**ФГБОУ ВПО**

Уфимский государственный авиационный технический университет

***«*Построение лексического и синтаксического анализатора, выполнение программы для заданной конструкции цикла*»***

«**Методы построения трансляторов**»

***Выполнил***

Вахрамов С.В.

**г. Уфа 2017**

# **Задание на лабораторную работу**

# Построить лексический, синтаксический анализаторы и выполнить заданную конструкцию.

Во всех приведенных вариантах идентификатор начинается с буквы, за которой может следовать произвольное количество букв или цифр (до 255 символов), константа – целочисленная константа, возможно со знаком, в диапазоне от -32768 до +32767. Ключевое слово output представляет собой инструкцию, по которой содержимое следующей за ней переменной выводится пользователю на экран. Ключевое слово input представляет собой инструкцию, по которой в следующую за ней переменную загружается введенное пользователем с клавиатуры значение.

**Конструкция** – оператор цикла без счетчика с постусловием.

do

<операторы>

loop until <логическое выражение>

Оператор цикла повторяет <операторы>, до тех пор, пока условие <логическое выражение> не станет истинным. Цикл выполняется хотя бы один раз.

* <логическое выражение> → <выражение сравнения> | <логическое выражение><бинарная логическая операция><выражение сравнения>
* <выражение сравнения> → <операнд> | <операнд><операция сравнения><операнд>
* <операция сравнения> → < | > | = | <>
* <бинарная логическая операция> → and|or
* <операнд> → <идентификатор>|<константа>
* <операторы> → <операторы>; <оператор> | <оператор>
* <оператор> → <идентификатор> = <арифметическое выражение> | input <идентификатор> | output <операнд>
* <арифметическое выражение> → <операнд> |
* <арифметическое выражение><арифметическая операция><операнд>
* <арифметическая операция> → + | - | / | \*

# **Теория**

* **Лексический анализ**

Лексическим анализом будем называть процесс перевода исходного текста программы в последовательность лексем. За выполнение лексического анализа отвечает лексический анализатор (сканер).

Лексемами будем считать минимально значимые с точки зрения синтаксиса языка единицы. В зависимости от языка, можно выделить различные типы лексем. Тем не менее, для языков высокого уровня можно определить следующие основные элементы:

- ключевые слова (begin, while, for), - идентификаторы (i, j, myfunction), - константы (123, 1E-5, “name”),

- знаки операций (:=, ++, <<),

- разделители (, ; .)

В процессе лексического анализа комментарии и незначащие символы никак не

учитываются, и на выходе лексического анализатора формируется последовательность лексем, которая может быть подана на вход синтаксического анализатора.

В самом анализаторе лексема описывается типом лексемы и некоторой информацией, которая может зависеть от типа лексемы. Для хранения информации по лексемам таких типов, как идентификаторы и константы, организуются отдельные таблицы. Данные этих таблиц будут использоваться на следующих этапах трансляции.

Сам алгоритм лексического анализа в большинстве случаев может быть построен как анализатор автоматного языка. В простых случаях границы лексем определяются по наличию незначащих символов или разделителей, после чего распознается сама лексема. Другим способом может быть определение типа лексемы по первому значащему символу, непосредственно после чего, может быть начато распознавание самой лексемы, которое продолжается, пока не встретится символ, противоречащий правилам построения лексем данного типа. В более сложных случаях лексический анализатор может представлять собой анализатор с возвратами.

При распознавании лексем используется так называемая таблица терминальных символов, к которым относят ключевые слова, знаки операций и другие символы. В этой таблице содержатся такие сведения, как код лексемы, символьное представление лексемы,

признак того, является ли данная лексема разделителем (например, знаки операций, как правило, являются разделителями) и т.д.

После распознавания типа лексемы, для идентификаторов и констант проверяется, есть ли уже такой идентификатор или константа в таблице идентификаторов или констант. Если такого идентификатора или константы еще нет, то в соответствующую таблицу помещается необходимая информация. После этого производится формирование выходной лексемы с сопутствующей информацией (тип лексемы, индекс в таблице идентификаторов или констант, положение лексемы в исходном тексе программы и т.д.).

* **Синтаксический анализ**

Метод рекурсивного спуска относится к нисходящим методам анализа КС-языков. Класс КС-грамматик, для которых может быть применен метод рекурсивного спуска, довольно узок, однако достаточно часто с использованием эквивалентных преобразований грамматику можно привести к классу, допускающему анализ методом рекурсивного спуска. Основным ограничением, накладываемым на грамматику G=(S,VN,VT,R) является то, что для правил A→α1|α2|...|αN, в которых нетерминальный символа A стоит в левой части, множества First1(α1), First1(α2), ..., First1(αN) не должны пересекаться, то есть:

First1(αi) ∩ First1(αj) = ∅, ∀ αi, αj: {A→αi, A→αj}⊆R

Собственно, построение анализатора методом рекурсивного спуска выполняется следующим образом:

1. Для каждого нетерминального символа A грамматики определяется своя процедура, которая отвечает за разбор цепочек, выводимых из нетерминального символа A.

2. Распознающая процедура для нетерминального символа A, принимает на вход текущую позицию просмотра входной строки и сопоставляет терминальный символ a, стоящий в позиции просмотра с символами множеств First1(α1), First1(α2), ..., First1(αN), составленных для правых частей правил грамматики A→α1|α2|...|αN.

3. В том случае, если символ a входит во множество First1(αi) для правила A→αi (a∈First1(αi)), то управление передается той ветви анализа процедуры, которая отвечает за распознавание соответствующей альтернативы A→αi.

4. Ветвь анализа, соответствующая альтернативе A→αi, строится как последовательность действий по распознаванию цепочки

αi=v1v2...vN, где v1,v2,...vN∈(VN∪VT).

При этом, если vi является терминальным символом, то он сопоставляется с текущим анализируемым символом a. Если символы совпадают (a=vi), то происходит переход к следующей позиции просмотра входной строки, в противном случае, фиксируется синтаксическая ошибка во входной строке.Eсли vi является нетерминальным символом, то вызывается процедура, отвечающая за разбор цепочек, выводимых из нетерминального символа vi. Таким образом, реализации

функции может содержать как обращение к другим процедурам, так и рекурсивное обращение к самой себе в том случае, когда правила для нетерминального символа A содержат рекурсию.

5. В том случае, если символ a не входит ни в одно из множеств First1(αi) (a∉First1(αi), i=1..N), и среди правил для A есть аннулирующее правило A→ε, то распознающая процедура для нетерминального символа A считается успешно завершенной. В противном случае фиксируется синтаксическая ошибка во входной строке.

На практике разработку анализатора методом рекурсивного спуска удобно выполнять по расширенной бэкусовой нормальной форме, в которой помимо обозначения альтернативы “|” при записи правил грамматики используется обозначение необязательных (факультативных) элементов “[...]” и повторяющихся элементов (итерация) “{...}”. В этом случае альтернативным частям правил будут соответствовать разные ветви условного оператора (или оператора множественного выбора) соответствующей распознающей процедуры. Факультативным элементам будут соответствовать отдельные условные операторы в ветвях соответствующих альтернатив, а повторяющимся элементам – циклы в ветвях соответствующих альтернатив.

# **Описание хода выполнения работы**

В процессе выполнения лабораторных работ пошагово был построен лексический анализатор, затем синтаксический анализатор, затем интерпретатор заданной конструкции.

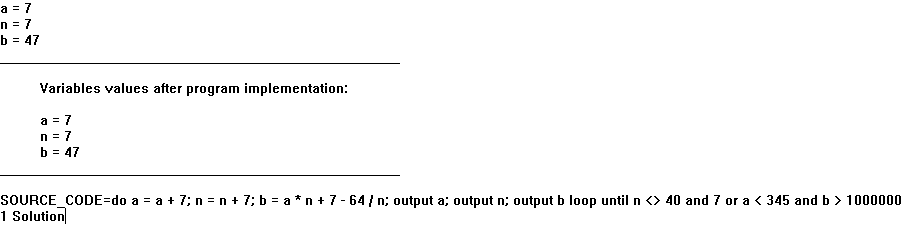
Решение задачи производилось с помощью языка Visual Prolog 5.2.1. Его особенности заключаются в том, что грамматика задаётся в явном виде, и проведение лексического и синтаксического анализа становится легко выполнимой задачей. В дальнейшем, для интерпретации программы не нужно будет использовать ПОЛИЗ (ПОЛьскую Инверсную Запись), так как программу можно просто выполнять пошагово по методу «как есть». То есть, например, видя конструкцию «а = 7 \* n», мы решим задачу ''7 \* n'' и запишем ответ в базу данных с ячейкой ''a''.

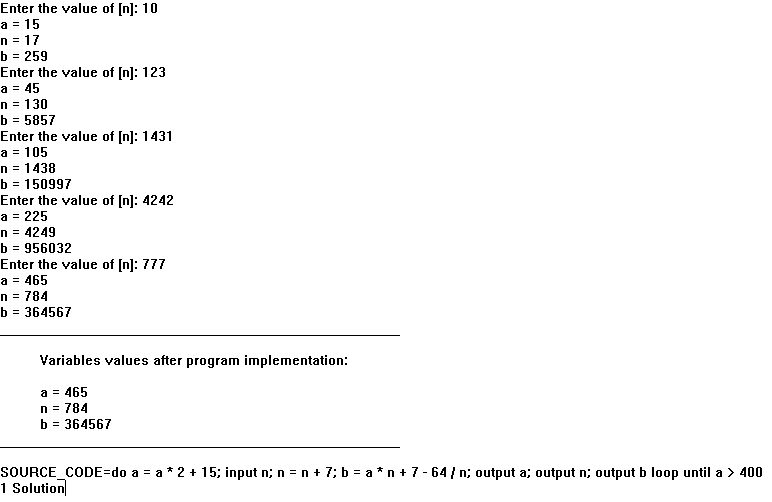
* **Основные этапы решения задачи**

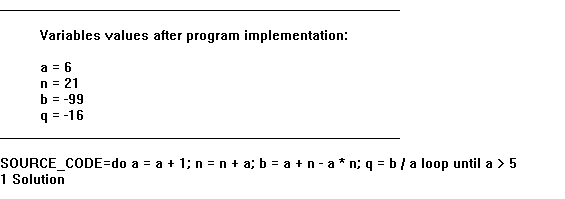
1. **Заполнить базы данных ключевых слов, спец. символов.**
2. **Считать заданный код программы**
3. **Провести лексический анализ**
4. **Провести синтаксический анализ**
5. **Инициализировать переменные для внутреннего выполнения кода**
6. **Выполнить код программы**
7. **После остановки программы показать таблицу значений переменных (идентификаторов)**

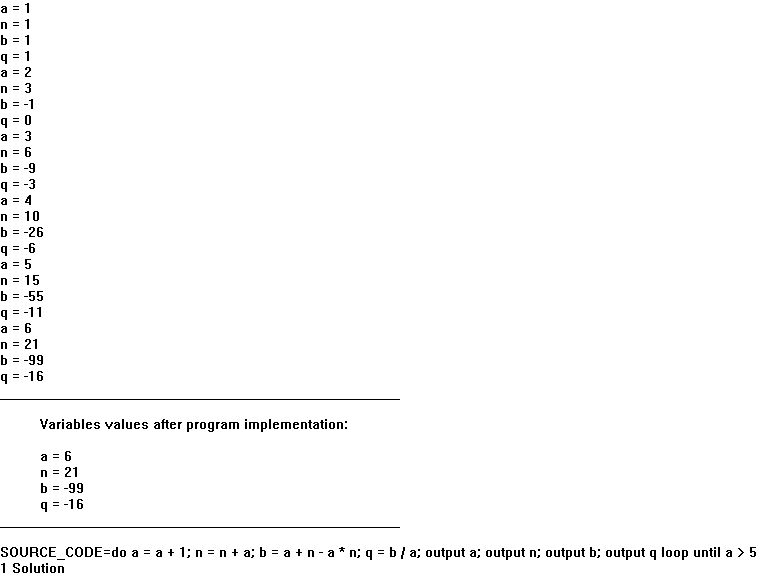
**Стоит заметить, что если какой-либо шаг выполнен неуспешно, то предыдущие шаги не выполняются. Программа завершается с ответом «Нет решения».**

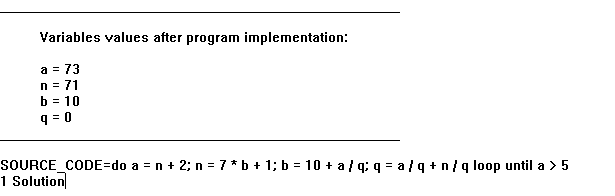
# **Примеры результатов работы (Visual Prolog 5.2.1)**

****

****

****

****

****

# **Вывод**

В результате выполнения работы были получены навыки построения лексического и синтаксического анализаторов. Теперь я имею представление о том, как строятся трансляторы исходных кодов программ.

В результате решения поставленных задач, были получены колоссальные навыки программирования на языке Visual Prolog.

Стоит заметить, что многие задачи и подзадачи на нём удалось решить легче (в моём понимании), чем на других процедурных или объектно-ориентированных языках программирования.