Project Report

On TALcompiler



University Institute of Engineering and Technology

Chhatrapati Shahu Ji Maharaj University, Kanpur

|  |
| --- |
| **Submitted To:** |
| Dr. Alok Kumar |
| CSE Department |
| UIET CSJMU Kanpur |



|  |
| --- |
| **Submitted By:** |
| Abhishek Singh  Roll No.-CSJMA16001390001  Anish Srivastava  Roll No.-CSJMA16001390008  Ayush Chaurasia  Roll No.-CSJMA16001390016 |
|  |
|  |



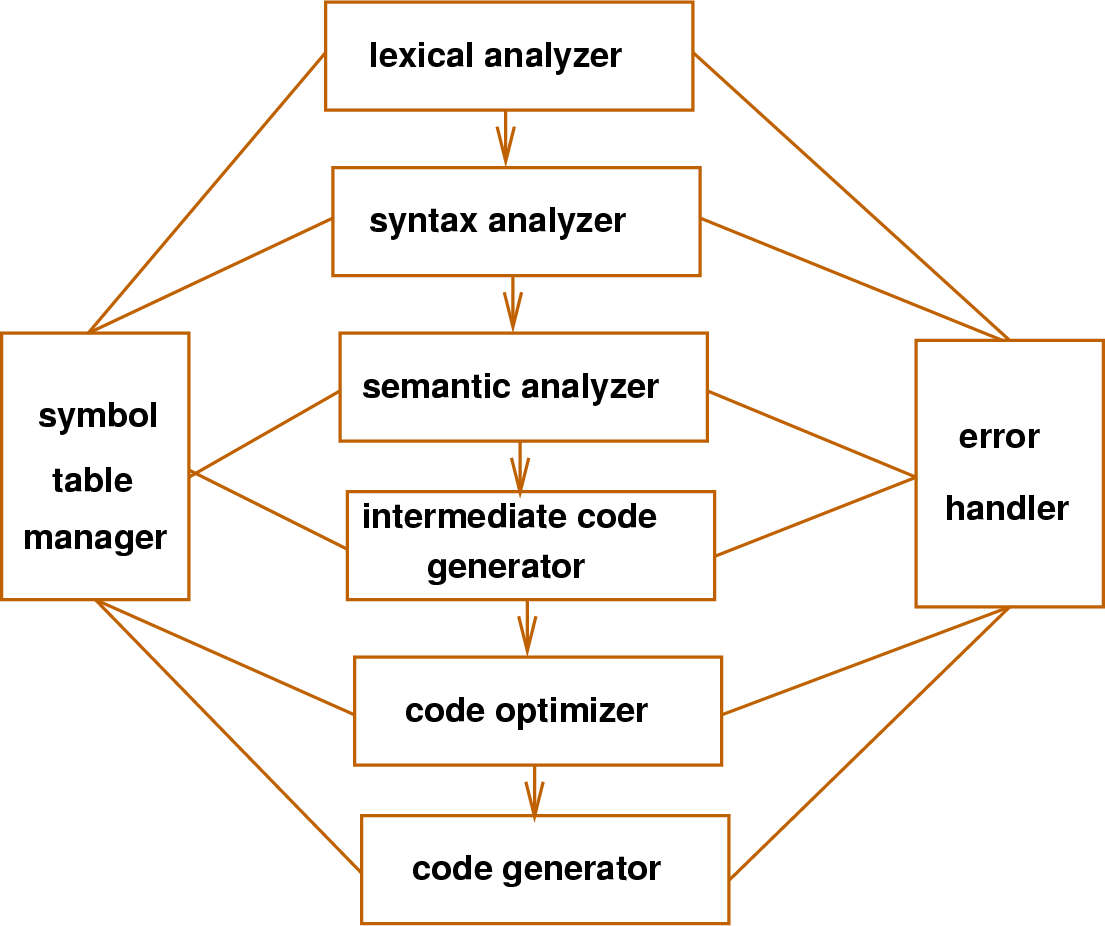
Table of Contents:

1. Introduction
2. Objective.
3. Features of TALcompiler.
4. Some Rules of TA language.
5. Implementation.
6. Execution of Sample Programs

Introduction

*“A compiler is a computer program that translates computer code written in one programming language (the source language) into another programming language (the target language).”*

# Architecture of Compiler:



Objective

* To build a compiler software for a Hypothetical Language which is **Triple A Language(TAL)** in our case.

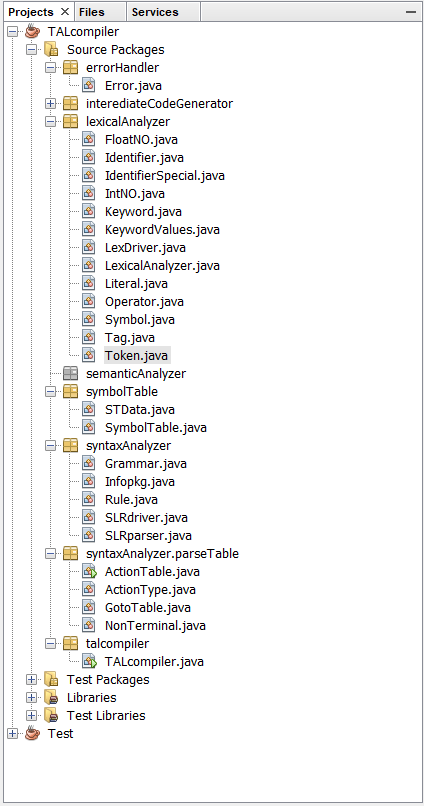
Features Of TALcompiler

* A program file with “.tal” extension should be given as input.
* Can detect **Lexical Errors** i.e. invalid identifiers, invalid characters, invalid symbols etc.
* Can detect syntax error in programs.
* Generates **tagtrace.log** file containing **Token values and tags** during Lexical Analysis.
* Generate **parselog.log** file containing all information about Syntax Analysis of entered program.
* Generates corresponding **Error Messages** if there is an error in input program.

Some Rules of Our Triple A Language

* Program should start with a pre-defined function named exe().
* Operations supported by the language are
  + Declaration .
  + Mathematical Expressions.
  + show() function.
  + showLine() function.

Implementation



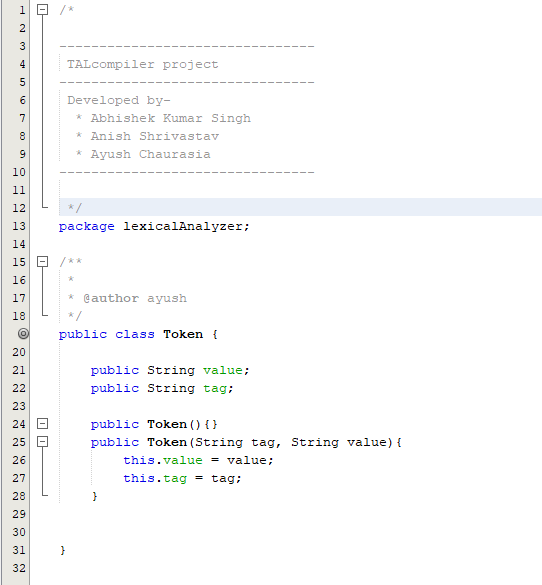
*Image : file hierarchy of TALcompiler*

# Lexical Analyzer

## Token class: Parent class of the following classes-

* FloatNO.java
* IntNo.java
* Identifier.java
* IdentifierSpecial.java
* Keyword.java
* Literal.java
* Operator.java
* Sysmbol.java

## Code:



## Tag class: Contains Tag values for all types of tokens.



## LexicalAnalyzer class: This class contains all the functionality provided by the Lexical Analyzer.

## Syntax Analyzer Grammar and Parse table

# The Grammar:

# 

**NON-TERMINALS**

S: start symbol

B: block of code

Y: generates desired frequency of **group of code**

G : group of code,

D : declaration.

E : equations (grammar from E to F)

P : print.

A : argument of print statement.

R : generates **Return statements**.

**TERMINALS**

e : exe

d : datatype

i : identifier

n : integer number.

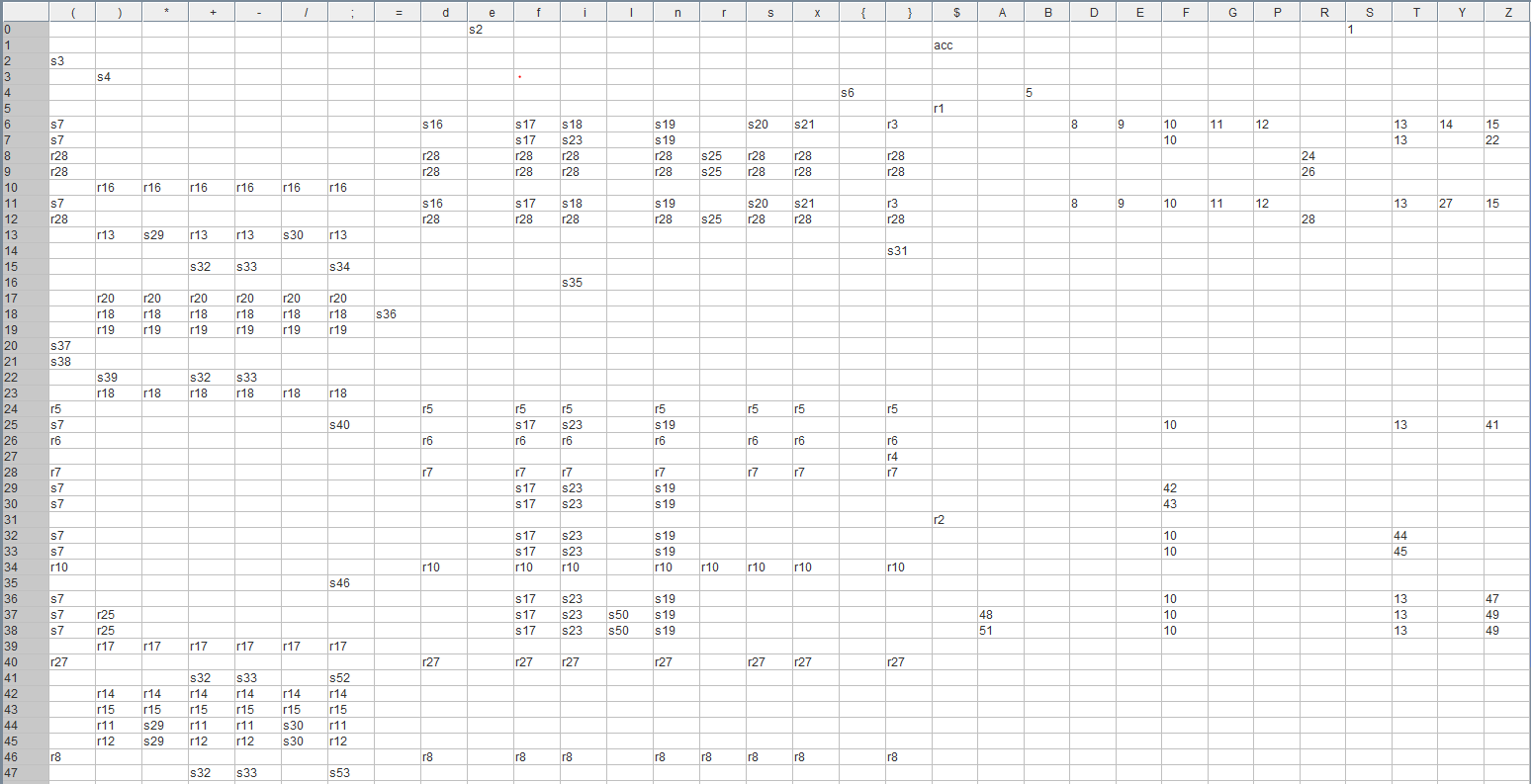
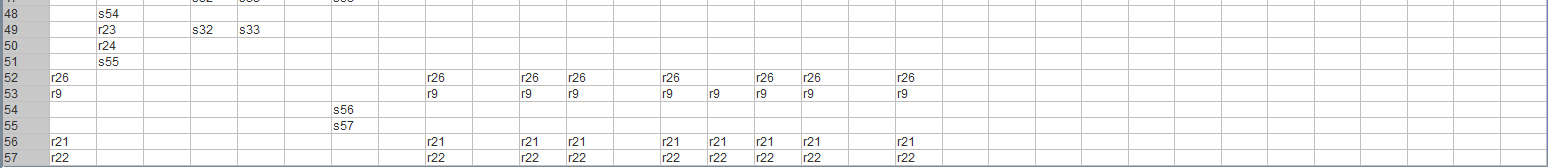
f : float number.

s : show (print function in our grammar)

x : showLine ( same as above)

r : means **reply (**that means **return)**

# Parse Table

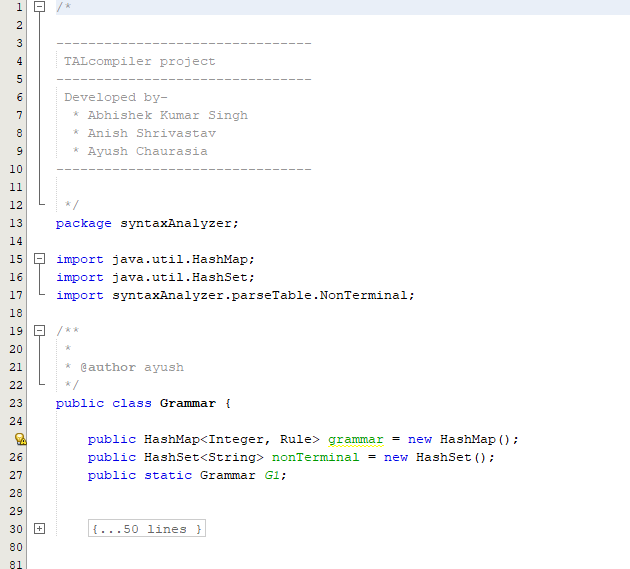


We have used SLR(1) parser in our project. Some classes that are used are described below.

ActionType class: Contains information about different types of action performed in SLR(1) parser.

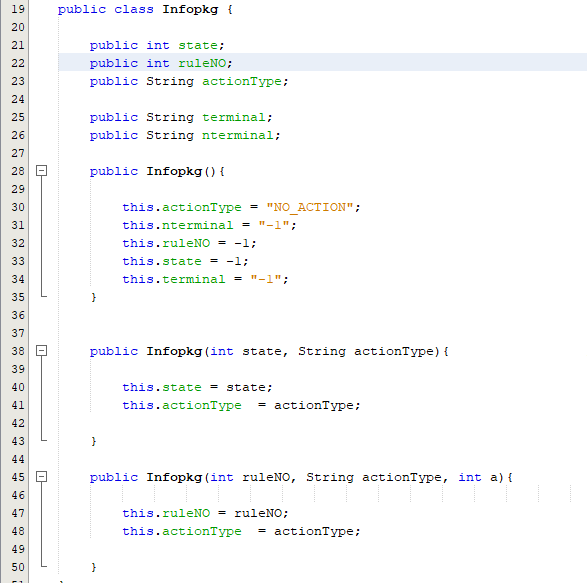


## Grammar class: Class for Grammar of our Triple A Language.

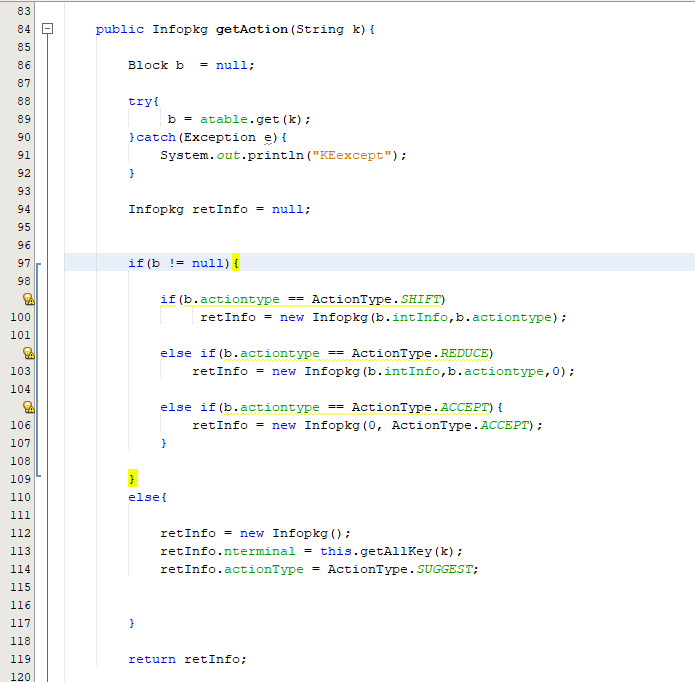


*Image: code segment of Grammar Class*

InfoPackage class: Data structure for sending and receiving information from Action and Goto Table.



ActionTable class: contains information and functionality of Action table.



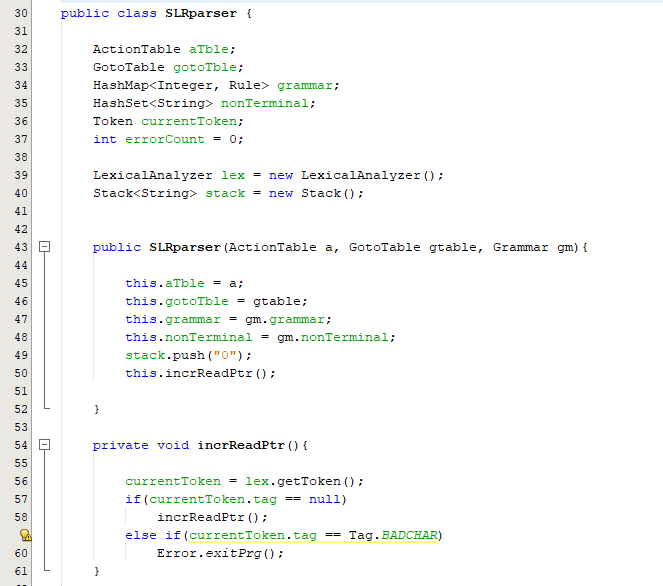
*Image: getAction method from ActionTable*

GotoTable class: contains information and functionality of Goto Table.



*Image: code of GotoTable*

SLRparser class: Contains all the functionality of SLR parser.

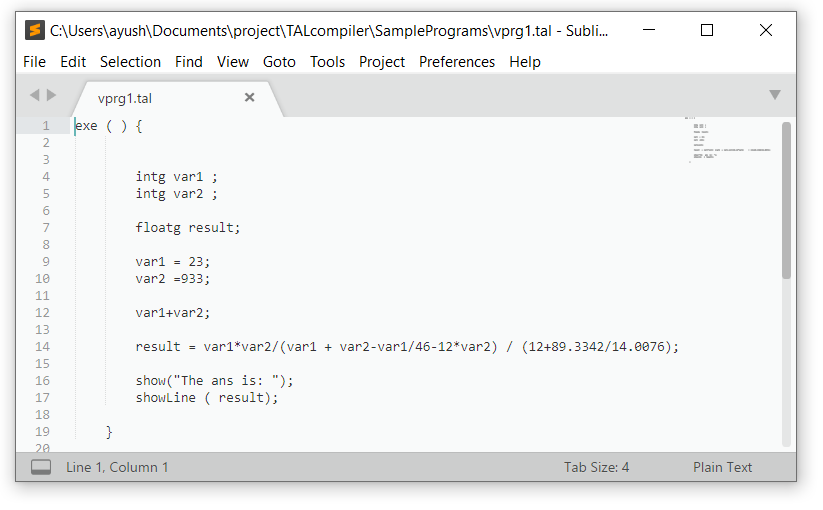


Execution

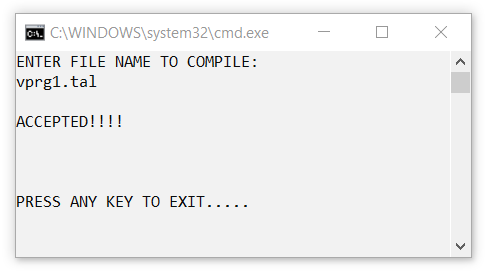
Execution of some sample program is shown below:

# **vprg1.tal**

Input:

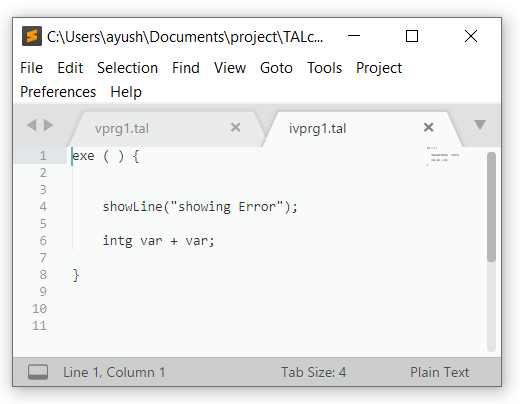


Output:

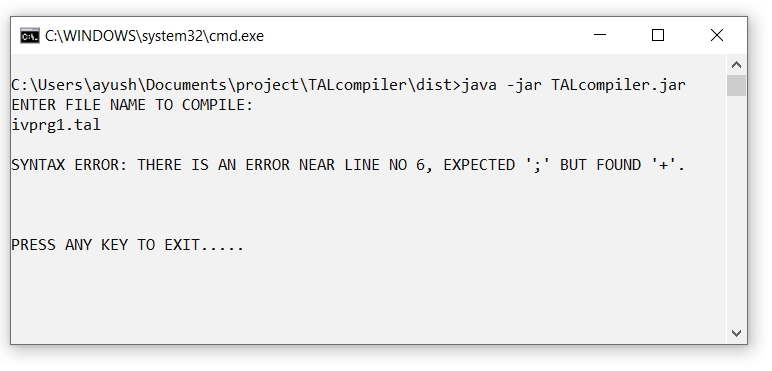


# **Ivprg1.tal**

Input:

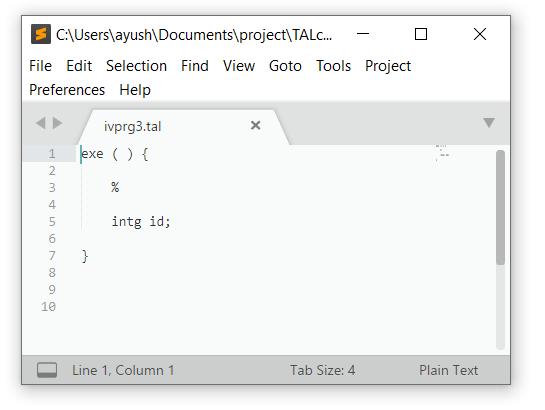


Output:

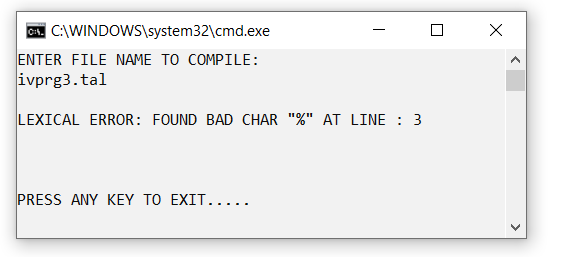


# **Invprg3.tal**

Input:



Output:



## Syntax Analyzer Grammar and Parse table

# The Grammar:

# 

**NON-TERMINALS**

S: start symbol

B: block of code

Y: generates desired frequency of **group of code**

G : group of code,

D : declaration.

E : equations (grammar from E to F)

P : print.

A : argument of print statement.

R : generates **Return statements**.

**TERMINALS**

e : exe

d : datatype

i : identifier

n : integer number.

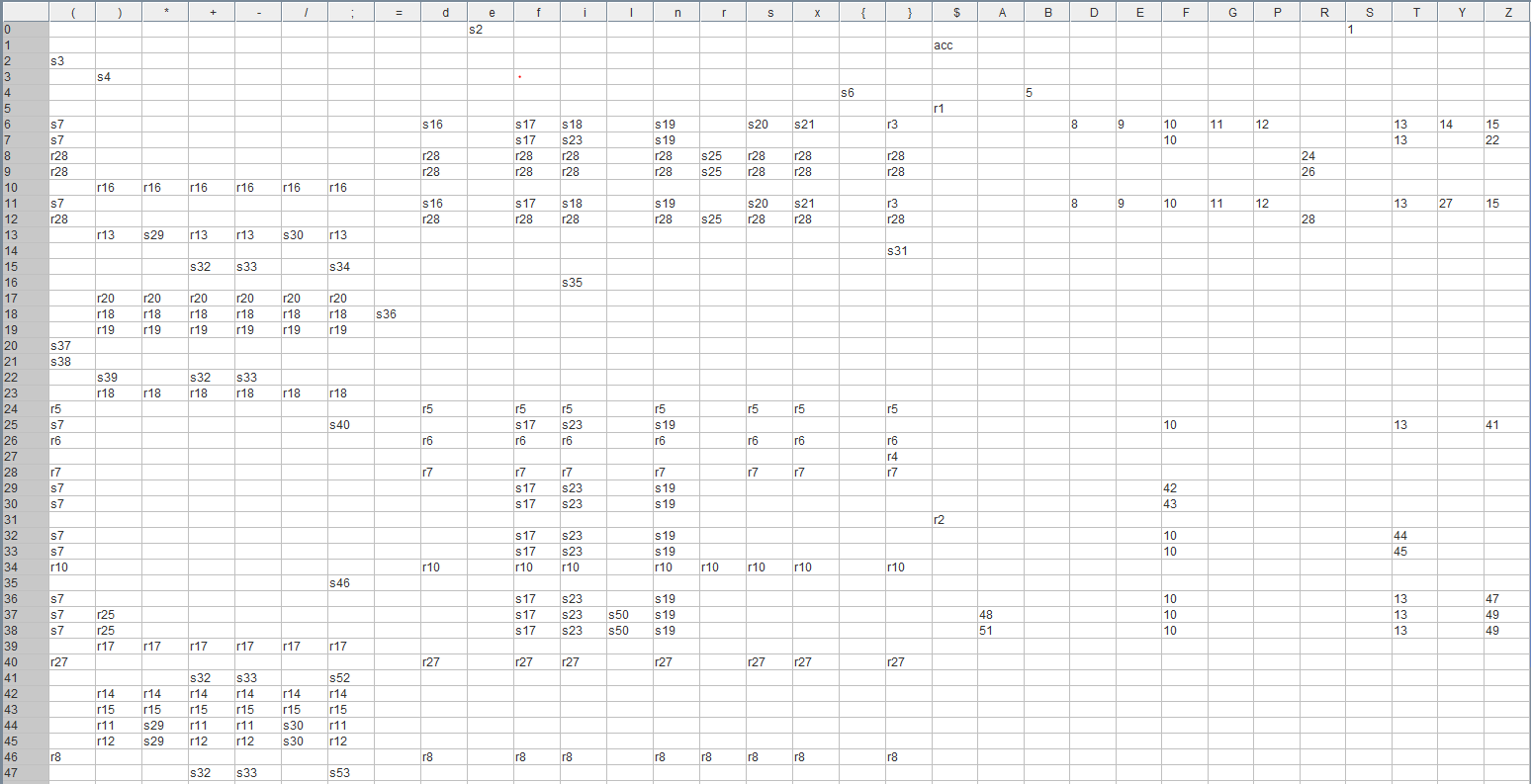
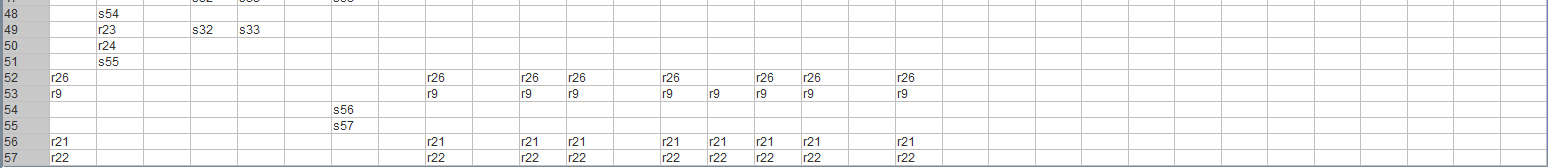
f : float number.

s : show (print function in our grammar)

x : showLine ( same as above)

r : means **reply (**that means **return)**

# Parse Table



# 