LL Parse Table Generator

A Project

Report

Submitted in the partial fulfillment of the

requirements for the award of the degree of

Bachelor of Technology in

Department of Computer Science and Engineering

By

Manichand (2010030455)

Sreevarun (2010030451)

Jashwanth reddy (2010039002)

Anil (2010030015)

Avinash(2010030358)

UNDER THE SUPERVISION OF

Dr.Sumith Hazra

Department of Computer Science and Engineering

K L University Hyderabad,

Aziz Nagar, Moinabad Road, Hyderabad – 500 075, Telangana, India.

March 2023-2024

Declaration

The Compiler Design Report entitled "**LL Parse Table Generator**" is a record of bonafide work of Manichand (2010030455), Sreevarun(20100030451), Jashwanth (2010039002),Anil(2010030015),Avinash(2010030358) submitted in partial fulfillment for the award of B.Tech in the Department of Computer Science and Engineering to the KL University, Hyderabad. The results embodied in this report have not been copied from any other Departments/ University/ Institute.

<Signature of the Students>

Certificate

This is to certify that the Compiler Design Report entitled “LL Parse Table Generator” is beingManichand(2010030455),Sreevarun(20100030451),Jashwanth(2010039002),Anil(2010030015),Avinash(2010030358) submitted in partial fulfillment for the award of B.Tech in CSE to the K L University, Hyderabad is a record of bonafide work carried out under our guidance and supervision.

The results embodied in this report have not been copied from any other departments/ University/Institute.

Signature of the Supervisor

Signature of the HOD **Signature of the External Examiner**

ACKNOWLEDGEMENT

First and foremost, we thank the lord almighty for all his grace & mercy showered upon us, for completing this project successfully.

We take grateful opportunity to thank our beloved Founder and Chairman who has given constant encouragement during our course and motivated us to do this project. We are grateful to our Principal **Dr. RAMAKRISHNA AKELLA** who has been constantly bearing the torch for all the curricular activities undertaken by us.

We pay our grateful acknowledgement & sincere thanks to our Head of the Department

Dr. ARPITA for his exemplary guidance, monitoring, and constant encouragement throughout the course of the project. We thank **Dr.Sumith Hazra** of our department who has supported throughout this project holding a position of supervisor.

We wholeheartedly thank all the teaching and non-teaching staff of our department without whom we wouldn't have made this project a reality. We would like to extend our sincere thanks especially to our parents, our family members and friends who have supported us to make this project a grand success.

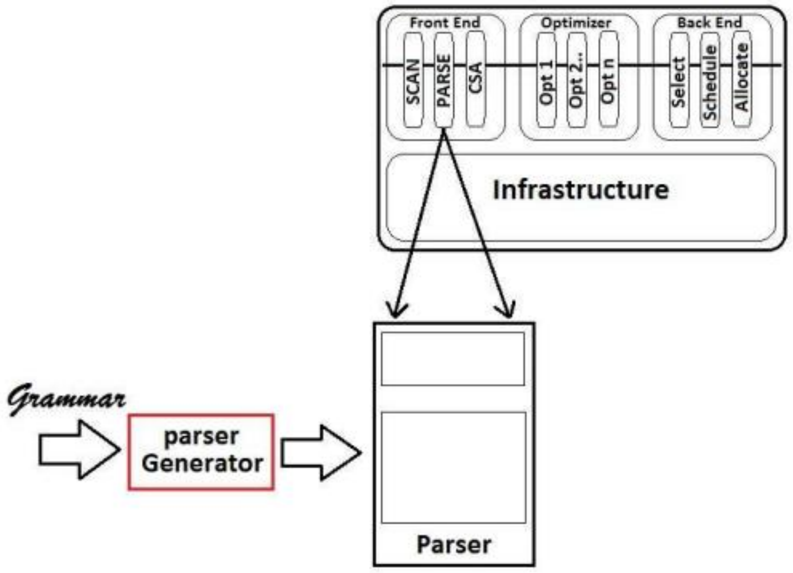
TABLE CONTENTS

ABSTRACT

Parser Generator is a tool that automate construction of tables for a given grammar

Parser Generator consumes the grammar and produces a pair of tables that drive an LR(1) parser. The tables encode all grammatical knowledge needed for parsing.

Parser Generator application is being implanted in cpp and provides facility to compute Terminal, Non terminal, First, Follow and LR items, and tables.



This application is consisted of following modules:

- Read Grammar from input text file
- Compute terminals and non terminal
- Compute First and Follow
- Compute item sets
- Construct Action and Goto table

INTRODUCTION

- LL parsing is a type of top-down parsing that uses a left-to-right scan of the input and a leftmost derivation.
- A parsing table is used in LL parsing to determine which production rule to apply at each step of the parse.

Department of Computer Application

Computer Applications is one of the thrust areas in science and technology. In appreciation of its

growing importance in business and visualising the career prospects, the University established

the Department of Computer Applications to facilitate research and human resource development

in the subject. The M.C.A. programme was started (1994) with a model curriculum prepared

jointly by ISTE and the Department of Electronics with minor modifications. The Syllabus is

updated periodically, based on the current trends and requirements of the industry. The

Department has a proud alumni, most of them being placed in much reputed international firms

like IBM, WIPRO, INFOSYS, TCS, CTS etc. The Research activities of the Department include

the subject areas Fuzzy sets and Logic Applications in Artificial Intelligence, Simulation,

Cryptography & Coding Theory, Algorithm, Pattern recognition, Internet-Marketing, Ecommerce and Internet Technology, Networking and Mobile Communication and Software

Engineering. Apart from this, the Department has taken up a challenging research project funded

by AICTE, Computer Assisted Classical Music. The Department of Computer Applications is

also doing consultancy work for public and private sector undertakings. The Department has an

excellent library with more than 3000 books and various national and international journals.The

Department has a well-equipped laboratory, which is being constantly updated with the latest

computers.

Synopsis

Parser Generator is a tool that automate construction of tables for a given

grammar

Parser Generator consumes the grammar and produces a pair of tables that drive an LR(1) parser. The tables encode all grammatical knowledge needed for parsing.

Parser Generator application is being implanted in cpp and provides facility to compute Terminal, Non terminal, First, Follow and LR items.

This application is consisted of following modules:

- Read Grammar from input text file
- Compute terminals and non terminal
- Compute First and Follow
- Compute item sets
- Construct Action and Goto table

SOFTWARE QUALITY ASSURANCE PLAN

a. Purpose

The purpose of Software Quality Assurance Plan is to provide management with appropriate visibility into the process being used by the software project and of the products being built. Software Quality Assurance involves reviewing and auditing the software products and activities to verify that they comply with the applicable procedures and standards and providing the software project and other appropriate managers with the results of these reviews and audits.

b. Reference Documents

- Project Plan document
- IEEE Standard Guide for Software Quality Assurance Plan

c. Management

Organization

The organization consists of supervisory committee, project manager, developer and two formal technical inspectors.

- Supervisory Committee:
- Project manager:
- Developer:
- Formal Technical Inspectors:

Responsibilities

Supervisory Committee

The committee will be responsible for attending the presentations and reviews at the end of each phase. After each presentation, the committee will provide feedback and suggestions regarding the software.

SYSTEM ANALYSIS

a. Proposed System

The proposed system is a application based tool, which will be generate a pair of tables. This proposed system consumes a grammar and generates Action table and GoTo table that drive the LR(1) parser. The table encode all of the grammatical knowledge needed for parsing. This proposed system precompiles into the tables all the knowledge required to identify handles, to decide when to shift and decide when to reduce and which rule to use in reduction.

LR(1) table construction is an elegant application of theory to practice the construction systematically builds up a handle recognizing DFA and then translate that model into pair of tables

b. Existing System

In existing system the parser may have selected the wrong production at some earlier step in the

process, in which backtrack lead it to correct choice.

Exiting system not suitable for left recursive production, this lead to a infinite sequence of expansions, each of which makes no progress

A production is left recursive if first symbol of right hand side is the same as the symbol of right hand side.

Grammars that exhibit left recursion can cause a existing system to expand forever while making

no progress

Exiting system is not suitable for those grammar which need backtrack.

Complications in Existing system

Eliminating left recursion

Fee -> Fee α

Fee -> β

After elimination of left recursion

Fee -> β Fie

Fie -> α Fie

Fie -> €

Eliminating to need to backtrack

c. Modular Description

This application is being developed for generating parser tables . This application is comprised of the following four modules:

- Read Grammar from input text file
- Compute First and Follow
- Compute item sets
- Construct Action and Goto Table

MODULES

READING MODULE:-

In this module we shall read LR grammar which is given in a Text file and

distinguish left side symbol ,right side symbol , Terminal symbol and Non-Terminal symbol and

store different symbol in different link list. After that we generate and set unique id for each

symbols of grammar.

1. User put a Grammar into file

2. GetLine() : Read a line from file and store it in a string

3. Getword () :Read a word from line store it another string

4. If word is " ->" then set ls variable zero and do nothing

5. If word is from left side and it is not inserted into variable list then insert this

word into variable list

6. If this word is already inserted into the terminal list then delate this word

from Terminal list.

7. SetId()

Generate and set id for each symbol in variable list and terminal list

Initialize Grammar Module:-

In this module we shall find id of each Terminal and Non-Terminal symbol of grammar and create production of this Id , Store each production in Grammar list.

1. GetId for each production stored in production list.
2. Createproduction() : create production of each rule given in file with it corresponding id .
3. Insert production into the grammar list.

Compute FIRST and FOLLOW Module :

In this module we will find FIRST and FOLLOW of each Terminal and Non-Terminal symbol of a Grammar.

FIRST is defined over T U NT. For a symbol define FIRST(a) as the set of words that can appear as first symbol in some string derive from 'a'

Each production of Grammar has a unique First set.

FOLLOW is defined for only NON- TERMINAL. FOLLOW(A) be the symbols that can occur immediately after some NON- TERMINAL 'A' in the valid sentence.

COMPUTE ITEM SETS :

In this module we will find the canonical collections of sets of items for a augmented Grammar .

An item of a Grammar G is a production of G with a dot at some position of the right side. An item indicates how much of a production we have seen at a given point in a parsing process.

To construct collection of items for a grammar we define a augmented grammar and two function closure and goto.

CONSTRUCTING LR PARSING TABLE:

In this module we shall show to construct LR parsing action and goto from deterministic finite automaton that recognizes viable prefixes.

Given a grammar G we augment G to produce G′, and from G′ we construct C, the canonical collection of sets of items for G′. we construct action , the parsing action function and goto , the goto function , from c using its algorithm. It requires us to know FOLLOW(A) for each NonTerminal A of a Grammar

REQUIREMENTS ANALYSIS:

a. Problem Definition

Requirement analysis is the process of defining out what the user requires from the system and defining the requirements clearly and in an unambiguous state. The outcome of the requirement analysis is the software developing activities. Thus it deals with understanding the problem goals and constraints. A requirement is a clearly short and concise piece of information, expressed as a fact. It can be written as a sentence or can be expressed using some kind of diagram.

Parsing is a important phase in compilation process, and after scanning the grammar parsing is to be needed for which parser generator is needed. Parser generator is essential in compiler and for that optimize table generator needed.

b. Functional Requirements

Functional requirements describe what the system should do. The functional requirements can be further categorized as follows.

- What inputs the system should accept
- The ability to make parser table for given grammar
- Exactly compute the LR items of the table
- What outputs the system should produce
- What data the system must store
- What are the computations to be done
- The timing and Synchronization of above

c. Non Functional Requirements

Non-Functional requirements are the constraints that must be adhered during development. They limit what resources can be used and set bounds on aspects of the software's quality. The following are some of the non-functional requirements.

- Response Time
- Throughput
- Recovery from failure
- Reusability
- Platform

d. Development Environment

The following requirements apply to the PARSER GENERATOR

ALGORITHMS

ALGORITHM 1: [Read Input Grammar from FIile]

```
STEP 1: START
STEP 2: For: each line in the File
 STEP 3: Set ls<- 1
STEP 4: Repeat
STEP 3: Get a word from a line
STEP 4: If word is “->”
STEP 5: then reset ls <- 0 and continue.
STEP 6: If :ls = 1
STEP 7: Insert word in PROD_Left list
STEP 8: If : word is not Exits in the non
 terminal list
STEP 9: Then insert word into non-terminal list
STEP 10: If :word Exits in the terminal list
STEP 11: Then delete word from terminal list
STEP 12: Else
STEP 13: If : word is not Exits in the non-terminal list and
word is not Exits in the the terminal list
STEP 14: Then insert word into non-terminal list
STEP 15: While : there is word in line
STEP 16: Insert right side string into PROD_Right side list
STEP 17: END for
```

STEP 18: generate and set id for each terminal and non terminal symbol

STEP 18: STOP

Explanation:

Lets us take an example

s Exp

Exp exp + term

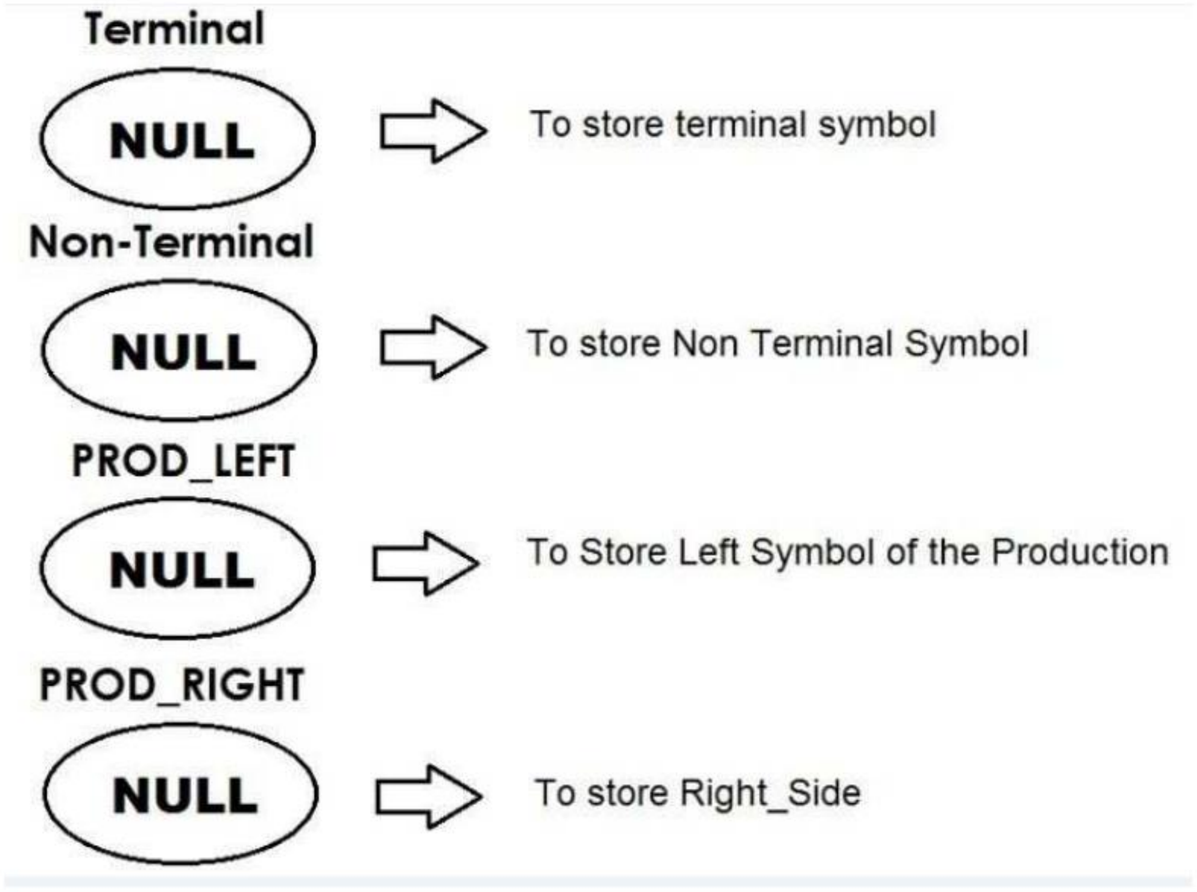
Exp term

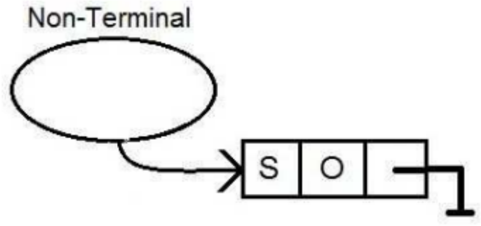
term term X factor

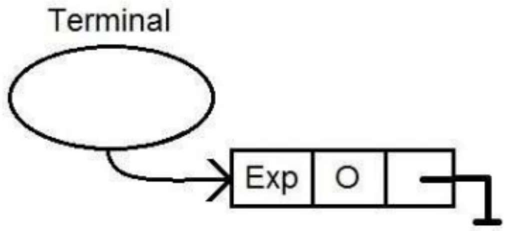
term factor

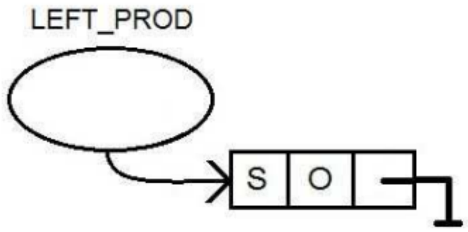
factor (Exp)

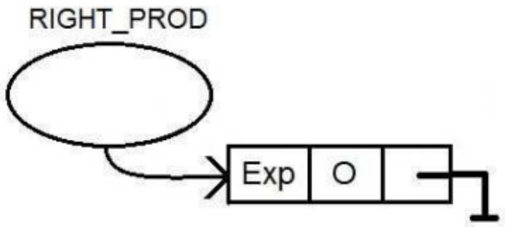
factor index


Terminal
NULL
To store terminal symbol
Non-Terminal
NULL
To store Non Terminal Symbol
PROD_LEFT
NULL
To Store Left Symbol of the Production
PROD_RIGHT
NULL
To store Right_Side

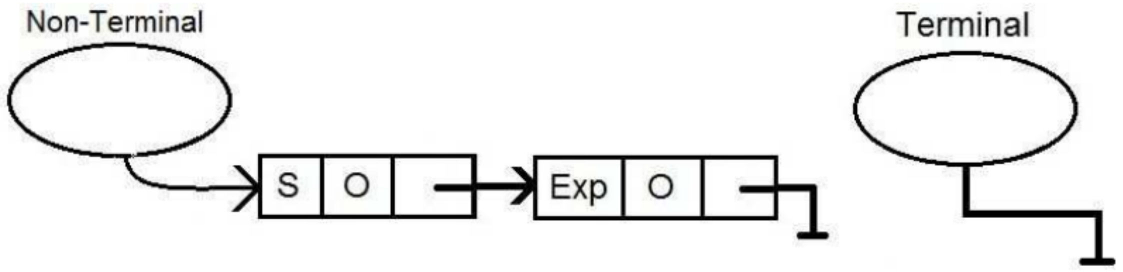




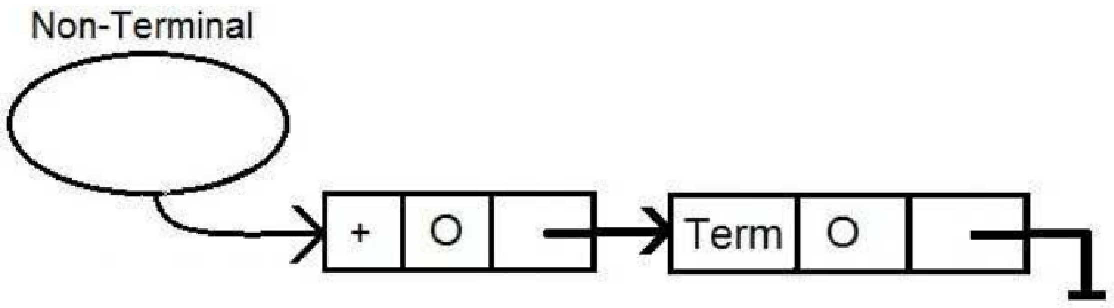


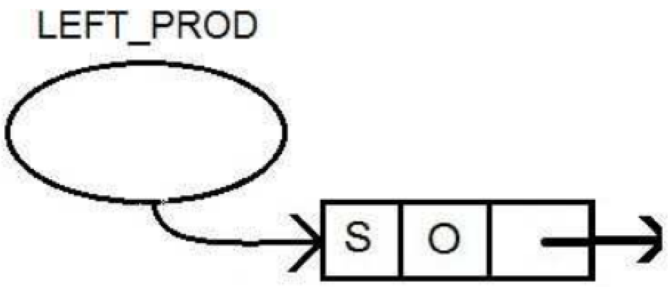


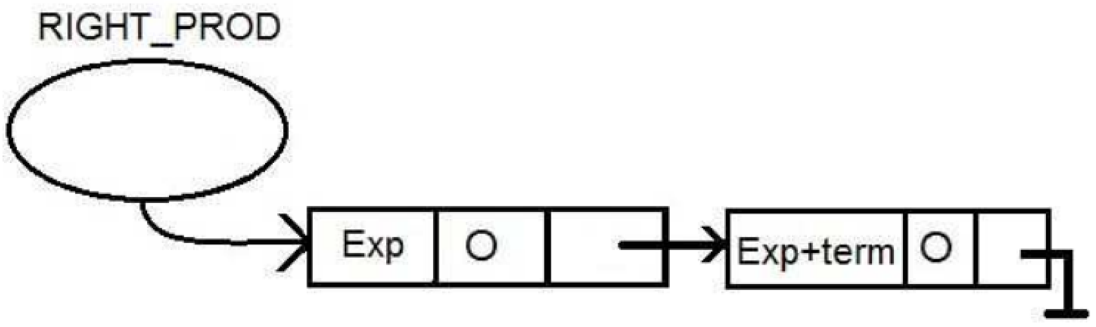
After Reading second line :



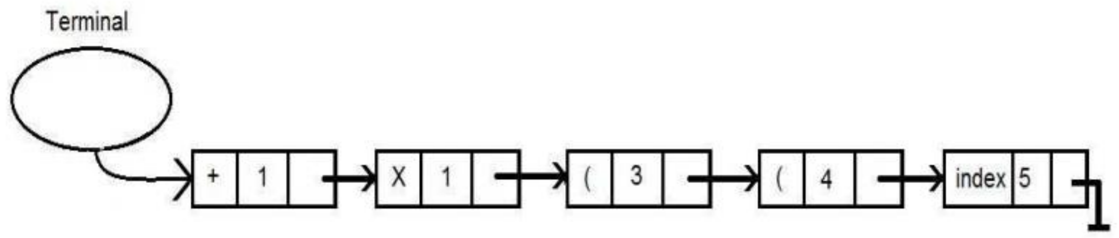
Since exp is already inserted into terminal list, it should be deleted from Terminal list.



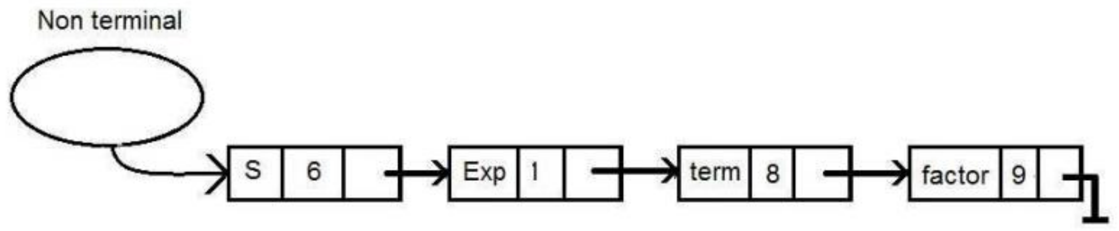




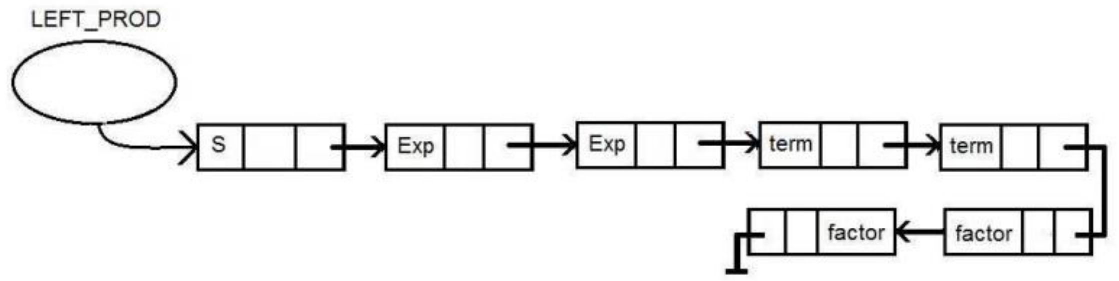
After insertion of each all production and setting id of each symbol of grammar the diagram is as follows:



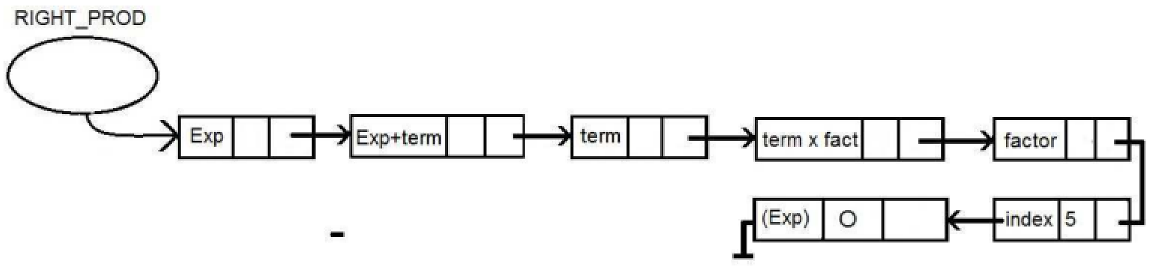
Terminal
+ 1
X 1
(3
(4
index 5



Non terminal
S 6
Exp 1
term 8
factor 9



LEFT_PROD
S
Exp
Exp
term
term
factor
factor



RIGHT_PROD
Exp
Exp+term
term
term x fact
factor
index 5
(Exp) O

ALGORITHM 2: [Compute First]

STEP 1: START

STEP 2: If X is a terminal, then FIRST(X) = { X }

STEP 3: if X -> € is production, then add € to FIRST(X)

STEP 4: If X is a non terminal and X -> YY2 . . Yk is a

production, then for some k >= 1 ,

then place' a 'in FIRST(X) if for some i,' a' is in FIRST(Yi),

and € is in all o FIRST(Y1.) ., . , FIRST(Yi-1); If € is in FIRST(Yj)

for all j = 1,2, . . . , k, then add € to FIRST(X). For

example, everything in FIRST(Y1) is surely in FIRST(X). If y1 does not

drive € , then we add F1RST(Y2), and SO on.

STEP 4: If X ->€ is a production, then add € to FIRST(X).

STEP 5: STOP

		+	X	(	)	index
+	0	0	0	0	0	0
X	0	0	0	0	0	0
(	0	0	0	0	0	0
)	0	0	0	0	0	0
Index	0	0	0	0	0	0
S	0	0	0	0	0	0
Exp	0	0	0	0	0	0
Term	0	0	0	0	0	0
Factor	0	0	0	0	0	0

a.
b.
c.
d.

		+	X	(	)	index
+	0	1	0	0	0	0
X	0	0	1	0	0	0
(	0	0	0	1	0	0
)	0	0	0	0	1	0
Index	0	0	0	0	0	1
S	0	0	0	1	0	1
Exp	0	0	0	1	0	1
Term	0	0	0	1	0	1
Factor	0	0	0	1	0	1

TESTING

BLACK BOX TESTING:

In a black box, the test item is treated as “black” since its logic unknown: all that is

known is what goes in and what comes out, or the input and output. In black box testing

you try various inputs and examine the resulting outputs, you can learn what the box does

but nothing more about how its conversion is implemented. Black box testing works very

nicely in the testing objects in an object oriented environment.

WHITE BOX TESTING:

White box testing assumes that the specific logic is important and must be tested to guarantee the system’s proper functioning. The main use of the white box testing is in error based testing. It is predict on close examination of procedural detail logical providing test cases that exercise specific sets of conditions and/or loops tests path enough the software. White box testing technique is also called basis path testing. The basis path method enables the test case designer to derive a logical complexity of a procedural design and use this measure as a guide for defining as basis set of execution path.

Scope of Future Applications:

As far as my concern, the future of this project is full of brightness. In the modern age the

life is going to be faster and faster everyone wants to save Time, Money and Energy. An SLR

parser will typically have more conflict states than an LALR parser. For real-world computer

languages, an SLR parser is usually not very adequate but good as learning tool. A grammar that

has no conflicts reported by an SLR parser generator is an SLR grammar.

The parser can parse a piece of text from beginning to end, but when you

delete or insert text it does the minimal amount of work and minimal amount of changes in the

token stream and syntax tree, instead of just reparsing everything from the beginning. In future

there could be more better way to find the LR items. So computing first and follow is easy.

There are also some chances for the improvement of some functionality which

is the matter of future enhancement

CONCLUSION:

The proposed application "PARSER GENERATOR" has been completed.

The entire process was not meant to make a parser generator to make the parsing work easy for whole compilation of program. After scanning a given grammar it needs to be parsed to generate the object code. This parser generator can be further optimized to make parsing much easier. There is no doubt in one's mind that is room for improvement.

Each and every phase in this project is developed keeping the goals in mind as far as concerned. Every application has its own limitations. Similarly this application has also its limitations, which can be upgraded in future.

Reference

Reference Books:

i. Roger S Pressman, “Software Engineering – A Practitioner’s approach”

McGraw – Hill International Editions, Fifth Edition, 2001

ii. Engineering in compiler

iii. Compilers Principles, techniques and tools-Alfred V.Aho,Monika S

lam,Ravi Sethi

Reference Websites:

- www.google.com
- www.compilerbooks.com
- www.roseindia.com