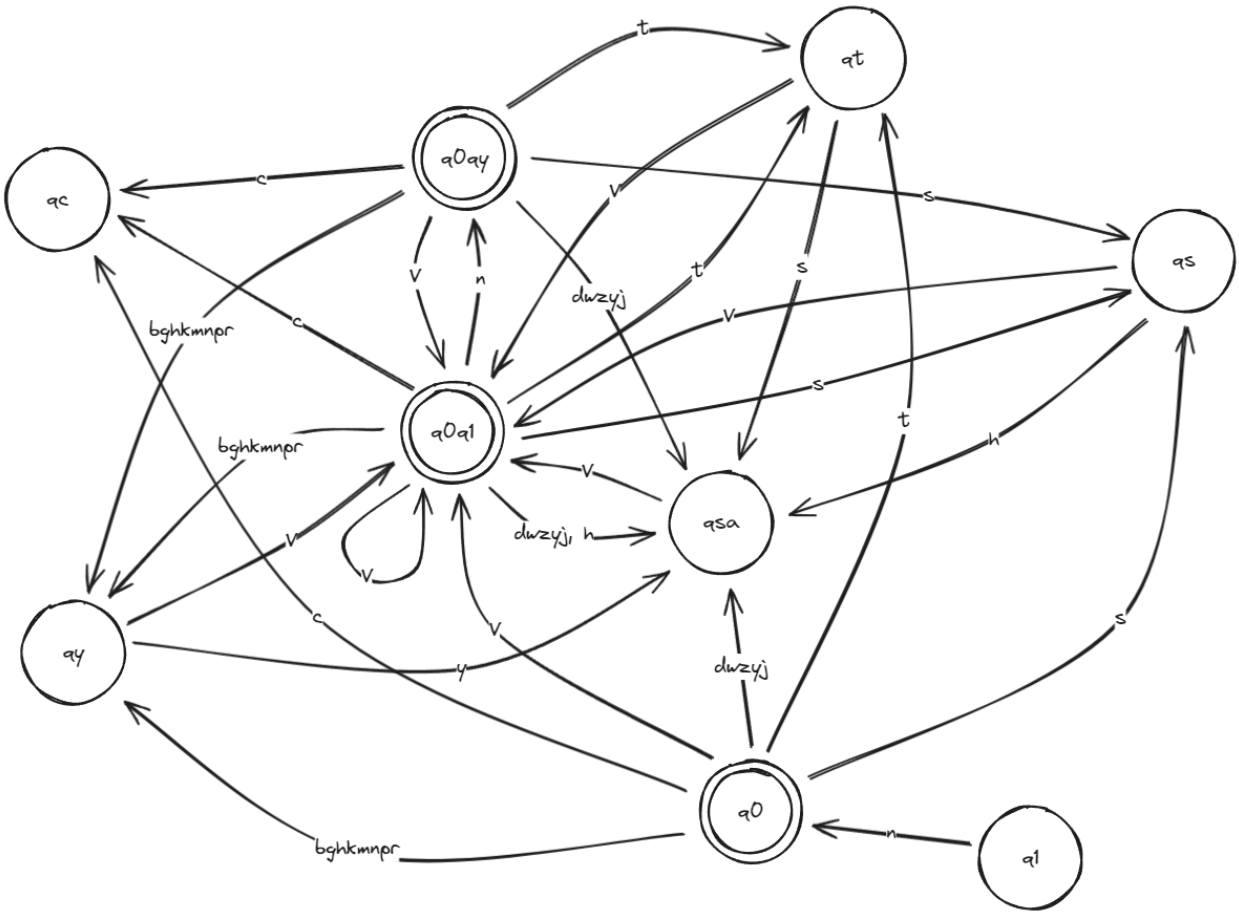
Japanese-English Translator

Team 10
Nathan Potraz
Adam Saltar
Alejandro Agustin

Section 0: State of the Program

The translator(part c) of our project works perfectly and passes all test cases given. No extra credit features were implemented. No bugs to report as well.

Section 1: DFA



Section 2: Scanner Code

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

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

//=================================================
// File scanner.cpp written by: Group Number: *10*
//=================================================

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

// WORD DFA
// Done by: *Adam Salter, Nathan Potraz*
// RE:   **


bool word(string s) {
   string array_of_strings[9] = {"q0", "q1", "qsa", "qy", "qt", "qs", "qc", "q0q1", "q0qy"};

   // q0 = state 0, q1 = state 1, qsa = state 2...
   //

   int state = 0;
   int charpos = 0;

   while (s[charpos] != '\0') {
     // cout << "CharPos: " << s[charpos] << ", State: " << state << endl; // For testing
      switch(state) {
         case 0: // q0
            switch(s[charpos]) {
               case 'a':
               case 'e':
               case 'i':
               case 'o':
               case 'u':
               case 'l':
               case 'E':
                  state = 7;
                  break;
```

```
        case 'n':
           state = 3;
           break;
        case 'd':
        case 'w':
        case 'z':
        case 'y':
        case 'j':
           state = 2;
           break;
        case 'b':
        case 'g':
        case 'h':
        case 'k':
        case 'm':
        case 'p':
        case 'r':
           state = 3;
           break;
        case 's':
           state = 5;
           break;
        case 't':
           state = 4;
           break;
        case 'c':
           state = 6;
           break;
        default:
           return false;
     }
     break;
  case 1: // q1
     if (s[charpos] == 'n')
        state = 0;
     break;
  case 2: // qsa
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
```

```
            state = 7;
            break;
case 3: // qy
   switch(s[charpos]) {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
      case 'l':
      case 'E':
         state = 7;
         break;
      case 'y':
         state = 2;
         break;
   }
   break;
case 4: // qt
   switch(s[charpos]) {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
      case 'l':
      case 'E':
         state = 7;
         break;
      case 's':
         state = 2;
         break;
   }
   break;
case 5: // qs
   switch(s[charpos]) {
      case 'a':
      case 'e':
      case 'i':
      case 'o':
      case 'u':
      case 'l':
      case 'E':
         state = 7;
```

```
            break;
        case 'h':
            state = 2;
            break;
    }
    break;
case 6: // qc
    if (s[charpos] == 'h')
        state = 2;
    break;
case 7: // q0q1
    switch(s[charpos]) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
            state = 7;
            break;
        case 'n':
            state = 8;
            break;
        case 'd':
        case 'w':
        case 'z':
        case 'y':
        case 'j':
            state = 2;
            break;
        case 'b':
        case 'g':
        case 'h':
        case 'k':
        case 'm':
        case 'p':
        case 'r':
            state = 3;
            break;
        case 's':
            state = 5;
            break;
        case 't':
```

```
            state = 4;
            break;
         case 'c':
            state = 6;
            break;
      }
      break;
   case 8: // q0qy
      switch(s[charpos]) {
         case 'a':
         case 'e':
         case 'i':
         case 'o':
         case 'u':
         case 'l':
         case 'E':
            state = 7;
            break;
         case 'n':
            state = 3;
            break;
         case 'd':
         case 'w':
         case 'z':
         case 'y':
         case 'j':
            state = 2;
            break;
         case 'b':
         case 'g':
         case 'h':
         case 'k':
         case 'm':
         case 'p':
         case 'r':
            state = 3;
            break;
         case 's':
            state = 5;
            break;
         case 't':
            state = 4;
            break;
         case 'c':
```

```
                state = 6;
                break;
            }
            break;
        }
        charpos++;
    }

    // Where did I end up?
    // cout << "State: " << state << endl; // For Testing
    if (state == 0 || state == 7 || state == 8)
        return true; // End in a final state
    else
        return false;
}




// PERIOD DFA
// Done by: Alejandro Agustin
//


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


    while (s[charpos] != '\0') {
        switch (state) {
            case 0:
                if (s[charpos] == '.')
                    state = 1;
                else
                    return false; // Not a period
                break;
            case 1:
                return false; // Already encountered a period, so any additional characters are
invalid
                break;
        }
        charpos++;
```

```
   }

   // Where did I end up?
   if (state == 1)
      return true; // End in a final state (period encountered)
   else
      return false;

}


// ------ Three  Tables ------------------------------------

// TABLES Done by: *Nathan Potraz*

// ** 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.
struct wordTable {
  string str;
  tokentype token;
};

wordTable reservedWords[19] = {
  {"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},
  {"mata", CONNECTOR},
  {"soshite", CONNECTOR},
  {"shikashi", CONNECTOR},
  {"dakara", 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: *Nathan Potraz*
int scanner(tokentype& tt, string& w)
{

  // ** Grab the next word from the file via fin

  fin >> w;

  // 1. If it is eofm, return right now.
  if(w == "eofm") {
    tt = EOFM;
    return tt;
  }

  /*  **
  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 foundWord = false;

    // for(wordTable word : reservedWords) {
    for(int i = 0; i < 19; i++) {
      if(reservedWords[i].str == w) {
        tt = reservedWords[i].token;
        foundWord = true;
        break;
      }
    }

    if(!foundWord) {
      char lastLetter = w[w.length() - 1];

      switch (lastLetter) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'n':
          tt = WORD1;
          break;
        case 'l':
        case 'E':
          tt = WORD2;
          break;
        default:
          tt = ERROR;
          break;
      }
    }

  } else if(period(w)) {
      tt = PERIOD;
  } else {
    tt = ERROR;
    cout << "ERROR: NOT VALID WORD OR PERIOD" << endl;
  }

  return tt;
}//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 << "\"" << theword << "\" is token type " << tokenName[thetype] << "\n" << endl;
    }

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

}// end
```

Section 3: Scanner Test Results

Scanner Test 1

```
Script started on 2024-05-16 13:31:45-07:00 [TERM="xterm-256color" TTY="/dev/pts/7"
COLUMNS="211" LINES="44"]
```

```
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ g++ scanner.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ ./a.out
[?2004lEnter the input file name: scannertest1
"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"rika" is token type WORD1

"desu" is token type IS

"." is token type PERIOD

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"sensei" is token type WORD1

"desu" is token type IS

"." is token type PERIOD

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"ryouri" is token type WORD1

"o" is token type OBJECT

"yarl" is token type WORD2

"masu" is token type VERB

"." is token type PERIOD

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"gohan" is token type WORD1
```

"o" is token type OBJECT

"seito" is token type WORD1

"ni" is token type DESTINATION

"agE" is token type WORD2

"mashita" is token type VERBPAST

"." is token type PERIOD

"shikashi" is token type CONNECTOR

"seito" is token type WORD1

"wa" is token type SUBJECT

"yorokobI" is token type WORD2

"masendeshita" is token type VERBPASTNEG

"." is token type PERIOD

"dakara" is token type CONNECTOR

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"kanashii" is token type WORD1

"deshita" is token type WAS

"." is token type PERIOD

"soshite" is token type CONNECTOR

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"toire" is token type WORD1

"ni" is token type DESTINATION

"ikl" is token type WORD2

"mashita" is token type VERBPAST

"." is token type PERIOD

"watashi" is token type PRONOUN

"wa" is token type SUBJECT

"nakl" is token type WORD2

"mashita" is token type VERBPAST

"." is token type PERIOD

End of file is encountered.
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ exit
[?2004lexit

Script done on 2024-05-16 13:32:05-07:00 [COMMAND_EXIT_CODE="0"]

Scanner Test 2

Script started on 2024-05-16 13:32:10-07:00 [TERM="xterm-256color" TTY="/dev/pts/7" COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ g++ scanner.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ ./a.out
[?2004lEnter the input file name: scannertest2
"daigaku" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"college" is token type ERROR

"kurasu" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD

"class" is token type ERROR

"hon" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"book" is token type ERROR

"tesuto" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"test" is token type ERROR

"ie" is token type WORD1

"home*" is token type ERROR

"isu" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"chair" is token type ERROR

"seito" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"student" is token type ERROR

"sensei" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"teacher" is token type ERROR

"tomodachi" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"friend" is token type ERROR

"jidoosha" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"car" is token type ERROR

"gyuunyuu" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD

"milk" is token type ERROR

"sukiyaki" is token type WORD1

"tenpura" is token type WORD1

"sushi" is token type WORD1

"biiru" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"beer" is token type ERROR

"sake" is token type WORD1

"tokyo" is token type WORD1

"kyuushuu" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"Osaka" is token type ERROR

"choucho" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"butterfly" is token type ERROR

"an" is token type WORD1

"idea" is token type WORD1

"yasashii" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"easy" is token type ERROR

"muzukashii" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"difficult" is token type ERROR

"ureshii" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD

"pleased" is token type ERROR

"shiawase" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"happy" is token type ERROR

"kanashii" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"sad" is token type ERROR

"omoi" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"heavy" is token type ERROR

"oishii" is token type WORD1

ERROR: NOT VALID WORD OR PERIOD
"delicious" is token type ERROR

"tennen" is token type WORD1

"natural" is token type ERROR

"nakI" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"cry" is token type ERROR

"ikI" is token type WORD2

"go*" is token type ERROR

"tabE" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"eat" is token type ERROR

"ukE" is token type WORD2

"take*" is token type ERROR

"kakl" is token type WORD2

"write" is token type WORD1

"yoml" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"read" is token type ERROR

"noml" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"drink" is token type ERROR

"agE" is token type WORD2

"give" is token type WORD1

"moral" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"receive" is token type ERROR

"butsl" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"hit" is token type ERROR

"kerl" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"kick" is token type ERROR

"shaberl" is token type WORD2

ERROR: NOT VALID WORD OR PERIOD
"talk" is token type ERROR

End of file is encountered.
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_A_Scanner[?2004h[potra002@empress Part_A_Scanner]$ exit
[?2004lexit

Script done on 2024-05-16 13:32:28-07:00 [COMMAND_EXIT_CODE="0"]

Section 4: Factored Rules

Parser Rules

<s> ::= [CONNECTOR] <noun> SUBJECT <after subject>
<after subject> ::= <verb> <tense> PERIOD
| <noun> <after noun>

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

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

Updated Left Factoring Rules

<story> ::= <s> {<s>}

<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: Parser/Translator Code

```
#include<iostream>
#include<fstream>
#include<string>
#include<map>
#include<sstream>
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.
*/

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

//=================================================
// File scanner.cpp written by: Group Number: *10*
//=================================================

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

// WORD DFA
// Done by: *Adam Salter, Nathan Potraz*
// RE:   **

string currWord;
map<string, string> Lexicon;  // Global variable for the lexicon
// saved_token is defined in scanner as a tokentype enum
string saved_lexeme, saved_E_word;  // Global variables for word management
ofstream outFile;  // File stream for output
// ifstream fin; - Defined in scanner section  // File stream for input

bool word(string s) {
   if (s.empty()) {
     return false;
   }

   string array_of_strings[9] = {"q0", "q1", "qsa", "qy", "qt", "qs", "qc", "q0q1", "q0qy"};

   // q0 = state 0, q1 = state 1, qsa = state 2...
   //
```

```
int state = 0;
int charpos = 0;
currWord = s;

while (s[charpos] != '\0') {
  // cout << "CharPos: " << s[charpos] << ", State: " << state << endl; // For testing
    switch(state) {
        case 0: // q0
            switch(s[charpos]) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'l':
                case 'E':
                    state = 7;
                    break;
                case 'n':
                    state = 3;
                    break;
                case 'd':
                case 'w':
                case 'z':
                case 'y':
                case 'j':
                    state = 2;
                    break;
                case 'b':
                case 'g':
                case 'h':
                case 'k':
                case 'm':
                case 'p':
                case 'r':
                    state = 3;
                    break;
                case 's':
                    state = 5;
                    break;
                case 't':
                    state = 4;
                    break;
```

```
        case 'c':
            state = 6;
            break;
        default:
            return false;
    }
    break;
case 1: // q1
    if (s[charpos] == 'n')
        state = 0;
    break;
case 2: // qsa
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
            state = 7;
            break;
        default:
            return false;
case 3: // qy
    switch(s[charpos]) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
            state = 7;
            break;
        case 'y':
            state = 2;
            break;
        default:
            return false;
    }
    break;
case 4: // qt
    switch(s[charpos]) {
        case 'a':
```

```
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
            state = 7;
            break;
        case 's':
            state = 2;
            break;
        default:
            return false;
    }
    break;
case 5: // qs
    switch(s[charpos]) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'l':
        case 'E':
            state = 7;
            break;
        case 'h':
            state = 2;
            break;
        default:
            return false;
    }
    break;
case 6: // qc
    if (s[charpos] == 'h')
        state = 2;
    else return false;
    break;
case 7: // q0q1
    switch(s[charpos]) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
```

```
            case 'u':
            case 'l':
            case 'E':
               state = 7;
               break;
            case 'n':
               state = 8;
               break;
            case 'd':
            case 'w':
            case 'z':
            case 'y':
            case 'j':
               state = 2;
               break;
            case 'b':
            case 'g':
            case 'h':
            case 'k':
            case 'm':
            case 'p':
            case 'r':
               state = 3;
               break;
            case 's':
               state = 5;
               break;
            case 't':
               state = 4;
               break;
            case 'c':
               state = 6;
               break;
            default:
               return false;
         }
         break;
      case 8: // q0qy
         switch(s[charpos]) {
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
```

```
            case 'l':
            case 'E':
               state = 7;
               break;
            case 'n':
               state = 3;
               break;
            case 'd':
            case 'w':
            case 'z':
            case 'y':
            case 'j':
               state = 2;
               break;
            case 'b':
            case 'g':
            case 'h':
            case 'k':
            case 'm':
            case 'p':
            case 'r':
               state = 3;
               break;
            case 's':
               state = 5;
               break;
            case 't':
               state = 4;
               break;
            case 'c':
               state = 6;
               break;
            default:
               return false;
         }
         break;
   }
   charpos++;
}

// Where did I end up?
// cout << "State: " << state << endl; // For Testing
if (state == 0 || state == 7 || state == 8)
   return true; // End in a final state
```

```
    else
        return false;
}

// PERIOD DFA
// Done by: Alejandro Agustin
//


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


    while (s[charpos] != '\0') {
        switch (state) {
            case 0:
                if (s[charpos] == '.')
                    state = 1;
                else
                    return false; // Not a period
                break;
            case 1:
                return false; // Already encountered a period, so any additional characters are
invalid
                break;
        }
        charpos++;
    }

    // Where did I end up?
    if (state == 1)
        return true; // End in a final state (period encountered)
    else
        return false;

}


// ------ Three  Tables ------------------------------------

// TABLES Done by: *Nathan Potraz*

// ** 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.
struct wordTable {
  string str;
  tokentype token;
};

wordTable reservedWords[19] = {
  {"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},
  {"mata", CONNECTOR},
  {"soshite", CONNECTOR},
  {"shikashi", CONNECTOR},
  {"dakara", 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: *Nathan Potraz*
int scanner(tokentype& tt, string& w)
{

  // ** Grab the next word from the file via fin

  fin >> w;

        // cout << "Scanner called using word: " << w << endl;

  // 1. If it is eofm, return right now.
  if(w == "eofm") {
    tt = EOFM;
    return tt;
  }

  /*  **
  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 foundWord = false;

    // cout << "LOOKING FOR WORD: " << w << endl;
    for(int i = 0; i < 19; i++) {
      if(reservedWords[i].str == w) {
        tt = reservedWords[i].token;
        foundWord = true;
        break;
      }
    }
```

```
    if(!foundWord) {
      char lastLetter = w[w.length() - 1];

      switch (lastLetter) {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'n':
          tt = WORD1;
          break;
        case 'l':
        case 'E':
          tt = WORD2;
          break;
        default:
          tt = ERROR;
          break;
      }
    }

  } else if(period(w)) {
      tt = PERIOD;
  } else {
    tt = ERROR;
    cout << "\nLexical error: " << w << " not a valid token" << endl;
  }

  // cout << "WORD " << w << " FOUND, TOKEN: " << tokenName[tt] << endl;
  return tt;
}//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 << "End of file is encountered." << endl;
//    fin.close();
//
// }// end

/* INSTRUCTION:  Complete all ** parts.
   You may use any method to connect this file to scanner.cpp
   that you had written.
  e.g. You can copy scanner.cpp here by:
         cp ../ScannerFiles/scanner.cpp .
       and then append the two files into one:
         cat scanner.cpp parser.cpp > myparser.cpp
*/

// ------------------------------ Parser -------------------------------------

//================================================
// File parser.cpp written by Group Number: *10*
//================================================

// ----- Four Utility Functions and Globals ---------------------------------

// ** Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
//    to display syntax error messages as specified by me.
tokentype saved_token; //stores last token
bool token_available = false; //indicates if there was a saved token ava

// Type of error: **
```

```
// Done by: Alejandro Agustin
void syntaxerror1( tokentype expected){
  cout << "\nSYNTAX ERROR: Expected " << tokenName[expected] << " at \"" <<
saved_lexeme << "\"" << endl;
  exit(1);
}
// Type of error: **
// Done by: Alejandro Agustin
void syntaxerror2( tokentype unexpected, string function ) {
  cout << "\nSYNTAX ERROR: Unexpected " << currWord << " in " << function << endl;
  exit(1);
}

// ** Need the updated match and next_token with 2 global vars
// saved_token and saved_lexeme

// Purpose: **
// Done by: Alejandro Agustin, Nathan Potraz
tokentype next_token(){
                    // string saved_lexeme; // The current word being looked at
   if(!token_available){
      scanner(saved_token, saved_lexeme);
      // cout << "saved_lexeme: " << saved_lexeme << endl;
      if(saved_token == EOFM) {
        cout << "\nSuccessfully parsed <story>" << endl;
        exit(0);
      }
      token_available = true;
  }
  return saved_token;
}

// Purpose: **
// Done by: Alejandro Agustin, Nathan Potraz
bool match(tokentype expected) {
  if (saved_token != expected){
    if (expected == VERB && (saved_token == VERBNEG || saved_token == VERBPAST ||
saved_token == VERBPASTNEG || saved_token == WORD2)) {
      token_available = false;
                                  // cout << "Matched " << tokenName[expected] << endl;
      return true;
    }
    if (expected == PRONOUN && saved_token == WORD1) {
      token_available = false;
```

```
                                    // cout << "Matched " << tokenName[expected] << endl;
      return true;
    }
    if (expected == IS && saved_token == WAS) {
      token_available = false;
                                    // cout << "Matched " << tokenName[expected] << endl;
      return true;
    }

      syntaxerror1(expected);
      return false;
  } else {
      token_available = false;
                                    // cout << "Matched " << tokenName[expected] << endl;
      return true;
  }
}

// ----- RDP functions - one per non-term ------------------

// ** Make each non-terminal into a function here
// ** Be sure to put the corresponding grammar rule above each function
// ** Be sure to put the name of the programmer above each function

// Processing <story>
// Done by: *Nathan Potraz*

// void story() {
//   cout << "Processing <story>\n" << endl;
//   s();
// }

// Processing <s>
// Grammar: <s> ::= [CONNECTOR]<noun>SUBJECT<after_subject>
// Done by: Adam Salter
// void s() {
//     cout << "Processing <s>" << endl;
//     if (next_token() == CONNECTOR) {  // Optional CONNECTOR
//         match(CONNECTOR);
//     }
//     noun();
//     match(SUBJECT);
//     after_subject();
// }
```

```
// Processing <noun>
// Done by: Adam Salter
// void noun() {
//     cout << "Processing <noun>" << endl;
//     if (next_token() == WORD1 || next_token() == PRONOUN) {
//         match(next_token()); // Matches either WORD1 or PRONOUN
//     } else {
//         syntaxerror2(next_token(), "<noun>");
//     }
// }

// Processing <after subject>
// Grammar: <after_subject> ::= <verb><tense>PERIOD | <noun><after_noun>
// Done by: Adam Salter
// void after_subject() {
//     cout << "Processing <after subject>" << endl;
//     switch(next_token()) {
//         case WORD2:
//         case VERB:
//             verb();
//             tense();
//             match(PERIOD);
//             s();
//             break;
//         case WORD1:
//         case PRONOUN:
//             noun();
//             after_noun();
//             break;
//         default:
//             syntaxerror2(next_token(), "<after subject>");
//     }
// }

// Processing <verb>
// Done by: Adam Salter
// void verb() {
//     cout << "Processing <verb>" << endl;
//     match(WORD2); // Assuming VERB corresponds to WORD2
// }

// Processing <tense>
```

```
// Done by: Adam Salter
// void tense() {
//     cout << "Processing <tense>" << endl;
//     if (next_token() == VERBPAST || next_token() == VERBPASTNEG || next_token() ==
VERB || next_token() == VERBNEG) {
//         match(next_token()); // Matches any valid tense
//     } else {
//         syntaxerror2(next_token(), "<tense>");
//     }
// }

// Processing <after noun>
// Grammar: <after_noun> ::= <be>PERIOD | DESTINATION<verb><tense>PEROID |
OBJECT<after_object>
// Done by: Adam Salter
// void after_noun() {
//     cout << "Processing <after noun>" << endl;
//     switch(next_token()) {
//         case IS:
//         case WAS:
//             be();
//             match(PERIOD);
//             s();
//             break;
//         case DESTINATION:
//             match(DESTINATION);
//             verb();
//             tense();
//             match(PERIOD);
//             s();
//             break;
//         case OBJECT:
//             match(OBJECT);
//             after_object();
//             break;
//         default:
//             syntaxerror2(next_token(), "<after noun>");
//     }
// }

// Processing <be>
// Done by: Adam Salter
// void be() {
//     cout << "Processing <be>" << endl;
```

```
//     if (next_token() == IS || next_token() == WAS) {
//         match(next_token());
//     } else {
//         syntaxerror2(next_token(), "<be>");
//     }
// }

// Processing <after object>
// Grammar: <after_object> ::= <verb><tense>PERIOD |
<noun>DESTINATION<verb><tense>PERIOD
// Done by: Adam Salter
// void after_object() {
//     cout << "Processing <after object>" << endl;
//     switch(next_token()) {
//         case WORD2:
//         case VERB:
//             verb();
//             tense();
//             match(PERIOD);
//             s();
//             break;
//         case WORD1:
//         case PRONOUN:
//             noun();
//             match(DESTINATION);
//             verb();
//             tense();
//             match(PERIOD);
//             s();
//             break;
//         default:
//             syntaxerror2(next_token(), "<after object>");
//     }
// }


string filename;

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

// The new test driver to start the parser
// Done by:  *Nathan Potraz*
// int main() {
//   cout << "Group 10 Parser" << endl;
```

```
//   cout << "Created by: Nathan Potraz, Adam Saltar, and Alejandro Agustin\n" << endl;
//   cout << "Enter the input file name: ";
//   cin >> filename;
//   fin.open(filename.c_str());
//
//   //** calls the <story> to start parsing
//   //** closes the input file
//
//   story();
//
//         fin.close();
//   return 0;
// }// end
//** require no other input files!
//** syntax error EC requires producing errors.txt of error messages
//
//
//================================================
// File translator.cpp written by Group Number: 10
//================================================


// Function to load lexicon from a file
void loadLexicon() {
   ifstream lexFile("lexicon.txt");
   if (!lexFile.is_open()) {
      cerr << "Failed to open lexicon.txt" << endl;
      return;
   }

   string line;
   while (getline(lexFile, line)) {
      istringstream iss(line);
      string japWord, engWord;

      // Read the Japanese word (first word on each line)
      iss >> japWord;

      // Read the rest of the line as the English translation
      getline(iss, engWord);

      // Trim leading spaces from the English word
      size_t start = engWord.find_first_not_of(" \t");
      engWord = (start == string::npos) ? "" : engWord.substr(start);
```

```
        Lexicon[japWord] = engWord;
    }

    lexFile.close();

    // Prints out the Lexicon Map
    // for (const auto& [key, value] : Lexicon) {
    //     cout << "Key[" << key << "]: " << "Value[" << value << "]" << endl;
    // }

}
// ** 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: **Alejandro Agustin**

// Function to translate Japanese words to English
void getEword() {
    auto it = Lexicon.find(saved_lexeme);
    if (it != Lexicon.end()) {
        saved_E_word = it->second;
    } else {
        saved_E_word = saved_lexeme; // If not found, use the Japanese word as a fallback
    }
}

// Done by: **Nathan Potraz**
// Function to generate lines of IR and write to translated.txt
void gen(string line_type) {
    if (!outFile.is_open()) {
        cerr << "Output file not open for writing." << endl;
        return;
    }
    outFile << line_type << ": ";
    if (line_type == "TENSE") {
        // outFile << saved_token; // TENSE uses tokens
        outFile << tokenName[saved_token]; // TENSE uses tokens
    } else {
        outFile << saved_E_word; // Other types use the saved English word
    }
    outFile << endl;
}
```

```
void afterObject();
void afterNoun();
void afterSubject();
void s();
void story();

// Done by: **Alejandro Agustin, Nathan Potraz**
// Function to handle the <story> rule
void story() {
   // cout << "story() called" << endl;
   s();
   while (saved_token != EOFM) {
      s();
   }
}
// *Done by:*Adam Salter, Nathan Potraz**
// Function to handle the <s> rule
// <s> ::= [CONNECTOR #getEword# #gen("CONNECTOR")#] <noun> #getEword#
//               SUBJECT #gen("ACTOR")# <after subject>
void s() {
   // cout << "s() called" << endl;
   next_token();
   // if (tokenName[saved_token] == "CONNECTOR") {
   if (saved_token == CONNECTOR) {
      match(CONNECTOR);
      // cout << "WE CONNECTOR" << endl;
      getEword();
      gen("CONNECTOR");
      next_token();
   }
   // Assuming <noun> is processed here
   else if(saved_token == PRONOUN || saved_token == WORD1) {
     match(PRONOUN);
     getEword();
     next_token();
     if(saved_token == SUBJECT) {
       match(SUBJECT);
       gen("ACTOR");
       afterSubject();
     } else {
       match(SUBJECT);
     }
   }
   else {
```

```
    syntaxerror2(saved_token, "<s>");
  }
}

// Done by: **Adam Salter, Nathan Potraz**
// Function to handle the <after subject> rule
// <after subject> ::= <verb> #getEword# #gen("ACTION")# <tense> #gen("TENSE")#
//                  PERIOD | <noun> #getEword# <after noun>
void afterSubject() {
  // cout << "afterSubject() called" << endl;
  next_token();
  // if (tokenName[saved_token] == "VERB") {
  if (saved_token == VERB || saved_token == WORD2) {
    match(VERB);
    getEword();
    gen("ACTION");
    next_token(); // Get tense
    match(VERB);
    gen("TENSE");
    next_token(); // Get period
    match(PERIOD);
    outFile << "\n";
  } else if(saved_token == PRONOUN || saved_token == WORD1){
    // Assuming <noun> is processed here
    match(PRONOUN);
    getEword();
    afterNoun();
  } else {
    syntaxerror2(saved_token, "<afterSubject>");
  }
}

// Done by: **Adam Salter, Nathan Potraz**
// Function to handle the <after noun> rule
// <after noun> ::= <be> #gen("DESCRIPTION")# #gen("TENSE")# PERIOD |
//                DESTINATION #gen("TO")# <verb> #getEword# #gen("ACTION")#
//                <tense> #gen("TENSE")# PERIOD |
//                OBJECT #gen("OBJECT")# <after object>
void afterNoun() {
  // cout << "afterNoun() called" << endl;
  next_token();
  // if (tokenName[saved_token] == "BE") {
  if (saved_token == IS || saved_token == WAS) {
    match(IS);
```

```
        gen("DESCRIPTION");
        // next_token(); // Get tense
        gen("TENSE");
        next_token(); // Get period
        match(PERIOD);
        outFile << "\n";
    // } else if (tokenName[saved_token] == "DESTINATION") {
    } else if (saved_token == DESTINATION) {
        match(DESTINATION);
        gen("TO");
        next_token(); // Get verb
        match(VERB);
        if(saved_token == VERB || saved_token == WORD2) {
          getEword();
          gen("ACTION");
          next_token(); // Get tense
          match(VERB);
          gen("TENSE");
          next_token(); // Get period
          match(PERIOD);
          outFile << "\n";
        }
    // } else if (tokenName[saved_token] == "OBJECT") {
    } else if (saved_token == OBJECT) {
        match(OBJECT);
        gen("OBJECT");
        afterObject();
    } else {
      syntaxerror2(saved_token, "<afterNoun>");
    }
}

// Done by: **Adam Salter, Nathan Potraz**
// Function to handle the <after object> rule
// <after object> ::= <verb> #getEword# #gen("ACTION")# <tense> #gen("TENSE")#
//                 PERIOD | <noun> #getEword# DESTINATION #gen("TO")# <verb>
//                 #getEword# #gen("ACTION")# <tense> #gen("TENSE")# PERIOD
void afterObject() {
    // cout << "afterObject() called" << endl;
    next_token();
    //if (tokenName[saved_token] == "VERB") {
    if (saved_token == VERB || saved_token == WORD2) {
        // cout << "WE A VERB" << endl;
        match(VERB);
```

```
        getEword();
        gen("ACTION");
        next_token(); // Get tense
        match(VERB);
        gen("TENSE");
        next_token(); // Get period
        match(PERIOD);
        outFile << "\n";
    } else if (saved_token == PRONOUN || saved_token == WORD1) {
        // cout << "WE A NOUN" << endl;
        match(PRONOUN);
        getEword();
        next_token(); // Get destination
        match(DESTINATION);
        gen("TO");
        next_token(); // Get verb
        match(VERB);
        getEword();
        gen("ACTION");
        next_token(); // Get tense
        match(VERB);
        gen("TENSE");
        next_token(); // Get period
        match(PERIOD);
        outFile << "\n";
    } else {
      syntaxerror2(saved_token, "<afterObject>");
    }
}

// Done by: **Adam Salter**
// Driver function
int main() {
    string filename;

    loadLexicon(); // Load the lexicon for translations
    outFile.open("translated.txt"); // Open the output file for IR generation
    if (!outFile.is_open()) {
        cerr << "Failed to open translated.txt" << endl;
        return -1;
    }

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

```
    fin.open(filename.c_str());
    if (!fin.is_open()) {
        cerr << "Failed to open input file." << endl;
        outFile.close(); // Close outFile before exiting
        return -1;
    }

    story(); // Call the story function to process the input file

    fin.close(); // Close the input file
    outFile.close(); // Close the output file

    return 0;
}
// Done by: **Adam Salter**
// 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: Test Results

Translator Test 1

```
Script started on 2024-05-16 13:42:41-07:00 [TERM="xterm-256color" TTY="/dev/pts/7"
COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[po
tra002@empress Part_C_Translator]$ g++ translator.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2
004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest1

Successfully parsed <story>
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[po
tra002@empress Part_C_Translator]$ cat translated.txt
[?2004lACTOR: 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

]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[po
tra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:43:12-07:00 [COMMAND_EXIT_CODE="0"]

Translator Test 2

Script started on 2024-05-16 13:43:34-07:00 [TERM="xterm-256color" TTY="/dev/pts/7"
COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[po
tra002@empress Part_C_Translator]$ g++ translator.cpp

[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest2

SYNTAX ERROR: Expected PERIOD at "ne"
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ cat translated.txt
[?2004lCONNECTOR: Then
ACTOR: I/me
DESCRIPTION: rika
TENSE: IS
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:44:02-07:00 [COMMAND_EXIT_CODE="0"]

Translator Test 3

Script started on 2024-05-16 13:44:16-07:00 [TERM="xterm-256color" TTY="/dev/pts/7" COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ g++ translator.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest3

SYNTAX ERROR: Expected SUBJECT at "de"
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ cat translated.txt
[?2004lCONNECTOR: Therefore
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:44:53-07:00 [COMMAND_EXIT_CODE="0"]

Translator Test 4

Script started on 2024-05-16 13:45:04-07:00 [TERM="xterm-256color" TTY="/dev/pts/7" COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ g++ translator.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest4.txt
Failed to open input file.
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest4

SYNTAX ERROR: Unexpected mashita in <afterNoun>
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ cat translated.txt
[?2004lACTOR: I/me
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:45:49-07:00 [COMMAND_EXIT_CODE="0"]

Translator Test 5

Script started on 2024-05-16 13:46:08-07:00 [TERM="xterm-256color" TTY="/dev/pts/7" COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ g++ transltor.cpp
[?2004l[01m[Kcc1plus:[m[K [01;31m[Kfatal error: [m[Ktransltor.cpp: No such file or directory
compilation terminated.
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ g++ transltor.cp[Ktranslator.cpp
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: tranpartCtest5

SYNTAX ERROR: Unexpected wa in <s>
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ cat translated.txt

[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:46:48-07:00 [COMMAND_EXIT_CODE="0"]

Translator Test 6

Script started on 2024-05-16 13:47:04-07:00 [TERM="xterm-256color" TTY="/dev/pts/7" COLUMNS="211" LINES="44"]
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ g++ translator.cpp
[?2004l./a.out]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ ./a.out
[?2004lEnter the input file name: partCtest6

Lexical error: apple not a valid token

SYNTAX ERROR: Unexpected apple in <s>
]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ cat translated.txt
[?2004l]0;potra002@empress:~/Documents/cs421/cs421-translator/Part_C_Translator[?2004h[potra002@empress Part_C_Translator]$ exit
[?2004lexit

Script done on 2024-05-16 13:47:25-07:00 [COMMAND_EXIT_CODE="0"]