CS 421 Final Project

Group 4

Micah Carver

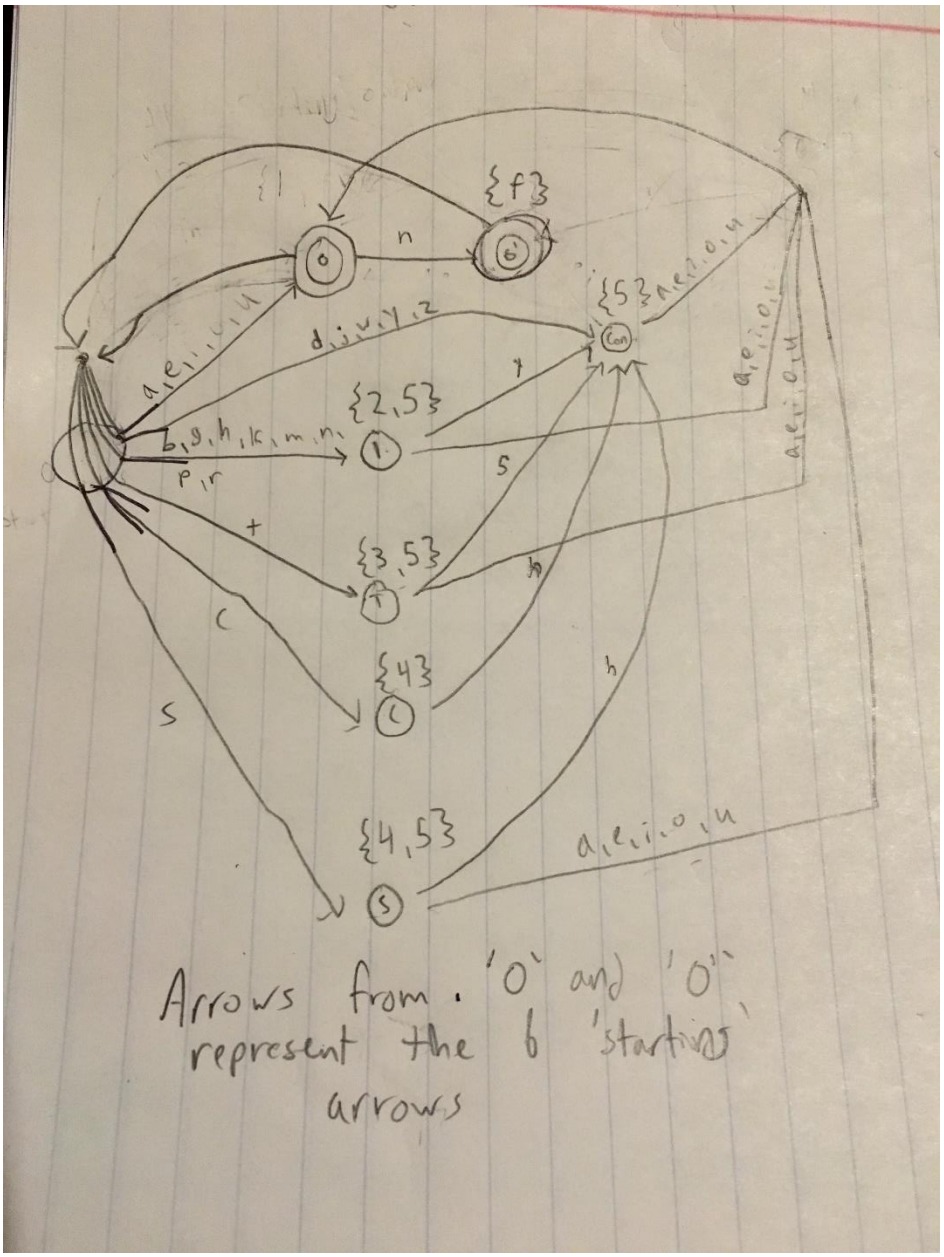
Gabriel Hunt

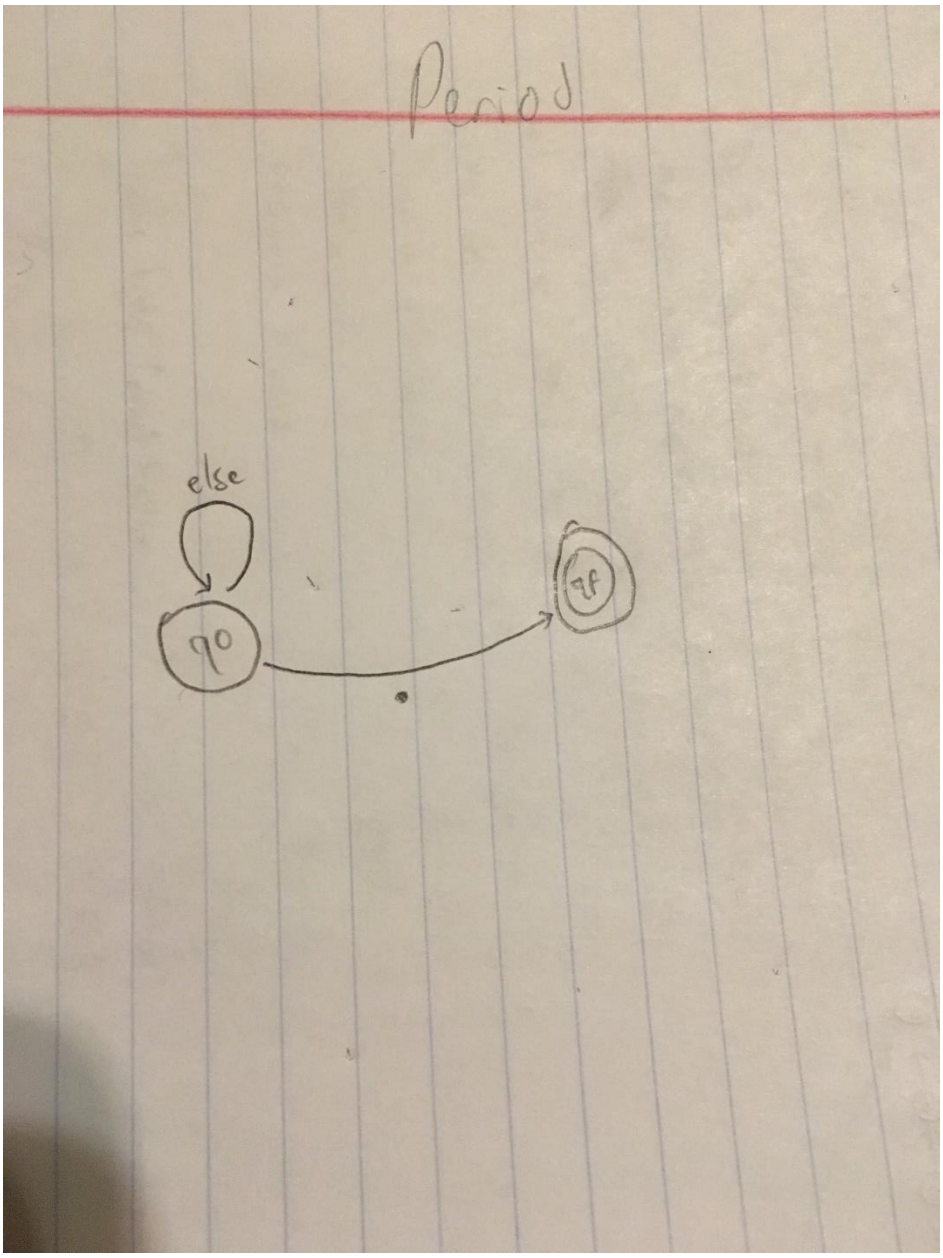
Andrew Hawn

State of the Program:

As far as I can tell, the program is working perfectly. We did none of the extra credit, but the required functions seem to be working flawlessly. We completed all parts of the assignment, and have encountered no bugs (that we know of) in our final implementation. If we could change anything, we would have implemented the extra credit to allow the disabling of traces to make the output much easier to read (especially for the translator portion).

1. DFA


{f}
n
{5}
a,e,i,o,u
d,j,v,y,z
a,e,i,o,u
{2,5}
b,g,h,k,m,n
p,r
t
{3,5}
c
{4}
s
{4,5}
a,e,i,o,u
Arrows from '0' and '0'' represent the 6 'starting' arrows



2. Scanner Code

```
//============================
=========================
                                // File scanner.cpp written by: Group Number: 4
                                // Done by: Micah McCarver
                                //=====================================================


                                #include<iostream>
                                #include<fstream>
```

```
#include<string>
#include<cctype>
#include<cstdlib>
using namespace std;


// MYTOKEN DFA to be replaced by the WORD DFA
// RE:   (((b|g|h|k|m|n|r) y^? (a|e|i|o|u|I|E) n^?) |
((a|e|i|o|u|I|E) n^?) | ((d|j|w|y|z) (a|e|i|o|u|I|E) n^?)
| ((t s^? (a|e|i|o|u|I|E) n^?) |  (c h (a|e|i|o|u|I|E)
n^?) | (s h^? (a|e|i|o|u|I|E) n^?)))*
//Using '?' to denote 0 or 1 occurences
//DFA Done By: Andrew Hawn
bool word(string s)
{
  int state = 0; //declarations
  int charpos = 0;

  while (s[charpos] != '\0') //charpos of letters
    {
      if(state == 0 && (s[charpos] == 'b' ||s[charpos] ==
'g' || s[charpos] == 'h' || s[charpos] == 'k' ||
s[charpos] == 'm' || s[charpos] == 'n' || s[charpos] ==
'r'))
       state = 1;
      else
       if (state == 0 && (s[charpos] == 'a' || s[charpos]
== 'e' ||s[charpos] == 'i' || s[charpos] == 'o' ||
s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] ==
'E'))
         state = 2;
       else
         if (state == 0 && (s[charpos] == 'd' ||
s[charpos] == 'j' || s[charpos] == 'w' || s[charpos] ==
'y' || s[charpos] == 'z'))
           state = 3;
         else
           if (state == 0 && s[charpos] == 's')
             state = 4;
           else
             if (state == 0 && s[charpos] == 'c')
              state = 5;
             else
```

```
                if (state == 0 && s[charpos] == 't')
                  state = 6;
                else
                  if (state == 1 && (s[charpos] == 'a' ||
s[charpos] == 'e' ||s[charpos] == 'i' || s[charpos] ==
'o' || s[charpos] == 'u' || s[charpos] == 'I' ||
s[charpos] == 'E'))
                      state = 2;
                  else
                    if (state == 1 && s[charpos] == 'y')
                      state = 3;
                    else
                      if (state == 2 && s[charpos] == 'n')
                       state = 0;
                      else
                       if (state == 2 && (s[charpos] == 'b'
||s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] ==
'k' || s[charpos] == 'm' || s[charpos] == 'n' ||
s[charpos] == 'r'))
                          state = 1;
                       else
                         if (state == 2 && (s[charpos] ==
'a' || s[charpos] == 'e' ||s[charpos] == 'i' ||
s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] ==
'I' || s[charpos] == 'E'))
                            state = 2;
                         else
                           if (state == 2 && (s[charpos] ==
'd' || s[charpos] == 'j' || s[charpos] == 'w' ||
s[charpos] == 'y' || s[charpos] == 'z'))
                              state = 3;
                            else
                              if (state == 2 && s[charpos]
== 's')
                                state = 4;
                              else
                               if (state == 2 && s[charpos]
== 'c')
                                 state = 5;
                               else
                                 if (state == 2 &&
s[charpos] == 't')
                                   state = 6;
```

```
                                else
                                  if (state == 3 &&
(s[charpos] == 'a' || s[charpos] == 'e' ||s[charpos] ==
'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
                                    state = 2;
                                  else
                                    if (state == 4 &&
(s[charpos] == 'a' || s[charpos] == 'e' ||s[charpos] ==
'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
                                     state = 2;
                                    else
                                     if (state == 4 &&
s[charpos] == 'h')
                                       state = 3;
                                     else
                                       if (state == 5 &&
s[charpos] == 'h')
                                         state = 3;
                                       else
                                         if (state == 6 &&
(s[charpos] == 'a' || s[charpos] == 'e' ||s[charpos] ==
'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
                                           state = 2;
                                         else
                                           if (state == 6
&& s[charpos] == 's')
                                             state = 3;
                                           else
                                            return(false);

      charpos++;
    }//end of while

  // where did I end up????
  if (state == 2 || state == 0) return(true);  //the
final state is where one ends up
  else return(false);  //if not returb false
}
```

```
//=====================================================
==============================================
//Add the PERIOD DFA here
//RE: (a|...|z)^* .
bool period(string s)
{
  int state = 0;
  int charpos = 0;

  while (s[charpos] != '\0') //implement the period dfa
    {
      if(state == 0 && s[charpos] == '.')
       state = 1;
      else
       return (false);

      charpos++;
    }

  if(state == 1) return (true);
  else return(false);
}

//=====================================================
==============================================

//Word Bank Done By: Gabriel

// Update the tokentype to be WORD1, WORD2, PERIOD,
ERROR, etc.
//these are the token types
enum tokentype {ERROR, WORD1, WORD2, PERIOD, VERB,
VERBNEG, VERBPAST, VERBPASTNEG,
               IS, WAS, OBJECT, SUBJECT, DESTINATION,
PRONOUN, CONNECTOR, EOFM};

//these are the token names
//string tokenName[30] = { }; for the display names of
tokens
string tokenName[30] = { "ERROR", "WORD1", "WORD2",
"PERIOD", "VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG",
                        "IS", "WAS", "OBJECT", "SUBJECT",
"DESTINATION", "PRONOUN", "CONNECTOR", "EOFM"};
```

```
//limit of amt of reserved words
const int amtOfWords = 19;

//setting up tble with reserved words and amt of words
string reservedwords[amtOfWords] =
  {  "masu", "masen", "mashita", "masendeshita", "desu",
"deshita", "o", "wa", "ni",
     "watashi", "anata", "kare", "konojo", "sore",
"mata", "soshite", "shikashi", "dakura", "eofm" };

string wordType[amtOfWords] =
  {  "VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS",
"WAS", "OBJECT", "SUBJECT", "DESTINATION",
     "PRONOUN", "PRONOUN", "PRONOUN", "PRONOUN",
"PRONOUN", "CONNECTOR", "CONNECTOR", "CONNECTOR",
     "CONNECTOR", "EOFM" };

ifstream fin;  //fin for file

//=====================================================
==============================================


//Done By: Micah Carver
//matching the reserved words and returning its type
void matchReserved(tokentype &a, string w)
{
  for(int i = 0; i<amtOfWords; i++)
    {
      if(w == reservedwords[i])
       {
         string type = wordType[i];
         for(int x = 0; x<16; x++)
           {
             if(type == tokenName[x])
              {
                a = static_cast<tokentype>(x); //static
cast x
                return;
              }
           }
       }
```

```
    }
}

//=================================================
=============================================

//Done by: All
// Scanner processes only one word each time it is called
//gives back token and word itself
void scanner(tokentype& a, string& w)
{

  //fin declared above now reading in
  fin >> w;
  bool result = true;

  /*Calling the token functions one after another (if-
then-else)
     And generate a lexical error message if both DFAs
failed.
     Let the token_type be ERROR in that case.*/

  if(w == "eofm")
    {
      exit(1);
      //exit if reach need be
    }
  else
    {
      result = period(w);
      if(result) //if result is true
        {
         a = PERIOD;
         return;
        }
      else
        {
         result = word(w);
         if(!result)
            {
              a = ERROR;
              cout << "Lexical error: " << w << " is not a
valid token" << endl;
```

```
            }
          else
            {
             bool test = isupper(w[w.size()-1]);
             if(test)
                {
                 a = WORD2;
                }
             else
                {
                 /*Making sure WORDs are checked against
the reservedwords list
                   If not reserved, token_type is WORD1 or
WORD2.*/
                 a = WORD1;
                 matchReserved(a, w);
                }
            }
        }
    }

}

//=================================================
=============================================

// The temporary test driver to just call the scanner
repeatedly
// This will go away after this assignment
// DO NOT CHANGE THIS!!!!!!
// Done by:  Rika
int main()
{
  tokentype thetype;
  string theword;
  string filename;


  cout << "Enter the input file name: ";
  cin >> filename;

  fin.open(filename.c_str());
```

```
  while (true)
    {
      scanner(thetype, theword);  // call the scanner

      cout << "Type is:" << tokenName[thetype] << endl;
      cout << "Word is:" << theword << endl;

    }

  fin.close();

}// end
```

3. Scanner Results

a. Test1

```
[hawn001@empress ScannerFiles]$ g+
[hawn001@empress ScannerFiles]$ ./
Enter the input file name: scanner
Type is:PRONOUN
Word is:watashi
Type is:SUBJECT
Word is:wa
Type is:WORD1
Word is:rika
Type is:IS
Word is:desu
Type is:PERIOD
Word is:.
Type is:PRONOUN
Word is:watashi
Type is:SUBJECT
Word is:wa
Type is:WORD1
Word is:sensei
Type is:IS
Word is:desu
Type is:PERIOD
Word is:.
Type is:PRONOUN
Word is:watashi
Type is:SUBJECT
Word is:wa
Type is:WORD1
Word is:ryouri
Type is:OBJECT
Word is:o
Type is:WORD2
Word is:yarI
Type is:VERB
Word is:masu
Type is:PERIOD
Word is:.
Type is:PRONOUN
Word is:watashi
Type is:SUBJECT
Word is:wa
Type is:WORD1
Word is:gohan
Type is:OBJECT
Word is:o
Type is:WORD1
Word is:seito
Type is:DESTINATION
Word is:ni
Type is:WORD2
Word is:agE
Type is:VERBPAST
Word is:mashita
Type is:PERIOD
Word is:.
Type is:CONNECTOR
Word is:shikashi
Type is:WORD1
Word is:seito
Type is:SUBJECT
Word is:wa
Type is:WORD2
Word is:nakI
Type is:VERBPAST
Word is:mashita
Type is:PERIOD
Word is:.
```

b. Test 2

```
[hawn001@empress ScannerFiles]$ g++ scanner.cpp
[hawn001@empress ScannerFiles]$ ./a.out
Enter the input file name: scannertest2
Type is:WORD1
Word is:daigaku
Lexical error: college is not a valid token
Type is:ERROR
Word is:college
Type is:WORD1
Word is:kurasu
Lexical error: class is not a valid token
Type is:ERROR
Word is:class
Type is:WORD1
Word is:hon
Lexical error: book is not a valid token
Type is:ERROR
Word is:book
Type is:WORD1
Word is:tesuto
Lexical error: test is not a valid token
Type is:ERROR
Word is:test
Type is:WORD1
Word is:ie
Lexical error: home* is not a valid token
Type is:ERROR
Word is:home*
Type is:WORD1
Word is:isu
Lexical error: chair is not a valid token
Type is:ERROR
Word is:chair
Type is:WORD1
Word is:seito
Lexical error: student is not a valid token
Type is:ERROR
Word is:student
Type is:WORD1
Word is:sensei
Lexical error: teacher is not a valid token
Type is:ERROR
Word is:teacher
Type is:WORD1
Word is:tomodachi
Lexical error: friend is not a valid token
Type is:ERROR
Word is:friend
Type is:WORD1
Word is:jidoosha
Lexical error: car is not a valid token
Type is:ERROR
Word is:car
Type is:WORD1
Word is:gyuunyuu
Lexical error: milk is not a valid token
Type is:ERROR
Word is:milk
Type is:WORD1
Word is:sukiyaki
Lexical error: tenpura is not a valid token
Type is:ERROR
Word is:tenpura
Type is:WORD1
Word is:sushi
Type is:WORD1
Word is:biiru
Lexical error: beer is not a valid token
Type is:ERROR
Word is:beer
Type is:WORD1
Word is:sake
Type is:WORD1
Word is:tokyo
Type is:WORD1
Word is:kyuushuu
Lexical error: Osaka is not a valid token
Type is:ERROR
Word is:Osaka
Type is:WORD1
Word is:choucho
Lexical error: butterfly is not a valid token
Type is:ERROR
Word is:butterfly
Type is:WORD1
Word is:an
Type is:WORD1
Word is:idea
Type is:WORD1
Word is:yasashii
Lexical error: easy is not a valid token
Type is:ERROR
Word is:easy
Type is:WORD1
Word is:muzukashii
Lexical error: difficult is not a valid token
Type is:ERROR
Word is:difficult
Type is:WORD1
Word is:ureshii
Lexical error: pleased is not a valid token
Type is:ERROR
Word is:pleased
Type is:WORD1
Word is:shiawase
Lexical error: happy is not a valid token
Type is:ERROR
Word is:happy
Type is:WORD1
Word is:kanashii
Lexical error: sad is not a valid token
Type is:ERROR
Word is:sad
Type is:WORD1
Word is:omoi
Lexical error: heavy is not a valid token
Type is:ERROR
Word is:heavy
Type is:WORD1
Word is:oishii
Lexical error: delicious is not a valid token
```

```
Type is:ERROR
Word is:delicious
Type is:WORD1
Word is:tennen
Lexical error: natural is not a valid token
Type is:ERROR
Word is:natural
Type is:WORD2
Word is:nakI
Lexical error: cry is not a valid token
Type is:ERROR
Word is:cry
Type is:WORD2
Word is:ikI
Lexical error: go* is not a valid token
Type is:ERROR
Word is:go*
Type is:WORD2
Word is:tabE
Lexical error: eat is not a valid token
Type is:ERROR
Word is:eat
Type is:WORD2
Word is:ukE
Lexical error: take* is not a valid token
Type is:ERROR
Word is:take*
Type is:WORD2
Word is:kakI
Lexical error: write is not a valid token
Type is:ERROR
Word is:write
Type is:WORD2
Word is:yomI
Lexical error: read is not a valid token
Type is:ERROR
Word is:read
Type is:WORD2
Word is:nomI
Lexical error: drink is not a valid token
Type is:ERROR
Word is:drink
Type is:WORD2
Word is:agE
Lexical error: give is not a valid token
Type is:ERROR
Word is:give
Type is:WORD2
Word is:moraI
Lexical error: receive is not a valid token
Type is:ERROR
Word is:receive
Type is:WORD2
Word is:butsI
Lexical error: hit is not a valid token
Type is:ERROR
Word is:hit
Type is:WORD2
Word is:kerI
Lexical error: kick is not a valid token
Type is:ERROR
```

```
Word is:kick
Type is:WORD2
Word is:shaberI
Lexical error: talk is not a valid token
Type is:ERROR
Word is:talk
```

4. Factored Rules (Screencap of the email I sent you)

1 <s> ::= [CONNECTOR] <noun> SUBJECT
<after subject>

2 <after subject> ::= <verb> <tense> PERIOD |
<noun> <after noun>

3 <after noun> ::= <be> PERIOD |
DESTINATION <verb> <tense> PERIOD|
OBJECT <after object>

4 <after object> ::= <verb> <tense> PERIOD |
<noun> DESTINATION <verb> <tense> PERIOD

5 <noun> ::= WORD1 | PRONOUN

6 <verb> ::= WORD2

7 <be> ::= IS | WAS

8 <tense> ::= VERBPAST | VERBPASTNEG |
VERB | VERBNEG

5. Parser Code (Copy of all our code, includes scanner)

```
#include<iost
ream>
                #include<fstream>
                #include<string>
                #include<vector>
                #include<cstdlib>
                #include<cctype>
                #include<cstdio>
                #include<time.h>
                #include<algorithm>
                #include<iterator>
                using namespace std;

                // INSTRUCTION:  copy and edit your parser.cpp to create this file.
                // cp ../ParserFiles/parser.cpp .
                // Complete all ** parts.
                // -----------------------------------------------------
```

```
//================================================
// File translator.cpp written by Group Number: * Group 4 *
//================================================

// ----- Changes to the parser.cpp --------------------

// ** Declare dictionary that will hold the content of lexicon.txt
// Make sure it is easy and fast to look up the translation
// Do not change the format or content of lexicon.txt

// ** Additions to parser.cpp here:
//    getEword - using the current lexeme, look up the English word
//               in the Lexicon if it is there -- save the result
//               in saved_E_word
//    gen(line_type) - using the line type,
//                     sends a line of an IR to translated.txt
//                     (saved_E_word or saved_token is used)

// ** Be sure to put the name of the programmer above each function

// ** Be sure to put the corresponding grammar
//    rule with semantic routines
//    above each non-terminal function
// ----------------------------------------

// Parser.cpp:

enum tokentype {
       ERROR, WORD1, WORD2, PERIOD, VERB, VERBNEG, VERBPAST, VERBPASTNEG,
       IS, WAS, OBJECT, SUBJECT, DESTINATION, PRONOUN, CONNECTOR, EOFM
};

//these are the token names
//string tokenName[30] = { }; for the display names of tokens
string tokenName[16] = { "ERROR", "WORD1", "WORD2", "PERIOD", "VERB",
"VERBNEG", "VERBPAST", "VERBPASTNEG",
                                          "IS", "WAS", "OBJECT",
"SUBJECT", "DESTINATION", "PRONOUN", "CONNECTOR", "EOFM" };

//limit of amt of reserved words
const int amtOfWords = 19;
```

```
//setting up tble with reserved words and amt of words
string reservedwords[amtOfWords] =
{ "masu", "masen", "mashita", "masendeshita", "desu", "deshita", "o", "wa",
"ni",
   "watashi", "anata", "kare", "konojo", "sore", "mata", "soshite",
"shikashi", "dakara", "eofm" };

string wordType[amtOfWords] =
{ "VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT",
"SUBJECT", "DESTINATION",
   "PRONOUN", "PRONOUN", "PRONOUN", "PRONOUN", "PRONOUN", "CONNECTOR",
"CONNECTOR", "CONNECTOR",
   "CONNECTOR", "EOFM" };

ifstream fin;  //fin for file


void story();
void sentence();
void afterSubject();
void afterNoun();
void afterObject();
void noun();
void verb();
void be();
void tense();

void syntax_error1(tokentype);
void syntax_error2(string);

tokentype next_token();
bool match(tokentype);

tokentype saved_token;
string saved_lexeme;
string filename;
bool tokenExists = false; //no starting token

void scanner(tokentype&, string&);

string getTranslation(string);
bool fillDictionary();
void getEword();
```

```
void gen(string);
string saved_E_word;


//** require no other input files!
//** syntax error EC requires producing errors.text of messages




//-----------------------PARSER FUNCTIONS-------------------------
//Done by: Micah
//<story> -> <sentence> { <sentence> }
void story()
{
	cout << "Processing <story>" << endl;
	cout << endl;

	sentence();
	while (next_token() != EOFM)
	{
		sentence();
	}
}
//Done by: All
//<sentence> -> [CONNECTOR] #getEword# #gen# <noun> #getEword# SUBJECT
#gen# <afterSubject>
void sentence()
{
	cout << "Processing <sentence>" << endl;

	if (next_token() == CONNECTOR)
	{
		match(CONNECTOR);
		getEword();
		gen("CONNECTOR");
	}

	switch (next_token())
	{
	case WORD1: case PRONOUN:
		noun();
		getEword();
```

```
                match(SUBJECT);
                gen("ACTOR");
                afterSubject();
                break;
        default:
                syntax_error2("<sentence>");
                return;
        }
}
//Done by: All
//<afterSubject> -> <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun>
#getEword# <afterNoun>
void afterSubject()
{
        cout << "Processing <afterSubject>" << endl;

        switch (next_token())
        {
        case WORD2:
                verb();
                getEword();
                gen("ACTION");
                tense();
                gen("TENSE");
                match(PERIOD);
                break;
        case WORD1: case PRONOUN:
                noun();
                getEword();
                afterNoun();
                break;
        default:
                syntax_error2("<afterSubject>");
                return;
        }
}
//Done by: All
//<afterNoun> -> <be> #getEword# #gen# #gen# PERIOD | DESTINATION #gen#
<verb> #getEword# #gen# <tense> #gen# PERIOD | OBJECT #gen# <afterObject>
void afterNoun()
{
        cout << "Processing <afterNoun>" << endl;
```

```
	switch (next_token())
	{
	case IS: case WAS:
		be();
		getEword();
		gen("DESCRIPTION");
		gen("TENSE");
		match(PERIOD);
		break;
	case DESTINATION:
		match(DESTINATION);
		gen("TO");
		verb();
		getEword();
		gen("ACTION");
		tense();
		gen("TENSE");
		match(PERIOD);
		break;
	case OBJECT:
		match(OBJECT);
		gen("OBJECT");
		afterObject();
		break;
	default:
		syntax_error2("<afterNoun>");
		return;
	}
}
//Done by: All
//<afterObject> -> <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun>
#getEword# DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen# PERIOD
void afterObject()
{
	cout << "Processing <afterObject>" << endl;

	switch (next_token())
	{
	case WORD2:
		verb();
		getEword();
		gen("ACTION");
		tense();
```

```
                gen("TENSE");
                match(PERIOD);
                break;
        case WORD1: case PRONOUN:
                noun();
                getEword();
                match(DESTINATION);
                gen("TO");
                verb();
                getEword();
                gen("ACTION");
                tense();
                gen("TENSE");
                match(PERIOD);
                break;
        default:
                syntax_error2("<afterObject>");
                return;
        }
}
//Done by: Andrew
//<noun> -> WORD1 | PRONOUN
void noun()
{
        cout << "Processing <noun>" << endl;

        switch (next_token())
        {
        case WORD1:
                match(WORD1);
                break;
        case PRONOUN:
                match(PRONOUN);
                break;
        default:
                syntax_error2("<noun>");
                return;
        }
}
//Done by: Andrew
//<verb> -> WORD2
void verb()
{
```

```
        cout << "Processing <verb>" << endl;

        switch (next_token())
        {
        case WORD2:
                match(WORD2);
                break;
        default:
                syntax_error2("<verb>");
                return;
        }
}
//Done by: Andrew
//<be> -> IS | WAS
void be()
{
        cout << "Processing <be>" << endl;

        switch (next_token())
        {
        case IS:
                match(IS);
                break;
        case WAS:
                match(WAS);
                break;
        default:
                syntax_error2("<be>");
                return;
        }
}
//Done by: Andrew
//<tense> -> VERBPAST | VERBPASTNEG | VERB | VERBNEG
void tense()
{
        cout << "Processing <tense>" << endl;

        switch (next_token())
        {
        case VERBPAST:
                match(VERBPAST);
                break;
        case VERBPASTNEG:
```

```
                match(VERBPASTNEG);
                break;
        case VERB:
                match(VERB);
                break;
        case VERBNEG:
                match(VERBNEG);
                break;
        default:
                syntax_error2("<tense>");
                return;
        }
}
//Done by: Gabriel
bool match(tokentype thetype)
{
        if (next_token() != thetype)
        {
                syntax_error1(thetype);
        }
        else
        {
                cout << "Matched " << tokenName[thetype] << endl;
                tokenExists = false;
                return true;
        }
}
//Done by: Gabriel
tokentype next_token()
{
        if (tokenExists == false)
        {
                scanner(saved_token, saved_lexeme);
                tokenExists = true;
        }
        return saved_token;
}
//Done by: Micah
void syntax_error1(tokentype thetype)
{
        cout << "SYNTAX ERROR: expected " << tokenName[thetype] << " but
found " << saved_lexeme << "." << endl;
        exit(1);
```

```
}
//Done by: Micah
void syntax_error2(string parserFunction)
{
	cout << "SYNTAX ERROR: unexpected " << saved_lexeme << " found in "
<< parserFunction << "." << endl;
	exit(1);
}
//----------------------SCANNER FUNCTIONS----------------------------

bool word(string s)
{
	int state = 0; //declarations
	int charpos = 0;

	while (s[charpos] != '\0') //charpos of letters
	{
		if (state == 0 && (s[charpos] == 'b' || s[charpos] == 'g' ||
s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos]
== 'n' || s[charpos] == 'r'))
			state = 1;
		else
			if (state == 0 && (s[charpos] == 'a' || s[charpos] ==
'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
				state = 2;
			else
				if (state == 0 && (s[charpos] == 'd' ||
s[charpos] == 'j' || s[charpos] == 'w' || s[charpos] == 'y' || s[charpos]
== 'z'))
					state = 3;
				else
					if (state == 0 && s[charpos] == 's')
						state = 4;
					else
						if (state == 0 && s[charpos] ==
'c')
							state = 5;
						else
							if (state == 0 &&
s[charpos] == 't')
								state = 6;
							else
```

```
                                                        if (state == 1 &&
(s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos]
== 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                                                state =
2;
                                                        else
                                                                if (state
== 1 && s[charpos] == 'y')

        state = 3;
                                                                else
                                                                        if
(state == 2 && s[charpos] == 'n')

        state = 0;

        else

        if (state == 2 && (s[charpos] == 'b' || s[charpos] == 'g' ||
s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos]
== 'n' || s[charpos] == 'r'))

                state = 1;

        else

                if (state == 2 && (s[charpos] == 'a' || s[charpos] == 'e' ||
s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos]
== 'I' || s[charpos] == 'E'))

                        state = 2;

                else

                        if (state == 2 && (s[charpos] == 'd' || s[charpos] ==
'j' || s[charpos] == 'w' || s[charpos] == 'y' || s[charpos] == 'z'))

                                state = 3;

                        else

                                if (state == 2 && s[charpos] == 's')
```

```
                                        state = 4;

                                    else

                                        if (state == 2 && s[charpos] == 'c')

                                            state = 5;

                                        else

                                            if (state == 2 && s[charpos] ==
't')

                                                state = 6;

                                            else

                                                if (state == 3 &&
(s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos]
== 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))

                                                    state = 2;

                                                else

                                                    if (state == 4 &&
(s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos]
== 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))

                                                        state =
2;

                                                    else

                                                        if (state
== 4 && s[charpos] == 'h')


                state = 3;

                                                        else
```

```
                                                                    if
(state == 5 && s[charpos] == 'h')

        state = 3;

        else

        if (state == 6 && (s[charpos] == 'a' || s[charpos] == 'e' ||
s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos]
== 'I' || s[charpos] == 'E'))

                state = 2;

        else

                if (state == 6 && s[charpos] == 's')

                        state = 3;

                else

                        return(false);

                charpos++;
        }//end of while

  // where did I end up????
        if (state == 2 || state == 0) return(true);  //the final state is
where one ends up
        else return(false);  //if not returb false
}
```

```
//=============================================================================
============================
//Add the PERIOD DFA here
//RE: implemented with while and if/else
bool period(string s)
{
	int state = 0;
	int charpos = 0;

	while (s[charpos] != '\0') //implement the period dfa
	{
		if (state == 0 && s[charpos] == '.')
			state = 1;
		else
			return (false);

		charpos++;
	}

	if (state == 1) return (true);
	else return(false);
}

//=============================================================================
============================

//matching the reserved words and returning its type
void matchReserved(tokentype &a, string w)
{
	for (int i = 0; i < amtOfWords; i++)
	{
		if (w == reservedwords[i])
		{
			string type = wordType[i];
			for (int x = 0; x < 16; x++)
			{
				if (wordType[i] == tokenName[x])
				{
					a = static_cast<tokentype>(x); //static
cast x

					return;
				}
			}
```

```
                        return;
                }
        }
        return;
}

//=========================================================================
============================

// Scanner processes only one word each time it is called
//gives back token and word itself
void scanner(tokentype& a, string& w)
{

        //fin declared above now reading in
        fin >> w;
        cout << "Scanner called using word: " << w << endl;
        bool result = true;

        /*Calling the token functions one after another (if-then-else)
         And generate a lexical error message if both DFAs failed.
         Let the token_type be ERROR in that case.*/

        if (w == "eofm")
        {
                //exit if reach need be
                exit(0);
        }
        else
        {
                result = period(w);
                if (result) //if result is true
                {
                        a = PERIOD;
                        return;
                }
                else
                {
                        result = word(w);
                        if (!result)
                        {
                                a = ERROR;
```

```
                        cout << "Lexical error: " << w << " is not a
valid token" << endl;
                }
                else
                {
                        bool test = isupper(w[w.size() - 1]);
                        if (test)
                        {
                                a = WORD2;
                        }
                        else
                        {
                                /*Making sure WORDs are checked against
the reservedwords list
                                 If not reserved, token_type is WORD1
or WORD2.*/
                                a = WORD1;
                                matchReserved(a, w);
                        }
                }
        }
}


// ----------------------------------------
// End of Parser.cpp


// Start of Translator additions
// ----------------------------------------


// GLOBALS (Gabriel -- added 12/7/18):

/* Note for dictionary: japasnese words will be listed first in the
dictionary,
              the next imediate element after will be the english word*/
vector<string> dictionary;
ifstream lexIn;
ofstream translated;
```

```
// METHODS (Gabriel -- added 12/7/18)
string getTranslation(string japWord)
{
	string englishWord;
	string temp;
	for (int i = 0; i < dictionary.size(); i++)
	{
		temp = dictionary[i];
		if (temp == japWord)
		{
			englishWord = dictionary[i + 1];
			return englishWord;
		}
	}
	return NULL;
}

bool fillDictionary()
{
	string japWord, engWord;
	try
	{
		lexIn.open("lexicon.txt");

	//lexIn.open("C:/Users/gabri/Documents/GitHub/CS421_Project/Translat
orFiles/lexicon.txt");
		while (!lexIn.eof())
		{
			lexIn >> japWord;
			lexIn >> engWord;
			dictionary.push_back(japWord);
			dictionary.push_back(engWord);
		}
		lexIn.close();
		return true;
	}

	catch (ifstream::failure e)
	{
		cout << "Problem reading from lexicon.text (see
fillDictionary method)" << endl;
		return false;
```

```
	}
}


//Done By: Micah
void getEword()
{

	string temp;
	string second;

	for (int i = 0; i < dictionary.size(); i++)
	{
		temp = dictionary[i];
		if (temp == saved_lexeme)
		{
			second = dictionary[i + 1];
			saved_E_word = second;
			return;
		}
	}

	saved_E_word = saved_lexeme;


}

//Done By: Andrew
void gen(string theType)
{
	if (theType == "TENSE")
	{
		cout << theType << " to " << tokenName[saved_token] << endl;
		translated << theType << " to " << tokenName[saved_token] <<
endl;
	}
	else
	{
		cout << theType << " to " << saved_E_word << endl;
		translated << theType << " to " << saved_E_word << endl;

	}
}
```

```
// The final test driver to start the translator
// Done by  * Gabriel Hunt *
int main()
{

	// Load the dictionary
	bool fillSuccess = fillDictionary();
	if (fillSuccess)
		cout << "Dictionary filled successfully!" << endl;
	if (!fillSuccess)
	{
		cout << "Problem loading dictionary" << endl;
		return 1;
	}



  //** opens the output file translated.txt

	cout << "Enter the input file name: ";
	cin >> filename;
	fin.open(filename.c_str());

	translated.open("translated.txt");

  //** calls the <story> to start parsing
  story();
  //** closes the input file
  //** closes traslated.txt
  fin.close();
  translated.close();
}// end
```

6. Final Test Results

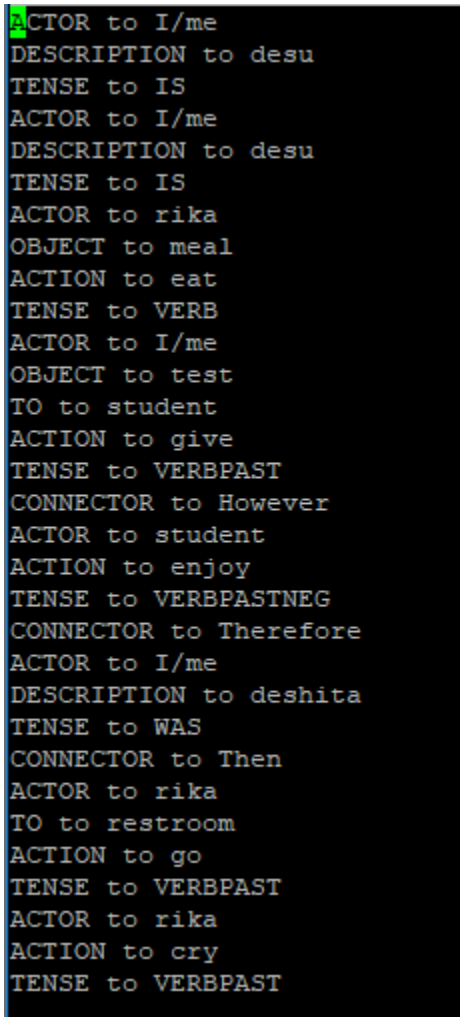
a. Test1

```
[hawn001@empress TranslatorFiles]$ g++ translator.cpp
[hawn001@empress TranslatorFiles]$ ./a.out
Dictionary filled successfully!
Enter the input file name: partCtest1
Processing <story>

Processing <sentence>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
DESCRIPTION to desu
TENSE to IS
Scanner called using word: .
Matched PERIOD
Scanner called using word: watashi
Processing <sentence>
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: sensei
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
DESCRIPTION to desu
TENSE to IS
Scanner called using word: .
Matched PERIOD
Scanner called using word: rika
Processing <sentence>
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
ACTOR to rika
Processing <afterSubject>
Scanner called using word: gohan
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: o
Matched OBJECT
OBJECT to meal
Processing <afterObject>
Scanner called using word: tabE
Processing <verb>
Matched WORD2
ACTION to eat
Processing <tense>
Scanner called using word: masu
Matched VERB
TENSE to VERB
Scanner called using word: .
Matched PERIOD
Scanner called using word: watashi
Processing <sentence>
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: tesuto
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: o
Matched OBJECT
OBJECT to test
Processing <afterObject>
Scanner called using word: seito
Processing <noun>
Matched WORD1
Scanner called using word: ni
Matched DESTINATION
TO to student
Processing <verb>
Scanner called using word: agE
Matched WORD2
ACTION to give
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
TENSE to VERBPAST
Scanner called using word: .
Matched PERIOD
Scanner called using word: shikashi
Processing <sentence>
Matched CONNECTOR
CONNECTOR to However
Scanner called using word: seito
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
ACTOR to student
Processing <afterSubject>
Scanner called using word: yorokobI
Processing <verb>
Matched WORD2
ACTION to enjoy
Processing <tense>
Scanner called using word: masendeshita
Matched VERBPASTNEG
TENSE to VERBPASTNEG
Scanner called using word: .
```

```
Matched PERIOD
Scanner called using word: dakara
Processing <sentence>
Matched CONNECTOR
CONNECTOR to Therefore
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: kanashii
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: deshita
Processing <be>
Matched WAS
DESCRIPTION to deshita
TENSE to WAS
Scanner called using word: .
Matched PERIOD
Scanner called using word: soshite
Processing <sentence>
Matched CONNECTOR
CONNECTOR to Then
Scanner called using word: rika
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
ACTOR to rika
Processing <afterSubject>
Scanner called using word: toire
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: ni
Matched DESTINATION
TO to restroom
Processing <verb>
Scanner called using word: ikI
Matched WORD2
ACTION to go
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
TENSE to VERBPAST
Scanner called using word: .
Matched PERIOD
Scanner called using word: rika
Processing <sentence>
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
ACTOR to rika
Processing <afterSubject>
Scanner called using word: nakI
Processing <verb>
Matched WORD2
ACTION to cry
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
TENSE to VERBPAST
Scanner called using word: .
Matched PERIOD
Scanner called using word: eofm
```

translated.txt



```
ACTOR to I/me
DESCRIPTION to desu
TENSE to IS
ACTOR to I/me
DESCRIPTION to desu
TENSE to IS
ACTOR to rika
OBJECT to meal
ACTION to eat
TENSE to VERB
ACTOR to I/me
OBJECT to test
TO to student
ACTION to give
TENSE to VERBPAST
CONNECTOR to However
ACTOR to student
ACTION to enjoy
TENSE to VERBPASTNEG
CONNECTOR to Therefore
ACTOR to I/me
DESCRIPTION to deshita
TENSE to WAS
CONNECTOR to Then
ACTOR to rika
TO to restroom
ACTION to go
TENSE to VERBPAST
ACTOR to rika
ACTION to cry
TENSE to VERBPAST
```

b. Test2

```
[hawn001@empress TranslatorFiles]$ ./a.out
Dictionary filled successfully!
Enter the input file name: partCtest2
Processing <story>

Processing <sentence>
Scanner called using word: soshite
Matched CONNECTOR
CONNECTOR to Then
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
DESCRIPTION to desu
TENSE to IS
Scanner called using word: ne
SYNTAX ERROR: expected PERIOD but found ne.
```

translated.txt

```
CONNECTOR to Then
ACTOR to I/me
DESCRIPTION to desu
TENSE to IS
```

c. Test3

```
[hawn001@empress TranslatorFiles]$ ./group4project.out
Dictionary filled successfully!
Enter the input file name: partCtest3
Processing <story>

Processing <sentence>
Scanner called using word: dakara
Matched CONNECTOR
CONNECTOR to Therefore
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: de
SYNTAX ERROR: expected SUBJECT but found de.
```

translated.txt

```
CONNECTOR to Therefore
```

d. Test4

```
[hawn001@empress TranslatorFiles]$ ./group4project.out
Dictionary filled successfully!
Enter the input file name: partCtest4
Processing <story>

Processing <sentence>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
ACTOR to I/me
Processing <afterSubject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: mashita
SYNTAX ERROR: unexpected mashita found in <afterNoun>.
```

translated.txt

```
ACTOR to I/me
```

e. Test5

```
[hawn001@empress TranslatorFiles]$ ./group4project.out
Dictionary filled successfully!
Enter the input file name: partCtest5
Processing <story>

Processing <sentence>
Scanner called using word: wa
SYNTAX ERROR: unexpected wa found in <sentence>.
```

translated.txt

(it’s empty)

f. Test6

```
[hawn001@empress TranslatorFiles]$ ./group4project.out
Dictionary filled successfully!
Enter the input file name: partCtest6
Processing <story>

Processing <sentence>
Scanner called using word: apple
Lexical error: apple is not a valid token
SYNTAX ERROR: unexpected apple found in <sentence>.
```

translated.txt

(empty again)