

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

Compiler Design Lab

V Sem AIML

**PRACTICAL No. 4**

**Topic:** Parsing**Platform:** Windows or Linux**Language to be used:** Python**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.**

**Practical - 4**

```
In [1]: def table(row,column):
        if row == 'S':
            if column in ["is","want","won","played","me","I","you","India","Australia","Ste
                return ["VP","NP"]
            else:
                return 0
        elif row == "NP":
            if column in ["me","I","you"]:
                return ["P"]
            elif column in ["India","Australia","Steve","John"]:
                return ["PN"]
            elif column in ["the","a","an"]:
                return ["N","D"]
            else:
                return 0
        elif row == "VP":
            if column in ["is","want","won","played"]:
                return ["NP","V"]
            else:
                return 0
        elif row == "N":
            if column in ["championship","ball","toss"]:
                return [column]
            else:
                return 0
        elif row == "V":
            if column in ["is","want","won","played"]:
                return [column]
            else:
                return 0
        elif row == "P":
            if column in ["me","I","you"]:
                return [column]
            else:
                return 0
        elif row == "PN":
            if column in ["India","Australia","Steve","John"]:
                return [column]
            else:
                return 0
        elif row == "D":
            if column in ["the","a","an"]:
                return [column]
            else:
                return 0
```

```

print("Please enter the string: ", end = "")
string = input()
string = list(string.split(" "))
stack = ["$", "S"]
if len(string) == 0:
    print("Invalid string")
else:
    while len(stack) != 0:
        print("stack : ", stack, "buffer : ", string)
        stc = stack[-1]
        if stc == "$":
            print("Valid string")
            break

```

```

terminals = "championship ball toss is want won played me I you India Australia"
terminals = list(terminals.split(" "))
nonTerminals = "S NP VP N V P PN D"
nonTerminals = list(nonTerminals.split(" "))
stc = stack[-1]
buff = string[0]
stack.remove(stc)
if stc in nonTerminals:
    print(f"{stc} is in nonTerminals")
    result = table(stc, buff)
    if result == 0:
        print("Invalid string")
        break
    else:
        for i in result:
            stack.append(i)
else:
    if buff == stc:
        print(f"buff = {buff}")
        string.remove(buff)
    else:
        print("Invalid string")
        break

```

```

Please enter the string: India won the championship
stack : ['$', 'S'] buffer : ['India', 'won', 'the', 'championship']
S is in nonTerminals
stack : ['$', 'VP', 'NP'] buffer : ['India', 'won', 'the', 'championship']
NP is in nonTerminals
stack : ['$', 'VP', 'PN'] buffer : ['India', 'won', 'the', 'championship']
PN is in nonTerminals
stack : ['$', 'VP', 'India'] buffer : ['India', 'won', 'the', 'championship']
buff = India
stack : ['$', 'VP'] buffer : ['won', 'the', 'championship']
VP is in nonTerminals
stack : ['$', 'NP', 'V'] buffer : ['won', 'the', 'championship']
V is in nonTerminals
stack : ['$', 'NP', 'won'] buffer : ['won', 'the', 'championship']
buff = won
stack : ['$', 'NP'] buffer : ['the', 'championship']
NP is in nonTerminals
stack : ['$', 'N', 'D'] buffer : ['the', 'championship']
D is in nonTerminals
stack : ['$', 'N', 'the'] buffer : ['the', 'championship']
buff = the
stack : ['$', 'N'] buffer : ['championship']
N is in nonTerminals
stack : ['$', 'championship'] buffer : ['championship']
buff = championship
stack : ['$'] buffer : []
Valid string

```

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

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  $\epsilon$ ):

```
E ::= T E'
E' ::= + T E'
E' ::= "
T ::= F T'
T' ::= * F T'
T' ::= "
F ::= ( E )
F ::= id
```

**Valid LL(1) Grammars**

For any production  $S \rightarrow A \mid B$ , 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  $\text{Follow}(A)$

**Formatting Instructions**

- The non-terminal on the left-hand-side of the first rule is the start non-terminal

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  $\epsilon$ ):

```
E ::= T E'
E' ::= + T E'
E' ::= "
T ::= F T'
T' ::= * F T'
T' ::= "
F ::= ( E )
F ::= id
```

**Valid LL(1) Grammars**

For any production  $S \rightarrow A \mid B$ , 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  $\text{Follow}(A)$

**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  $S ::= \text{start } \$$

**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`

Generate tables

Generate tables

## 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' | $E' ::= \epsilon$ | $E' ::= + T E'$   |                 |               | $E' ::= \epsilon$ |              |
| T  |                   |                   |                 | $T ::= F T'$  |                   | $T ::= F T'$ |
| T' | $T' ::= \epsilon$ | $T' ::= \epsilon$ | $T' ::= * F T'$ |               | $T' ::= \epsilon$ |              |
| F  |                   |                   |                 | $F ::= ( E )$ |                   | $F ::= id$   |

## 3. Parsing

Token stream separated by spaces:

Start/Reset Step Forward

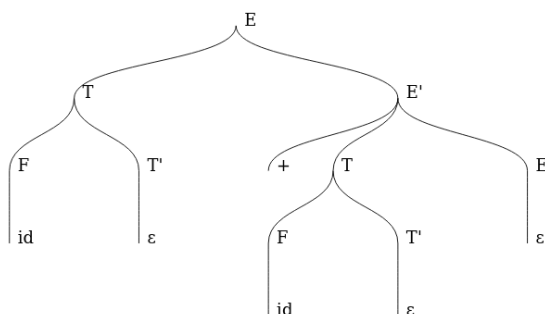
Stack

Remaining Input

Rule

Match \$

Partial Parse Tree



Practical 4 Problem defin ×LL(1) Parser Generator ×Compiler Design - Lexical ×

← → ↻https://www.cs.princeton.edu/courses/archive/spring20/cos320/LL1/90% ☆🔒 ⬇️ ☰

### 3. Parsing

Token stream separated by spaces:

Stack

Remaining Input

Rule

Match \$

#### Partial Parse Tree

```
graph TD; E --> T1[T]; E --> E_prime[E']; T1 --> F1[F]; T1 --> T_prime1[T']; F1 --> id1[id]; T_prime1 --> epsilon1[ε]; E_prime --> plus[+]; E_prime --> T2[T]; E_prime --> E_prime2[E']; T2 --> F2[F]; T2 --> T_prime2[T']; F2 --> id2[id]; T_prime2 --> star[*]; T_prime2 --> F3[F]; T_prime2 --> T_prime3[T']; F3 --> id3[id]; T_prime3 --> epsilon2[ε]; E_prime2 --> epsilon3[ε];
```