SL Language Manual

CS W4115: Programming Languages and Translators

Professor Stephen A. Edwards

Computer Science Department

Submitted by

Majid Khan {UID = mk2759, Email = majidkhan@yahoo.com }

1. Introduction

Search engines are still emerging markets and lack tools which enable building search applications e.g. Search engines, price matching engines, Meta crawlers etc. faster and easier. This search language would provide user capability to create text search applications with lesser efforts. The scope of this project would be limited to text search in HTML and plain text files.

A typical search application should provide

1) Ability to define repositories and maintain repositories
2) Ability to understand document formats and manipulate data inside repositories
3) Ability to gather statistical data based on repository data and then store it for later use.
4) Ability to provide operations on statistical data to perform and produce results

1.1 Glossary

Repository – A repository would be a hierarchy of directories containing documents.

Document – HTML and plain texts would be the file types used as documents. Only files with extensions html, htm, txt and without extensions (would be considered as plain text) would be processed by this language.

Data manipulation – It is a multi step process which is as following

Parse data and get words.

Perform stemming on these words (http://en.wikipedia.org/wiki/Stemming)

Get frequencies of stemmed words and create an inverted vector table using this data.

Calculate term frequency–inverse document frequency (tf-idf) for every stemmed word.

Store this data in for future use.

Search – Search would need two items. Query string and a repository on which we have already performed data manipulation. Query is a series of alphanumeric words separated by white space.

This process would return a list of documents and also a number that represents the closeness of query with document.

2. Lexical conventions

There are six kinds of tokens: identifiers, keywords, constants, strings, expression operators, and separators. White space would be ignored (would not be converted to any tag) but it may be serving as tag separators. A token is the longest consuctive string not separated by white space and other declared seperators e.g. Comma, semiColon, braces, new line, space, tab etc.

2.1 Comments

C style multiline comments are used i.e. Anything between /* and */ is comment. There is no hierarchy of comments i.e. /* /* */ is one comment while /* /* */ */ would be an error because outermost */ is extra.

2.2 Identifiers

An identifier is a sequence of letters and digits; the first character must be alphabetic. Identifiers are case insensitive. Although it could be controlled but identifiers could be of any length. All part of identifier is significant.

2.3 Keywords

integer
null
if
else
for
to
each
string
name
value
document
documentlist
operation
repository
return
returns

2.4 Types and Literals

There are several kinds of constants, as follows:

2.4.1 Integer

A float consists of an integer part, a decimal point, and a fraction part. The integer and fraction parts both consist of a sequence of digits. Either the integer part or the fraction part (not both) may be missing.

2.4.2 String

A string is a sequence of characters surrounded by double quotes `“`. A string has the type arrayofcharacters (see below) and refers to an area of storage initialized with the given characters. In a string, the character `"` (double quote) must be preceded by a `”` (double quote). Character constant is missing and this could be fulfilled by a string of size 1.

2.4.3 Repository

Repository is the base datatype that holds information about data source. It could be defined in two ways, creating from scratch or from some operation on existing repositories.

Repository data type needs three data items to be created "location", "inlcude" and "type". String "location" contains a value that points to a directory structure (compiler would not validate value string contains a valid directory). String value "include" contains a string that contains comma separated wildcard file patterns (default value if not provided is "*.html,*.txt,*" which means all files that has extension .html, .txt and without extensions in all subdirectories would be considered as doucments). String value "type" contains the method which would be use to produce statistical data for search. First parameter itself is the repository which we need to initialize.

Example initialization:
Arep = makerepository(
”c:\adirectory\anotherDirectory”,”*.html,*.txt,*.htm,*”,”cosine”);

2.4.4 Query

Query is the base data type that holds search string, repository reference and where the search result would be stored. It has two String variables "query" and "store". String; value "query" is a string that we are searching in repository and "store" is the file name where query results would be stored (if it is already present then file would be overwritten).

Example initialization:

docList = performquery (arep,”search this data please”,”search.query”)

2.4.5 Document and DocumentList

Document type is not in the scope of current project (I mentioned earlier that if time would permit I will add this). Query result can be returned as a collection of Document which is a builtin type called Document list. A for each loop can extract documents from documentList type.

Result of the query would be sorted in descending order with respect to matching criteria returned by statisitcal measure used. (This is not in the scope).

Both of the higlighted items that I thought might not be implemented are implemented.

2.4.6 Block

'{' starts a new code block and matching '}' ends this code block. A block can contain zero or more expressions.

2.4.7 Separator

';' is expression separator. A semicolon without any expression is null statement. Multiple null statements can exist in a sequence. '(' and ')' could be used for expression grouping.

2.4.8 Expression

Following sections are in the order of the precedence of operations.

2.4.8.1 Primary expressions

2.4.8.1.1 Identifier

Identified is a primary expression.

2.4.8.1.2 Constants

Cosntants are primary expressions.

2.4.8.1.3 Function call

A function call is followed by an optional list of parameters enclosed in parentheses separated by commas. It may contain an optional return value at the end. For e.g. performquery which was invoked as following

```
docList = performquery (arep, ”search this data please”,”search.query”);
```

Function documentQuery would be defined as

```
operation performquery (query q, reposiotry arep,
string querystring,
string store) returns documentList
{
/* body */
};
```

2.4.8.1.4 Binary Operators

+, - , * , / are the binary operators supported. * and / have same precedency and its higher than the precedence of + and -.

+ opeartor is defined on strings and results in concatenation of two strings

+ operator is defined on repository and results in addition of two repositories to one.

2.4.8.1.5 Relational operators

All relational operators have same precedence

> (expression > expression) greater than
< (expression < expression) less than
>= (expression >= expression) greater than or equal to
<= (expression <= expression) less than or equal to
<> (expression <> expression) not equal to
! (!(expression)) not of the result of expression
!= (expression != expression) not equal to

2.4.8.1.6 Declaration

Declarations are done at the start of the program. Each declaration is terminated by semicolon.

```
DECLARATIONLIST        :    DECLARATION *;

DECLARATION      :        TYPE IDENTIFIER SEMICOLON   ;
```

2.4.8.1.7 Assignment

An assignment has an identifier that is lvalue and it has a `=` then a statement. This statement is evaluated and the result is then stored in lvalue.

```
BINARYSTATEMENT :      IDENTIFIER
                       | BINARYSTATEMENT BINARYOPERATOR
                       (IDENTIFIER | FUNCTIONCALL);


ASSIGNMENTSTATEMENT         :         IDENTIFIER '=' ( IDENTIFIER |
FUNCTIONCALL ) (BINARYOPERATOR BINARYSTATEMENT)*;
```

2.4.8.1.8 If – Else

If statement is a very simple statement which contains a boolean expression that either returns zero or not zero (zero means false and non zero means true). At runtime the boolean statement would be evaluated and if result equals to zero then else would be executed otherwise the first set of statements would be executed.

```
IFSTATEMENT        :        'IF' ‘(‘ BOOLEANSTATEMENT ‘)’
                                    ( STATEMENT | '{' STATEMENTS '}')
                            ('ELSE' ( STATEMENT | '{' STATEMENTS '}') )?
```

2.4.8.1.9 For and For each

For has two flavors one that is to execute loop for a range of numbers. For loop variables values can be modified inside the loop (start, end and index variables). Another variation is to loop through the collection of documents (for each).

```
FORSTATEMENT     :        'FOR'  (IDENTIFIER '=' NUMBER 'TO' NUMBER
                          | 'EACH' IDENTIFIER 'IN' IDENTIFIER)
                          ( STATEMENT | '{' STATEMENTS '}');
```

3. Grammar/Lexer/Parser/Tree Walker

```
class SLLexer extends Lexer;

options {
k=2;
charVocabulary='\3' ..'\377';
caseSensitive = false;
caseSensitiveLiterals = false;
testLiterals=false;
}


tokens {

  PROGRAM;
  DECLARATION;
  ASSIGNMENTSTATEMENT;
  THENSTATEMENT;
  ELSESTATEMENT;
  FORSTATEMENT;
  TYPELIST;
  RETURNTYPE;
  SIGNEDFACTOR;
  FUNCTIONBODY;
  CODEBLOCK;
  INTEGER =  "integer" ;
  STRING = "string";
  REPOSITORY = "repository" ;
  DOCUMENTLIST = "documentlist" ;
  DOCUMENT = "document";
  QUERY = "query";
  DO              = "do"             ;
  ELSE            = "else"           ;
  EACH                  = "each"                        ;
  END             = "end"            ;
  FOR             = "for"            ;
  IF              = "if"             ;
  IN              = "in"             ;
  NULL            = "null"           ;
  OPERATION              ;
  TO              = "to"             ;
  TYPE            = "type"           ;
  UNTIL           = "until"          ;
  VAR             = "var"            ;
  MAIN            = "main"                ;
  RETURNS         = "returns"                     ;
  RETURN          = "return";
}
```

```
IDENTIFIER options {testLiterals=true;} :
('a' .. 'z') ( ('0'..'9') | ( 'a' .. 'z') )*;


protected ALPHABET : ('a' .. 'z');

protected NUMBER          : ('0'..'9')+;
 LEFTSMALLPARAN : '(';
 RIGHTSMALLPARAN : ')';

 LEFTPARAN : '{';
 RIGHTPARAN : '}';

 COMMA           :             ',';
  PLUS : '+' ;
  MINUS : '-' ;
   DIVIDE : '/' ;
  MULTIPLY: '*';


 EQUAL : '=';
 GT  : '>' ;
 LT  : '<' ;
 GTE : '>' '=' ;
 NE  : '!' '=' ;
 LTE : '<' '=' ;
 OR       : '|';
 AND     : '&';
 NOT : '!' ;
 SEMICOLON :';';

STRINGLITERAL
  : '"'!
    ( '"' '"'!
    | ~('"')
    )*
    ( '"'!
    |
    )
   ;


INTEGERLITERAL  :     NUMBER ('.' (NUMBER)? )?  | ('.') NUMBER    ;
```

```
/* comment is either copied from class or web or course material*/
Comment: '/'
( ('*') => '*'
(
options {greedy=false;}:
(
('\r' '\n') => '\r' '\n' { newline(); }
| '\r' { newline(); }
| '\n' { newline(); }
| ~( '\n' | '\r' )
)
)*
"*/"
| ((~'\n'))* '\n' { newline(); }
)
{ $setType(Token.SKIP); }
          ;


WS    : ( ' '
        | '\r' ('\n')?{ newline(); }
        | '\n'{ newline(); }
        | '\t'
        )
        {$setType(Token.SKIP);}
   ;
```

```
class SLParser extends Parser;
options {
k=2;
buildAST = true;
}

{

        java.util.HashSet identifiers = new java.util.HashSet();
        java.util.HashSet functions = new java.util.HashSet();
        java.util.HashSet procedures = new java.util.HashSet();

        boolean isProcedure=true;
        private void loadBuiltinOperations()
        {

                functions.add("getdocumenturi");
                functions.add("getdocumentdata");
                functions.add("makerepository");
                functions.add("performquery");
                procedures.add("print");
                procedures.add("processrepository");

        }



}

type:   INTEGER | STRING | REPOSITORY |     DOCUMENTLIST |       DOCUMENT |
        QUERY;

declaration :       VAR! type IDENTIFIER SEMICOLON!
                    {
        if (identifiers.contains(#IDENTIFIER.getText().toLowerCase()))
                throw new exceptions.VariableAlreadyDeclaredException(#IDENTIFIER.getText() + "
        is already declared");
        else
                identifiers.add(#IDENTIFIER.getText().toLowerCase());

        }
        {#declaration = #([DECLARATION,"DECLARATION"], declaration);}          ;

ifstatement :  IF^ LEFTSMALLPARAN! booleanexpression RIGHTSMALLPARAN! thenStatement
(options {greedy = true;}:elseStatement)?;


thenStatement: ( codeblock)
        {#thenStatement = #([THENSTATEMENT,"THENSTATEMENT"], thenStatement);}    ;
elseStatement: ELSE! ( codeblock)
        {#elseStatement = #([ELSESTATEMENT,"ELSESTATEMENT"], elseStatement);};


forstatement :
        (("for"! IDENTIFIER EQUAL! expression TO! expression (codeblock) )
| ("for"! "each"! IDENTIFIER IN! IDENTIFIER (codeblock) ))
{#forstatement = #([FORSTATEMENT,"FORSTATEMENT"], forstatement);};


functioncall  : functionName:IDENTIFIER LEFTSMALLPARAN! (expression ( COMMA! expression)*)?
RIGHTSMALLPARAN!
            {  if (!functions.contains(#functionName.getText().toLowerCase())  &&
!procedures.contains(#functionName.getText().toLowerCase()))
                        throw new RuntimeException("Function " + #functionName.getText() + " is not declared
");

            #functioncall = #([FUNCTIONCALL,"FUNCTIONCALL"], functioncall);}
;

booleanexpression : binaryterm ( (EQUAL^ | GT^  |  LT^  | GTE^ | NE^  | LTE^ ) binaryterm)*;

binaryterm : booleanterm (OR^ booleanterm)* ;

booleanterm : booleanfactor ( AND^  booleanfactor)*;

booleanfactor : (NOT^)? expression  ;

expression :  term ( (PLUS^ | MINUS^) term )? ;

term :  signedfactor ((MULTIPLY^ | DIVIDE^) signedfactor)? ;

signedfactor :  (MINUS^)? factor ;

factor :  IDENTIFIER
| functioncall
            { if (!functions.contains(returnAST.getFirstChild().getText().toLowerCase()))
                        throw new RuntimeException("Function " + returnAST.getFirstChild().getText() + "
does not return any value. Cannot be used in expressions ");
            }

| INTEGERLITERAL | STRINGLITERAL | NULL
| LEFTSMALLPARAN! expression RIGHTSMALLPARAN! ;

assignmentstatement :  IDENTIFIER  EQUAL! expression
{#assignmentstatement = #([ASSIGNMENTSTATEMENT,"ASSIGNMENTSTATEMENT"],
assignmentstatement);};

codeblock:   (LEFTPARAN!        (statement)*  RIGHTPARAN! | statement)
{#codeblock = #([CODEBLOCK,"CODEBLOCK"], codeblock);};


program :{loadBuiltinOperations(); }            (declaration)* (functiondeclaration)* MAIN! LEFTPARAN!
(statement)*  RIGHTPARAN!  EOF!

                        {#program = #([PROGRAM,"PROGRAM"],program); };

returnstatement :
                        "return"! expression
                        {#returnstatement = #([RETURN,"RETURN"], returnstatement);};
```

```
statement :

            (
            ifstatement |
            forstatement |
            returnstatement SEMICOLON! |
            assignmentstatement SEMICOLON! |
            functioncall SEMICOLON!
            )
            ;



functiondeclaration : {isProcedure=false; } "operation"! functionName:IDENTIFIER
LEFTSMALLPARAN! (typelist) RIGHTSMALLPARAN! (returntype )
functionBody    {#functiondeclaration = #([OPERATION,"OPERATION"], functiondeclaration);}
{
            if (functions.contains(#functionName.getText().toLowerCase())  ||
procedures.contains(#functionName.getText().toLowerCase()))
                        throw new RuntimeException("Function " + #functionName.getText() + " is already
declared ");
            if (isProcedure) procedures.add(#functionName.getText().toLowerCase());
            else functions.add(#functionName.getText().toLowerCase());
} ;

returntype: (RETURNS! type  ) {#returntype = #([RETURNTYPE,"RETURNTYPE"], returntype);}
| { isProcedure=false; } {#returntype = #([RETURNTYPE,"RETURNTYPE"], returntype);};

typelist :
    (type IDENTIFIER (COMMA! type IDENTIFIER )*   )
{#typelist = #([TYPELIST,"TYPELIST"], typelist);}
| // nothing
{#typelist = #([TYPELIST,"TYPELIST"], typelist);}
;

functionBody :LEFTPARAN! (statement)* RIGHTPARAN!
  {#functionBody = #([FUNCTIONBODY,"FUNCTIONBODY"], functionBody);}
;
```

```
{
import java.util.HashMap;
}
/*******************************************************
*                              Tree Walker                              *
*******************************************************/
class SLWalker extends TreeParser;
{
HashMap functionMap = new HashMap();
HashMap variables = new HashMap();
}


program

{   SLDataTypeInfo slDataTypeInfo;
    // first load methods so that later calls could be executed.
    SLUtils.loadBuiltinMethods(functionMap);
            AST t = _t.getFirstChild();
            boolean bFuncsFound = false;
            while (t != null) {
                        if (t.getText().toLowerCase().equals("operation")) {
                            _t = t;

                            functiondeclaration(t.getFirstChild());
                        }
                        t = t.getNextSibling();
            }

}

  : #(PROGRAM (slDataTypeInfo=evaluateTree)*
    )
  ;

type returns [SLTypes t]
{ t = null; }
  : ( "repository"  { t = SLTypes.REPOSITORY;  }
    | "string"  { t = SLTypes.STRING;  }
    | "documentlist"   { t = SLTypes.DOCUMENTLIST;   }
    | "query"   { t = SLTypes.QUERY;   }
    | "integer"   { t = SLTypes.INTEGER;   }
    | "document" { t = SLTypes.DOCUMENT; } )
  ;


functiondeclaration
{
java.util.Vector v = new java.util.Vector();
SLTypes returnType= null;
AST current = _t;
}
:#(fname:IDENTIFIER

            {
               AST tArgs = current.getNextSibling();
            v=typelist(tArgs);
            tArgs = tArgs.getNextSibling();
            AST typeAST = tArgs.getFirstChild();
            if (typeAST!=null)
               returnType=type(typeAST);
            else returnType = null;
            tArgs = tArgs.getNextSibling().getFirstChild();

                SLFunction aFunction = new SLFunction(fname.getText(), returnType, v, tArgs);
                        functionMap.put(fname.getText(),aFunction);
             }

)
;



typelist returns [java.util.Vector v]
{   v = new java.util.Vector();
    SLTypes thistype = null;

}
: #(TYPELIST
    (thistype=type name:IDENTIFIER {v.add(new
SLDataTypeInfo(thistype,#name.getText().toLowerCase())) ;}
     )*
    )

;

argument [java.util.Vector v]
{SLTypes thisType = null; SLDataTypeInfo slDataTypeInfo = null;}
:(thisType=type IDENTIFIER
{ slDataTypeInfo = new SLDataTypeInfo(thisType,#IDENTIFIER.getText() );
  v.add(slDataTypeInfo);
}
);

evaluateTree returns [SLDataTypeInfo slDataTypeInfo]
{
SLDataTypeInfo a = null;
SLDataTypeInfo b = null;
SLDataTypeInfo c = null;
SLDataTypeInfo d = null;
slDataTypeInfo = null;
}
 : #(DECLARATION { SLTypes slType = null; }
(slType=type IDENTIFIER
{ slDataTypeInfo = new SLDataTypeInfo(slType,#IDENTIFIER.getText() );
  variables.put(#IDENTIFIER.getText(),slDataTypeInfo );
}

             )
   )
  | #(ASSIGNMENTSTATEMENT  a=evaluateTree b=evaluateTree)
                                    { slDataTypeInfo =  a.assign(b); }
```

```
| #(CODEBLOCK (a=evaluateTree)*)


| #(FORSTATEMENT {int numberofChildren = #FORSTATEMENT.getNumberOfChildren();
   }
  //a = evaluateTree b=evaluateTree codeOrStopCondition:. forLoopCode:.
   {
     a=evaluateTree(_t);
     _t = _retTree;
     b=evaluateTree(_t);
     _t = _retTree;
     AST codeOrStopCondition = (AST)_t;
     _t = _t.getNextSibling();
     AST forLoopCode = (AST)_t;

     if (numberofChildren>3)
     {
        a.assign(b);
        c = evaluateTree(codeOrStopCondition);
        while(a.compareTo(c)<=0)
        {

           d = evaluateTree(forLoopCode);
           c = evaluateTree(codeOrStopCondition);
           a.increment();
        }
     }
     else
     {

       SLDataTypeInfo currentDocumentHolder = a;
       SLDocument currentDocument = new SLDocument();
       currentDocumentHolder.setType(SLTypes.DOCUMENT);
       SLDocumentList loopDocumentList = (SLDocumentList) b.getValue();
       java.util.Iterator loopDocumentListIterator = loopDocumentList.documents.values().iterator();
       while (loopDocumentListIterator .hasNext())

        {
           SLDocument thisDocument = (SLDocument) loopDocumentListIterator.next();
           String fileName = (String) thisDocument.getName();
           Float tfidf = (Float) thisDocument.getQueryRelevance();
           currentDocument.setQueryRelevance(tfidf);
           currentDocument.setPath(fileName);
           currentDocument.setName(fileName);
           currentDocumentHolder.setValue(currentDocument);
           variables.put(a.getName(),currentDocumentHolder);

           c = evaluateTree(codeOrStopCondition);
           variables.remove(a.getName());

        }
     }
     //todo fix for each loop
```

```
    }


    )
   | #(OR a=evaluateTree
   {
            b=evaluateTree(_t);
            slDataTypeInfo =  a.or(b);
    }
  )
  | #(AND  a=evaluateTree
                        {
                                        b=evaluateTree(_t);
                                        slDataTypeInfo =  a.and(b);
                        }
              )

  | #(EQUAL     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.equals(b); } )
  | #(NE     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.notEqual(b); } )
  | #(LT     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.lessThan(b); } )
  | #(LTE     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.lessThanOrEqual(b); } )
  | #(GT     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.greaterThan(b); } )
  | #(GE     a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.greaterThanOrEqual(b); } )
  | #(PLUS   a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.add(b); } )
  | #(MINUS  a=evaluateTree b=evaluateTree { if (b==null)
                              {
                                 SLDataTypeInfo dummy = new
SLDataTypeInfo(SLTypes.INTEGER,"0",true);
                                 slDataTypeInfo =  dummy.subtract(a);
                              }
                              else
                              {
                                slDataTypeInfo =  a.subtract(b);
                              }
                             } )
  | #(MULTIPLY    a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.multiply(b); } )
  | #(DIVIDE    a=evaluateTree b=evaluateTree { slDataTypeInfo =  a.divide(b); } )
  | #(NOT    a=evaluateTree        { slDataTypeInfo =  a.not(); } )
  | #(SIGNEDFACTOR a=evaluateTree  { slDataTypeInfo =  a.unary_minus(); } )
  | INTEGERLITERAL      { slDataTypeInfo = new SLDataTypeInfo(SLTypes.INTEGER,
#INTEGERLITERAL.getText(),true); }
  | STRINGLITERAL        { slDataTypeInfo = new SLDataTypeInfo(SLTypes.STRING,
#STRINGLITERAL.getText(),true); }
  | #(IF a=evaluateTree thenAST:. (elseAST:.)?)
      {
        if (  ((Float)a.getValue()).intValue() !=0 )
           slDataTypeInfo =  evaluateTree( #thenAST );
        else if ( elseAST != null )
           slDataTypeInfo =  evaluateTree( #elseAST );

      }
  | #(RETURN a=evaluateTree)
            {
                      slDataTypeInfo = a;
            }
```

```
 | #(FUNCTIONCALL
          {
                    String fName = _t.getText();
                    SLFunction slFunction =(SLFunction) functionMap.get(fName);
                    java.util.HashMap vArgs = new java.util.HashMap();
                    java.util.Vector functionArguments = slFunction.getArguments();
                    AST tArgs = _t.getNextSibling();
                    int count = 0;
                    while(tArgs != null) {
        SLDataTypeInfo arugmentNameAndType = (SLDataTypeInfo)
functionArguments.elementAt(count);
                              a = evaluateTree(tArgs);
        if (!a.isLiteral())
          vArgs.put(arugmentNameAndType.getName() ,variables.get(a.getName()));
        else vArgs.put(arugmentNameAndType.getName() ,a);
                              tArgs = tArgs.getNextSibling();
                              count++;
                    }
                    java.util.HashMap copyMap = variables;
                    variables = vArgs;
                    if (!slFunction.isBuiltin())
                       slDataTypeInfo =  evaluateTree(slFunction.getBody());
                    else slDataTypeInfo =  slFunction.invoke(variables);
                    variables = copyMap;
          }
  )
 | #(IDENTIFIER

                    {
                              slDataTypeInfo = (SLDataTypeInfo) variables.get(#IDENTIFIER.getText());
                    }
   );
```

4. Supporting Code

CodeRunner.java

```
import antlr.debug.misc.ASTFrame;

import java.io.DataInputStream;
import java.io.FileInputStream;

public class CodeRunner {
    public static void main(String[] args) throws Exception {
        FileInputStream filename = new FileInputStream(args[0]);
        SLLexer lexer = new SLLexer(new DataInputStream(filename));
        SLParser parser = new SLParser(lexer);
        parser.program();
        antlr.CommonAST ast = (antlr.CommonAST) parser.getAST();
        SLWalker walker = new SLWalker();
        walker.program(ast);

    }
}
```

MethodNotImplementedException.java

```
package exceptions;

/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 22, 2007
 * Time: 5:36:06 PM
 * To change this template use File | Settings | File Templates.
 */
public class MethodNotImplementedException extends RuntimeException{
    public MethodNotImplementedException() {
        super("Method not implemented");    //To change body of overridden methods use File | Settings | File
Templates.

    }

    public MethodNotImplementedException(String message) {
        super(message);    //To change body of overridden methods use File | Settings | File Templates.
    }
}
```

OperationNotSupportedException.java

```
package exceptions;

/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 22, 2007
 * Time: 6:00:53 PM
 * To change this template use File | Settings | File Templates.
 */
public class OperationNotSupportedException extends RuntimeException {
    public OperationNotSupportedException() {
        super("This operation is not supported");
    }

    public OperationNotSupportedException(String message) {
        super(message);
    }
}
```

VariableAlreadyDeclaredException.java

```
package exceptions;

/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 24, 2007
 * Time: 5:59:39 PM
 * To change this template use File | Settings | File Templates.
 */
public class VariableAlreadyDeclaredException extends RuntimeException {
    public VariableAlreadyDeclaredException(String s) {
    }
}
```

Porter.java

```
/**
 *
 * Poter.java is downloaded from some website. I am sorry that i do not have the reference.
 * I am not the author of this file. It was a working file available on net that implements porter stemming
algorithm.
 *
 */



class NewString {
  public String str;

  NewString() {
     str = "";
  }
}

public class Porter {

  private String Clean( String str ) {
     int last = str.length();
```

```
   Character ch = new Character( str.charAt(0) );
   String temp = "";

   for ( int i=0; i < last; i++ ) {
      if ( ch.isLetterOrDigit( str.charAt(i) ) )
         temp += str.charAt(i);
   }

   return temp;
} //clean

private boolean hasSuffix( String word, String suffix, NewString stem ) {

   String tmp = "";

   if ( word.length() <= suffix.length() )
      return false;
   if (suffix.length() > 1)
      if ( word.charAt( word.length()-2 ) != suffix.charAt( suffix.length()-2 ) )
         return false;

   stem.str = "";

   for ( int i=0; i<word.length()-suffix.length(); i++ )
       stem.str += word.charAt( i );
   tmp = stem.str;

   for ( int i=0; i<suffix.length(); i++ )
       tmp += suffix.charAt( i );

   if ( tmp.compareTo( word ) == 0 )
      return true;
   else
      return false;
}

private boolean vowel( char ch, char prev ) {
   switch ( ch ) {
      case 'a': case 'e': case 'i': case 'o': case 'u':
        return true;
      case 'y': {

        switch ( prev ) {
          case 'a': case 'e': case 'i': case 'o': case 'u':
            return false;

          default:
            return true;
        }
      }

      default :
        return false;
   }
}
```

```
private int measure( String stem ) {

  int i=0, count = 0;
  int length = stem.length();

  while ( i < length ) {
     for ( ; i < length ; i++ ) {
         if ( i > 0 ) {
            if ( vowel(stem.charAt(i),stem.charAt(i-1)) )
               break;
         }
         else {
            if ( vowel(stem.charAt(i),'a') )
              break;
         }
     }

     for ( i++ ; i < length ; i++ ) {
         if ( i > 0 ) {
            if ( !vowel(stem.charAt(i),stem.charAt(i-1)) )
                break;
            }
         else {
            if ( !vowel(stem.charAt(i),'?') )
               break;
         }
     }
    if ( i < length ) {
       count++;
       i++;
    }
  } //while

  return(count);
}

private boolean containsVowel( String word ) {

   for (int i=0 ; i < word.length(); i++ )
       if ( i > 0 ) {
          if ( vowel(word.charAt(i),word.charAt(i-1)) )
             return true;
       }
       else {
          if ( vowel(word.charAt(0),'a') )
             return true;
       }

   return false;
}

private boolean cvc( String str ) {
   int length=str.length();

   if ( length < 3 )
      return false;
   if ( (!vowel(str.charAt(length-1),str.charAt(length-2)) )
      && (str.charAt(length-1) != 'w') && (str.charAt(length-1) != 'x') && (str.charAt(length-1) != 'y')
      && (vowel(str.charAt(length-2),str.charAt(length-3))) ) {

      if (length == 3) {
         if (!vowel(str.charAt(0),'?'))
            return true;
         else
            return false;
      }
      else {
         if (!vowel(str.charAt(length-3),str.charAt(length-4)) )
            return true;
         else
            return false;
      }
   }

   return false;
}

private String step1( String str ) {

   NewString stem = new NewString();

   if ( str.charAt( str.length()-1 ) == 's' ) {
      if ( (hasSuffix( str, "sses", stem )) || (hasSuffix( str, "ies", stem)) ){
         String tmp = "";
         for (int i=0; i<str.length()-2; i++)
             tmp += str.charAt(i);
         str = tmp;
      }
      else {
         if ( ( str.length() == 1 ) && ( str.charAt(str.length()-1) == 's' ) ) {
            str = "";
            return str;
         }
         if ( str.charAt( str.length()-2 ) != 's' ) {
            String tmp = "";
            for (int i=0; i<str.length()-1; i++)
                tmp += str.charAt(i);
            str = tmp;
         }
      }
   }

   if ( hasSuffix( str,"eed",stem ) ) {
         if ( measure( stem.str ) > 0 ) {
            String tmp = "";
            for (int i=0; i<str.length()-1; i++)
                tmp += str.charAt( i );
            str = tmp;
         }
   }
   else {
      if (  (hasSuffix( str,"ed",stem )) || (hasSuffix( str,"ing",stem )) ) {
         if (containsVowel( stem.str ))  {

            String tmp = "";
            for ( int i = 0; i < stem.str.length(); i++)
                tmp += str.charAt( i );
            str = tmp;
            if ( str.length() == 1 )
               return str;

            if ( ( hasSuffix( str,"at",stem) ) || ( hasSuffix( str,"bl",stem ) ) || ( hasSuffix( str,"iz",stem) ) ) {
               str += "e";

            }
            else {
               int length = str.length();
               if ( (str.charAt(length-1) == str.charAt(length-2))
                  && (str.charAt(length-1) != 'l') && (str.charAt(length-1) != 's') && (str.charAt(length-1) !=
'z') ) {

                  tmp = "";
                  for (int i=0; i<str.length()-1; i++)
                      tmp += str.charAt(i);
                  str = tmp;
               }
               else
                  if ( measure( str ) == 1 ) {
                     if ( cvc(str) )
                        str += "e";
                  }
            }
         }
      }
   }

   if ( hasSuffix(str,"y",stem) )
      if ( containsVowel( stem.str ) ) {
         String tmp = "";
         for (int i=0; i<str.length()-1; i++ )
             tmp += str.charAt(i);
         str = tmp + "i";
      }
   return str;
  }

  private String step2( String str ) {

     String[][] suffixes = { { "ational", "ate" },
                             { "tional",  "tion" },
                             { "enci",    "ence" },
                             { "anci",    "ance" },
                             { "izer",    "ize" },
                             { "iser",    "ize" },
                             { "abli",    "able" },
                             { "alli",    "al" },
                             { "entli",   "ent" },
                             { "eli",     "e" },
                             { "ousli",   "ous" },
                             { "ization", "ize" },
                             { "isation", "ize" },
                             { "ation",   "ate" },
                             { "ator",    "ate" },
                             { "alism",   "al" },
                             { "iveness", "ive" },
                             { "fulness", "ful" },
                             { "ousness", "ous" },
                             { "aliti",   "al" },
                             { "iviti",   "ive" },
                             { "biliti",  "ble" }};
 NewString stem = new NewString();


 for ( int index = 0 ; index < suffixes.length; index++ ) {
    if ( hasSuffix ( str, suffixes[index][0], stem ) ) {
       if ( measure ( stem.str ) > 0 ) {
             str = stem.str + suffixes[index][1];
             return str;
       }
    }
 }

 return str;
  }

  private String step3( String str ) {

      String[][] suffixes = { { "icate", "ic" },
                              { "ative", "" },
                              { "alize", "al" },
                              { "alise", "al" },
                              { "iciti", "ic" },
                              { "ical",  "ic" },
                              { "ful",   "" },
                              { "ness",  "" }};
      NewString stem = new NewString();

      for ( int index = 0 ; index<suffixes.length; index++ ) {
         if ( hasSuffix ( str, suffixes[index][0], stem ))
             if ( measure ( stem.str ) > 0 ) {
                str = stem.str + suffixes[index][1];
                return str;
             }
      }
      return str;
  }

  private String step4( String str ) {

 String[] suffixes = { "al", "ance", "ence", "er", "ic", "able", "ible", "ant", "ement", "ment", "ent", "sion",
"tion",
                 "ou", "ism", "ate", "iti", "ous", "ive", "ize", "ise"};
 NewString stem = new NewString();

 for ( int index = 0 ; index<suffixes.length; index++ ) {
     if ( hasSuffix ( str, suffixes[index], stem ) ) {

             if ( measure ( stem.str ) > 1 ) {
                str = stem.str;
                return str;
             }
     }
 }
 return str;
}

private String step5( String str ) {

 if ( str.charAt(str.length()-1) == 'e' ) {
      if ( measure(str) > 1 ) {/* measure(str)==measure(stem) if ends in vowel */
         String tmp = "";
         for ( int i=0; i<str.length()-1; i++ )
             tmp += str.charAt( i );
         str = tmp;
      }
      else
         if ( measure(str) == 1 ) {
             String stem = "";
             for ( int i=0; i<str.length()-1; i++ )
                 stem += str.charAt( i );

             if ( !cvc(stem) )
                str = stem;
         }
 }

 if ( str.length() == 1 )
      return str;
 if ( (str.charAt(str.length()-1) == 'l') && (str.charAt(str.length()-2) == 'l') && (measure(str) > 1) )
      if ( measure(str) > 1 ) {/* measure(str)==measure(stem) if ends in vowel */
         String tmp = "";
         for ( int i=0; i<str.length()-1; i++ )
             tmp += str.charAt( i );
         str = tmp;
      }
 return str;
}

private String stripPrefixes ( String str) {

 String[] prefixes = { "kilo", "micro", "milli", "intra", "ultra", "mega", "nano", "pico", "pseudo"};

 int last = prefixes.length;
 for ( int i=0 ; i<last; i++ ) {
     if ( str.startsWith( prefixes[i] ) ) {
        String temp = "";
        for ( int j=0 ; j< str.length()-prefixes[i].length(); j++ )
                temp += str.charAt( j+prefixes[i].length() );
```

```
          return temp;
        }
    }

    return str;
  }


  private String stripSuffixes( String str ) {

    str = step1( str );
    if ( str.length() >= 1 )
      str = step2( str );
    if ( str.length() >= 1 )
      str = step3( str );
    if ( str.length() >= 1 )
      str = step4( str );
    if ( str.length() >= 1 )
      str = step5( str );

    return str;
  }


  public String stripAffixes( String str ) {

    str = str.toLowerCase();
    str = Clean(str);

    if (( str != "" ) && (str.length() > 2)) {
      str = stripPrefixes(str);

      if (str != "" )
        str = stripSuffixes(str);

    }

    return str;
    } //stripAffixes

} //class
```

SLDataTypeInfo.java

```
import exceptions.MethodNotImplementedException;
import exceptions.OperationNotSupportedException;

import java.util.HashSet;
import java.util.Iterator;

/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 22, 2007
 * Time: 3:02:10 PM
 * To change this template use File | Settings | File Templates.
 */

public class SLDataTypeInfo implements Cloneable, Comparable {

    public static final SLDataTypeInfo FALSE = new SLDataTypeInfo(SLTypes.INTEGER, "0", true);
    // anything not zero is true.
    public static final SLDataTypeInfo TRUE = new SLDataTypeInfo(SLTypes.INTEGER, "1", true);
    SLTypes type;
    String name;
    Object value;
    boolean isLiteral;


    public boolean isLiteral() {
        return isLiteral;
    }

    public void setLiteral(boolean literal) {
        isLiteral = literal;
    }

    public SLDataTypeInfo(SLTypes type, String name) {
        this.type = type;
        this.name = name;
        this.isLiteral = false;
    }

    public SLDataTypeInfo(SLTypes type, String name, Object value) {
        this.type = type;
        this.name = name;
        setValue(value);
        this.isLiteral = false;
    }

    public SLDataTypeInfo(SLTypes type, Object value, boolean isLiteral) {
        this.type = type;
        this.name = null;
        setValue(value);
        this.isLiteral = true;
    }
    public SLTypes getType() {
        return type;
    }

    public void setType(SLTypes type) {
        this.type = type;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public Object getValue() {

        return value;
    }

    public void setValue(Object value) {
        String strValue = value.toString();
        if (type == null) this.value = value;
        else
            switch (type) {
                case INTEGER:
                    this.value = new Float(strValue);
                    break;
                case STRING:
                case QUERY:
                case DOCUMENTLIST:
                case REPOSITORY:
                case DOCUMENT:
                    this.value = value;
                    break;
                default:
                    throw new RuntimeException("Unexpected type detected");
            }
    }


    /**
     * Should never be used
     */
    public SLDataTypeInfo() {
        name = null;
    }


    public SLDataTypeInfo assign(SLDataTypeInfo b) throws MethodNotImplementedException,
ClassCastException {
        //TODO FIX CASTINGS (STRING-->QUERY AND QUERY-->sTRING ETC)
        if (!this.type.equals(b.getType()))
            throw new ClassCastException(b.getType() + " cannot be assinged to a " + getType());
        value = b.value;

        return this;
    }

    public SLDataTypeInfo unary_minus() throws MethodNotImplementedException,
OperationNotSupportedException {
        switch (type) {
            case INTEGER:
                value = new Float(((Float) value).floatValue() * -1);
                break;

            default:
                throw new OperationNotSupportedException();
        }
        return this;
    }

    /**
     * Integer + Integer = Integer
     * String + Integer | String = String
     * Query + String = Query
     * String + Query = Query (this and prev are interchangeable)
     * Document + anything is not supported
     * DocumentList + DocumentList = a union of documents
     * Query + Query = Query (that contains union of search words)
     * Repository + Repository = a union of two repositories
     *
     * @param b
     * @return
     * @throws MethodNotImplementedException
     */


    public SLDataTypeInfo add(SLDataTypeInfo b) throws MethodNotImplementedException {

        SLDataTypeInfo returnData = null;
        try {
            returnData = (SLDataTypeInfo) this.clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
        }
        switch (type) {
            case INTEGER:
                switch (b.getType()) {
                    case INTEGER:
                        returnData.value = new Float(((Float) value).floatValue() + (Float) b.getValue());
                        returnData.type = SLTypes.INTEGER;
                        break;
                    case STRING:
                        returnData.type = SLTypes.STRING;
                        returnData.value = "" + ((Float) getValue()).toString() + b.getValue();
                        break;
                    default:
                        throw new OperationNotSupportedException();
                }
                break;
            case STRING:
                switch (b.getType()) {
                    case INTEGER:
                    case STRING:
                    case QUERY:

                        returnData.value = value + b.getValue().toString();
                        returnData.type = SLTypes.STRING;

                        break;

                    default:
                        throw new OperationNotSupportedException();
                }
                break;
            case DOCUMENT:
                throw new OperationNotSupportedException();

            case DOCUMENTLIST:
                switch (b.getType()) {
                    case DOCUMENTLIST:

                        returnData.value = new HashSet();
                        ((HashSet) returnData.value).addAll((HashSet) value);
                        ((HashSet) returnData.value).addAll((HashSet) b.getValue());
                        returnData.type = SLTypes.DOCUMENTLIST;

                        break;

                    default:
                        throw new OperationNotSupportedException();
                }
                break;

            case REPOSITORY:
                //todo need to think about this
                returnData.value = new Float(((Float) value).floatValue() * -1);
                returnData.type = SLTypes.REPOSITORY;
                break;
            case QUERY:
                switch (b.getType()) {
                    case INTEGER:
                    case STRING:
                    case QUERY:

                        returnData.value = value + b.getValue().toString();
                        returnData.type = SLTypes.QUERY;
                        break;

                    default:
                        throw new OperationNotSupportedException();
                }
                break;

            default:
                throw new OperationNotSupportedException();
        }
        return returnData;


    }

    public SLDataTypeInfo subtract(SLDataTypeInfo b) throws MethodNotImplementedException {
        switch (type) {
            case INTEGER:
                switch (b.getType()) {
                    case INTEGER:
                        value = new Float(((Float) value).floatValue() - (Float) b.getValue());
                        break;
                    default:
                        throw new OperationNotSupportedException();
                }
                break;
            case STRING:
                throw new OperationNotSupportedException();
            case DOCUMENT:
                throw new OperationNotSupportedException();

            case DOCUMENTLIST:
                switch (b.getType()) {
                    case DOCUMENTLIST:
                        HashSet bSet = (HashSet) b.getValue();
                        HashSet aSet = (HashSet) getValue();
                        if (bSet != null && aSet != null) {
                            Iterator iterator = bSet.iterator();
                            while (iterator.hasNext()) {
                                aSet.remove(iterator.next());

                            }
                        }
                        break;

                    default:
                        throw new OperationNotSupportedException();
                }
                break;

            case REPOSITORY:
                //todo need to think about this
                value = new Float(((Float) value).floatValue() * -1);
                break;
            case QUERY:
                throw new OperationNotSupportedException();

            default:
                throw new OperationNotSupportedException();
        }
        return this;
    }

    public SLDataTypeInfo multiply(SLDataTypeInfo b) throws MethodNotImplementedException {
        switch (type) {
            case INTEGER:
                switch (b.getType()) {
                    case INTEGER:
                        value = new Float(((Float) value).floatValue() * (Float) b.getValue());
                        break;
                    default:
                        throw new OperationNotSupportedException();
                }
                break;
            default:
                throw new OperationNotSupportedException();
        }


        return this;
    }

    public SLDataTypeInfo divide(SLDataTypeInfo b) throws MethodNotImplementedException {
        switch (type) {
            case INTEGER:
                switch (b.getType()) {
                    case INTEGER:

                        value = new Float(((Float) value).floatValue() / (Float) b.getValue());
                        break;
                    default:
                        throw new OperationNotSupportedException();
                }
                break;
            default:
                throw new OperationNotSupportedException();
        }


        return this;
    }


    public SLDataTypeInfo greaterThan(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (this.type == b.getType()
                && compareTo(b) > 0) return TRUE;
        return FALSE;
    }

    public SLDataTypeInfo greaterThanOrEqual(SLDataTypeInfo b) throws
MethodNotImplementedException {
        if (this.type == b.getType()
                && compareTo(b) >= 0) return TRUE;
        return FALSE;

    }

    public SLDataTypeInfo lessThan(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (this.type == b.getType()
                && compareTo(b) < 0) return TRUE;
        return FALSE;
    }

    public SLDataTypeInfo lessThanOrEqual(SLDataTypeInfo b) throws MethodNotImplementedException
{
        if (this.type == b.getType()
                && compareTo(b) <= 0) return TRUE;
        return FALSE;
    }

    public SLDataTypeInfo equals(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (this.type == b.getType()
                && this.value.equals(b.getType())) return TRUE;
        return FALSE;


    }

    public SLDataTypeInfo notEqual(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (this.type == b.getType()
                && compareTo(b) != 0) return TRUE;
        return FALSE;

    }

    public SLDataTypeInfo and(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (((Float) this.value).floatValue() == 0 || ((Float) this.value).floatValue() == 0)
            return FALSE;
        return TRUE;
    }

    public SLDataTypeInfo or(SLDataTypeInfo b) throws MethodNotImplementedException {
        if (((Float) this.value).floatValue() != 0 || ((Float) this.value).floatValue() != 0)
            return TRUE;
        return FALSE;
    }

    public SLDataTypeInfo not() throws MethodNotImplementedException {
        if (this.getValue().equals(FALSE)) return TRUE;
        return FALSE;
    }


    public SLDataTypeInfo setIntValue(String text) {
        this.value = new Float(text);
        this.type = SLTypes.INTEGER;
        return this;  //To change body of created methods use File | Settings | File Templates.
    }

    public SLDataTypeInfo setStringValue(String text) {
        this.value = new String(text);
        this.type = SLTypes.STRING;
        return this;  //To change body of created methods use File | Settings | File Templates.

    }
    public SLDataTypeInfo increment()
    {
        this.value = ((Float)this.value).floatValue()+1;
        return this;
    }
    public SLDataTypeInfo decrement()
    {
        this.value = ((Float)this.value).floatValue()-1;
        return this;
    }
    public int compareTo(Object o) {
        SLDataTypeInfo slDataTypeInfo = (SLDataTypeInfo) o;
        if (this.type == slDataTypeInfo.type) {
            switch (type) {
                case INTEGER:
                    return ((Float) value).compareTo((Float) slDataTypeInfo.getValue());
                case STRING:
                    ;
                case QUERY:
                    return ((String) value).compareTo((String) slDataTypeInfo.getValue());
                default:
                    throw new OperationNotSupportedException(this.getType() + " and  " +
slDataTypeInfo.getType() + " cannot be compared.");

            }
        }
        return 0;  //To change body of implemented methods use File | Settings | File Templates.
    }

    public Object clone() throws CloneNotSupportedException {
        SLDataTypeInfo newlyCreatedClone = (SLDataTypeInfo) super.clone();
        newlyCreatedClone.value = value;
        newlyCreatedClone.type = type;
        newlyCreatedClone.setName("");

        return newlyCreatedClone;
    }

    public String toString() {
        return value.toString();    //To change body of overridden methods use File | Settings | File Templates.
    }
}
```

SLDocument.java

```
import java.util.HashMap;

/**
 * Created by IntelliJ IDEA.
 * User: khanm
 * Date: Dec 5, 2007
 * Time: 1:27:06 PM
 * To change this template use File | Settings | File Templates.
 */

public class SLDocument
{

    String name;
    String path;
    String repository;
    float queryRelevance;
    HashMap terms;
    int totalTerms;

    public float getQueryRelevance() {
        return queryRelevance;
    }

    public void setQueryRelevance(float queryRelevance) {
        this.queryRelevance = queryRelevance;
    }

    public SLDocument() {
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }

    public String getRepository() {
        return repository;
    }

    public void setRepository(String repository) {
        this.repository = repository;
    }
    public HashMap getTerms() {
        return terms;
    }

    public void setTerms(HashMap terms) {
        this.terms = terms;
    }

    public String getData() {
        //todo get data from the files.
        return "";
    }

    public String toString() {
        return  name;
    }

}
```

SLDocumentList.java

```
import java.util.TreeMap;

/**
 * Created by IntelliJ IDEA.
 * User: khanm
 * Date: Dec 5, 2007
 * Time: 1:36:26 PM
 * To change this template use File | Settings | File Templates.
 */
public class SLDocumentList

{
    TreeMap documents = new TreeMap();


    public TreeMap getDocuments() {
        return documents;
    }

    public void setDocuments(TreeMap documents) {
        this.documents = documents;
    }

    public void addDocument(SLDocument slDocument)
    {
        if (documents.containsKey(slDocument.getQueryRelevance()))
        {
            slDocument.setQueryRelevance(slDocument.queryRelevance+0.000001f);
            addDocument(slDocument);
        }
        else documents.put(slDocument.getQueryRelevance(),slDocument);

    }
    public SLDocument removeDocument (float queryRelevance)
    {
        if (documents.containsKey(queryRelevance))
        {
            return (SLDocument) documents.remove(queryRelevance);
        }
        return null;
    }

    public String toString() {
        return documents.toString();    //To change body of overridden methods use File | Settings | File
Templates.
    }
}
```

SLFunction.java

```
import antlr.collections.AST;

import java.util.Vector;
import java.util.HashMap;

/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 23, 2007
 * Time: 12:26:20 AM
 * To change this template use File | Settings | File Templates.
 */
public class SLFunction {

    String name;
    SLTypes returnType;
    AST body;
    Vector arguments;
    boolean builtin = false;

    /**
     * This method returns the value true if a method is not coded by user and is provided in language.
     *
     * @return
     */
    public boolean isBuiltin() {
        return builtin;
    }

    public SLFunction(String name, SLTypes returnType, Vector arguments, AST body) {
        this.name = name;
        this.returnType = returnType;
        this.body = body;
        this.arguments = arguments;
        builtin =  body == null;
    }

    /**
     * This method invokes all the builtin methods. As defined in the Tree walker, if a function is builtin
     * This method is called with arguments and then this method routes the call to appropriate method
     * @param vArgs
     * @return
     */

    public SLDataTypeInfo invoke(HashMap vArgs)
    {   SLDataTypeInfo slDataTypeInfo=null;
        if (body==null)// builtin function
        {
           if (name.equalsIgnoreCase("print"))
              SLUtils.print(vArgs);
           else if (name.equals("makerepository"))
           {    return SLUtils.makerepository(vArgs);

           }
           else if (name.equals("processrepository"))
              return SLUtils.processrepository(vArgs);
           else if (name.equals("performquery"))
              return SLUtils.performquery(vArgs);
           else if (name.equals("getdocumenturi"))
              return SLUtils.getDocumentURI(vArgs);
           else if (name.equals("getdocumentdata"))
              return SLUtils.getDocumentURI(vArgs);

           else throw new RuntimeException("Unknown function call.");
        }


        return slDataTypeInfo;
    }

    public Vector getArguments() {
        return arguments;
    }

    public void setArguments(Vector arguments) {
        this.arguments = arguments;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public SLTypes getReturnType() {
        return returnType;
    }

    public void setReturnType(SLTypes returnType) {
        this.returnType = returnType;
    }
```

```
    public AST getBody() {
        return body;
    }

    public void setBody(AST body) {
        this.body = body;
    }


    public String toString() {
        return name + "\n" + body.toString();
    }
}
```

SLRepository.java

```
import java.io.Serializable;
import java.util.HashMap;
import java.util.StringTokenizer;

/**
 * Created by IntelliJ IDEA.
 * User: khanm
 * Date: Dec 6, 2007
 * Time: 6:30:18 PM
 * To change this template use File | Settings | File Templates.
 */

class TermDocumentFrequencyHolder implements Serializable {
    int totalFrequency = 0;

    public int getTotalFrequency() {
        return totalFrequency;
    }

    public HashMap getTermDocumentFrequency() {
        return termDocumentFrequency;
    }

    public void setTotalFrequency(int totalFrequency) {
        this.totalFrequency = totalFrequency;
    }

    public String getTerm() {
        return term;
    }

    public void setTerm(String term) {
        this.term = term;
    }

    String term;
    HashMap termDocumentFrequency = new HashMap();
}

public class SLRepository implements Serializable {
```

```
String location;
String include;
String type;
boolean processed;
int totalNumberOfDocument = 0;
String matches[];
HashMap documentMap = new HashMap();

public SLRepository(String location, String include, String type) {
   this.location = location;
   setInclude(include);
   this.type = type;
}

public void incDocumentCount() {
   totalNumberOfDocument++;
}

public int getDocumentCount() {
   return totalNumberOfDocument;
}

public void updateTerm(String term, String file) {
   TermDocumentFrequencyHolder tmap = (TermDocumentFrequencyHolder) documentMap.get(term);
   if (tmap == null) {
      tmap = new TermDocumentFrequencyHolder();
      documentMap.put(term, tmap);

   }
   Integer documentFreq;
   documentFreq = (Integer) tmap.getTermDocumentFrequency().get(file);
   if (documentFreq == null) {
      documentFreq = 1;
      tmap.getTermDocumentFrequency().put(file, documentFreq);
   } else {
      tmap.getTermDocumentFrequency().put(file, documentFreq + 1);
   }
   tmap.setTotalFrequency(tmap.getTotalFrequency() + 1);

}


public void removeDocument(String file) {

}

public String getLocation() {
   return location;
}

public void setLocation(String location) {
   this.location = location;
}

public boolean isProcessed() {
   return processed;
}

public void setProcessed(boolean processed) {
    this.processed = processed;
}

public String getInclude() {
    return include;
}

public void setInclude(String include) {
    this.include = include;

    if (include == null || include.trim().equals(""))
        include = "*";
    StringTokenizer st = new StringTokenizer(include, ",");
    matches = new String[st.countTokens()];
    for (int count = 0; count < matches.length; count++)
    //while (st.hasMoreTokens())
    {
        String currentMatch = st.nextToken().trim();
        if (currentMatch.equals("*") || currentMatch.equals("*.*")) {
            matches = new String[]{".*"};
            break;
        } else {


            StringBuffer sb = new StringBuffer();
            for (int counter = 0; counter < currentMatch.length(); counter++) {
                switch (currentMatch.charAt(counter)) {
                    case'.':
                        sb.append("\\.");
                        break;
                    case'*':
                        sb.append(".*");
                        break;
                    case'?':
                        sb.append(".");
                        break;
                    default:
                        sb.append(currentMatch.charAt(counter));
                }


            }
            matches[count] = sb.toString();
        }

    }

}

public String getType() {
    return type;
}
```

```
    public void setType(String type) {
        this.type = type;
    }

    public String[] getMatches() {
        return matches;
    }
}
```

SLTypes.java

```
/**
 * Created by IntelliJ IDEA.
 * User: khan
 * Date: Nov 22, 2007
 * Time: 5:23:09 PM
 * To change this template use File | Settings | File Templates.
 */
public enum SLTypes {
    INTEGER, STRING, REPOSITORY, DOCUMENTLIST, DOCUMENT, QUERY, VOID;


}
```

SLUtils.java

```
import org.apache.lucene.analysis.standard.StandardTokenizer;
import org.htmlparser.Parser;
import org.htmlparser.beans.StringBean;
import org.htmlparser.lexer.StringSource;
import org.htmlparser.parserapplications.StringExtractor;
import org.xml.sax.SAXException;

import java.io.*;
import java.util.*;

/**
 * Created by IntelliJ IDEA.
 * User: khanm
 * Date: Dec 5, 2007
 * Time: 1:01:59 PM
 * To change this template use File | Settings | File Templates.
 */
public class SLUtils {

    /**
     * This function registers builtin library methods and their parameters.
     *
     * @param functionMap
     */
    public static void loadBuiltinMethods(HashMap functionMap) {
        Vector args = new Vector();
        args.add(new SLDataTypeInfo(SLTypes.STRING, "args1"));
        SLFunction printFunction = new SLFunction("print", null, args, null);
        functionMap.put("print", printFunction);

        args = new Vector();
        args.add(new SLDataTypeInfo(SLTypes.DOCUMENT, "document"));
        SLFunction getDocumentURIFunction = new SLFunction("getdocumenturi", SLTypes.STRING, args,
null);
        functionMap.put("getdocumenturi", getDocumentURIFunction);
        args = new Vector();
        args.add(new SLDataTypeInfo(SLTypes.DOCUMENT, "document"));
        SLFunction getDocumentDataFunction = new SLFunction("getdocumentdata", SLTypes.STRING,
args, null);
        functionMap.put("getdocumentdata", getDocumentDataFunction);

        args = new Vector();////( string location, string include, string type) returns repository
        args.add(new SLDataTypeInfo(SLTypes.STRING, "location"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "include"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "type"));
        SLFunction makerepositoryFunction = new SLFunction("makerepository", SLTypes.REPOSITORY,
args, null);
        functionMap.put("makerepository", makerepositoryFunction);


        args = new Vector();
        args.add(new SLDataTypeInfo(SLTypes.REPOSITORY, "arep"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "querystring"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "store"));
        SLFunction performQueryFunction = new SLFunction("performquery", SLTypes.DOCUMENTLIST,
args, null);
        functionMap.put("performquery", performQueryFunction);


        args = new Vector();////(string location, string stemmer , string overwrite,string filename)
        args.add(new SLDataTypeInfo(SLTypes.STRING, "location"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "stemmer"));
        args.add(new SLDataTypeInfo(SLTypes.INTEGER, "overwrite"));
        args.add(new SLDataTypeInfo(SLTypes.STRING, "filename"));
        SLFunction processrepositoryFunction = new SLFunction("processrepository", null, args, null);
        functionMap.put("processrepository", processrepositoryFunction);


    }

    /**
     * this method prints the argument to conosole
     *
     * @param args
     * @return
     */
    public static SLDataTypeInfo print(HashMap args) {
        System.out.println( args.get("args1"));
        return null;
    }
    /**
     * This method returns the URI for the document passed as parameter.
     *
     * @param args
     * @return
     */
    public static SLDataTypeInfo getDocumentURI(HashMap args) //document;
    {
        return new SLDataTypeInfo(SLTypes.STRING, args.get("document").toString(), true);
    }

    /**
     * This method returns the data for the document passed as parameter.
     *
     * @param args
     * @return
     */
    public static SLDataTypeInfo getDocumentData(HashMap args) //document;
    {
        return new SLDataTypeInfo(SLTypes.STRING, ((SLDocument) args.get(0)).getData(), true);

    }

    /**
     * This method instantiates the repository.
     *
     * @param args
     * @return
     */
    public static SLDataTypeInfo makerepository(HashMap args) //( string location, string include, string
type) returns repository;
    {
        SLRepository slRepository = new SLRepository((String) ((SLDataTypeInfo)
args.get("location")).getValue(),
                (String) ((SLDataTypeInfo) args.get("include")).getValue(),
                (String) ((SLDataTypeInfo) args.get("type")).getValue());
        return new SLDataTypeInfo(SLTypes.REPOSITORY, slRepository, true);
    }

    /**
     * This method computes statistical data on all the document that are contained in repository
     * and store the result for later use. This data is then used by performquery operation at the time of
     * search. Only implementation of tf-idf using cosine similarity is implemented.
     *
     * @param args
     * @return
     */


    public static SLDataTypeInfo processrepository(HashMap args) //(string location, string stemmer , string
overwrite,string filename)
    {

        SLRepository slRepository = ((SLRepository) ((SLDataTypeInfo) args.get("location")).getValue());
        String location = slRepository.getLocation();
        String stemmer = (String) ((SLDataTypeInfo) args.get("stemmer")).getValue();    // default value
```

```
porter would be used
        String overwrite = (String) ((SLDataTypeInfo) args.get("overwrite")).getValue();// default value yes
would be used
        String filename = (String) ((SLDataTypeInfo) args.get("filename")).getValue();  //
        Stack fileStalk = new Stack();
        File start = new File(location);
        fileStalk.push(start);
        do {


            File f = (File) fileStalk.pop();
            File[] categories = f.listFiles();

            for (int categoryCount = 0; categoryCount < categories.length; categoryCount++) {

                File category = categories[categoryCount];
                if (category.isDirectory()) fileStalk.push(category);

                else {
                    if (patternMatchesFile(slRepository.getMatches(), category.toString())) {
                        processFile(category, slRepository);
                        System.out.println("category = " + category);
                    }
                }

                //File[] instances = category.listFiles();

            }
        }
        while ((!fileStalk.isEmpty()));
        try {
            FileOutputStream fos = new FileOutputStream(slRepository.getLocation()+"\\"+filename);
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(slRepository);
            oos.flush();
            oos.close();
            fos.close();
        } catch (FileNotFoundException e) {
            e.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
        } catch (IOException e) {
            e.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
        }

        return null;
    }

    private static void processFile(File file, SLRepository repository) {
        try {
            List terms = readFile(file);
            repository.incDocumentCount();
            Iterator termsIterator = terms.iterator();
            while (termsIterator.hasNext())
            {
                String term = (String) termsIterator.next();
                repository.updateTerm(term,file.toString());
            }
        } catch (Exception e) {
            e.printStackTrace();  //To change body of catch statement use File | Settings | File Templates.
        }
    }

    /**
     * This method executes the query on repository and stores result in file passed in store
     * parameter and return documentList in reference.
     *
     * @param args
     * @return
     */
    public static SLDataTypeInfo performquery(HashMap args) //(repository arep, string queryString, string
store) returns DocumentList;

    {   SLDocumentList searchResult =  new SLDocumentList();
        SLDataTypeInfo returnValue = new
SLDataTypeInfo(SLTypes.DOCUMENTLIST,searchResult,true);
        SLRepository slRepository = ((SLRepository) ((SLDataTypeInfo) args.get("arep")).getValue());
        String location = slRepository.getLocation();
        String queryString = (String) ((SLDataTypeInfo) args.get("querystring")).getValue();    // default value
porter would be used
        String store = (String) ((SLDataTypeInfo) args.get("store")).getValue();// default value yes would be
used
        //String filename = (String) ((SLDataTypeInfo) args.get("filename")).getValue();  //
        StringTokenizer st = new StringTokenizer(queryString);
        while(st.hasMoreTokens())
        {

            String queryWord = st.nextToken();
            queryWord = queryWord.toLowerCase().trim();
            queryWord = porter.stripAffixes(queryWord);
            TermDocumentFrequencyHolder docHolder = (TermDocumentFrequencyHolder)
slRepository.documentMap.get(queryWord);
            if (docHolder!=null)//no document found

            {
                Iterator keysetIterator = docHolder.getTermDocumentFrequency().entrySet().iterator();
                while (keysetIterator.hasNext())
                {
                    Map.Entry entry = (Map.Entry) keysetIterator.next();
                    double totalNumberOfDocuments = slRepository.getDocumentCount();
                    double numberOfDocumentsThatContainThisTerm =
docHolder.getTermDocumentFrequency().size();
                    String thisDocument = (String) entry.getKey();
                    Integer thisDocumentsTermFrequency = (Integer) entry.getValue();
                    float tfidf = (float) (thisDocumentsTermFrequency * Math.log(totalNumberOfDocuments/
(numberOfDocumentsThatContainThisTerm)));
                    SLDocument slDocument = new SLDocument();
                    slDocument.setName(thisDocument);
                    slDocument.setPath(thisDocument);
                    slDocument.setQueryRelevance(-tfidf);
                    searchResult.addDocument(slDocument);
                }
            }
        }
        return returnValue;
    }

    /**
     * @param pattern
     * @param file
     * @return
     */
    public static boolean patternMatchesFile(String[] pattern, String file) {
        boolean patternMatched = false;
        int count = 0;
        while (!patternMatched && count < pattern.length) {
            patternMatched = file.matches(pattern[count]);
            count++;
        }
        return patternMatched;
    }

    private static final int MAX_WORD_SIZE = 20;
    static HashMap stopMap = new HashMap();


    static Porter porter = new Porter();


    public static List readFile(File file) throws Exception, SAXException {
        String url = "";


        if (file.exists())
            url = file.toURL().toString();
        //todo exception handling when file does not exsit

        StringExtractor stringExtractor = new StringExtractor(url);

        ArrayList features = new ArrayList();
        getFeatures(stringExtractor.extractStrings(false), features, porter,
                true);
        return features;
    }





    public static void getFeatures(String data, ArrayList words, Porter
            stemmer, boolean stemming) throws Exception {

        if (data != null) {
            // Parse and add tokens....
//              System.out.println(st);
            StandardTokenizer lt = new StandardTokenizer(new
                    StringReader(data));
            org.apache.lucene.analysis.Token t = null;
            while ((t = lt.next()) != null) {
```

```
            String token = t.termText().toLowerCase();

            token = token.trim();
            if (token.length() < MAX_WORD_SIZE) {
                if (!stopMap.containsKey(token)) {
                    if (stemming) {
                        token = stemmer.stripAffixes(token);
                    }
                    words.add(token);
                }

            } else continue;

            // Add word to this file (or increase frequency)


        }
    } else {
        System.out.println("Can't retrieve text content from the page...");
    }
}

//some stop words. these words would be removed before stemming.

private static String[] stopWords =
        {
"a", "a's", "able", "about", "above", "according", "accordingly", "across", "actually", "after",
"afterwards", "again", "against", "ain't", "all", "allow", "allows", "almost", "alone", "along",
"already", "also", "although", "always", "am", "among", "amongst", "an", "and", "another", "any",
"anybody", "anyhow", "anyone", "anything", "anyway", "anyways", "anywhere", "apart", "appear",
"appreciate", "appropriate", "are", "aren't", "around", "as", "aside", "ask", "asking",
"associated", "at", "available", "away", "awfully", "b", "be", "became", "because", "become",
"becomes", "becoming", "been", "before", "beforehand", "behind", "being", "believe", "below",
"beside", "besides", "best", "better", "between", "beyond", "both", "brief", "but", "by", "c",
"c'mon", "c's", "came", "can", "can't", "cannot", "cant", "cause", "causes", "certain",
"certainly", "changes", "clearly", "co", "com", "come", "comes", "concerning", "consequently",
"consider", "considering", "contain", "containing", "contains", "corresponding", "could",
"couldn't", "course", "currently", "d", "definitely", "described", "despite", "did", "didn't",
"different", "do", "does", "doesn't", "doing", "don't", "done", "down", "downwards", "during",
"e", "each", "edu", "eg", "eight", "either", "else", "elsewhere", "enough", "entirely",
"especially", "et", "etc", "even", "ever", "every", "everybody", "everyone", "everything",
"everywhere", "ex", "exactly", "example", "except", "f", "far", "few", "fifth", "first", "five",
"followed", "following", "follows", "for", "former", "formerly", "forth", "four", "from",
"further", "furthermore", "g", "get", "gets", "getting", "given", "gives", "go", "goes",
"going", "gone", "got", "gotten", "greetings", "h", "had", "hadn't", "happens", "hardly",
"has", "hasn't", "have", "haven't", "having", "he", "he's", "hello", "help", "hence", "her",
"here", "here's", "hereafter", "hereby", "herein", "hereupon", "hers", "herself", "hi", "him",
"himself", "his", "hither", "hopefully", "how", "howbeit", "however", "i", "i'd", "i'll",
"i'm", "i've", "ie", "if", "ignored", "immediate", "in", "inasmuch", "inc", "indeed", "indicate",
"indicated", "indicates", "inner", "insofar", "instead", "into", "inward", "is", "isn't", "it",
"it'd", "it'll", "it's", "its", "itself", "j", "just", "k", "keep", "keeps", "kept", "know",
"knows", "known", "l", "last", "lately", "later", "latter", "latterly", "least", "less",
"lest", "let", "let's", "like", "liked", "likely", "little", "look", "looking", "looks", "ltd",
"m", "mainly", "many", "may", "maybe", "me", "mean", "meanwhile", "merely", "might", "more",
"moreover", "most", "mostly", "much", "must", "my", "myself", "n", "name", "namely", "nd",
"near", "nearly", "necessary", "need", "needs", "neither", "never", "nevertheless", "new",
    "next", "nine", "no", "nobody", "non", "none", "noone", "nor", "normally", "not", "nothing",
    "novel", "now", "nowhere", "o", "obviously", "of", "off", "often", "oh", "ok", "okay", "old",
    "on", "once", "one", "ones", "only", "onto", "or", "other", "others", "otherwise", "ought",
    "our", "ours", "ourselves", "out", "outside", "over", "overall", "own", "p", "particular",
    "particularly", "per", "perhaps", "placed", "please", "plus", "possible", "presumably", "probably",
    "provides", "q", "que", "quite", "qv", "r", "rather", "rd", "re", "really", "reasonably",
    "regarding", "regardless", "regards", "relatively", "respectively", "right", "s", "said",
    "same", "saw", "say", "saying", "says", "second", "secondly", "see", "seeing", "seem", "seemed",
    "seeming", "seems", "seen", "self", "selves", "sensible", "sent", "serious", "seriously", "seven",
    "several", "shall", "she", "should", "shouldn't", "since", "six", "so", "some", "somebody",
    "somehow", "someone", "something", "sometime", "sometimes", "somewhat", "somewhere", "soon",
    "sorry", "specified", "specify", "specifying", "still", "sub", "such", "sup", "sure", "t",
    "t's", "take", "taken", "tell", "tends", "th", "than", "thank", "thanks", "thanx", "that",
    "that's", "thats", "the", "their", "theirs", "them", "themselves", "then", "thence", "there",
    "there's", "thereafter", "thereby", "therefore", "therein", "theres", "thereupon", "these",
    "they", "they'd", "they'll", "they're", "they've", "think", "third", "this", "thorough",
    "thoroughly", "those", "though", "three", "through", "throughout", "thru", "thus", "to",
    "together", "too", "took", "toward", "towards", "tried", "tries", "truly", "try", "trying",
    "twice", "two", "u", "un", "under", "unfortunately", "unless", "unlikely", "until", "unto",
    "up", "upon", "us", "use", "used", "useful", "uses", "using", "usually", "uucp", "v", "value",
    "various", "very", "via", "viz", "vs", "w", "want", "wants", "was", "wasn't", "way", "we",
    "we'd", "we'll", "we're", "we've", "welcome", "well", "went", "were", "weren't", "what", "what's",
    "whatever", "when", "whence", "whenever", "where", "where's", "whereafter", "whereas", "whereby",
    "wherein", "whereupon", "wherever", "whether", "which", "while", "whither", "who", "who's",
    "whoever", "whole", "whom", "whose", "why", "will", "willing", "wish", "with", "within",
    "without", "won't", "wonder", "would", "would", "wouldn't", "x", "y", "yes", "yet", "you",
    "you'd", "you'll", "you're", "you've", "your", "yours", "yourself", "yourselves", "z", "zero"
            };

    static {
        for (int count = 0; count < stopWords.length; count++) {


            String stopWord = stopWords[count];
            if (stopWord != null) {
                stopMap.put(stopWord.trim(), count);

            }
        }

    }


}
```

------------- Total code 2250 lines -------------

5. A Sample

```
/* This piece of code does everything that a search program does. It create
indexes, removes stop words, performs stemming and run search and return a
sorted list of documents based on keyword.

Repository arep; /*Program starts with declarations */
Query aQyery;
int counter;
string q;
string myQuery;
string documentURI;
string documentData;
DocumentList docList;
Document currentDocument;

Main
{
/* an assignment expression */
q = "assignment";

arep = makerepository (”.\docs” ,”*.html,*.txt,*” ,”cosine”);

/* this initiates the repository and initialize directory location,
file patterns and method of computation */

processrepository (arep ,"paice" ,"true" ,"arep.rep");

/* this command processes repositories and perform statistical measures that are
required for searching */

myquery = "factor";

docList = performquery (aQyery, arep, myQuery ,”search.query”);
/* this will search factor in repository and return documents in docList */
/* for loop flavor that runs on document collections */
for each currentDocument in docList
        {
                documentURI = getDocumentURI (currentDocument);
                documentData = getDocumentData (currentDocument);
                print (documentURI);
                print (documentData);
        }
}
```

6. Builtin functions

getDocumentURI document;

This method returns the URI for the document passed as parameter.

getDocumentData document;

This method returns the data for the document passed as parameter.

Print string;

This method prints the string to console.

operation makerepository (string location, string include, string type) returns repository;

This method instantiates the repository.

operation performquery (repository arep, string queryString, string store) returns DocumentList;

This method executes the query on repository and stores result in file passed in store (name value) and return documentList in reference (name value).

operation processrepository (string location, string stemmer , string overwrite, string filename);

This method computes statistical data on all the document that are contained in repository and store the result for later use. This data is then used by performquery operation at the time of search. Most likely only implementation of tf-idf using cosine similarity would be implemented in term project.