Name: Gaurav Kedia

Roll No: E-39

Section: E-2

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]:

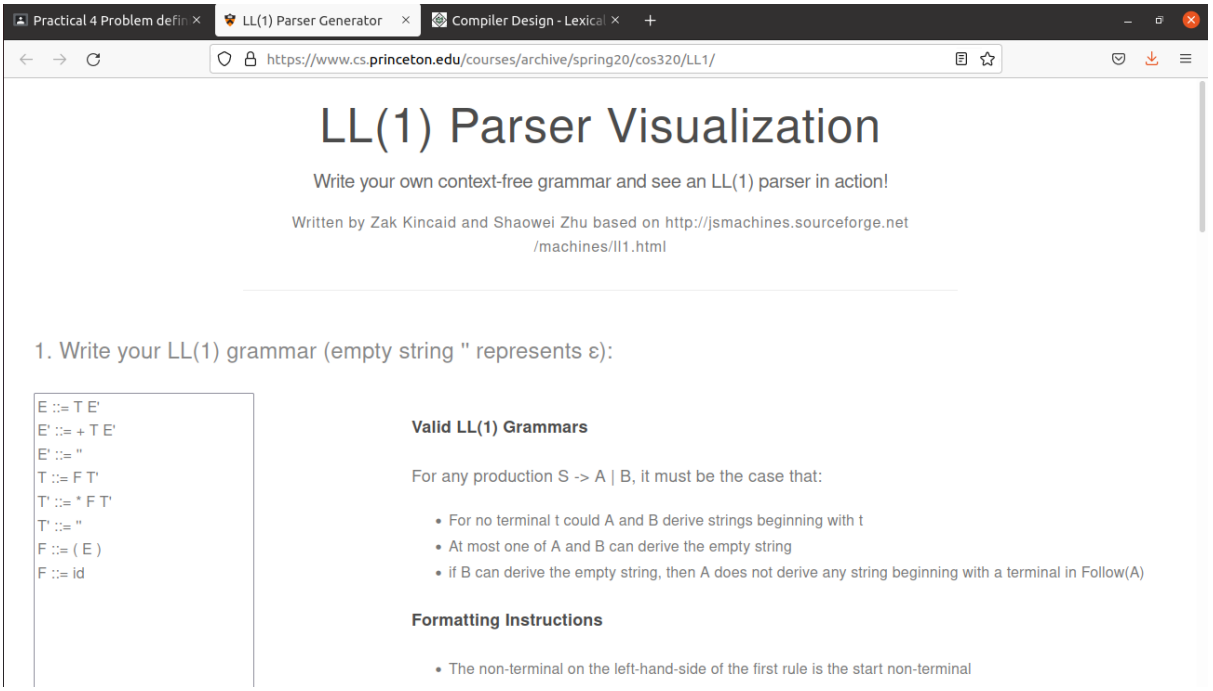
```
def table(row,column):
  if row == 'S':
    if column in ["is","want","won","played","me","I","you","India","Australia","Ste
      return ["VP","NP"]
    else:
      return 0
  elif row == "NP":
    if column in ["me","I","you"]:
      return ["P"]
    elif column in ["India","Australia","Steve","John"]:
      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
```

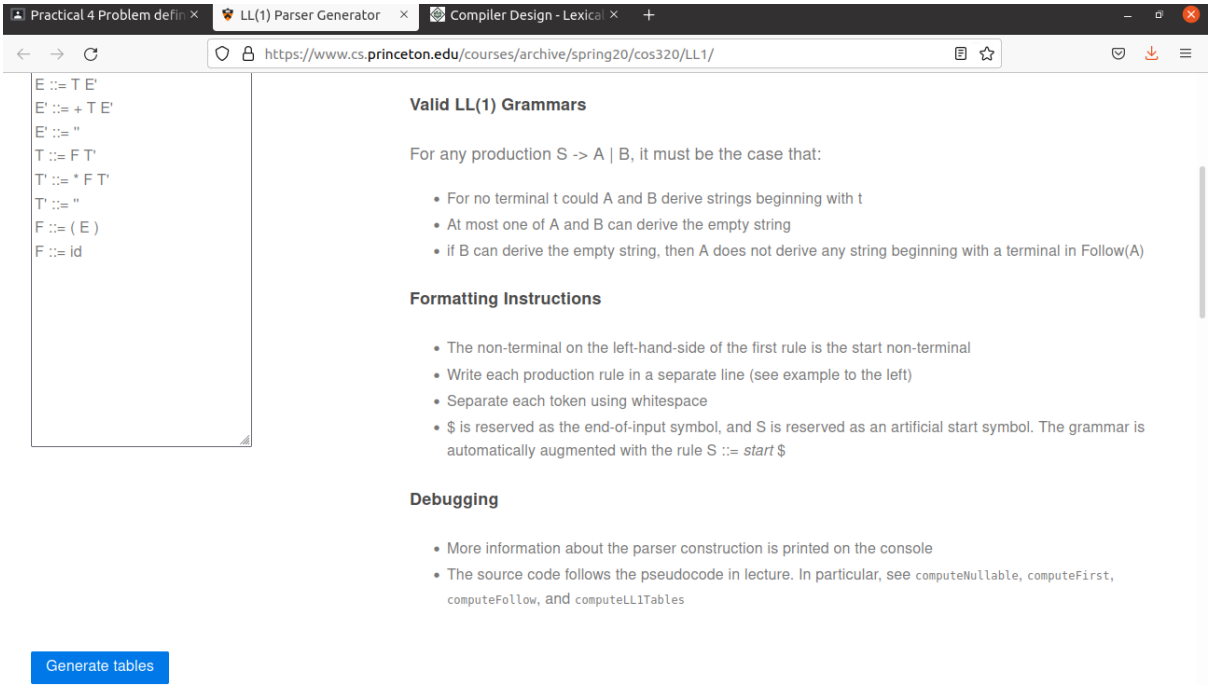
```
print("Please enter the string: ", end = "")
string = input()
string = list(string.split(" "))
stack = ["$","S"]
if len(string) == 0:
  print("Invalid string")
else:
  while len(stack) != 0:
    print("stack : ", stack, "buffer : ", string)
    stc = stack[-1]
    if stc == "$":
      print("Valid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

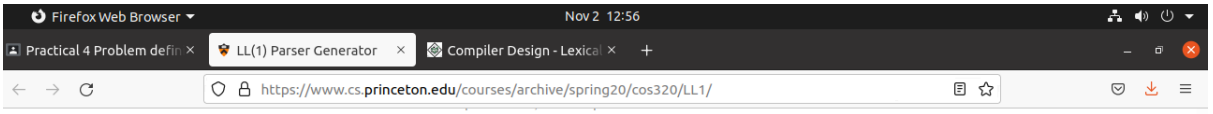
```
    terminals = "championship ball toss is want won played me I you India Australia
    terminals = list(terminals.split(" "))
    nonTerminals = "S NP VP N V P PN D"
    nonTerminals = list(nonTerminals.split(" "))
    stc = stack[-1]
    buff = string[0]
    stack.remove(stc)
    if stc in nonTerminals:
      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.





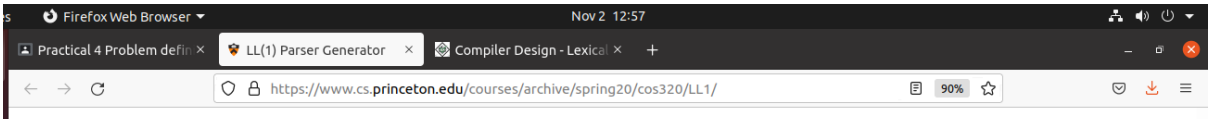


Generate tables

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' ::= 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

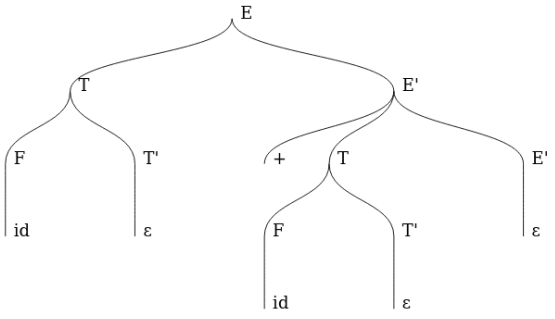
Stack

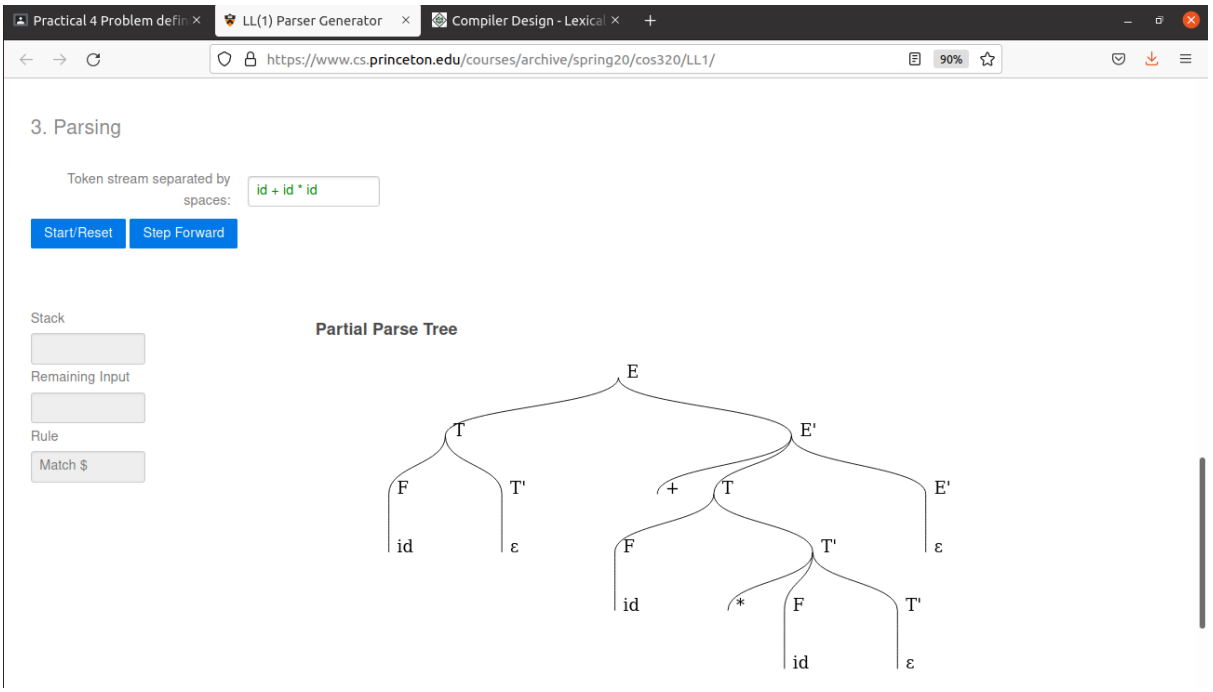
Remaining Input

Rule

Match $

Partial Parse Tree




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:
id + id * id
Start/Reset
Step Forward
Stack
Remaining Input
Rule
Match $
Partial Parse Tree
E
T
E'
F
T'
+
T
E'
id
ε
F
T'
ε
id
*
F
T'
id
ε