SECTION 0

Final Project

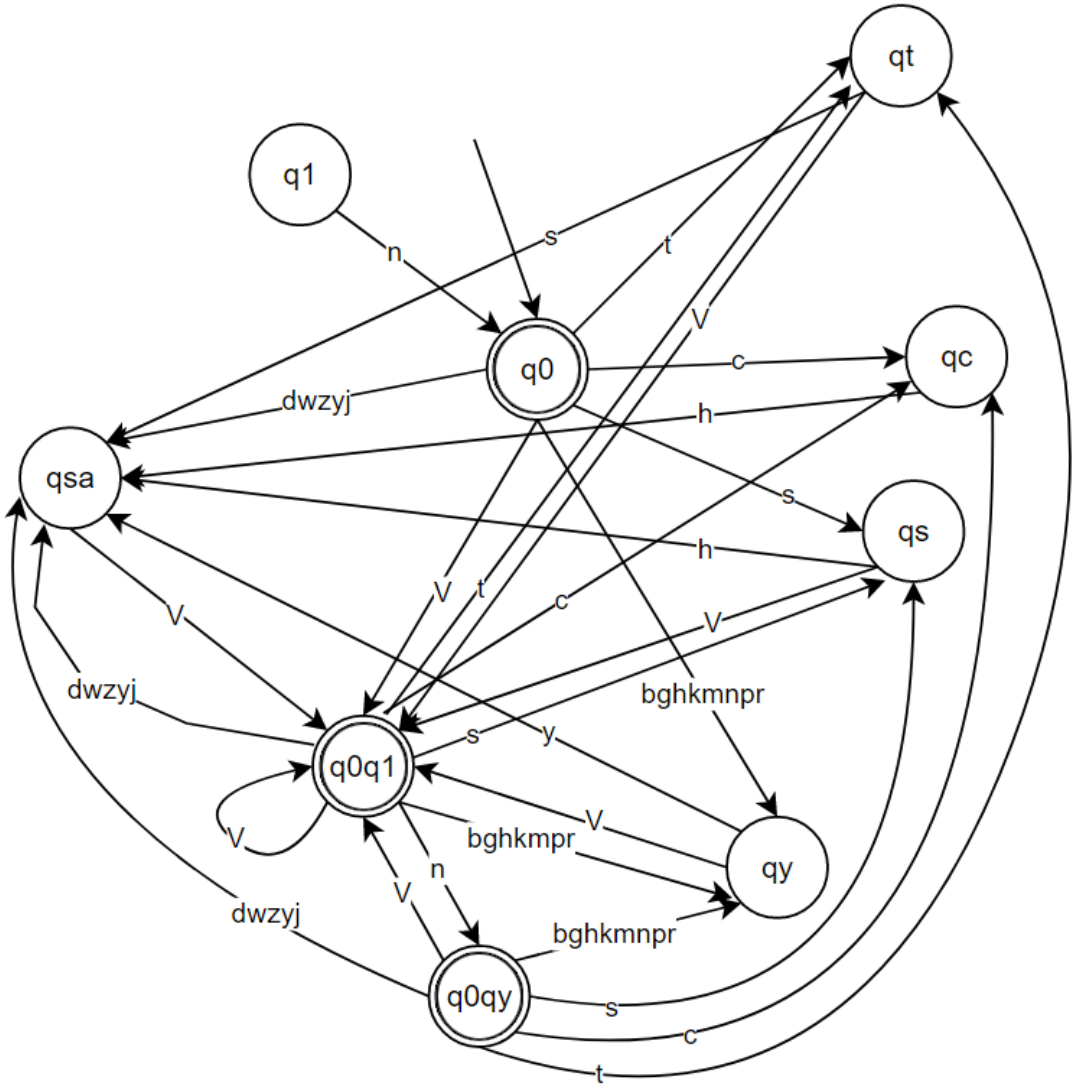
Group 5

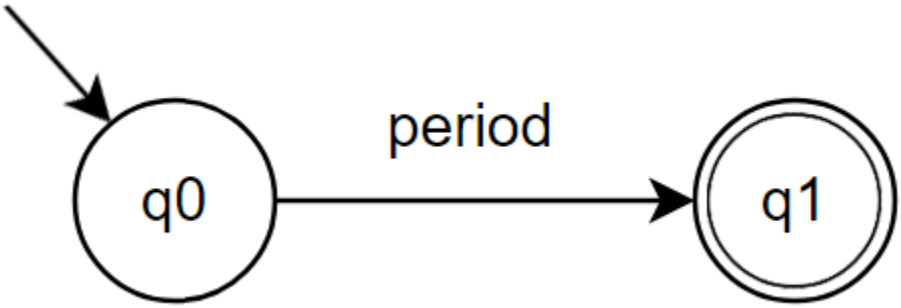
Justin Luzano, Kevin Andersen, David Coats

State of the program:

- Everything is working perfectly
- We have completed everything
- There are no bugs in the code
- We did not implement any of the extra credit features

SECTION 1 - DFA (final version)





SECTION 2 - Scanner.cpp Code

```
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

/* Look for all **'s and complete them */

//=================================================
// File scanner.cpp written by: Group Number: ** 5
//=================================================

// --------- Two DFAs ---------------------------------

// WORD DFA
// Done by: ** Justin
// RE:   ** RE = (vowel | vowel n | consonant vowel | consonant vowel n |
consonant-pair vowel | consonant-pair vowel n)^+
bool word (string s)
{
  int state = 0;
  int charpos = 0;
  // replace the following todo the word dfa  **
  /*
  1 = q0q1
  2 = qc
  3 = qs
  4 = qt
  5 = qy
  6 = qsa
  7 = q0qy
  */

  while (s[charpos] != '\0')
  {
    bool vowels = s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] ==
'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' ||
s[charpos] == 'E';
```

```
    bool bghkmpr = s[charpos] == 'b' || s[charpos] == 'g' || s[charpos] ==
'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' ||
s[charpos] == 'r';
    bool dwzyj = s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] ==
'z' || s[charpos] == 'y' || s[charpos] == 'j';
    /*
    Trs(q0, V) = q0q1
    Trs(q0, t) = qt
    Trs(q0, s) = qs
    Trs(q0, c) = qc
    Trs(q0, dwzyj) = qsa
    Trs(q0, bghkmnpr) = qy
    */
    if (state == 0 && vowels)
        state = 1;
    else if (state == 0 && s[charpos] == 't')
        state = 4;
    else if (state == 0 && s[charpos] == 's')
        state = 3;
    else if (state == 0 && s[charpos] == 'c')
        state = 2;
    else if (state == 0 && dwzyj)
        state = 6;
    else if (state == 0 && (bghkmpr || s[charpos] == 'n'))
        state = 5;

//---------------------------------------------------------------------------
--
    /*
    Trs(qy, V) = q0q1
    Trs(qy, y) = qsa
    */
    else if (state == 5 && vowels)
        state = 1;
    else if (state == 5 && s[charpos] == 'y')
        state = 6;

//---------------------------------------------------------------------------
--
    /*
```

```
    Trs(qt, V) = q0q1
    Trs(qt, s) = qsa
    */
    else if (state == 4 && vowels)
        state = 1;
    else if (state == 4 && s[charpos] == 's')
        state = 6;

//-------------------------------------------------------------------------
--
    /*
    Trs(qs, V) = q0q1
    Trs(qs, h) = qsa
    */
    else if (state == 3 && vowels)
        state = 1;
    else if (state == 3 && s[charpos] == 'h')
        state = 6;

//-------------------------------------------------------------------------
--
    /*
    Trs(qc, h) = qsa
    */
    else if (state == 2 && s[charpos] == 'h')
        state = 6;

//-------------------------------------------------------------------------
--
    /*
    Trs(qsa, V) = q0q1
    */
    else if (state == 6 && vowels)
        state = 1;

//-------------------------------------------------------------------------
--
    /*
    Trs(q0q1, V) = q0q1
    Trs(q0q1, t) = qt
```

```
    Trs(q0q1, s) = qs
    Trs(q0q1, c) = qc
    Trs(q0q1, bghkmpr) = qy
    Trs(q0q1, n) = q0qy
    Trs(q0q1, dwzyj) = qsa
    */
    else if (state == 1 && vowels)
        state = 1;
    else if (state == 1 && s[charpos] == 't')
        state = 4;
    else if (state == 1 && s[charpos] == 's')
        state = 3;
    else if (state == 1 && s[charpos] == 'c')
        state = 2;
    else if (state == 1 && bghkmpr)
        state = 5;
    else if (state == 1 && s[charpos] == 'n')
        state = 7;
    else if (state == 1 && dwzyj)
        state = 6;

//-------------------------------------------------------------------------
--
    /*
    Trs(q0qy, V) = q0q1
    Trs(q0qy, c) = qc
    Trs(q0qy, s) = qs
    Trs(q0qy, t)  = qt
    Trs(q0qy, bghkmnpr) = qy
    Trs(q0qy, dwzyj) = qsa
    */
    else if (state == 7 && vowels)
        state = 1;
    else if (state == 7 && s[charpos] == 'c')
        state = 2;
    else if (state == 7 && s[charpos] == 's')
        state = 3;
    else if (state == 7 && s[charpos] == 't')
        state = 4;
    else if (state == 7 && (bghkmpr || s[charpos] == 'n'))
```

```
        state = 5;
    else if (state == 7 && dwzyj)
        state = 6;

    charpos++;
  }//end of while

  // where did I end up????
  return state == 1 || state == 7;
}

// PERIOD DFA
// Done by: ** Justin and comments by Kevin
bool period(string s)
{
    int state = 0;       // Initialize the state to 0
    int charpos = 0;     // Initialize the character position to 0

    // Loop through each character in the input string
    while (s[charpos] != '\0')
    {
        // Check the current state and character
        if (state == 0 && s[charpos] == '.')
        {
            // If the current state is 0 and the current character is a
period, transition to state 1
            state = 1;
        }
        else
        {
            // If the current state is not 0 or the current character is
not a period, return false
            return false;
        }
        charpos++;      // Increment the character position
    }

    // If the loop completes and the final state is 1, return true;
otherwise, return false
    return state == 1;
```

```
}
// ------ Three  Tables ------------------------------------

// TABLES Done by: ** Justin and Kevin

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

// ** For the display names of tokens - must be in the same order as the
tokentype.
string tokenName[30] = {"WORD1", "WORD2", "PERIOD", "ERROR", "EOFM",
"VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS",
"OBJECT","SUBJECT", "DESTINATION", "PRONOUN", "CONNECTOR"};

// ** Need the reservedwords table to be set up here.
// ** Do not require any file input for this. Hard code the table.
// ** a.out should work without any additional files.
string reservedWords[19][2] =
{
  {"masu", "VERB"},
  {"masen", "VERBNEG"},
  {"mashita", "VERBPAST"},
  {"masendeshita", "VERBPASTNEG"},
  {"desu", "IS"},
  {"deshita", "WAS"},
  {"o", "OBJECT"},
  {"wa", "SUBJECT"},
  {"ni", "DESTINATION"},
  {"watashi", "PRONOUN"},
  {"anata", "PRONOUN"},
  {"kare", "PRONOUN"},
  {"kanojo", "PRONOUN"},
  {"sore", "PRONOUN"},
  {"shikashi", "CONNECTOR"},
  {"dakara", "CONNECTOR"},
  {"mata", "CONNECTOR"},
  {"soshite", "CONNECTOR"},
```

```
  {"eofm", "EOFM"}
};

// ------------ Scanner and Driver -----------------------

ifstream fin;  // global stream for reading from the input file

// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// ** Done by: Justin and Kevin
int scanner(tokentype& tt, string& w)
{
  // ** Grab the next word from the file via fin
  // 1. If it is eofm, return right now.
  fin >> w;
  if (w == "eofm")
    {
      tt = EOFM;
      return 0;              // 1. If it is eofm, return right now.
    }

  /* **
  2. Call the token functions (word and period)
  one after another (if-then-else).
  Generate a lexical error message if both DFAs failed.
  Let the tokentype be ERROR in that case.
  3. If it was a word,
  check against the reservedwords list.
  If not reserved, tokentype is WORD1 or WORD2
  decided based on the last character.
  4. Return the token type & string (pass by reference)
  */

  if (word(w))
{
    bool reserved = false;
    int size = sizeof(reservedWords) / sizeof(*reservedWords);
    for (int i = 0; i < size ; i++)
    {
```

```
        // Checking against the reserved words list if it's a word
        if (w == reservedWords[i][0])
        {
            // If it's a reserved word, get the associated token type
            string temp = reservedWords[i][1];
            int num = sizeof(tokenName) / sizeof(tokenName[0]);
            for (int j = 0; j < num; j++)
            {
                // Check the token type against the list of token names
                if (temp == tokenName[j])
                {
                    // If the token type is found, set the token type and
mark the word as reserved
                    tt = (tokentype)j;
                    reserved = true;
                    break;
                }
            }
            break;
        }
    }

    if (reserved)
      return 0;
    char back = w[w.size() - 1];
    //Tokentype is WORD1 or WORD2 based on the last character
    if (back == 'I' || back == 'E')
        tt = WORD2;
    else if (back == '*')
    {
        tt = ERROR;
        cout << "Lexical error: " << w << " is not a valid token" << endl;
    }
    else
        tt = WORD1;
  }
  //If the last character is a period
  else if (period(w))
      tt = PERIOD;
  else
```

```
  {
      //If both DFAs fail
      tt = ERROR;
      cout << "Lexical error: " << w << " is not a valid token" << endl;
    }
    return 0;
}//the end of scanner

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

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

  fin.open(filename.c_str());

  // the loop continues until eofm is returned.
   while (true)
    {
       scanner(thetype, theword);  // call the scanner which sets
                                   // the arguments
       if (theword == "eofm") break;  // stop now

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

   cout << "End of file is encountered." << endl;
   fin.close();

}// end
```

SECTION 3 - Original Scanner Test Results

```
Script started on Sat 13 May 2023 06:03:33 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/ScannerFiles[?1034h[luzan003@e
mpress ScannerFiles]$ g++ scanner.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/ScannerFiles[luzan003@empress
ScannerFiles]$ ./a.out
Enter the input file name: scannertest1
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:yorokobI

Type is:VERBPASTNEG
Word is:masendeshita

Type is:PERIOD
Word is:.

Type is:CONNECTOR
Word is:dakara

Type is:PRONOUN
Word is:watashi

Type is:SUBJECT
Word is:wa

Type is:WORD1
Word is:kanashii

Type is:WAS
Word is:deshita

Type is:PERIOD
```

```
Word is:.

Type is:CONNECTOR
Word is:soshite

Type is:PRONOUN
Word is:watashi

Type is:SUBJECT
Word is:wa

Type is:WORD1
Word is:toire

Type is:DESTINATION
Word is:ni

Type is:WORD2
Word is:ikI

Type is:VERBPAST
Word is:mashita

Type is:PERIOD
Word is:.

Type is:PRONOUN
Word is:watashi

Type is:SUBJECT
Word is:wa

Type is:WORD2
Word is:nakI

Type is:VERBPAST
Word is:mashita

Type is:PERIOD
Word is:.
```

```
End of file is encountered.
]0;luzan003@empress:~/cs421files/CS421Progs/ScannerFiles[luzan003@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

Type is:WORD1
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

Type is:WORD1
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

Type is:WORD1
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

Type is:WORD1
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

Type is:WORD1
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

End of file is encountered.
]0;luzan003@empress:~/cs421files/CS421Progs/ScannerFiles[luzan003@empress
ScannerFiles]$ exit
exit

Script done on Sat 13 May 2023 06:04:03 PM PDT
```

SECTION 4 - Factored Rules

```
<s> ::= [CONNECTOR #getEword# #gen(CONNECTOR)#] <noun> #getEword# SUBJECT
#gen(ACTOR)# <after subject>
<after subject> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)#
PERIOD | <noun> #getEword# <after noun>
<after noun> ::= <be> #gen(DESCRIPTION)# #gen(TENSE)# PERIOD  |
DESTINATION #gen(TO)# <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)#
PERIOD | OBJECT #gen(OBJECT)# <after object>
<after object> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)#
PERIOD | <noun> #getEword# DESTINATION #gen(TO)# <verb> #getEword#
#gen(ACTION)# <tense> #gen(TENSE)# PERIOD
```

SECTION 5 - Updated Parser Code for Translation

```
#include<iostream>
#include<fstream>
#include<string>
#include <stdlib.h>
using namespace std;

/* INSTRUCTION:  copy your parser.cpp here
      cp ../ParserFiles/parser.cpp .
   Then, insert or append its contents into this file and edit.
   Complete all ** parts.
*/

//=================================================
// PARSER CODE
//=================================================

ifstream fin;
ofstream write;

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

// ** For the display names of tokens - must be in the same order as the
tokentype.
string tokenName[30] = {"WORD1", "WORD2", "PERIOD", "ERROR", "EOFM",
"VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS",
"OBJECT","SUBJECT", "DESTINATION", "PRONOUN", "CONNECTOR"};

string reservedWords[19][2] =
{
  {"masu", "VERB"},
  {"masen", "VERBNEG"},
  {"mashita", "VERBPAST"},
  {"masendeshita", "VERBPASTNEG"},
  {"desu", "IS"},
  {"deshita", "WAS"},
```

```
  {"o", "OBJECT"},
  {"wa", "SUBJECT"},
  {"ni", "DESTINATION"},
  {"watashi", "PRONOUN"},
  {"anata", "PRONOUN"},
  {"kare", "PRONOUN"},
  {"kanojo", "PRONOUN"},
  {"sore", "PRONOUN"},
  {"shikashi", "CONNECTOR"},
  {"dakara", "CONNECTOR"},
  {"mata", "CONNECTOR"},
  {"soshite", "CONNECTOR"},

  {"eofm", "EOFM"}
};

int scanner(tokentype& tt, string& w);
bool word(string s);
bool period(string s);

string filename;
tokentype saved_token;
string saved_lexeme;
bool token_available = false;

void syntaxerror1(tokentype expected, string lexeme)
{
   cout << "SYNTAX ERROR: expected " << tokenName[expected] << " but found
" << lexeme << endl;
   exit(EXIT_FAILURE);
}

void syntaxerror2(string lexeme, string parserFunction)
{
   cout << "SYNTAX ERROR: unexpected " << parserFunction << " found in "
<< lexeme << endl;
   exit(EXIT_FAILURE);
}

tokentype next_token()
```

```
{
    if (!token_available)                    // if there is no saved token
yet
    {
        scanner(saved_token, saved_lexeme);  // call scanner to grab a new
token
        cout << "Scanner called using word: " << saved_lexeme << endl;
        if (saved_lexeme == "eofm")
        {
            cout << "Successfully parsed <story>." << endl;
            exit(1);
        }
        token_available = true;              // mark that fact that you
have saved it
    }
    return saved_token;    // return the saved token
}

bool match(tokentype expected)
{
    if (next_token() != expected)  // mismatch has occurred with the next
token
    { // calls a syntax error function here to  generate a syntax error
message here and do recovery
        syntaxerror1(expected, saved_lexeme);
    }
    else  // match has occurred
    {
        cout << "Matched " << tokenName[expected] << endl;
        token_available = false;  // eat up the token
        return true;              // say there was a match
    }
}

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

  while (s[charpos] != '\0')
```

```
  {
    bool vowels = s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] ==
'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' ||
s[charpos] == 'E';
    bool bghkmpr = s[charpos] == 'b' || s[charpos] == 'g' || s[charpos] ==
'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' ||
s[charpos] == 'r';
    bool dwzyj = s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] ==
'z' || s[charpos] == 'y' || s[charpos] == 'j';

    if (state == 0 && vowels)
        state = 1;
    else if (state == 0 && s[charpos] == 't')
        state = 4;
    else if (state == 0 && s[charpos] == 's')
        state = 3;
    else if (state == 0 && s[charpos] == 'c')
        state = 2;
    else if (state == 0 && dwzyj)
        state = 6;
    else if (state == 0 && (bghkmpr || s[charpos] == 'n'))
        state = 5;

    else if (state == 5 && vowels)
        state = 1;
    else if (state == 5 && s[charpos] == 'y')
        state = 6;

    else if (state == 4 && vowels)
        state = 1;
    else if (state == 4 && s[charpos] == 's')
        state = 6;

    else if (state == 3 && vowels)
        state = 1;
    else if (state == 3 && s[charpos] == 'h')
        state = 6;

    else if (state == 2 && s[charpos] == 'h')
        state = 6;
```

```
    else if (state == 6 && vowels)
        state = 1;

    else if (state == 1 && vowels)
        state = 1;
    else if (state == 1 && s[charpos] == 't')
        state = 4;
    else if (state == 1 && s[charpos] == 's')
        state = 3;
    else if (state == 1 && s[charpos] == 'c')
        state = 2;
    else if (state == 1 && bghkmpr)
        state = 5;
    else if (state == 1 && s[charpos] == 'n')
        state = 7;
    else if (state == 1 && dwzyj)
        state = 6;

    else if (state == 7 && vowels)
        state = 1;
    else if (state == 7 && s[charpos] == 'c')
        state = 2;
    else if (state == 7 && s[charpos] == 's')
        state = 3;
    else if (state == 7 && s[charpos] == 't')
        state = 4;
    else if (state == 7 && (bghkmpr || s[charpos] == 'n'))
        state = 5;
    else if (state == 7 && dwzyj)
        state = 6;
    else
        return false;

    charpos++;
  }//end of while

  return state == 1 || state == 7;
}
```

```
bool period(string s)
{
    int state = 0;       // Initialize the state to 0
    int charpos = 0;     // Initialize the character position to 0

    // Loop through each character in the input string
    while (s[charpos] != '\0')
    {
        // Check the current state and character
        if (state == 0 && s[charpos] == '.')
        {
            // If the current state is 0 and the current character is a
period, transition to state 1
            state = 1;
        }
        else
        {
            // If the current state is not 0 or the current character is
not a period, return false
            return false;
        }
        charpos++;      // Increment the character position
    }

    // If the loop completes and the final state is 1, return true;
otherwise, return false
    return state == 1;
}

int scanner(tokentype& tt, string& w)
{
  fin >> w;
  if (w == "eofm")
    {
      tt = EOFM;
      return 0;              // 1. If it is eofm, return right now.
    }

  if (word(w))
{
```

```
    bool reserved = false;
    int size = sizeof(reservedWords) / sizeof(*reservedWords);
    for (int i = 0; i < size ; i++)
    {
        // Checking against the reserved words list if it's a word
        if (w == reservedWords[i][0])
        {
            // If it's a reserved word, get the associated token type
            string temp = reservedWords[i][1];
            int num = sizeof(tokenName) / sizeof(tokenName[0]);
            for (int j = 0; j < num; j++)
            {
                // Check the token type against the list of token names
                if (temp == tokenName[j])
                {
                    // If the token type is found, set the token type and
mark the word as reserved
                    tt = (tokentype)j;
                    reserved = true;
                    break;
                }
            }
            break;
        }
    }

    if (reserved)
      return 0;
    char back = w[w.size() - 1];
    //Tokentype is WORD1 or WORD2 based on the last character
    if (back == 'I' || back == 'E')
        tt = WORD2;
    else if (back == '*')
    {
        tt = ERROR;
        cout << "Lexical error: " << w << " is not a valid token" << endl;
    }
    else
        tt = WORD1;
  }
```

```
  //If the last character is a period
  else if (period(w))
      tt = PERIOD;
  else
  {
      //If both DFAs fail
      tt = ERROR;
      cout << "Lexical error: " << w << " is not a valid token" << endl;
    }
    return 0;

}//the end of scanner

//================================================
// END OF PARSER CODE
//================================================

//================================================
// File translator.cpp written by Group Number: ** 5
//================================================

// ----- Additions to the parser.cpp ---------------------

// ** Declare Lexicon (i.e. 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
//  Done by: ** Justin
string dictionary[53][2] = { " " }; // 2D array to hold dictionary
string saved_E_word; // Holds the English translation of saved_lexeme

// Function to read in the contents of lexicon.txt into the dictionary
array
void makeLexicon()
{
    fin.open("lexicon.txt"); // Open lexicon.txt file
    int i = 0;
    while (!fin.eof()) // Read the file until the end
    {
        string japaneseWord, englishWord;
```

```
        fin >> japaneseWord;
        fin >> englishWord;
        dictionary[i][0] = japaneseWord; // Japanese word will go into the
first column
        dictionary[i][1] = englishWord; // English word will go into the
second column
        i++;
    }
    fin.close();
}

void getEword(); // Forward declaration
void gen(string); // Forward declaration

// ** Additions to parser.cpp here:
//    getEword() - using the current saved_lexeme, look up the English
word
//                 in Lexicon if it is there -- save the result
//                 in saved_E_word
//  Done by: Kevin
void getEword()
{
    int rows = sizeof(dictionary) / sizeof(dictionary[0]); // Calculate
the number of rows in the dictionary array
    for (int i = 0; i < rows; i++) // Iterate through each row of the
dictionary array
    {
        if (dictionary[i][0] == saved_lexeme) // If the Japanese word
matches the saved_lexeme
        {
            saved_E_word = dictionary[i][1]; // Save the English
translation
            return;
        }
    }
    saved_E_word = saved_lexeme; // If no match was found, save the
saved_lexeme as is
}

//    gen(line_type) - using the line type,
```

```
//                    sends a line of an IR to translated.txt
//                    (saved_E_word or saved_token is used)
//  Done by: Justin
void gen(string line_type)
{
    if (line_type == "TENSE") // If line_type is TENSE
    {
        string tense = tokenName[saved_token]; // Get the name of the
saved token
        write << line_type << ":  " << tense << endl; // Write the tense
and its name to translated.txt
        return;
    }
    write << line_type << ":  " << saved_E_word << endl; // Write the
line_type and the saved_E_word to translated.txt
}

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

// ** Comment update: Be sure to put the corresponding grammar
//    rule with semantic routine calls
//    above each non-terminal function

// ** Each non-terminal function should be calling
//    getEword and/or gen now.

//Grammar: <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:
            syntaxerror2("tense", saved_lexeme);
        break;
    }
}

// Grammar: <be> ::= IS | WAS
void be()
{
    cout << "Processing <be>" << endl;
    switch(next_token())
    {
        case IS:
          match(IS);
      break;
        case WAS:
            match(WAS);
        break;
        default:
            syntaxerror2("be", saved_lexeme);
        break;
    }
}

// Grammar: <verb> ::= WORD2
void verb()
{
    cout << "Processing <verb>" << endl;
    match(WORD2);
}

// Grammar: <noun> ::= WORD1 | PRONOUN
void noun()
{
    cout << "Processing <noun>" << endl;
    switch(next_token())
```

```
    {
        case WORD1:
            match(WORD1);
        break;
        case PRONOUN:
            match(PRONOUN);
        break;
        default:
            syntaxerror2("noun", saved_lexeme);
        break;
    }
}

// Grammar: <after object> ::= <verb> #getEword# #gen(ACTION)# <tense>
#gen(TENSE)# PERIOD | <noun> #getEword# DESTINATION #gen(TO)# <verb>
#getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD
//  Done by: ** Kevin
void afterObject()
{
    cout << "Processing <afterObject>" << endl;  // display current
production rule

    switch (next_token())
    {
    case WORD2:
        verb();
        getEword();
        gen("ACTION");
        tense();
        gen("TENSE");
        match(PERIOD);  // if next token is a verb, call the verb
function, generate ACTION and TENSE and match PERIOD token
        break;
    case WORD1:
        noun();
        getEword();
        match(DESTINATION);  // if next token is a noun, call the noun
function and match DESTINATION token
        gen("TO");
        verb();
```

```
        getEword();
        gen("ACTION");
        tense();
        gen("TENSE");
        match(PERIOD);  // call the verb function, generate ACTION and
TENSE and match PERIOD token
        break;
    case PRONOUN:
        noun();
        getEword();
        match(DESTINATION);  // if next token is a pronoun, call the noun
function and match DESTINATION token
        gen("TO");
        verb();
        getEword();
        gen("ACTION");
        tense();
        gen("TENSE");
        match(PERIOD);  // call the verb function, generate ACTION and
TENSE and match PERIOD token
        break;
    default:
        syntaxerror2("afterObject", saved_lexeme);  // throw syntax error
if none of the above cases are true
        break;
    }
}


// Grammar: <after noun> ::= <be> #gen(DESCRIPTION)# #gen(TENSE)# PERIOD
| DESTINATION #gen(TO)# <verb> #getEword# #gen(ACTION)# <tense>
#gen(TENSE)# PERIOD | OBJECT #gen(OBJECT)# <after object>
//  Done by: ** Kevin
void afterNoun()
{
    cout << "Processing <afterNoun>" << endl;
    switch (next_token())
    {
    case IS:
        be();
```

```
        // generate a description
        gen("DESCRIPTION");
        // generate the tense
        gen("TENSE");
        match(PERIOD);
        break;
    case WAS:
        be();
        // generate a description
        gen("DESCRIPTION");
        // generate the tense
        gen("TENSE");
        match(PERIOD);
        break;
    case DESTINATION:
        match(DESTINATION);
        // generate "TO"
        gen("TO");
        verb();
        getEword();
        // generate the action
        gen("ACTION");
        // generate the tense
        tense();
        gen("TENSE");
        match(PERIOD);
        break;
    case OBJECT:
        match(OBJECT);
        // generate "OBJECT"
        gen("OBJECT");
        afterObject();
        break;
    default:
        syntaxerror2("afterNoun", saved_lexeme);
        break;
    }
}
```

```
// Grammar: <after subject> ::= <verb> #getEword# #gen(ACTION)# <tense>
#gen(TENSE)# PERIOD | <noun> #getEword# <after noun>
//  Done by: ** Justin & Kevin
void afterSubject()
{
    // prints out a message indicating that the function is processing
<afterSubject>
    cout << "Processing <afterSubject>" << endl;

    // a switch statement that checks the token returned by next_token()
function
    switch (next_token())
    {
        // if the token is WORD2:
    case WORD2:
        // call verb() function
        verb();
        // get the English word equivalent of the verb and add it to the
generated translation
        getEword();
        gen("ACTION");
        // call tense() function
        tense();
        // add the tense to the generated translation
        gen("TENSE");
        // match the token with PERIOD
        match(PERIOD);
        break;

        // if the token is WORD1:
    case WORD1:
        // call noun() function
        noun();
        // get the English word equivalent of the noun and add it to the
generated translation
        getEword();
        // call afterNoun() function
        afterNoun();
        break;
```

```
        // if the token is PRONOUN:
    case PRONOUN:
        // call noun() function
        noun();
        // get the English word equivalent of the noun and add it to the
generated translation
        getEword();
        // call afterNoun() function
        afterNoun();
        break;

        // if the token is not one of the above three:
    default:
        // call syntaxerror2() function with appropriate error message and
the saved_lexeme
        syntaxerror2("afterSubject", saved_lexeme);
        break;
    }
}


// Grammar: <s> ::= [CONNECTOR #getEword# #gen(CONNECTOR)#] <noun>
#getEword# SUBJECT #gen(ACTOR)# <after subject>
//  Done by: ** Justin & Kevin
void callStory()
{
    cout << "Processing <s>" << endl;

    // If the next token is a CONNECTOR, then process it before the noun
and subject
    if (next_token() == CONNECTOR)
    {
        match(CONNECTOR); // Match the CONNECTOR token
        getEword(); // Get the English equivalent of the CONNECTOR
        gen("CONNECTOR"); // Generate the CONNECTOR translation
        noun(); // Process the noun
        getEword(); // Get the English equivalent of the noun
        match(SUBJECT); // Match the SUBJECT token
        gen("ACTOR"); // Generate the actor translation
```

```
        afterSubject(); // Process the rest of the sentence after the
subject
        write << endl; // Write the translated sentence to the output file
    }
    else
    {
        noun(); // Process the noun
        getEword(); // Get the English equivalent of the noun
        match(SUBJECT); // Match the SUBJECT token
        gen("ACTOR"); // Generate the actor translation
        afterSubject(); // Process the rest of the sentence after the
subject
        write << endl; // Write the translated sentence to the output file
    }
}


// ---------------- Driver ---------------------------

// The final test driver to start the translator
// Done by:  ** Justin & Kevin
int main()
{
    //** Open the output file for writing
    write.open("translated.txt");

    //** Create the lexicon for translation
    makeLexicon();

    //** Prompt the user for the input file name and open the file
    cout << "Enter the input file name: ";
    cin >> filename;
    fin.open(filename.c_str());

    //** Begin parsing the input file using the <story> rule
    while (true)
        callStory();

    //** Close the input and output files
    fin.close();
```

```
    write.close();
}// end
//** require no other input files!
//** syntax error EC requires producing errors.txt of error messages
//** tracing On/Off EC requires sending a flag to trace message output
functions
```

SECTION 6 - Final Test Results

Test 1

```
Script started on Sat 13 May 2023 07:37:48 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$  ./a.out
Enter the input file name: partCtest1
Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: sensei
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
```

```
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: gohan
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: o
Matched OBJECT
Processing <afterObject>
Scanner called using word: tabE
Processing <verb>
Matched WORD2
Processing <tense>
Scanner called using word: masu
Matched VERB
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: tesuto
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: o
Matched OBJECT
Processing <afterObject>
Scanner called using word: seito
Processing <noun>
```

```
Matched WORD1
Scanner called using word: ni
Matched DESTINATION
Processing <verb>
Scanner called using word: agE
Matched WORD2
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: shikashi
Matched CONNECTOR
Processing <noun>
Scanner called using word: seito
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: yorokobI
Processing <verb>
Matched WORD2
Processing <tense>
Scanner called using word: masendeshita
Matched VERBPASTNEG
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: dakara
Matched CONNECTOR
Processing <noun>
Scanner called using word: watashi
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: kanashii
Processing <noun>
Matched WORD1
```

```
Processing <afterNoun>
Scanner called using word: deshita
Processing <be>
Matched WAS
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: soshite
Matched CONNECTOR
Processing <noun>
Scanner called using word: rika
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: toire
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: ni
Matched DESTINATION
Processing <verb>
Scanner called using word: ikI
Matched WORD2
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: nakI
Processing <verb>
Matched WORD2
Processing <tense>
```

```
Scanner called using word: mashita
Matched VERBPAST
Scanner called using word: .
Matched PERIOD
Processing <s>
Scanner called using word: eofm
Successfully parsed <story>.
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 07:38:07 PM PDT
```

Test 1 Translated.txt

```
ACTOR:  I/me
DESCRIPTION:  rika
TENSE:  IS

ACTOR:  I/me
DESCRIPTION:  teacher
TENSE:  IS

ACTOR:  rika
OBJECT:  meal
ACTION:  eat
TENSE:  VERB

ACTOR:  I/me
OBJECT:  test
TO:  student
ACTION:  give
TENSE:  VERBPAST

CONNECTOR:  However
ACTOR:  student
ACTION:  enjoy
TENSE:  VERBPASTNEG

CONNECTOR:  Therefore
```

```
ACTOR:  I/me
DESCRIPTION:  sad
TENSE:  WAS

CONNECTOR:  Then
ACTOR:  rika
TO:  restroom
ACTION:  go
TENSE:  VERBPAST

ACTOR:  rika
ACTION:  cry
TENSE:  VERBPAST
ACTOR:  I/me
DESCRIPTION:  rika
TENSE:  IS

ACTOR:  I/me
DESCRIPTION:  teacher
TENSE:  IS

ACTOR:  rika
OBJECT:  meal
ACTION:  eat
TENSE:  VERB

ACTOR:  I/me
OBJECT:  test
TO:  student
ACTION:  give
TENSE:  VERBPAST

CONNECTOR:  However
ACTOR:  student
ACTION:  enjoy
TENSE:  VERBPASTNEG

CONNECTOR:  Therefore
ACTOR:  I/me
DESCRIPTION:  sad
```

```
TENSE:  WAS

CONNECTOR:  Then
ACTOR:  rika
TO:  restroom
ACTION:  go
TENSE:  VERBPAST

ACTOR:  rika
ACTION:  cry
TENSE:  VERBPAST
```

Test 2

```
Script started on Sat 13 May 2023 08:48:15 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ ./a.out
Enter the input file name: partCtest2
Processing <s>
Scanner called using word: soshite
Matched CONNECTOR
Processing <noun>
Scanner called using word: watashi
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <afterSubject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <afterNoun>
Scanner called using word: desu
Processing <be>
Matched IS
Scanner called using word: ne
SYNTAX ERROR: expected PERIOD but found ne
```

```
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 08:48:32 PM PDT
```

Test 2 Translated.txt

```
CONNECTOR:  Then
ACTOR:  I/me
DESCRIPTION:  rika
TENSE:  IS
```

Test 3

```
Script started on Sat 13 May 2023 08:51:37 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$  ./a.out
Enter the input file name: partCtest3
Processing <s>
Scanner called using word: dakara
Matched CONNECTOR
Processing <noun>
Scanner called using word: watashi
Matched PRONOUN
Scanner called using word: de
SYNTAX ERROR: expected SUBJECT but found de
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 08:51:58 PM PDT
```

Test 3 Translated.txt

```
CONNECTOR:  Therefore
```

Test 4

```
Script started on Sat 13 May 2023 08:52:56 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ ./a.out
Enter the input file name: partCtest4
Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
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
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 08:53:13 PM PDT
```

Test 4 Translated.txt

```
ACTOR:  I/me
```

Test 5

```
Script started on Sat 13 May 2023 08:54:08 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ ./a.out
```

```
Enter the input file name: partCtest5
Processing <s>
Scanner called using word: wa
Processing <noun>
SYNTAX ERROR: unexpected wa found in noun
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 08:54:22 PM PDT
```

Test 5 Translated.txt

Nothing was written to Translated.txt

Test 6

```
Script started on Sat 13 May 2023 08:55:24 PM PDT
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[?1034h[luzan003@empress TranslatorFiles]$ g++ translator.cpp
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ ./a.out
Enter the input file name: partCtest6
Processing <s>
Lexical error: apple is not a valid token
Scanner called using word: apple
Processing <noun>
SYNTAX ERROR: unexpected apple found in noun
]0;luzan003@empress:~/cs421files/CS421Progs/TranslatorFiles
[luzan003@empress TranslatorFiles]$ exit
exit

Script done on Sat 13 May 2023 08:55:41 PM PDT
```

Test 6 Translated.txt

Nothing was written to Translated.txt