5.1. Depth-First Visit of a Tree

The program presented in this section parses a given input string denoting a tree, and then visits that tree in a depth-first manner. The productions of the context-free grammar which generates the input string are the following ones:

```
tree ::= char | - char tree | (tree char tree)
char ::= '0' | '1' |...| '9'
```

The axiom is `tree`. No blanks are allowed before, in between, and after the input string. The tree which corresponds to the input string is then printed as a string which is generated by the following productions from the axiom `tree`:

```
tree ::= n | (n.tree) | (tree.n.tree)
n    ::= 0 | 1 |...| 9
```

Thus, if we execute the following two commands:

```
javac DepthFirstVisit.java
java  DepthFirstVisit
```

we get:

```
Input a tree according to the grammar given in the program, please.
```

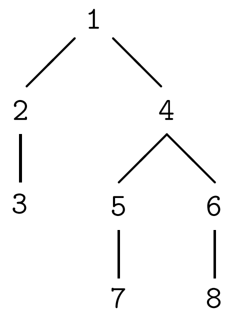
Then, if we type the following input string:

```
(-231(-564-78))
```

we get:

```
Tree in input:
((2.3).1.((5.6).4.(7.8)))

Depth first visit of the tree:
3 2 6 5 8 7 4 1
```

Indeed, the string `(-231(-564-78))` represents the tree:



whose depth first visit is: `3 2 6 5 8 7 4 1`.

In giving names to the files we have assumed that the word '`node`' is equivalent to the word '`tree`', because every node in a given tree is accessible from the root node of the tree and, in this sense, the root node is equivalent to the whole tree.

The expression: `Character.getNumericValue(c)` below is the same as: `c-48`. For instance, the numeric value of the character `'8'` is the integer `8`.

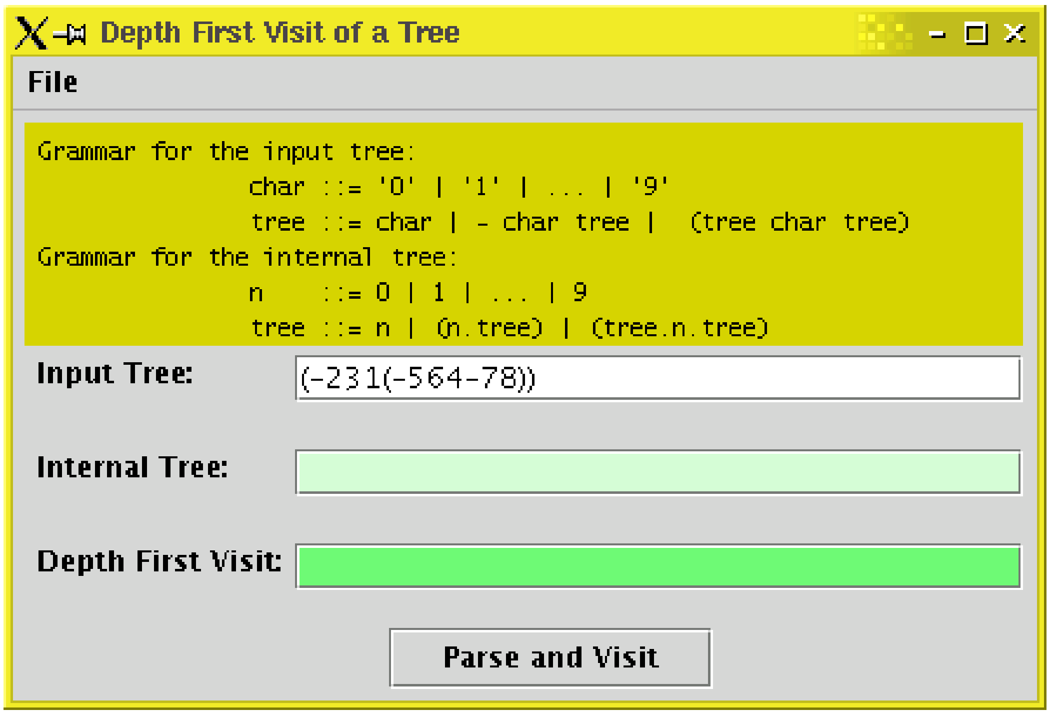
```
/**
 * ========================================================================
 *                      DEPTH FIRST VISIT OF A TREE
 *
 * Grammar G for the input tree:
 *                      char ::= '0' | '1' | ... | '9'
 *                      tree ::= char | - char tree | (tree char tree)
 *
 * Grammar for printing the input tree:
 *                      tree ::= n | (n.tree) | (tree.n.tree)
 *
 * No blanks are allowed before, in between, and after the input string.
 * A tree is input as a string generated by the grammar G and then
 * it is visited in the depth first manner.
 * ========================================================================
 */
import java.io.*;
/**
 * ========================================================================
 *                   Class which constructs a Node
 *
 * This class is used for constructing a Node of a binary tree.
 * By default, a Node is a binary node.
 * A Node can be extended for constructing a Node which is
 * either a binary node, or a unary node, or a leaf node.
 * ========================================================================
 */
class Node {
   public Object  value;      // value field
   public Node    left;       // reference to the left  child node
   public Node    right;      // reference to the right child node
  /**
   * @param v : value at the node
   * @param l : reference to the left  child node
   * @param r : reference to the right child node
   */                                   // constructor of a node
   public Node( Object v, Node l, Node r) {
      if ( v == null )  throw new IllegalArgumentException( );
      value  = v;
      left   = l;
      right  = r;
   }
}
/**
 * ========================================================================
 *             Class which constructs a binary node with two child nodes
 * ========================================================================
 */
class BinaryNode extends Node {
                                        // constructor of a binary node
  /**
   * @param v : value at the node
   * @param l : reference to the left  child node
   * @param r : reference to the right child node
   */
   public BinaryNode(Object v, Node l, Node r) {
      super(v, l, r);
   }
  /**
   * @return a string representing the tree whose root is the given node
   */
   public String toString(){
      return "("+ left.toString() +"."+ value +"."+ right.toString() +")";
   }
}
/**
 * ==========================================================================
 *            Class which constructs a unary node with one child node
 * ==========================================================================
 */
class UnaryNode extends Node {
                                        // constructor of a unary node
  /**
   * @param v     : value at the node
   * @param child : reference to the child node
   */

   public UnaryNode(Object v, Node child) {
          super(v, child, null);
   }
  /**
   * @return a string representing the tree whose root is the given node
   */
     public String toString() {
          return "(" + value + "." + left.toString() + ")";
     }
}
/**
 * ==========================================================================
 *            Class which constructs a leaf node
 * ==========================================================================
 */
class LeafNode extends Node {
                                        // constructor of a leaf
  /**
   * @param v : value at the node
   */
   public LeafNode(Object v) {
           super(v, null, null);
   }
  /**
   * @return a string representing the leaf node
   */
   public String toString() {
          return "" + value;
   }
}
/**
 * ==========================================================================
 *            Class ParseException
 * ==========================================================================
 */
class ParseException extends Exception {

                        // constructor of a new instance of ParseException
   public ParseException() {
          super("Parse  exception!");
   }
}
// --------------------------------------------------------------------------
public class DepthFirstVisit {
// ==========================================================================
    /**
     * This class DepthFirstVisit transforms the input string into a tree
     * of type Node.
     * The string encodes the tree as indicated by the grammar G.
     * @param  string: the string which encodes the given tree
     * @return the root of the tree generated by the parsing
     * @throws IOException or ParseException
     */
                         // stringReader is a sequence of characters,
                         // which are read one at a time, by
                         // the method read() of the class StringReader

   public static Node parse(BufferedReader stringReader)
                                  throws IOException, ParseException {
      char c = (char)stringReader.read();
      if ( ('0' <= c ) && ( c <= '9') ) {
         return new LeafNode(new Integer(Character.getNumericValue(c)));
      } else
      if ( c == '-') {
         char c2        = (char)stringReader.read();
         Integer value  = new Integer(Character.getNumericValue(c2));
         Node childTree   = parse(stringReader);
         UnaryNode node = new UnaryNode(value, childTree);
         return node;
      } else
      if ( c == '(' ) {
         Node leftTree   = parse(stringReader);
         char c2         = (char)stringReader.read();
         Integer value   = new Integer(Character.getNumericValue(c2));
         Node rightTree  = parse(stringReader);
         BinaryNode node = new BinaryNode(value, leftTree, rightTree);
         c2              = (char)stringReader.read();
         return node;
      } else {
      throw new ParseException();
      }
   }
  /** Depth first visit of the given tree
   *  @param tree : tree to be visited in a depth first manner
   *  @return the sequence of nodes which is the depth first visit of
   *          the given tree
   */
   public static String depthFirstVisit(Node tree) {
      StringBuffer buffer = new StringBuffer();
// --------------------- RECURSIVE VERSION -----------------------------
 if ( tree instanceof LeafNode ) {                                     //
           buffer.append(((LeafNode)tree).value + " ");                //
      } else                                                           //
      if ( tree instanceof UnaryNode ) {                               //
           buffer.append(depthFirstVisit(((UnaryNode)tree).left)       //
                     + tree.value + " ");                              //
      } else                                                           //
      if ( tree instanceof BinaryNode ) {                              //
             buffer.append(depthFirstVisit(((BinaryNode)tree).left)    //
                     + depthFirstVisit(((BinaryNode)tree).right)       //
                     + tree.value + " ");                              //
      };                                                               //
// --------------------- END OF RECURSIVE VERSION ----------------------
      return buffer.toString();
   }
// ---------------------------------------------------------------------
    public static void main(String[] args) throws IOException {

       System.out.println("Input a tree according to the grammar given "
                          + "in the program, please.");
       Node tree = null;
       try {
       tree = parse(new BufferedReader(new InputStreamReader(System.in)));
       } catch ( IOException e ) {
            e.printStackTrace();
       } catch ( ParseException pex ) {
            System.out.println(pex.getMessage());
       }
       if (tree != null) {
            System.out.println("\nTree in input:\n" + tree);
            System.out.println("\nDepth first visit of the tree:");
            System.out.println(depthFirstVisit(tree));
       }
    }
}
/**
 * input:  output:
 * ---------------------------------------------------------------------
 * javac DepthFirstVisit.java
 * java  DepthFirstVisit
 *
 *     Input a tree according to the grammar given in the program, please.
 * (-231(-564-78))
 *
 *     Tree in input:
 *     ((2.3).1.((5.6).4.(7.8)))
 *
 *     Depth first visit of the tree:
 *     3 2 6 5 8 7 4 1
 * ---------------------------------------------------------------------
 */
```

The following program is like the above one, but it uses a graphical interface. It can be run by typing the following commands:

```
javac DepthFirstVisit.java
javac DepthFirstVisitGUI.java
java  DepthFirstVisitGUI
```

```
/**
 * =====================================================================
 *                    DEPTH FIRST VISIT OF A TREE
 *                      Graphical User Iterface
 * Grammar G for the input tree:
 *                      char ::= '0' | '1' | ... | '9'
 *                      tree ::= char | - char tree | (tree char tree)
 *
 * Grammar for printing the input tree:
 *                     tree ::= n | (n.tree) | (tree.n.tree)
 *
 * A tree is input as a string generated by the grammar G and then
 * it is visited in the depth first manner.
 *
 * This program uses the class DepthFirstVisit in the file named
 * DepthFirstVisit.java. The file DepthFirstVisit.java should be stored
 * in the same folder where is stored this file DepthFirstVisitGUI.java.
 *
 * @author Corrado Di Pietro. Modifications by Alberto Pettorossi.
 * ======================================================================
 */
import java.io.*;
import javax.swing.JFrame;
// -----------------------------------------------
public class DepthFirstVisitGUI extends JFrame {
    /** Creates new form DepthFirstVisitGUI */
    public DepthFirstVisitGUI() {
        initComponents();
    }
    /** This method initComponents() is called from within the constructor
     *  to initialize the form.
     *  WARNING: Do NOT modify this code. The content of this method is
     *  always regenerated by the Form Editor.
     */
    private void initComponents() {//GEN-BEGIN:initComponents
        jLabel1   = new javax.swing.JLabel();
        jLabel2   = new javax.swing.JLabel();
        jLabel3   = new javax.swing.JLabel();
        jLabel4   = new javax.swing.JLabel();
        txInput   = new javax.swing.JTextField();
        txOutTree = new javax.swing.JTextField();
        txOutDPF  = new javax.swing.JTextField();
        btParse   = new javax.swing.JButton();
        menuBar   = new javax.swing.JMenuBar();
        fileMenu  = new javax.swing.JMenu();
        exitMenuItem = new javax.swing.JMenuItem();

        getContentPane().setLayout(null);
        setTitle("Depth First Visit of a Tree");
        setFont(new java.awt.Font("Default", 0, 10));
        setLocationRelativeTo(null);
        addWindowListener(new java.awt.event.WindowAdapter() {
            public void windowClosing(java.awt.event.WindowEvent evt) {
                exitForm(evt);
            }
        });
// ------------------------------------------------------------
        jLabel1.setBackground(new java.awt.Color(204, 204, 0));
        jLabel1.setFont(new java.awt.Font("Default", 0, 10));
        jLabel1.setText("<HTML>\n<PRE>\n"
        +" Grammar for the input tree:\n"
        +"                 char ::= '0' | '1' | ... | '9'\n"
        +"                 tree ::= char | - char tree | "
        +" (tree char tree)\n"
        +" Grammar for the internal tree:\n"
        +"                 n    ::= 0 | 1 | ... | 9\n"
        +"                 tree ::= n | (n.tree) | (tree.n.tree)\n"
        +"<PRE>\n</HTML>");

        jLabel1.setVerticalAlignment(javax.swing.SwingConstants.TOP);
        jLabel1.setOpaque(true);
        getContentPane().add(jLabel1);
        jLabel1.setBounds(5, 5, 425, 95);
// ------------------------------------------------------------
        jLabel2.setText("Input Tree:");
        getContentPane().add(jLabel2);
        jLabel2.setBounds(10, 104, 90, 16);
        getContentPane().add(txInput);
        txInput.setBounds(120, 104, 310, 20);
// ------------------------------------------------------------
        jLabel3.setText("Internal Tree:");
        getContentPane().add(jLabel3);
        jLabel3.setBounds(10, 144, 90, 16);
        txOutTree.setBackground(new java.awt.Color(204, 255, 204));
        getContentPane().add(txOutTree);
        txOutTree.setBounds(120, 144, 310, 20);
// ------------------------------------------------------------
        jLabel4.setText("Depth First Visit:");
        getContentPane().add(jLabel4);
        jLabel4.setBounds(10, 184, 140, 16);
        txOutDPF.setBackground(new java.awt.Color(102, 255, 102));
        getContentPane().add(txOutDPF);
        txOutDPF.setBounds(120, 184, 310, 20);
// ------------------------------------------------------------
        btParse.setText("Parse and Visit");
        btParse.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btParseActionPerformed(evt);
                }
            });
        getContentPane().add(btParse);
        btParse.setBounds(160, 220, 138, 26);
// ------------------------------------------------------------
        fileMenu.setText("File");
        exitMenuItem.setText("Exit");
        exitMenuItem.addActionListener(
                                    new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                exitMenuItemActionPerformed(evt);
                }
            });

        fileMenu.add(exitMenuItem);
        menuBar.add(fileMenu);
        setJMenuBar(menuBar);

        java.awt.Dimension screenSize =
             java.awt.Toolkit.getDefaultToolkit().getScreenSize();
        setBounds((screenSize.width-444)/2,
                  (screenSize.height-300)/2, 444, 300);
    } //GEN-END:initComponents

    private void btParseActionPerformed(java.awt.event.ActionEvent evt) {
      //GEN-FIRST:event_btParseActionPerformed
      // Add your handling code here:
           Node tree = null;
           try {    // from String to StringReader and to BufferedReader
               BufferedReader inputline
                    = new BufferedReader(
                                     new StringReader(txInput.getText()));
               tree = DepthFirstVisit.parse(inputline);
           } catch ( IOException e ) {
               e.printStackTrace();
               txOutTree.setText("IOException!");
           } catch ( ParseException pex ){
               txOutTree.setText(pex.getMessage());
        }
        if (tree != null) {
            txOutTree.setText(tree.toString());
            txOutDPF.setText(DepthFirstVisit.depthFirstVisit(tree));
        }
    } //GEN-LAST:event_btParseActionPerformed

    private void exitMenuItemActionPerformed(
                                     java.awt.event.ActionEvent evt) {
      //GEN-FIRST:event_exitMenuItemActionPerformed
        System.exit(0);
    } //GEN-LAST:event_exitMenuItemActionPerformed

    /** Exit the Application */
    private void exitForm(java.awt.event.WindowEvent evt) {
      //GEN-FIRST:event_exitForm
        System.exit(0);
    } //GEN-LAST:event_exitForm

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        new DepthFirstVisitGUI().show();
    }

    // Variables declaration - do not modify   //GEN-BEGIN:variables
    private javax.swing.JMenuItem exitMenuItem;
    private javax.swing.JMenu fileMenu;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JTextField txInput;
    private javax.swing.JTextField txOutTree;
    private javax.swing.JTextField txOutDPF;
    private javax.swing.JButton btParse;
    // End of variables declaration            //GEN-END:variables
}
/**
 * input:        output:
 * -----------------------------------------------------------------------
 * javac DepthFirstVisit.java
 * javac DepthFirstVisitGUI.java
 * java  DepthFirstVisitGUI
 *
 *              (see figures below)
 * -----------------------------------------------------------------------
 */
```

After typing the string `(-231(-564-78))` we have:



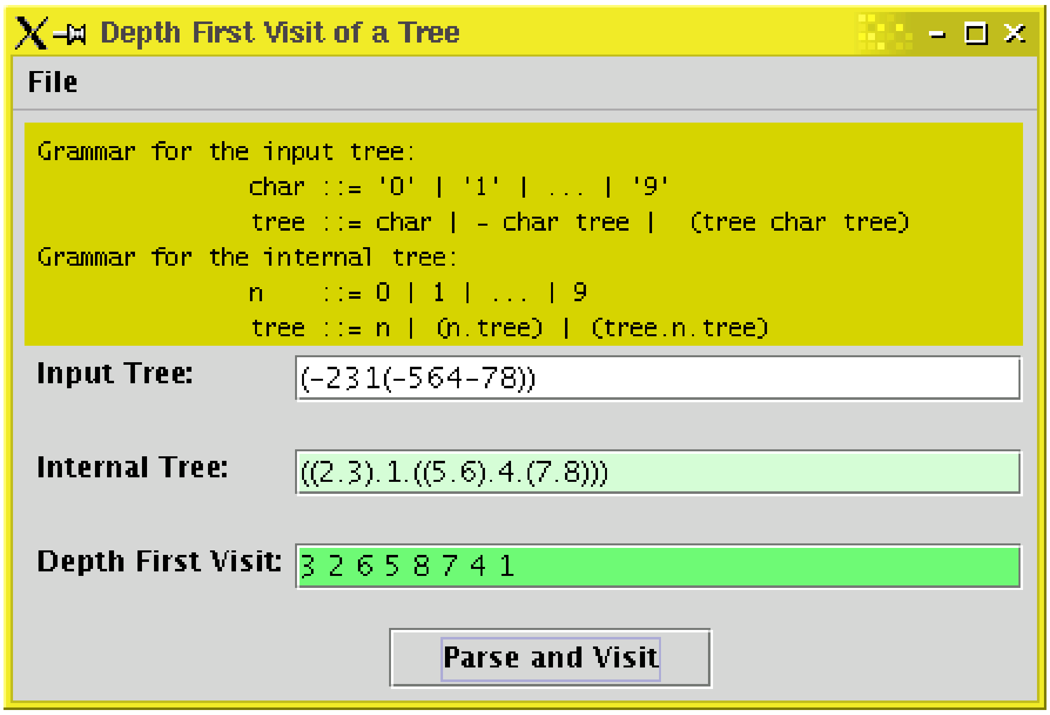
```
Grammar for the input tree:
                char ::= '0' | '1' | ... | '9'
                tree ::= char | - char tree |  (tree char tree)
Grammar for the internal tree:
                n    ::= 0 | 1 | ... | 9
                tree ::= n | (n.tree) | (tree.n.tree)
```

Input Tree: (-231(-564-78))

Internal Tree:

Depth First Visit:

Then by clicking the button ‘`Parse and Visit`’ we get:



```
Grammar for the input tree:
                char ::= '0' | '1' | ... | '9'
                tree ::= char | - char tree |  (tree char tree)
Grammar for the internal tree:
                n    ::= 0 | 1 | ... | 9
                tree ::= n | (n.tree) | (tree.n.tree)
```

Input Tree: (-231(-564-78))

Internal Tree: ((2.3).1.((5.6).4.(7.8)))

Depth First Visit: 3 2 6 5 8 7 4 1