Name: Chetan Pardhi
Roll no: B22
Batch: B1
CD, Pract4

Problem statement:

(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 based on the grammar or not.

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

Code:

```
import java.util.*;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Map;
import java.util.TreeMap;

public class Main {

  public static void main(String[] args) {
    Map<String, Map<String, ArrayList<String>>> table = new
TreeMap<>();
    Map<String, String> map = new TreeMap<>();
    map.putIfAbsent("C", "Championship");
    map.putIfAbsent("b", "ball");
```

```
    map.putIfAbsent("t", "toss");
    map.putIfAbsent("is", "is");
    map.putIfAbsent("wa", "want");
    map.putIfAbsent("won", "won");
    map.putIfAbsent("P", "Played");
    map.put("me", "me");
    map.put("I", "I");
    map.put("y", "you");
    map.put("India", "India");
    map.put("AUS", "Australia");
    map.put("St", "Steve");
    map.put("J", "John");
    map.put("the", "the");
    map.put("a", "a");
    map.put("an", "an");
    Map<String, ArrayList<String>> r = new TreeMap<>();
    for (Map.Entry<String, String> val : map.entrySet()) {
      if (
        val.getKey() == "C" || val.getKey() == "b" ||
val.getKey() == "t"
      ) continue;
      r.put(val.getValue(), new
ArrayList<>(Arrays.asList("NP", "VP")));
    }
    table.put("S", r);

    r = new TreeMap<>();
    ArrayList<String> k = new
ArrayList<>(Arrays.asList("me", "I", "y"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList<>(Arrays.asList("P")));
```

```
    }
    k = new ArrayList<>(Arrays.asList("India", "AUS", "St",
"J"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList<>(Arrays.asList("PN")));
    }
    k = new ArrayList<>(Arrays.asList("the", "a", "an"));
    for (String l : k) {
      r.put(map.get(l), new ArrayList<>(Arrays.asList("D",
"N")));
    }
    table.put("NP", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("is", "wa", "won",
"P"));
    for (String l : k) {
      r.put(map.get(l), new ArrayList(Arrays.asList("V",
"NP")));
    }
    table.put("VP", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("C", "b", "t"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList(Arrays.asList(map.get(l))));
    }
    table.put("N", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("is", "wa", "won",
"P"));
    for (String l : k) {
```

```
      r.put(map.get(l), new
ArrayList(Arrays.asList(map.get(l))));
    }
    table.put("V", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("me", "I", "y"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList(Arrays.asList(map.get(l))));
    }
    table.put("P", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("India", "AUS", "St",
"J"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList(Arrays.asList(map.get(l))));
    }
    table.put("PN", r);
    r = new TreeMap<>();
    k = new ArrayList<>(Arrays.asList("the", "a", "an"));
    for (String l : k) {
      r.put(map.get(l), new
ArrayList(Arrays.asList(map.get(l))));
    }
    table.put("D", r);
    for (Map.Entry<String, Map<String, ArrayList<String>>>
l : table.entrySet()) {
      System.out.println(l.getKey() + " -> ");
      for (Map.Entry<String, ArrayList<String>> i :
l.getValue().entrySet()) {
        System.out.print("\t" + i.getKey() + " -> ");
```

```
        System.out.println(i.getValue());
      }
    }
    // String inputString = "India won the Championship
$";
    String inputString = "India won the $";
    String[] ipArr = inputString.split(" ");
    System.out.println(new
ArrayList(Arrays.asList(ipArr)));
    ArrayList<String> terminals = new ArrayList<>();
    for (String key : map.keySet()) {
      terminals.add(map.get(key));
    }
    ArrayList<String> nonTerminals = new ArrayList<>();
    for (String key : table.keySet()) {
      nonTerminals.add(key);
    }
    // System.out.println(nonTerminals);
    // System.out.println(terminals);

    Stack<String> stack = new Stack<>();
    boolean errF = false;
    int i = 0;
    stack.push("$");
    stack.push("S");
    while (!stack.empty()) {
      System.out.print("stack : ");

      System.out.println(stack);
      String A = stack.peek();
      String ree = ipArr[i];
      if (terminals.contains(A) || A.equals("$")) {
```

```
        if (A.equals(ree)) {
          i++;
          stack.pop();
        } else {
          errF = true;
          break;
        }
      } else if (nonTerminals.contains(A)) {
        Map<String, ArrayList<String>> bigger =
table.get(A);
        ArrayList<String> smaller = bigger.get(ree);
        System.out.println(smaller);
        if (smaller == null) {
          errF = true;
          System.out.println("ntbreak");
          break;
        } else {
          stack.pop();
          for (int iter = smaller.size() - 1; iter >= 0;
iter--) {
            stack.push(smaller.get(iter));
          }
        }
      }
    }
    System.out.print("stack : ");
    System.out.println(stack);
    if (errF) {
      System.out.println("invalid");
    } else {
      System.out.println("valid");
    }
```

```
    }
}
```

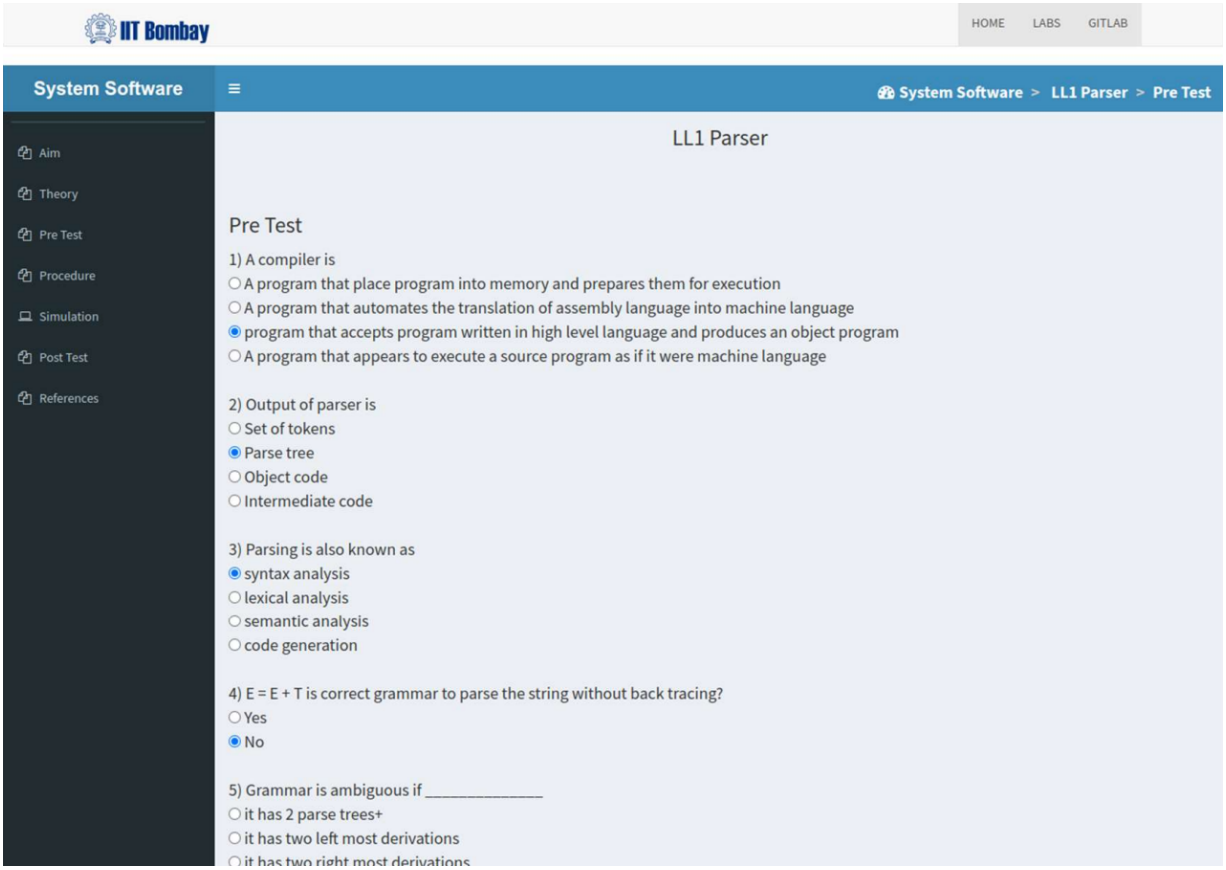
Screen-Shot:

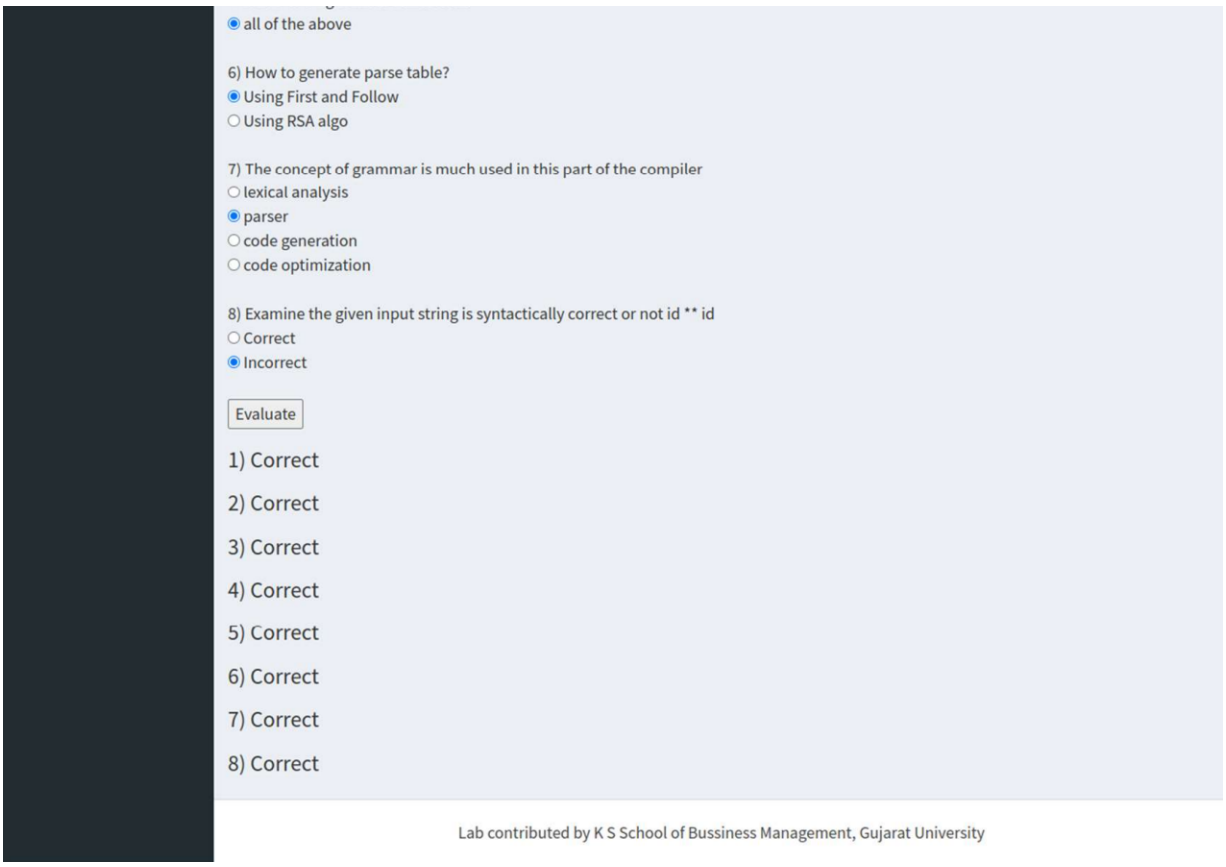
```
PS C:\Users\Chetan> cd "c:\Users\Chetan\" ; if ($?) { j
Note: Main.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
D ->
        a -> [a]
        an -> [an]
        the -> [the]
N ->
        Championship -> [Championship]
        ball -> [ball]
        toss -> [toss]
NP ->
        Australia -> [PN]
        I -> [P]
        India -> [PN]
        John -> [PN]
        Steve -> [PN]
        a -> [D, N]
        an -> [D, N]
        me -> [P]
        the -> [D, N]
        you -> [P]
P ->
        I -> [I]
        me -> [me]
        you -> [you]
PN ->
        Australia -> [Australia]
        India -> [India]
        John -> [John]
        Steve -> [Steve]
S ->
        Australia -> [NP, VP]
        I -> [NP, VP]
        India -> [NP, VP]
        John -> [NP, VP]
        Played -> [NP, VP]
        Steve -> [NP, VP]
        a -> [NP, VP]
        an -> [NP, VP]
        is -> [NP, VP]
        me -> [NP, VP]
        the -> [NP, VP]
        want -> [NP, VP]
        won -> [NP, VP]
        you -> [NP, VP]
```

```
        you -> [NP, VP]
V ->
        Played -> [Played]
        is -> [is]
        want -> [want]
        won -> [won]
VP ->
        Played -> [V, NP]
        is -> [V, NP]
        want -> [V, NP]
        won -> [V, NP]
[India, won, the, $]
stack : [$, S]
[NP, VP]
stack : [$, VP, NP]
[PN]
stack : [$, VP, PN]
[India]
stack : [$, VP, India]
stack : [$, VP]
[V, NP]
stack : [$, NP, V]
[won]
stack : [$, NP, won]
stack : [$, NP]
[D, N]
stack : [$, N, D]
[the]
stack : [$, N, the]
stack : [$, N]
null
ntbreak
stack : [$, N]
invalid
PS C:\Users\Chetan>
```

Pract-4b

Pre Test :


IIT Bombay
HOME
LABS
GITLAB
System Software
System Software > LL1 Parser > Pre Test
Aim
Theory
Pre Test
Procedure
Simulation
Post Test
References
LL1 Parser
Pre Test
1) A compiler is
A program that place program into memory and prepares them for execution
A program that automates the translation of assembly language into machine language
program that accepts program written in high level language and produces an object program
A program that appears to execute a source program as if it were machine language
2) Output of parser is
Set of tokens
Parse tree
Object code
Intermediate code
3) Parsing is also known as
syntax analysis
lexical analysis
semantic analysis
code generation
4) E = E + T is correct grammar to parse the string without back tracing?
Yes
No
5) Grammar is ambiguous if _____
it has 2 parse trees+
it has two left most derivations
it has two right most derivations


all of the above
6) How to generate parse table?
Using First and Follow
Using RSA algo
7) The concept of grammar is much used in this part of the compiler
lexical analysis
parser
code generation
code optimization
8) Examine the given input string is syntactically correct or not id ** id
Correct
Incorrect
Evaluate
1) Correct
2) Correct
3) Correct
4) Correct
5) Correct
6) Correct
7) Correct
8) Correct
Lab contributed by K S School of Bussiness Management, Gujarat University

Post Test :

Post Test

1) Once you land on the simulator page what is the default condition grammar for LL(1) parser?

- (●) Left recursive eliminated grammar
- ( ) Left recursive grammar
- ( ) Right recursive grammar
- ( ) Any grammar

2) What result do you expect when top down parsing will perform using right recursive grammar?

- ( ) Infinite loop
- (●) Top down parsing with Back tracking
- ( ) Parse the given input
- ( ) Syntax analysis

3) What importance of parser table to parse the given input string?

- ( ) Top down parsing with back tracking
- ( ) Bottom up parsing
- ( ) A formal grammar that contains left recursion cannot be parsed by a LL(k)-parser
- (●) Automatic Predictive top down parsing of the input string without backtracking

4) What do you observe when left recursive grammar is used without rewriting?

- ( ) A formal grammar that contains left recursion can be parsed by a LL(k)-parser
- ( ) Top down parsing with Back tracking
- (●) It leads them into infinite recursion

5) What is the use of first and follow algorithm?

- ( ) LL1 parsing of the given input string
- ( ) To generate left recursive eliminated grammar
- ( ) To generate left recursive grammar
- (●) predictive parse table construction for LL1 parsing

6) Generate LL(1) parsing for the grammar given in Task 2 for given input string (LO4) (LO5) id + id ** (id + id)
id +* id + (id * id)

7) analyze
id*+id

Evaluate

1) Correct

2) Correct

3) Correct

4) Correct

5) Correct

Creating Parse tree :

Partial Parse Tree