BHARATIYA VIDYA BHAVAN'S
SARDAR PATEL INSTITUTE OF TECHNOLOGY
(Empowered Autonomous Institute Affiliated to University of Mumbai)
[Knowledge is Nectar]

Department of Computer Engineering

e

Course - System Programming and Compiler Construction (SPCC)

Aim	Design syntax analyser for various grammars and implement using different parsing techniques* (Top-down and Bottom-up).
Objective	The goal of this experiment is to assess and comprehend the efficiency of diverse parsing techniques in syntax analysis. The objective involves the implementation of various parsers to evaluate their accuracy and effectiveness in analyzing the structural aspects of programming languages. Furthermore, the aim is to compare the performance of these parsing techniques to discern their strengths and limitations in practical applications. This endeavor is intended to deepen our understanding of compiler design and enhance parsing strategies for improved syntax analysis.
Theory	Syntax Analysis: ● Syntax analysis, also known as parsing, is a fundamental process in compiler design that involves analyzing a sequence of tokens to determine its grammatical structure according to the rules of a formal grammar. ● It is the second phase of the compilation process, following lexical analysis. Syntax analysis checks the source code for syntactical correctness and builds a data structure, typically a parse tree or an abstract syntax tree, that represents the hierarchical structure of the program. ● This process helps in identifying any syntax errors in the source code, ensuring that the code adheres to the syntax rules of the programming language. ● The outcome of syntax analysis is crucial for the subsequent phases of compilation, such as semantic analysis and code generation.[1] There are several types of parsing algorithms used in syntax analysis, including: 1. LL parsing: This is a top-down parsing algorithm that starts with the root of the parse tree and constructs the tree by successively expanding non-terminals. LL parsing is known for its simplicity and ease of implementation. LR parsing: This is a bottom-up parsing algorithm that starts with the leaves of the parse tree and constructs the tree by successively reducing terminals. LR parsing is more powerful than LL parsing and can handle a larger class of grammars. 3. LR(1) parsing: This is a variant of LR parsing that uses lookahead to disambiguate the grammar. 4. LALR parsing: This is a variant of LR parsing that uses a reduced set of lookahead symbols to reduce the number of states in the LR parser. 5. Once the parse tree is constructed, the compiler can perform semantic analysis to check if the source code makes sense and follows the semantics of the programming language.

6. The parse tree or AST can also be used in the code generation phase of the compiler design to generate intermediate code or machine code.[1]

Features of syntax analysis:

- Syntax Trees: Syntax analysis creates a syntax tree, which is a hierarchical representation of the code's structure. The tree shows the relationship between the various parts of the code, including statements, expressions, and operators.
- Context-Free Grammar: Syntax analysis uses context-free grammar to define the syntax of the programming language. Context-free grammar is a formal language used to describe the structure of programming languages.
- Top-Down and Bottom-Up Parsing: Syntax analysis can be performed using two main approaches: top-down parsing and bottom-up parsing. Top-down parsing starts from the highest level of the syntax tree and works its way down, while bottom-up parsing starts from the lowest level and works its way up.
- Error Detection: Syntax analysis is responsible for detecting syntax errors in the code. If the code does not conform to the rules of the programming language, the parser will report an error and halt the compilation process.
- Intermediate Code Generation: Syntax analysis generates an intermediate representation of the code, which is used by the subsequent phases of the compiler. The intermediate representation is usually a more abstract form of the code, which is easier to work with than the original source code.
- Optimization: Syntax analysis can perform basic optimizations on the code, such as removing redundant code and simplifying expressions.[2]

LL(1) Parser:

An LL(1) Parser operates by scanning the input from left to right, as indicated by the first 'L'. The second 'L' signifies the use of leftmost derivation in constructing the derivation tree. The '1' denotes the parser's lookahead capability, specifying it can preview one symbol ahead to make parsing decisions.

FIRST Sets:

The FIRST set for a non-terminal 'A' encompasses the set of terminals that might appear at the beginning of any string derived from 'A'. If 'A' can lead to an empty string, this empty string is also included in its FIRST set.
To compute the FIRST set, follow these rules:

1. If 'x' is a terminal, then FIRST(x) = { 'x' }.
2. For a production rule x -> ?, include ? in FIRST(x).

3. For a production X -> Y1 Y2 Y3...Yn, FIRST(X) = FIRST(Y1). If FIRST(Y1) includes ?, then FIRST(X) = { FIRST(Y1) - ? } U { FIRST(Y2) }. If all FIRST(Yi) for i=1 to n include ?, add ? to FIRST(X).

FOLLOW Sets:

FOLLOW sets help identify the terminal symbol that directly follows a non-terminal in a language, aiding in making parsing decisions without backtracking. The FOLLOW set is particularly useful for handling non-terminals that vanish on the right-hand side. The rules for computing the FOLLOW set are:

1. FOLLOW(S) = { $ }, with 'S' being the starting non-terminal.
2. For a production A -> pBq, add everything in FIRST(q) except Є to FOLLOW(B).
3. If A -> pB, then everything in FOLLOW(A) is in FOLLOW(B).
4. If A -> pBq and FIRST(q) includes Є, then FOLLOW(B) includes { FIRST(q) - Є } U FOLLOW(A).[1]

Conditions and Parse Table Construction:

Ensure the grammar is free from left recursion, unambiguous, and left-factored to be deterministic.
To build the parse table:

1. Verify the grammar meets essential conditions.
2. Calculate First() and Follow() for all non-terminals, where First() identifies possible starting terminals from a variable, and Follow() identifies the terminal following a variable in derivation.
3. For each production A -> α, populate the table with A -> α for each terminal in First(α).
4. If First(α) includes ε, then for each terminal in Follow(A), enter A -> ε in the table.
5. Include A -> ε in the table for the $ symbol if First(α) contains ε and Follow(A) includes $.

6.In constructing the parsing table, non-terminals fill rows and terminals fill columns. Null productions are placed under Follow elements, while other productions are placed under First set elements.

Advantages of LL(1) Parser:

- Ease of Implementation: Straightforward to design and debug due to simple left-to-right parsing and leftmost derivation construction.
- Parsing Efficiency: Suitable for real-time applications with linear time parsing for well-structured grammars.

- Early Error Detection: Detects syntactical errors immediately, enhancing development feedback.

Disadvantages of LL(1) Parser:

- Limited Grammar Support: Struggles with left-recursive grammars without transformation, complicating design.
- Single Token Lookahead: Can't handle grammars needing more than one token lookahead, restricting its applicability.
- Grammar Restrictions: Requires unambiguous and properly factored grammars, limiting language feature inclusion.

SLR Parser:

SLR is simple LR. It is the smallest class of grammar having few states. SLR is very easy to construct and is similar to LR parsing. The only difference between SLR parser and LR(0) parser is that in LR(0) parsing table, there's a chance of 'shift reduce' conflict because we are entering 'reduce' corresponding to all terminal states. We can solve this problem by entering 'reduce' corresponding to FOLLOW of LHS of production in the terminating state. This is called SLR(1) collection of items

Steps for constructing the SLR parsing table :

1. Writing augmented grammar
2. LR(0) collection of items to be found
3. Find FOLLOW of LHS of production
4. Defining 2 functions:goto[list of terminals] and action[list of non-terminals] in the parsing table

Advantages of SLR Parser:

1. Simple and easy to understand.
2. Small table sizes, making them faster to construct.

3.Requires less space for parsing tables.

Disadvantages of SLR Parser:

1. Limited lookahead, making it less powerful than other parsing techniques.

May generate false-positive shift-reduce conflicts.

Implementation / Code

1.LL(1):

a. PrettyPrinter.java:

```
import java.io.PrintStream;
import java.util.*;
import static java.lang.String.format;
import static java.lang.System.out;

public final class PrettyPrinter {

    private static final char BORDER_KNOT = '+';
    private static final char HORIZONTAL_BORDER = '-';
    private static final char VERTICAL_BORDER = '|';

    private static final String DEFAULT_AS_NULL = "(NULL)";

    private final PrintStream out;
    private final String asNull;

    public PrettyPrinter(PrintStream out) {
        this(out, DEFAULT_AS_NULL);
    }

    public PrettyPrinter(PrintStream out, String asNull) {
        if ( out == null ) {
            throw new IllegalArgumentException("No print stream provided");
        }
        if ( asNull == null ) {
            throw new IllegalArgumentException("No NULL-value placeholder
provided");
        }
        this.out = out;
        this.asNull = asNull;
    }
    public void convert(ArrayList<String> table[][])
    {
        String t[][]=new String[table.length][table[0].length];
        int i,j;
        for(i=0;i<table.length;i++)
        {
            for(j=0;j<table[0].length;j++)
            {
                StringJoiner sj = new StringJoiner(",");
                for(String str:table[i][j])
                {
```

```
                    sj.add(str);
                }
                t[i][j]=sj+"";
            }
        }
        print(t);
    }
    public void print(String[][] table) {
        if ( table == null ) {
            throw new IllegalArgumentException("No tabular data provided");
        }
        if ( table.length == 0 ) {
            return;
        }
        final int[] widths = new int[getMaxColumns(table)];
        adjustColumnWidths(table, widths);
        printPreparedTable(table, widths, getHorizontalBorder(widths));
    }

    private void printPreparedTable(String[][] table, int widths[], String
horizontalBorder) {
        final int lineLength = horizontalBorder.length();
        out.println(horizontalBorder);
        for ( final String[] row : table ) {
            if ( row != null ) {
                out.println(getRow(row, widths, lineLength));
                out.println(horizontalBorder);
            }
        }
    }

    private String getRow(String[] row, int[] widths, int lineLength) {
        final StringBuilder builder = new
StringBuilder(lineLength).append(VERTICAL_BORDER);
        final int maxWidths = widths.length;
        for ( int i = 0; i < maxWidths; i++ ) {
            builder.append(padRight(getCellValue(safeGet(row, i, null)),
widths[i])).append(VERTICAL_BORDER);
        }
        return builder.toString();
    }

    private String getHorizontalBorder(int[] widths) {
        final StringBuilder builder = new StringBuilder(256);
```

```
        builder.append(BORDER_KNOT);
        for ( final int w : widths ) {
            for ( int i = 0; i < w; i++ ) {
                builder.append(HORIZONTAL_BORDER);
            }
            builder.append(BORDER_KNOT);
        }
        return builder.toString();
    }

    private int getMaxColumns(String[][] rows) {
        int max = 0;
        for ( final String[] row : rows ) {
            if ( row != null && row.length > max ) {
                max = row.length;
            }
        }
        return max;
    }

    private void adjustColumnWidths(String[][] rows, int[] widths) {
        for ( final String[] row : rows ) {
            if ( row != null ) {
                for ( int c = 0; c < widths.length; c++ ) {
                    final String cv = getCellValue(safeGet(row, c, asNull));
                    final int l = cv.length();
                    if ( widths[c] < l ) {
                        widths[c] = l;
                    }
                }
            }
        }
    }

    private static String padRight(String s, int n) {
        return format("%1$-" + n + "s", s);
    }

    private static String safeGet(String[] array, int index, String
defaultValue) {
        return index < array.length ? array[index] : defaultValue;
    }

    private String getCellValue(Object value) {
```

```
        return value == null ? asNull : value.toString();
    }

}
```

b. AugmentedFirstAndFollow.java

```
import java.util.*;
import java.io.*;
class AugmentedFirstAndFollow
{
    String fLine="";
    ArrayList<String> ntMap=new ArrayList<>();
    ArrayList<String> tMap=new ArrayList<>();
    HashSet<String> ntCount=new HashSet<>();
    HashSet<String> tCount=new HashSet<>();
    HashMap<String,ArrayList<String>> _first=new HashMap<>();
    HashMap<String,ArrayList<String>> _follow=new HashMap<>();
    AugmentedFirstAndFollow()
    {
        //ntCount.add("E'");
        ntCount.add("E'");
        ntCount.add("T");
        ntCount.add("F");
        ntCount.add("E");
        ntCount.add("T'");
        //ntCount.add("Y");
        tCount.add("(");
        tCount.add(")");
        tCount.add("*");
        tCount.add("+");
        tCount.add("$");
        //tCount.add("-");
        //tCount.add("/");
        tCount.add("id");
    }
    public static void main(String args[])throws IOException
    {
        AugmentedFirstAndFollow obj=new AugmentedFirstAndFollow();
        ArrayList<String> arr[][]=obj.module1("in4.txt");
        /*if(arr==null)
        {
            System.out.println("Exiting...:(");
        }
        obj.module2(arr,"id $ ");
```

```
        obj.module2(arr,"( id ) * ( id ) * id $");
        obj.module2(arr,"id + id + + id $ ");*/
    }
    String join(ArrayList<String> v, String delim)
    {
        StringBuilder ss=new StringBuilder();
        for(int i = 0; i < v.size(); ++i)
        {
            if(i != 0)
                ss.append(delim);
            ss.append(v.get(i));
        }
        return ss.toString();
    }
    boolean isNonTerminal(char ch)
    {
        return ch>=65&&ch<=91;
    }
    String getChar(String s)
    {
        return s+"'";
    }
    public void module2( ArrayList<String> arr[][],String expr)
    {
        String tokens[]=expr.split(" ");
        Stack<String> stack=new Stack<>();
        stack.push("$");
        stack.push(fLine);
        int i,j,l;
        for(i=0;i<tokens.length;)
        {
            if(stack.empty())
                break;
            String ch=stack.peek();
            String top=tokens[i];
            //System.out.println(ch+","+top+","+i);
            if(ntCount.contains(ch))
            {
                stack.pop();
                if(arr[ntMap.indexOf(ch)][tMap.indexOf(top)].size()==0)
                {
                    System.out.println("Can't be derived :(");
                    break;
                }
```

```
                String str=arr[ntMap.indexOf(ch)][tMap.indexOf(top)].get(0);
                str=str.substring(str.indexOf("->")+2).trim();
                String s[]=str.split(" ");
                //System.out.println("Index:
"+ntMap.indexOf(ch)+","+tMap.indexOf(top)+","+"String: "+str);
                l=s.length;
                for(j=l-1;j>=0;j--)
                {
                    if(s[j].equals("@"))
                        continue;
                    stack.push(s[j]);
                }
            }
            else
            {
                if(ch.equals(top))
                {
                    i++;
                    stack.pop();
                }
                else
                {
                    System.out.println("Can't be derived :(");
                    break;
                }
            }
            //System.out.println("Stack: "+stack);
        }
        if(stack.empty()&&i==tokens.length)
        System.out.println("Can be derived :) ");
    }
    public ArrayList<String>[][] module1(String filename)throws IOException
    {
        BufferedReader br=new BufferedReader(new FileReader(filename));
        String str="";
        int i,line=0,j;

        HashMap<String,ArrayList<ArrayList<String>>> hm=new HashMap<>();
        HashMap<String,ArrayList<String>> first=new HashMap<>();
        HashMap<String,ArrayList<String>> follow=new HashMap<>();
        while((str=br.readLine())!=null)//Read the string, put in map
        {
            int l=str.indexOf("->");
            String left=str.substring(0,l).trim();
```

```
            String right=str.substring(l+2).trim();
            l=right.length();
            String tokens[]=right.split("[|]");
            System.out.println("Tokens: "+Arrays.toString(tokens));
            if(line==0)
            {
                fLine=left;
            }
            line++;
            ArrayList<ArrayList<String>> al=new ArrayList<>();
            for(i=0;i<tokens.length;i++)
            {
                String s[]=tokens[i].trim().split(" ");
                ArrayList<String> temp=new ArrayList<>();
                for(j=0;j<s.length;j++)
                    temp.add(s[j]);
                al.add(temp);
            }
            hm.put(left,al);//Put all the rules in the map
        }
        //Next segemnt for LR parsing only
        if(fLine.equals("E"))
        {
            ArrayList<ArrayList<String>> al=new ArrayList<>();
            ArrayList<String> al2=new ArrayList<>();
            al2.add("E");
            al.add(al2);
            hm.put("E'",al);
        }
        System.out.println(hm);
        br.close();
        //System.out.println(follow.get("E"));
        //System.out.println("Fline: "+fLine);
        //try
        //{
            System.out.println("The first set: ");
            calculateFirst(hm,first);
            System.out.println("The follow set:");
            calculateFollow(hm,first,follow);
            System.out.println("Table: ");
            _first=first;
            _follow=follow;
            //return null;
            return getMatrix(tCount,ntCount,first,follow,hm);
```

```
        /*}catch(Exception e)
        {
            System.out.println("Not a well defined grammar. Please check for
infinite productions/left recursion,etc.");
        }
        System.out.println(fLine);
        return null;*/
    }
    ArrayList<String>[][] getMatrix(HashSet<String> tCount, HashSet<String>
ntCount,HashMap<String,ArrayList<String>> first,
HashMap<String,ArrayList<String>> follow,
    HashMap<String,ArrayList<ArrayList<String>>> hm)
    {
        ArrayList<String> arr[][]=new
ArrayList[ntCount.size()][tCount.size()];
        ntMap.addAll(ntCount);
        tMap.addAll(tCount);
        //System.out.println(ntMap);
        //System.out.println(tMap);
        int i,j;
        for(i=0;i<ntCount.size();i++)
        {
            for(j=0;j<tCount.size();j++)
            {
                arr[i][j]=new ArrayList<>();
                /*if(i==0)
                System.out.print(tMap.get(j)+"\t\t");*/
            }
        }
        //System.out.println();
        for(Map.Entry<String, ArrayList<ArrayList<String>>>
mp:hm.entrySet())
        {
            String terminal=mp.getKey();
            ArrayList<ArrayList<String>> rules=mp.getValue();
            //System.out.println("Considering: "+terminal+"->"+rules);
            for(ArrayList<String> rule:rules)
            {
                String ss=rule.get(0);
                if(ntCount.contains(ss))
                {
                    ArrayList<String> al=first.get(ss);
                    for(String str:al)
                    {
```

```
                        ss=str.trim();
                        int row=ntMap.indexOf(terminal);
                        int col=tMap.indexOf(ss);
                        //System.out.println(row+","+col+","+terminal+"-
>"+rule);
                        //if(arr[row][col].size()==0)
                        arr[row][col].add(terminal+" -> "+join(rule,"
"));
                    }
                }
                else if(!ss.equals("@"))
                {
                    int row=ntMap.indexOf(terminal);
                    int col=tMap.indexOf(ss);
                    //System.out.println(row+","+col+","+terminal+"-
>"+rule);
                    //if(arr[row][col].size()==0)
                    arr[row][col].add(terminal+" -> "+join(rule," "));
                }
            }
            if(first.get(terminal).contains("@"))
            {
                for(String str:follow.get(terminal))
                {
                    int row=ntMap.indexOf(terminal);
                    int col=tMap.indexOf(str);
                    //System.out.println(row+","+col+","+terminal+"->@");
                    //if(arr[row][col].size()==0)
                    arr[row][col].add(terminal+" -> @ ");
                }
            }
        }
        boolean flag=false;
        for(i=0;i<ntCount.size();i++)
        {
            //System.out.print(ntMap.get(i)+"\t\t");
            for(j=0;j<tCount.size();j++)
            {
                //System.out.print(arr[i][j]+"\t\t");
                if(arr[i][j].size()>1)
                    flag=true;
            }
            //System.out.println();
        }
```

```
        ArrayList<String> pretty[][]=new
ArrayList[ntCount.size()+1][tCount.size()+1];
        for(i=0;i<ntCount.size()+1;i++)
        {
            for(j=0;j<tCount.size()+1;j++)
            {
                pretty[i][j]=new ArrayList<>();
                /*if(i==0)
                System.out.print(tMap.get(j)+"\t\t");*/
            }
        }
        pretty[0][0].add("(0,0)");
        for(i=1;i<tCount.size()+1;i++)
        {
            pretty[0][i].add(tMap.get(i-1)+"");
        }
        for(i=1;i<ntCount.size()+1;i++)
        {
            pretty[i][0].add(ntMap.get(i-1)+"");
        }
        for(i=1;i<ntCount.size()+1;i++)
        {
            for(j=1;j<tCount.size()+1;j++)
            {
                pretty[i][j].addAll(arr[i-1][j-1]);
            }
        }
        PrettyPrinter printer = new PrettyPrinter(System.out);
        printer.convert(pretty);
        if(flag)
        {
            System.out.println("Not a LL(1) grammar :(");
        }
        else
        {
            System.out.println("Grammar is LL(1) :) ");
        }
        return arr;
    }
    void calculateFirst(HashMap<String,ArrayList<ArrayList<String>>> hm,
HashMap<String,ArrayList<String>> first)
    {
        for(Map.Entry<String,ArrayList<ArrayList<String>>> mp:hm.entrySet())
        {
```

```
            String terminal=mp.getKey();
            ArrayList<ArrayList<String>> rules=mp.getValue();
            //System.out.println("Terminal,Rules: "+terminal+","+rules);
            if(first.get(terminal)!=null)//Already done;
            {
                System.out.println(terminal+"-
>"+unique(first.get(terminal)));
                continue;
            }
            System.out.println(terminal+"-
>"+unique(calculateFirstUtil(hm,first,terminal,rules)));
        }
    }
    ArrayList<String>
calculateFirstUtil(HashMap<String,ArrayList<ArrayList<String>>> hm,
HashMap<String,ArrayList<String>> first,
    String terminal, ArrayList<ArrayList<String>> rules)
    {
         if(first.get(terminal)!=null)
            return first.get(terminal);
         //System.out.println("Terminal,Rules: "+terminal+","+rules);
         for(ArrayList<String> rule:rules)
         {
             int i,l=rule.size();
             for(i=0;i<l;i++)
             {
                 //char ch=rule.charAt(i);
                 String ss=rule.get(i);
                 ArrayList<String> a=first.get(terminal);
                 if(a==null)
                 {
                     a=new ArrayList<>();
                     first.put(terminal,a);//no NPE
                     a=first.get(terminal);
                 }
                 if(!ntCount.contains(ss))
                 {
                     a.add(ss);
                     first.put(terminal,a);
                     break;
                 }
                 else
                 {
                     if(ss.equals(terminal))//for left-recursive grammars
```

```
                        break;
                    ArrayList<String>
temp=calculateFirstUtil(hm,first,ss,hm.get(ss));
                    a.addAll(temp);
                    first.put(terminal,a);
                    if(!temp.contains("@"))
                        break;
                }
            }
        }
        return first.get(terminal);
    }
    /*ArrayList<String> uniqueAndEpsilonLess(ArrayList<ArrayList<String>>
al)
    {
        ArrayList<String> a=new ArrayList<>();
        a.add("@");
        al.remove(a);
        return unique(al);
    }
    ArrayList<String> unique(ArrayList<ArrayList<String>> al)
    {
        HashSet<ArrayList<String>> hs=new HashSet<>();
        hs.addAll(al);
        al.clear();
        al.addAll(hs);
        return al;
    }*/
    ArrayList<String> uniqueAndEpsilonLess(ArrayList<String> al)
    {
        HashSet<String> hs=new HashSet<>();
        hs.addAll(al);
        hs.remove("@");
        al.clear();
        al.addAll(hs);
        return al;
    }
    ArrayList<String> unique(ArrayList<String> al)
    {
        HashSet<String> hs=new HashSet<>();
        hs.addAll(al);
        al.clear();
        al.addAll(hs);
        return al;
```

```
    }
    void calculateFollow(HashMap<String,ArrayList<ArrayList<String>>> hm,
HashMap<String,ArrayList<String>> first, HashMap<String,ArrayList<String>>
follow)
    {
        for(Map.Entry<String,ArrayList<ArrayList<String>>> mp:hm.entrySet())
        {
            String terminal=mp.getKey();
            ArrayList<ArrayList<String>> rules=mp.getValue();
            if(follow.get(terminal)!=null)//Already done;
            {
                System.out.println(terminal+"-
>"+uniqueAndEpsilonLess(follow.get(terminal)));
                continue;
            }
            System.out.println(terminal+"-
>"+uniqueAndEpsilonLess(calculateFollowUtil(hm,first,follow,terminal,0)));
        }
    }
    ArrayList<String>
calculateFollowUtil(HashMap<String,ArrayList<ArrayList<String>>> hm,
HashMap<String,ArrayList<String>> first,
    HashMap<String,ArrayList<String>> follow, String terminal,int count)
    {
        //System.out.println("Computing: "+terminal);
        if(count>=10)
            System.exit(0);
        if(follow.get(terminal)!=null)
            return follow.get(terminal);
        for(Map.Entry<String,ArrayList<ArrayList<String>>> mp:hm.entrySet())
        {
            ArrayList<ArrayList<String>> rules=mp.getValue();
            for(ArrayList<String> rule:rules)
            {
                 int i,l=rule.size();
                 if(rule.contains("@")&&l==1)
                    continue;
                 for(i=0;i<l;i++)
                 {
                     //char ch=rule.charAt(i);
                     String ss=rule.get(i);
                     //System.out.println("Considering:
"+ss+",Terminal:"+terminal);
                     ArrayList<String> a=follow.get(terminal);
```

```
                    if(a==null)
                       a=new ArrayList<>();
                    follow.put(terminal,a);//No NPE
                    //Use when just computing first follow
                    /*if((fLine).equals(terminal)&&!a.contains("$"))
                    {
                        a.add("$");
                        follow.put(terminal,a);
                        a=follow.get(terminal);
                        //System.out.println("1: "+follow);
                    }*/
                    if((fLine+"'").equals(terminal)&&!a.contains("$"))
                    {
                        a.add("$");
                        follow.put(terminal,a);
                        a=follow.get(terminal);
                        //System.out.println("1: "+follow);
                    }
                    if(terminal.equals(ss)&&i!=l-1)
                    {
                        i++;
                        ss=rule.get(i).trim();
                        //System.out.println("We're in");
                        if(!ntCount.contains(ss))
                        {
                            if(!ss.equals("@"))
                            {
                                a.add(ss);
                                follow.put(terminal,a);
                                //System.out.println("2: "+follow);
                            }
                        }
                        else
                        {
                            //System.out.println("Adding all the stuff of
first: "+ss);
                            a.addAll(first.get(ss));
                            while(first.get(ss).contains("@")&&i+1<l)
                            {
                                i++;
                                ss=rule.get(i);
                                a.addAll(first.get(ss));
                            }
```

```
                            if(i==l-
1&&!mp.getKey().equals(terminal))//Reached end while calculting follow
                            {
                                //System.out.println("Reached end, key is:
"+mp.getKey());
                                ArrayList<String>
temp=calculateFollowUtil(hm,first,follow,mp.getKey(),count+1);
                                a.addAll(temp);
                            }
                            follow.put(terminal,a);
                            //System.out.println("3: "+follow);
                        }
                    }
                    else if(terminal.equals(ss)&&i==l-
1&&!mp.getKey().equals(terminal))
                    {
                        //System.out.println("Trying to compute:
"+mp.getKey());
                        ArrayList<String>
temp=calculateFollowUtil(hm,first,follow,mp.getKey(),count+1);
                        a.addAll(temp);
                        follow.put(terminal,a);
                        //System.out.println("4: "+follow);
                    }
                }
            }
        }
        return follow.get(terminal);
    }
}
```

c. LR_parser.java:

```
import java.util.*;
import java.io.*;
class LR_parser
{
    HashMap<String,ArrayList<ArrayList<String>>> rules;
    HashSet<String> terminals;
    HashSet<String> nonTerminals;
    ArrayList<String> allSymbols;
    ArrayList<String> table[][];
    String startSymbol;
    DFA dfa;
    int minId;
```

```
static class Pair
{
    String rule;
    int dot;
    Pair(String c,int d)
    {
        rule=c;
        dot=d;
    }
    @Override
    public String toString()
    {
        int l=rule.indexOf("->"),i;
        String left=rule.substring(0,l).trim();
        String right=rule.substring(l+2).trim();
        StringBuilder sb=new StringBuilder();
        sb.append(left+" -> ");
        String tokens[]=right.split(" ");
        for(i=0;i<tokens.length;i++)
        {
            if(i==dot)
                sb.append(". ");
            sb.append(tokens[i]+" ");
        }
        if(i==dot)
            sb.append(". ");
        return sb.toString();
    }
    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + rule.hashCode();
        result = prime * result + dot;
        return result;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
```

```
            Pair other = (Pair) obj;
            if (dot!= other.dot)
                return false;
            if (!rule.equals(other.rule))
                return false;
            return true;
        }
    }
    static class DFA
    {
        HashSet<Pair> rules;
        int id;
        HashMap<String, DFA> transitions;
        DFA()
        {
            rules=new HashSet<>();
            id=-1;
            transitions=new HashMap<>();
        }
        @Override
        public String toString()
        {
            return "Id: "+id+" , Rules: "+rules+"\n";//Mapping:
"+transitions+"\n\n";
        }
    }
    LR_parser()
    {
        rules=new HashMap<>();
        startSymbol="";
        terminals=new HashSet<>();
        terminals.add("(");
        terminals.add("+");
        terminals.add("*");
        terminals.add(")");
        terminals.add("id");
        //terminals.add("=");
        terminals.add("/");
        terminals.add("-");
        nonTerminals=new HashSet<>();
        nonTerminals.add("E");
        nonTerminals.add("T");
        nonTerminals.add("F");
        allSymbols=new ArrayList<>();
```

```
        allSymbols.addAll(nonTerminals);
        allSymbols.addAll(terminals);
        minId=-1;
        dfa=new DFA();
    }
    private int getId()
    {
        return ++minId;
    }
    static void modify(AugmentedFirstAndFollow utils,LR_parser obj)
    {
        utils.ntCount.clear();
        utils.ntCount.addAll(obj.nonTerminals);
        utils.tCount.clear();
        utils.tCount.addAll(obj.terminals);
        utils.tCount.add("$");
        utils.ntCount.add(obj.startSymbol+"'");
    }
    public static void main(String args[])throws IOException
    {
        LR_parser obj=new LR_parser();
        obj.read_grammar("Grammar.txt");
        AugmentedFirstAndFollow utils=new AugmentedFirstAndFollow();
        obj.augment();
        modify(utils,obj);
        utils.module1("Grammar.txt");
        obj.unAugment();
        HashSet<Pair> hs=new HashSet<>();
        HashSet<Pair> closure=new HashSet<>();
        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));
        HashSet<Pair> goTo=new HashSet<>();
        /*goTo.add(new Pair("E -> T",0));
        goTo.add(new Pair("E -> T * F",0));
        System.out.println(obj.getGoto(goTo,"T"));
        goTo.clear();
        goTo.add(new Pair("F -> ( E )",0));
        System.out.println(obj.getGoto(goTo,"("));
        goTo.clear();
        goTo.add(new Pair("F -> ( E )",0));
        System.out.println(obj.getGoto(goTo,"id"));*/
        ArrayList<DFA> states=obj.buildDFA();
        obj.getParsingTable(states,utils);
        obj.parse("id $");
```

```
        /*for(;;)
        {
            closure.clear();
            String str=br.readLine();
            int k=Integer.parseInt(br.readLine());
            if(str.equals("1"))
                break;
            closure.add(new Pair(str,k));
            obj.getClosure(closure);
            System.out.println(closure);
        }*/
    }
    public void read_grammar(String filePath)
    {
        String str="";
        int line=0;
        try
        {
            BufferedReader br=new BufferedReader(new FileReader(filePath));
            while((str=br.readLine())!=null)//Read the string, put in map
            {
                int l=str.indexOf("->"),i,j;
                String left=str.substring(0,l).trim();
                String right=str.substring(l+2).trim();
                l=right.length();
                String tokens[]=right.split("[|]");
                //System.out.println("Tokens: "+Arrays.toString(tokens));
                if(line==0)
                {
                    startSymbol=left;
                }
                line++;
                ArrayList<ArrayList<String>> al=new ArrayList<>();
                for(i=0;i<tokens.length;i++)
                {
                    String s[]=tokens[i].trim().split(" ");
                    ArrayList<String> temp=new ArrayList<>();
                    for(j=0;j<s.length;j++)
                        temp.add(s[j]);
                    al.add(temp);
                }
                rules.put(left,al);//Put all the rules in the map
                line++;
            }
```

```
            br.close();
        }catch(Exception e)
        {
            System.out.println("Parse failure: "+e.getMessage());
            System.exit(0);
        }
    }
    String join(ArrayList<String> v, String delim)
    {
        StringBuilder ss=new StringBuilder();
        for(int i = 0; i < v.size(); ++i)
        {
            if(i != 0)
                ss.append(delim);
            ss.append(v.get(i));
        }
        return ss.toString();
    }
    public void getClosure(HashSet<Pair> closure)
    {
        boolean done=false;
        while(!done)
        {
            done=true;
            Iterator iterator=closure.iterator();
            HashSet<Pair> addAble=new HashSet<>();
            while(iterator.hasNext())
            {
                Pair pair=(Pair)iterator.next();
                //System.out.println(pair.rule);
                int l=pair.rule.indexOf("->"),i;
                //System.out.println(l);
                String left=pair.rule.substring(0,l).trim();
                String right=pair.rule.substring(l+2).trim();
                String tokens[]=right.split(" ");
                if(pair.dot>=tokens.length||pair.dot<0)
                    continue;
                else if(nonTerminals.contains(tokens[pair.dot]))
                {
                    ArrayList<ArrayList<String>>
al=rules.get(tokens[pair.dot]);
                    for(i=0;i<al.size();i++)
                    {
                        String str=join(al.get(i)," ");
```

```
                    Pair p=new Pair(tokens[pair.dot]+" -> "+str,0);
                    //System.out.println(p);
                    if(!closure.contains(p))
                    {
                        //System.out.println("Inside: "+p);
                        done=false;
                        addAble.add(new Pair(tokens[pair.dot]+" ->
"+str.trim(),0));
                    }
                }
            }
        }
        /*System.out.println(closure+"\n"+addAble);
        try
        {
            Thread.sleep(5000);
        }catch(Exception e)
        {
        }*/
        closure.addAll(addAble);
    }
}
public HashSet<Pair> getGoto(HashSet<Pair> X, String I)
{
    HashSet<Pair> goTo=new HashSet<>();
    HashSet<Pair> add=new HashSet<>();
    for(Pair p: X)
    {
        String str[]=p.rule.substring(p.rule.indexOf("-
>")+2).trim().split(" ");
        //System.out.println(Arrays.toString(str));
        if(p.dot>=str.length||p.dot<0)
            continue;
        if(str[p.dot].equals(I))
            goTo.add(new Pair(p.rule,p.dot+1));
    }
    for(Pair p:goTo)
    {
        HashSet<Pair> temp=new HashSet<>();
        temp.add(p);
        getClosure(temp);
        add.addAll(temp);
    }
    goTo.addAll(add);
```

```
        return goTo;
    }
    public void augment()
    {
        ArrayList<ArrayList<String>> al=new ArrayList<>();
        ArrayList<String> al2=new ArrayList<>();
        al2.add(startSymbol);
        al.add(al2);
        rules.put(startSymbol+"'",al);
    }
    public void unAugment()
    {
        rules.remove(startSymbol+"'");
    }
    public ArrayList<DFA> buildDFA()
    {
        augment();
        dfa.id=getId();
        for(Map.Entry<String, ArrayList<ArrayList<String>>> mp:
rules.entrySet())
        {
            for(ArrayList<String> al: mp.getValue())
            {
                dfa.rules.add(new Pair(mp.getKey()+" -> "+join(al," "),0));
            }
        }
        //System.out.println(dfa);
        ArrayList<DFA> states=new ArrayList<>();
        states.add(dfa);
        int i=0;
        boolean done=false;
        while(!done)
        {
            done=true;
            for(i=0;i<states.size();i++)
            {
                DFA current=states.get(i);
                for(String str: allSymbols)//All symbol iteration
                {
                    HashSet<Pair> Goto=getGoto(current.rules,str);
                    int index=getIndex(Goto,states);
                    if(index>=0)
                    {
                        //System.out.println("Old");
```

```
                    current.transitions.put(str,states.get(index));
                }
                else
                {
                    //System.out.println("New");
                    DFA new_DFA=new DFA();
                    new_DFA.id=getId();
                    new_DFA.rules=Goto;
                    states.add(new_DFA);
                    current.transitions.put(str,new_DFA);
                    done=false;
                }
            }
        }
        //break;
    }
    System.out.println(states);
    print_transitions(states);
    unAugment();
    return states;
}
public void getParsingTable(ArrayList<DFA>
states,AugmentedFirstAndFollow utils)
{
    terminals.add("$");
    table=new
ArrayList[states.size()][terminals.size()+nonTerminals.size()];
    int i,j;
    for(i=0;i<states.size();i++)
        for(j=0;j<terminals.size()+nonTerminals.size();j++)
            table[i][j]=new ArrayList<>();
    ArrayList<String> colMap=new ArrayList<>();
    colMap.addAll(terminals);
    colMap.addAll(nonTerminals);
    allSymbols.clear();
    allSymbols.addAll(colMap);
    System.out.println(colMap);
    System.out.println(allSymbols);
    int row=0;
    boolean isLR=true;
    for(DFA dfa: states)
    {
        for(Pair p:dfa.rules)
        {
```

```
                String str[]=p.rule.substring(p.rule.indexOf("-
>")+2).trim().split(" ");
                String left=p.rule.substring(0,p.rule.indexOf("->")).trim();
                if(p.dot<0||p.dot>str.length)
                    continue;
                if(left.equals(startSymbol+"'")&&p.dot==str.length)
                {
                    table[row][colMap.indexOf("$")].add("Accept :)");
                }
                else if(p.dot==str.length)
                {
                    ArrayList<String> al=utils._follow.get(left);
                    for(String s:al)
                    {
                        table[row][colMap.indexOf(s)].add("Reduce "+p.rule);
                    }
                }
                else if(terminals.contains(str[p.dot]))
                {
                    HashSet<Pair> hs=getGoto(dfa.rules,str[p.dot]);
                    int index=getIndex(hs,states);
                    if(index>=0)
                    {
                        table[row][colMap.indexOf(str[p.dot].trim())].add("S
hift "+index);
                    }
                }
            }
            for(String ss:nonTerminals)
            {
                HashSet<Pair> hs=getGoto(dfa.rules,ss);
                int index=getIndex(hs,states);
                if(index>=0&&states.get(index).rules.size()!=0)
                {
                    table[row][colMap.indexOf(ss)].add(index+"");
                }
            }
            row++;
        }
        ArrayList<String> pretty[][]=new
ArrayList[table.length+1][table[0].length+1];
        for(i=0;i<pretty.length;i++)
        {
            for(j=0;j<pretty[0].length;j++)
```

```
            {
                pretty[i][j]=new ArrayList<>();
                /*if(i==0)
                System.out.print(tMap.get(j)+"\t\t");*/
            }
        }
        pretty[0][0].add("State");
        for(i=1;i<pretty[0].length;i++)
        {
            pretty[0][i].add(colMap.get(i-1)+"");
        }
        for(i=1;i<pretty.length;i++)
        {
            pretty[i][0].add((i-1)+"");
        }
        for(i=1;i<pretty.length;i++)
        {
            for(j=1;j<pretty[0].length;j++)
            {
                if(table[i-1][j-1].size()>1)
                    isLR=false;
                pretty[i][j].addAll(table[i-1][j-1]);
            }
        }
        PrettyPrinter printer = new PrettyPrinter(System.out);
        printer.convert(pretty);
        terminals.remove("$");
        if(isLR)
            System.out.println("Grammar is LR :)");
        else
            System.out.println("Grammar isn't LR :(");
    }
    public void parse(String _toParse)
    {
        Stack<String> stack=new Stack<>();
        ArrayList<ArrayList<String>> al=new ArrayList<>();
        ArrayList<String> a=new ArrayList<>();
        a.add("Step");
        a.add("Stack");
        a.add("Action");
        a.add("Input");
        al.add(a);
        stack.push("$");
        stack.push("0");
```

```
        int pointer=0,i,step=0;
        String toParse[]=_toParse.split(" ");
        //System.out.println("Step:\t\tStack\t\t\t\t\tAction\t\tInput");
        while(!stack.empty()&&pointer<toParse.length)
        {
            a=new ArrayList<>();
            int row=Integer.parseInt(stack.peek());
            int col=allSymbols.indexOf(toParse[pointer]);
            a.add(step+"");
            a.add(stack+"");
            //System.out.print(step+"\t\t"+stack+"\t\t\t\t\t");
            if(table[row][col].size()==0)
            {
                a.add("Parse error");
                al.add(a);
                pretty_it(al);
                System.exit(0);
                break;
            }
            String action=table[row][col].get(0);
            a.add(action);
            //System.out.print(action+"\t\t");
            String str[]=action.split(" ");
            String left=str[0].trim();
            String right="";
            for(i=1;i<str.length;i++)
                right+=str[i]+" ";
            right=right.trim();
            if(left.equals("Shift"))
            {
                //stack.pop();
                stack.push(toParse[pointer]);
                stack.push(right);
                pointer++;
            }
            else if(left.equals("Reduce"))
            {
                left=right.substring(0,right.indexOf("->")).trim();
                right=right.substring(right.indexOf("->")+2).trim();
                for(i=0;i<2*(str.length-3);i++)
                {
                    if(stack.size()!=0)
                        stack.pop();
                    else
```

```
                    {
                        a.add("Parse error");
                        al.add(a);
                        pretty_it(al);
                        //System.out.println("Parse error");
                        System.exit(0);
                    }
                }
                int top=Integer.parseInt(stack.peek());
                stack.push(left);
                stack.push(table[top][allSymbols.indexOf(left)].get(0));
            }
            else if(left.equals("Accept")&&pointer==toParse.length-1)
            {
                String ppp="";
                for(i=pointer;i<toParse.length;i++)
                    ppp+=toParse[i]+" ";
                //System.out.println(ppp+"\t\t");
                a.add(ppp);
                al.add(a);
                pretty_it(al);
                System.out.println("Woohoo, accepted :) ");
                System.exit(0);
            }
            String pp="";
            for(i=pointer;i<toParse.length;i++)
                pp+=toParse[i]+" ";
            //System.out.println(pp+"\t\t");
            a.add(pp);
            al.add(a);
            step++;
        }
    }
    private void pretty_it(ArrayList<ArrayList<String>> al)
    {
        int n=al.size(),i,j;
        String t[][]=new String[n][4];
        for(i=0;i<n;i++)
        {
            for(j=0;j<4;j++)
            {
                if(al.get(i).size()==3&&j==3)
                    t[i][j]="Parse Error";
                else
```

```
                    t[i][j]=al.get(i).get(j);
                }
            }
            PrettyPrinter printer = new PrettyPrinter(System.out);
            printer.print(t);
        }
        private int getIndex(HashSet<Pair> Goto,ArrayList<DFA> states)
        {
            int i=0;
            for(DFA dfa: states)
            {
                if(dfa.rules.containsAll(Goto)&&Goto.containsAll(dfa.rules))
                    return i;
                i++;
            }
            return -1;
        }
        private void print_transitions(ArrayList<DFA> states)
        {
            for(DFA dfa: states)
            {
                System.out.println("Map for state: "+dfa);
                System.out.println("Transitions: ");
                for(Map.Entry<String, DFA> mp:dfa.transitions.entrySet())
                {
                    System.out.println(mp.getKey()+"->"+mp.getValue().rules+" (
S"+getIndex(mp.getValue().rules,states)+" )");
                }
                System.out.println();
            }
        }
}
```

2.SLR(1)

```
#include<stdio.h>
#include<ctype.h>
#include<string.h>

void followfirst(char , int , int);
void findfirst(char , int , int);
void follow(char c);

int count,n=0;
```

```
char calc_first[10][100];
char calc_follow[10][100];
int m=0;
char production[10][10], first[10];
char f[10];
int k;
char ck;
int e;

int main(int argc,char **argv)
{
    int jm=0;
    int km=0;
    int i,choice;
    char c,ch;
    printf("How many productions ? :");
    scanf("%d",&count);
    printf("\nEnter %d productions in form A=B where A and B are grammar
symbols :\n\n",count);
    for(i=0;i<count;i++)
    {
        scanf("%s%c",production[i],&ch);
    }
    int kay;
    char done[count];
    int ptr = -1;
    for(k=0;k<count;k++){
        for(kay=0;kay<100;kay++){
            calc_first[k][kay] = '!';
        }
    }
    int point1 = 0,point2,xxx;
    for(k=0;k<count;k++)
    {
        c=production[k][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(c == done[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        findfirst(c,0,0);
        ptr+=1;
```

```
        done[ptr] = c;
        printf("\n First(%c)= { ",c);
        calc_first[point1][point2++] = c;
        for(i=0+jm;i<n;i++){
            int lark = 0,chk = 0;
            for(lark=0;lark<point2;lark++){
                if (first[i] == calc_first[point1][lark]){
                    chk = 1;
                    break;
                }
            }
            if(chk == 0){
                printf("%c, ",first[i]);
                calc_first[point1][point2++] = first[i];
            }
        }
        printf("}\n");
        jm=n;
        point1++;
    }
    printf("\n");
    printf("-----------------------------------------------\n\n");
    char donee[count];
    ptr = -1;
    for(k=0;k<count;k++){
        for(kay=0;kay<100;kay++){
            calc_follow[k][kay] = '!';
        }
    }
    point1 = 0;
    int land = 0;
    for(e=0;e<count;e++)
    {
        ck=production[e][0];
        point2 = 0;
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == donee[kay])
                xxx = 1;
        if (xxx == 1)
            continue;
        land += 1;
        follow(ck);
        ptr+=1;
```

```
        donee[ptr] = ck;
        printf(" Follow(%c) = { ",ck);
        calc_follow[point1][point2++] = ck;
        for(i=0+km;i<m;i++){
            int lark = 0,chk = 0;
            for(lark=0;lark<point2;lark++){
                if (f[i] == calc_follow[point1][lark]){
                    chk = 1;
                    break;
                }
            }
            if(chk == 0){
                printf("%c, ",f[i]);
                calc_follow[point1][point2++] = f[i];
            }
        }
        printf(" }\n\n");
        km=m;
        point1++;
    }
    char ter[10];
    for(k=0;k<10;k++){
        ter[k] = '!';
    }
    int ap,vp,sid = 0;
    for(k=0;k<count;k++){
        for(kay=0;kay<count;kay++){
            if(!isupper(production[k][kay]) && production[k][kay]!= '#' &&
production[k][kay] != '=' && production[k][kay] != '\0'){
                vp = 0;
                for(ap = 0;ap < sid; ap++){
                    if(production[k][kay] == ter[ap]){
                        vp = 1;
                        break;
                    }
                }
                if(vp == 0){
                    ter[sid] = production[k][kay];
                    sid ++;
                }
            }
        }
    }
    ter[sid] = '$';
```

```
    sid++;
    printf("\n\t\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer
:-");
    printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
^^\n");
    printf("\n\t\t\t==========================================================
============================================================\n");
    printf("\t\t\t\t|\t");
    for(ap = 0;ap < sid; ap++){
        printf("%c\t\t",ter[ap]);
    }
    printf("\n\t\t\t==========================================================
============================================================\n");
    char first_prod[count][sid];
    for(ap=0;ap<count;ap++){
        int destiny = 0;
        k = 2;
        int ct = 0;
        char tem[100];
        while(production[ap][k] != '\0'){
            if(!isupper(production[ap][k])){
                tem[ct++] = production[ap][k];
                tem[ct++] = '_';
                tem[ct++] = '\0';
                k++;
                break;
            }
            else{
                int zap=0;
                int tuna = 0;
                for(zap=0;zap<count;zap++){
                    if(calc_first[zap][0] == production[ap][k]){
                        for(tuna=1;tuna<100;tuna++){
                            if(calc_first[zap][tuna] != '!'){
                                tem[ct++] = calc_first[zap][tuna];
                            }
                            else
                                break;
                        }
                    break;
                    }
                }
                tem[ct++] = '_';
            }
```

```
            k++;
        }
        int zap = 0,tuna;
        for(tuna = 0;tuna<ct;tuna++){
            if(tem[tuna] == '#'){
                zap = 1;
            }
            else if(tem[tuna] == '_'){
                if(zap == 1){
                    zap = 0;
                }
                else
                    break;
            }
            else{
                first_prod[ap][destiny++] = tem[tuna];
            }
        }
    }
    char table[land][sid+1];
    ptr = -1;
    for(ap = 0; ap < land ; ap++){
        for(kay = 0; kay < (sid + 1) ; kay++){
            table[ap][kay] = '!';
        }
    }
    for(ap = 0; ap < count ; ap++){
        ck = production[ap][0];
        xxx = 0;
        for(kay = 0; kay <= ptr; kay++)
            if(ck == table[kay][0])
                xxx = 1;
        if (xxx == 1)
            continue;
        else{
            ptr = ptr + 1;
            table[ptr][0] = ck;
        }
    }
    for(ap = 0; ap < count ; ap++){
        int tuna = 0;
        while(first_prod[ap][tuna] != '\0'){
            int to,ni=0;
            for(to=0;to<sid;to++){
```

```
                if(first_prod[ap][tuna] == ter[to]){
                    ni = 1;
                }
            }
            if(ni == 1){
                char xz = production[ap][0];
                int cz=0;
                while(table[cz][0] != xz){
                    cz = cz + 1;
                }
                int vz=0;
                while(ter[vz] != first_prod[ap][tuna]){
                    vz = vz + 1;
                }
                table[cz][vz+1] = (char)(ap + 65);
            }
            tuna++;
        }
    }
    for(k=0;k<sid;k++){
        for(kay=0;kay<100;kay++){
            if(calc_first[k][kay] == '!'){
                break;
            }
            else if(calc_first[k][kay] == '#'){
                int fz = 1;
                while(calc_follow[k][fz] != '!'){
                    char xz = production[k][0];
                    int cz=0;
                    while(table[cz][0] != xz){
                        cz = cz + 1;
                    }
                    int vz=0;
                    while(ter[vz] != calc_follow[k][fz]){
                        vz = vz + 1;
                    }
                    table[k][vz+1] = '#';
                    fz++;
                }
                break;
            }
        }
    }
    for(ap = 0; ap < land ; ap++){
```

```
        printf("\t\t\t   %c\t|\t",table[ap][0]);
        for(kay = 1; kay < (sid + 1) ; kay++){
            if(table[ap][kay] == '!')
                printf("\t\t");
            else if(table[ap][kay] == '#')
                printf("%c=#\t\t",table[ap][0]);
            else{
                int mum = (int)(table[ap][kay]);
                mum -= 65;
                printf("%s\t\t",production[mum]);
            }
        }
        printf("\n");
        printf("\t\t\t-------------------------------------------------------
-----------------------------------------------------------------");
        printf("\n");
    }
    int j;
    printf("\n\nPlease enter the desired INPUT STRING = ");
    char input[100];
    scanf("%s%c",input,&ch);
    printf("\n\t\t\t\t\t==========================================================
======================\n");
    printf("\t\t\t\t\tStack\t\t\tInput\t\t\tAction");
    printf("\n\t\t\t\t\t==========================================================
======================\n");
    int i_ptr = 0,s_ptr = 1;
    char stack[100];
    stack[0] = '$';
    stack[1] = table[0][0];
    while(s_ptr != -1){
        printf("\t\t\t\t\t\t");
        int vamp = 0;
        for(vamp=0;vamp<=s_ptr;vamp++){
            printf("%c",stack[vamp]);
        }
        printf("\t\t\t");
        vamp = i_ptr;
        while(input[vamp] != '\0'){
            printf("%c",input[vamp]);
            vamp++;
        }
        printf("\t\t\t");
        char her = input[i_ptr];
```

```
        char him = stack[s_ptr];
        s_ptr--;
        if(!isupper(him)){
            if(her == him){
                i_ptr++;
                printf("POP ACTION\n");
            }
            else{
                printf("\nString Not Accepted by LL(1) Parser !!\n");
                exit(0);
            }
        }
        else{
            for(i=0;i<sid;i++){
                if(ter[i] == her)
                    break;
            }
            char produ[100];
            for(j=0;j<land;j++){
                if(him == table[j][0]){
                    if (table[j][i+1] == '#'){
                        printf("%c=#\n",table[j][0]);
                        produ[0] = '#';
                        produ[1] = '\0';
                    }
                    else if(table[j][i+1] != '!'){
                        int mum = (int)(table[j][i+1]);
                        mum -= 65;
                        strcpy(produ,production[mum]);
                        printf("%s\n",produ);
                    }
                    else{
                        printf("\nString Not Accepted by LL(1) Parser
!!\n");
                        exit(0);
                    }
                }
            }
            int le = strlen(produ);
            le = le - 1;
            if(le == 0){
                continue;
            }
            for(j=le;j>=2;j--){
```

```
                s_ptr++;
                stack[s_ptr] = produ[j];
            }
        }
    }
    printf("\n\t\t\t==========================================================
==========================================================================\n");
    if (input[i_ptr] == '\0'){
        printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");
    }
    else
        printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");
    printf("\t\t\t==========================================================
==========================================================================\n");
}

void follow(char c)
{
    int i ,j;
    if(production[0][0]==c){
        f[m++]='$';
    }
    for(i=0;i<10;i++)
    {
        for(j=2;j<10;j++)
        {
            if(production[i][j]==c)
            {
                if(production[i][j+1]!='\0'){
                    followfirst(production[i][j+1],i,(j+2));
                }
                if(production[i][j+1]=='\0'&&c!=production[i][0]){
                    follow(production[i][0]);
                }
            }
        }
    }
}

void findfirst(char c ,int q1 , int q2)
{
    int j;
    if(!(isupper(c))){
        first[n++]=c;
```

```
    }
    for(j=0;j<count;j++)
    {
        if(production[j][0]==c)
        {
            if(production[j][2]=='#'){
                if(production[q1][q2] == '\0')
                    first[n++]='#';
                else if(production[q1][q2] != '\0' && (q1 != 0 || q2 != 0))
                {
                    findfirst(production[q1][q2], q1, (q2+1));
                }
                else
                    first[n++]='#';
            }
            else if(!isupper(production[j][2])){
                first[n++]=production[j][2];
            }
            else {
                findfirst(production[j][2], j, 3);
            }
        }
    }
}

void followfirst(char c, int c1 , int c2)
{
    int k;
    if(!(isupper(c)))
        f[m++]=c;
    else{
        int i=0,j=1;
        for(i=0;i<count;i++)
        {
            if(calc_first[i][0] == c)
                break;
        }
        while(calc_first[i][j] != '!')
        {
            if(calc_first[i][j] != '#'){
                f[m++] = calc_first[i][j];
            }
            else{
                if(production[c1][c2] == '\0'){
```

```
                    follow(production[c1][0]);
                }
                else{
                    followfirst(production[c1][c2],c1,c2+1);
                }
            }
            j++;
        }
    }
}
```

Output

```
Enter 3 productions in form A=B where A and B are grammar symbols :

E=BB
B=cB
B=d

 First(E)= { c, d, }

 First(B)= { c, d, }

------------------------------------------------

 Follow(E) = { $,  }

 Follow(B) = { c, d, $,  }


                              The LL(1) Parsing Table for the above grammer :-
                              ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

                         ========================================================================
                                 |       c                 d                 $
                         ========================================================================
                          E      |       E=BB              E=BB
                         ------------------------------------------------------------------------
                          B      |       B=cB              B=d
                         ------------------------------------------------------------------------

Please enter the desired INPUT STRING = cdcd

                                  ================================================================
                                        Stack                   Input                   Action
                                  ================================================================
                                        $E                      cdcd                    E=BB
                                        $BB                     cdcd                    B=cB
                                        $BBc                    cdcd                    POP ACTION
                                        $BB                     dcd                     B=d
                                        $Bd                     dcd                     POP ACTION
                                        $B                      cd                      B=cB
                                        $Bc                     cd                      POP ACTION
                                        $B                      d                       B=d
                                        $d                      d                       POP ACTION
                                        $
                                  ================================================
                                  YOUR STRING HAS BEEN ACCEPTED !!
                                  ================================================
```

Conclusion

During this experiment, I gained insights into the operation of an LL(1) parser and SLR. I actively engaged in the development and application of computing 'first' and 'follow' sets for a given production, leveraging these concepts to construct a parse tree. Moreover, I assessed the validity of a particular string through parsing and implemented a stack to

	facilitate this procedural aspect. This hands-on experience provided a practical understanding of parser functionality and its application in string validation.
References	[1] GeeksforGeeks. (2023b, August 22). Construction of LL 1 parsing table. https://www.geeksforgeeks.org/construction-of-ll1-parsing-table/ [2] GeeksforGeeks. (2023a, April 19). Introduction to Syntax analysis in Compiler Design. https://www.geeksforgeeks.org/introduction-to-syntax-analysis-in-compiler-design/ [3] GeeksforGeeks. (2023b, June 10). FOLLOW set in syntax analysis. https://www.geeksforgeeks.org/follow-set-in-syntax-analysis/?ref=lbp [4] Siddhantbajaj. (n.d.). GitHub - siddhantbajaj1/LL-1-Parsing-Table: The following repo contains a C code which takes a regular grammar as input and generates the FirstPos , and FollowPos of the non - Terminals and Displays its corresponding LL(1) Parsing Table. Then using that table and an input string it simulates the LL(1) parsing action. GitHub. https://github.com/siddhantbajaj1/LL-1-Parsing-Table [5] What are the main advantages and disadvantages of LL and LR parsing? (n.d.). Software Engineering Stack Exchange. https://softwareengineering.stackexchange.com/questions/19541/what-are-the-mai n-advantages-and-disadvantages-of-ll-and-lr-parsing