CME 3203

Theory of Computation

CONTEXT FREE GRAMMAR

2016510040 Elif Karataş
2016510001 Berk Adalı

Description:

Context free grammar is a formal grammar which is used to generate all possible strings in a given formal language.

We have parsed the CFG1 grammar in our homework and then we put it in the parse tree. We have seen all the strings that can be formed in this way. We have obtained the data group. We checked the input we received from the user with we traveled the tree inorder. If the string exists in the tree, program was accepted as we have show on screen. If the string doesn't exist in the tree, program was rejected.

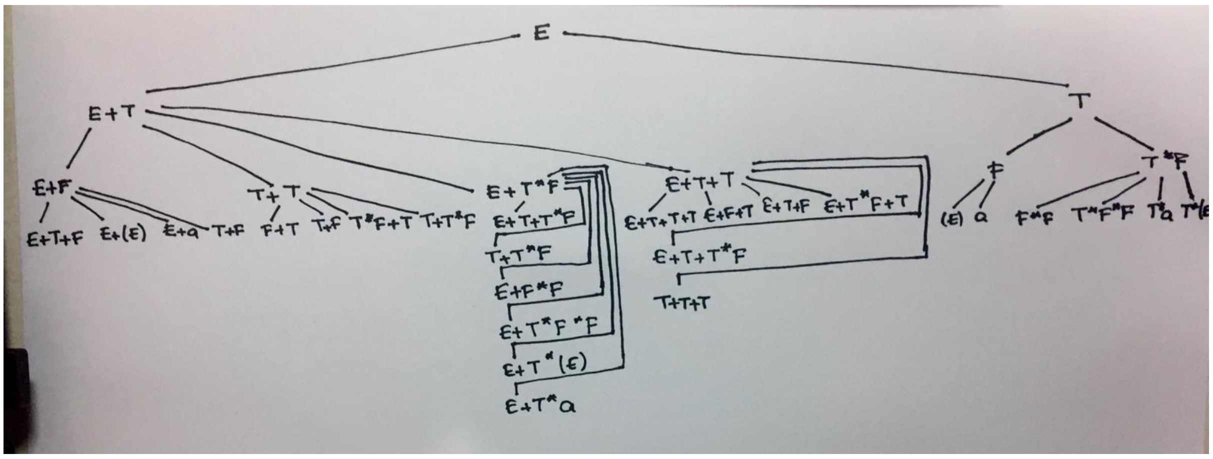
CFG1:

E>E+T|T

T>F|T*F

F>(E)|a

Parsing Tree:



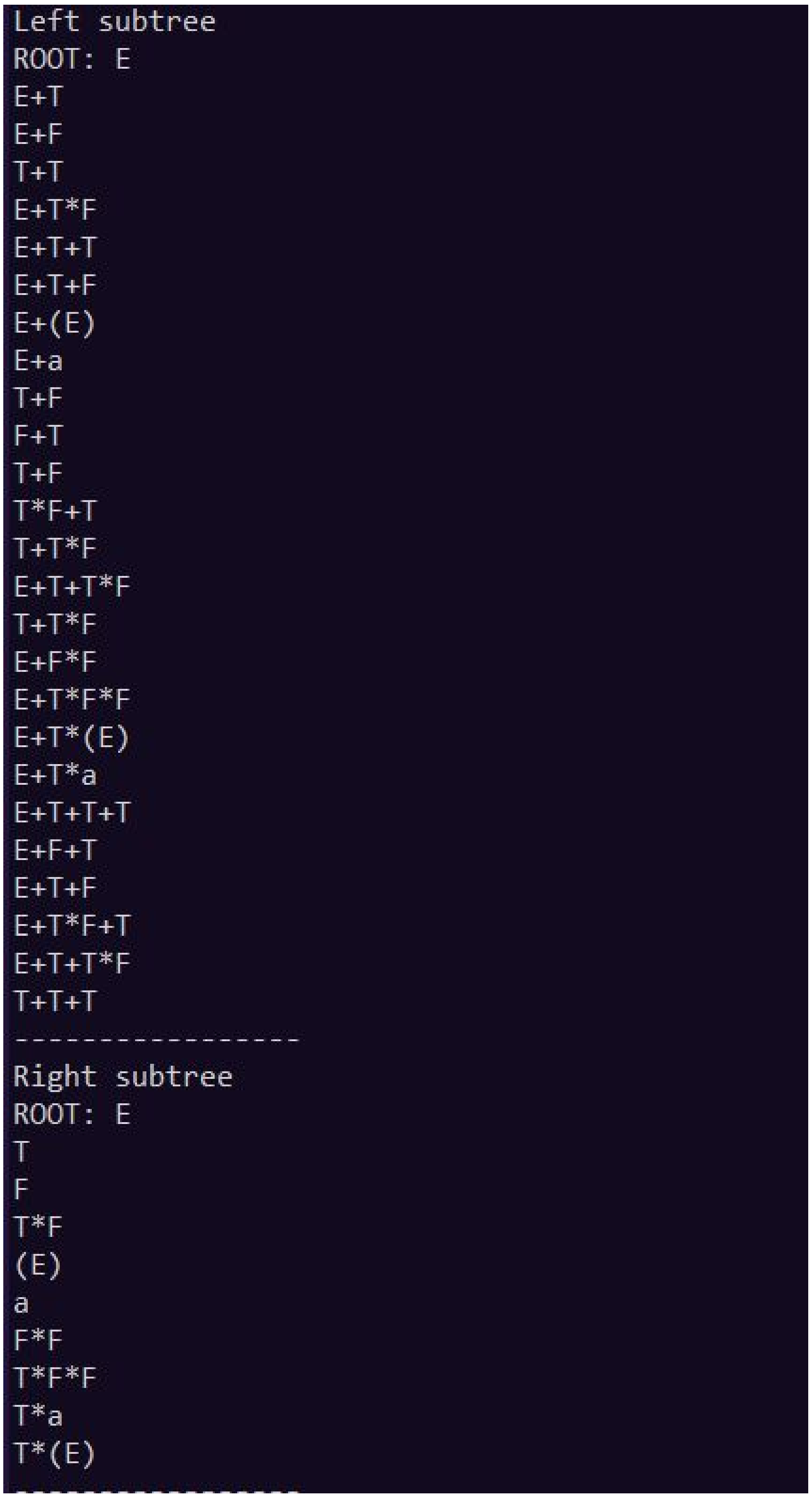
Adding only one possibility to the parse tree.

Pseudo Code:

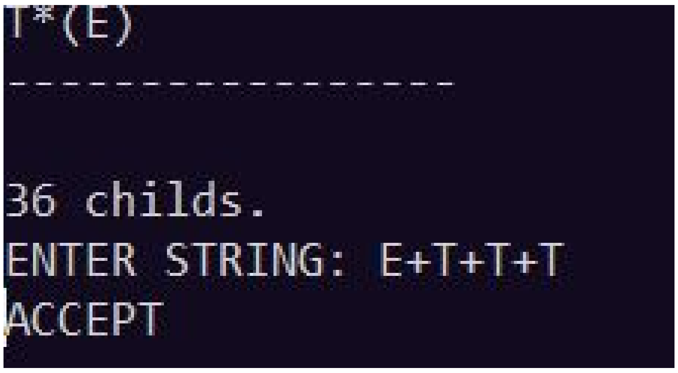
```
public class ParseTree {
defining root with null value
public void insertLeftCFG(terminals-->read from text,non_terminals,terminal){
        if(root==null){
            root-->read from txt
            adding root's data to array list for searching
        }
      current=root
      parent=null
      if(current's first child == null){
                current's first child-->read from text
                reading just first child because of providing left subtree
                adding current's first child's data to array list for searching
        }
}
current-->current's child
parent -->current
parent's child data-->read from text with separating terminals
adding parent childs' data for every node/every leaf to array list for searching

// It is containing with current-->current's child-->child and for every child, add to
array list and read data from txt file
}
public void insertRightCFG(terminals-->read from text,non_terminals,terminal){
        if(root==null){
            root-->read from text
            adding root's data to array list for searching
        }
      current=root
      parent=null
      if(current's second child == null){
                current's second child-->read from text
                reading just second child because of providing right subtree
                adding current's first child's data to array list for searching
        }
}
current-->current's child
parent -->current
parent's child data-->read from text with separating terminals
adding parent childs' data for every node/every leaf to array list for searching
```

```
// It is containing with current-->current's child-->child and for every child, add to
array list and read data from txt file
}
public boolean Search(String a-->input string) {
        searching all of the array list and
        if( array list's any member equals to input string )
            true
}
public class Main {
        firstly, reading text file and inserting the terminals
        secondly, taking input from user
        after that controlling the input string with if conditions
        firstly, checking the search function if it is true
        if it is not, check with separating if it is true
        after that, eliminate invalid characters and symbols if it is false
}
```

Sample Screenshots of Program:

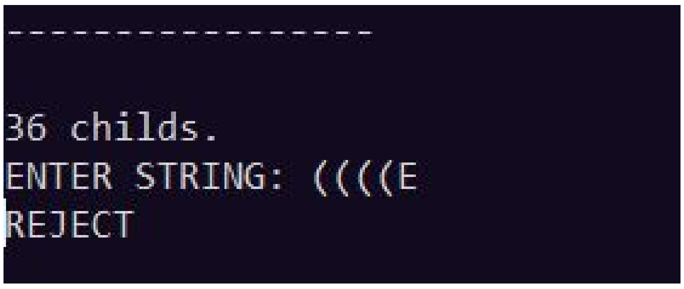


```
Left subtree
ROOT: E
E+T
E+F
T+T
E+T*F
E+T+T
E+T+F
E+(E)
E+a
T+F
F+T
T+F
T*F+T
T+T*F
E+T+T*F
T+T*F
E+F*F
E+T*F*F
E+T*(E)
E+T*a
E+T+T+T
E+F+T
E+T+F
E+T*F+T
E+T+T*F
T+T+T
-----------------
Right subtree
ROOT: E
T
F
T*F
(E)
a
F*F
T*F*F
T*a
T*(E)
```



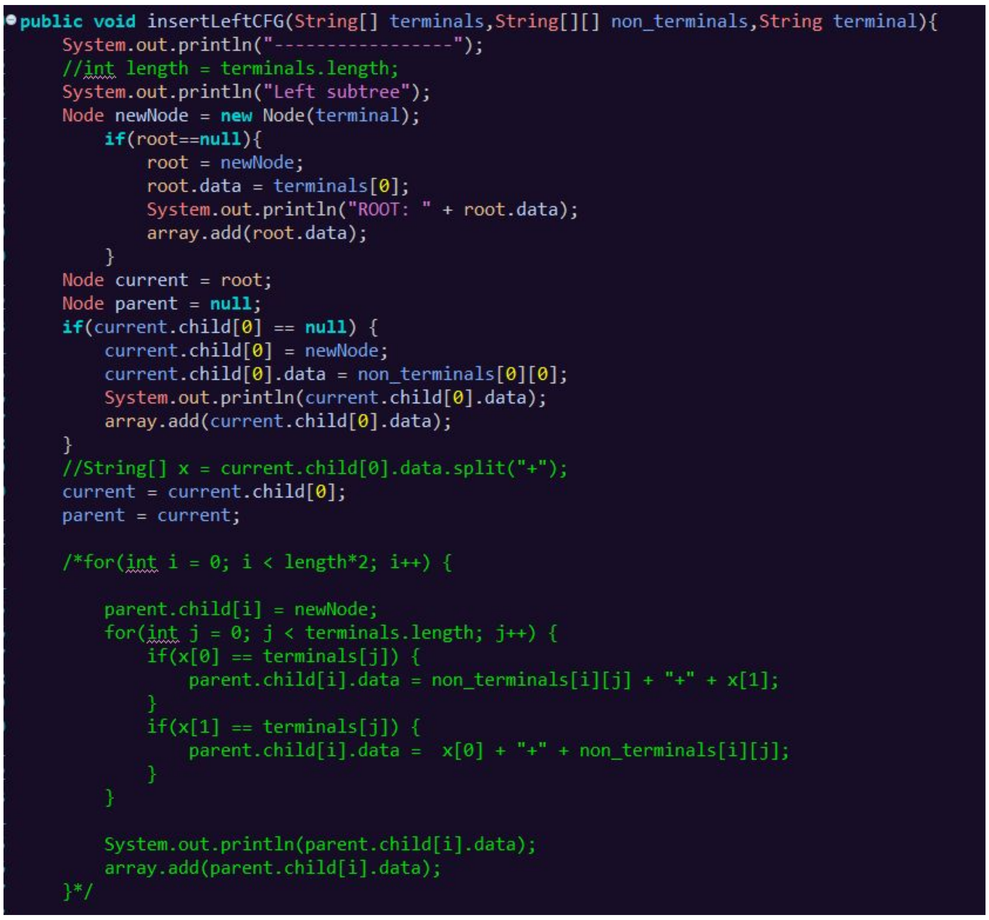
```
T*(E)
-----------------

36 childs.
ENTER STRING: E+T+T+T
ACCEPT
```

Accept input string sample.



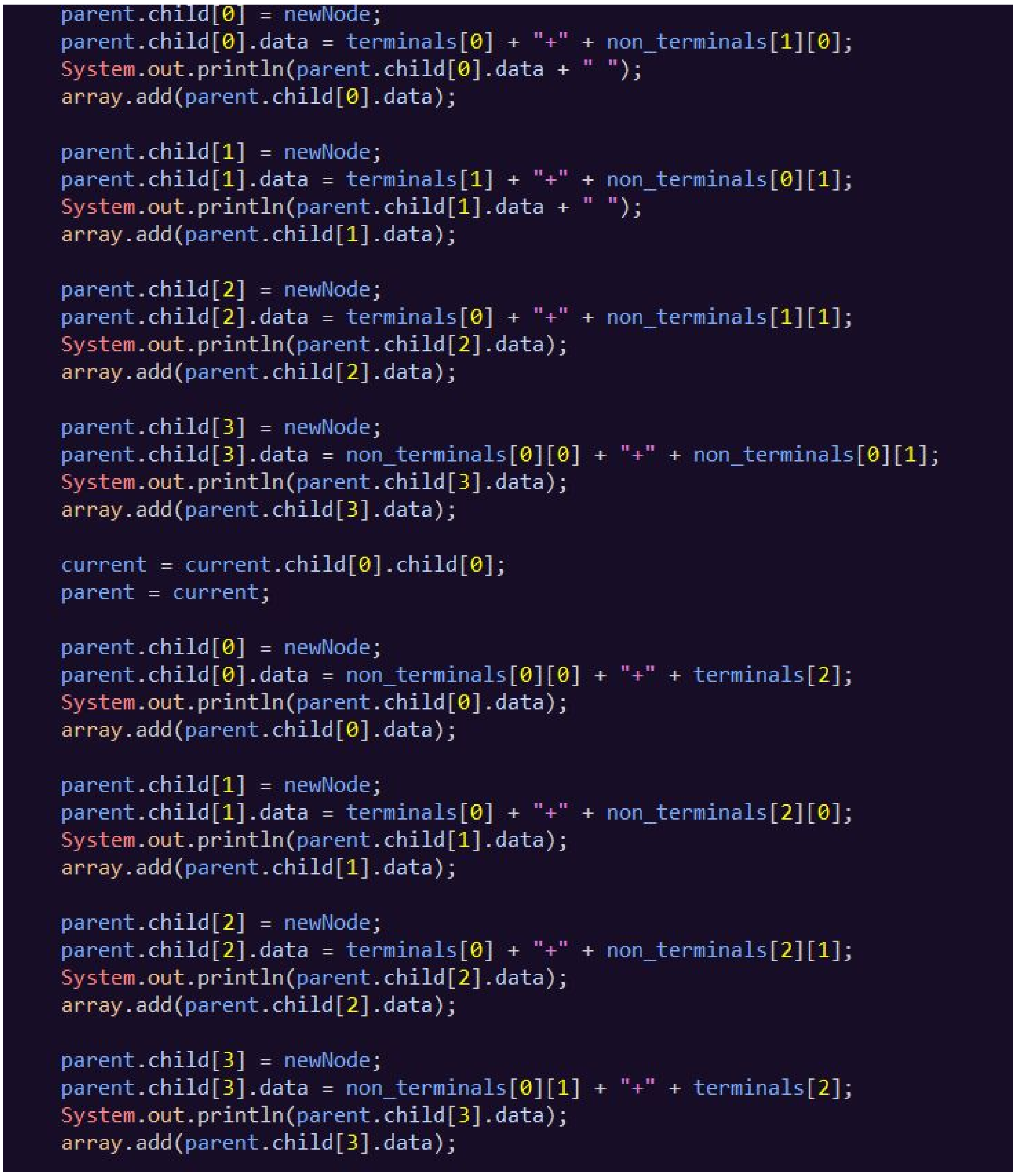
```
-----------------

36 childs.
ENTER STRING: ((((E
REJECT
```

Reject input string sample.



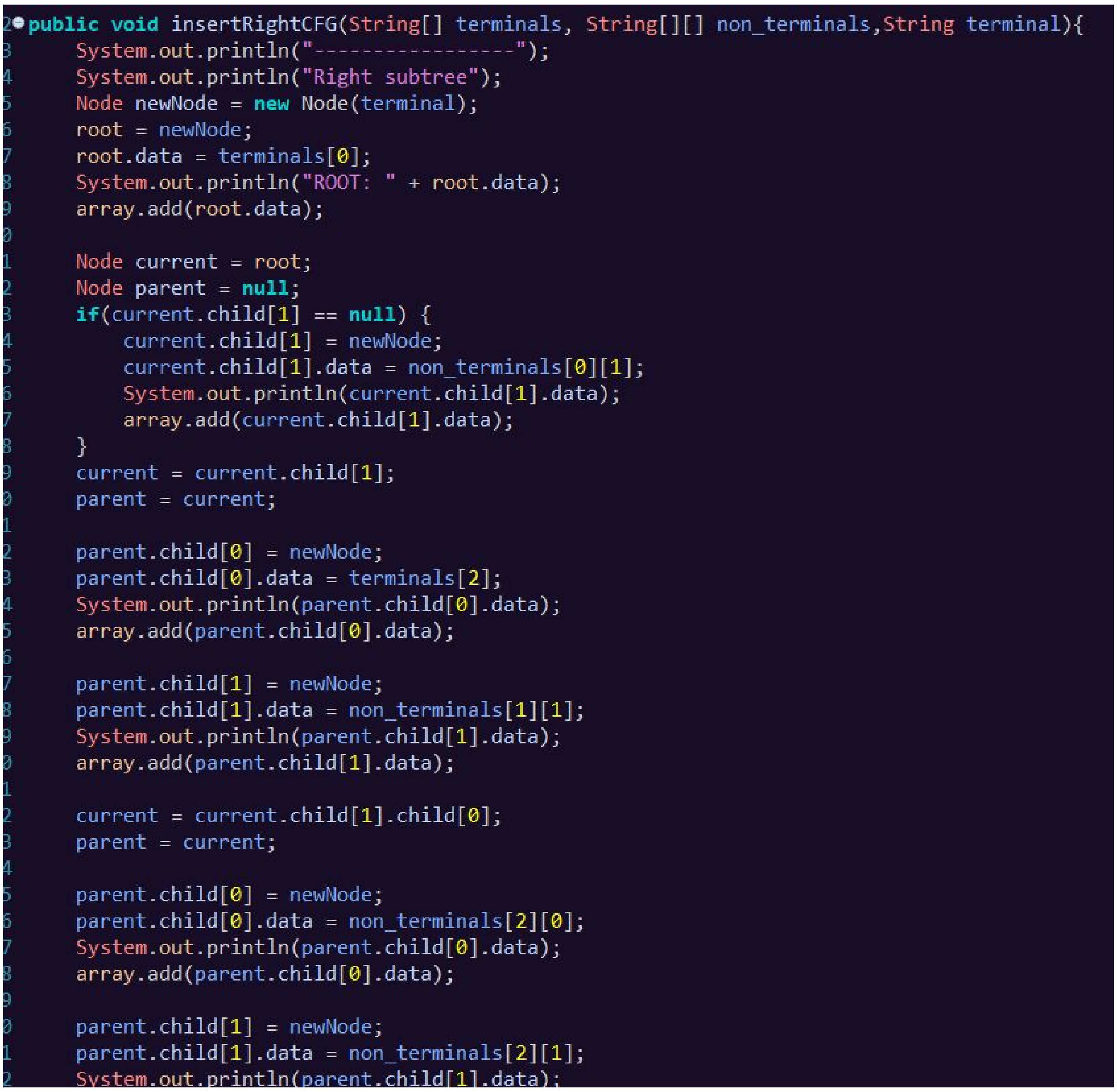
```java
public void insertLeftCFG(String[] terminals,String[][] non_terminals,String terminal){
    System.out.println("-----------------");
    //int length = terminals.length;
    System.out.println("Left subtree");
    Node newNode = new Node(terminal);
        if(root==null){
            root = newNode;
            root.data = terminals[0];
            System.out.println("ROOT: " + root.data);
            array.add(root.data);
        }
    Node current = root;
    Node parent = null;
    if(current.child[0] == null) {
        current.child[0] = newNode;
        current.child[0].data = non_terminals[0][0];
        System.out.println(current.child[0].data);
        array.add(current.child[0].data);
    }
    //String[] x = current.child[0].data.split("+");
    current = current.child[0];
    parent = current;

    /*for(int i = 0; i < length*2; i++) {

        parent.child[i] = newNode;
        for(int j = 0; j < terminals.length; j++) {
            if(x[0] == terminals[j]) {
                parent.child[i].data = non_terminals[i][j] + "+" + x[1];
            }
            if(x[1] == terminals[j]) {
                parent.child[i].data =  x[0] + "+" + non_terminals[i][j];
            }
        }

        System.out.println(parent.child[i].data);
        array.add(parent.child[i].data);
    }*/
```

Tried code samples in comment lines but failed.



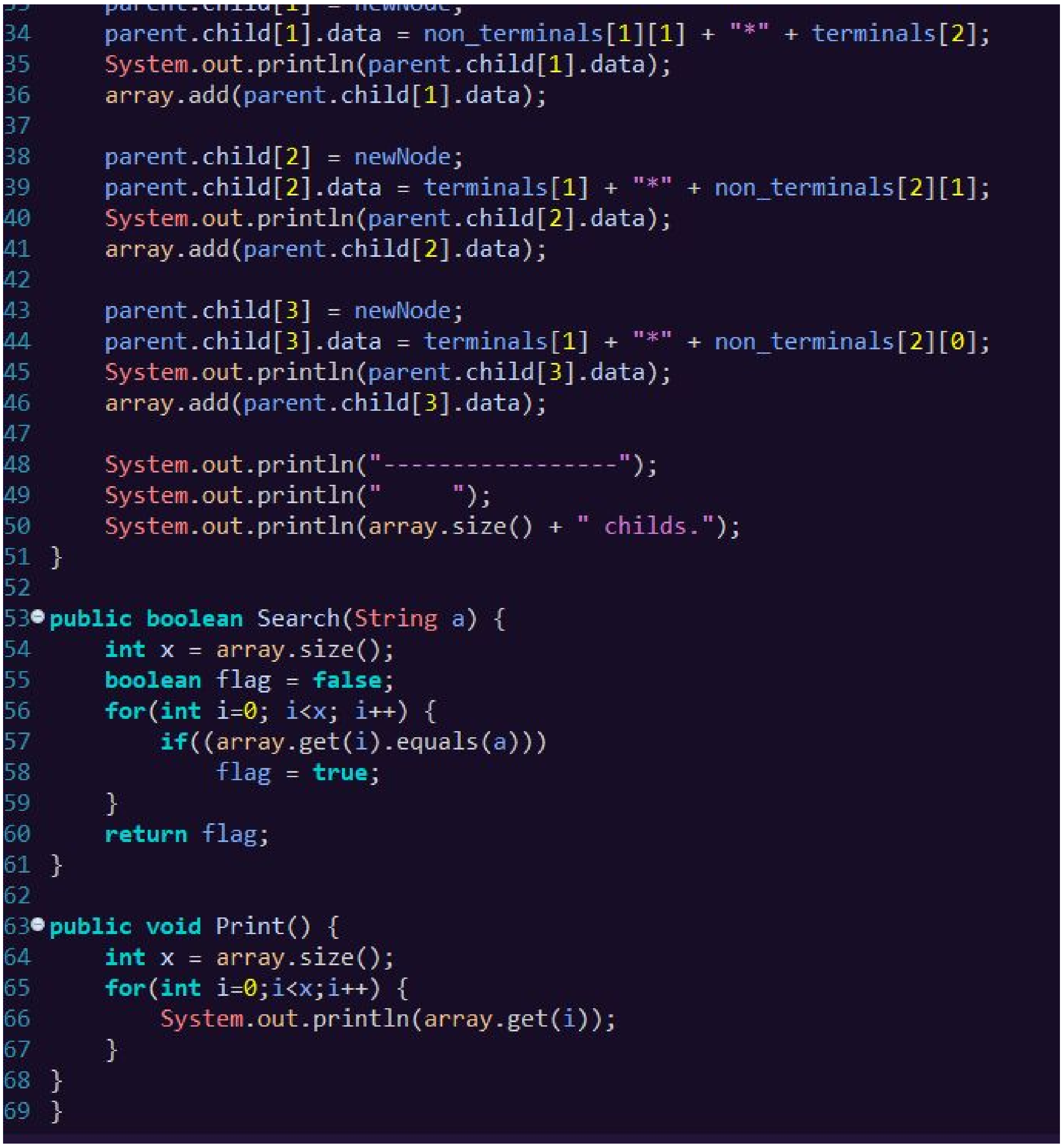
```
    parent.child[0] = newNode;
    parent.child[0].data = terminals[0] + "+" + non_terminals[1][0];
    System.out.println(parent.child[0].data + " ");
    array.add(parent.child[0].data);

    parent.child[1] = newNode;
    parent.child[1].data = terminals[1] + "+" + non_terminals[0][1];
    System.out.println(parent.child[1].data + " ");
    array.add(parent.child[1].data);

    parent.child[2] = newNode;
    parent.child[2].data = terminals[0] + "+" + non_terminals[1][1];
    System.out.println(parent.child[2].data);
    array.add(parent.child[2].data);

    parent.child[3] = newNode;
    parent.child[3].data = non_terminals[0][0] + "+" + non_terminals[0][1];
    System.out.println(parent.child[3].data);
    array.add(parent.child[3].data);

    current = current.child[0].child[0];
    parent = current;

    parent.child[0] = newNode;
    parent.child[0].data = non_terminals[0][0] + "+" + terminals[2];
    System.out.println(parent.child[0].data);
    array.add(parent.child[0].data);

    parent.child[1] = newNode;
    parent.child[1].data = terminals[0] + "+" + non_terminals[2][0];
    System.out.println(parent.child[1].data);
    array.add(parent.child[1].data);

    parent.child[2] = newNode;
    parent.child[2].data = terminals[0] + "+" + non_terminals[2][1];
    System.out.println(parent.child[2].data);
    array.add(parent.child[2].data);

    parent.child[3] = newNode;
    parent.child[3].data = non_terminals[0][1] + "+" + terminals[2];
    System.out.println(parent.child[3].data);
    array.add(parent.child[3].data);
```

Adding childs to the parse tree.



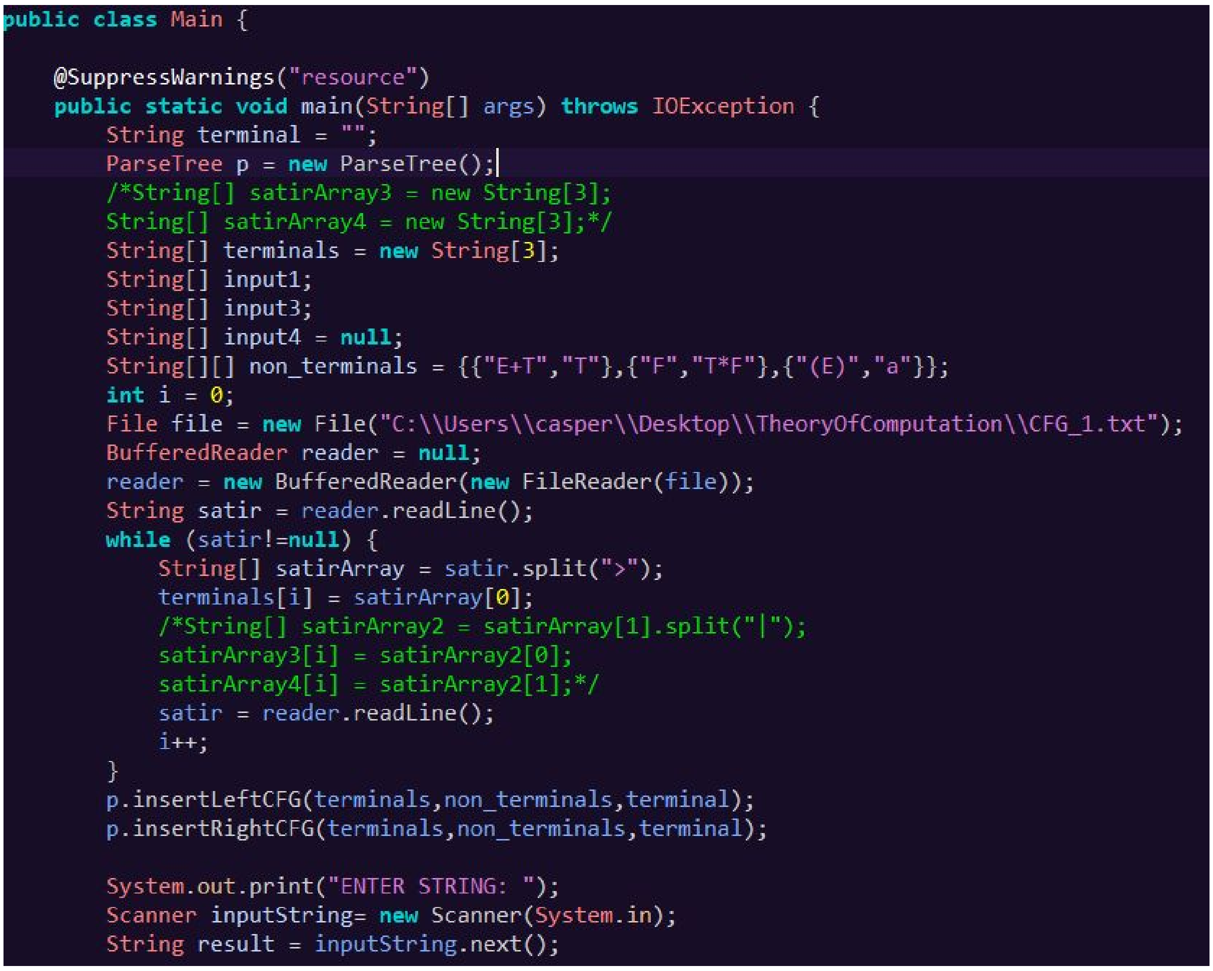
```java
public void insertRightCFG(String[] terminals, String[][] non_terminals,String terminal){
    System.out.println("-----------------");
    System.out.println("Right subtree");
    Node newNode = new Node(terminal);
    root = newNode;
    root.data = terminals[0];
    System.out.println("ROOT: " + root.data);
    array.add(root.data);

    Node current = root;
    Node parent = null;
    if(current.child[1] == null) {
        current.child[1] = newNode;
        current.child[1].data = non_terminals[0][1];
        System.out.println(current.child[1].data);
        array.add(current.child[1].data);
    }
    current = current.child[1];
    parent = current;

    parent.child[0] = newNode;
    parent.child[0].data = terminals[2];
    System.out.println(parent.child[0].data);
    array.add(parent.child[0].data);

    parent.child[1] = newNode;
    parent.child[1].data = non_terminals[1][1];
    System.out.println(parent.child[1].data);
    array.add(parent.child[1].data);

    current = current.child[1].child[0];
    parent = current;

    parent.child[0] = newNode;
    parent.child[0].data = non_terminals[2][0];
    System.out.println(parent.child[0].data);
    array.add(parent.child[0].data);

    parent.child[1] = newNode;
    parent.child[1].data = non_terminals[2][1];
    System.out.println(parent.child[1].data);
```

Inserting right subtree childs.



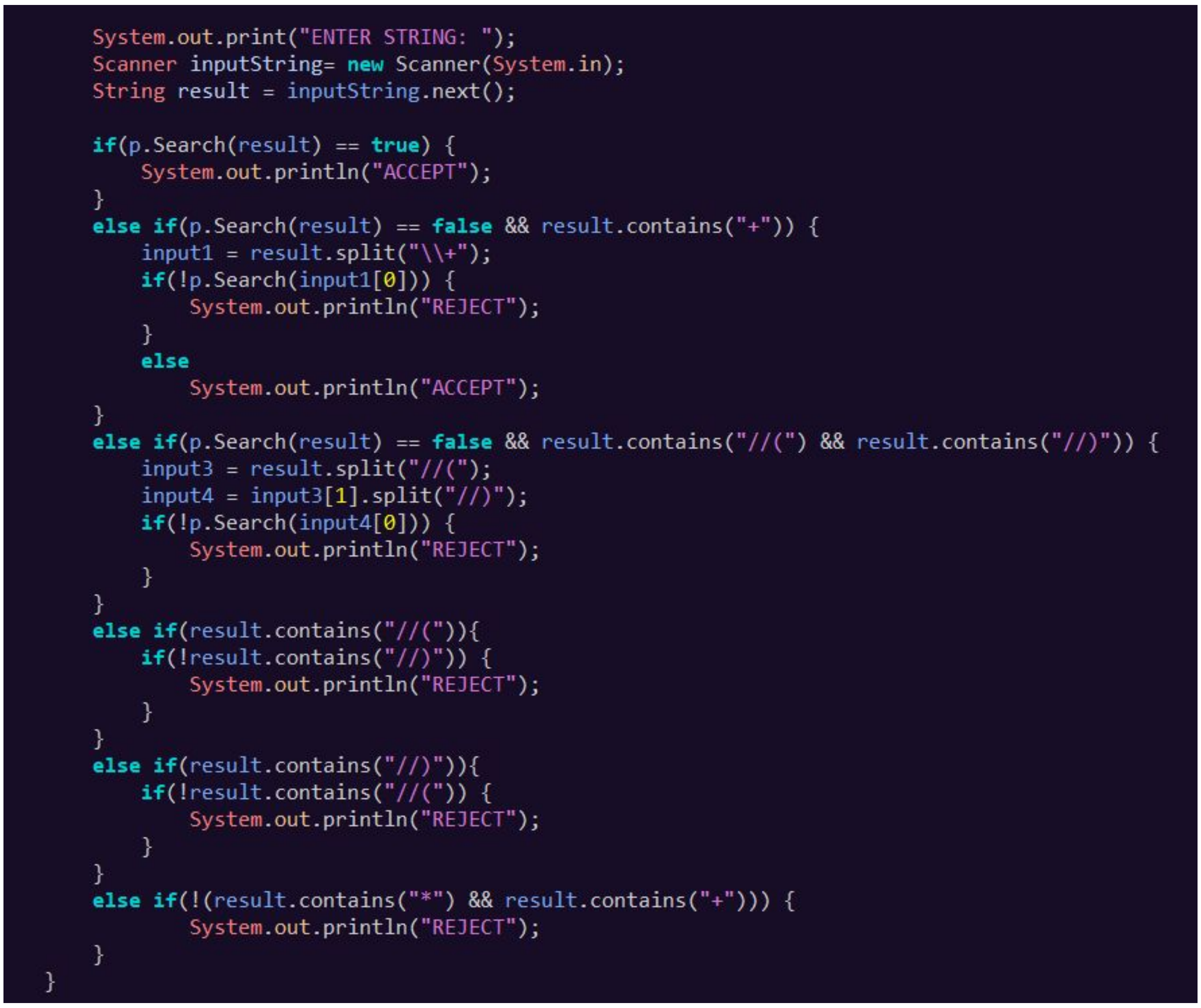
```java
    parent.child[1] = newNode;
    parent.child[1].data = non_terminals[1][1] + "*" + terminals[2];
    System.out.println(parent.child[1].data);
    array.add(parent.child[1].data);

    parent.child[2] = newNode;
    parent.child[2].data = terminals[1] + "*" + non_terminals[2][1];
    System.out.println(parent.child[2].data);
    array.add(parent.child[2].data);

    parent.child[3] = newNode;
    parent.child[3].data = terminals[1] + "*" + non_terminals[2][0];
    System.out.println(parent.child[3].data);
    array.add(parent.child[3].data);

    System.out.println("-----------------");
    System.out.println("     ");
    System.out.println(array.size() + " childs.");
}

public boolean Search(String a) {
    int x = array.size();
    boolean flag = false;
    for(int i=0; i<x; i++) {
        if((array.get(i).equals(a)))
            flag = true;
    }
    return flag;
}

public void Print() {
    int x = array.size();
    for(int i=0;i<x;i++) {
        System.out.println(array.get(i));
    }
}
}
```

Search and Print funcitons.



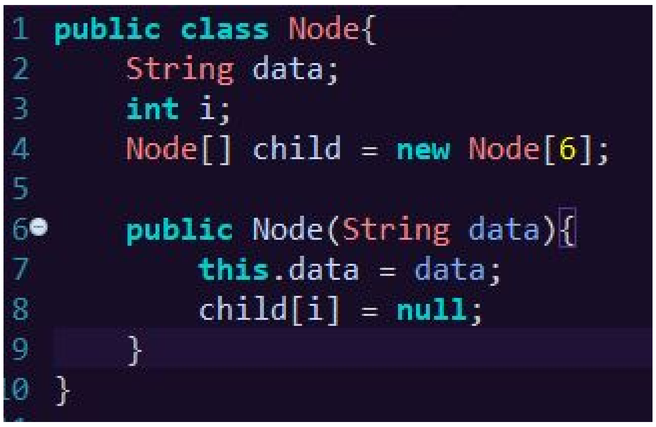
```java
public class Main {

    @SuppressWarnings("resource")
    public static void main(String[] args) throws IOException {
        String terminal = "";
        ParseTree p = new ParseTree();
        /*String[] satirArray3 = new String[3];
        String[] satirArray4 = new String[3];*/
        String[] terminals = new String[3];
        String[] input1;
        String[] input3;
        String[] input4 = null;
        String[][] non_terminals = {{"E+T","T"},{"F","T*F"},{"(E)","a"}};
        int i = 0;
        File file = new File("C:\\Users\\casper\\Desktop\\TheoryOfComputation\\CFG_1.txt");
        BufferedReader reader = null;
        reader = new BufferedReader(new FileReader(file));
        String satir = reader.readLine();
        while (satir!=null) {
            String[] satirArray = satir.split(">");
            terminals[i] = satirArray[0];
            /*String[] satirArray2 = satirArray[1].split("|");
            satirArray3[i] = satirArray2[0];
            satirArray4[i] = satirArray2[1];*/
            satir = reader.readLine();
            i++;
        }
        p.insertLeftCFG(terminals,non_terminals,terminal);
        p.insertRightCFG(terminals,non_terminals,terminal);

        System.out.print("ENTER STRING: ");
        Scanner inputString= new Scanner(System.in);
        String result = inputString.next();
```

Reading text file for CFG1 and inserting childs. Tried code samples in comment lines but failed.



```
        System.out.print("ENTER STRING: ");
        Scanner inputString= new Scanner(System.in);
        String result = inputString.next();

        if(p.Search(result) == true) {
            System.out.println("ACCEPT");
        }
        else if(p.Search(result) == false && result.contains("+")) {
            input1 = result.split("\\+");
            if(!p.Search(input1[0])) {
                System.out.println("REJECT");
            }
            else
                System.out.println("ACCEPT");
        }
        else if(p.Search(result) == false && result.contains("//(") && result.contains("//)")) {
            input3 = result.split("//(");
            input4 = input3[1].split("//)");
            if(!p.Search(input4[0])) {
                System.out.println("REJECT");
            }
        }
        else if(result.contains("//(")){
            if(!result.contains("//)")) {
                System.out.println("REJECT");
            }
        }
        else if(result.contains("//)")){
            if(!result.contains("//(")) {
                System.out.println("REJECT");
            }
        }
        else if(!(result.contains("*") && result.contains("+"))) {
                System.out.println("REJECT");
        }
    }
```

If controls to accept or reject.



```
public class Node{
    String data;
    int i;
    Node[] child = new Node[6];

    public Node(String data){
        this.data = data;
        child[i] = null;
    }
}
```

Node class.