Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering
Session: 2022-2023

Compiler Design Lab **V Sem AIML**

Name:- Krishna Mundada **Roll:- 45**

PRACTICAL No. 4

Topic: Parsing

Platform: Windows or Linux

Language to be used: Python or Java (based on the companies targeted for placement)

Aim: (A) Write a program to validate a natural language sentence. Design a natural language grammar, compute and input the LL (1) table. Validate if the given sentence is valid or not based on the grammar.

Input: NLP grammar and LL (1) parsing table (from file)
Implementation: String parsing rules
Output: Each step-in string parsing and whether the input string is valid or invalid.

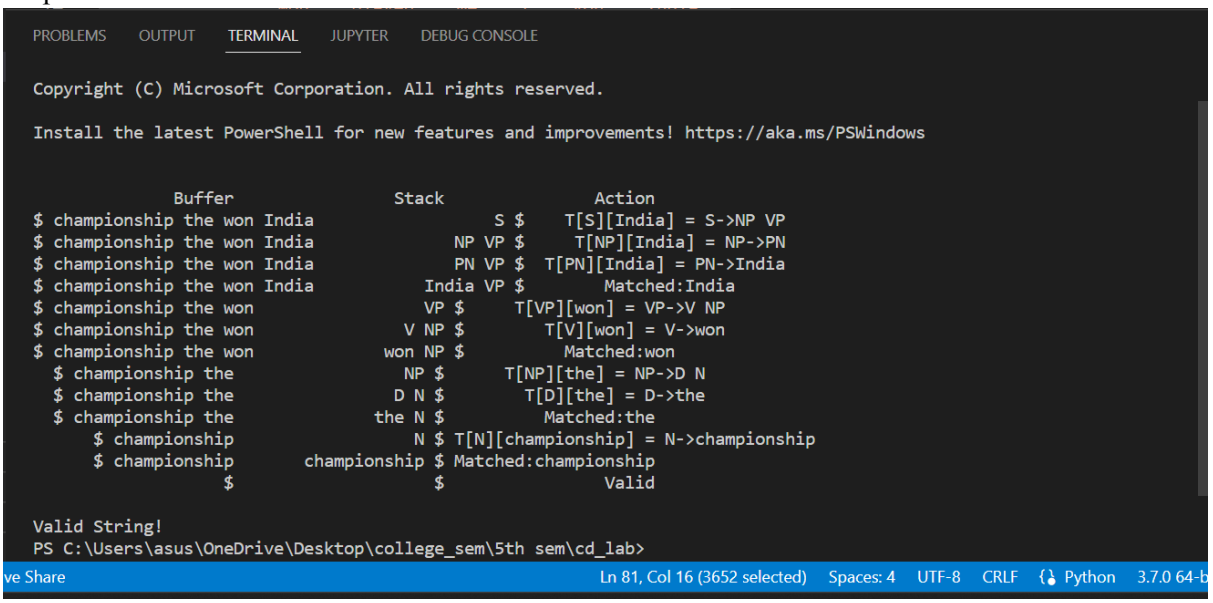
Code⇒

```
table = [["", "", "","S->NP VP", "S-> NP VP", "S->NP VP",
     "S->NP VP", "S->NP VP", "S->NP VP", "S->NP VP", "S->NP VP",
     "S->NP VP", "S->NP VP", "S->NP VP",
     "S->NP VP", "S->NP VP", "S->NP VP"],["", "", "", "",
     "", "", "", "NP->P",
     "NP->P", "NP->P", "NP->PN", "NP->PN",
     "NP->PN", "NP->PN", "NP->D N", "NP->D N", "NP->D N"],
    ["", "", "", "VP->V NP", "VP->V NP", "VP->V NP", "VP->V NP",
     "", "", "", "", "", "", "", "", "", ""],
     ["N->championship", "N->ball", "N->toss", "", "", "", "", "", "",
"", "", "", "", "", "", "", ""],
     ["", "", "", "V->is", "V->want", "V->won", "V->played", "", "",
"", "", "", "", "", "", "", ""],
     ["", "", "", "", "", "", "", "P->me", "P->I", "P->you", "", "",
"", "", "", "", "", ""],
     ["", "", "", "", "", "", "", "", "", "", "PN->India",
"PN->Australia", "PN->Steve", "PN->John", "", "", ""],
     ["", "", "", "", "", "", "", "", "", "", "", "", "", "", "D->the",
"D->a", "D->an"]
     ]



def validate(parsing_table,table_term_list, input_string,term_userdef):

    print(f"\nValidate String => {input_string}\n")

    stack = ['S', '$']
    buffer = []
```

```
    input_string = input_string.split()
    input_string.reverse()
    buffer = ['$'] + input_string

    print("{:>20} {:>20} {:>20}".
          format("Buffer", "Stack","Action"))

    while True:
        # end loop if all symbols matched
        if stack == ['$'] and buffer == ['$']:
            print("{:>20} {:>20} {:>20}"
                  .format(' '.join(buffer),
                          ' '.join(stack),
                          "Valid"))
            return "\nValid String!"
        elif stack[0] not in term_userdef:
            # take font of buffer (y) and tos (x)
            x = list(['S', 'NP', 'VP', 'N', 'V', 'P', 'PN',
'D']).index(stack[0])
            y = table_term_list.index(buffer[-1])
            if parsing_table[x][y] != '':
                # format table entry received
                entry = parsing_table[x][y]
                print("{:>20} {:>20} {:>25}".
                      format(' '.join(buffer),
                             ' '.join(stack),
                             f"T[{stack[0]}][{buffer[-1]}] = {entry}"))
                lhs_rhs = entry.split("->")
                lhs_rhs[1] = lhs_rhs[1].replace('#', '').strip()
                entryrhs = lhs_rhs[1].split()
                stack = entryrhs + stack[1:]
            else:
                return f"\nInvalid String! No rule at " \
                       f"Table[{stack[0]}][{buffer[-1]}]."
        else:
            # stack top is Terminal
            if stack[0] == buffer[-1]:
                print("{:>20} {:>20} {:>20}"
                      .format(' '.join(buffer),
                              ' '.join(stack),
                              f"Matched:{stack[0]}"))
                buffer = buffer[:-1]
                stack = stack[1:]
            else:
                return "\nInvalid String! " \
                       "Unmatched terminal symbols"

nonterm_userdef = ['S', 'NP', 'VP', 'N', 'V', 'P', 'PN', 'D']
term_userdef = ["championship", "ball", "toss", "is", "want",
                "won", "played", "me", "I", "you", "India",
```

```
                "Australia","Steve", "John", "the", "a", "an"]
tabTerm = ["championship", "ball", "toss", "is", "want",
                "won", "played", "me", "I", "you", "India",
                "Australia","Steve", "John", "the", "a", "an", "$"]
sample_input_string = "India won the championship"
validity = validate(table,tabTerm, sample_input_string, term_userdef)
print(validity)
```

Output:-



(B) Use Virtual Lab on LL1 parser to validate the string and verify your string validation using simulation.

Link for Virtual Lab:

http://vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/system_deligators/labs/exp2/index.php

Output: Validation from Virtual lab simulator

IoT Based Fire Alar... Control Lights with... https://www.astagu... (DOC) Fall of Indian... Heart Disease with...

LL(1) Parser Visualization

Write your own context-free grammar and see an LL(1) parser in action!

Written by Zak Kincaid and Shaowei Zhu based on
http://jsmachines.sourceforge.net/machines/ll1.html

1. Write your LL(1) grammar (empty string '' represents ε):

```
S ::= NP VP
NP ::= P
NP ::= PN
NP ::= D N
VP ::= V NP
N ::= championship
N ::= ball
N ::= toss
V ::= is
V ::= want
V ::= won
V ::= played
P ::= me
P ::= I
P ::= you
PN ::= India
PN ::= Australia
PN ::= Steve
PN ::= John
D ::= the
D ::= a
D ::= an
```

Generate tables

Valid LL(1) Grammars

For any production S -> A | B, it must be the case that:

- For no terminal t could A and B derive strings beginning with t
- At most one of A and B can derive the empty string
- if B can derive the empty string, then A does not derive any string beginning with a terminal in Follow(A)

Formatting Instructions

- The non-terminal on the left-hand-side of the first rule is the start non-terminal
- Write each production rule in a separate line (see example to the left)
- Separate each token using whitespace
- $ is reserved as the end-of-input symbol, and S is reserved as an artificial start symbol. The grammar is automatically augmented with the rule S ::= *start* $

Debugging

- More information about the parser construction is printed on the console
- The source code follows the pseudocode in lecture. In particular, see `computeNullable`, `computeFirst`, `computeFollow`, and `computeLL1Tables`

IoT Based Fire Alar... Control Lights with... https://www.astagu... (DOC) Fall of Indian... Heart Disease with...

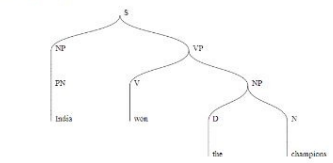
nd Transition Table

First	Follow
u, India, Australia, Steve, John, the, a, an	$
u, India, Australia, Steve, John, the, a, an	is, want, won, played, $
is, want, won, played	$
championship, ball, toss	is, want, won, played, $
is, want, won, played	me, I, you, India, Australia, Steve, John, the, a, an
me, I, you	is, want, won, played, $
India, Australia, Steve, John	is, want, won, played, $
the, a, an	championship, ball, toss

	$	championship	ball	toss	is	want	won	played	me	I	you	India	Australia	Steve	John	the	a	an
S									S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $	S ::= NP VP $
NP												NP ::= PN	NP ::= PN	NP ::= PN	NP ::= PN	NP ::= D N	NP ::= D N	NP ::= D N
VP					VP ::= V NP	VP ::= V NP	VP ::= V NP	VP ::= V NP										
N		N ::= championship	N ::= ball	N ::= toss														
V					V ::= is	V ::= want	V ::= won	V ::= played										
P									P ::= me	P ::= I	P ::= you							
PN												PN ::= India	PN ::= Australia	PN ::= Steve	PN ::= John			
D																D ::= the	D ::= a	D ::= an

n the championship

Partial Parse Tree



IoT Based Fire Alar... Control Lights with... https://www.astagu... (DOC) Fall of Indian... Heart Disease with...

```
PN ::= John
D ::= the
D ::= a
D ::= an
```

Generate tables

- More information about the parser construction is printed on the console
- The source code follows the pseudocode in lecture. In particular, see `computeNullable`, `computeFirst`, `computeFollow`, and `computeLL1Tables`

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	✕	me, I, you, India, Australia, Steve, John, the, a, an	$
NP	✕	me, I, you, India, Australia, Steve, John, the, a, an	is, want, won, played, $
VP	✕	is, want, won, played	$
N	✕	championship, ball, toss	is, want, won, played, $
V	✕	is, want, won, played	me, I, you, India, Australia, Steve, John, the, a, an
P	✕	me, I, you	is, want, won, played, $
PN	✕	India, Australia, Steve, John	is, want, won, played, $
D	✕	the, a, an	championship, ball, toss

	$	championship	ball	toss	is	want	won	
S								
NP								
VP					VP ::= V NP	VP ::= V NP	VP ::= V NP	VI
N		N ::= championship	N ::= ball	N ::= toss				
V					V ::= is	V ::= want	V ::= won	V
P								
PN								
D								

3. Parsing

Token stream separated by spaces: India won the championship

Start/Reset Step Forward

Stack