System Software Practicals

Name : Pradip .S. Karmakar

Roll No : 10

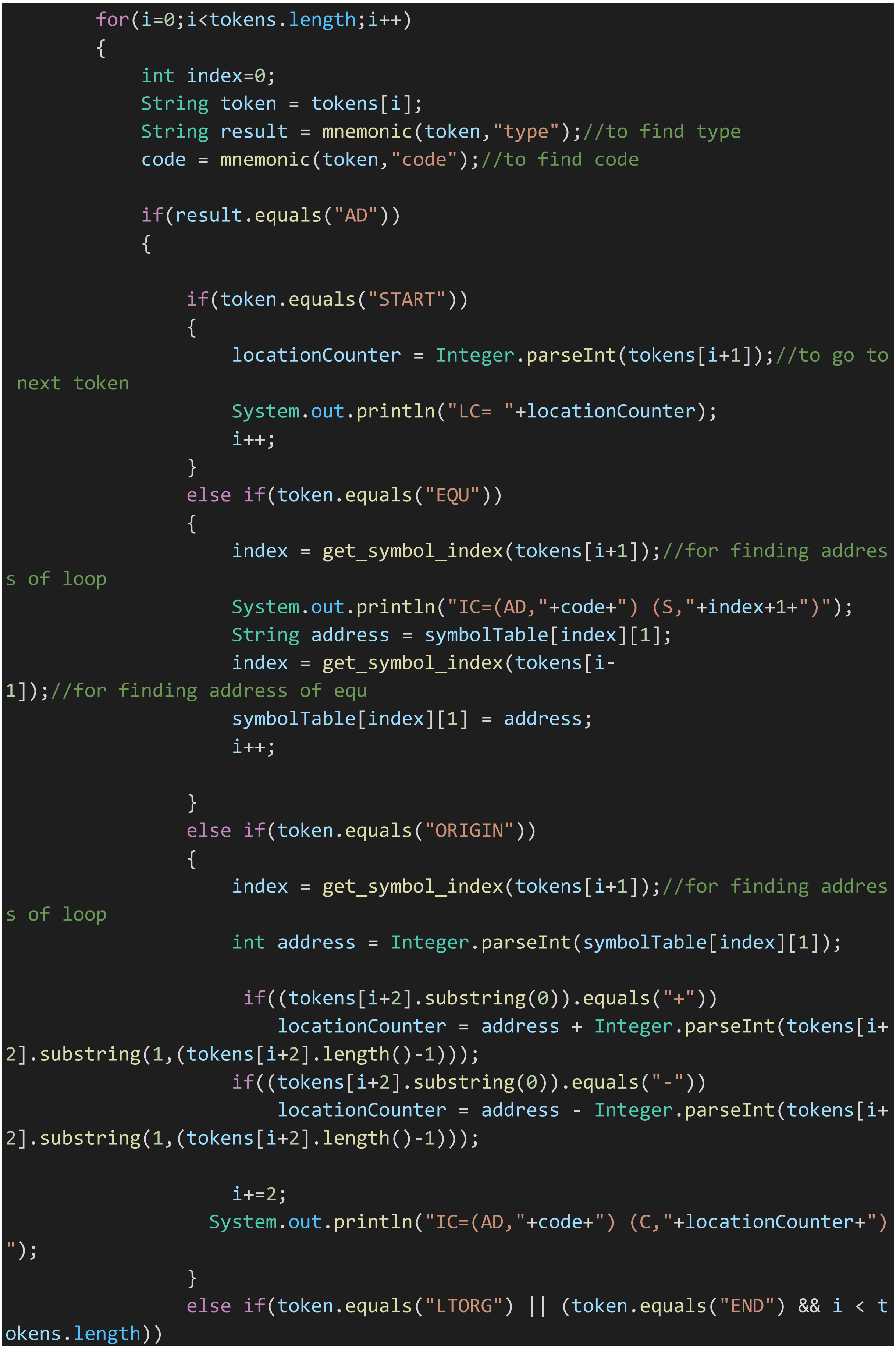
Class : MCA – 3

Subject : System Software

---

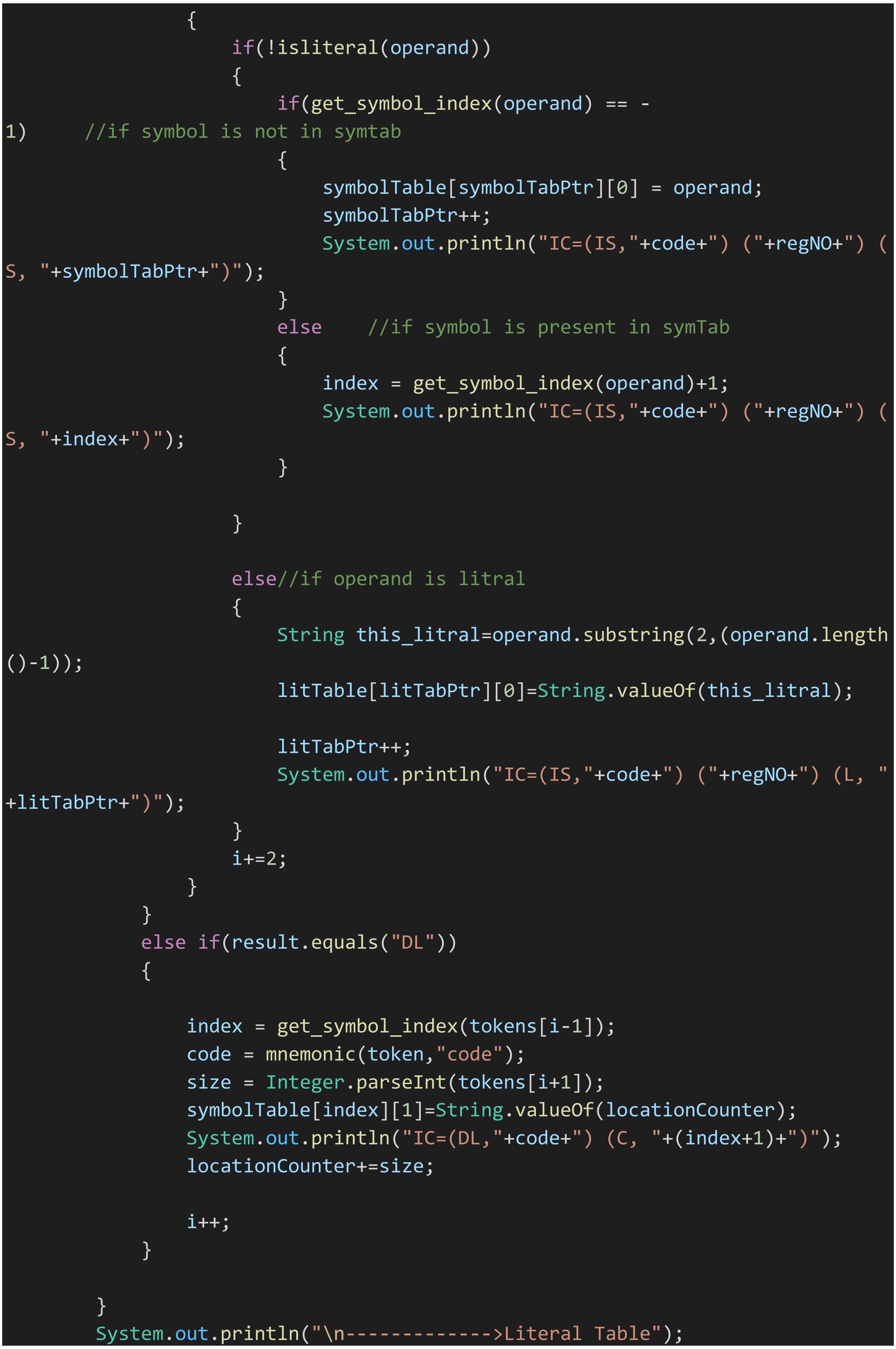
1. Assembler

```
import java.util.StringTokenizer;

public class assembler
{
    static String[][] symbolTable=new String[10][2] ;//this is symbol table
    static String[][] litTable=new String[10][2] ;//this is literal table
    static int[] poolTable= new int[10];//this is literal table
    static int locationCounter =0;
    static int poolTabPtr = 0;//pooltable pointer
    static int litTabPtr = 0;//literaltable pointer
    static int symbolTabPtr=0;

   public static void main(String[] args)
   {
        poolTable[0]=1;
        String statements = "START 200\nREAD A\nREAD B\nMOVER AREG,A\nADD AREG
,B\nMOVEM AREG,RESULT\nPRINT A\nPRINT B\nPRINT RESULT\nA DS 0\nB DS 1\nRESULT
DS 0\nEND";

        String delimiters = "[, \n\t]"; //comma,space,new line, tab are delimi
ters
        String[] tokens = statements.split(delimiters, 0);

        String code;
        String regNO;
        int size=0;
        int i=0;
```

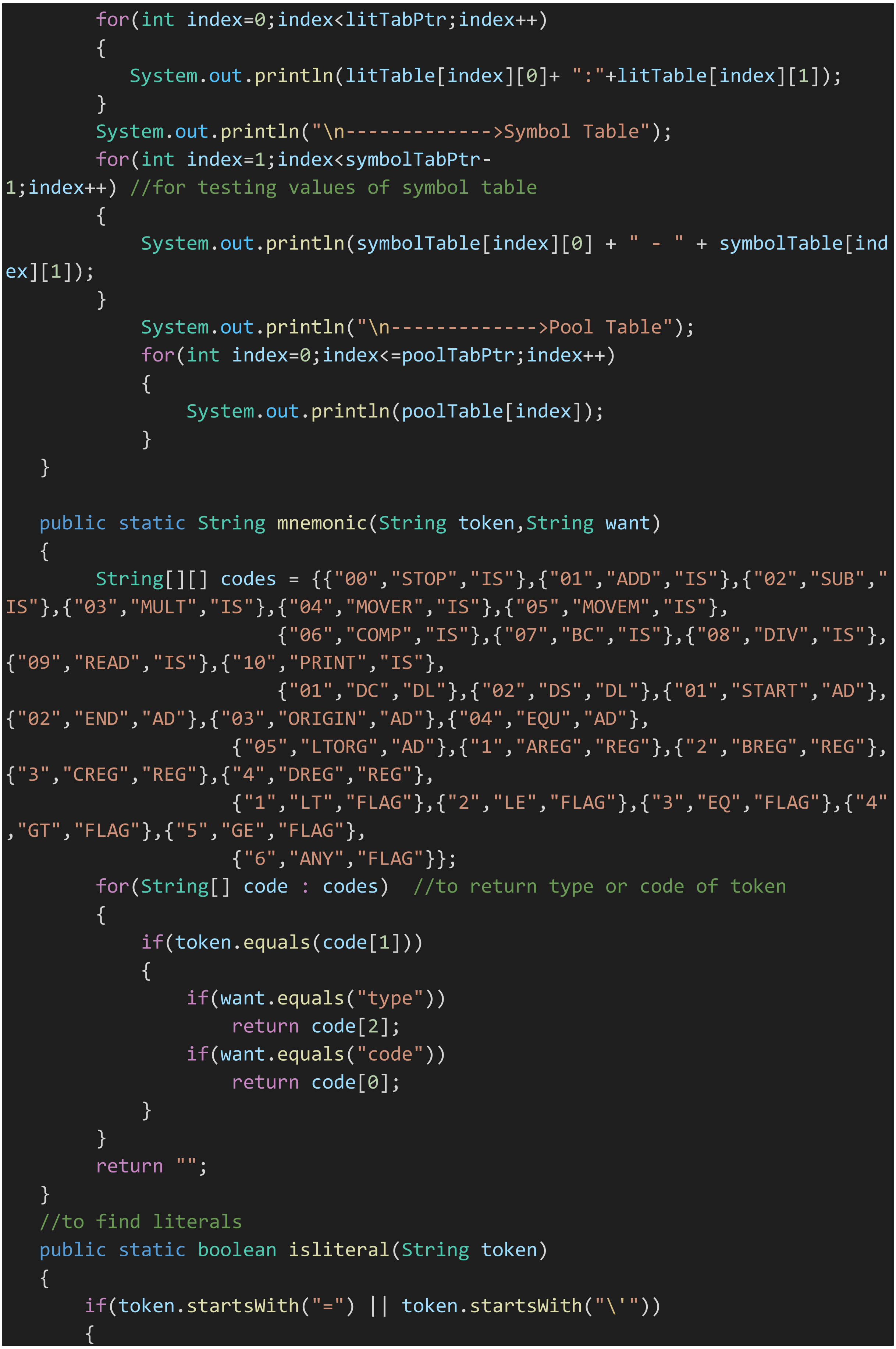


```
        for(i=0;i<tokens.length;i++)
        {
            int index=0;
            String token = tokens[i];
            String result = mnemonic(token,"type");//to find type
            code = mnemonic(token,"code");//to find code

            if(result.equals("AD"))
            {

                if(token.equals("START"))
                {
                    locationCounter = Integer.parseInt(tokens[i+1]);//to go to
 next token
                    System.out.println("LC= "+locationCounter);
                    i++;
                }
                else if(token.equals("EQU"))
                {
                    index = get_symbol_index(tokens[i+1]);//for finding addres
s of loop
                    System.out.println("IC=(AD,"+code+") (S,"+index+1+")");
                    String address = symbolTable[index][1];
                    index = get_symbol_index(tokens[i-
1]);//for finding address of equ
                    symbolTable[index][1] = address;
                    i++;

                }
                else if(token.equals("ORIGIN"))
                {
                    index = get_symbol_index(tokens[i+1]);//for finding addres
s of loop
                    int address = Integer.parseInt(symbolTable[index][1]);

                     if((tokens[i+2].substring(0)).equals("+"))
                        locationCounter = address + Integer.parseInt(tokens[i+
2].substring(1,(tokens[i+2].length()-1)));
                    if((tokens[i+2].substring(0)).equals("-"))
                        locationCounter = address - Integer.parseInt(tokens[i+
2].substring(1,(tokens[i+2].length()-1)));

                    i+=2;
                  System.out.println("IC=(AD,"+code+") (C,"+locationCounter+")
");
                }
                else if(token.equals("LTORG") || (token.equals("END") && i < t
okens.length))
```

```
                {

                    for(index = poolTable[poolTabPtr]-
1;index<litTabPtr;index++)
                    {
                        litTable[index][1] = String.valueOf(locationCounter);
                        System.out.println("IC=(AD,"+litTable[index][0]+") (C,
"+locationCounter+")");
                        locationCounter++;
                    }

                    poolTabPtr++;
                    poolTable[poolTabPtr] = litTabPtr+1;

                }


            }
            else if(result == "" && !isliteral(token))//for label
            {
                index=get_symbol_index(token);

                if(mnemonic(tokens[i+1],"type").equals("IS"))
                {
                    if(index== -1)  //labal is not inserted in symolTable
                    {
                        symbolTable[symbolTabPtr][0] = token;
                        symbolTable[symbolTabPtr][1] = String.valueOf(location
Counter);//to insert lc in symbol table
                        symbolTabPtr++;
                    }
                    else
                    {
                        symbolTable[index][1]=String.valueOf(locationCounter);
                    }
                }
            }
            else if(result.equals("IS"))
            {

                regNO= mnemonic(tokens[i+1],"code");//to find register number
eg.1 for AREG
                locationCounter++;
                String operand = tokens[i+2];
                if(token.equals("STOP"))    //if stop condition
                    System.out.println("IC=(IS,00)");

                else
```



```
                {
                    if(!isliteral(operand))
                    {
                        if(get_symbol_index(operand) == -
1)      //if symbol is not in symtab
                        {
                            symbolTable[symbolTabPtr][0] = operand;
                            symbolTabPtr++;
                            System.out.println("IC=(IS,"+code+") ("+regNO+") (
S, "+symbolTabPtr+")");
                        }
                        else    //if symbol is present in symTab
                        {
                            index = get_symbol_index(operand)+1;
                            System.out.println("IC=(IS,"+code+") ("+regNO+") (
S, "+index+")");
                        }

                    }

                    else//if operand is litral
                    {
                        String this_litral=operand.substring(2,(operand.length
()-1));

                        litTable[litTabPtr][0]=String.valueOf(this_litral);

                        litTabPtr++;
                        System.out.println("IC=(IS,"+code+") ("+regNO+") (L, "
+litTabPtr+")");
                    }
                    i+=2;
                }
            }
            else if(result.equals("DL"))
            {

                index = get_symbol_index(tokens[i-1]);
                code = mnemonic(token,"code");
                size = Integer.parseInt(tokens[i+1]);
                symbolTable[index][1]=String.valueOf(locationCounter);
                System.out.println("IC=(DL,"+code+") (C, "+(index+1)+")");
                locationCounter+=size;

                i++;
            }

        }
        System.out.println("\n------------->Literal Table");
```



```
        for(int index=0;index<litTabPtr;index++)
        {
           System.out.println(litTable[index][0]+ ":"+litTable[index][1]);
        }
        System.out.println("\n------------->Symbol Table");
        for(int index=1;index<symbolTabPtr-
1;index++) //for testing values of symbol table
        {
            System.out.println(symbolTable[index][0] + " - " + symbolTable[ind
ex][1]);
        }
            System.out.println("\n------------->Pool Table");
            for(int index=0;index<=poolTabPtr;index++)
            {
                System.out.println(poolTable[index]);
            }
   }

   public static String mnemonic(String token,String want)
   {
        String[][] codes = {{"00","STOP","IS"},{"01","ADD","IS"},{"02","SUB","
IS"},{"03","MULT","IS"},{"04","MOVER","IS"},{"05","MOVEM","IS"},
                        {"06","COMP","IS"},{"07","BC","IS"},{"08","DIV","IS"},
{"09","READ","IS"},{"10","PRINT","IS"},
                        {"01","DC","DL"},{"02","DS","DL"},{"01","START","AD"},
{"02","END","AD"},{"03","ORIGIN","AD"},{"04","EQU","AD"},
                    {"05","LTORG","AD"},{"1","AREG","REG"},{"2","BREG","REG"},
{"3","CREG","REG"},{"4","DREG","REG"},
                    {"1","LT","FLAG"},{"2","LE","FLAG"},{"3","EQ","FLAG"},{"4"
,"GT","FLAG"},{"5","GE","FLAG"},
                    {"6","ANY","FLAG"}};
        for(String[] code : codes)  //to return type or code of token
        {
            if(token.equals(code[1]))
            {
                if(want.equals("type"))
                    return code[2];
                if(want.equals("code"))
                    return code[0];
            }
        }
        return "";
   }
   //to find literals
   public static boolean isliteral(String token)
   {
       if(token.startsWith("=") || token.startsWith("\'"))
       {
           return true;
       }
       return false;
   }

   //tocheck already exist
   public static int get_symbol_index(String token)
   {
        int index;
        for(index=0;index<symbolTabPtr;index++)
        {
            if(symbolTable[index][0].equals(token))
            {
                return index;
            }
        }


       return -1;
   }
}
```

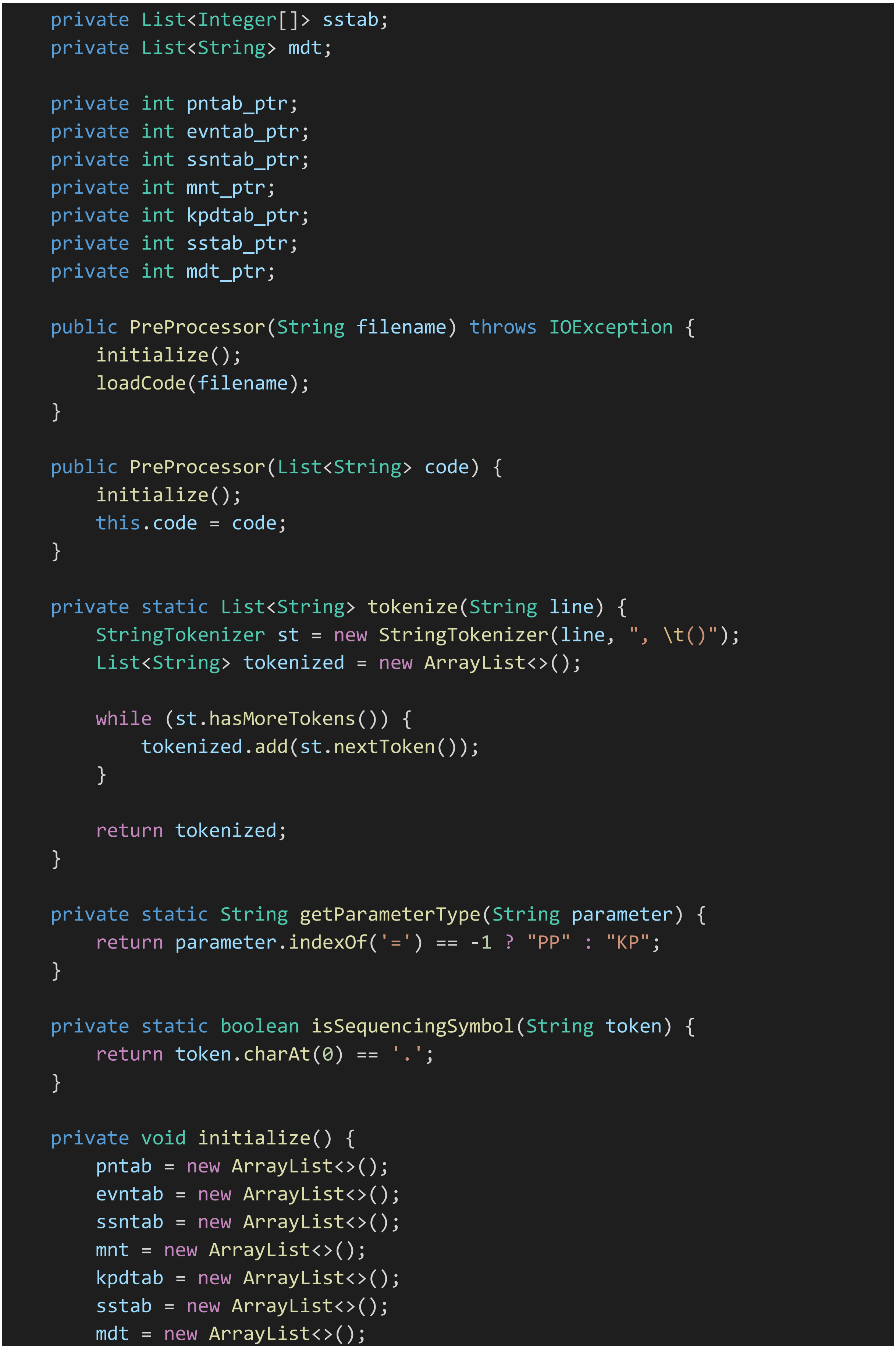
Output :

```
PS D:\MCA\MCA SEM 3\SS> java .\SS_Assembler.java

LC= 200

IC=(IS,09) () (S, 1)

IC=(IS,04) (1) (S, 3)

IC=(IS,01) (1) (S, 2)

IC=(IS,05) (1) (S, 4)

IC=(IS,10) () (S, 5)

IC=(IS,10) () (S, 3)

IC=(DL,02) (C, 3)

IC=(DL,02) (C, 2)
```

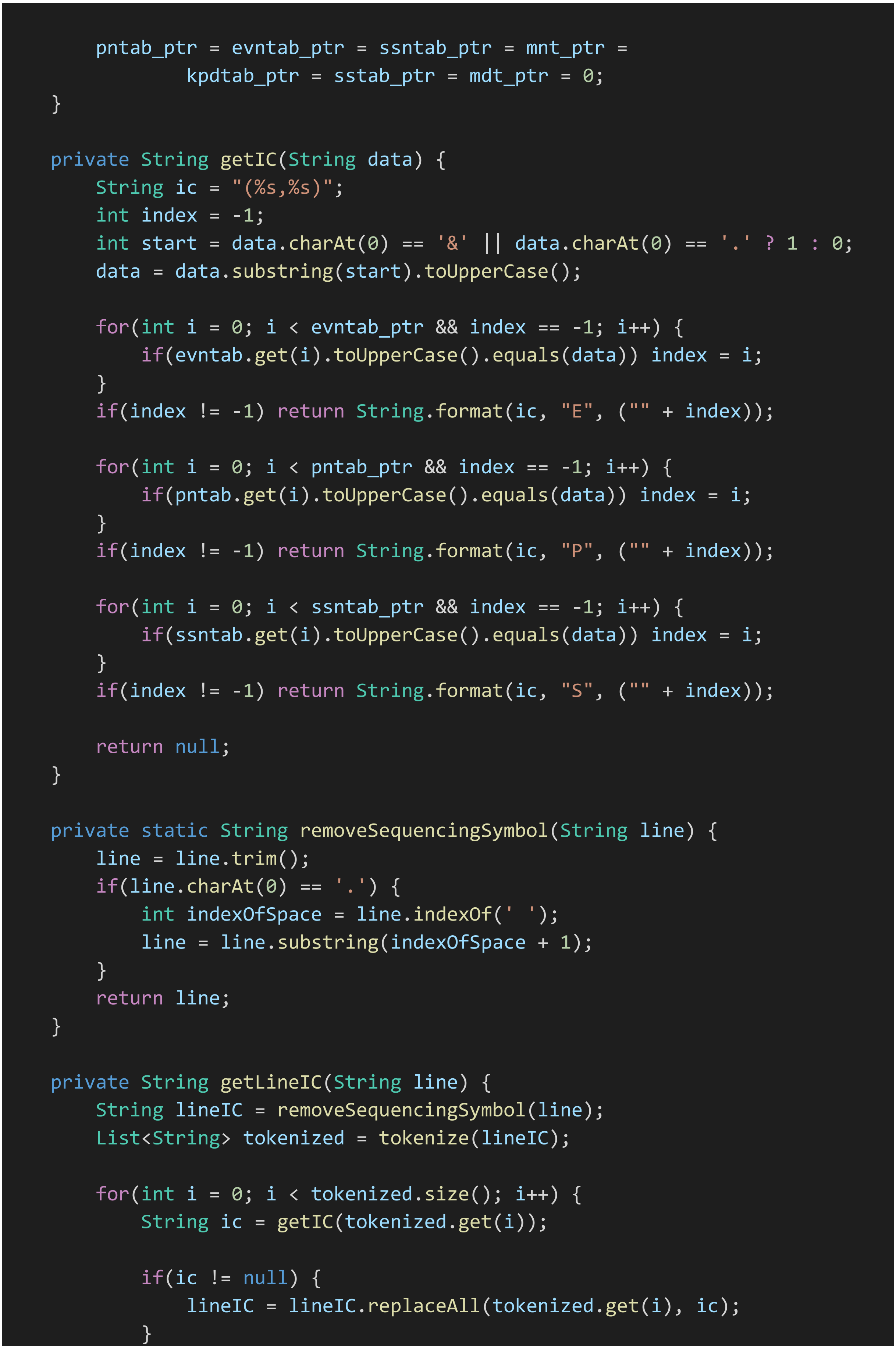
```
IC=(DL,02) (C, 4)


------------->Literal Table


------------->Symbol Table

B - 206

A - 206

RESULT - 207


------------->Pool Table

1

1
```

2. Macro Preprosessor

```
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.StringTokenizer;


public class Macro {

    public static void main(String[] args) throws IOException {

        List<String> input = new ArrayList<>();

        input.add("\tMACRO");
        input.add("\tCLEARMEM &X, &N, &REG=AREG");
        input.add("\tLCL &M");
        input.add("\t&M SET 0");
        input.add("\tMOVER &REG, ='0'");
        input.add(".MORE MOVEM &REG, &X + &M");
        input.add("\t&M SET &M+1");
        input.add("\tAIF (&M NE &N) .MORE");
        input.add("\tMEND");
        input.add("\tMMEND");

        System.out.println("Starting Preprocessing...");

        PreProcessor pr = new PreProcessor(input);

        pr.showCode();
        pr.analyze();
        pr.showTables();

        System.out.println("Ending Preprocessing...");
    }
}

class PreProcessor {
    private List<String> code;

    private List<String> pntab;
    private List<String> evntab;
    private List<String> ssntab;
    private List<MacroData> mnt;
    private List<String[]> kpdtab;
```

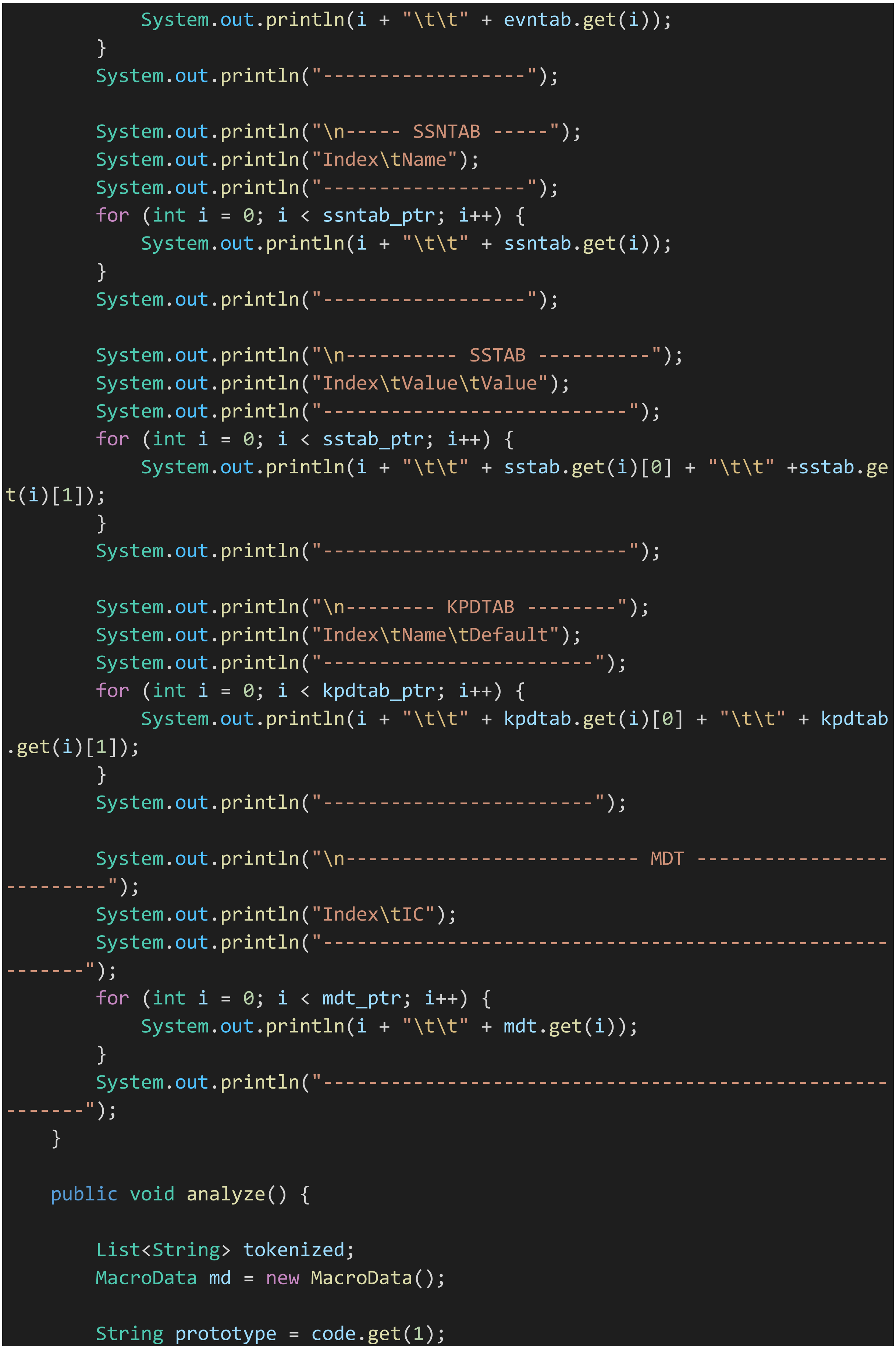


```java
    private List<Integer[]> sstab;
    private List<String> mdt;

    private int pntab_ptr;
    private int evntab_ptr;
    private int ssntab_ptr;
    private int mnt_ptr;
    private int kpdtab_ptr;
    private int sstab_ptr;
    private int mdt_ptr;

    public PreProcessor(String filename) throws IOException {
        initialize();
        loadCode(filename);
    }

    public PreProcessor(List<String> code) {
        initialize();
        this.code = code;
    }

    private static List<String> tokenize(String line) {
        StringTokenizer st = new StringTokenizer(line, ", \t()");
        List<String> tokenized = new ArrayList<>();

        while (st.hasMoreTokens()) {
            tokenized.add(st.nextToken());
        }

        return tokenized;
    }

    private static String getParameterType(String parameter) {
        return parameter.indexOf('=') == -1 ? "PP" : "KP";
    }

    private static boolean isSequencingSymbol(String token) {
        return token.charAt(0) == '.';
    }

    private void initialize() {
        pntab = new ArrayList<>();
        evntab = new ArrayList<>();
        ssntab = new ArrayList<>();
        mnt = new ArrayList<>();
        kpdtab = new ArrayList<>();
        sstab = new ArrayList<>();
        mdt = new ArrayList<>();
```

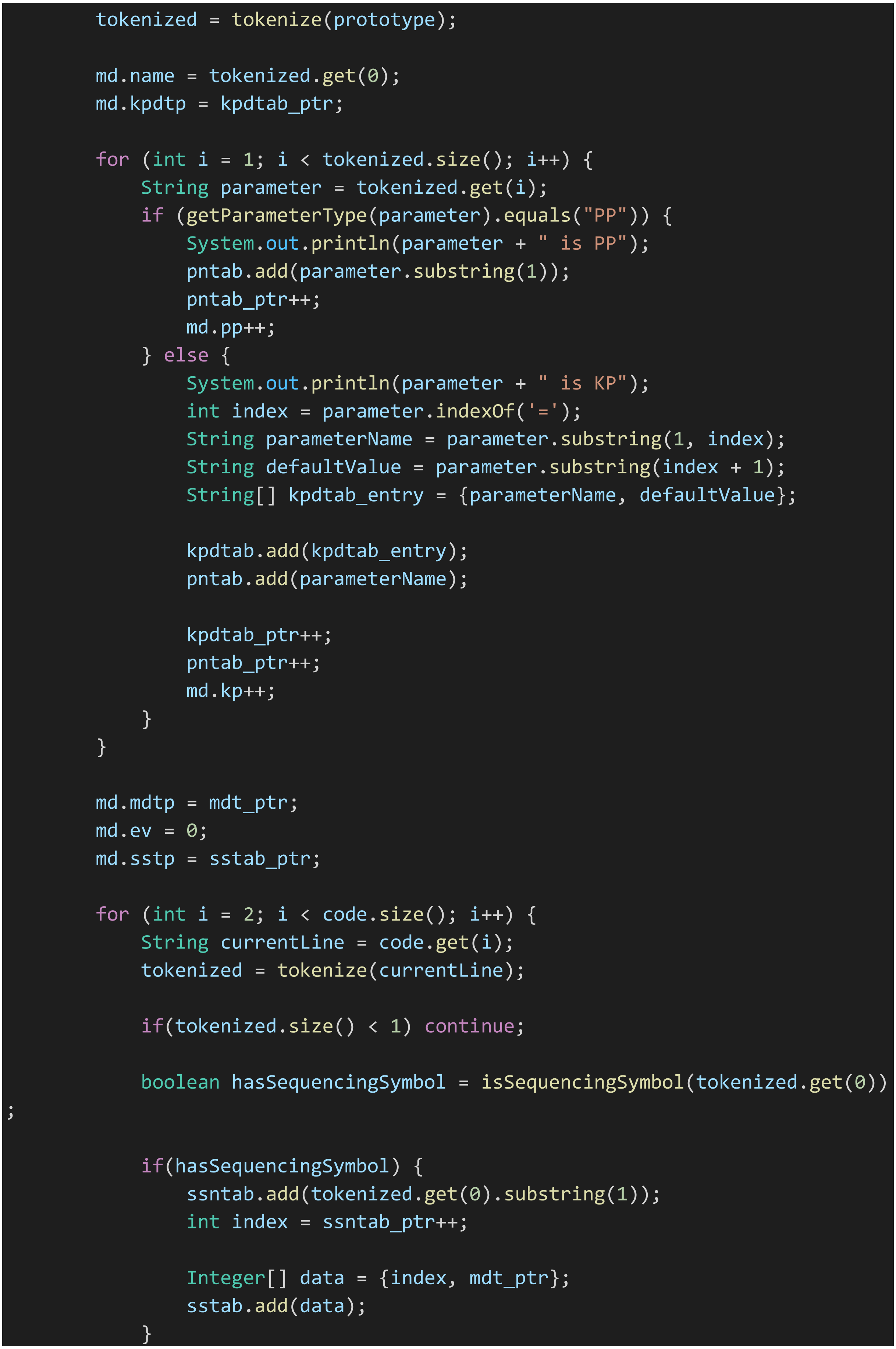


```
    pntab_ptr = evntab_ptr = ssntab_ptr = mnt_ptr =
            kpdtab_ptr = sstab_ptr = mdt_ptr = 0;
}

private String getIC(String data) {
    String ic = "(%s,%s)";
    int index = -1;
    int start = data.charAt(0) == '&' || data.charAt(0) == '.' ? 1 : 0;
    data = data.substring(start).toUpperCase();

    for(int i = 0; i < evntab_ptr && index == -1; i++) {
        if(evntab.get(i).toUpperCase().equals(data)) index = i;
    }
    if(index != -1) return String.format(ic, "E", ("" + index));

    for(int i = 0; i < pntab_ptr && index == -1; i++) {
        if(pntab.get(i).toUpperCase().equals(data)) index = i;
    }
    if(index != -1) return String.format(ic, "P", ("" + index));

    for(int i = 0; i < ssntab_ptr && index == -1; i++) {
        if(ssntab.get(i).toUpperCase().equals(data)) index = i;
    }
    if(index != -1) return String.format(ic, "S", ("" + index));

    return null;
}

private static String removeSequencingSymbol(String line) {
    line = line.trim();
    if(line.charAt(0) == '.') {
        int indexOfSpace = line.indexOf(' ');
        line = line.substring(indexOfSpace + 1);
    }
    return line;
}

private String getLineIC(String line) {
    String lineIC = removeSequencingSymbol(line);
    List<String> tokenized = tokenize(lineIC);

    for(int i = 0; i < tokenized.size(); i++) {
        String ic = getIC(tokenized.get(i));

        if(ic != null) {
            lineIC = lineIC.replaceAll(tokenized.get(i), ic);
        }
```

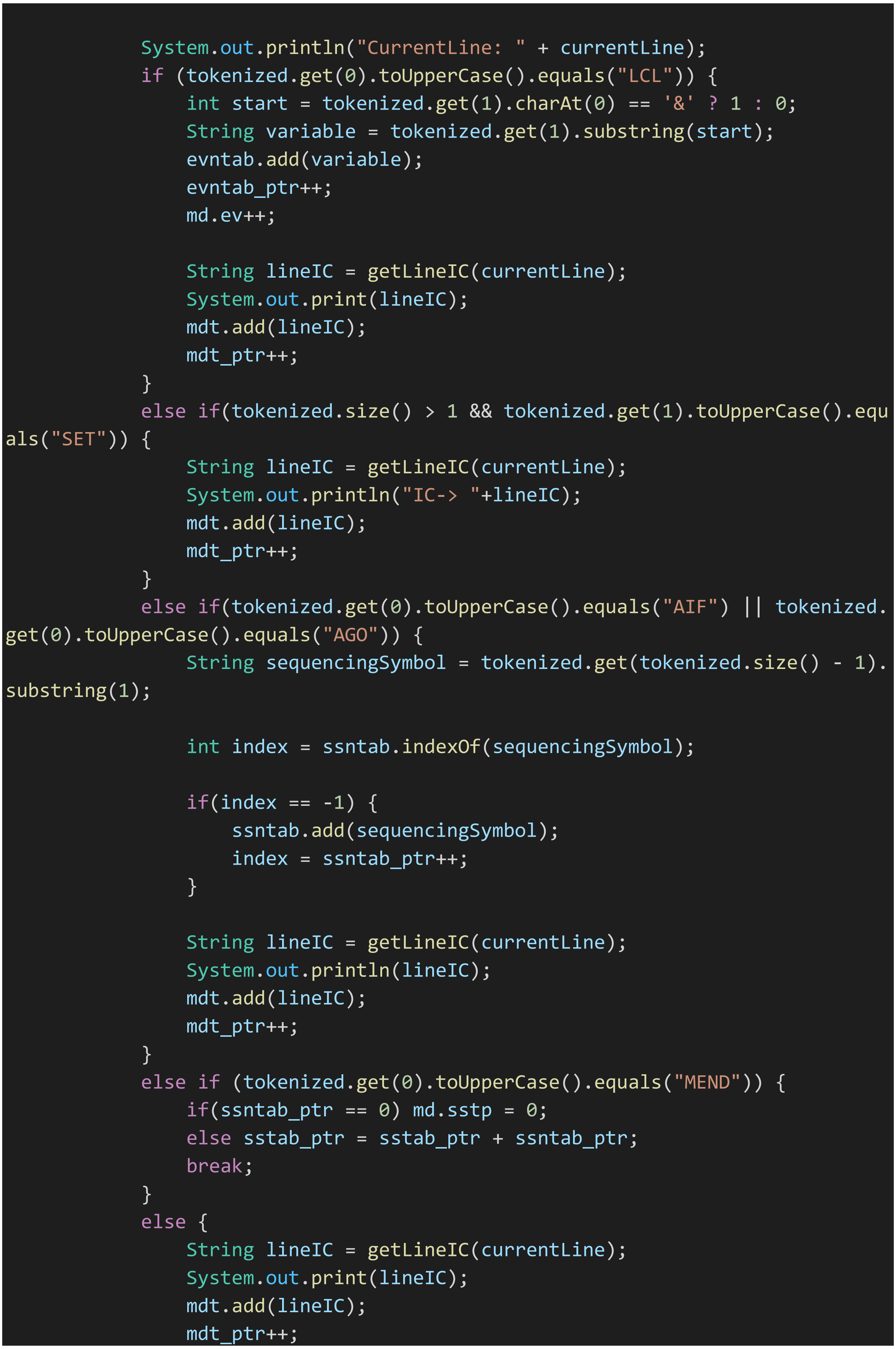
```java
        }
        return lineIC;
    }

    private void loadCode(String filename) throws IOException {
        BufferedReader reader = new BufferedReader(new FileReader(filename));
        code = new ArrayList<>();

        String line;

        while ((line = reader.readLine()) != null) {
            code.add(line);
        }

        if (reader != null) reader.close();
    }

    public void showCode() {
        for (int i = 0; i < code.size(); i++) {
            System.out.println(code.get(i));
        }
    }

    public void showTables() {
        System.out.println("\n------- TABLES -------\n");

        System.out.println("-------------------- MNT --------------------");
        System.out.println("MACRONAME\t#PP\t#KP\t#EV\tMDTP\tKPDTP\tSSTP");
        System.out.println("---------------------------------------------");
        for (int i = 0; i < mnt_ptr; i++) {
            MacroData md = mnt.get(i);
            System.out.println(md.name + "\t" + md.pp + "\t" + md.kp + "\t" + md.ev + "\t" + md.mdtp + "\t\t" + md.kpdtp + "\t\t" + md.sstp);
        }
        System.out.println("---------------------------------------------");


        System.out.println("\n----- PNTAB -----");
        System.out.println("Index\tName");
        System.out.println("-----------------");
        for (int i = 0; i < pntab_ptr; i++) {
            System.out.println(i + "\t\t" + pntab.get(i));
        }
        System.out.println("-----------------");

        System.out.println("\n----- EVNTAB -----");
        System.out.println("Index\tName");
        System.out.println("------------------");
        for (int i = 0; i < evntab_ptr; i++) {
```



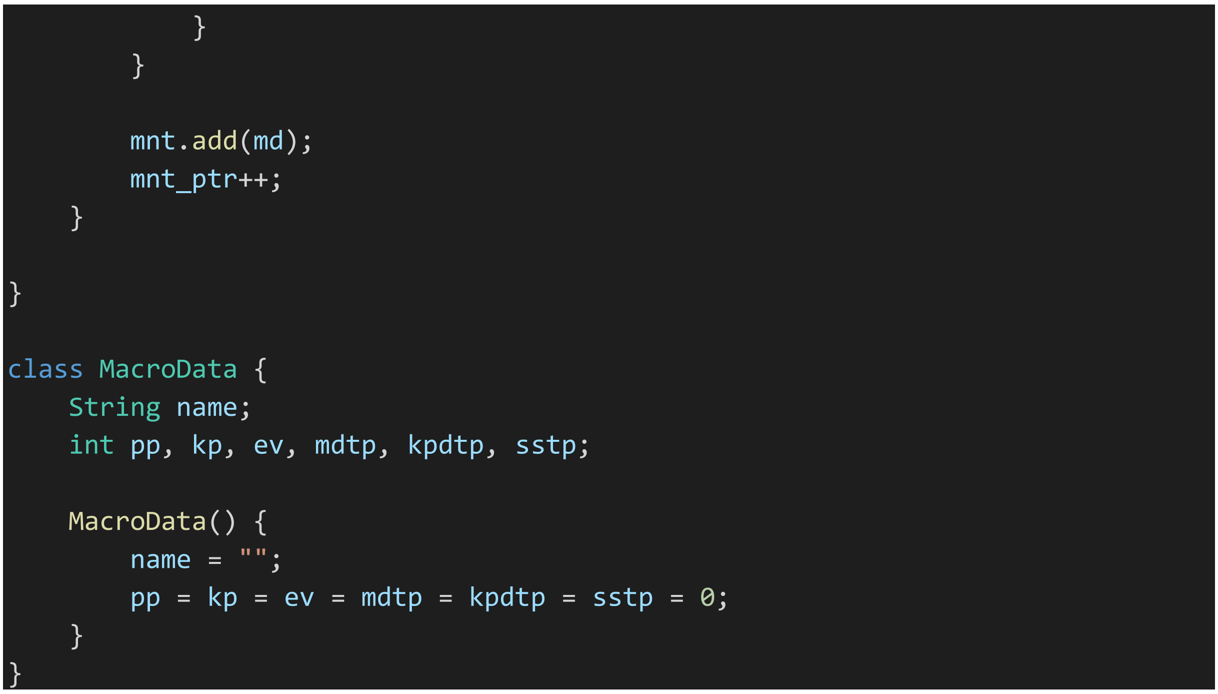
```
            System.out.println(i + "\t\t" + evntab.get(i));
        }
        System.out.println("------------------");

        System.out.println("\n----- SSNTAB -----");
        System.out.println("Index\tName");
        System.out.println("------------------");
        for (int i = 0; i < ssntab_ptr; i++) {
            System.out.println(i + "\t\t" + ssntab.get(i));
        }
        System.out.println("------------------");

        System.out.println("\n---------- SSTAB ----------");
        System.out.println("Index\tValue\tValue");
        System.out.println("---------------------------");
        for (int i = 0; i < sstab_ptr; i++) {
            System.out.println(i + "\t\t" + sstab.get(i)[0] + "\t\t" +sstab.ge
t(i)[1]);
        }
        System.out.println("---------------------------");

        System.out.println("\n-------- KPDTAB --------");
        System.out.println("Index\tName\tDefault");
        System.out.println("------------------------");
        for (int i = 0; i < kpdtab_ptr; i++) {
            System.out.println(i + "\t\t" + kpdtab.get(i)[0] + "\t\t" + kpdtab
.get(i)[1]);
        }
        System.out.println("------------------------");

        System.out.println("\n-------------------------- MDT -----------------
---------");
        System.out.println("Index\tIC");
        System.out.println("--------------------------------------------------
-------");
        for (int i = 0; i < mdt_ptr; i++) {
            System.out.println(i + "\t\t" + mdt.get(i));
        }
        System.out.println("--------------------------------------------------
-------");
    }

    public void analyze() {

        List<String> tokenized;
        MacroData md = new MacroData();

        String prototype = code.get(1);
```



```
        tokenized = tokenize(prototype);

        md.name = tokenized.get(0);
        md.kpdtp = kpdtab_ptr;

        for (int i = 1; i < tokenized.size(); i++) {
            String parameter = tokenized.get(i);
            if (getParameterType(parameter).equals("PP")) {
                System.out.println(parameter + " is PP");
                pntab.add(parameter.substring(1));
                pntab_ptr++;
                md.pp++;
            } else {
                System.out.println(parameter + " is KP");
                int index = parameter.indexOf('=');
                String parameterName = parameter.substring(1, index);
                String defaultValue = parameter.substring(index + 1);
                String[] kpdtab_entry = {parameterName, defaultValue};

                kpdtab.add(kpdtab_entry);
                pntab.add(parameterName);

                kpdtab_ptr++;
                pntab_ptr++;
                md.kp++;
            }
        }

        md.mdtp = mdt_ptr;
        md.ev = 0;
        md.sstp = sstab_ptr;

        for (int i = 2; i < code.size(); i++) {
            String currentLine = code.get(i);
            tokenized = tokenize(currentLine);

            if(tokenized.size() < 1) continue;

            boolean hasSequencingSymbol = isSequencingSymbol(tokenized.get(0))
;

            if(hasSequencingSymbol) {
                ssntab.add(tokenized.get(0).substring(1));
                int index = ssntab_ptr++;

                Integer[] data = {index, mdt_ptr};
                sstab.add(data);
            }
```



```
            System.out.println("CurrentLine: " + currentLine);
            if (tokenized.get(0).toUpperCase().equals("LCL")) {
                int start = tokenized.get(1).charAt(0) == '&' ? 1 : 0;
                String variable = tokenized.get(1).substring(start);
                evntab.add(variable);
                evntab_ptr++;
                md.ev++;

                String lineIC = getLineIC(currentLine);
                System.out.print(lineIC);
                mdt.add(lineIC);
                mdt_ptr++;
            }
            else if(tokenized.size() > 1 && tokenized.get(1).toUpperCase().equals("SET")) {
                String lineIC = getLineIC(currentLine);
                System.out.println("IC-> "+lineIC);
                mdt.add(lineIC);
                mdt_ptr++;
            }
            else if(tokenized.get(0).toUpperCase().equals("AIF") || tokenized.get(0).toUpperCase().equals("AGO")) {
                String sequencingSymbol = tokenized.get(tokenized.size() - 1).substring(1);

                int index = ssntab.indexOf(sequencingSymbol);

                if(index == -1) {
                    ssntab.add(sequencingSymbol);
                    index = ssntab_ptr++;
                }

                String lineIC = getLineIC(currentLine);
                System.out.println(lineIC);
                mdt.add(lineIC);
                mdt_ptr++;
            }
            else if (tokenized.get(0).toUpperCase().equals("MEND")) {
                if(ssntab_ptr == 0) md.sstp = 0;
                else sstab_ptr = sstab_ptr + ssntab_ptr;
                break;
            }
            else {
                String lineIC = getLineIC(currentLine);
                System.out.print(lineIC);
                mdt.add(lineIC);
                mdt_ptr++;
```



```java
            }
        }

        mnt.add(md);
        mnt_ptr++;
    }

}

class MacroData {
    String name;
    int pp, kp, ev, mdtp, kpdtp, sstp;

    MacroData() {
        name = "";
        pp = kp = ev = mdtp = kpdtp = sstp = 0;
    }
}
```

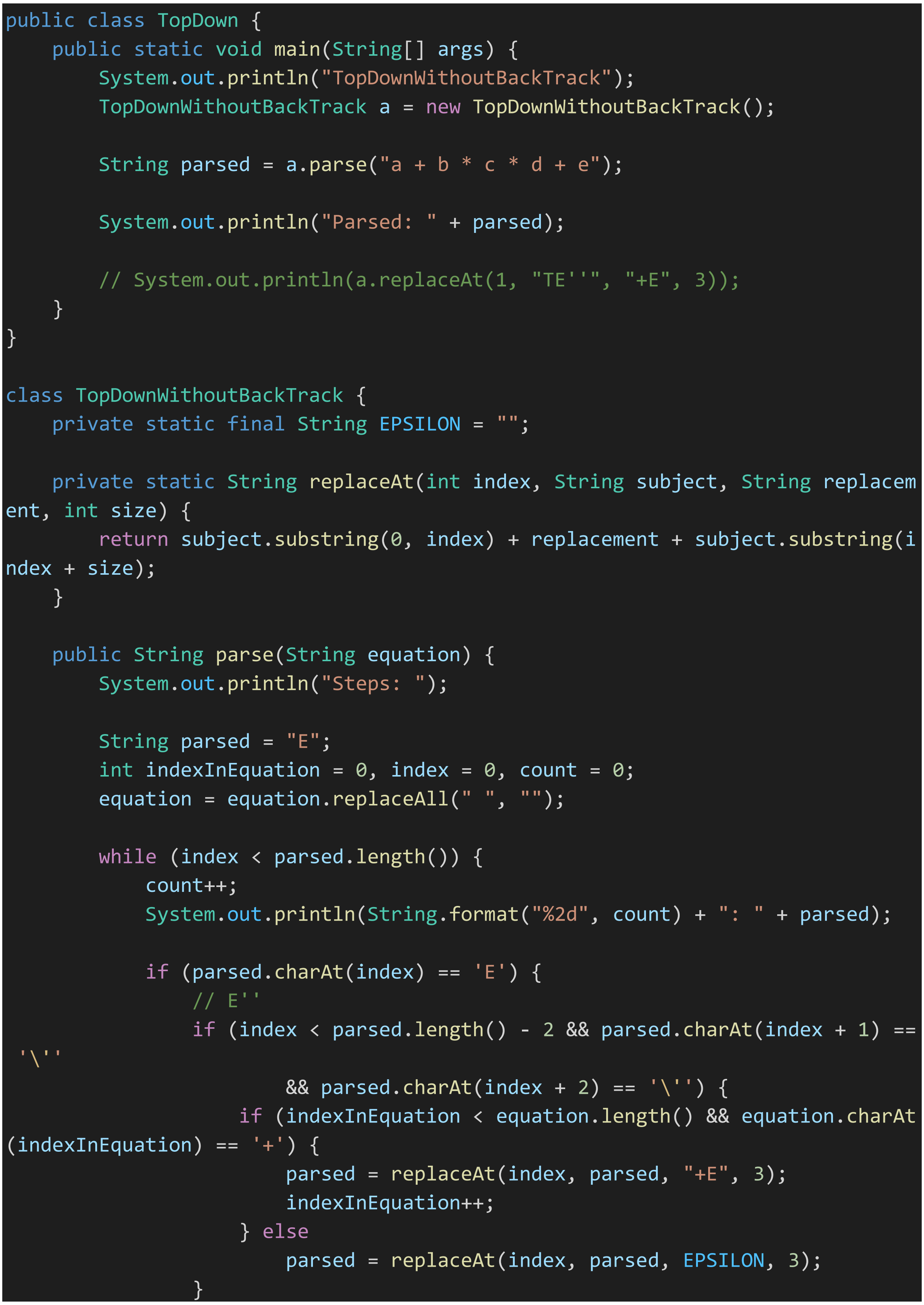
Output :

```
PS D:\MCA\MCA SEM 3\SS> java .\Macro.java
Starting Preprocessing...
        MACRO
        CLEARMEM &X, &N, &REG=AREG
        LCL &M
        &M SET 0
        MOVER &REG, ='0'
.MORE MOVEM &REG, &X + &M
        &M SET &M+1
        AIF (&M NE &N) .MORE
        MEND
        MMEND
&X is PP
&N is PP
&REG=AREG is KP
CurrentLine:    LCL &M
LCL (E,0)CurrentLine:   &M SET 0
IC-> (E,0) SET 0
CurrentLine:    MOVER &REG, ='0'
```

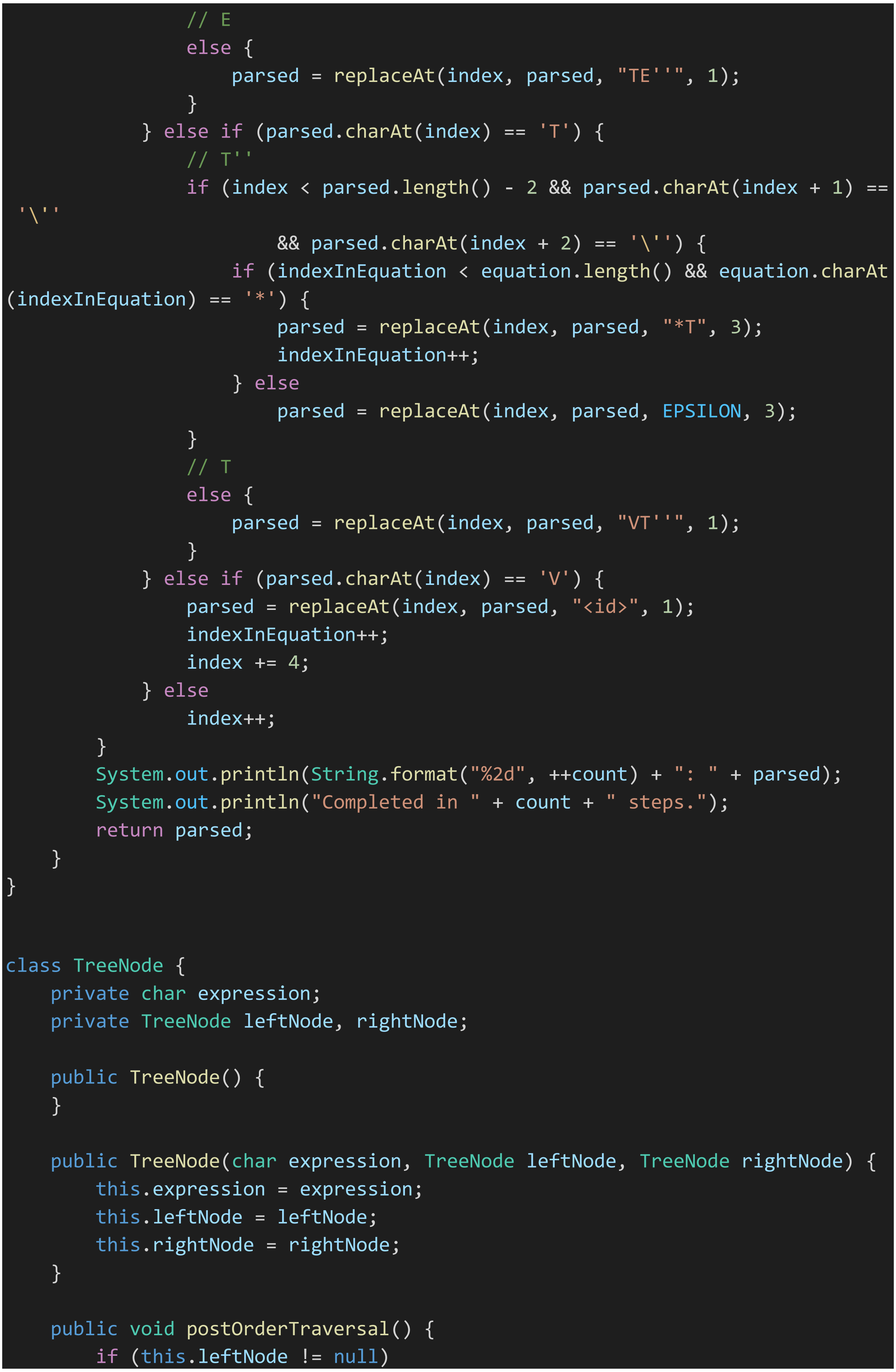
```
MOVER (P,2), ='0'CurrentLine: .MORE MOVEM &REG, &X + &M
MOVEM (P,2), (P,0) + (E,0)CurrentLine:  &M SET &M+1
IC-> (E,0) SET (E,0)+1
CurrentLine:    AIF (&M NE &N) .MORE
AIF ((E,0) NE (P,1)) (S,0)
CurrentLine:    MEND

------- TABLES -------

-------------------- MNT --------------------
MACRONAME      #PP     #KP     #EV     MDTP    KPDTP   SSTP
--------------------------------------------
CLEARMEM       2       1       1       0           0           0
--------------------------------------------

----- PNTAB -----
Index   Name
-----------------
0           X
1           N
2           REG
-----------------

----- EVNTAB -----
Index   Name
------------------
0           M
------------------

----- SSNTAB -----
Index   Name
------------------
0           MORE
------------------
```

```
---------- SSTAB ----------
Index   Value   Value
--------------------------
0           0           3
--------------------------

-------- KPDTAB --------
Index   Name    Default
-----------------------
0           REG          AREG
-----------------------

------------------------- MDT -------------------------
Index   IC
------------------------------------------------------
0           LCL (E,0)
1           (E,0) SET 0
2           MOVER (P,2), ='0'
3           MOVEM (P,2), (P,0) + (E,0)
4           (E,0) SET (E,0)+1
5           AIF ((E,0) NE (P,1)) (S,0)
------------------------------------------------------
Ending Preprocessing...
```

3. Top Down Without Backtracking



```
public class TopDown {
    public static void main(String[] args) {
        System.out.println("TopDownWithoutBackTrack");
        TopDownWithoutBackTrack a = new TopDownWithoutBackTrack();

        String parsed = a.parse("a + b * c * d + e");

        System.out.println("Parsed: " + parsed);

        // System.out.println(a.replaceAt(1, "TE''", "+E", 3));
    }
}

class TopDownWithoutBackTrack {
    private static final String EPSILON = "";

    private static String replaceAt(int index, String subject, String replacem
ent, int size) {
        return subject.substring(0, index) + replacement + subject.substring(i
ndex + size);
    }

    public String parse(String equation) {
        System.out.println("Steps: ");

        String parsed = "E";
        int indexInEquation = 0, index = 0, count = 0;
        equation = equation.replaceAll(" ", "");

        while (index < parsed.length()) {
            count++;
            System.out.println(String.format("%2d", count) + ": " + parsed);

            if (parsed.charAt(index) == 'E') {
                // E''
                if (index < parsed.length() - 2 && parsed.charAt(index + 1) ==
 '\''
                        && parsed.charAt(index + 2) == '\'') {
                    if (indexInEquation < equation.length() && equation.charAt
(indexInEquation) == '+') {
                        parsed = replaceAt(index, parsed, "+E", 3);
                        indexInEquation++;
                    } else
                        parsed = replaceAt(index, parsed, EPSILON, 3);
                }
```



```
                // E
                else {
                    parsed = replaceAt(index, parsed, "TE''", 1);
                }
            } else if (parsed.charAt(index) == 'T') {
                // T''
                if (index < parsed.length() - 2 && parsed.charAt(index + 1) ==
 '\''
                        && parsed.charAt(index + 2) == '\'') {
                    if (indexInEquation < equation.length() && equation.charAt
(indexInEquation) == '*') {
                        parsed = replaceAt(index, parsed, "*T", 3);
                        indexInEquation++;
                    } else
                        parsed = replaceAt(index, parsed, EPSILON, 3);
                }
                // T
                else {
                    parsed = replaceAt(index, parsed, "VT''", 1);
                }
            } else if (parsed.charAt(index) == 'V') {
                parsed = replaceAt(index, parsed, "<id>", 1);
                indexInEquation++;
                index += 4;
            } else
                index++;
        }
        System.out.println(String.format("%2d", ++count) + ": " + parsed);
        System.out.println("Completed in " + count + " steps.");
        return parsed;
    }
}


class TreeNode {
    private char expression;
    private TreeNode leftNode, rightNode;

    public TreeNode() {
    }

    public TreeNode(char expression, TreeNode leftNode, TreeNode rightNode) {
        this.expression = expression;
        this.leftNode = leftNode;
        this.rightNode = rightNode;
    }

    public void postOrderTraversal() {
        if (this.leftNode != null)
```

```
            leftNode.postOrderTraversal();

        if (this.rightNode != null)
            rightNode.postOrderTraversal();

        System.out.print(this.expression);
    }
}
```

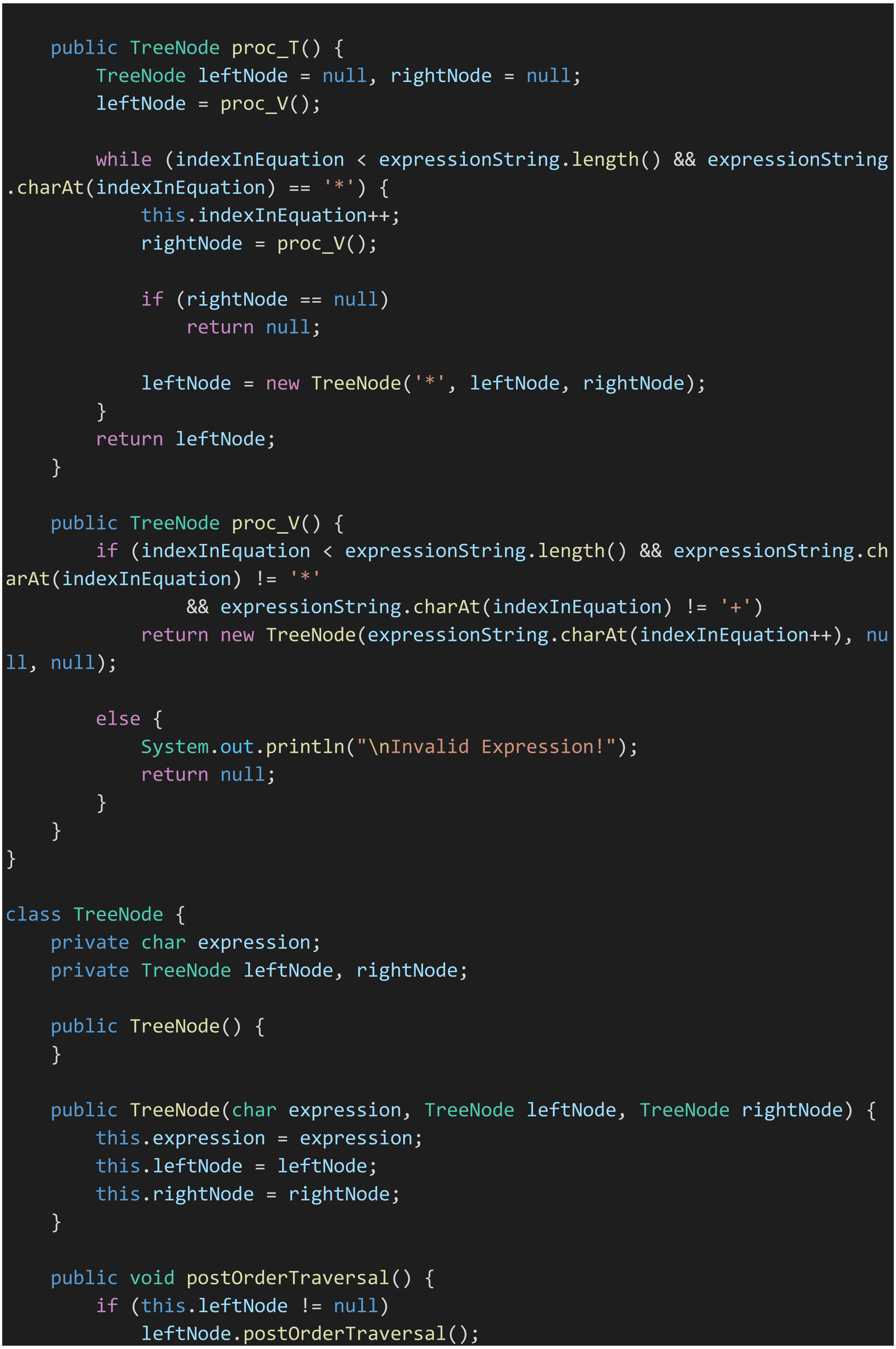
Output :

```
PS D:\MCA\MCA SEM 3\SS\Parsers> java .\TopDown.java
TopDownWithoutBackTrack
Steps:
 1: E
 2: TE"
 3: VT"E"
 4: <id>T"E"
 5: <id>E"
 6: <id>+E
 7: <id>+E
 8: <id>+TE"
 9: <id>+VT"E"
10: <id>+<id>T"E"
11: <id>+<id>*TE"
12: <id>+<id>*TE"
13: <id>+<id>*VT"E"
14: <id>+<id>*<id>T"E"
15: <id>+<id>*<id>*TE"
16: <id>+<id>*<id>*TE"
17: <id>+<id>*<id>*VT"E"
18: <id>+<id>*<id>*<id>T"E"
19: <id>+<id>*<id>*<id>E"
20: <id>+<id>*<id>*<id>+E
21: <id>+<id>*<id>*<id>+E
22: <id>+<id>*<id>*<id>+TE"
23: <id>+<id>*<id>*<id>+VT"E"
```

```
24: <id>+<id>*<id>*<id>+<id>T''E''
25: <id>+<id>*<id>*<id>+<id>E''
26: <id>+<id>*<id>*<id>+<id>
Completed in 26 steps.
Parsed: <id>+<id>*<id>*<id>+<id>
```

4 . Recursive Decent Parser

```java
import java.util.Scanner;

public class RD {
    public static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {
        System.out.print("Enter the Expression: ");
        String expression = scanner.nextLine();
        RecursiveDescentParser recursiveDescentParsing = new RecursiveDescentParser(expression);
        TreeNode rootNode;

        rootNode = recursiveDescentParsing.proc_E();

        if (rootNode != null) {
            rootNode.postOrderTraversal();
        }
    }
}

class RecursiveDescentParser {
    private String expressionString;
    private int indexInEquation = 0;

    public RecursiveDescentParser(String expressionString) {
        this.expressionString = expressionString;
        this.indexInEquation = 0;
    }

    public TreeNode proc_E() {
        TreeNode leftNode = null, rightNode = null;
        leftNode = proc_T();

        while (indexInEquation < expressionString.length() && expressionString.charAt(indexInEquation) == '+') {
            this.indexInEquation++;
            rightNode = proc_T();

            if (rightNode == null)
                return null;

            leftNode = new TreeNode('+', leftNode, rightNode);
        }
        return leftNode;
    }
```



```
    public TreeNode proc_T() {
        TreeNode leftNode = null, rightNode = null;
        leftNode = proc_V();

        while (indexInEquation < expressionString.length() && expressionString
.charAt(indexInEquation) == '*') {
            this.indexInEquation++;
            rightNode = proc_V();

            if (rightNode == null)
                return null;

            leftNode = new TreeNode('*', leftNode, rightNode);
        }
        return leftNode;
    }

    public TreeNode proc_V() {
        if (indexInEquation < expressionString.length() && expressionString.ch
arAt(indexInEquation) != '*'
                && expressionString.charAt(indexInEquation) != '+')
            return new TreeNode(expressionString.charAt(indexInEquation++), nu
ll, null);

        else {
            System.out.println("\nInvalid Expression!");
            return null;
        }
    }
}

class TreeNode {
    private char expression;
    private TreeNode leftNode, rightNode;

    public TreeNode() {
    }

    public TreeNode(char expression, TreeNode leftNode, TreeNode rightNode) {
        this.expression = expression;
        this.leftNode = leftNode;
        this.rightNode = rightNode;
    }

    public void postOrderTraversal() {
        if (this.leftNode != null)
            leftNode.postOrderTraversal();
        if (this.rightNode != null)
            rightNode.postOrderTraversal();

        System.out.print(this.expression);
    }
}
```

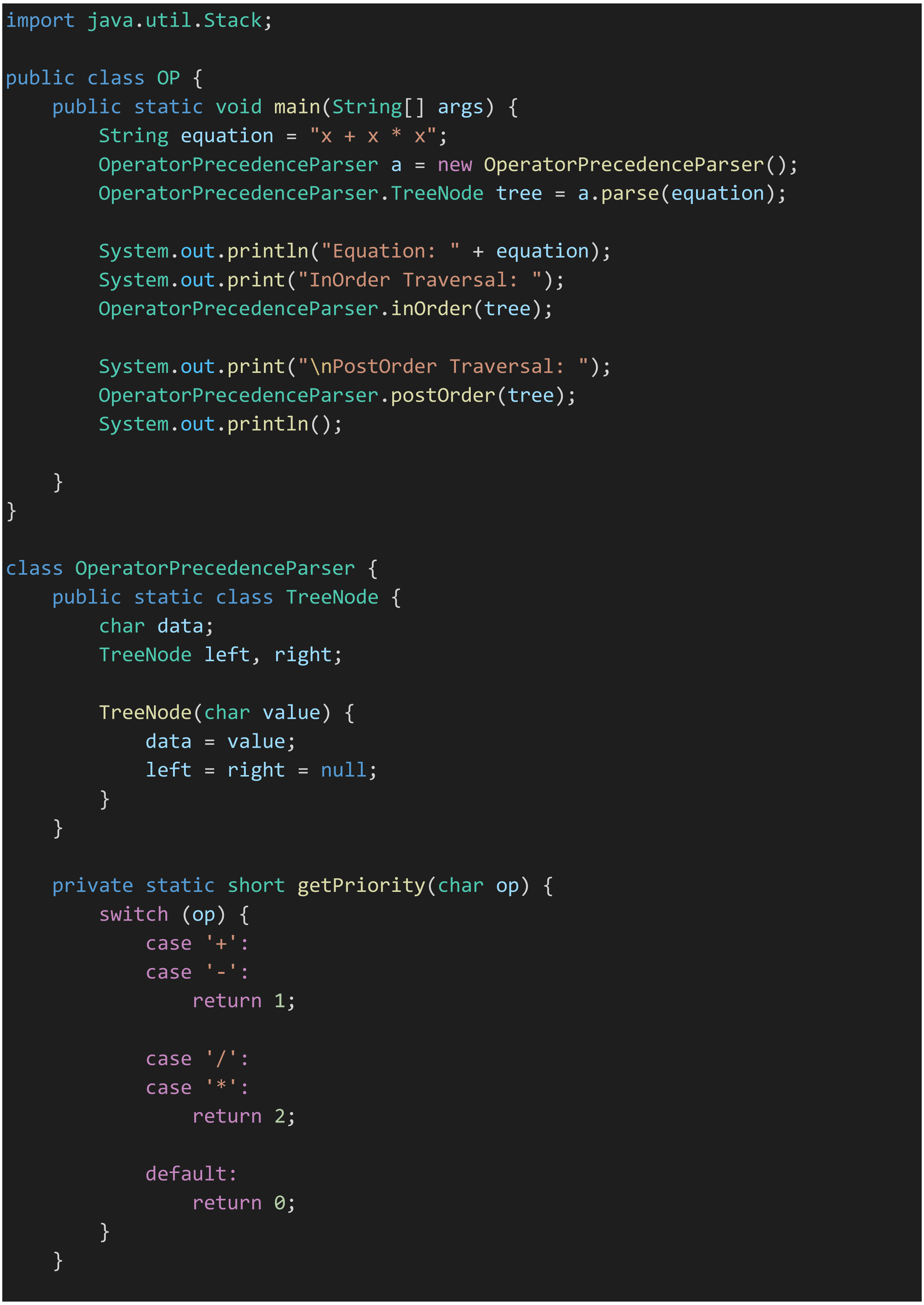
Output:

PS D:\MCA\MCA SEM 3\SS\Parsers> java .\RD.java

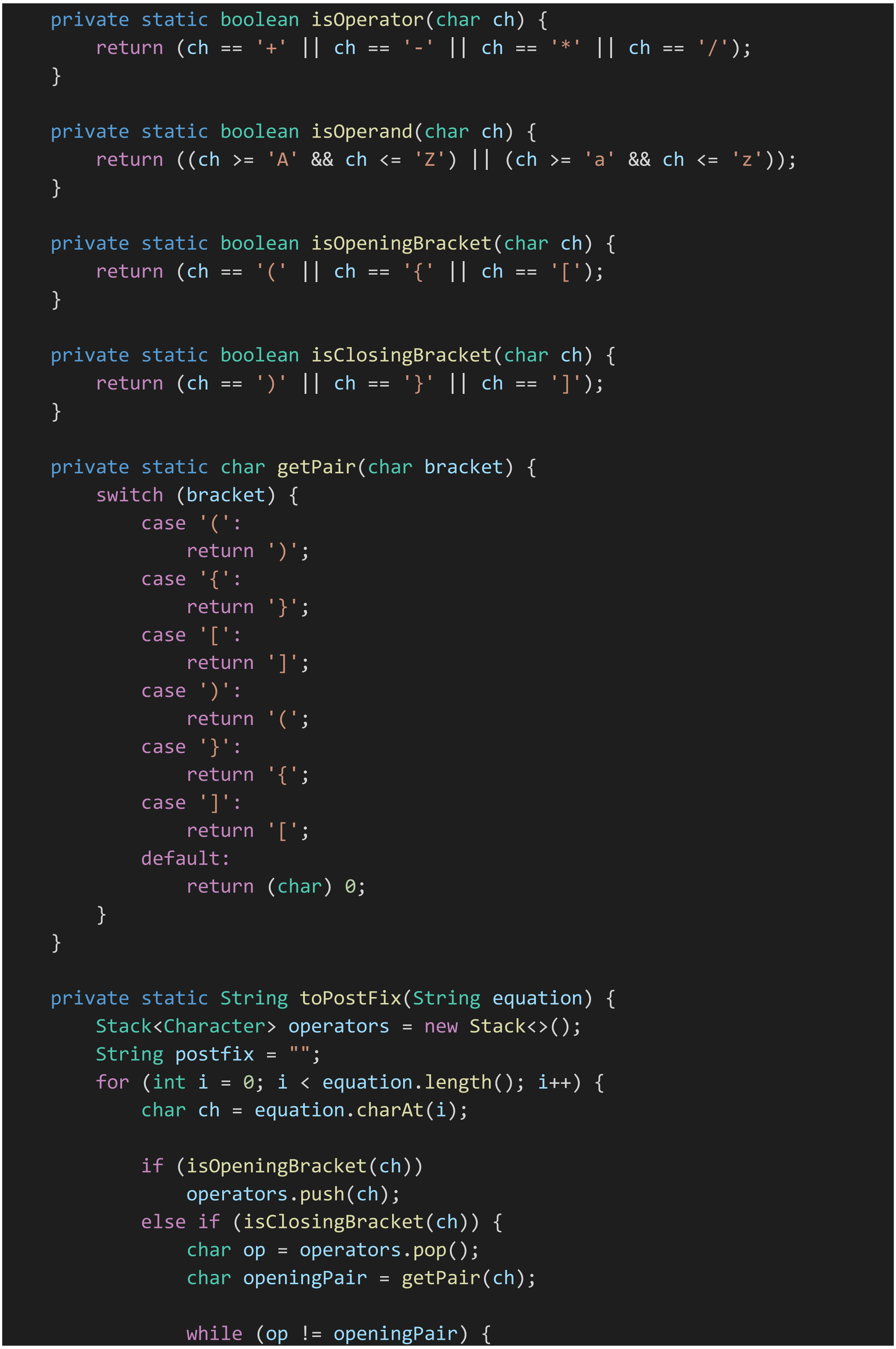
Enter the Expression: x+x*x

xxx*+

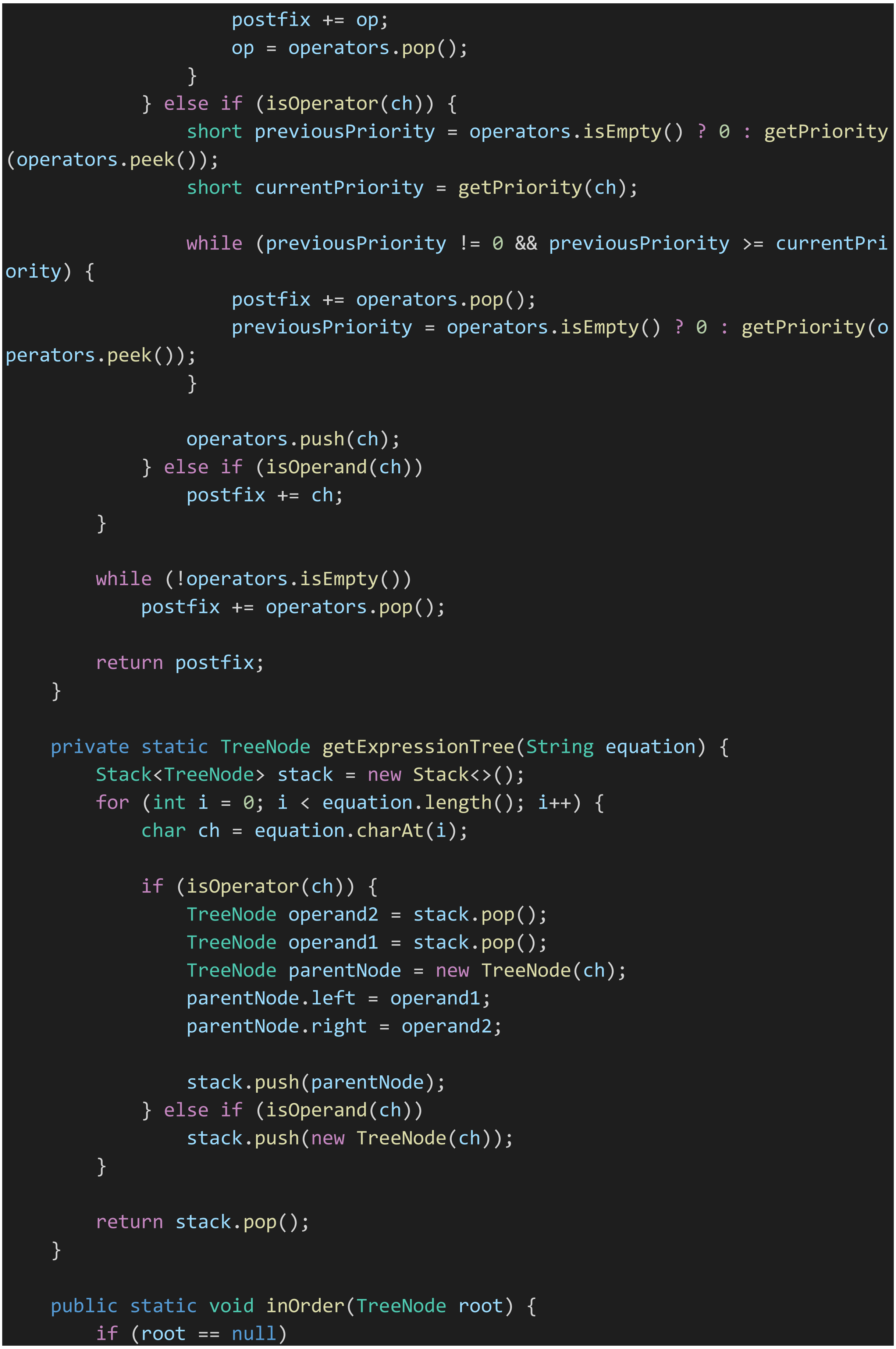
5 . Operator Precedence Parser



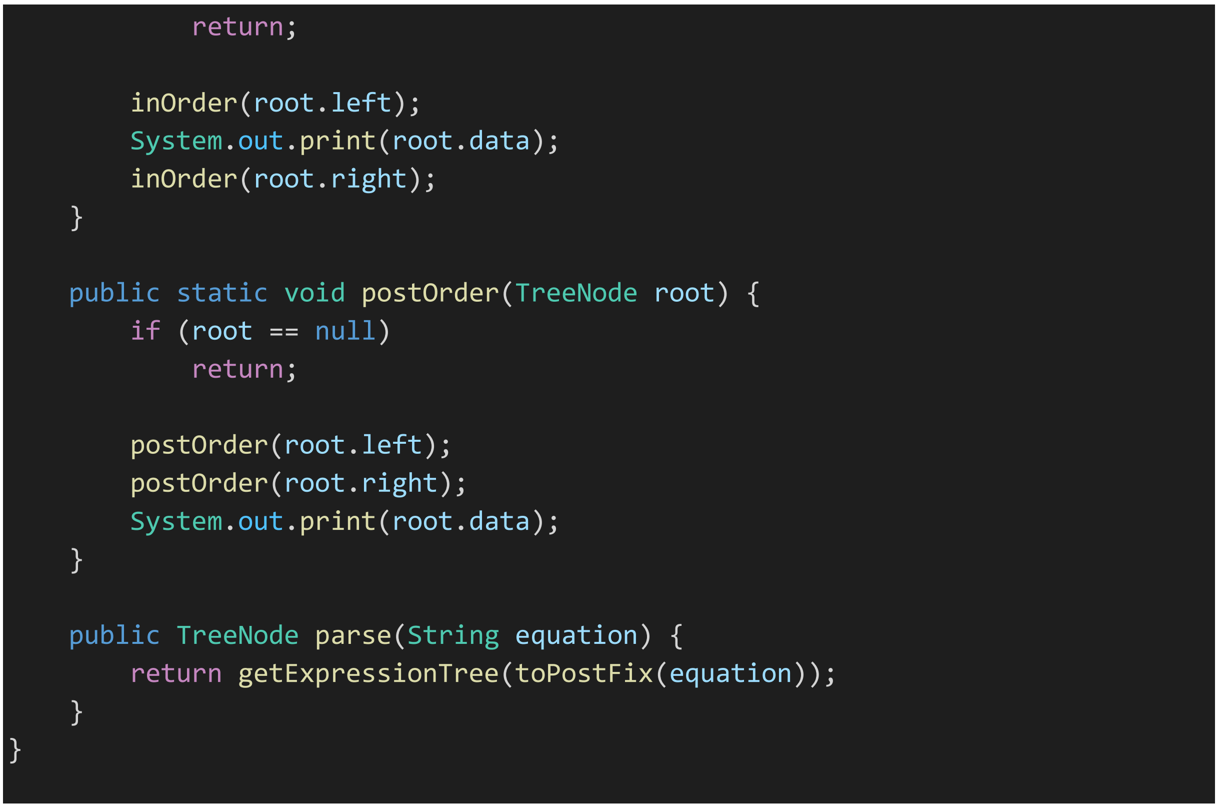
```
import java.util.Stack;

public class OP {
    public static void main(String[] args) {
        String equation = "x + x * x";
        OperatorPrecedenceParser a = new OperatorPrecedenceParser();
        OperatorPrecedenceParser.TreeNode tree = a.parse(equation);

        System.out.println("Equation: " + equation);
        System.out.print("InOrder Traversal: ");
        OperatorPrecedenceParser.inOrder(tree);

        System.out.print("\nPostOrder Traversal: ");
        OperatorPrecedenceParser.postOrder(tree);
        System.out.println();

    }
}

class OperatorPrecedenceParser {
    public static class TreeNode {
        char data;
        TreeNode left, right;

        TreeNode(char value) {
            data = value;
            left = right = null;
        }
    }

    private static short getPriority(char op) {
        switch (op) {
            case '+':
            case '-':
                return 1;

            case '/':
            case '*':
                return 2;

            default:
                return 0;
        }
    }
```



```java
    private static boolean isOperator(char ch) {
        return (ch == '+' || ch == '-' || ch == '*' || ch == '/');
    }

    private static boolean isOperand(char ch) {
        return ((ch >= 'A' && ch <= 'Z') || (ch >= 'a' && ch <= 'z'));
    }

    private static boolean isOpeningBracket(char ch) {
        return (ch == '(' || ch == '{' || ch == '[');
    }

    private static boolean isClosingBracket(char ch) {
        return (ch == ')' || ch == '}' || ch == ']');
    }

    private static char getPair(char bracket) {
        switch (bracket) {
            case '(':
                return ')';
            case '{':
                return '}';
            case '[':
                return ']';
            case ')':
                return '(';
            case '}':
                return '{';
            case ']':
                return '[';
            default:
                return (char) 0;
        }
    }

    private static String toPostFix(String equation) {
        Stack<Character> operators = new Stack<>();
        String postfix = "";
        for (int i = 0; i < equation.length(); i++) {
            char ch = equation.charAt(i);

            if (isOpeningBracket(ch))
                operators.push(ch);
            else if (isClosingBracket(ch)) {
                char op = operators.pop();
                char openingPair = getPair(ch);

                while (op != openingPair) {
```



```
                    postfix += op;
                    op = operators.pop();
                }
            } else if (isOperator(ch)) {
                short previousPriority = operators.isEmpty() ? 0 : getPriority
(operators.peek());
                short currentPriority = getPriority(ch);

                while (previousPriority != 0 && previousPriority >= currentPri
ority) {
                    postfix += operators.pop();
                    previousPriority = operators.isEmpty() ? 0 : getPriority(o
perators.peek());
                }

                operators.push(ch);
            } else if (isOperand(ch))
                postfix += ch;
        }

        while (!operators.isEmpty())
            postfix += operators.pop();

        return postfix;
    }

    private static TreeNode getExpressionTree(String equation) {
        Stack<TreeNode> stack = new Stack<>();
        for (int i = 0; i < equation.length(); i++) {
            char ch = equation.charAt(i);

            if (isOperator(ch)) {
                TreeNode operand2 = stack.pop();
                TreeNode operand1 = stack.pop();
                TreeNode parentNode = new TreeNode(ch);
                parentNode.left = operand1;
                parentNode.right = operand2;

                stack.push(parentNode);
            } else if (isOperand(ch))
                stack.push(new TreeNode(ch));
        }

        return stack.pop();
    }

    public static void inOrder(TreeNode root) {
        if (root == null)
```

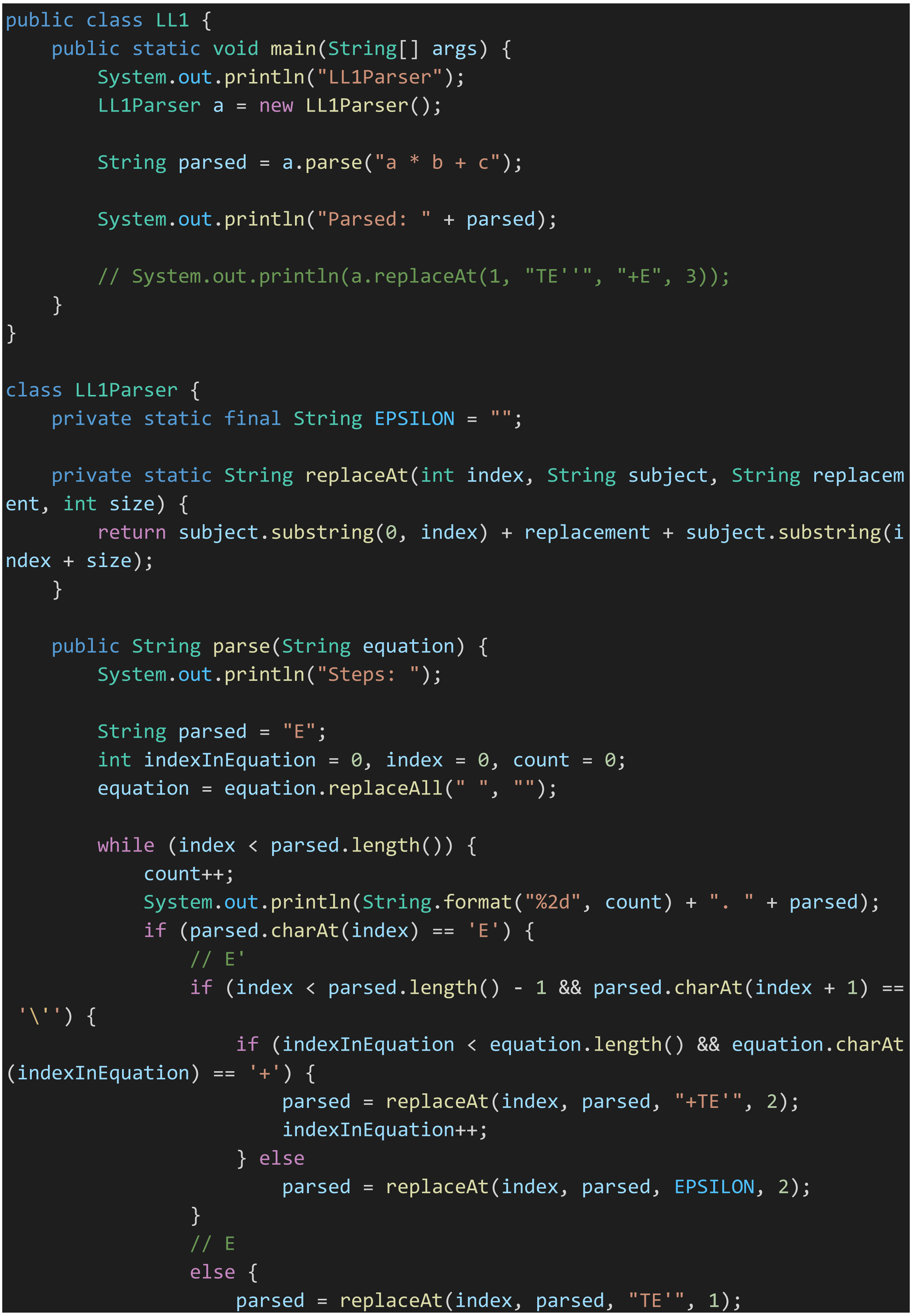


```
            return;

        inOrder(root.left);
        System.out.print(root.data);
        inOrder(root.right);
    }

    public static void postOrder(TreeNode root) {
        if (root == null)
            return;

        postOrder(root.left);
        postOrder(root.right);
        System.out.print(root.data);
    }

    public TreeNode parse(String equation) {
        return getExpressionTree(toPostFix(equation));
    }
}
```

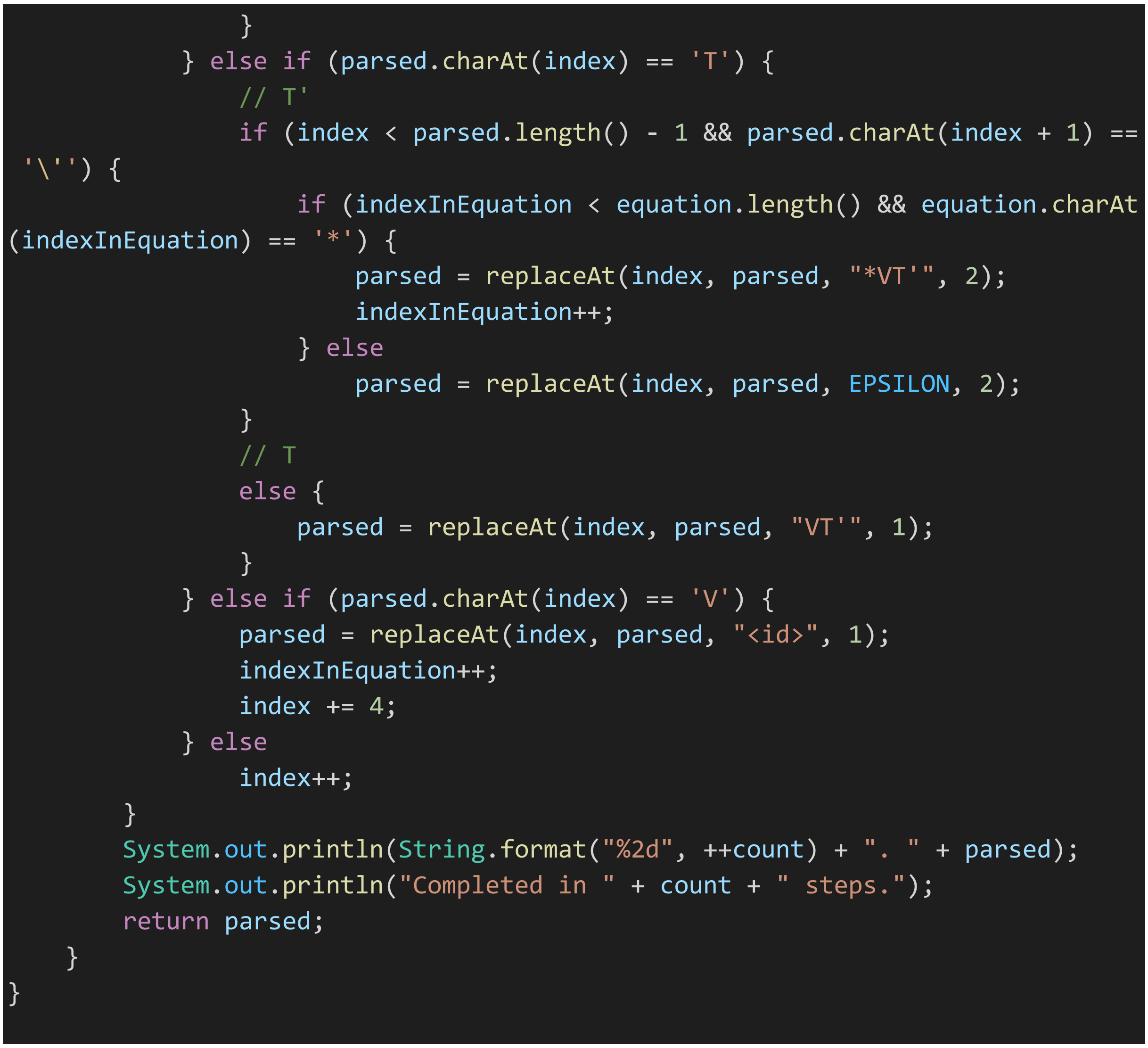
Output:

```
PS D:\MCA\MCA SEM 3\SS\Parsers> java .\OP.java
Equation: x + x * x
InOrder Traversal: x+x*x
PostOrder Traversal: xxx*+
```

6. LL1 Parser



```
public class LL1 {
    public static void main(String[] args) {
        System.out.println("LL1Parser");
        LL1Parser a = new LL1Parser();

        String parsed = a.parse("a * b + c");

        System.out.println("Parsed: " + parsed);

        // System.out.println(a.replaceAt(1, "TE''", "+E", 3));
    }
}

class LL1Parser {
    private static final String EPSILON = "";

    private static String replaceAt(int index, String subject, String replacem
ent, int size) {
        return subject.substring(0, index) + replacement + subject.substring(i
ndex + size);
    }

    public String parse(String equation) {
        System.out.println("Steps: ");

        String parsed = "E";
        int indexInEquation = 0, index = 0, count = 0;
        equation = equation.replaceAll(" ", "");

        while (index < parsed.length()) {
            count++;
            System.out.println(String.format("%2d", count) + ". " + parsed);
            if (parsed.charAt(index) == 'E') {
                // E'
                if (index < parsed.length() - 1 && parsed.charAt(index + 1) ==
 '\'') {
                    if (indexInEquation < equation.length() && equation.charAt
(indexInEquation) == '+') {
                        parsed = replaceAt(index, parsed, "+TE'", 2);
                        indexInEquation++;
                    } else
                        parsed = replaceAt(index, parsed, EPSILON, 2);
                }
                // E
                else {
                    parsed = replaceAt(index, parsed, "TE'", 1);
```



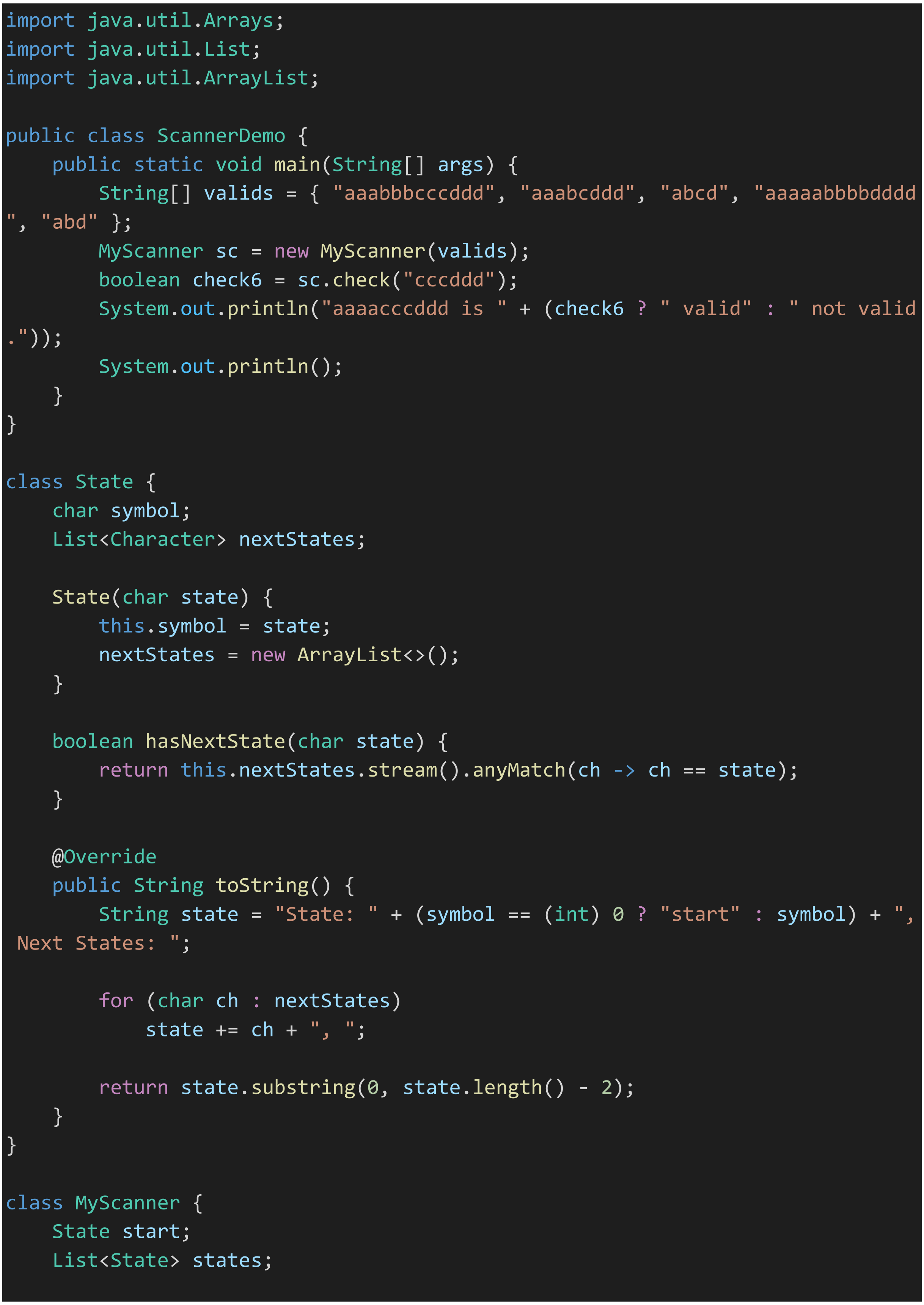
```
        }
      } else if (parsed.charAt(index) == 'T') {
        // T'
        if (index < parsed.length() - 1 && parsed.charAt(index + 1) ==
 '\'') {
          if (indexInEquation < equation.length() && equation.charAt
(indexInEquation) == '*') {
            parsed = replaceAt(index, parsed, "*VT'", 2);
            indexInEquation++;
          } else
            parsed = replaceAt(index, parsed, EPSILON, 2);
        }
        // T
        else {
          parsed = replaceAt(index, parsed, "VT'", 1);
        }
      } else if (parsed.charAt(index) == 'V') {
        parsed = replaceAt(index, parsed, "<id>", 1);
        indexInEquation++;
        index += 4;
      } else
        index++;
    }
    System.out.println(String.format("%2d", ++count) + ". " + parsed);
    System.out.println("Completed in " + count + " steps.");
    return parsed;
  }
}
```

Output :

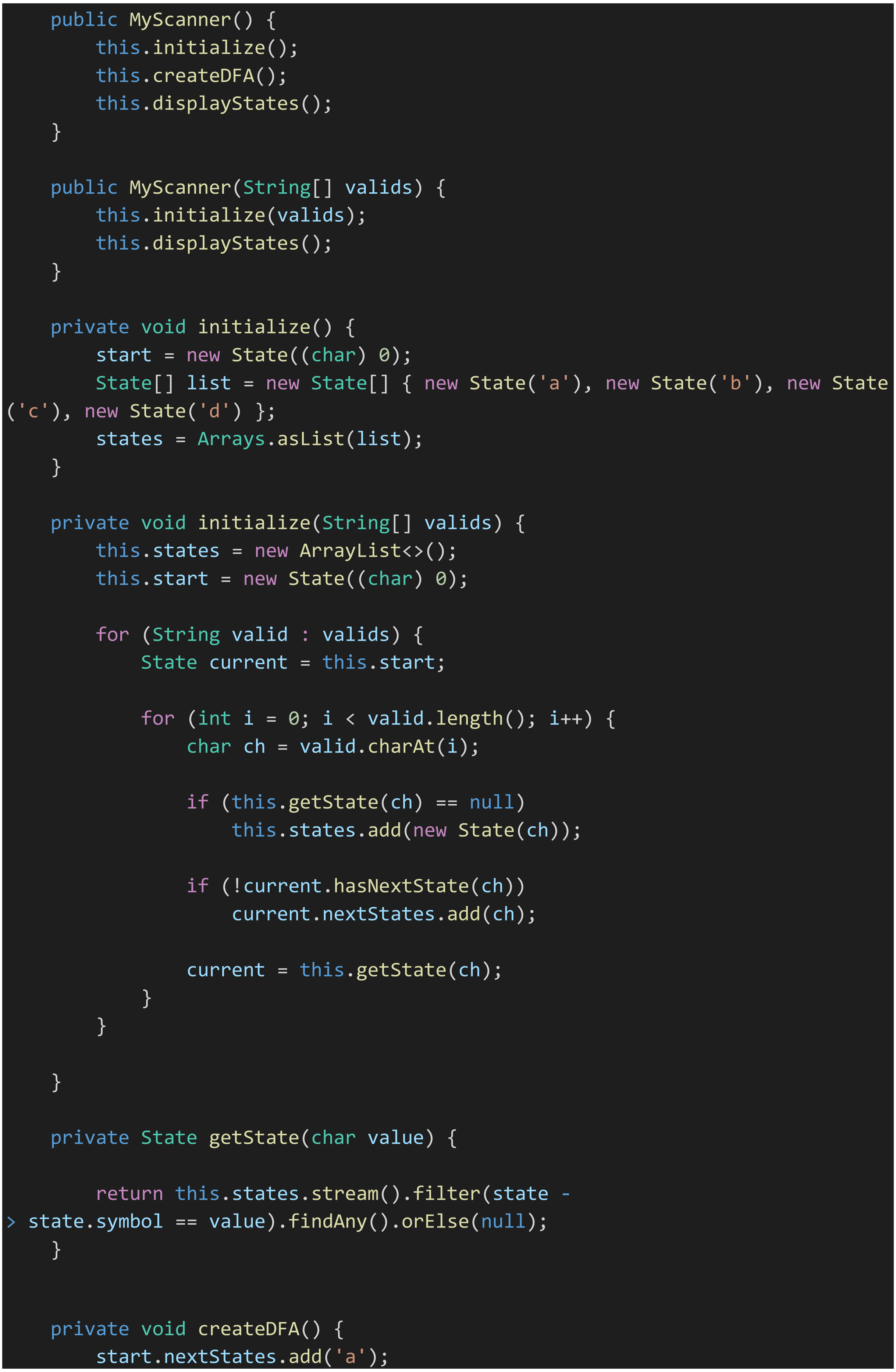
```
PS D:\MCA\MCA SEM 3\SS\Parsers > java .\LL1.java
LL1Parser
Steps:
 1. E
 2. TE'
 3. VT'E'
 4. <id>T'E'
 5. <id>*VT'E'
 6. <id>*VT'E'
 7. <id>*<id>T'E'
 8. <id>*<id>E'
 9. <id>*<id>+TE'
```

```
10. <id>*<id>+TE'
11. <id>*<id>+VT'E'
12. <id>*<id>+<id>T'E'
13. <id>*<id>+<id>E'
14. <id>*<id>+<id>
Completed in 14 steps.
Parsed: <id>*<id>+<id>
```

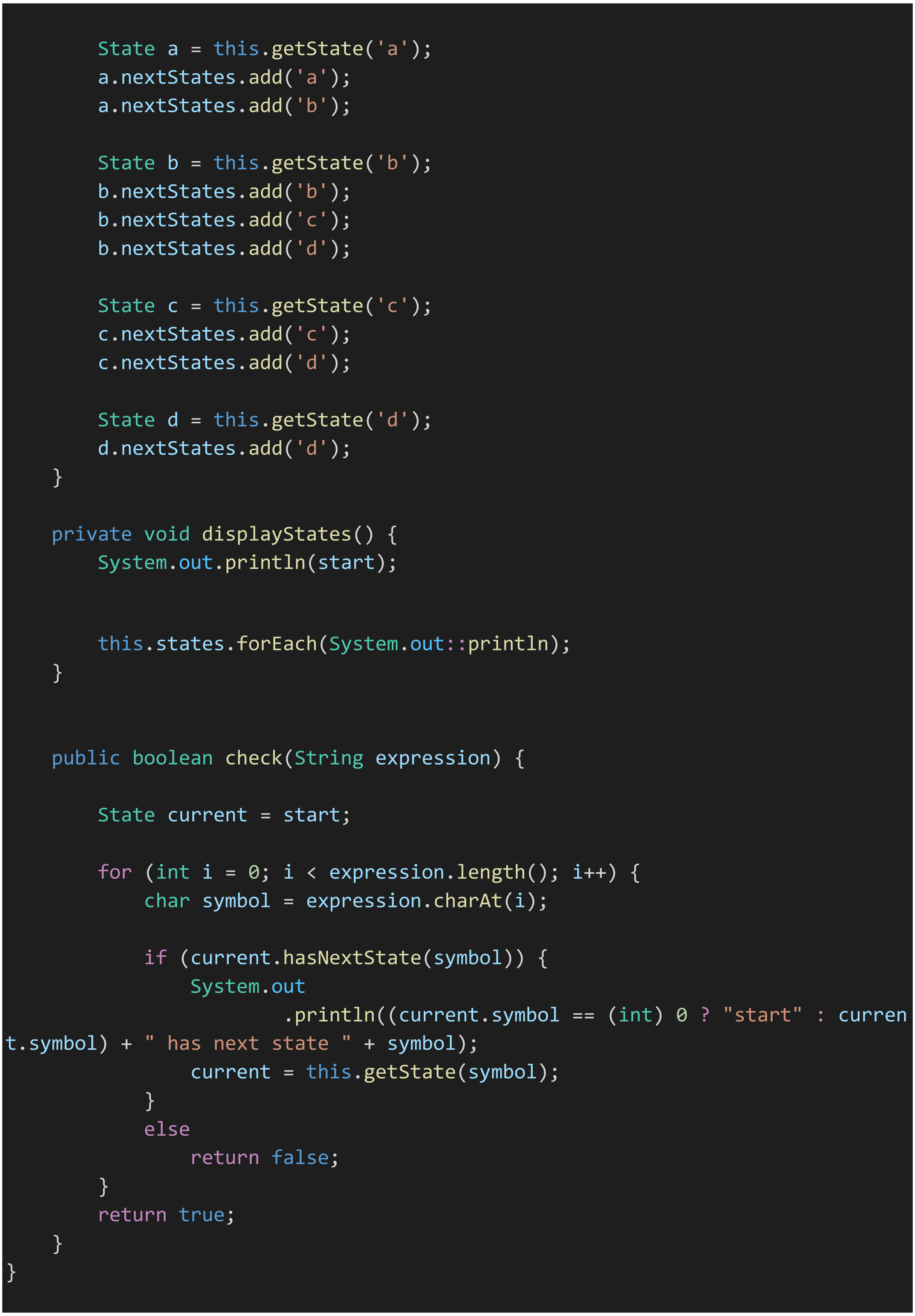
7. Scanner



```java
import java.util.Arrays;
import java.util.List;
import java.util.ArrayList;

public class ScannerDemo {
    public static void main(String[] args) {
        String[] valids = { "aaabbbcccddd", "aaabcddd", "abcd", "aaaaabbbbdddd", "abd" };
        MyScanner sc = new MyScanner(valids);
        boolean check6 = sc.check("cccddd");
        System.out.println("aaaacccddd is " + (check6 ? " valid" : " not valid."));
        System.out.println();
    }
}

class State {
    char symbol;
    List<Character> nextStates;

    State(char state) {
        this.symbol = state;
        nextStates = new ArrayList<>();
    }

    boolean hasNextState(char state) {
        return this.nextStates.stream().anyMatch(ch -> ch == state);
    }

    @Override
    public String toString() {
        String state = "State: " + (symbol == (int) 0 ? "start" : symbol) + ", Next States: ";

        for (char ch : nextStates)
            state += ch + ", ";

        return state.substring(0, state.length() - 2);
    }
}

class MyScanner {
    State start;
    List<State> states;
```



```
    public MyScanner() {
        this.initialize();
        this.createDFA();
        this.displayStates();
    }

    public MyScanner(String[] valids) {
        this.initialize(valids);
        this.displayStates();
    }

    private void initialize() {
        start = new State((char) 0);
        State[] list = new State[] { new State('a'), new State('b'), new State
('c'), new State('d') };
        states = Arrays.asList(list);
    }

    private void initialize(String[] valids) {
        this.states = new ArrayList<>();
        this.start = new State((char) 0);

        for (String valid : valids) {
            State current = this.start;

            for (int i = 0; i < valid.length(); i++) {
                char ch = valid.charAt(i);

                if (this.getState(ch) == null)
                    this.states.add(new State(ch));

                if (!current.hasNextState(ch))
                    current.nextStates.add(ch);

                current = this.getState(ch);
            }
        }

    }

    private State getState(char value) {

        return this.states.stream().filter(state -
> state.symbol == value).findAny().orElse(null);
    }


    private void createDFA() {
        start.nextStates.add('a');
```



```java
        State a = this.getState('a');
        a.nextStates.add('a');
        a.nextStates.add('b');

        State b = this.getState('b');
        b.nextStates.add('b');
        b.nextStates.add('c');
        b.nextStates.add('d');

        State c = this.getState('c');
        c.nextStates.add('c');
        c.nextStates.add('d');

        State d = this.getState('d');
        d.nextStates.add('d');
    }

    private void displayStates() {
        System.out.println(start);


        this.states.forEach(System.out::println);
    }


    public boolean check(String expression) {

        State current = start;

        for (int i = 0; i < expression.length(); i++) {
            char symbol = expression.charAt(i);

            if (current.hasNextState(symbol)) {
                System.out
                        .println((current.symbol == (int) 0 ? "start" : curren
t.symbol) + " has next state " + symbol);
                current = this.getState(symbol);
            }
            else
                return false;
        }
        return true;
    }
}
```

Output :

```
PS D:\MCA\MCA SEM 3\SS\scanner> java .\ScannerDemo.java
State: start, Next States: a
State: a, Next States: a, b
State: b, Next States: b, c, d
State: c, Next States: c, d
State: d, Next States: d
aaaacccddd is  not valid.
```