Дополнительный лекционный материал по деревьям выражений (Зайцев В.Е., 14 мая 2014 г.)

Выражения:

- арифметические;
- логические;
- информатические.

Нелинейность и рекурсивность выражений. Фон Неймановская нежелательность скобочных структур.

Деревья выражений. Фон Неймановская причина их бинарности.

Бесскобочность представления выражений деревьями.

Разнофиксные формы записи выражений:

- префиксная;
- инфиксная;
- постфиксная.

Их связь с обходами деревьев выражений.

Фон Неймановские преимущества бесскобочной записи.

Пример: псевдокод программы построения и распечатки дерева выражения по его линейной скобочной (инфиксной записи).

Особенности примера:

- упрощённые арифметические выражения;
- рекурсивно-итеративная интерпретация при удалении распечатанного дерева.

Полурекурсивная версия: сочетание рекурсивного спуска направо с итеративной обработкой слева.

Реализация парсера по грамматике. Нормальный алгоритм.

```
; Наглядный пример грамматики простого предшествования
; на основе грамматики: page 476 АНО (volume 1)
; Copyright (c) Иваницкий Николай, 1997.
; [E)] <=> A
; ВХОД: На ленте - выражение, заключённое с двух сторон в
символы $
; ВЫХОД: YES!, если цепочка допускается грамматикой.
#LENTA $(((a+a)*a+a*a)+((a+a)*a+a*a))$

; Отношения предшествования:
T) -> T>)
T+ -> T>+
F) -> F>)
F+ -> F>+
F* -> F>*
[T]) -> [T]>)
[T]+ -> [T]>+
A) -> A>)
A+ -> A>+
A* -> A>*
a) -> a>)
a+ -> a>+
a* -> a>*
)) -> )>)
)+ -> )>+
)* -> )>*
(E -> (<E
(T -> (<T
(F -> (<F
([T] -> (<[T]
(a -> (<a
(+ -> (<+
(( -> (<(
+T -> +<T
+F -> +<F
+a -> +<a
```

```
+( -> +<(
*a -> *<a
*( -> *<(
T$ -> T>$
F$ -> F>$
[T]$ -> [T]>$
A$ -> A>$
a$ -> a>$
)$ -> )>$
$E -> $<E
E$ -> E>$
$T -> $<T
$F -> $<F
$[T] -> $<[T]
$a -> $<a
$( -> $<(
$+ -> $<+
;
; Правила свёртки (обычные продукции, вывернутые
наизнанку)
<E+[T]> -> E
<+[T]> -> E     Грамматика взята из
<[T]> -> E        первого тома Axo (page 476)
<T> -> [T]
<T*F> -> T
<(A> -> F
<F> -> T
<a> -> F
<E)> -> A
<E> ->
; приведём ко всюду определённому алгоритму:
$$ ->. YES!
> ->.  _Error:_пропущено_начало_основы_
< ->.  _Error:_пропущен_конец_основы!_
  ->.  _General_fault_error!_
;
;  P.S. а ведь если расширить синтаксис NAM до
;       a1 -> a2 [: b1 -> b2]
;  причём команда после : выполняется <= выполняется
команда до :,
;  то получится
```

Другой пример:
преобразование выражения из инфиксной формы в постфиксную.
Нерекурсивная версия.

Алгоритм Рутисхаузера (1951).

«Танцевальная процессия вокруг скобочных скал».

Один из наиболее ранних алгоритмов.

Предполагает полную скобочную структуру выражения – такую форму записи, при которой порядок действий задается расстановкой скобок.

Неявное старшинство операций при этом не учитывается. Например:

$$D = ((C-(B*L))+K)$$

Обрабатывая выражение с полной скобочной структурой, алгоритм присваивает каждому символу – лексеме исходного выражения -- номер уровня по следующему правилу:

- если это открывающаяся скобка или переменная, то значение уровня увеличивается на 1;
- если лексема -- знак операции или закрывающаяся скобка, то уровень уменьшается на 1.

Для выражения (A+(B+C)) значения уровней таковы:

№ литеры	1	2	3	4	5	6	7	8	9
Символы (лексемы!)	(	A	+	(	B	*	C	)	)
Номера уровней	1	2	1	2	3	2	3	2	1

Основные этапы алгоритма Рутисхаузера:

1. расставить уровни;

2. отыскать элементы строки с максимальным значением уровня;

3. выделить тройку - два операнда с максимальным значением уровня и операцию, которая заключена между ними;

4. результат выполнения выделенной операции обозначить вспомогательной переменной;

5. из исходной строки удалить выделенную тройку вместе с её скобками, а на ее место поместить вспомогательную переменную, обозначающую результат, со значением уровня на единицу меньшим, чем у выделенной тройки;

6. выполнять п.п. 2 - 5 до тех пор, пока во входной строке не останется одна переменная, обозначающая общий результат выражения.

Пример разбора :

Генерируемые тройки	Выражение
	((((A + B) * C) / D) - E) 0 1 2 3 4 5 4 5 4 3 4 3 2 3 2 1 2 1 0
+ A B - > R	(((R * C) / D) - E) 0 1 2 3 4 3 4 3 2 3 2 1 2 1 0
* R C -> S	((S / D) - E) 0 1 2 3 2 3 2 1 2 1 0
/ S D -> Q	(Q - E) 0 1 2 1 2 1 0
- Q E -> T	T

Алгоритм Бауэра и Замельзона

Ранний стековый метод.

Используются два стека и таблица функций перехода.
Один стек (T) используется при трансляции выражения, а второй (E) -- во время интерпретации выражения.

В таблице переходов задаются функции, которые должен выполнить транслятор при разборе выражения:

- f1: заслать операцию из входной строки в стек T; читать следующий символ строки;
- f2: выделить тройку – взять операцию с вершины стека T и два операнда с вершины стека E;

  вспомогательную переменную – результат операции занести в стек E;

  заслать операцию из входной строки в стек T;

  читать следующую лексему строки;

  f3: исключить лексему из стека T;

  читать следующий символ строки;
- f4: выделить тройку – взять операцию с вершины стека T и два операнда с вершины стека E;

  вспомогательную переменную – результат операции, занести в стек E;

  по таблице определить функцию для данной лексемы входной строки;
- f5: выдача сообщения об ошибке;
- f6: завершение работы.

Таблица переходов для алгебраических выражений будет иметь вид (знак $ является признаком пустого стека или пустой строки):

		Операция из входной строки					
		$	(	+	-	*	/	)
Операция на вершине стека T	$	6	1	1	1	1	1	5
	(	5	1	1	1	1	1	3
	+	4	1	2	2	1	1	4
	-	4	1	2	2	1	1	4
	*	4	1	4	4	2	2	4
	/	4	1	4	4	2	2	4

Выражение просматривается слева направо и над каждой лексемой выполняются следующие действия: если очередная лексема входной строки является операндом, то она безусловно переносится в стек E; если это операция, то по таблице функций перехода определяется номер функции для выполнения.

Для выражения *A+(B-C)*D* будут проделаны следующие действия:

Стек E	*Стек T*	Лексема	Функция	Тройка
$	*$*	*A*		
$A	*$*	*+*	1	
$A	*$+*	*(*	1	
$A	*$+(*	*B*		
$AB	*$+(*	*-*	1	
$AB	*$+(-*	*C*		
$ABC	*$+(-*	*)*	4	*- B C ->R*
$AR	*$+(*	*)*	3	
$AR	*$+*	***	1	
$AR	*$+**	*D*		
$ARD	*$+**	*$*	4	** R D ->Q*
$AQ	*$+*	*$*	4	*+ A Q ->S*
|$S	*$*	*$*	конец	

Алгоритм сортировочной станции Дейкстры — способ разбора математических выражений, представленных в обычной инфиксной нотации.

Результат -- в виде обратной польской записи или в виде дерева выражения.

Алгоритм изобретен Эдсгером Дейкстрой и назван им алгоритмом сортировочной станции, поскольку напоминает действие железнодорожной сортировочной станции.

Имеется байка про вагоны с туалетами.

Так же, как и при вычислении значений выражений в обратной польской записи, алгоритм работает при помощи стека.

Для преобразования в обратную польскую нотацию используется 2 очереди: входная и выходная, и стек для хранения операций, еще не добавленных в выходную очередь.

При преобразовании выражения считывается лексема и производятся действия, зависящие от её конкретного вида.

Пока не все синтаксические символы (лексемы) обработаны:

Считать лексему.

Если это число, то добавить в очередь вывода.

Если — функция, то поместить в стек.

Если — разделитель аргументов функции (например запятая), то:
пока символ на вершине стека не открывающая скобка, перекладывать операции из стека в выходную очередь. Если в стеке не было открывающей скобки, то в выражении пропущен разделитель аргументов функции (запятая), либо пропущена открывающая скобка. Диагностика!

Если — операция op1, то:
пока присутствует на вершине стека лексема операции op2, и
либо операция op1 левоассоциативна и её приоритет меньше чем у операции op2 либо равен ему,
или операция op1 правоассоциативна и её приоритет меньше чем у op2,
переложить op2 из стека в выходную очередь;
положить op1 на стек.

Если — открывающая скобка, то положить её на стек.

Если — закрывающая скобка:
пока на вершине стека не открывающая скобка, перекладывать операции из стека в выходную очередь.

вынуть открывающую скобку из стека, но не добавлять в выходную очередь.

если на вершине стека лексема функции, добавить её в выходную очередь.

если стек закончился до того, как была встречена открывающая скобка, то в выражении пропущена скобка.

Диагностика!

Если очередь лексем на входе пуста:

пока есть операции в стеке:

если операция на вершине стека — скобка, то в выражении присутствует незакрытая скобка.

Диагностика!

Переложить операцию из стека в выходную очередь.

Конец!!!

Каждая лексема-число, функция или операция выводится только один раз.

Каждый символ-функция, операция или круглая скобка будут добавлены и удалены из стека по одному разу.

Постоянное количество операций на лексему, линейная сложность алгоритма O(n).

Простой пример:

Входная очередь лексем: 3 + 4

Добавим 3 в выходную очередь (если прочитан операнд, то он сразу направляется на выход).

Помещаем + в стек операций.

Операнд 4 следует на выход.
Выражение прочитано, все оставшиеся в стеке операции выталкиваются на выход. В нашем примере в стеке содержится только +.

Выходная строка: 3 4 +

В данном примере проявляются некоторые правила: все числа переносятся в выходную строку сразу после прочтения; когда выражение прочитано полностью, все оставшиеся в стеке операции выталкиваются в выходную строку. Операнды для них занесены туда ранее!

Сложный пример.

Приоритеты:

- ^ высокий
- * / средний
- + - низкий

Трасса процесса обработки выражения 3 + 4 * 2 / (1 - 5)^2 :

Лексема: 3.
Тип лексемы – операнд.
3 идёт в выходную очередь
Выход: 3

Лексема: +
Операция.
+ идёт на стек.
Выход: 3
Стек: +

4
Операнд
4 идёт на выход
Выход: 3 4
Стек: +
*
Операция, кладём * на стек
Выход: 3 4
Стек: + *

2
Операнд, 2 идёт на выход
Выход: 3 4 2
Стек: + *

/

Выталкиваем * из стека на выход, кладём / на стек
Выход: 3 4 2 *
Стек: + /

(
Открывающая скобка (идёт в стек
Выход: 3 4 2 *
Стек: + / (

1
Операнд 1 идёт на выход
Выход: 3 4 2 * 1
Стек: + / (

-
Операция - в стек
Выход: 3 4 2 * 1
Стек: + / (-

5
5 идёт на выход
Выход: 3 4 2 * 1 5
Стек: + / (-

)
Выталкиваем - из стека в выходную очередь,
Выталкиваем (
Выход: 3 4 2 * 1 5 -
Стек: + /

^ символ возведения в степень (** в Фортране!)
^ -- на стек
Выход: 3 4 2 * 1 5 -
Стек: + / ^

2
Операнд 2 идёт на выход
Выход: 3 4 2 * 1 5 - 2
Стек: + / ^

Конец выражения

Выталкиваем все элементы из *стека* в выходную *очередь* (инверсия, однако!)

Выход: 3 4 2 * 1 5 - 2 ^ / +

Программа на языке Си.