Name: Varun Khadayate	Subject: Compiler Design
Roll No: A016	Date of Submission: 24th September 2021

Aim
To design shift reduce parser from given grammar. (shift reduce)

Program logic
Shift Reduce parser
attempts for the construction of parse in a similar manner as done in bottom up parsing i.e. the parse tree is constructed from leaves(bottom) to the root(up). A more general form of shift reduce parser is LR parser.
This parser requires some data structures i.e.

- A input buffer for storing the input string.
- A stack for storing and accessing the production rules.

Basic Operations –
Shift
This involves moving of symbols from input buffer onto the stack.

Reduce
If the handle appears on top of the stack, then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.

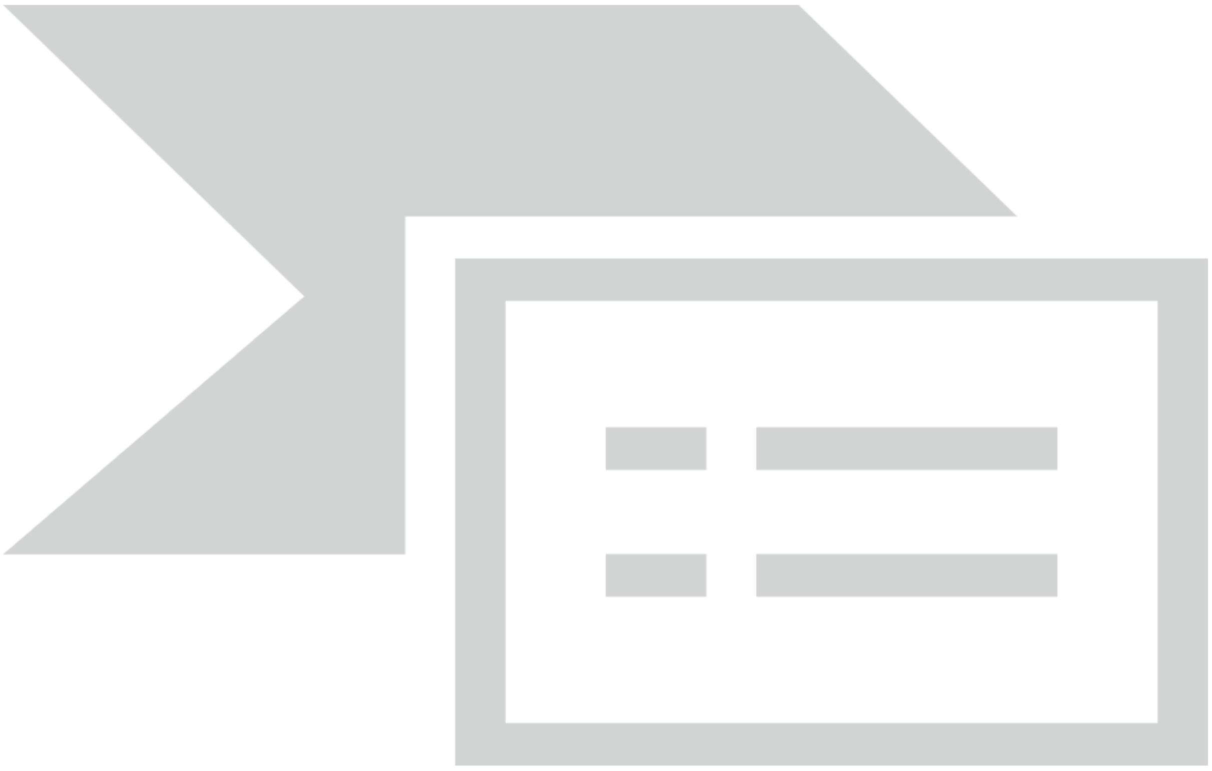
Accept
If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it is means successful parsing is done.

Error
This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

Example 1
Consider the grammar

S –> S + S
S –> S * S
S –> id

Perform Shift Reduce parsing for input string “id + id + id”.



Example 2

Consider the grammar

E –> 2E2
E –> 3E3
E –> 4

Perform Shift Reduce parsing for input string “32423”.

Example 3

Consider the grammar

S –> (L) | a
L –> L , S | S

Perform Shift Reduce parsing for input string “(a , (a , a)) “.

Stack	Input Buffer	Parsing Action
$	(a , (a , a)) $	Shift
$ (	a , (a , a)) $	Shift
$ (a	, (a , a)) $	Reduce S → a
$ (S	, (a , a)) $	Reduce L → S
$ (L	, (a , a)) $	Shift
$ (L ,	(a , a)) $	Shift
$ (L , (	a , a)) $	Shift
$ (L , (a	, a)) $	Reduce S → a
$ (L , (S	, a)) $	Reduce L → S
$ (L , (L	, a)) $	Shift
$ (L , (L ,	a)) $	Shift
$ (L , (L , a	)) $	Reduce S → a
$ (L , (L , S)	)) $	Reduce L →L , S
$ (L , (L	)) $	Shift
$ (L , (L)	) $	Reduce S → (L)
$ (L , S	) $	Reduce L → L , S
$ (L	) $	Shift
$ (L)	$	Reduce S → (L)

Stack	Input Buffer	Parsing Action
$ S	$	Accept

Lab Assignment

What is shift reduce Parser

A shift-reduce parser is a class of efficient, table-driven bottom-up parsing methods for computer languages and other notations formally defined by a grammar. The parsing methods most used for parsing programming languages, LR parsing and its variations, are shift-reduce methods.

What are different types?

There are two main categories of shift reduce parsing as follows:

1. Operator-Precedence Parsing
2. LR-Parser

Specify rules shift reduce parser.

Shift

This involves moving of symbols from input buffer onto the stack.

Reduce

If the handle appears on top of the stack, then, its reduction by using appropriate production rule is done i.e. RHS of production rule is popped out of stack and LHS of production rule is pushed onto the stack.

Accept

If only start symbol is present in the stack and the input buffer is empty then, the parsing action is called accept. When accept action is obtained, it is means successful parsing is done.

Error

This is the situation in which the parser can neither perform shift action nor reduce action and not even accept action.

Define algorithm for shift reduce parser.

1. Get the input expression and store it in the input buffer.
2. Read the data from the input buffer one at the time.
3. Using stack and push & pop operation shift and reduce symbols with respect to production rules available.
4. Continue the process till symbol shift and production rule reduce reaches the start symbol.
5. Display the Stack Implementation table with corresponding Stack actions with input symbols.

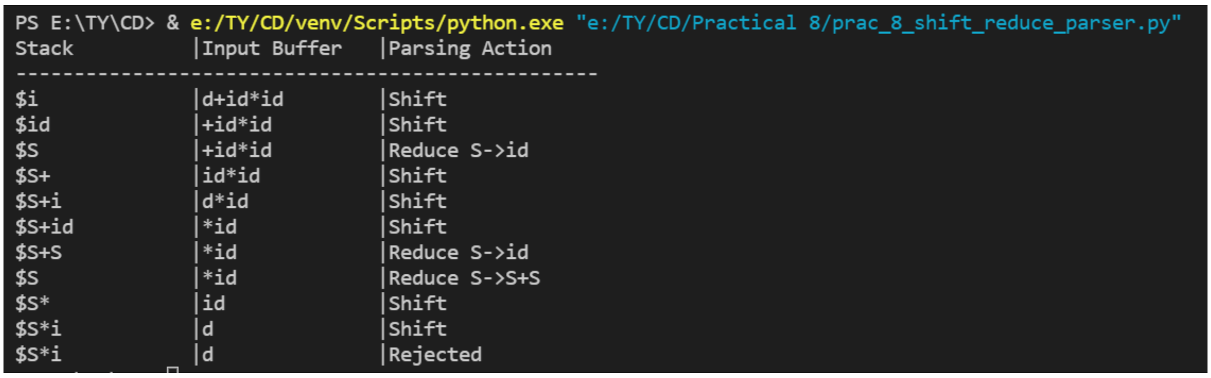
Lab Assignment Program

Write a program to design a shift reduce parser from given grammar. (shift reduce)

Code

```
gram = {
    "S":["S+S","S*S","id"]
}
starting_terminal = "S"
inp = "id+id*id"

stack = "$"
print(f'{"Stack": <15}'+"|"+f'{"Input Buffer": <15}'+"|"+f'Parsing Action')
print(f'{"-":-<50}')

while True:
    action = True
    i = 0
    while i<len(gram[starting_terminal]):
        if gram[starting_terminal][i] in stack:
            stack = stack.replace(gram[starting_terminal][i],starting_terminal
)
            print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Reduce S-
>{gram[starting_terminal][i]}')
            i=-1
            action = False
        i+=1
    if len(inp)>1:
        stack+=inp[0]
        inp=inp[1:]
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Shift')
        action = False

    if inp == "$" and stack == ("$"+starting_terminal):
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Accepted')
        break

    if action:
        print(f'{stack: <15}'+"|"+f'{inp: <15}'+"|"+f'Rejected')
        break
```

Output



```
PS E:\TY\CD> & e:/TY/CD/venv/Scripts/python.exe "e:/TY/CD/Practical 8/prac_8_shift_reduce_parser.py"
Stack          |Input Buffer   |Parsing Action
--------------------------------------------------
$i             |d+id*id        |Shift
$id            |+id*id         |Shift
$S             |+id*id         |Reduce S->id
$S+            |id*id          |Shift
$S+i           |d*id           |Shift
$S+id          |*id            |Shift
$S+S           |*id            |Reduce S->id
$S             |*id            |Reduce S->S+S
$S*            |id             |Shift
$S*i           |d              |Shift
$S*i           |d              |Rejected
```