Perpustakaan Unika

Appendix

1.LexicalTree.java (For Tries Search Algorithm)

```
00001 package MateusIDE.LexicalTree;
00002
00004 class LexicalTree extends LexicalRoot implements TreeTriesSearchAlgorithm {
00005
00007     public LexicalTree()
00008     {
00009       super();
00010     }
00022     public boolean triesSearch(String word,TreeLink headRoot)
00023     {
00024       int i = 0;
00025       String legalWord = ""+this.changeToLegalChar(word);
00026       System.out.println("\nBEGIN TRIES-SEARCH FOR: "+legalWord);
00027       this.rootNode = headRoot;
00028       TreeLink travelNode;
00029       if(legalWord.length() == 1) {
00030         travelNode = this.preOrderSearch('z',this.rootNode).right;
00031         if(((String)travelNode.data).compareTo(legalWord) == 0)
00032         {
00033           System.out.println(legalWord+" ---> Word Found\n");
00034           return true;
00035         }
00036       }
00037       for(i=0; i<legalWord.length(); i++)
00038       {
00039         boolean found = searchRoot(legalWord,legalWord.charAt(i),headRoot);
00040         if(found==true) {
00041           return true; }
00042       }
00043       System.out.println(legalWord+" -->Word Not Found\n");
00044       return false;
00045     }
00057     public boolean searchRoot(String word,Character chr,TreeLink root)
00058     {
```

```
00059        TreeLink travelNode;
00060        travelNode = this.preOrderSearch(chr,this.rootNode);
00061        if(travelNode.subNode !=null)
00062        {
00063          travelNode =  this.toSubNode(travelNode);
00064          System.out.println("ROOT GO TO SUB NODE OF THIS CHARACTER-->
"+chr);
00065          this.rootNode = travelNode;
00066          travelNode = this.preOrderSearch('z',this.rootNode).right;
00067          if(((String)travelNode.data).compareTo(word) == 0)
00068          {
00069            System.out.println(word+"---> Word Found\n");
00070            return true;
00071          }
00072        }
00073        return false;
00074      }
00085      public boolean searchAndDeleteWord(String word,TreeLink headRoot)
00086      {
00087        int i = 0;
00088        String legalWord = ""+this.changeToLegalChar(word);
00089        System.out.println("\nBEGIN SEARCH AND DELETE FOR: "+legalWord);
00090        this.rootNode = headRoot;
00091
00092        TreeLink travelNode;
00093        if(legalWord.length() == 1) {
00094          travelNode = this.preOrderSearch('z',this.rootNode).right;
00095          if(((String)travelNode.data).compareTo(legalWord) == 0 )
00096          {
00097            System.out.println(legalWord+" ---> Word Found\n"); //return true;
00098            System.out.println("ALL OPEN NODE IN THIS ROOT:
"+this.checkOpenNode(this.rootNode));
00099            travelNode.data = "--";
00100            System.out.println("CHANGE WORD DATA INTO: --");
00101            System.out.println("WORD HAVE BEEN DELETED!-->"+legalWord);
00102            return true;
00103          }
00104
00105        }
```

```
00106
00107         for(i=0; i<legalWord.length(); i++)
00108         {
00109           boolean found =
searchRootDelete(legalWord,legalWord.charAt(i),headRoot);
00110           if(found==true) {
00111             return true; }
00112         }
00113         System.out.println(legalWord+" -->Word Not Found\n");
00114         return false;
00115       }
00127       public boolean searchRootDelete(String word,Character chr,TreeLink root)
00128       {
00129         TreeLink travelNode;
00130         travelNode = this.preOrderSearch(chr,this.rootNode);
00131         TreeLink cutSubNode = travelNode;
00132         if(travelNode.subNode !=null)
00133         {
00134           travelNode =  this.toSubNode(travelNode);
00135           System.out.println("ROOT GO TO SUB NODE OF THIS CHARACTER-->
"+chr);
00136           this.rootNode = travelNode;
00137           travelNode = this.preOrderSearch('z',this.rootNode).right;
00138           if(((String)travelNode.data).compareTo(word) == 0 &&
this.checkAllOpenNode(this.rootNode) == "")
00139           {
00140             System.out.println(word+"---> Word Found\n");
00141             System.out.println("OPEN NODE IN THIS ROOT:
"+this.checkAllOpenNode(this.rootNode)+"\n");
00142           //travelNode.data = "$$";
00143           //System.out.println("CHANGE WORD DATA INTO: --");
00144             System.out.println("CUT SUB NODE "+cutSubNode.data);
00145             cutSubNode.subNode = null;
00146             System.out.println("WORD HAVE BEEN DELETED!-->"+word);
00147             return true;
00148           }
00149           else if(((String)travelNode.data).compareTo(word) == 0 &&
this.checkAllOpenNode(this.rootNode).length()>0)
00150           {
00151             System.out.println(word+"---> Word Found\n");
```

```
00152             travelNode.data = "--";
00153             System.out.println("CHANGE WORD DATA INTO: --");
00154             System.out.println("WORD HAVE BEEN DELETED!-->"+word);
00155             return true;
00156           }
00157
00158         }
00159         return false;
00160       }
00161
00171       public TreeLink travelingNode(String word,TreeLink headRoot)
00172       {
00173         int i = 0; char ch='a';
00174         String legalWord = ""+this.changeToLegalChar(word);
00175         System.out.println("BEGIN TRAVELING NODE FOR WORD WHICH BEGIN
WITH CHARACTER: "+legalWord);
00176         this.rootNode = headRoot;
00177         for(i=0; i<legalWord.length(); i++)
00178         {
00179           travelNode = traveling(legalWord,legalWord.charAt(i));
00180           ch = legalWord.charAt(i);
00181           if(travelNode==null){ System.out.println(ch+" -->RETURNING ROOT NODE
POSITION LAST SUBNODE NULL");
00182             return travelNode=null;}
00183         }
00184         System.out.println(ch+" -->RETURNING ROOT NODE POSITION");
00185         return travelNode;
00186
00187
00188       }
00198       public TreeLink traveling(String word,Character chr)
00199       {
00200         travelNode = this.preOrderSearch(chr,this.rootNode);
00201         if(travelNode.subNode !=null)
00202         {
00203           travelNode =  this.toSubNode(travelNode);
00204           System.out.println("ROOT GO TO SUB NODE OF THIS CHARACTER-->
"+chr);
00205           this.rootNode = travelNode;
```

Perpustakaan Unika

```
00206           return this.rootNode;
00207         }
00208         return null;
00209       }
00210
00217       public static void main(String args[]) //test this class
00218       {
00219         LexicalTree lexicalTree = new LexicalTree();
00220         lexicalTree.headRoot = lexicalTree.inputWord("yoyo",lexicalTree.headRoot);
00221         lexicalTree.headRoot = lexicalTree.inputWord("b",lexicalTree.headRoot);
00222         lexicalTree.headRoot = lexicalTree.inputWord("c",lexicalTree.headRoot);
00223         lexicalTree.headRoot = lexicalTree.inputWord("d",lexicalTree.headRoot);
00224         lexicalTree.headRoot = lexicalTree.inputWord("i",lexicalTree.headRoot);
00225       // lexicalTree.searchAndDeleteWord("mandir",lexicalTree.headRoot);
00226       // lexicalTree.searchAndDeleteWord("mando",lexicalTree.headRoot);
00227       //  lexicalTree.headRoot =
lexicalTree.inputWord("mandir",lexicalTree.headRoot);
00228        //  lexicalTree.searchAndDeleteWord("yoyo",lexicalTree.headRoot);
00229       //  lexicalTree.searchAndDeleteWord("mandi",lexicalTree.headRoot);
00230         lexicalTree.triesSearch("yoyo",lexicalTree.headRoot);
00231       //System.out.println(((Character)"B".charAt(0)).charValue());
00232       }
00233
00234
00235   }
```

2.FinalLexicalTree.java (For Predicting Word Algorithm)

Perpustakaan Unika

```
00001   package MateusIDE.LexicalTree;
00008    public class FinalLexicalTree extends LexicalTree implements
PredictingWordAlgorithm{
00015      public FinalLexicalTree() {
00016        super();
00017      }
00027      public String predictingWord(String predictWord,TreeLink headRoot)
00028     {
00029        int check = 0;
00030        String openNode="";
00031        String rootData="";
00032        predictWord = ""+this.changeToLegalChar(predictWord); // menghindari null
00033        if(predictWord.compareTo("")==0) {
00034          return "";  }
00035        rootNode = headRoot;
00036        String predict = "";
00037        System.out.println("--------PREDICT WORD----------");
00038        Query query = new Query();
00039        openNode = checkOpenNode(headRoot);
00040        rootData = (String)this.inOrderSearch('z',this.rootNode).right.data;
00041        System.out.println("OPEN NODE  "+openNode);
00042        if(rootData.compareTo("  ") ==0 || rootData.compareTo(" ")==0)
00043        {
00044        //do nothing
00045        }
00046        else {
00047             // DIA HARUS TETAP MENULUSURI KEBAWAH YO!
00048
00049          if(rootData.startsWith(predictWord) == true)
00050          {
00051            predict = predict + rootData+"\n";
00052          }
00053        }
00054        if(openNode.compareTo("")!=0) {
00055          System.out.println("FIRST ENQUE PROCESS");
00056          for(int i=0; i<openNode.length(); i++)
00057          {
00058            travelNode = this.inOrderSearch(openNode.charAt(i),headRoot);
00059            if((Character)travelNode.data ==(Character)predictWord.charAt(check))
```

```
00060             {
00061                query.enque(travelNode);
00062             }
00063          }
00064       }
00065       while(query.totalQuery>0)
00066       {
00067          check++;
00068          System.out.println("DEQUE PROCESS");
00069          TreeLink go = query.deque(); // link to left!
00070          String goTo = ""+(Character)go.left.data;
00071          this.travelNode = this.travelingNode(goTo,go.left);
00072          openNode = checkOpenNode(travelNode);
00073          System.out.println("OPEN NODE :  "+openNode);
00074          rootData = (String)this.inOrderSearch('z',this.rootNode).right.data;
00075          if(rootData.compareTo("  ") ==0 || rootData.compareTo(" ")==0)
00076          {
00077          //do nothing
00078          }
00079          else {
00080            if(rootData.startsWith(predictWord) == true)
00081            {
00082               System.out.println("rootData:"+rootData);
00083               predict = predict + rootData+"\n";
00084            }
00085          }
00086          if(openNode.compareTo("")!=0)
00087          {
00088            for(int i=0; i<openNode.length(); i++)
00089            {
00090               if(check < predictWord.length())
00091               {
00092                  System.out.println("ENQUE PROCESS 1:");
00093                  System.out.println("check: "+  predictWord.charAt(check));
00094                  travelNode = this.inOrderSearch(openNode.charAt(i),rootNode);
00095                  if((Character)travelNode.data ==
(Character)predictWord.charAt(check))
00096                  { query.enque(travelNode); }
00097               //else { query.enque(travelNode);   }
```

```
00098             }
00099             else if (check >= predictWord.length())
00100             {
00101               System.out.println("ENQUE PROCESS 2");
00102               travelNode = this.inOrderSearch(openNode.charAt(i),rootNode);
00103               query.enque(travelNode);   }
00104           }
00105         }
00106       }
00107       System.out.println("SearchAllWord: "+predict);
00108       return predict;
00109     }
00110
00121     public String searchAllWord(TreeLink headRoot)
00122     {
00123       rootNode = headRoot;
00124       String predict = "";
00125       System.out.println("-------SEARCH ALL WORD-----");
00126       Query query = new Query();
00127       String openNode = checkOpenNode(headRoot);
00128       String rootData = (String)this.inOrderSearch('z',this.rootNode).right.data;
00129       System.out.println("OPEN NODE  "+openNode);
00130       if(rootData.compareTo("  ") ==0 || rootData.compareTo(" ")==0)
00131       {
00132       //do nothing
00133       }
00134       else {   predict = predict + rootData+"\n";  }
00135       if(openNode.compareTo("")!=0) {
00136         for(int i=0; i<openNode.length(); i++)
00137         {
00138           travelNode = this.inOrderSearch(openNode.charAt(i),headRoot);
00139           query.enque(travelNode);
00140         }
00141       }
00142       while(query.totalQuery>0)
00143       {
00144         TreeLink go = query.deque(); // link to left!
00145         String goTo = ""+(Character)go.left.data;
```

Perpustakaan Unika

```
00146           this.travelNode = this.travelingNode(goTo,go.left);
00147           openNode = checkOpenNode(travelNode);
00148           System.out.println("OPEN NODE :  "+openNode);
00149           rootData = (String)this.inOrderSearch('z',this.rootNode).right.data;
00150           if(rootData.compareTo("  ") ==0 || rootData.compareTo(" ")==0)
00151           {
00152           //do nothing
00153           }
00154           else {   predict = predict + rootData+"\n";  }
00155           if(openNode.compareTo("")!=0) {
00156             for(int i=0; i<openNode.length(); i++)
00157             {
00158               travelNode = this.inOrderSearch(openNode.charAt(i),rootNode);
00159               query.enque(travelNode);
00160             }
00161           }
00162         }
00163         System.out.println("SearchAllWord: "+predict);
00164         return predict;
00165       }
00166       /*
00167       A Method For testing this class
00168       @param : args instance of array of string
00169       input : args instance of array of string
00170       output : process (string) to console
00171       */
00172        public static void main(String args[]) //test this class
00173       {
00174         FinalLexicalTree finalLexicalTree = new FinalLexicalTree();
00175         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("S",finalLexicalTree.headRoot);
00176         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("g",finalLexicalTree.headRoot);
00177
00178         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("class",finalLexicalTree.headRoot);
00179         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("Yoyo",finalLexicalTree.headRoot);
00180         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("{",finalLexicalTree.headRoot);
```

Perpustakaan Unika

```
00181         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("}",finalLexicalTree.headRoot);
00182         finalLexicalTree.headRoot =
finalLexicalTree.inputWord("s",finalLexicalTree.headRoot);
00183         System.out.println("PredictWord: "+
finalLexicalTree.predictingWord("c",finalLexicalTree.headRoot));
00184         finalLexicalTree.triesSearch("S",finalLexicalTree.headRoot); //dinormalkan
00185         finalLexicalTree.searchAndDeleteWord("Object",finalLexicalTree.headRoot);
00186         System.out.println("PredictWord: "+
finalLexicalTree.predictingWord("Y",finalLexicalTree.headRoot));
00187         System.out.println("SearchAllWord: "+
finalLexicalTree.searchAllWord(finalLexicalTree.headRoot));
00188        //  System.out.println("PredictWord: "+
finalLexicalTree.predictingWord("S",finalLexicalTree.headRoot));
00189        // System.out.println("PredictWord: "+
finalLexicalTree.predictingWord("Si",finalLexicalTree.headRoot));
00190       // System.out.println("SearchAllWord: "+
finalLexicalTree.searchAllWord(finalLexicalTree.headRoot));
00191     }
00192   }
```

3. Class Highlight (For Word and Syntax Checker)

```
00991  class Highlight implements ActionListener, Runnable
00992     {
00995        private Thread run1;
00996
```

Perpustakaan Unika

```
00998       private Caret caret;
01001       private StyleContext sc;
01002
01005       private ActionEvent e;
01008       private StyleContext context;
01010       private Style style;
01011
01013        public Highlight()
01014       {
01015         context = new StyleContext();
01016         style = context.getStyle(StyleContext.DEFAULT_STYLE);
01017       }
01024        public void actionPerformed(ActionEvent event)
01025       {
01026         e = event;
01027         highlight.setEnabled(false);
01028         unhighlight.setEnabled(false);
01029         System.out.println("BUTTON EAST PANEL PRESSED,
HIGHLIGHT/UNHIGHLIGHT!");
01030         try {
01031           doc = getTextDocument();
01032          // MiddlePanelGui.wordEditor.setDocument(doc);
01033           Caret caret = MiddlePanelGui.wordEditor.getCaret();
01034           caret.setSelectionVisible(true);
01035           MiddlePanelGui.wordEditor.setCaretPosition(0);
01036           caret.setVisible(true);
01037           run1 = new Thread(this);
01038           run1.setDaemon(true);
01039           searchEditor.setText("");
01040           textArea.setText("");
01041           sc = StyleContext.getDefaultStyleContext();
01042           System.out.println("TO THREAD");
01043           MiddlePanelGui.progressBar.setIndeterminate(true);
01044           run1.start();
01045         }
01046            catch(Exception er) {System.out.println(er);
01047             MiddlePanelGui.progressBar.setIndeterminate(false);
01048           }
01049
01050
01051       }
```

Perpustakaan Unika

```
01058         public synchronized void run()
01059        {
01060          try{
01061            int pos = 0;
01062            boolean found = false;
01063            String word="";
01064            sbufer = new StringBuffer(doc.getText(0, doc.getLength()));
01065            sbufer.append(" ");
01066            int i=0;
01067            MiddlePanelGui.wordEditor.setCaretPosition(0);
01068            System.out.println("IN COMMENT");
01069            if(sbufer != null)
01070            {
01071              if(e.getSource() == unhighlight )
01072              {
01073                System.out.println("UNHIGHLIGHT WORD");
01074                JTextField field = new JTextField(word);
01075                try{
01076                  StyleConstants.setForeground(style,Color.black);
01077                  StyleConstants.setBold(style, false);
01078                  StyledDocument docSty =
MiddlePanelGui.wordEditor.getStyledDocument();
01079                  docSty.setCharacterAttributes(0, doc.getLength(),style,true);
01080                }
01081                  catch (Exception e) {System.out.println(e);
01082                    MiddlePanelGui.progressBar.setIndeterminate(false);
01083                  }
01084              }
01085              else {
01086                System.out.println("HIGHLIGHT WORD");
01087                i=0;
01088                System.out.println("GET TEXT STATEMENT!!"+sbufer);
01089                while(i < sbufer.length())
01090                {
01091                  if(sbufer.charAt(i) == '\n'||sbufer.charAt(i) == ' ' ||sbufer.charAt(i) ==
'('||sbufer.charAt(i) == ')' ||sbufer.charAt(i) == '<' ||sbufer.charAt(i) == '>'||sbufer.charAt(i) ==
';'||sbufer.charAt(i) == '{' ||sbufer.charAt(i) == '}' ||sbufer.charAt(i) == '[' ||sbufer.charAt(i) ==
']'||sbufer.charAt(i) == '!'||sbufer.charAt(i) == '@'||sbufer.charAt(i) == '+'||sbufer.charAt(i) ==
'#'||sbufer.charAt(i) == '%'||sbufer.charAt(i) == '^' ||sbufer.charAt(i) == '&'||sbufer.charAt(i) ==
'*'||sbufer.charAt(i) == '-'||sbufer.charAt(i) == '+'||sbufer.charAt(i) == '='||sbufer.charAt(i) == '|'
|| sbufer.charAt(i) == ',' || sbufer.charAt(i) == '/' || sbufer.charAt(i) == '"' || sbufer.charAt(i) ==
((char)39) )
01092                  {
```

Perpustakaan Unika

```
01093                     System.out.println("Comment: "+comment1+" Mark: "+mark+"
Aposthrope: "+apostrophe+" comment2: "+comment2+" "+sbufer.charAt(i));
01094                     if(i > 0 && comment1 == true &&  sbufer.charAt(i) == '*' &&
sbufer.charAt(i-1) == '/'  && comment2 == true && mark == true && apostrophe == true )
01095                     {
01096                       System.out.println("IN HERE!! 1");
01097                       MiddlePanelGui.wordEditor.setCaretPosition(i);
01098                       comment1 = false;
01099                       xCom++;
01100                       stackCom[xCom] = sbufer.charAt(i);
01101                       System.out.println("IN HERE!! LOLOS");
01102                     }
01103                     else if(i > 0 && comment1 == false &&  sbufer.charAt(i) == '/' &&
sbufer.charAt(i-1) == '*' && comment2 == true && mark == true && apostrophe == true)
01104                     {
01105                       System.out.println("IN HERE!! 2");
01106                       MiddlePanelGui.wordEditor.setCaretPosition(i);
01107                       comment1 = false;
01108
01109                       if(stackCom[xCom] == '*' && xCom > -1 ) // pop
01110                       {
01111                         comment1 = true;
01112                         xCom--;
01113                       }
01114                     }
01115                     if(i > 0 && comment2 == true &&  sbufer.charAt(i) == '/' &&
sbufer.charAt(i-1) == '/' && comment1 == true && mark == true && apostrophe == true )
01116                     {
01117                       System.out.println("IN HERE!! 3");
01118                       MiddlePanelGui.wordEditor.setCaretPosition(i);
01119                       comment2 = false;
01120                     }
01121                     else if(comment2 == false && sbufer.charAt(i) == '\n'  && comment1
== true && mark == true && apostrophe == true  )
01122                     {
01123                       MiddlePanelGui.wordEditor.setCaretPosition(i);
01124                       comment2 = true;
01125                     }
01126                     if( apostrophe == true && sbufer.charAt(i) == ((char)39) &&
comment1 == true && comment2 == true && mark == true)
01127                     {
01128                       MiddlePanelGui.wordEditor.setCaretPosition(i);
```

Perpustakaan Unika

```
01129                     System.out.println("Aposthrope last: "+apostrophe);
01130                     apostrophe = false;
01131                     xAps++;
01132                   }
01133                   else if(apostrophe == false && sbufer.charAt(i) == ((char)39) &&
comment1 == true && comment2 == true && mark == true  )
01134                   {
01135                     MiddlePanelGui.wordEditor.setCaretPosition(i);
01136                     apostrophe = false;
01137                     if(xAps > -1 ) // pop
01138                     {
01139                       apostrophe  = true;
01140                       xAps--;
01141                     }
01142                   }
01143                   if(sbufer.charAt(i) == ""  && mark == true && comment1 == true &&
comment2 == true && apostrophe == true)
01144                   {
01145                     MiddlePanelGui.wordEditor.setCaretPosition(i);
01146                     mark = false;
01147                     xMark++;
01148                   }
01149                   else if( sbufer.charAt(i) == ""  && mark == false && comment1 ==
true && comment2 == true && apostrophe == true )
01150                   {
01151                     MiddlePanelGui.wordEditor.setCaretPosition(i);
01152                     mark = false;
01153
01154                     if(xMark > -1 ) // pop
01155                     {
01156                       mark  = true;
01157                       xMark--;
01158                     }
01159                   }
01160
01161
01162                   else if( expressionOpen(sbufer.charAt(i)) == true && e.getSource()
== syntax && comment1 == true && mark == true && apostrophe == true && comment2 ==
true )
01163                   {
01164                     x++;
01165                     JTextField field = new JTextField(""+sbufer.charAt(i));
```

Perpustakaan Unika

```
01166                     stack[x] = sbufer.charAt(i);
01167                     idx[x] = new Integer(0);
01168                     StyleConstants.setForeground(style,Color.red);
01169                     StyleConstants.setBold(style, true);
01170                     //StyleConstants.setUnderline(style,true);
01171                     String breaks = findWordNext(field);
01172                     if(breaks.compareTo("")==0){
01173                       break; }
01174
01175                     MiddlePanelGui.wordEditor.setCharacterAttributes(style, true);
01176                     idx[x] = MiddlePanelGui.wordEditor.getCaretPosition();
01177                   }
01178                   else if ( expressionClose(sbufer.charAt(i)) == true  && e.getSource()
== syntax && comment1 == true && mark == true && apostrophe == true && comment2 ==
true )
01179                   {
01180
01181                     System.out.println("IN BRACKETS ");
01182                     JTextField field = new JTextField(""+sbufer.charAt(i));
01183
01184                     StyleConstants.setForeground(style,Color.red);
01185                     StyleConstants.setBold(style, true);
01186                     //StyleConstants.setUnderline(style,true);
01187                     String breaks = findWordNext(field);
01188                     if(breaks.compareTo("")==0){
01189                       break; }
01190                     MiddlePanelGui.wordEditor.setCharacterAttributes(style, true);
01191
01192                     int tmp = MiddlePanelGui.wordEditor.getCaretPosition();
01193                     Character chr = expressionVersus(sbufer.charAt(i));
01194                     System.out.println(chr);
01195                     if(x >= 0)
01196                     {
01197                       System.out.println(chr);
01198                       if(stack[x] == chr   ) // pop
01199                       {
01200                         if(inBracket == false)
01201                         {
01202                           MiddlePanelGui.wordEditor.setCaretPosition(idx[x]-1);
01203                           field = new JTextField(""+chr);
01204                           StyleConstants.setForeground(style,new Color(0, 170, 0));
01205                           StyleConstants.setBold(style, true);
```

```
01206                         //StyleConstants.setUnderline(style,false);
01207                           breaks = findWordNext(field);
01208                           if(breaks.compareTo("")==0){
01209                             break; }
01210                           MiddlePanelGui.wordEditor.setCharacterAttributes(style,
true);
01211                           MiddlePanelGui.wordEditor.setCaretPosition(tmp-1);
01212                           field = new JTextField(""+sbufer.charAt(i));
01213                           StyleConstants.setForeground(style,new Color(0, 170, 0));
01214                           StyleConstants.setBold(style, true);
01215                         //StyleConstants.setUnderline(style,false);
01216                           breaks = findWordNext(field);
01217                           if(breaks.compareTo("")==0){
01218                             break; }
01219                           MiddlePanelGui.wordEditor.setCharacterAttributes(style,
true);
01220                           x--;
01221                         }
01222                       }
01223                     }
01224                     MiddlePanelGui.wordEditor.setCaretPosition(tmp);
01225                   }
01226
01227                   if( e.getSource() == highlight && comment1 == true && mark ==
true && apostrophe == true && comment2 == true)
01228                   {
01229                     found =
WestPanelGuiAction.lexicalTree.triesSearch(word,WestPanelGuiAction.lexicalTree.headRo
ot);
01230
01231                     if(found)
01232                     {
01233                       JTextField field = new JTextField(word);
01234
01235                       try{
01236                         StyleConstants.setForeground(style,Color.blue);
01237                         StyleConstants.setBold(style, true);
01238                         String breaks =  findWordNext(field);
01239                         if(breaks.compareTo("")==0){
01240                           break; }
01241                         MiddlePanelGui.wordEditor.setCharacterAttributes(style,
true);
```

```
01242                       }
01243                          catch (Exception e) {System.out.println(e);
01244                            MiddlePanelGui.progressBar.setIndeterminate(false);
01245                          }
01246                      }
01247                     }
01248
01249                       word="";
01250                     }
01251                     else
01252                     {
01253                       word = word +  sbufer.charAt(i);
01254                     }
01255                     i++;
01256                  }
01257               }
01258               if(found && e.getSource() == highlight && comment1 == true && mark ==
true && apostrophe == true && comment2 == true)
01259               {
01260                  JTextField field = new JTextField(word);
01261
01262                  try{
01263                     //StyleContext context = new StyleContext();
01264                     //Style style = context.getStyle(StyleContext.DEFAULT_STYLE);
01265                     //StyleConstants.setAlignment(style, StyleConstants.ALIGN_RIGHT);
01266                     StyleConstants.setForeground(style,Color.red);
01267                     StyleConstants.setBold(style, true);
01268                     findWordNext(field);
01269                     MiddlePanelGui.wordEditor.setCharacterAttributes(style, true);
01270                  }
01271                     catch (Exception e) {System.out.println(e);
01272                       MiddlePanelGui.progressBar.setIndeterminate(false);
01273                     }
01274
01275               }
01276
01277               MiddlePanelGui.progressBar.setIndeterminate(false);
01278               unhighlight.setEnabled(true);
01279               highlight.setEnabled(true);
01280               syntax.setEnabled(true);
01281               doc = null;
```

Perpustakaan Unika

```
01282              idx  = null;
01283              stack = null;
01284              stackCom = null;
01285              MiddlePanelGui.wordEditor.setVisible(false);
01286              MiddlePanelGui.wordEditor.setVisible(true);
01287              if(Thread.currentThread()==run1)
01288              {
01289                 synchronized(run1){
01290                    sbufer = null;
01291                    System.out.println("IN STOP THREAD");
01292                    run1.interrupt(); run1.notify();
01293                    run1 =null;
01294                    forceGarbageCollector(10);
01295                 }
01296              }
01297           }
01298        }
01299           catch(Exception er) {System.out.println(er);
01300            unhighlight.setEnabled(true);
01301            highlight.setEnabled(true);
01302            MiddlePanelGui.progressBar.setIndeterminate(false);
01303           }
01304     }
01305
01314      public boolean expressionOpen(Character tChar)
01315     {
01316       if (tChar == '(' ) {
01317          return true; }
01318       else if (tChar == '[' )
01319       {
01320          return true; }
01321       else if (tChar == '{')
01322       {
01323          return true; }
01324       //else if (tChar == '<')
01325     //  {
01326          //return true; }
01327       return false;
01328     }
01338      public boolean expressionClose(Character tChar)
01339     {
```

Perpustakaan Unika

```
01340         if (tChar == ')') {
01341           return true; }
01342         else if (tChar == ']') {
01343           return true; }
01344         else if (tChar == '}') {
01345           return true; }
01346         //else if (tChar == '>') {
01347           //return true; }
01348         return false;
01349       }
01358       public Character expressionVersus(Character tChar)
01359       {
01360         if (tChar == '(') {
01361           return ')'; }
01362         else if (tChar == ')') {
01363           return '('; }
01364         else if (tChar == '[') {
01365           return ']'; }
01366         else if (tChar == ']') {
01367           return '['; }
01368         else if (tChar == '{') {
01369           return '}'; }
01370         else if (tChar == '}') {
01371           return '{'; }
01372         //else if (tChar == '<') {
01373       //     return '>'; }
01374         //else if (tChar == '>') {
01376         return ' ';
01377       }
01378
01379     }
```