УДК 519.685

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ НА С++ ДЛЯ АВТОМАТИЗАЦИИ ОБРАБОТКИ ФУНКЦИОНАЛЬНЫХ ВЫРАЖЕНИЙ

Теодор Заркуа

Грузинский технический университет

Аннотация

Данная работа посвящена вопросам программной реализации расширения алгоритмического языка С++ путем привнесения в него средств для автоматизации обработки функциональных выражений, заданных в естественном виде. Указанные средства существенно используют свойства бесскобочных (т.н. польских) записей функциональных (алгебраических) выражений, основывающихся на понятии сопряженного, исследованных в цикле работ автора, посвященных этому вопросу.

Дается описание некоторых ключевых моментов реализации и описание интерфейса предлагаемого программного обеспечения. Подробно излагается реализация символьного дифференцирования, опирающаяся на использовании предложенного автором внутреннего языка для описания функциональных преобразований алгебраических выражений. Несмотря на то, что здесь описывается программное обеспечение на языке С++,подход имеет общий характер и его основные положения легко могут быть перенесены на другие объектно-ориентированные языки программирования. В заключении, автор намечает некоторые направления для дальнейших разработок.

Abstract

This work is devoted to the issues of a software implementation for expansion of algorithmic language C ++, by bringing in it the automatization of the processing of functional expressions, set in a natural form. These funds are substantially used properties of bracket-free (the so-called Polish) notation of functional (algebraic) expressions that are based on the concept of the conjugation, investigated in a series of works of the author on the subject.

A description of some of the key aspects of implementation and a description of the interface of the proposed software. More specifically sets out the implementation of differentiation, based on the use of the internal language proposed by the author to describe the functional transformations of algebraic expressions. Despite the fact that here is describes the software in C ++, the approach is general in nature and its main points can easily be transferred to other object-oriented programming languages. In conclusion, the author outlines some directions for further research.

Введение

Проблемные программисты, работающие на алгоритмических языках (в том числе на С++) до сих пор не имели удобных средств для обработки функциональных выражений, заданных в естественном (инфиксном) виде, скажем, строкой. Между тем во многих случаях, наличие таких возможностей позволило бы обходить использование численных методов, заменяя их прямыми. Кроме того, упомянутые возможности позволили бы проблемному программисту выносить функциональные выражения в число данных, задаваемых программе извне, а не оставлять заготовку функционального выражения в исходном тексте программы, тем самым обрекая её на необходимость перетранслирования для каждого нового заполнения упомянутого выражения. Более того,

возможно заслуживает исследования вопрос о группе задач, решение которых становится принципиально возможным только благодаря привнесению стандартных средств для обработки функциональных выражений на уровень алгоритмических языков.

Представление функциональных выражений

Общеизвестно, что для атоматизации обработки алгебраических выражений удобно использовать бесскобочные, т.н. польские записи, названные так в честь их автора, польского математика Яна Лукасевича [1]. В работах [2]-[5] были исследованы свойства бесскобочных выражений, основанные на **понятии сопряженного, введенного автором**. Основополагающий результат этих работ кратко можно сформулировать следующим образом – **имея префиксную форму функционального выражения нет необходимости преобразовывать ее к постфиксному виду, т.к. для вычисления значения выражения достаточно обработать префиксную форму справа налево алгоритмом, идентичным по своей сложности с классическим алгоритмом, применяемым для этой цели к постфиксной форме**.

Совершенно очевидно, что этот результат **позволил акцентироваться на получении префиксной формы функционального выражения и реализации его функциональных преобразований.** В работе [5] довольно подробно изучен вопрос о том как удобнее получить префиксную форму функционального выражения.

В работе [6] было предложено представлять префиксные выражения в виде последовательнсти, каждый элемент которой принадлежит типу, описанному следующим образом:

```
struct Telem
{char operand;
 union {char nomer;  double value;
        }
};
```

При этом, если значение поля **operand** равно **0**, то это операция и нас интересует ее номер (поле **nomer**), а если **operand** не равен **0**, то это операнд. Причем, если значение поля **operand** равно **1**, то это переменная и в дальнейшем нас может интересовать ее номер, т.е. поле **nomer** (если переменных несколько) и значение (которое будет задаваться извне), а если **operand** равен **2**, то это константа и тогда для нас представляет интерес поле **value**, содержащее соответствующее значение. В этой же работе было предложено в качестве контейнера для последовательности, соответствующей значению префиксной записи использовать класс, являющийся потомком стандартного класса **vector**, в который добавлены такие возможности, как выделение подвектора, конкатенация вектора с элементом и вектора с вектором, вставка вектора в вектор и элемента в вектор,…

Реализация символьного дифференцирования

В работе [6] был предложен внутренний язык для представления формул дифференцирования, в котором каждая формула описывается строкой. При этом, константы представляют сами себя (**e** обозначает число Непера), операция, для наглядности, обозначается соответствующим знаком, элементарная функция – соответствующей малой буквой латинского алфавита, начиная с буквы ‘**f**’, первый и второй операнды выражения – соответственно буквами ‘**a**’и ‘**b**’ , производные этих операндов – соответственно, буквами ‘**A**’ и ‘**B**’. В этих терминах, основные формулы дифференцирования записываются следующим образом:

Выражение для дифференцирования		Производная на внутреннем языке
В префиксной форме	На внутреннем языке	
+uv	+ab	+AB
-uv	-ab	-AB
*uv	*ab	+*Ab*aB
/uv	/ab	/-*Ab*aB*bb
^uv	^ab	+**Ab^a-b1**A^abqa
sin u	Fa	*gaA
cos u	Ga	s*faA
tg u	Ha	/Anga
ctg u	Ia	s-Anfa
arcsin u	Ja	/Ao-1na
arcos u	Ka	s/Ao-1na
arctg u	ℓa	/A+1na
arcctd u	Ma	s/A+1na
sqr u	Na	*2*aA
sqrt u	Oa	/A*2oa
exp u	Pa	*paA
ℓn u	Qa	/Aa
ℓg u	Ra	/A*aqD
s u	Sa	sA

Для того, чтобы понять приведенное, необходимо учесть, что выражения приведены в префиксном виде, и что **Sin** обозначается буквой **f**, **Cos** - буквой **g**, **Ln** – буквой **q**, **Lg** - буквой **r** и т.д.. В частности, запись **/A*aqD** есть дробь, в числителе которой производная аргумента, а в знаменателе – произведение, первый операнд которого (**a**) является аргументом исходной функции, второй (**qD**) значением выражения **Ln(10)**.

После вышеприведенного практически ясен смысл фрагмента реализации:

```
string Formulebi[]={"+AB","-AB","+*Ab*aB","/-*Ab*aBnb","+**Ab^a-b1**B^abqa",
    "*gaA","s*faA","/Anga","s-Anfa","/Ao-1na","s/Ao-1na",
    "/A+1na","s/A+1na","*2*aA","/A*2oa","*paA","/Aa","/A*aqD","sA"};
string Funqciebi[]={"sin","cos","tg","ctg","arcsin","arcos","arctg","arcctd","sqr","sqrt","exp","ln", "lg","s"}; string mq="+-*/^";
```

Для полной ясности достаточно сказать, что **Formulebi** – это массив формул дифференцирования, **Funqciebi** – это массив, содержащий названия функций, фигурирующих в формулах, а в строке **mq** перечислены арифметические операции, причем знак ^ означает возведение в степень. Осталось добавить, что буква **D** в предпоследней формуле означает константу **10**. Благодаря этому последнему обозначению удалось добиться того, что в формулах каждый элемент представлен ровно одним символом, что существенно упростило реализацию.

Функция, реализующая дифференцирование, получая на вход исходное выражение в префиксном виде, по его первому элементу отыскивает соответствующую формулу и строит результат в полном соответствии с этой формулой. Совершенно очевидно, что при

этом функция должна уметь выделять из заданного выражения первый и второй аргументы, чему способствует подобранная для выражения структура данных. Очевидно, при необходимости, функция должна обращаться к самой себе. Немаловажным моментом является умение функции упростить полученный результат.

Использование программного обеспечения

Осталось перечислить средства, которые уже имеются в распоряжении проблемного программиста C++ для обработки функциональных выражений.

Для представления функционального выражения используется класс **Tgam**, который описывается следующим образом: **typedef Vector<Telem> Tgam;** Здесь **Vector** – это потомок стандартного класса **vector**, расширенный операциями "|" и "|=" (соответственно, конкатенация и коррекция конкатенацией), а также методами **SubVec, Find** и **Card** (соответственно, выделение подвектора, поиск в векторе элемента, мощность множества элементов вектора).

Дальше приведем краткое описание функций, работающих с классом **Tgam**.

Tgam Togam(string x, int &res) – эта функция обеспечивает перевод функционального выражения, содержащего не более одной переменной, обозначенной **x,** заданного в инфиксном виде строкой - в префиксное выражение, т.е. объект класса **Tgam**. При этом, если все в порядке, то второй параметр возвращает отрицательное значение (**-1**). Если же исходное выражение не корректно, второй параметр дает неотрицательное число – номер позиции, отсчитанный с нуля, в котором обнаружена ошибка (строго говоря – номер позиции, начиная с которого исходное выражение не может быть корректным), при этом сама функция возвращает объект класса **Tgam**, длиной **0**.

string Tostring(Tgam x) – эта функция обеспечивает преобразование, обратное предыдущему. Выражение из служебного (внутреннего) вида (объект класса **Tgam**) переводится в строку, содержащую соответствующее ему инфиксное выражение. В реализации представляют интерес два обстоятельства. Первое – то, что перевод производится обработкой префиксного выражения справа налево, в соответствии с алгоритмом, предложенным в [3-5]. Второе, обеспечение получения префиксного выражения с минимальным количеством скобок.

double Gamotvla(Tgam G, double x=1) – данная функция вычисляет значение выражения, заданного во внутреннем представлении (первый аргумент) для значения переменной (если таковая есть), заданного вторым аргументом. Как видим, результат выдается в виде **double.** Вычисления также производятся по правилам, предусмотренным для типа **double.** Автор рассматривает целесообразность реализации аналога данной функции, работающего, по правилам для целых и дающего результат также целого типа.

Tgam Dif(Tgam x) – данная функция по заданному во внутреннем представлении корректному префиксному выражению (не более чем с одной переменной), возвращает также во внутреннем представлении результат его дифференцирования.

Tgam Gamartiveba(Tgam G) – эта функция обеспечивает упрощение выражения, заданного во внутреннем представлении и используется функцией дифференцирования перед возвращением результата, но может быть использована и независимо. Для иллюстрации работы данной функции рассмотрим конкретный пример. Результатом дифференцирования, соответствующего выражению **(2.14*x-15)*cos(x)** будет выражение, инфиксная форма которого имеет вид **(0*x+2.14*1-0)*cos(x)+ (2.14*x-15)*s(sin(x)+1).** И лишь благодаря функции упрощения результат принимает вид, инфиксное соответствующее которого выглядит более привычным образом: **2.14*cos(x)+ (2.14*x-15)*s(sin(x)).** Напомним, что **s(x)** равносильно **-x**.

Tgam a(Tgam x) и **Tgam b(Tgam x)** – эти функции возвращают, соответственно первый и второй операнды, заданного выражения и используются функцией **Dif,** но могут представлять интерес и непосредственно.

Заключение

В дальнейшем предлагается развивать данную проблематику в следующих направлениях:

1. Разработать класс, предназначенный для проблемных программистов, работающих на языке C++, с целью максимально повысить удобство средств для обработки функциональных выражений, предоставляемых в их распоряжение.
2. Перенести предлагаемый подход на другие алгоритмические языки программирования.
3. Исследовать возможность реализации автоматизации обработки функциональных выражений со многими переменными.
4. Применить предлагаемое программное обеспечение для обучения студентов дифференцированию функций.
5. Исследовать действенность авторского метода реализации символьного дифференцирования (через внутренний язык) применительно к символьному интегрированию.
6. Выделить класс задач, для которых использование предлагаемых средств качественно облегчает и улучшает решение.

Остается сказать, что автор и его ученики работают по всем вышеперечисленным направлениям и в скором времени представят коллегам полученные ими новые результаты.

Список использованных источников:

1. Грис Д. Конструирование компиляторов для цифровых вычислительных машин. - Москва, "Мир", 1975.

2. Заркуа Т. К свойству перевернутой польской записи. Internet-Edication-Sciense-2008. New Informational and Computer Tecnologies in Education and Sciense. H. Intelligence Information Systems. 2008 year, p.543-544

3. Заркуа Т. Некоторые способы обработки функциональных выражений. Winter Programming School. Материалы зимней школы по программированию, Харьков, ХНУРЭ, 2009, p. 205-211.

4. Zarkua T. An Approach to Functional Expressions Processing Automation. Billetin №2 of Saint Andrew the First-Called Georgian University of the Patriarchy of Georgia, Tbilisi, 2009, p. 50-61.

5. Т.Заркуа.Автоматизация обработки функциональных выражений на уровне алгоритмических языков. Internet-Edication-Sciense-2010. New Informational and Computer Tecnologies in Education and Sciense.p.201-206.

6. Т.Заркуа.К вопросу об использовании понятия сопряженного по отношению к бесскобочным выражениям. Internet-Edication-Sciense-2012. New Informational and Computer Tecnologies in Education and Sciense.p.156-157.