Institute of Technology, Carlow
B.Sc. in Software Engineering

CW228

Code Listing

C Maintenance Tool

Name: Anna-Christina Friedrich

ID: C00132716

Supervisor: Dr. Christophe Meudec

Submission Date: 16.04.2010

Table of Contents

1. CMT.java

```
/*************************************************************************
 *                              CMT.java                                 *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This is the main class of the tool. All datastructures that are used *
 * are implemented here. Parameter passed via command prompt are passed *
 * to an object of ComLineInterpreter.                                   *
 *                                                                       *
 *************************************************************************/

/**
 * imported packages
 */
import java.io.*;
import java.util.ArrayList;
import java.util.Stack;

/**
 * class definition
 */
public class CMT {


      /**the binary tree to store all variable identifier*/
      static public Tree<Identifier> idTree=new Tree<Identifier>();

      /**the binary tree to store all function identifier*/
      static public Tree<Function> functionTree=new Tree<Function>();

      /**represents current scope in a C file*/
      static public Integer scope=0;

      /**Stack that stores current scopes*/
      static public Stack<Scope> scopeStack=new Stack<Scope>();

      /**list containing all referenced filenames*/
      static public ArrayList<File> filenames=new ArrayList<File>();

      /**suffix of C sourcefiles*/
      final static public String cSuffix = ".c";

      /**suffix of textfiles*/
      final static public String txtSuffix = ".txt";
```

```
	/**suffix of C header files*/
	final static public String hSuffix = ".h";

	/**
	 *
	 * @param args : array of commands passed via command prompt
	 */
	public static void main(String[] args) {

		try{
			new ComLineInterpreter(args);
		}catch(ArrayIndexOutOfBoundsException e){
			ComLineInterpreter.writeInfo();
		}
	}
}
```

2. Identifier.java

```
/*************************************************************************
 *                        Identifier.java                                *
 *************************************************************************
 *                                                                       *
 * Project:          C Code Maintenance Tool                             *
 *                   B. Sc. Software Engineering                         *
 *                   4th Year Project                                    *
 *                                                                       *
 * Author:           Anna-Christina Friedrich                            *
 *                   C00132716                                           *
 *                   C00132716@itcarlow.ie                               *
 *                                                                       *
 * Supervisor:       Dr. Christophe Meudec                               *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class represents a variable declaration. It contains get and     *
 * set methods, a constructor and overwrites the 'toString()' method     *
 * and 'compareTo()' method which is needed for tree search and          *
 * insertion.                                                            *
 *                                                                       *
 *************************************************************************/

public class Identifier implements Comparable<Identifier> {

      /**variable is increased to create unique renaming values*/
      static private int I=0;

      /**the name of the variable*/
      private String name;

      /**new name for variable used in rename all*/
      private String rename;

      /**line where variable is declared*/
      private int line;

      /**column where variablename starts
       * not needed for now */
      private int column;

      /**name of the sourcefile in which variable is declared*/
      private String source;

      /**the binary tree to store all occurrences of the variable*/
      private Tree<Occurrence> occurrences;

      /**number of the scope in which the variable is defined*/
      private Integer scope;

      /**
       * Constructor of the class. Assigns values to attributes and
       * instantiates attributes.
       *
       * @param name : name of the variable
       * @param line : line in which variable is declared
       * @param col  : column where variable name starts
```

```
 * @param sour : name of sourcefile in which variable is declared
 * @param scope: number of the scope in which variable is declared
 */
public Identifier(String name, int line, int col, String sour,
                                                    Integer scope) {
      this.line=line;
      this.name=name;
      this.column=col;
      this.source=sour;
      this.occurrences = new Tree<Occurrence>();
      this.scope=scope;
      this.rename= "id"+String.valueOf(Identifier.I);
      Identifier.I++;
}

/**
 *
 * @return the name of the identifier
 */
public String getName() {
      return name;
}

/**
 *
 * @return declaration line of the identifier
 */
public int getLine() {
      return line;
}

/**
 * Specifies how the objects are stored in the tree using
 * line values and scope values. We have to use scope value as well,
 * so that identifier with the same name but in different scopes
 * can be stored, too.
 */
@Override
public int compareTo(Identifier id) {
      int returnval=1;
      if(this.name.compareTo(id.name)==0 &&
                  this.scope.compareTo(id.scope)==0)
            returnval=0;

      return returnval;
}

/**
 *
 * @return list of occurrences
 */
public Tree<Occurrence> getOccurrences() {
      return occurrences;
}

@Override
public String toString() {
      return "Identifier  "+ name +"\nscope=" + scope +
               ", line=" +line  +
               ", source=" + source + ", \nOccurrences=\n"
               + occurrences.traversForward().toString()+"\n\n";
```

```
	}

	/**
	 *
	 * @return the scope number in which variable is declared
	 */
	public int getScope() {
		return scope;
	}

	/**
	 *
	 * @return the new name of variable for renaming
	 */
	public String getRename() {
		return rename;
	}

	/**
	 *
	 * @param rename : the new name of variable for renaming
	 */
	public void setRename(String rename) {
		this.rename = rename;
	}

}
```

3. Function.java

```
/*************************************************************************
 *                          Function.java                                *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class represents a function declaration. It contains get and     *
 * set methods, a constructor and overwrites the 'toString()' method     *
 * and 'compareTo()' method which is needed for tree search and          *
 * insertion.                                                            *
 *                                                                       *
 *************************************************************************/
public class Function implements Comparable<Function>{

      /**variable is increased to create unique renaming values*/
      static private int I=0;

      /**the name of the function*/
      private String name;

      /**new name for function used in rename all*/
      private String rename;

      /**line where function is declared*/
      private int declaration_line;

      /**line where function is defined*/
      private int definition_line;

      /**column where functionname starts
       * not needed for now */
      private int column;

      /**name of the sourcefile in which function is declared*/
      private String source;

      /**the binary tree to store all occurrences of the function*/
      private Tree<Occurrence> occurrences;

      /**number of the scope in which the function is declared*/
      private Integer scope;


      /**
       * Constructor of the class. Assigns values to attributes and
       * instantiates attributes.
       *
```

```
 * @param name : name of the function
 * @param line : line in which function is declared
 * @param col  : column where function name starts
 * @param sour : name of sourcefile in which function is declared
 * @param scope: number of the scope in which function is declared
 */
public Function(String name, int line, int col, String sour,
                                                Integer scope){
      this.declaration_line=line;
      this.name=name;
      this.column=col;
      this.source=sour;
      this.occurrences = new Tree<Occurrence>();
      this.scope=scope;
      this.rename= "f"+String.valueOf(Function.I);
      Function.I++;
}


/**
 *
 * @return the name of the identifier
 */
public String getName() {
      return name;
}

/**
 * Specifies how the objects are stored in the tree
 * using line values.
 */
@Override
public int compareTo(Function fu) {
      return this.name.compareTo(fu.name);
}

/**
 *
 * @return list of occurrences
 */
public Tree<Occurrence> getOccurrences() {
      return occurrences;
}

@Override
public String toString() {
      return "Identifier  "+ name +"\nscope=" + scope +
               ", declaration line=" + declaration_line  +
               ", definition line="  + definition_line   +
               ", source=" + source + ", \noccurrences=\n"
      + occurrences.traversForward().toString()+"\n\n";
}

/**
 *
 * @return the scope number in which function is declared
 */
public int getScope() {
      return scope;
}
```

```
	/**
	 * Inserts the definition line as an occurrence.
	 *
	 * @param definitionLine : line number in which function is defined
	 */
	public void setDefinition_line(int definitionLine) {
		definition_line = definitionLine;
		this.occurrences.insert(new Occurrence(name, definitionLine,0,
							source, scope, rename));
	}

	/**
	 *
	 * @return the line in which function is defined
	 */
	public int getDefinition_line() {
		return definition_line;
	}

	/**
	 *
	 * @return the line in which function is declared
	 */
	public int getDeclaration_line() {
		return declaration_line;
	}

	/**
	 *
	 * @return the new name of function for renaming
	 */
	public String getRename() {
		return rename;
	}

	/**
	 *
	 * @param rename : the new name of function for renaming
	 */
	public void setRename(String rename) {
		this.rename = rename;
	}

}
```

4. Occurrence.java

```
/*************************************************************************
 *                        Occurrence.java                                *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class represents an occurrence of a function or variable        *
 * identifier. It contains get and set methods, a constructor and        *
 * overwrites the 'toString()' method and 'compareTo()' method which is *
 * needed for tree search and insertion.                                 *
 *                                                                       *
 *************************************************************************/

public class Occurrence implements Comparable<Occurrence> {

      /**the name of the identifier*/
      private String name;

      /**new name for identifier used in rename all*/
      private String rename;

      /**line where identifier occurs*/
      private int line;

      /**column where identifiername starts
       * not needed for now */
      private int column;

      /**name of the sourcefile in which identifier occurs*/
      private String source;

      /**number of the scope in which the identifier occurs*/
      private Integer scope;

      /**needed for generating html files to see whether the identifier
       * name is already written in one line */
      private boolean htmlreplaced;

      /**
       * Constructor of the class. Assigns values to attributes and
       * instantiates attributes.
       *
       * @param name   : name of the identifier
       * @param line   : line in which identifier occurs
       * @param col    : column where identifier name starts
       * @param sour   : name of sourcefile in which identifier occurs
       * @param scope  : number of the scope in which identifier occurs
```

```
 * @param rename : new name of the identifier, used in rename all
 */
public Occurrence(String name, int line, int col, String sour,
                                            Integer scope, String rename) {

      this.name = name;
      this.line=line;
      this.column=col;
      this.source=sour;
      this.scope=scope;
      this.rename=rename;
      this.htmlreplaced=false;
}

@Override
public String toString() {
      return "Occurrence "+name+" : scope=" + scope +", line=" +
                  line +", source=" + source + "\n";
}

/**
 *
 * @return line number in which identifier is defined
 */
public int getLine() {
      return line;
}

/**
 * Specifies how the objects are stored in the tree
 * using line values.
 */
@Override
public int compareTo(Occurrence occ) {
      return this.line-occ.line;
}

/**
 *
 * @return the name of the identifier
 */
public String getName() {
      return name;
}

/**
 *
 * @return the new name of identifier for renaming
 */
public String getRename() {
      return rename;
}

/**
 *
 * @param rename : the new name of identifier for renaming
 */
public void setRename(String rename) {
      this.rename = rename;
}
```

```
	/**
	 * Needed for generating html files.
	 *
	 * @return whether identifier is written already in one line
	 */
	public boolean isHtmlreplaced() {
		return htmlreplaced;
	}

	/**
	 * Needed for generating html files.
	 *
	 * @param htmlreplaced : whether identifier is written already in one line
	 */
	public void setHtmlreplaced(boolean htmlreplaced) {
		this.htmlreplaced = htmlreplaced;
	}

}
```

5. Scope.java

```
/**************************************************************************
 *                              Scope.java                                *
 **************************************************************************
 *                                                                        *
 * Project:         C Code Maintenance Tool                               *
 *                  B. Sc. Software Engineering                           *
 *                  4th Year Project                                      *
 *                                                                        *
 * Author:          Anna-Christina Friedrich                              *
 *                  C00132716                                             *
 *                  C00132716@itcarlow.ie                                 *
 *                                                                        *
 * Supervisor:      Dr. Christophe Meudec                                 *
 *                                                                        *
 * Summary:                                                               *
 *                                                                        *
 * This class represents the scope in a sourcefile. It consists of an    *
 * integer attribute as a unique id for each scope and of a boolean       *
 * attribute, which indicates if the next scope on the stack needs to be*
 * popped as well.                                                        *
 *                                                                        *
 **************************************************************************/

public class Scope {

      /**unique number of the scope*/
      private int number;

      /**indicates whether one or two times popping from stack*/
      private boolean popTwice;

      /**
       * Constructor assigns passed arguments to attributes.
       *
       * @param idNumber : number of the scope
       * @param popTwice : decision variable if you pop once or twice
       */
      public Scope(int idNumber, boolean popTwice) {
            this.number = idNumber;
            this.popTwice = popTwice;
      }

      /**
       *
       * @return whether to pop twice or not
       */
      public boolean isPopTwice() {
            return popTwice;
      }

      /**
       *
       * @param popTwice : pop twice or not
       */
      public void setPopTwice(boolean popTwice) {
            this.popTwice = popTwice;
      }
```

```
	/**
	 *
	 * @return the number of the scope
	 */
	public int getNumber() {
		return number;
	}

}
```

6. Tree.java

```
/*************************************************************************
 *                              Tree.java                                *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class represents a binary tree. It is flexible in using because *
 * generics are used in the tree as well as in the nodes of the tree.    *
 * Methods for insertion, searching and traversion are implemented.      *
 *                                                                       *
 *************************************************************************/

/**
 * imported packages
 */
import java.util.ArrayList;


/**
 * class definition
 */
public class Tree<E extends Comparable<E>>{

      /**represents the root of the tree*/
      private Node<E> root = null;

      /**
       *
       * @return whether tree is empty or not
       */
      boolean isEmpty(){ return root == null;}

      /**
       * Inserts content e into a node inside the tree
       *
       * @param e content for the node
       */
      void insert(E e){
            if (root == null) root = new Node<E>(e);
            else rekInsert(e, root);
      }

      /**
       * The right position is found recursively.
       *
       * @param e : content for the node
       * @param k : parent node where to insert content
```

```
 */
void rekInsert(E e, Node<E> k){
      if (e.compareTo(k.content) <= 0)
            if (k.l == null) k.l = new Node<E>(e, k);
            else rekInsert(e, k.l);
      else
            if (k.r == null) k.r = new Node<E>(e, k);
            else rekInsert(e, k.r);
}

/**
 * Searches for the minimum of the tree recursively.
 *
 * @return node containing minimum content
 */
Node<E> searchMin(){ return searchMin(root);}
Node<E> searchMin(Node<E> kp){
      if (kp == null) return kp;
      while (kp.l != null) kp = kp.l;
      return kp;
}

/**
 * Searches for the maximum of the tree recursively.
 *
 * @return node containing maximum content
 */
Node<E> searchMax(){ return searchMax(root); }
Node<E> searchMax(Node<E> kp){
      if (kp == null) return kp;
      while (kp.r != null) kp = kp.r;
      return kp;
}

/**
 * Searches for the successor of a given node.
 *
 * @param kp : node for which successor is needed
 * @return the successor of the passed node
 */
Node<E> successor(Node<E> kp){
      if (kp == null) return kp;
      if (kp.r != null) return searchMin(kp.r);
      Node<E> above = kp.a;
      while((above != null) && (above.r == kp)) {
            kp = above;
            above = kp.a;
      }
      return above;
}

/**
 * Searches for the predecessor of a given node.
 *
 * @param kp : node for which predecessor is needed
 * @return the predecessor of the passed node
 */
Node<E> predecessor(Node<E> kp){
      if (kp == null) return kp;
      if (kp.l != null) return searchMax(kp.l);
      Node<E> above = kp.a;
```

```
      while((above != null) && (above.l == kp)) {
            kp = above;
            above = kp.a;
      }
      return above;
}

/**
 * Deletes a node from the tree recursively.
 *
 * @param kp : the node that is to be deleted
 */
void deleteNode(Node<E> kp){
      if (kp == null) return;
      if (kp.l == null || kp.r == null) delete1(kp);
      else delete2(kp);
}

/**
 * Helps delete method if there is just one child node.
 *
 * @param child : child of the node that is to be deleted
 */
void delete1(Node<E> child){  // deletes node having max one child
      // collect grandchild - can be null
      Node<E> grandChild = (child.l == null) ? child.r : child.l;
if (child == root)  { root = grandChild;  root.a = null; return; }
// now it's obvious- father exists
Node<E> father = child.a;
//connect father with grandchild
if (father.l == child) father.l = grandChild;
else father.r = grandChild;
// connect grandchild with father
if (grandChild != null) grandChild.a = father;
}

/**
 * Helps delete method if there are two child nodes.
 *
 * @param kp : node with two childs that is to be deleted
 */
void delete2(Node<E> kp){
      Node<E> min = searchMin(kp.r);
      kp.content = min.content; // copy content up
      delete1(min);
}

/**
 * Method to start recursive search at the root node.
 *
 * @param e : content we search for
 * @return the result of recursive search
 */
Node<E> search(E e){ return rekSearch(e, root);}

/**
 * Decides into which direction the search goes on- left or right.
 *
 * @param e : content of the node we search for
 * @param k : node where search starts
 * @return in the end the node that was searched for or null
```

```
	 */
	Node<E> rekSearch(E e, Node<E> k){
		if (k == null) return null;
		if (e.compareTo(k.content) == 0) return k;
		if (e.compareTo(k.content) < 0) return rekSearch(e, k.l);
		else return rekSearch(e, k.r);
	}

	/**
	 * Traverses the tree forward.
	 *
	 * @return a list containing all nodes in a particular order
	 */
	ArrayList<Node<E>> traversForward(){
		ArrayList<Node<E>> nodes= new ArrayList<Node<E>>();
		Node<E> kp = searchMin();
		while (kp != null) {
			nodes.add(kp);
			kp = successor(kp);
		}
		return nodes;
	}

	/**
	 * Prints the nodes of the tree backwards on the console.
	 *
	 */
	void outputBackwards(){
		System.out.println("output backwards");
		Node<E> kp = searchMax();
		while (kp != null) {
			System.out.println(kp);
			kp = predecessor(kp);
		}
		System.out.println();
	}
}
```

7. Node.java

```
/**************************************************************************
 *                        Node.java                                       *
 **************************************************************************
 *                                                                        *
 * Project:         C Code Maintenance Tool                               *
 *                  B. Sc. Software Engineering                           *
 *                  4th Year Project                                      *
 *                                                                        *
 * Author:          Anna-Christina Friedrich                              *
 *                  C00132716                                             *
 *                  C00132716@itcarlow.ie                                 *
 *                                                                        *
 * Supervisor:      Dr. Christophe Meudec                                 *
 *                                                                        *
 * Summary:                                                               *
 *                                                                        *
 * This class represents a node of the binary tree. Each node has         *
 * content, which can be of any type, one parent node, one left and one *
 * right node.                                                            *
 *                                                                        *
 **************************************************************************/

public class Node<E extends Comparable<E>>{
   E content;
   Node<E> a, l, r;
   Node(E el){ content = el;}
   Node(E el, Node<E> above){ content = el; a = above;}
   public String toString(){  return content.toString();}
}
```

8. Rename.java

```
/*************************************************************************
 *                            Rename.java                                *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class offers two different constructors. One is for renaming     *
 * one variable identifier, the other one is for renaming all identifier*
 * of a whole file using the first constructor.                          *
 *                                                                       *
 *************************************************************************/

/**
 * imported packages
 */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;

/**
 * class definition
 */
public class Rename {

      /**String that is put to the end of a filename to indicate
       * that it is modified*/
      final static public String mod = "Modified";

      /**name of the source file that is to be renamed*/
      private String filename;

      /**number of the scope the identifier is defined*/
      private int scope;

      /**name of the identifier*/
      private String id;

      /**new name for the identifier*/
      private String newId;

      /**name of the new generated file*/
      private String newfile;

      /**
```

```
 * First Constructor which reads the source file, walks through it
 * line by line and compares each word of one line with certain
 * Identifier and its occurrences.
 *
 *
 * @param filena : name of the source file in which identifier is
 *                 defined
 * @param sco    : scope of the identifier definition
 * @param iden   : name of the identifier
 * @param newI   : new name for the identifier
 */
public Rename(String filena, int sco, String iden, String newI) {
      this.filename = filena;
      this.scope = sco;
      this.id = iden;
      this.newId=newI;

      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object

      // create a temporary file
      File tempf = null;
      try {
            tempf = File.createTempFile("temp","temper");
      } catch (IOException e1) {e1.printStackTrace();}

      // copy the content of the source file to the temporary file
      // so that we can work on the temporary file and the original
      // file remains unchanged
      Cp.copyFile(filename, tempf.getAbsolutePath());

      try {
            //write into the temporary file
            out = new FileOutputStream(tempf.getAbsolutePath());

            // Connect print stream to the output stream
            p = new PrintStream( out );

            BufferedReader in = new BufferedReader(
                              new FileReader(filename));
            String line = null;
            int linenumber=1;

            Node<Identifier> identifier=CMT.idTree.search(
                                          new Identifier(id,0,0,"",scope));

            if(identifier!=null){
                  // for each line...
                  while ((line = in.readLine()) != null) {

                        Splitter spl=new Splitter("\\W+");
                        String[] linearray = spl.split(line);
                        String composedline=new String();

              // if there is a self written include file - rename it
                        for(int ind=0;ind<linearray.length;ind++){

                          if(line.startsWith("#include")){

                              for(File file:CMT.filenames){
                                    String filename=file.getName();
```

```
                                    filename=filename.substring(0,
                                          filename.length()-2);
                                    if(linearray[ind].equals(filename)){
                                          linearray[ind]+=mod;
                                    }
                               }
                          }composedline+=linearray[ind];
                     }
                     line=composedline;

                     String[] idrename = spl.split(line);
                     String idcomposed=new String();

               // go through each word of the line and compare with
               // identifier name
               if(linenumber==identifier.content.getLine()){
                    for(int index=0;index<idrename.length;index++){
                                    if(idrename[index].equals(id))
                                          idrename[index]=newId;
                                    idcomposed+=idrename[index];
                               }
                               line=idcomposed;
                     }

                     // rename identifier occurrences the same way
                     // as identifier
                     for(Node<Occurrence> o:
                    identifier.content.getOccurrences().traversForward()){

                    Splitter splo=new Splitter("\\W+");
                               String[] linearrOcc = splo.split(line);
                               String occComposed=new String();
                               if(linenumber==o.content.getLine()){
                      for(int index=0;index<idrename.length;index++){
                             if(linearrOcc[index].equals(id))
                                 linearrOcc[index]=newId;
                             occComposed+=linearrOcc[index];
                                  }
                                  line=occComposed;
                               }
                     }
                     p.println(line);
                     linenumber++;
                }
           }
           p.close();
     } catch (IOException e) {e.printStackTrace();}

     // create name of modified file
     if(!filename.contains(mod)){
           if(filename.endsWith(CMT.cSuffix))
          newfile=filename.replace(CMT.cSuffix, mod+CMT.cSuffix);
            else if(filename.endsWith(CMT.hSuffix))
          newfile=filename.replace(CMT.hSuffix, mod+CMT.hSuffix);
     }
     else newfile=filename;

// now copy the content of the temporary file, where identifier
// are renamed, to the new generated file
     Cp.copyFile(tempf.getAbsolutePath(), newfile);
```

```
	}

	/**
	 * Constructor which renames all identifier of a source file.
	 *
	 * @param filenam : name of the source file in which identifier
	 *                  is defined
	 */
	public Rename(String filenam){
		this.filename = filenam;

		// use first constructor to rename each identifier on its own
		for(Node<Identifier> temp:CMT.idTree.traversForward()){
			Rename re=new Rename(filename, temp.content.getScope(),
				temp.content.getName(),temp.content.getRename());
			filename=re.newfile;
		}
		new RenameFunction(filename);
	}

}
```

9. RenameFunction.java

```
/*************************************************************************
 *                        RenameFunction.java                            *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Author:          Anna-Christina Friedrich                             *
 *                  C00132716                                            *
 *                  C00132716@itcarlow.ie                                *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class offers two different constructors. One is for renaming     *
 * one function identifier, the other one is for renaming all function   *
 * identifier of a whole file using the first constructor.               *
 *                                                                       *
 *************************************************************************/

/**
 * imported packages
 */
import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;

/**
 * class definition
 */
public class RenameFunction {

      /**name of the source file that is to be renamed*/
      private String filename;

      /**number of the scope the identifier is defined*/
      private int scope;

      /**name of the identifier*/
      private String id;

      /**new name for the identifier*/
      private String newId;

      /**name of the new generated file*/
      private String newfile;

      /**
       * First Constructor which reads the source file, walks through it
       * line by line and compares each word of one line with certain
       * Identifier and its occurrences.
       *
```

```
 *
 * @param filena : name of the source file in which identifier is
 *                 defined
 * @param sco    : scope of the identifier definition
 * @param iden   : name of the identifier
 * @param newI   : new name for the identifier
 */
public RenameFunction(String filena, int sco, String iden, String newI){
      super();
      this.filename = filena;
      this.scope = sco;
      this.id = iden;
      this.newId=newI;

      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object

      // create a temporary file
      File tempf = null;
      try {
            tempf = File.createTempFile("temp","temper");
      } catch (IOException e1) {e1.printStackTrace();}

      // copy the content of the source file to the temporary file
      // so that we can work on the temporary file and the original
      // file remains unchanged
      Cp.copyFile(filename, tempf.getAbsolutePath());

      try {
            //write into the temporary file
            out = new FileOutputStream(tempf.getAbsolutePath());

            // Connect print stream to the output stream
            p = new PrintStream( out );

            BufferedReader in = new BufferedReader(
                                          new FileReader(filename));
            String line = null; int linenumber=1;
            Node<Function> identifier=CMT.functionTree.search(
                                    new Function(id,0,0,"",scope));
            if(identifier!=null){
                  // for each line...
                  while ((line = in.readLine()) != null) {
                        Splitter spl=new Splitter("\\W+");
                        String[] linearray = spl.split(line);
                        String composedline=new String();

                  // if there is a self written include file - rename it
                        for(int ind=0;ind<linearray.length;ind++){

                              if(line.startsWith("#include")){

                                    for(File file:CMT.filenames){
                                      String filename=file.getName();
                                      filename=filename.substring(0,
                                                      filename.length()-2);
                                      if(linearray[ind].equals(filename)){
                                          linearray[ind]+=Rename.mod;
                                      }
                                    }
                              }composedline+=linearray[ind];
```

```
                    }
                    line=composedline;

                    String[] idrename = spl.split(line);
                    String idcomposed=new String();

                    // go through each word of the line and compare
                    // with identifier name
                    if((linenumber==
                         identifier.content.getDeclaration_line()
                         ||
                         linenumber==
                         identifier.content.getDefinition_line())
                         &&
                         !identifier.content.getName().equals("main"))
                          {

                       for(int index=0;index<idrename.length;index++){

                                if(idrename[index].equals(id))
                                      idrename[index]=newId;
                                idcomposed+=idrename[index];

                       }
                       line=idcomposed;
                    }

                    // rename identifier occurrences the same way
                    // as identifier
                    for(Node<Occurrence> o:
                      identifier.content.getOccurrences()
                                                .traversForward()){

                      Splitter splo=new Splitter("\\W+");
                      String[] linearrOcc = splo.split(line);
                      String occComposed=new String();
                      if(linenumber==o.content.getLine()){
                        for(int index=0;index<idrename.length;index++)
                        {
                                if(linearrOcc[index].equals(id))
                                            linearrOcc[index]=newId;
                                occComposed+=linearrOcc[index];
                        }
                        line=occComposed;
                      }
                    }
                    p.println(line);

                    linenumber++;

              }
         }
         p.close();
} catch (IOException e) {e.printStackTrace();}

// create name of modified file
if(!filename.contains(Rename.mod)){
  if(filename.contains(CMT.cSuffix))
    newfile=filename.replace(CMT.cSuffix,Rename.mod+CMT.cSuffix);
  else if(filename.contains(CMT.hSuffix))
    newfile=filename.replace(CMT.hSuffix,Rename.mod+CMT.hSuffix);
```

```
            }
            else newfile=filename;

            // now copy the content of the temporary file, where identifier
            // are renamed, to the new generated file
            Cp.copyFile(tempf.getAbsolutePath(), newfile);

      }

      /**
       * Constructor which renames all identifier of a source file.
       *
       * @param filenam : name of the source file in which identifier
       *                  is defined
       */
      public RenameFunction(String filenam){
            this.filename = filenam;

            // use first constructor to rename each identifier on its own
            for(Node<Function> temp:CMT.functionTree.traversForward()){
                  RenameFunction re=new RenameFunction(
                        filename,temp.content.getScope(),temp.content.getName(),
                        temp.content.getRename());
                  filename=re.newfile;
            }
      }

}
```

10. ComLineInterpreter.java

```
/**************************************************************************
 *                  ComLineInterpreter.java                               *
 **************************************************************************
 *                                                                        *
 * Project:         C Code Maintenance Tool                               *
 *                  B. Sc. Software Engineering                           *
 *                  4th Year Project                                      *
 *                                                                        *
 * Author:          Anna-Christina Friedrich                              *
 *                  C00132716                                             *
 *                  C00132716@itcarlow.ie                                 *
 *                                                                        *
 * Supervisor:      Dr. Christophe Meudec                                 *
 *                                                                        *
 * Summary:                                                               *
 *                                                                        *
 * This is the class that interprets the passed arguments from command  *
 * prompt. First of all arguments that occur in couples are filtered,   *
 * like 'src', 'id', 'scope' and 'newid'. Afterwards the arguments where*
 * action happens are parsed, like creating files and renaming.          *
 *                                                                        *
 **************************************************************************/


/**
 * imported packages
 */
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;
import org.antlr.runtime.ANTLRFileStream;
import org.antlr.runtime.CharStream;
import org.antlr.runtime.CommonTokenStream;
import org.antlr.runtime.RecognitionException;
import org.antlr.runtime.TokenStream;

/**
 * class definition
 */
public class ComLineInterpreter {

      /**used inside grammar, whether renaming or not*/
      static public boolean renameTag=false;

      /**used inside grammar, whether renaming all or not*/
      static public boolean renameAllTag=false;

      /**used inside grammar, whether text file or not*/
      static public boolean textTag=false;

      /**used inside grammar, whether Html file or not*/
      static public boolean htmlTag=false;
```

```
/**to store commands where action takes place*/
private ArrayList<String> commandLine=new ArrayList<String>();

/**name of the file that is currently parsed*/
private static String filename;

/**for renaming: name of identifier*/
private static String idname;

/**for renaming: new name for identifier*/
private static String newidname;

/**for renaming: number of the scope the identifier is in*/
private static int scope=-1;

/**
 * Constructor of the class
 * assigns values from command prompt to attributes
 *
 * @param commandLine : array including arguments passed via
 *                                  command prompt
 */
public ComLineInterpreter(String[] commandLine) {
      File file;
      if(commandLine[0]==null)writeInfo();
      else{
            for(int i=0;i<commandLine.length;i++){

                  if(commandLine[i].equals("src")){
                        i++;
                        filename=commandLine[i];
                        file=new File(filename);
                        CMT.filenames.add(file);
                  }else if(commandLine[i].equals("id")){
                        i++;
                        idname=commandLine[i];
                  }else if(commandLine[i].equals("scope")){
                        i++;
                        scope=Integer.parseInt(commandLine[i]);
                  }else if(commandLine[i].equals("newid")){
                        i++;
                        newidname=commandLine[i];
                  }else this.commandLine.add(commandLine[i]);
            }
            doIt();
      }
}

/**
 * First the commands are iterated to set the tags, this has to be
 * done before action happens. Afterwards the trees are filled and
 * commands are executed. If nothing is passed or 'info' a syntax
 * information is displayed.
 */
private void doIt() {
      for(String command:this.commandLine){
            if(command.equals("createText")){
                  textTag=true;
            }else if(command.equals("renameAll")){
                  renameAllTag=true;
```

```
            }else if(command.equals("createHtml")){
                  htmlTag=true;
            }else if(command.equals("rename")){
                  renameTag=true;
            }
      }
      if(!commandLine.isEmpty()){
            if(commandLine.get(0).equals("info")){
                  writeInfo();
            }else{
                  try {
                        fillTree(filename);
                  } catch (RecognitionException e1) {
                        e1.printStackTrace();
                  } catch (IOException e1) {
                        e1.printStackTrace();
                  }
                  for(String command:this.commandLine){
                        if(command.equals("createText")){
                              createText(filename);

                        }else if(command.equals("renameAll")){
                              renameAll(filename);

                        }else if(command.equals("createHtml")){
                              writeHtml(filename);

                        }else if(command.equals("rename")){
                              rename(filename);
                        }
                  }
            }
      }
}

/**
 * Writes syntax information to the console.
 */
public static  void writeInfo() {
      System.out.println("\n************************SYNTAX*********" +
      "***************************************");
      System.out.println("*                                                " +
      "                                *");
      System.out.println("* Pass a selection of following syntax as " +
      "arguments of the jar file.           *");
      System.out.println("*                                                " +
      "                                *");
      System.out.println("* -specify the location of the C file; nec" +
      "cessary for each execution!          *");
      System.out.println("*       src 'path to the C file'                 " +
      "                                *");
      System.out.println("*                                                " +
      "                                *");
      System.out.println("* -create a textfile as a listing of all i" +
      "dentifier                            *");
      System.out.println("*       createText                               " +
      "                                *");
      System.out.println("*                                                " +
      "                                *");
      System.out.println("* -create an html file to browse through t" +
      "he C file                            *");
```

```
      System.out.println("*       createHtml                          " +
      "                                *");
      System.out.println("*                                           " +
      "                                *");
      System.out.println("* -rename all identifier of the C file      " +
      "                                *");
      System.out.println("*       renameAll                           " +
      "                                *");
      System.out.println("*                                           " +
      "                                *");
      System.out.println("* -rename one identifier of the C file;     " +
      "                                *");
      System.out.println("*  first create textfile to see the scopen" +
      "umber of the identifier!        *");
      System.out.println("*       rename id 'the name of identifier'" +
      "                                *");
      System.out.println("*       newid 'new name of identifier'      " +
      "                                *");
      System.out.println("*       scope 'number of scope'             " +
      "                                *");
      System.out.println("*                                           " +
      "                                *");
      System.out.println("* You can arrange the order of your comman" +
      "d as you like.                  *");
      System.out.println("* But make sure that src, id, newid and sc" +
      "ope are always                  *");
      System.out.println("* followed by their value.                  " +
      "                                *");
      System.out.println("*                                           " +
      "                                *");
      System.out.println("*******************************************" +
      "*********************************");
}

/**
 * Checks whether a variable or a function is to be renamed and
 * creates corresponding objects.
 *
 * @param filename : name of the file that is currently parsed
 */
public static void rename(String filename) {

      if(idname!=null && newidname!=null && scope!=-1){
            Node<Identifier> temp=CMT.idTree.search(new Identifier

(idname,0, 0, "",scope));
            if(temp!=null)new Rename(filename, scope, idname, newidname);
            else new RenameFunction(filename, scope, idname, newidname);
      }else System.err.println("Use 'info' to see the syntax.");
      System.out.println(filename+": Renaming is ready.");
}

/**
 * The C file is parsed here. First the global scope of the file is
 * pushed on the scope stack, then the Lexer object creates a
 * token stream, which is passed to the Parser object. To start the
 * parsing the initiation rule of the grammar file needs to be called.
 *
 * @param filename : name of the file that is currently parsed
 * @throws RecognitionException
 * @throws IOException
```

```
 */
public static void fillTree(String filename)
                                throws RecognitionException, IOException{

      CMT.scopeStack.push(new Scope(CMT.scope,false));

      try{
            CharStream charStream = new ANTLRFileStream(filename);
            CLexer lexer = new CLexer(charStream );
            TokenStream tokenStream = new CommonTokenStream(lexer);
            CParser parser = new CParser(tokenStream );
            parser.translation_unit();
      }catch(FileNotFoundException fe){
            System.err.println("Cannot find the file: "+filename);
      }
}

/**
 * Creates a new text file and prints the variable list, function list
 * and all file paths to the text file.
 *
 * @param filename : name of the file that is currently parsed
 */
public static void createText(String filename){
      String newfile = null;
      if(filename.endsWith(CMT.cSuffix))
            newfile=filename.replace(CMT.cSuffix, "c"+CMT.txtSuffix);
      if(filename.endsWith(CMT.hSuffix))
            newfile=filename.replace(CMT.hSuffix, "h"+CMT.txtSuffix);
      FileOutputStream out; // declare a file output object
      PrintStream p; // declare a print stream object

      try
      {
            // Create a new file output stream

            out = new FileOutputStream(newfile);

            // Connect print stream to the output stream
            p = new PrintStream( out );

            p.println("VARIABLE LIST");
            p.println();
            p.println (CMT.idTree.traversForward().toString());
            p.println();
            p.println("FUNCTION LIST");
            p.println();
            p.println (CMT.functionTree.traversForward().toString());
            p.println();
            p.println("FILE PATHS");
            p.println();
            for(File f:CMT.filenames){
                  p.println (f.toString());
            }
            p.close();
      }
      catch (Exception e)
      {
            System.err.println ("Error writing to file");
      }
      System.out.println(filename+": Textfile is ready.");
```

```
	}

	/**
	 * Creates a Rename object to rename all identifier of the current
	 * file 'filename'
	 *
	 * @param filename : name of the file that is currently parsed
	 */
	public static void renameAll(String filename){
		new Rename(filename);
	}

	/**
	 * Creates a new html file for the currently parsed source file.
	 * Furthermore the javascript library 'prototype.js' is copied into
	 * the same folder.
	 *
	 * @param filename : name of the file that is currently parsed
	 */
	public static void writeHtml(String filename){

		String  newline;
		File outFile;
		FileWriter fw;
		BufferedWriter bw;

		newline = System.getProperty("line.separator");
		try {

			outFile = new File(filename+".html");
			fw = new FileWriter(outFile);
			bw = new BufferedWriter(fw);
			Html.write(bw,newline,filename);
			//copy prototype into folder
			File f=new File(filename);
			String proto=f.getPath().substring(0, f.getPath().length()-
			                                      f.getName().length());
			proto+="prototype.js";
			f=new File(proto);
			if(!f.exists())
				Cp.copyFile("prototype.js", proto);

			bw.close();
		}
		catch (IOException ioe) { }
		System.out.println(filename+": Html is ready.");
	}

}
```

11. Html.java

```
/**************************************************************************
 *                              Html.java                                 *
 **************************************************************************
 *                                                                        *
 * Project:         C Code Maintenance Tool                               *
 *                  B. Sc. Software Engineering                           *
 *                  4th Year Project                                      *
 *                                                                        *
 * Author:          Anna-Christina Friedrich                              *
 *                  C00132716                                             *
 *                  C00132716@itcarlow.ie                                 *
 *                                                                        *
 * Supervisor:      Dr. Christophe Meudec                                 *
 *                                                                        *
 * Summary:                                                               *
 *                                                                        *
 * This class writes the html output file. Html tags, CSS and             *
 * Javascript are written in the file. Making it possible to click on     *
 * an identifier name, identifier are assigned by using the tree          *
 * datastructure.                                                         *
 *                                                                        *
 **************************************************************************/

/**
 * imported packages
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.IOException;

/**
 * class definition
 */
public class Html {

      static void write(BufferedWriter bw, String newline, String code)
      throws IOException {
            //starting html
            bw.write("<!DOCTYPE html PUBLIC \"-//W3C//DTD XHTML 1.0 " +
                        "Transitional//EN\" \"http://www.w3.org/TR/xhtml1/" +
                        "DTD/xhtml1-transitional.dtd\">");
            bw.write(" <head> "+newline);
            bw.write(" <title> "+newline);
            bw.write(code+" "+newline);
            bw.write(" </title> "+newline);

            //starting CSS
            bw.write("<style type='text/css'>"+newline);
            bw.write("ul {"+newline);
            bw.write("padding: 0;"+newline);
            bw.write("margin: 0;"+newline);
            bw.write("list-style: none;"+newline);
            bw.write("}"+newline);
            bw.write("li {"+newline);
            bw.write("float: left;"+newline);
            bw.write("position: relative;"+newline);
```

```
bw.write("cursor:pointer;"+newline);
bw.write(" }"+newline);
bw.write("li ul {"+newline);
bw.write("width: 15em;"+newline);
bw.write("display: none;"+newline);
bw.write("position: absolute; "+newline);
bw.write("top: 1em;"+newline);
bw.write("left: 0;"+newline);
bw.write("background-color: white;"+newline);
bw.write("border: 1px solid #7d6340;"+newline);
bw.write("border-width: 1px 0;"+newline);
bw.write("}"+newline);
bw.write("li > ul {"+newline);
bw.write("top: auto;"+newline);
bw.write("left: auto;"+newline);
bw.write("}"+newline);
bw.write("a:link, a:visited, a:active, a:hover"+newline);
bw.write("{ "+newline);
bw.write("text-decoration:underline;"+newline);
bw.write("cursor:pointer;"+newline);
bw.write("}"+newline);
bw.write("li:hover ul, li.over ul{ display: block;}"+newline);
bw.write("</style>"+newline);
// end of CSS

// starting javascript
bw.write("<script type='text/javascript' src='prototype.js'>" +
          "</script>"+newline);
bw.write("<script language='javascript'>"+newline);
bw.write("var oldid;"+newline);
bw.write("function getNext(id){"+newline);
bw.write("var info=[];"+newline);
bw.write("var nullinfo=[];"+newline);
bw.write("info=id.split('z');"+newline);
bw.write("var i;"+newline);
bw.write("var keys = hash.keys();"+newline);
bw.write("var newid;"+newline);
bw.write("for( i in keys ){"+newline);
bw.write("if(info[0]==keys[i]){"+newline);
bw.write("newid=keys[i];"+newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("var num=parseInt(hash.get(newid));"+newline);
bw.write("var fromhtml=parseInt(info[1]);"+newline);
bw.write("if( fromhtml<num-1)num=fromhtml+1;"+newline);
bw.write("else num=-1;"+newline);
bw.write("id=info[0]+'z'+num;"+newline);
bw.write("if(oldid != null)"+newline);
bw.write("$(oldid).style.backgroundColor = 'white';"+newline);
bw.write("$(id).style.backgroundColor = 'yellow';"+newline);
bw.write("window.location.href='#'+id;"+newline);
bw.write("oldid=id;"+newline);
bw.write("}"+newline);
bw.write("function getPrev(id){"+newline);
bw.write("var info=[];"+newline);
bw.write("var nullinfo=[];"+newline);
bw.write("info=id.split('z');"+newline);
bw.write("var i;"+newline);
bw.write("var keys = hash.keys();"+newline);
bw.write("var newid;"+newline);
bw.write("for( i in keys ){"+newline);
```

```
bw.write("if(info[0]==keys[i]){"+newline);
bw.write("newid=keys[i];"+newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("var num=parseInt(hash.get(newid));"+newline);
bw.write("var fromhtml=parseInt(info[1]);"+newline);
bw.write("if( fromhtml>-1)num=fromhtml-1;"+newline);
bw.write("else num=parseInt(hash.get(newid))-1;"+newline);
bw.write("id=info[0]+'z'+num;"+newline);
bw.write("if(oldid != null)"+newline);
bw.write("$(oldid).style.backgroundColor = 'white';"+newline);
bw.write("$(id).style.backgroundColor = 'yellow';"+newline);
bw.write("window.location.href='#'+id;"+newline);
bw.write("oldid=id;"+newline);
bw.write("}"+newline);
bw.write("startList = function() {"+newline);
bw.write("if (document.all&&document.getElementById) {"+newline);
bw.write("navRoot = document.getElementById('nav');"+newline);
bw.write("for(i=0; i<navRoot.childNodes.length; i++) {"+newline);
bw.write("node = navRoot.childNodes[i];"+newline);
bw.write("if (node.nodeName=='LI') {"+newline);
bw.write("node.onmouseover=function() {"+newline);
bw.write("this.className+='over';"+newline);
bw.write("}"+newline);
bw.write("node.onmouseout=function() {"+newline);
bw.write("this.className=this.className.replace('over','');"
                                                    +newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("}"+newline);
bw.write("window.onload=startList;"+newline);
bw.write(""+newline);
// fill Hash structure with unique ids and their number
// of occurrences
bw.write("var hash = new Hash({"+newline);

for(Node<Identifier> id: CMT.idTree.traversForward() ){
      bw.write(id.content.getRename()+":"
            +id.content.getOccurrences().traversForward().size()
                  +","+newline);
}
for(Node<Function> id: CMT.functionTree.traversForward() ){

      bw.write(id.content.getRename()+":"
            +id.content.getOccurrences().traversForward().size()
                  +","+newline);
}

bw.write("})"+newline);
bw.write("</script>"+newline);
// end of javascript

bw.write("</head>"+newline);
bw.write(" <body> "+newline);
bw.write(" "+newline);
bw.write("<ul  id='nav'>"+newline);

// write the sourcefile
BufferedReader in = new BufferedReader(new FileReader(code));
```

```
String line = null;
int row=1;
// for each line..
while ((line = in.readLine()) != null) {

      Splitter spl=new Splitter("\\W+");
      String[] lineArray = spl.split(line);
      String help=new String();
      // ...split the line into a string array
      // put each word into li tag so that everything
      // has the right order
      for(String word:lineArray){

            help+="<li>"+word+"</li>";

      }
      lineArray=spl.split(help);
      boolean insidePrint=false;

      // for each variable identifier...
      for(Node<Identifier> id:CMT.idTree.traversForward()){
            // if printf - jump over it
            for(int i=0;i<lineArray.length;i++){

                  if(lineArray[i].contains("printf")){

                        i+=7;
                        insidePrint=true;

                  }
                  if(lineArray[i].contains("\""))insidePrint=false;
                  // compare each array element with each identifier
                  // and if it matches assign onCLick events
                  if(row==id.content.getLine() &&
                        lineArray[i].equals(id.content.getName()) &&
                        !insidePrint){

                     lineArray[i-2]+="id='"+id.content.getRename()+
                                          "z-1'";
                     lineArray[i]+="<ul>"
                        +"<li><a  onClick=\"getNext('"+
                        id.content.getRename()+"z-1');\">show next"
                        +"</a>"
                        +"  <a  onClick=\"getPrev('"+
                        id.content.getRename()+"z-1');\">show " +
                        "previous</a></li>"
                        +"</ul> ";
                  }

                  // for each array element in this line compare
                  // with each occurrence
                  // and if it matches assign onCLick events
                  int num=0;
                  for(Node<Occurrence> o:
                     id.content.getOccurrences().traversForward()){
                        if(row==o.content.getLine() &&
                         lineArray[i].equals(o.content.getName()) &&
                         !insidePrint){

                              if(o.content.isHtmlreplaced())
                                    num=id.content.getOccurrences().
```

```
                        traversForward().indexOf(o)+1;
                  else
                        num=id.content.getOccurrences().
                        traversForward().indexOf(o);
                  lineArray[i-2]+=" id='"+
                         id.content.getRename()
                         +"z"+num+"'";
                  lineArray[i]+="<ul>"
                        +"<li><a  onClick=\"getNext('"+
                        id.content.getRename()
                        +"z"+num+"');\">" +
                        "show next</a>"
                        +"  <a  onClick=\"getPrev('"+
                        id.content.getRename()
                        +"z"+num+"');\">" +
                        "show previous</a></li>"
                        +"</ul> ";

                  o.content.setHtmlreplaced(true);

            }
      }

   }
}
// for each function identifier...
for(Node<Function> func:CMT.functionTree.traversForward()){
   // if printf - jump over it
   for(int i=0;i<lineArray.length;i++){
      if(lineArray[i].contains("printf")){
         i+=7;
         insidePrint=true;
      }
      if(lineArray[i].contains("\""))insidePrint=false;
      // compare each array element with each identifier
      // and if it matches assign onCLick events
      if(row==func.content.getDeclaration_line() &&
          lineArray[i].equals(func.content.getName()) &&
          !insidePrint){

         lineArray[i-2]+="id='"+func.content.getRename()
                          +"z-1'";
         lineArray[i]+="<ul>"
            +"<li><a  onClick=\"getNext('"+
            func.content.getRename()+"z-1');\">show " +
            "next</a>"
            +"  <a  onClick=\"getPrev('"+
            func.content.getRename()+"z-1');\">show " +
            "previous</a></li>"
            +"</ul> ";
      }

      // for each array element in this line compare
      // with each occurrence
      // and if it matches assign onCLick events
      int num=0;
      for(Node<Occurrence> o:
        func.content.getOccurrences().traversForward()){
          if(row==o.content.getLine() &&
           lineArray[i].equals(o.content.getName()) &&
           !insidePrint){
```

```
                                    if(o.content.isHtmlreplaced())
                                       num=func.content.getOccurrences().
                                             traversForward().indexOf(o)+1;
                                    else
                                       num=func.content.getOccurrences().
                                             traversForward().indexOf(o);
                                    lineArray[i-2]+=" id='"+
                                                func.content.getRename()+
                                                "z"+num+"'";

                                    lineArray[i]+="<ul>"
                                     +"<li><a  onClick=\"getNext('"+
                                     func.content.getRename()+"z"+num+"')"
                                     +";\">show next</a>"
                                     +"  <a  onClick=\"getPrev('"+
                                     func.content.getRename()+"z"+num+"')"
                                     +";\">show previous</a></li>"
                                     +"</ul> ";

                                    o.content.setHtmlreplaced(true);

                                }
                             }

                          }
                    }

                    //convert array into string
                    String theLine = new String();
                    for(String word:lineArray){
                          theLine+=word;
                    }
                    line=theLine;
                    bw.write("<pre><li>"+row+" </li>"+line+"</pre><br>"+newline);
                    row++;
              }

              bw.write(" "+newline);
              bw.write(" </body>"+newline);
              bw.write("</html> "+newline);
              // end of html

              // set html replace to false for the next file
              //that uses those identifier
              for(Node<Identifier> id:CMT.idTree.traversForward()){
                    for(Node<Occurrence> o:
                          id.content.getOccurrences().traversForward())
                          o.content.setHtmlreplaced(false);
              }

              for(Node<Function> func:CMT.functionTree.traversForward()){
                    for(Node<Occurrence> o:
                          func.content.getOccurrences().traversForward())
                          o.content.setHtmlreplaced(false);
              }

        }

}
```

12. Cp.java

```
/**************************************************************************
 *                                Cp.java                                 *
 **************************************************************************
 *                                                                        *
 * Project:         C Code Maintenance Tool                               *
 *                  B. Sc. Software Engineering                           *
 *                  4th Year Project                                      *
 *                                                                        *
 * Author:          Copyright © Galileo Press 2009                        *
 *                                                                        *
 * Supervisor:      Dr. Christophe Meudec                                 *
 *                                                                        *
 * Summary:                                                               *
 *                                                                        *
 * This class copies the content of a file to another file.               *
 * The code is from an open book by Galileo Computing.                    *
 *                                                                        *
 * Link:                                                                  *
 *  http://openbook.galileocomputing.de/javainsel8/                       *
 *  javainsel_14_003.htm#mjdbd1ea4c51ff569d905c698602549180               *
 *                                                                        *
 **************************************************************************/


import java.io.*;

public class Cp
{
  static void copy( InputStream in, OutputStream out ) throws IOException
  {
    byte[] buffer = new byte[ 0xFFFF ];
    for ( int len; (len = in.read(buffer)) != -1; )
      out.write( buffer, 0, len );
  }

  static void copyFile( String src, String dest )
  {
    FileInputStream  fis = null;
    FileOutputStream fos = null;

    try
    {
      fis = new FileInputStream( src );
      fos = new FileOutputStream( dest );

      copy( fis, fos );
    }
    catch ( IOException e ) {
      e.printStackTrace();
    }
    finally {
      if ( fis != null )
        try { fis.close(); } catch ( IOException e ) { }
      if ( fos != null )
        try { fos.close(); } catch ( IOException e ) { }
    }
  }
```

```
  public static void main( String[] args )
  {
    if ( args.length != 2 )
      System.err.println( "Benutzung: copy <src> <dest>" );
    else
      copyFile( args[0], args[1] );
  }
}
```

13. Splitter.java

```
/*************************************************************************
 *                           Splitter.java                               *
 *************************************************************************
 *                                                                       *
 * Project:         C Code Maintenance Tool                              *
 *                  B. Sc. Software Engineering                          *
 *                  4th Year Project                                     *
 *                                                                       *
 * Supervisor:      Dr. Christophe Meudec                                *
 *                                                                       *
 * Summary:                                                              *
 *                                                                       *
 * This class splits a String into a String array using default          *
 * delimiter(white space)or specific delimiters passed via constructor. *
 *                                                                       *
 * Link:                                                                 *
 *  http://stackoverflow.com/questions/275768/is-there-a-way-to-split    *
 *  -strings-with-string-split-and-include-the-delimiters                *
 *                                                                       *
 *************************************************************************/

import java.util.regex.*;
import java.util.LinkedList;

public class Splitter {
    private static final Pattern DEFAULT_PATTERN = Pattern.compile("\\s+");

    private Pattern pattern;
    private boolean keep_delimiters;

    public Splitter(Pattern pattern, boolean keep_delimiters) {
        this.pattern = pattern;
        this.keep_delimiters = keep_delimiters;
    }
    public Splitter(String pattern, boolean keep_delimiters) {
        this(Pattern.compile(pattern==null?"":pattern), keep_delimiters);
    }
    public Splitter(Pattern pattern) { this(pattern, true); }
    public Splitter(String pattern) { this(pattern, true); }
    public Splitter(boolean keep_delimiters) { this(DEFAULT_PATTERN,
                                                    keep_delimiters); }
    public Splitter() { this(DEFAULT_PATTERN); }
    /**
     *
     * @param text to be split
     * @return string array WITH delimiters
     */
    public String[] split(String text) {
        if (text == null) {
            text = "";
        }

        int last_match = 0;
        LinkedList<String> splitted = new LinkedList<String>();

        Matcher m = this.pattern.matcher(text);
```

```
        while (m.find()) {

            splitted.add(text.substring(last_match,m.start()));

            if (this.keep_delimiters) {
                splitted.add(m.group());
            }

            last_match = m.end();
        }

        splitted.add(text.substring(last_match));

        return splitted.toArray(new String[splitted.size()]);
    }

//    public static void main(String[] argv) {
//        if (argv.length != 2) {
//            System.err.println("Syntax: java Splitter <pattern> <text>");
//            return;
//        }
//
//        Pattern pattern = null;
//        try {
//            pattern = Pattern.compile(argv[0]);
//        }
//        catch (PatternSyntaxException e) {
//            System.err.println(e);
//            return;
//        }
//
//        Splitter splitter = new Splitter(pattern);
//
//        String text = argv[1];
//        int counter = 1;
//        for (String part : splitter.split(text)) {
//            System.out.printf("Part %d: \"%s\"\n", counter++, part);
//        }
//    }
}

/*
    Example:
    > java Splitter "\W+" "Hello World!"
    Part 1: "Hello"
    Part 2: " "
    Part 3: "World"
    Part 4: "!"
    Part 5: ""
*/
```

14. C.g

```
/***************************************************************************
 *                                  C.g                                    *
 ***************************************************************************
 *                                                                         *
 * Project:         C Code Maintenance Tool                                *
 *                  B. Sc. Software Engineering                            *
 *                  4th Year Project                                       *
 *                                                                         *
 * Author:          Terence Parr                                           *
 *                                                                         *
 * Link:            http://www.antlr.org/grammar/1153358328744/C.g         *
 *                                                                         *
 * Adjusted by:     Anna-Christina Friedrich                               *
 *                  C00132716                                              *
 *                  C00132716@itcarlow.ie                                  *
 *                                                                         *
 * Supervisor:      Dr. Christophe Meudec                                  *
 *                                                                         *
 * Summary:                                                                *
 *                                                                         *
 * This is the grammar file for the C language regarding ANSI C            *
 * standard. Original file is written by Terence Parr. In order to         *
 * answer the purposes of CMT some methods were added in the @member       *
 * part of the grammar. Furthermore all rules according to file            *
 * inclusion and nearly each action code section was included for CMT.     *
 * Handling Typedefs has already been in the original grammar file.        *
 *                                                                         *
 ***************************************************************************/
grammar C;
options {
      language=Java;
    backtrack=true;
    memoize=true;
    k=2;

}

scope Symbols {

        Set types; // only track types in order to get parser working
}

@header {

import java.util.Set;
import java.util.HashSet;
import java.util.ArrayList;
import java.io.File;
import com.darwinsys.lang.GetOpt;
import com.darwinsys.util.Debug;

}

@members {
          // decide if scopestack needs to be popped twice,
          // also needed to set the attribute of Scope objects
                  boolean popTwice=false;
```

```
        // inside original grammar file
boolean isTypeName(String name) {
      for (int i = Symbols_stack.size()-1; i>=0; i--) {
              Symbols_scope scope =
                (Symbols_scope)Symbols_stack.get(i);
              if ( scope.types.contains(name) ) {
                      return true;
              }
      }
      return false;
      //return true;
 }

/*
* Inserts a new identifier definition into the tree.
*
* parameter id : token is the name of identifier
*/
void insertId(Token id){
               String d = id.getText();             // name
   int line=id.getLine();               // definition line
           int col=id.getCharPositionInLine();    // column number
           File cFile=new File(input.getSourceName());//sourcefile
           String source= cFile.getName();   // name of sourcefile

   //if it is a function identifier search for the
   //function in the tree, either result is null so that a
   //new function is inserted or a function is found and
   //the definition line is set
   if(input.LT(1).getText().equals("("){
               Node<Function> temp = CMT.functionTree.search(
                  new Function(d,line,col,source,
                                CMT.scopeStack.peek().getNumber()));
               if(temp!=null)
                       temp.content.setDefinition_line(line);
               else CMT.functionTree.insert(new Function(d,line,
                    col,source,CMT.scopeStack.peek().getNumber()));
   }else {// if identifier is variable, insert into tree
      CMT.idTree.insert(new Identifier(d,line,col,source,
                       CMT.scopeStack.peek().getNumber()));
   }
}

        /*
* Inserts a new identifier occurrence into the tree.
*
* parameter id : token is the name of identifier
* returns whether occurrence was inserted or not
*/
boolean insertOcc(Token id){

   boolean found=false;
   String d = id.getText();              // name
   int line=id.getLine();                // definition line
   int col=id.getCharPositionInLine();   // column number
   File cFile=new File(input.getSourceName());// sourcefile
   String source= cFile.getName();       // sourcefile name

   // if occurrence belongs to function,
   // search for definition and insert occurrence
```

```
    if(input.LT(1).getText().equals("("){
       Node<Function> temp = CMT.functionTree.search(
         new Function(d,line,col,source,
                      CMT.scopeStack.peek().getNumber()));
       if(temp!=null)
          temp.content.getOccurrences().insert(
          new Occurrence(d,line,col,source,
          CMT.scopeStack.peek().getNumber(),temp.content.getRename()));
     }
     else{
         // if occurrence belongs to variable we have to
         // search in each currently valid scope, because
         // the 'compareTo' method is implemented to
         // compare name and scope
         Integer currentScope=CMT.scopeStack.peek().getNumber();
         Stack<Scope> helpStack=new Stack<Scope>();

         // search for identifier definition
         // if it is not in the current scope go one valid
         // scope deeper remember all valid scopes by
         // pushing it onto the helpStack
         Node<Identifier> temp = CMT.idTree.search(
           new Identifier(d,line,col,source,CMT.scopeStack.
                                peek().getNumber()));
         if(temp!=null){
           temp.content.getOccurrences().insert(
             new Occurrence(d,line,col,source,currentScope,
                                temp.content.getRename()));
           found=true;
         }else{

           helpStack.push(CMT.scopeStack.pop());
           while(!CMT.scopeStack.empty() &&  !found){

             Node<Identifier> temp2 = CMT.idTree.search(
               new Identifier(d,line,col,source,
                            CMT.scopeStack.peek().getNumber()));
             if(temp2!=null){
               temp2.content.getOccurrences().insert(
                 new Occurrence(d,line,col,source,currentScope,
                                temp2.content.getRename()));
               found=true;
             }
             helpStack.push(CMT.scopeStack.pop());
           }
         }
         // put everything back on the scope stack
         while(!helpStack.empty()){
           CMT.scopeStack.push(helpStack.pop());
         }
     }
      return found;
 }

/*
 * Inserts a new identifier occurrence into the tree
 * without regard to the validity of the scope.
 * This is needed to insert structure names, because you can't
 * predict whether it is a new structure or just an occurrence
 * of an existing structure due to the fact, that you can't
 * always distiguish between definition and occurrence of a
```

```
* structure.
*
* parameter id : token is the name of identifier
*
*/
void insert_searchAllScope(Token id){

     boolean found=false;
     String d = id.getText();              // name
     int line=id.getLine();                // definition line
     int col=id.getCharPositionInLine();   // column number
     File cFile=new File(input.getSourceName());// sourcefile
     String source= cFile.getName();       // sourcefile name

     // first search in the current scope for a variable
     Node<Identifier> temp = CMT.idTree.search(
       new Identifier(d,line,col,source,
                         CMT.scopeStack.peek().getNumber()));
     if(temp!=null){
         temp.content.getOccurrences().insert(
           new Occurrence(d,line,col,source,CMT.scopeStack.
                 peek().getNumber(),temp.content.getRename()));

         found=true;
     }
     else{// if it is not inside the current scope
          // search backwards in each scope
         int currentScope=CMT.scopeStack.peek().getNumber();
         while(currentScope!=-1 && !found){

             Node<Identifier> temp2 = CMT.idTree.search(
               new Identifier(d,line,col,source,
                         CMT.scopeStack.peek().getNumber()));
             if(temp2!=null){
               temp2.content.getOccurrences().insert(
                 new Occurrence(d,line,col,source,currentScope,
                                temp2.content.getRename()));
               found=true;
             }
             currentScope--;
         }
     }
     // if no definition is found, insert as definition
     if(!found) insertId(id);
         }

         /*
      * Searchs for a certain file inside a given directory.
      *
      * parameter dir : directory in which search takes place
      * parameter find : name of the file that is searched for
      * returns the searched file
      */
         File searchFile(File dir, String find) {

     File[] files = dir.listFiles();
     File matches = null;
     if (files != null) {
          for (int i = 0; i < files.length; i++) {
             if (files[i].getName().equalsIgnoreCase(find)) {
               matches=files[i];
```

```
                    }
                    if (matches == null && files[i].isDirectory()) {
                      matches=searchFile(files[i],find);
                    }
                  }
              }
              return matches;
          }

}


translation_unit
scope Symbols; // entire file is a scope
@init {
  $Symbols::types = new HashSet();

}
        : preprocess* external_declaration+
        | external_declaration+
        ;

/** Either a function definition or any other kind of C decl/def.
 *  The LL(*) analysis algorithm fails to deal with this due to
 *  recursion in the declarator rules.  I'm putting in a
 *  manual predicate here so that we don't backtrack over
 *  the entire function.  Further, you get a better error
 *  as errors within the function itself don't make it fail
 *  to predict that it's a function.  Weird errors previously.
 *  Remember: the goal is to avoid backtrack like the plague
 *  because it makes debugging, actions, and errors harder.
 *
 *  Note that k=1 results in a much smaller predictor for the
 *  fixed lookahead; k=2 made a few extra thousand lines. ;)
 *  I'll have to optimize that in the future.
 */


preprocess
    : '#include' fileInclusion
    ;

fileInclusion
@init
{
  File  sourceFile = null;
  File  currentDirectory=null;
  String  includeFile = null;
}
    : f=include_file
        { //get the currently parsed file
          sourceFile = new File(String.valueOf(input.getSourceName()));
          //get the directory of the source file
          currentDirectory = sourceFile.getAbsoluteFile();
          includeFile=$f.text;

          if(includeFile.startsWith("<"))
          {//do nothing with system libraries

          }else{
              //if it is a self written library indicated by quotes
```

```
            includeFile = includeFile.substring(1,includeFile.length()-1);
            if(!includeFile.startsWith("/")){
               File currentdir=currentDirectory;
                  File incFile=null;
                  //search through directories backwards for include file
                  do{
                  currentdir = currentdir.getParentFile();
                      incFile =searchFile(currentdir, includeFile);
                  }while(incFile==null);

                // if the file hasn't already been referenced
                // execute according to tags
                if (incFile!=null && !CMT.filenames.contains(incFile)){
                  CMT.filenames.add(incFile);
                  try{
                    ComLineInterpreter.fillTree(incFile.getPath());
                    if(ComLineInterpreter.textTag)
                      ComLineInterpreter.createText(incFile.getPath());
                    if(ComLineInterpreter.renameAllTag)
                      ComLineInterpreter.renameAll(incFile.getPath());
                    if(ComLineInterpreter.htmlTag)
                      ComLineInterpreter.writeHtml(incFile.getPath());
                    if(ComLineInterpreter.renameTag)
                      ComLineInterpreter.rename(incFile.getPath());

                    //search for the definition C file of the header file
                    //and adjust execution as well
                    //there is not always a defining C file for a header file,
                    //therefore we just go back 3 directories so that it is
                    //not taking too much time hopefully
                    String newfile=includeFile.replace(
                                                  CMT.hSuffix,CMT.cSuffix);
                    File curdir=currentDirectory;
                    File newf = null;
                    int folderNum=0;
                    do{
                      curdir = curdir.getParentFile();
                      newf =searchFile(curdir, newfile);
                      folderNum++;
                    }while(newf==null && folderNum<2);

                    if (newf!=null && !CMT.filenames.contains(newf)){
                        CMT.filenames.add(newf);
                        ComLineInterpreter.fillTree(newf.getPath());
                        if(ComLineInterpreter.textTag)
                          ComLineInterpreter.createText(newf.getPath());
                        if(ComLineInterpreter.renameAllTag)
                          ComLineInterpreter.renameAll(newf.getPath());
                        if(ComLineInterpreter.htmlTag)
                          ComLineInterpreter.writeHtml(newf.getPath());
                        if(ComLineInterpreter.renameTag)
                          ComLineInterpreter.rename(newf.getPath());
                    }
                  }
                  catch(Exception e){;}
                }
            }
          }
        }
;
```

```
include_file
    : '<' IDENTIFIER '.h' '>'
      |   STRING_LITERAL
      ;


external_declaration
options {k=1;}
        : ( declaration_specifiers? declarator declaration* '{' )=>
            function_definition
        | declaration
        ;

function_definition
scope Symbols; // put parameters and locals into same scope for now
@init {
  $Symbols::types = new HashSet();
}
        :       declaration_specifiers? declarator
                (       declaration+ compound_statement // K&R style
                |       compound_statement             // ANSI style
                )
        ;

declaration
scope {
  boolean isTypedef;
}
@init {
  $declaration::isTypedef = false;
}
        : 'typedef' declaration_specifiers? {$declaration::isTypedef=true;}
          init_declarator_list? ';' // special case, looking for typedef
        | declaration_specifiers init_declarator_list? ';'
        ;

declaration_specifiers
        :   (   storage_class_specifier
                |   type_specifier
        |   type_qualifier
        )+
        ;

init_declarator_list
        : init_declarator (',' init_declarator)*
        ;

init_declarator
        : declarator ('=' initializer)?
        ;

storage_class_specifier
        : 'extern'
        | 'static'
        | 'auto'
        | 'register'
        ;

type_specifier
        :( 'void'
        | 'char'
```

```
        | 'short'
        | 'int'
        | 'long'
        | 'float'
        | 'double'
        | 'signed'
        | 'unsigned'
        | struct_or_union_specifier
        | enum_specifier
        | type_id)
        ;

type_id
    :   {isTypeName(input.LT(1).getText())}?

        id=IDENTIFIER

        {insert_searchAllScope($id);

        }
    ;

struct_or_union_specifier
options {k=3;}
scope Symbols; // structs are scopes
@init {
  $Symbols::types = new HashSet();
}       //define a scope for body of structure
        : struct_or_union id=IDENTIFIER?

        {CMT.scope++;CMT.scopeStack.push(new Scope(CMT.scope,false));}

          '{' struct_declaration_list '}'

        {insert_searchAllScope($id);CMT.scopeStack.pop();}

        | struct_or_union id=IDENTIFIER

        {insert_searchAllScope($id);}
        ;

struct_or_union
        : 'struct'
        | 'union'
        ;

struct_declaration_list
        : struct_declaration+
        ;

struct_declaration
        : specifier_qualifier_list struct_declarator_list ';'
        ;

specifier_qualifier_list
        : ( type_qualifier | type_specifier )+
        ;

struct_declarator_list
        : struct_declarator (',' struct_declarator)*
        ;
```

```
struct_declarator
        : declarator (':' constant_expression)?
        | ':' constant_expression
        ;

enum_specifier
options {k=3;}
        : 'enum' '{' enumerator_list '}'
        | 'enum' id=IDENTIFIER '{' enumerator_list '}'{insertId($id);}
        | 'enum' id=IDENTIFIER {insertOcc($id);}
        ;

enumerator_list
        : enumerator (',' enumerator)*
        ;

//for now no storing of enum list
enumerator
        : IDENTIFIER ('=' constant_expression)?
        ;

type_qualifier
        : 'const'
        | 'volatile'
        ;

declarator
        : pointer? direct_declarator
        | pointer
        ;

direct_declarator
        :(   id=IDENTIFIER
            { if ($declaration.size()>0&&$declaration::isTypedef) {
                      $Symbols::types.add($IDENTIFIER.text);
              }
              insertId($id);
            }
         | '(' declarator ')'
         )
         declarator_suffix*
        ;

declarator_suffix
        :   '[' constant_expression ']'
    |   '[' ']'
    |   '('
    //create new scope for parameterlist of function
    {popTwice=true;CMT.scope++;
     CMT.scopeStack.push(new Scope(CMT.scope,false));}

     parameter_type_list ')'

    //if prototype, pop the scope, otherwise the parameter are valid
    //in function body
    {if(input.LT(1).getText().equals(";")){
            CMT.scopeStack.pop();
    }}
    |   '(' identifier_list ')'
    |   '(' ')'
```

```
        ;

pointer
        : '*' type_qualifier+ pointer?
        | '*' pointer
        | '*'
        ;

parameter_type_list
        : parameter_list (',' '...')?
        ;

parameter_list
        : parameter_declaration (',' parameter_declaration)*
        ;

parameter_declaration
        : declaration_specifiers (declarator|abstract_declarator)*
        ;

identifier_list
        : IDENTIFIER (',' IDENTIFIER)*
        ;

type_name
        : specifier_qualifier_list abstract_declarator?
        ;

abstract_declarator
        : pointer direct_abstract_declarator?
        | direct_abstract_declarator
        ;

direct_abstract_declarator
        :       ( '(' abstract_declarator ')' | abstract_declarator_suffix )
abstract_declarator_suffix*
        ;

abstract_declarator_suffix
        :       '[' ']'
        |       '[' constant_expression ']'
        |       '(' ')'
        |       '(' parameter_type_list ')'
        ;

initializer
        : assignment_expression
        | '{' initializer_list ','? '}'
        ;

initializer_list
        : initializer (',' initializer)*
        ;

// E x p r e s s i o n s

argument_expression_list
        :   assignment_expression (',' assignment_expression)*
        ;

additive_expression
```

```
        : (multiplicative_expression)
          ('+' multiplicative_expression | '-' multiplicative_expression)*
        ;

multiplicative_expression
        : (cast_expression)
          ('*' cast_expression | '/' cast_expression | '%' cast_expression)*
        ;

cast_expression
        : '(' type_name ')' cast_expression
        | unary_expression
        ;

unary_expression
        : postfix_expression
        | '++' unary_expression
        | '--' unary_expression
        | unary_operator cast_expression
        | 'sizeof' unary_expression
        | 'sizeof' '(' type_name ')'
        ;

postfix_expression
        :   primary_expression
        (   '[' expression ']'
        |   '(' ')'
        |   '(' argument_expression_list ')'

        //this is where to start solving the structure problem
        |   '.' id=IDENTIFIER  {insertOcc($id);}
        |   '->' id=IDENTIFIER {insertOcc($id);}
        |   '++'
        |   '--'
        )*
        ;

unary_operator
        : '&'
        | '*'
        | '+'
        | '-'
        | '~'
        | '!'
        ;

primary_expression
        : id=IDENTIFIER      {insertOcc($id);}
        | constant
        | '(' expression ')'
        ;

constant
    :   HEX_LITERAL
    |   OCTAL_LITERAL
    |   DECIMAL_LITERAL
    |   CHARACTER_LITERAL
        |       STRING_LITERAL
    |   FLOATING_POINT_LITERAL
    ;
```

```
/////

expression
        : assignment_expression (',' assignment_expression)*
        ;

constant_expression
        : conditional_expression
        ;

assignment_expression
        : conditional_expression
        | lvalue assignment_operator assignment_expression
        ;

lvalue
        : unary_expression
        ;

assignment_operator
        : '='
        | '*='
        | '/='
        | '%='
        | '+='
        | '-='
        | '<<='
        | '>>='
        | '&='
        | '^='
        | '|='
        ;

conditional_expression
        : logical_or_expression ('?' expression ':' conditional_expression)?
        ;

logical_or_expression
        : logical_and_expression ('||' logical_and_expression)*
        ;

logical_and_expression
        : inclusive_or_expression ('&&' inclusive_or_expression)*
        ;

inclusive_or_expression
        : exclusive_or_expression ('|' exclusive_or_expression)*
        ;

exclusive_or_expression
        : and_expression ('^' and_expression)*
        ;

and_expression
        : equality_expression ('&' equality_expression)*
        ;
equality_expression
        : relational_expression (('=='|'!=') relational_expression)*
        ;
```

```
relational_expression
        : shift_expression (('<'|'>'|'<='|'>=') shift_expression)*
        ;

shift_expression
        : additive_expression (('<<'|'>>') additive_expression)*
        ;

// S t a t e m e n t s

statement
        : labeled_statement
        | compound_statement
        | expression_statement
        | selection_statement
        | iteration_statement
        | jump_statement
        ;

labeled_statement
        //have to use this method, because you can't predict if it
        //is a definition or an occurrence of the identifier
        : id=IDENTIFIER ':' statement {insert_searchAllScope($id);}
        | 'case' constant_expression ':' statement
        | 'default' ':' statement
        ;

compound_statement
scope Symbols; // blocks have a scope of symbols
@init {
  $Symbols::types = new HashSet();

}        //create new scope,if it is a function body
         //set boolean attribute to true
        : { CMT.scope++;
            if(popTwice){
                  CMT.scopeStack.push(new Scope(CMT.scope,true));
                  popTwice=false;
            }
            else CMT.scopeStack.push(new Scope(CMT.scope,false));
          }

            '{' declaration* statement_list? '}'

            //pop scope from stack, if it is end of function body
            //pop twice
          { if(CMT.scopeStack.peek().isPopTwice()){
                  CMT.scopeStack.pop();
                  CMT.scopeStack.pop();
            }else CMT.scopeStack.pop();
          }
        ;

statement_list
        : statement+
        ;

expression_statement
        : ';'
        | expression ';'
        ;
```

```
selection_statement
        : 'if' '(' expression ')' statement
            (options {k=1; backtrack=false;}:'else' statement)?
        | 'switch' '(' expression ')' statement
        ;

iteration_statement
        : 'while' '(' expression ')' statement
        | 'do' statement 'while' '(' expression ')' ';'
        | 'for' '(' expression_statement expression_statement expression? ')'
          statement
        ;

jump_statement
        //have to use this method, because you can't predict if it
        //is a definition or an occurrence of the identifier
        : 'goto' id=IDENTIFIER ';'{insert_searchAllScope($id);}
        | 'continue' ';'
        | 'break' ';'
        | 'return' ';'
        | 'return' expression ';'
        ;


IDENTIFIER
        :     LETTER(LETTER|'0'..'9')*
           ;

fragment
LETTER
        :       '$'
        |       'A'..'Z'
        |       'a'..'z'
        |       '_'
        ;

CHARACTER_LITERAL
    :   '\'' ( EscapeSequence | ~('\''|'\\') ) '\''
    ;

STRING_LITERAL
    :  '"' ( EscapeSequence | ~('\\'|'"') )* '"'
    ;

HEX_LITERAL : '0' ('x'|'X') HexDigit+ IntegerTypeSuffix? ;

DECIMAL_LITERAL : ('0' | '1'..'9' '0'..'9'*) IntegerTypeSuffix? ;

OCTAL_LITERAL : '0' ('0'..'7')+ IntegerTypeSuffix? ;

fragment
HexDigit : ('0'..'9'|'a'..'f'|'A'..'F') ;

fragment
IntegerTypeSuffix
        :       ('u'|'U')? ('l'|'L')
        |       ('u'|'U')  ('l'|'L')?
        ;

FLOATING_POINT_LITERAL
```

```
    :   ('0'..'9')+ '.' ('0'..'9')* Exponent? FloatTypeSuffix?
    |   '.' ('0'..'9')+ Exponent? FloatTypeSuffix?
    |   ('0'..'9')+ Exponent FloatTypeSuffix?
    |   ('0'..'9')+ Exponent? FloatTypeSuffix
        ;

fragment
Exponent : ('e'|'E') ('+'|'-')? ('0'..'9')+ ;

fragment
FloatTypeSuffix : ('f'|'F'|'d'|'D') ;

fragment
EscapeSequence
    :   '\\' ('b'|'t'|'n'|'f'|'r'|'\"'|'\''|'\\')
    |   OctalEscape
    ;

fragment
OctalEscape
    :   '\\' ('0'..'3') ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7') ('0'..'7')
    |   '\\' ('0'..'7')
    ;

fragment
UnicodeEscape
    :   '\\' 'u' HexDigit HexDigit HexDigit HexDigit
    ;

WS  :  (' '|'\r'|'\t'|'\u000C'|'\n') {$channel=HIDDEN;}
    ;

COMMENT
    :   '/*' ( options {greedy=false;} : . )* '*/' {$channel=HIDDEN;}
    ;

LINE_COMMENT
    : '//' ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    ;

 //have to list each pre processing command, because #include is excluded
LINE_COMMAND
    :
      '#'
      ('define'|'undef'|'if'|'else'|'endif'|'line'|'elif'|'pragma'|'error'|'#')
      ~('\n'|'\r')* '\r'? '\n' {$channel=HIDDEN;}
    ;
```