Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Кафедра «Информационных технологий и систем»

**Проектирование синтаксического анализатора**

Лабораторная работа

по дисциплине:

«Теория языков программирования и методы трансляции»

Разработал:

Студент группы 8091

Кудряшов И. С. \_\_\_\_\_\_

«\_\_» \_\_\_\_\_2020г

Проверил:

Макаров В. А. \_\_\_\_\_\_

«\_\_» \_\_\_\_\_2020г

**Великий Новгород**

**2020**

**Постановка задачи**

**Формулировка**

Разработать программу, решающую задачу синтаксического разбора исходного текста программы. Распознаваемый язык выбирается самостоятельно. В качестве прототипов может быть выбран синтаксис следующих языков программирования: Паскаль, СИ, Бейсик.

**Грамматика**

STATS -> STATS STAT

STATS -> STAT

STAT -> ASSIGNMENT\_EXP ;

STAT -> { STATS }

ASSIGNMENT\_EXP -> var = EXP

EXP -> AND\_EXP

EXP -> OR\_EXP

EXP -> LOG\_EXP

AND\_EXP -> LOG\_EXP

AND\_EXP -> LOG\_EXP && EXP

OR\_EXP -> AND\_EXP

OR\_EXP -> LOG\_EXP || EXP

LOG\_EXP -> ADDITIVE\_EXP

LOG\_EXP -> ADDITIVE\_EXP comp\_operator ADDITIVE\_EXP

ADDITIVE\_EXP -> MULT\_EXP

ADDITIVE\_EXP -> ADDITIVE\_EXP pm\_operator MULT\_EXP

MULT\_EXP -> CAST\_EXP

MULT\_EXP -> MULT\_EXP md\_operator CAST\_EXP

CAST\_EXP -> UNARY\_EXP

CAST\_EXP -> ( type\_name ) CAST\_EXP

UNARY\_EXP -> unary\_operator CAST\_EXP

UNARY\_EXP -> prefix\_operator UNARY\_EXP

UNARY\_EXP -> var

UNARY\_EXP -> const

**Описание реализации**

**Описание выбора метода синтаксического анализа**

Был выбран восходящий метод синтаксического анализа, а именно метод «перенос-опознание» (shift-reduce). Поскольку грамматика имела пересечения множества выбора и нисходящие методы были неприменимы, было принято решение использовать именно данный метод.

**Описание структуры синтаксического анализатора**

В методе «перенос-опознание» мы находим FIRST() и NEXT() от каждого элемента грамматики и на основе четырёх правил делаем управляющую таблицу (или две таблицы – таблица переносов и таблица свертываний, а потом соединяем их).

**Принцип переноса:**

Управляющая таблица строится по следующему правилу. Операция ПЕРЕНОС для магазинного символа **<В>** и терминала aвносится в таблицу, когда выполняется одно из условий:

1) имеется правило <A>→ <B>β , β - цепочка из терминальных и нетерминальных символов, ***a ∈*** FIRST (β);

2) если вместо **<В>** стоит маркер дна магазина и ***a ∈*** FIRST (S), где S – стартовый нетерминал грамматики.

**Принцип свертывания:**

В управляющей таблице ячейка на пересечении магазинного символа **<В>** и терминала aобеспечивает вызов процедуры ОПОЗНАТЬ в следующих случаях:

1. для правила <A>→ α<B> терминал ***a ∈*** NEXT (<A>)
2. если <B> является стартовым нетерминал грамматики ***a =*** -|

После построения управляющей таблицы мы формируем некоторые правила свертки, которые позволяют нам свернуть выражение в стеке к определенному нетерминалу грамматики. При сдвиге мы добавляем новую лексему в стэк и смещаем указатель.

**Описание реализации**

При обнаружении ошибки в ходе синтаксического анализа, программа выдает исключение, содержащее номер строки, на которой программа обнаружила ошибку и тип лексемы.

**Выводы**

В ходе проделанной работы получился синтаксический анализатор, способный проверить соответствие поступающих лексем грамматике. Для решения поставленной проблемы был найден на github вспомогательный скрипт на javascript позволяющий автоматически построить управляющую таблицу на основе входной грамматики, которую можно использовать для реализации алгоритма «перенос-опознание»., Этот скрипт при участии специалиста по javascript был переписан на c++ что позволило повысить производительность и интегрировать ранее написанный лексический анализатор.

Исходный код можно найти на: https://github.com/abubaca4/2020-comp