттерна на вычисленное в рекурсивном вызове значение второй операции и изменением индекса второй операции текущего паттерна на индекс операции (по информационной таблице лексем), следующей за последним индексом, обработанным в рекурсивном вызове;
- скобочные фрагменты выражений рассматриваются как сложные операнды первого типа, требующие предварительного вычисления, и интерпретируются путём рекурсивного вызова интерпретатора от

внутрискобочной записи, с последующей подстановкой результата вычисления в качестве соответствующего операнда текущего паттерна;

- признаком завершения вложенных рекурсивных вызовов при вычислении скобочных записей является обнуление счётчика алгебраической суммы числа обработанных скобок;
- математические функции рассматриваются как сложные операнды второго типа, требующие двойного предварительного вычисления (рекурсивный вызов интерпретатора от скобочной записи аргументной части функций с последующим вычислением значения самой функции от полученного значения её аргументной части), с дальнейшей подстановкой вычисленного значения функции в качестве соответствующего операнда текущего паттерна команд.

Выявление и нейтрализация ошибок периода исполнения

На каждом этапе вычислений, для каждой из арифметических операций и математических функций, должен осуществляться автоматический анализ возможного возникновения математических ошибок (например, деление на ноль или выход за границы области определения функции). В случае обнаружения подобных ошибок:

- вычисление должно прерываться;
- вызывающему модулю должен возвращаться не результат вычисления, а стандартный результат ошибки периода исполнения;

- в специальной глобальной переменной должен выставляться код ошибки, возникшей в процессе вычисления.

Результаты разработки

В результате проделанной работы был спроектирован и реализован базовый вариант универсального компактного интерпретатора математических выражений, обеспечивающий достаточно простой механизм встраивания в прикладные проекты, разрабатываемые в инструментальной среде «Microsoft Visual Studio».

Далее представлены некоторые технические характеристики разработанного интерпретатора.

Поддерживаемые математические операции, функции и объекты

Все, поддерживаемые на данный момент времени математические объекты, константы, операции и функции, допустимые в записи распознаваемых и интерпретируемых математических выражений, представлены в таблице 1.

Таблица 1.

Поддерживаемые математические операции, функции и объекты.

Код лексемы	Код функции	Формат записи во входном языке	Описание
0	---	нет	Пусто. (Например, отсутствие операнда в случае завершения последовательности).
Числа и операции			
1	---	Число	Правильная запись числа (включая как простую, так и экспоненциальную формы)
2	---	+	Операция сложения
3	---	-	Операция вычитания
4	---	*	Операция умножения
5	---	/	Операция деления
6	---	^	Операция возведения в степень, включая вещественные значения показателя (отрицательный показатель должен указываться в

			скобках)
7	---	(	Начало скобочной записи
8	---	)	Окончание скобочной записи
Математические константы (рассматриваются как безаргументные функции)			
9	1	Pi	Константа «Пи»
10	2	e	Константа «e» («e» должно быть в латинской раскладке. В противном случае будет распознано как «неизвестная функция»)
Прямые и обратные тригонометрические функции			
11	3	sin	Синус
12	4	cos	Косинус
13	5	tg	Тангенс
14	6	ctg	Котангенс
15	7	sec	Секанс
16	8	cosec	Косеканс
17	9	asin	Арксинус
18	10	acos	Арккосинус
19	11	atg	Арктангенс
20	12	actg	Арккотангенс
21	13	asec	Арксеканс
22	14	acosec	Арккосеканс
Прямые и обратные гиперболические функции			
23	15	hsin	Гиперболический синус
24	16	hcos	Гиперболический косинус
25	17	htg	Гиперболический тангенс
26	18	hctg	Гиперболический котангенс
27	19	hsec	Гиперболический секанс
28	20	hcosec	Гиперболический косеканс
29	21	hasin	Гиперболический арксинус
30	22	hacos	Гиперболический арккосинус
31	23	hatg	Гиперболический арктангенс
32	24	hactg	Гиперболический арккотангенс
33	25	hasec	Гиперболический арксеканс
34	26	hacosec	Гиперболический арккосеканс
Логарифмические, степенные и дополнительные функции			
35	27	ln	Натуральный логарифм
36	28	lg	Десятичный логарифм
37	29	---	Резервный индекс
38	30	abs	Модуль (абсолютное значение)
39	31	sqr	Квадрат
40	32	sqrt	Квадратный корень
41	33	cub	Куб
42	34	cubrt	Кубический корень
43-96	---	---	резерв на 52 дополнительные функции
Переменные и параметры			
97	---	X	Переменная «X»
98	---	Y	Переменная «Y»
99	---	Z или a	Переменная «Z» или параметр «a». Одновременное использование переменной «Z» и параметра «a» не допускается, так как они будут идентифицированы как один и тот же объект.

Обнаруживаемые лексические, синтаксические и семантические ошибки

Текущая версия компилятора обеспечивает выявление тридцати восьми различных лексических, синтаксических и семантических ошибок в записи математических выражений во входном потоке. Все, обнаруживаемые в данной версии интерпретатора ошибки и соответствующие им двустрочные сообщения, перечислены в таблице 2. Вторая строка в некоторых сообщениях может отсутствовать.

Таблица 2.

Синтаксические и семантические ошибки, обнаруживаемые лексико-семантическим анализатором интерпретатора.

№	Генерируемое двустрочное сообщение	
	Первая строка сообщения	**Вторая строка сообщения (может отсутствовать)**
0	--- Пустая строка, т.е. НЕТ ОШИБОК	---
1	Ошибка в записи числа	Число может содержать не более 14 цифр
2		Число не может содержать более 1 точки
3		Ошибка в записи знака порядка
4		Порядок числа может содержать не более 3-х цифр
5		Порядок числа не может содержать точку
6		Неопределяемая ошибка в записи порядка
7		В записи порядка отсутствуют цифры
8		Порядок числа должен лежать в пределах: от e-307 до e+307
9		Значение выходит за пределы 1.7e±307
10	Неизвестная функция	---
11	Выражение не может начинаться с оператора '*' '/' '^'	
12	Выражение не может начинаться с правой скобки	
13	Оператор '*' '/' '^'	не может находиться сразу после '('
14	Нельзя ставить подряд несколько операторов	---
15	Пустые скобки '()'	

16	Пропущен оператор между скобками ') ('	
17	Для данной правой скобки ')'	нет соответствующей левой скобки '('
18	После имени функции должна стоять левая скобка '('	---
19	Отсутствует связь между числом и функцией	Возможно пропущен оператор +-*/^
20	Отсутствует связь между правой скобкой и функцией	
21	Отсутствует связь между константой и функцией	
22	Отсутствует связь между двумя числами	
23	Отсутствует связь между правой скобкой и числом	
24	Отсутствует связь между константой и числом	
25	Отсутствует связь между числом и константой	
26	Отсутствует связь между правой скобкой и константой	
27	Отсутствует связь между константой и константой	
28	Выражение не должно заканчиваться оператором	---
29	Выражение не должно заканчиваться именем функции	
30	Выражение не должно заканчиваться левой скобкой	
31	Левых скобок больше, чем правых	
32	Отсутствует связь между переменной или параметром и функцией	Возможно пропущен оператор +-*/^
33	Отсутствует связь между переменной или параметром и числом	
34	Отсутствует связь между переменной или параметром и константой	
35	Отсутствует связь между числом и переменной или параметром	
36	Отсутствует связь между правой скобкой и переменной или параметром", "Возможно пропущен оператор +-*/^)"	
37	Отсутствует связь между константой и переменной или параметром	
38	Две переменных или параметра подряд	

Обнаруживаемые математические ошибки

Разработанный интерпретатор, помимо множества лексических, синтаксических и семантических ошибок, выявляемых на этапе предварительного лексико-семантического анализа входного потока данных, позволяет также выявлять ряд математических ошибок, обнаружение которых возможно только на этапе непосредственных вычислений. К списку подобных ошибок относятся такие ошибки как: деление на ноль, выход за пределы области определения вычисляемых функций, и т.д.

Полный список математически-обусловленных ошибок периода исполнения, выявляемых и нейтрализуемых данной версией интерпретатора, перечислены в таблице 3.

Таблица 3.

Математические ошибки, обнаруживаемые интерпретатором.

№ ошибки	Генерируемое двустрочное сообщение
0	Нет ошибок.
1	Деление на ноль.
2	Невычислимое выражение. tg(Pi/2 + n*Pi) не существует.
3	Невычислимое выражение. ctg(0 + n*Pi) не существует.
4	Невычислимое выражение. sec(Pi/2 + n*Pi) не существует.
5	Невычислимое выражение. cosec(0 + n*Pi) не существует.
6	Невычислимое выражение. arcsin(X) при |X|>1 не существует.
7	Невычислимое выражение. arccos(X) при |X|>1 не существует.
8	Невычислимое выражение. arcsec(X) при |X|<1 не существует.
9	Невычислимое выражение. arccosec(X) при |X|<1 не существует.
10	Невычислимое выражение. hctg(0) не существует.
11	Невычислимое выражение. hcosec(0) не существует.
12	Невычислимое выражение. ahcos(X) при X<1 не существует.
13	Невычислимое выражение. ahtg(X) при |X|>=1 не существует.
14	Невычислимое выражение. ahctg(X) при |X|<=1 не существует.
15	Невычислимое выражение. ahsec(X) при (X<=0 и X>1) не существует.
16	Невычислимое выражение. ahcosec(X) при X=0 не существует.
17	Невычислимое выражение. ln(X) при X<=0 не существует.
18	Невычислимое выражение. lg(X) при X<=0 не существует.
19	Невычислимое выражение. sqrt(X) при X<0 не существует.
20	Операция возведения в степень X^a при X=0 и a<0 не определена.

Программный интерфейс

Разработанный анализатор-интерпретатор состоит из двух программных модулей:

- **Math_Lex_Syntax_Analiz.cpp** – модуль, реализующий функции лексико-семантического анализатора;
- **Math_Interpretator.cpp** – модуль, реализующий функции конечного вычислителя с контролем математических ошибок периода исполнения.

Для включения указанных модулей в любое, разрабатываемое на уровне Win-API приложение, в среде «Microsoft Visual Studio», необходимо выполнить следующие действия:

- включить в проект модуль интерпретатора директивой компилятора: **#include "Math_Interpretator.cpp"** (модуль должен располагаться в соответствующем проекту каталоге);
- подключение лексико-семантического анализатора будет выполнено автоматически за счёт прописанной в модуле интерпретатора директивы: **#include "Math_Lex_Syntax_Analiz.cpp"** (при условии, что данный модуль также расположен в соответствующем каталоге);

Для размещения текстовой строки, содержащей математическое выражение в символьной форме, в модуле лексико-семантического анализатора предопределена строковая переменная длиной в 500 символов:

char DATA_TEXT[500];

Вызывающее приложение должно организовать ввод или автоматическое заполнение указанной строки, например, так:

DATA_TEXT ="z*ln(abs(z))^2*(cos(x+z)+abs(sin(x-z))) /10";

Вызов лексико-семантического анализатора выполняется командой:

ErrorCode = __LSA_Syntax_Analiz(DATA_TEXT, &ErrPos, N);

где:

- **DATA_TEXT** – исходный текст математического выражения;
- **ErrorCode** – код обнаруженной ошибки (если 0 – то ошибок нет)

- **&ErrPos** – позиция обнаруженной ошибки;

  (перечисленные переменные предопределены в модуле лексико-семантического анализатора)

- **N** – номер набора данных.

Номер набора данных (нумерация начинается с нуля) предназначен для обеспечения возможности предварительного анализа нескольких независимых входных потоков данных без перехода к вычислениям. Для каждого такого потока данных модуль лексико-семантического анализатора сформирует независимую последовательность команд и таблиц. На данный момент времени реализована поддержка двух потоков данных, так как для решения большинства задач двух потоков достаточно. При желании количество потоков может быть легко увеличено путем изменения верхней границы первого индекса в генерируемых таблицах в коде анализатора:

```
int    __LSA_List_Of_Code[2][N_LEX];
int    __LSA_N_Code[2];
double __LSA_List_Of_Const[2][N_LEX];
```

Выдача сообщений об ошибках, возникающих на стадии лексико-семантического анализа, в простейшем случае может быть организована в вызывающем приложении следующим образом:

```
char    S_Err[150];
ErrorCode = __LSA_Syntax_Analiz(DATA_TEXT, &ErrPos, 0);
if (ErrorCode != 0)
```

```
{ sprintf (S_Err, "%s\n%s", __LSA_LIST_ERRORS[ErrorCode][0],
__LSA_LIST_ERRORS[ErrorCode][1]);
  MessageBoxA(Hwnd, S_Err, "Ошибка в формуле", MB_OK | MB_ICONSTOP);
}
```

где:

- **S_Err** – строковая переменная, определяемая вызывающим приложением, предназначенная для формирования текста сообщения об ошибке;
- **Hwnd** – хендл окна вызывающего приложения, в котором организуется выдача сообщения.

В случае отсутствия лексических, синтаксических и семантических ошибок, выявляемых на этапе анализа входного потока и формирования последовательности команд, непосредственные вычисления могут быть инициированы при помощи следующей команды:

REZ =__EVA_Interpretator(X, Y, Za, N);

где:

- **REZ** – получаемое значение типа **«double»;**
- **X, Y, Za** – текущие значения соответствующих переменных или параметров (также типа **«double»**), если таковые присутствуют во входном математическом выражении; при их отсутствии в выражении данные поля игнорируются;

- **N** – номер потока команд (номер предварительно разобранного выражения), интерпретацию которого необходимо выполнить.

Ситуации возникновения ошибок периода исполнения могут быть отслежены путём контроля предопределённых в интерпретаторе переменных:

- **__EVA_ERR_NUM_CODE** – код ошибки;
- **__EVA_ERR_NUM_LEX** – номер лексемы, при обработке которой возникла ошибка.

Если процесс вычисления завершен удачно – то обе указанные переменные будут иметь нулевые значения.

В случаях же возникновения подобных ошибок, текстовое сообщение о них может быть получено из общей таблицы ошибок периода исполнения по выставленному коду:

```
char    S_Err[150];
sprintf (S_Err, "%s",  __EVA_RUNTIME_ERRORS [__EVA_ERR_NUM_CODE]);
```

Дополнительные замечания

Во-первых, хотелось бы отметить то, что разделение функций интерпретатора на два этапа: анализ входного потока (включая формирование последовательности команд) и непосредственный вычислитель (вызов которого может быть отложен), а также возможность использования в интерпретируемых выражениях переменных и параметров (реальные значения которых передаются вычислителю при его вызове) – обеспечивает возможность использования данного интерпретатора не только для

организации простых вычислений, но и для таких вычислительных процессов, в которых необходимо осуществить многократно повторяющиеся расчёты значений одних и тех же математических выражений от различных значений, входящих в эти выражения переменных и параметров. В качестве примера таких задач можно привести задачу табулирования значений функций, заданных аналитически, например, в целях дальнейшего построения и визуализации их графиков.

Также следует отметить, что ответственность за технологически верное включение представленного интерпретатора в проекты, а также адекватный перехват, обработку и отображение соответствующих сообщений об обнаруженных анализатором и вычислителем ошибках, лежит на разработчиках приложений, принявших решение об использовании данного интерпретатора.

Примеры использования

Подводя итоги проделанной работы хотелось бы продемонстрировать возможности использования, разработанного авторами данной статьи универсального встраиваемого интерпретатора математических выражений, на примере нескольких программных приложений. На следующих рисунках представлены скриншоты результатов подобного использования.

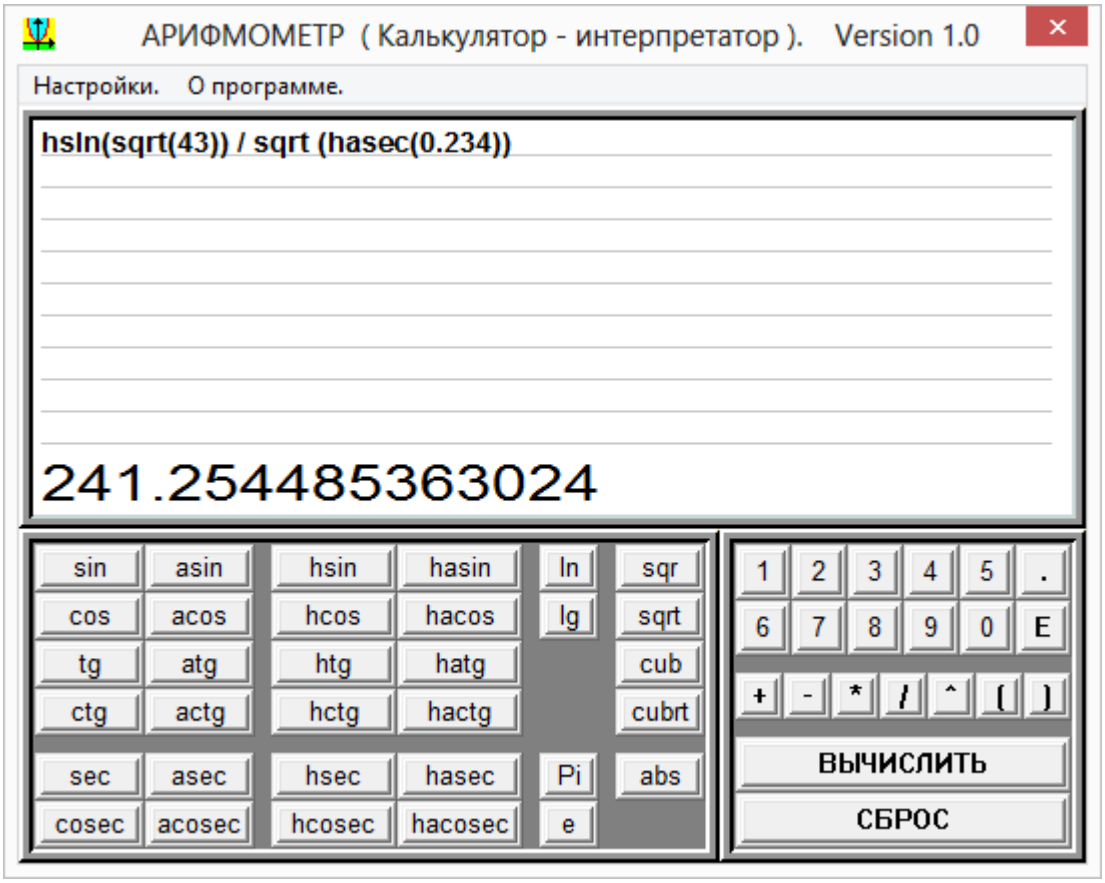

Рис. 1. Использование в приложении, реализующем функции интерпретирующего калькулятора.

Рис. 2. Использование в приложении, предназначенном для визуализации графических решений параметрических уравнений.

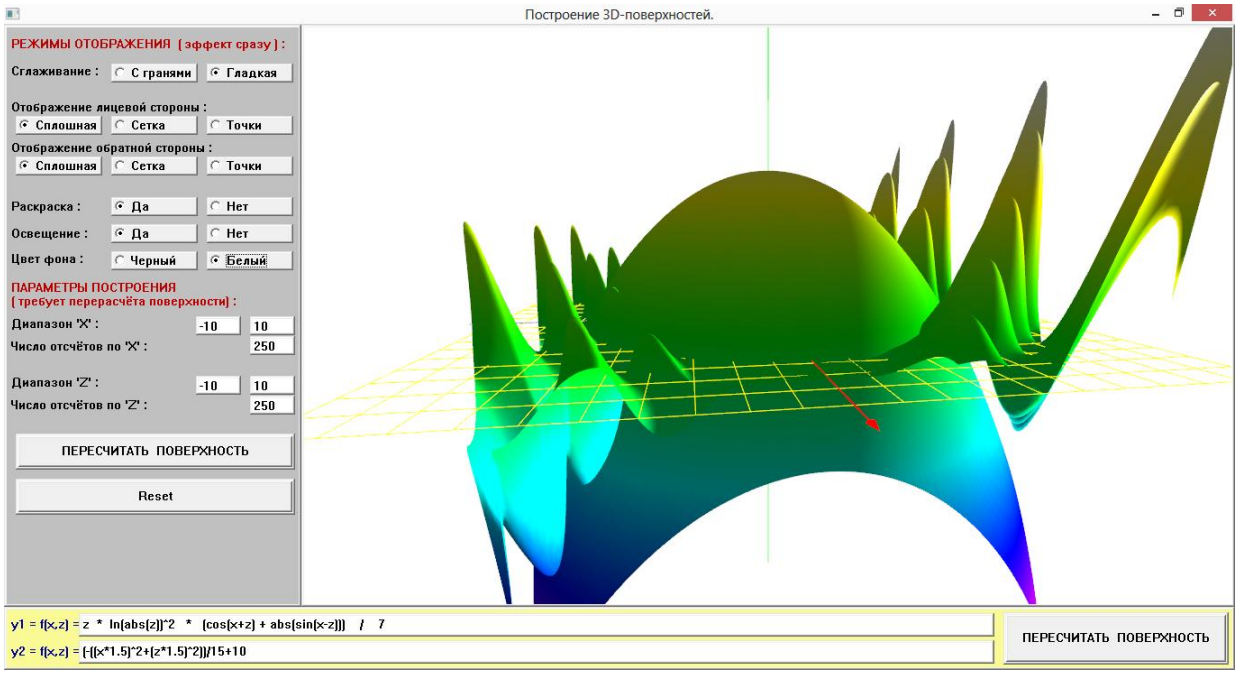

Рис. 3. Использование в приложении, предназначенном для визуализации 3D-поверхностей, заданных в аналитической форме.

Библиографический список

1. Альфред В. Ахо, Моника С. Лам, Рави Сети, Джеффри Д. Ульман. Компиляторы: принципы, технологии и инструментарий. – М.: Издательский дом "Вильямс", 2015. – 1184 с.
2. Карпов Ю.Г. Теория и технология программирования. Основы построения трансляторов. - СПБ.: БХВ, 2005. – 271 с.
3. Робин Хантер. Основные концепции компиляторов. - М.: Издательский дом "Вильямс", 2002. - 256 с.

4. Аносов Ю.В., Данилин А.Н., Курдюмов Н.Н. О жесткостях проволочных конструкций спирального типа // Труды МАИ, 2015, №80: http://www.mai.ru/science/trudy/published.php?ID=56958
5. Данилин А.Н., Козлов К.С., Кузнецова Е.Л., Тарасов С.С. Моделирование колебаний гасителя вибрации проводов воздушных систем энергоснабжения // Труды МАИ, 2013, №64: http://www.mai.ru/science/trudy/published.php?ID=36556
6. Крупенин А.М., Мартиросов М.И. Численное моделирование поведения трехслойной прямоугольной пластины при вертикальном ударе о жидкость // Труды МАИ, 2013, №69: http://www.mai.ru/science/trudy/published.php?ID=43066
7. Лурье С.А., Соляев Ю.О., Нгуен К., Медведский А.Л., Рабинский Л.Н. Исследование локальных эффектов в распределении температурных напряжений на контактных границах слоистых сред // Труды МАИ, 2013, №71: http://www.mai.ru/science/trudy/published.php?ID=47084
8. Формалев В.Ф., Кузнецова Е.Л., Селин И.А. Методика, алгоритм и программный комплекс решения задач о тепловом состоянии теплозащитных композиционных материалов при аэродинамическом нагреве // Труды МАИ, 2014, №72: http://www.mai.ru/science/trudy/published.php?ID=47581