Name: Diksha Gupta
Roll No. 27

PRACTICAL NO. 04

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.

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

Link for Virtual Lab:

Link 1: http://vlabs.iitb.ac.in/vlabs-dev/vlab_bootcamp/bootcamp/system_deligators/labs/exp2/index.php

Link 2: https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/

Output: Validation from Virtual lab simulator

Details:

PART A:

- Construct and consider a natural language grammar that can validate an English sentence.
- Solve the NLP grammar by hand for LL (1) parser and create parsing table
- Input the above parsing table and grammar using a file.
- Write program for performing string validation

PART B:

- Go to Virtual lab: Go through all the tabs, paste screen shots for all steps (including tests), validate your string parsing with the simulator (screen shot expected).

PART A:

CODE :

```
# hard coding the parsing table
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} {:>40}".format("Buffer", "Stack","Action"))
    while True:
      # end loop if all symbols matched
      if stack == ['$'] and buffer == ['$']:
           print("{:>20} {:>20} {:>50}".format(' '.join(buffer),'
'.join(stack),"Valid"))
           return "\nValid String!"
      elif stack[0] not in term_userdef:
```

```
            # take front 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} {:>50}".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} {:>50}"
              .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 :

```
Validate String => India won the championship

            Buffer              Stack                          Action
$ championship the won India                S $                  T[S][India] = S->NP VP
$ championship the won India             NP VP $                 T[NP][India] = NP->PN
$ championship the won India             PN VP $                T[PN][India] = PN->India
$ championship the won India          India VP $                           Matched:India
$ championship the won                  VP $                T[VP][won] = VP->V NP
$ championship the won                V NP $                   T[V][won] = V->won
$ championship the won              won NP $                           Matched:won
  $ championship the                  NP $                T[NP][the] = NP->D N
  $ championship the                D N $                  T[D][the] = D->the
  $ championship the              the N $                           Matched:the
      $ championship                  N $          T[N][championship] = N->championship
      $ championship       championship $                Matched:championship
                  $                   $                           Valid

Valid String!
```

PART B:

Go to Virtual lab: Go through all the tabs, paste screen shots for all steps (including tests), validate your string parsing with the simulator (screen shot expected).

2. Nullable/First/Follow Table and Transition Table

Nonterminal	Nullable?	First	Follow
S	×	(, id	
E	×	(, id	), $
E'	✓	+	), $
T	×	(, id	+,), $
T'	✓	*	+,), $
F	×	(, id	+, *,), $

	$	+	*	(	)	id
S				S ::= E $		S ::= E $
E				E ::= T E'		E ::= T E'
E'	E' ::= ε	E' ::= + T E'			E' ::= ε	
T				T ::= F T'		T ::= F T'
T'	T' ::= ε	T' ::= ε	T' ::= * F T'		T' ::= ε	
F				F ::= (E)		F ::= id

3. Parsing

Token stream separated by spaces: id + id

Start/Reset | Step Forward

Stack

Remaining Input

Rule

Match $

Partial Parse Tree