**0 - Cover Sheet**

CS421 Project Report

# Japanese-to-English Translator // Group: 11

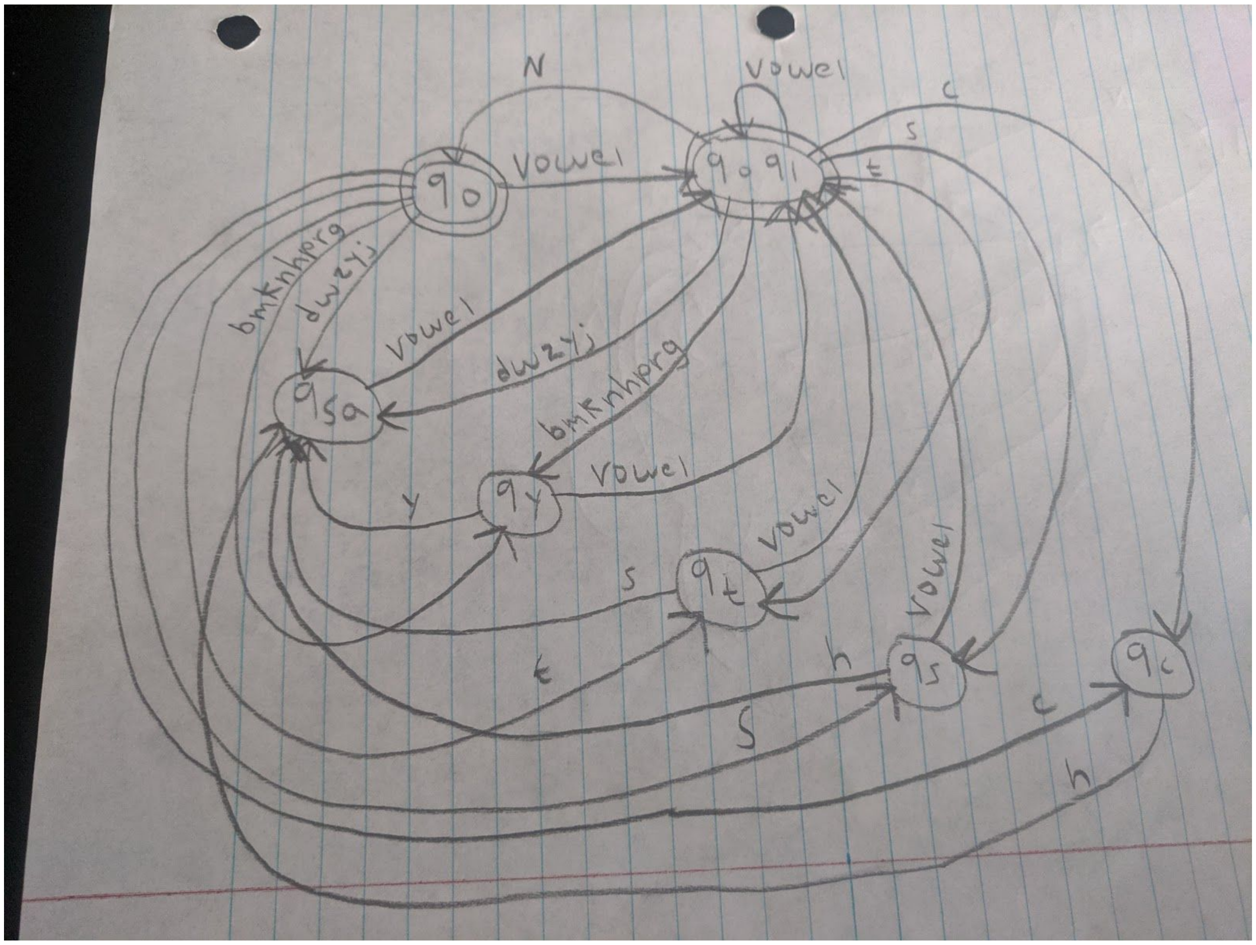
Done by:

Paolo Rimando
Ralph Lira
Esai Delgado

State of Program:

- The scanner, parser, and translator of the project is fully completed and working perfectly with no bugs that were detected.
- Extra Credit feature that were implemented is the ability to enable and disable tracing messages. To accomplish this, (if-statement) conditions were added to precede trace message couts. Traces would only display if bool display_trace was true. A prompt was added to the main driver that asks if the user wants to display trace messages; bool display_trace was true or false depending on the response.

## 1 - Final version of Japanese word DFA



**2 - DFA 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: 11
//=====================================================

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

// WORD DFA
// Done by: Paolo Rimando
/* 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;

  while(s[charpos] != '\0')
    {
      //cout << "state " << state << endl;
      //cout << "char " << s[charpos] << endl;

      if(state == 0){// q0 ************
            switch(s[charpos]){
            case 'a': case 'e':
            case 'i': case 'o':
            case 'u':
            case 'E': case 'I':
              state = 1; // q010
              break;
            case 'd': case 'w':
        case 'y': case 'z':
        case 'j':
```

```
    state = 2; // qsa
    break;
      case 'b': case 'm':
  case 'k': case 'n':
  case 'h': case 'p':
  case 'r': case 'g':
    state = 3; // qy
    break;
      case 't':
        state = 4; // qt
        break;
      case 's':
        state = 5; // qs
        break;
      case 'c':
        state = 6; // qc
        break;
      default:
        return false;
      }
}
else if(state == 1){// q0q1 ************
      switch(s[charpos]){
      case 'n':
        state = 0; // q0
        break;
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        break; // stay q0q1
      case 'd': case 'w':
  case 'y': case 'z':
  case 'j':
        state = 2; // qsa
        break;
      case 'b': case 'm':
  case 'k':
  case 'h': case 'p':
  case 'r': case 'g':
        state = 3; // qy
        break;
      case 't':
        state = 4; // qt
        break;
  case 's':
        state = 5; // qs
        break;
  case 'c':
        state = 6; // qc
        break;
      default:
        return false;
      }
}
else if(state == 2){// qsa ************
      switch(s[charpos]){
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        state = 1; // q0q1
        break;
      default:
        return false;
      }
}
else if(state == 3){// qy ************
      switch(s[charpos]){
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        state = 1; // q0q1
        break;
  case 'y':
        state = 2; // qsa
        break;
      default:
        return false;
      }
}
else if(state == 4){// qt ************
      switch(s[charpos]){
      case 'a': case 'e':
  case 'i': case 'o':
      case 'u':
      case 'E': case 'I':
        state = 1; // q0q1
            break;
      case 's':
        state = 2; // qsa
            break;
      default:
        return false;
          }
    }
    else if(state == 5){// qs ************
          switch(s[charpos]){
          case 'a': case 'e':
      case 'i': case 'o':
      case 'u':
      case 'E': case 'I':
        state = 1; // q0q1
            break;
      case 'h':
        state = 2; // qsa
            break;
      default:
        return false;
      }
    }
    else if(state == 6){// qc ************
          switch(s[charpos]){
      case 'h':
        state = 2; // qsa
            break;
      default:
        return false;
      }
    }

    charpos++;
  }

if(state == 0 || state == 1)
  {return true;}
else{return false;}
```

```
  //}
  /* replace the following todo the word dfa  **
  while (s[charpos] != '\0')
    {
      if (state == 0 && s[charpos] == 'a')
      state = 1;
      else
      if (state == 1 && s[charpos] == 'b')
      state = 2;
      else
      if (state == 2 && s[charpos] == 'b')
      state = 2;
      else
              return(false);
      charpos++;
    }//end of while

  // where did I end up????
  if (state == 2) return(true);  // end in a final state
   else return(false);
  */
}

// PERIOD DFA
// Done by: ** Paolo Rimando
bool period (string s)
{  // complete this **
  if(s == "."){
    return true;
  }else{
    return false;
  }
}

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

// TABLES Done by: Ralph Lira

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

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

// ** 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[30][2] = {
  {"masu", ""},
  {"masen", ""},
  {"mashita", ""},
  {"masendeshita", ""},
  {"desu", ""},
  {"deshita", ""},
  {"o", ""},
  {"wa", ""},
  {"ni", ""},
  {"watashi", "I/me"},
  {"anata", "you"},
  {"kare", "he/him"},
  {"kanojo", "she/her"},
  {"sore", "it"},
  {"mata", "Also"},
  {"soshite", "Then"},
  {"shikashi", "However"},
  {"dakara", "Therefore"},
  {"eofm", ""}

};

tokentype reservedWordsType[30] = {
  VERB,
  VERBNEG,
  VERBPAST,
  VERBPASTNEG,
  IS,
  WAS,
  OBJECT,
  SUBJECT,
  DESTINATION,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  CONNECTOR,
  CONNECTOR,
  CONNECTOR,
  CONNECTOR,
  EOFM
};

// ------------ Scaner 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: Esai Delgado
int scanner(tokentype& tt, string& w)
{
  bool reserved = false;
  // ** Grab the next word from the file via fin
  // 1. If it is eofm, return right now.
  fin >> w;
  if(w == "eofm")
    return 0;

  /*  **
  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(period(w))
```

```
    tt = PERIOD;
  else if(word(w))
    {
      for(int i = 0; i < 30; i++)
            {
              if(w == reservedWords[i][0])
                {
                  tt = reservedWordsType[i];
                  reserved = true;
                }
            }
      if(!reserved)
            {
              if(w[w.length() - 1] == 'I' || w[w.length() - 1] == 'E')
                tt = WORD2;
              else
                tt = WORD1;
            }
    }
  else
    {
      cout << "LEXICAL ERROR: " << w << " is not a valid token" << endl;
      tt = ERROR;
    }
  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:  Rika
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
```

**3 - Original scanner test results**

**Test 1 - with no lexical errors**

```
Script started on Wed 11 Dec 2019 09:06:55 PM PST
]0;lira012@empress:~/CS421Progs/ScannerFiles [?1034h[lira012@empress ScannerFiles]$
g++ scanner.cpp
]0;lira012@empress:~/CS421Progs/ScannerFiles [lira012@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;lira012@empress:~/CS421Progs/ScannerFiles [lira012@empress ScannerFiles]$ ^C
]0;lira012@empress:~/CS421Progs/ScannerFiles [lira012@empress ScannerFiles]$ exit
exit

Script done on Wed 11 Dec 2019 09:07:25 PM PST
```

**Test 2 - with all kinds of lexical errors**

```
Script started on Wed 11 Dec 2019 09:07:35 PM PST
]0;lira012@empress:~/CS421Progs/ScannerFiles [?1034h[lira012@empress ScannerFiles]$
ex./a.oug++ scanner.cpexitg++ scanner.cpp
]0;lira012@empress:~/CS421Progs/ScannerFiles [lira012@empress ScannerFiles]$ g++
scanner.cpexit./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
LEXICAL ERROR: Osaka is not a valid token
Type is:ERROR
Word is:Osaka
Type is:WORD1
Word is:choucho
LEXICAL ERROR: butterfly is not a valid token
Type is:ERROR
Word is:butterfly
Type is:WORD1
Word is:an
Type is:WORD1
Word is:idea
Type is:WORD1
Word is:yasashii
LEXICAL ERROR: easy is not a valid token
Type is:ERROR
Word is:easy
```

```
Type is:WORD1
Word is:muzukashii
LEXICAL ERROR: difficult is not a valid token
Type is:ERROR
Word is:difficult
Type is:WORD1
Word is:ureshii
LEXICAL ERROR: pleased is not a valid token
Type is:ERROR
Word is:pleased
Type is:WORD1
Word is:shiawase
LEXICAL ERROR: happy is not a valid token
Type is:ERROR
Word is:happy
Type is:WORD1
Word is:kanashii
LEXICAL ERROR: sad is not a valid token
Type is:ERROR
Word is:sad
Type is:WORD1
Word is:omoi
LEXICAL ERROR: heavy is not a valid token
Type is:ERROR
Word is:heavy
Type is:WORD1
Word is:oishii
LEXICAL ERROR: delicious is not a valid token
Type is:ERROR
Word is:delicious
Type is:WORD1
Word is:tennen
LEXICAL ERROR: natural is not a valid token
Type is:ERROR
Word is:natural
Type is:WORD2
Word is:nakI
LEXICAL ERROR: cry is not a valid token
Type is:ERROR
Word is:cry
Type is:WORD2
Word is:ikI
LEXICAL ERROR: go* is not a valid token
```

```
Type is:ERROR
Word is:go*
Type is:WORD2
Word is:tabE
LEXICAL ERROR: eat is not a valid token
Type is:ERROR
Word is:eat
Type is:WORD2
Word is:ukE
LEXICAL ERROR: take* is not a valid token
Type is:ERROR
Word is:take*
Type is:WORD2
Word is:kakI
LEXICAL ERROR: write is not a valid token
Type is:ERROR
Word is:write
Type is:WORD2
Word is:yomI
LEXICAL ERROR: read is not a valid token
Type is:ERROR
Word is:read
Type is:WORD2
Word is:nomI
LEXICAL ERROR: drink is not a valid token
Type is:ERROR
Word is:drink
Type is:WORD2
Word is:agE
LEXICAL ERROR: give is not a valid token
Type is:ERROR
Word is:give
Type is:WORD2
Word is:moraI
LEXICAL ERROR: receive is not a valid token
Type is:ERROR
Word is:receive
Type is:WORD2
Word is:butsI
LEXICAL ERROR: hit is not a valid token
Type is:ERROR
Word is:hit
Type is:WORD2
```

```
Word is:kerI
LEXICAL ERROR: kick is not a valid token
Type is:ERROR
Word is:kick
Type is:WORD2
Word is:shaberI
LEXICAL ERROR: talk is not a valid token
Type is:ERROR
Word is:talk
End of file is encountered.
]0;lira012@empress:~/CS421Progs/ScannerFiles [lira012@empress ScannerFiles]$ exit
exit

Script done on Wed 11 Dec 2019 09:07:54 PM PST
```

**4 - Factored Rules with new non-terminal names and semantic routines**

Updated Factor Rules – Group #11

FACTORED:

1 <s> ::= [CONNECTOR #getEword #gen(CONNECTOR)#] <noun> #getEword# SUBJECT #getEword #gen(ACTOR)# <after subject>

2 <after subject> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD | <noun> #getEword# <after noun>

3 <after noun> ::= <be> #gen(DESCRIPTION)# #gen(TENSE)# PERIOD  | DESTINATION #gen(TO)# <verb>  #getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD | OBJECT #gen(OBJECT)# <after object>

4 <after object> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD | <noun> #getEword# DESTINATION #gen(TO)# <verb>#getEword# #gen(ACTION)# <tense>#gen(TENSE)# PERIOD

5 <noun> ::= WORD1 | PRONOUN

6 <verb> ::= WORD2

7 <be> ::=   IS | WAS

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

**5 - Updated parser code for translation (translator.cpp)**
**(*translator.cpp includes both scanner.cpp and parser.cpp*)**

```
#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.
*/

//=============================================================================
==================
//Japanese-to-English Translator Project done by: Paolo Rimando, Esai Delgado, Ralph Lira
//=============================================================================
==================

//=====================================================
//File scanner.cpp written by Group Number: **11
//=====================================================

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

// WORD DFA
// Done by: Paolo Rimando
/* 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;

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

      if(state == 0){// q0 ************
      switch(s[charpos]){
      case 'a': case 'e':
      case 'i': case 'o':
      case 'u':
      case 'E': case 'I':
        state = 1; // q010
        break;
      case 'd': case 'w':
  case 'y': case 'z':
  case 'j':
    state = 2; // qsa
    break;
      case 'b': case 'm':
  case 'k': case 'n':
  case 'h': case 'p':
  case 'r': case 'g':
    state = 3; // qy
    break;
      case 't':
        state = 4; // qt
        break;
      case 's':
        state = 5; // qs
        break;
      case 'c':
        state = 6; // qc
        break;
      default:
        return false;
      }
}
else if(state == 1){// q0q1 ************
      switch(s[charpos]){
      case 'n':
        state = 0; // q0
        break;
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        break; // stay q0q1
      case 'd': case 'w':
  case 'y': case 'z':
```

```
  case 'j':
        state = 2; // qsa
        break;
      case 'b': case 'm':
  case 'k':
  case 'h': case 'p':
  case 'r': case 'g':
        state = 3; // qy
        break;
      case 't':
        state = 4; // qt
        break;
  case 's':
        state = 5; // qs
        break;
  case 'c':
        state = 6; // qc
        break;
      default:
        return false;
      }
}
else if(state == 2){// qsa ************
      switch(s[charpos]){
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        state = 1; // q0q1
        break;
      default:
        return false;
      }
}
else if(state == 3){// qy ************
      switch(s[charpos]){
      case 'a': case 'e':
  case 'i': case 'o':
  case 'u':
  case 'E': case 'I':
        state = 1; // q0q1
        break;
  case 'y':
          state = 2; // qsa
          break;
        default:
          return false;
        }
  }
  else if(state == 4){// qt ************
        switch(s[charpos]){
        case 'a': case 'e':
    case 'i': case 'o':
    case 'u':
    case 'E': case 'I':
      state = 1; // q0q1
          break;
    case 's':
      state = 2; // qsa
          break;
    default:
      return false;
        }
  }
  else if(state == 5){// qs ************
        switch(s[charpos]){
        case 'a': case 'e':
    case 'i': case 'o':
    case 'u':
    case 'E': case 'I':
      state = 1; // q0q1
          break;
    case 'h':
      state = 2; // qsa
          break;
    default:
      return false;
    }
  }
  else if(state == 6){// qc ************
        switch(s[charpos]){
    case 'h':
      state = 2; // qsa
          break;
    default:
      return false;
      }
    }

    charpos++;
  }

  if(state == 0 || state == 1)
    {return true;}
  else{return false;}

}

// PERIOD DFA
// Done by: ** Paolo Rimando
bool period (string s)
{  // complete this **
  if(s == "."){
    return true;
  }else{
    return false;
  }
}

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

// TABLES Done by: Ralph Lira

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

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

// ** 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[30][2] = {
  {"masu", ""},
  {"masen", ""},
  {"mashita", ""},
  {"masendeshita", ""},
  {"desu", ""},
  {"deshita", ""},
  {"o", ""},
  {"wa", ""},
  {"ni", ""},
  {"watashi", "I/me"},
  {"anata", "you"},
  {"kare", "he/him"},
  {"kanojo", "she/her"},
  {"sore", "it"},
  {"mata", "Also"},
  {"soshite", "Then"},
  {"shikashi", "However"},
  {"dakara", "Therefore"},
  {"eofm", ""}

};

tokentype reservedWordsType[30] = {
  VERB,
  VERBNEG,
  VERBPAST,
  VERBPASTNEG,
  IS,
  WAS,
  OBJECT,
  SUBJECT,
  DESTINATION,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  PRONOUN,
  CONNECTOR,
  CONNECTOR,
  CONNECTOR,
  CONNECTOR,
  EOFM
```

```
};

// ------------ Scaner and Driver -----------------------

ifstream fin;  // global stream for reading from the input file
ofstream fout; // global stream for printing to translated.txt
bool display_trace = true;

// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// ** Done by: Esai Delgado
int scanner(tokentype& tt, string& w)
{
  bool reserved = false;
  // ** Grab the next word from the file via fin
  // 1. If it is eofm, return right now.
  fin >> w;
  if(display_trace)
    cout << "Scanner called using word: " << w << endl;
  if(w == "eofm")
    return 0;

  /*  **
  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(period(w))
    tt = PERIOD;
  else if(word(w))
   {
     for(int i = 0; i < 30; i++)
           {
             if(w == reservedWords[i][0])
               {
                 tt = reservedWordsType[i];
                 reserved = true;
               }
           }
     if(!reserved)
           {
             if(w[w.length() - 1] == 'I' || w[w.length() - 1] == 'E')
               tt = WORD2;
             else
               tt = WORD1;
           }
     }
  else
     {
     cout << endl << "LEXICAL ERROR: " << w << " is not a valid token" << endl;
     tt = ERROR;
     }
  return 0;

}//the end of scanner

//=================================================
// File parser.cpp written by Group Number: **11
//=================================================

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

// ** Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
//    to display syntax error messages as specified by me.

// Type of error: Match fails
// Done by: Ralph Lira
void syntaxerror1(tokentype token, string saved_lexeme)
{
  cout << endl << "SYNTAX ERROR: expected " << tokenName[token] << " but found " <<
saved_lexeme << endl;
  exit(1);
}
// Type of error: switch results into default
// Done by: Ralph Lira
void syntaxerror2(string saved_lexeme, string pfunc)
{
  cout << endl << "SYNTAX ERROR: unexpected " << saved_lexeme << " found in " << pfunc
<< endl;
  exit(1);
}

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

void s();
void after_subject();
void after_noun();
void after_object();
void noun();
void verb();
void be();
void tense();

void getEword();
void gen(string);

tokentype saved_token; //global var for the token the scanner returned
string saved_lexeme;//global var to save string returned from scanner
string saved_E_word;//global var to retrieve the english translation of word
bool token_available; //indicates whether we have saved a token to eat up or not

// Purpose: Look ahead to see what token comes next from the scanner
// Done by: Esai Delgado
tokentype next_token()
{
  if(!token_available)
    {
      scanner(saved_token, saved_lexeme);
      token_available = true;
    }
  return saved_token;
}

// Purpose: Checks and eats up the expected token
// Done by: Esai Delgado
bool match(tokentype expected)
{
  if(next_token() != expected)
    {
      syntaxerror1(expected, saved_lexeme);
    }
  else
    {
      if(display_trace)
            cout << "Matched " << tokenName[expected] << endl;
      token_available = false;
      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

// Grammar: ** <story> ::= <s> {<s>}
// Done by: ** Paolo Rimando
void story()
{
  cout << "Processing <story>" << endl << endl;
  s();
  while(true) //loop for <story> until told otherwise
    {
      switch(next_token()) //look ahead
            {
            case CONNECTOR:
            case WORD1:
            case PRONOUN:
              s();
              break;
            default: //next token is not start of another <story>
              cout << endl << "Successfully parsed <story>." << endl;
              return; //exit loop
            }
    }
}

// Grammar: <s> ::= [CONNECTOR #getEword #gen(CONNECTOR)#] <noun> #getEword#
SUBJECT #getEword #gen(ACTOR)# <after subject>
// Done by: Ralph Lira
void s()
```

```
{
  if(display_trace)
    cout << "Processing <s>" << endl;
  if(next_token()==CONNECTOR)
    {
      match(CONNECTOR);
      getEword();
      gen("CONNECTOR");
      }
  noun();
  getEword();
  match(SUBJECT);
  gen("ACTOR");
  after_subject();
}

//Grammar: <after_subject> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)#
PERIOD | <noun> #getEword# <after noun>
// Done by: Esai Delgado
void after_subject()
{
  if(display_trace)
    cout << "Processing <after_subject>" << endl;
  switch(next_token())
    {
    case WORD2: // if upcoming <verb>
      verb();
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      match(PERIOD);
      break;
    case WORD1:
    case PRONOUN: // if upcoming <noun>
      noun();
      getEword();
      after_noun();
      break;
    default: //next token not start of expected nonterminal
      syntaxerror2(saved_lexeme, "afterSubject");
    }
}
```

```
//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: Paolo Rimando
void after_noun()
{
  if(display_trace)
  cout << "Processing <after_noun>" << endl;
  switch(next_token())
    {
    case IS:
    case WAS://if upcoming <be>
      be();
      gen("DESCRIPTION");
      gen("TENSE");
      match(PERIOD);
      break;
    case DESTINATION://if upcoming DESTINATION
      match(DESTINATION);
      gen("TO");
      verb();
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      match(PERIOD);
      break;
    case OBJECT://if upcoming OBJECT
      match(OBJECT);
      gen("OBJECT");
      after_object();
      break;
    default:
      syntaxerror2(saved_lexeme, "afterNoun");
    }
}

//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: Ralph Lira
void after_object()
```

```
{
  if(display_trace)
  cout << "Processing <after_object>" << endl;
  switch(next_token())
    {
    case WORD2://if upcoming <verb>
      verb();
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      match(PERIOD);
      break;
    case WORD1:
    case PRONOUN://if upcoming <noun>
      noun();
      getEword();
      match(DESTINATION);
      gen("TO");
      verb();
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      match(PERIOD);
      break;
    default:
      syntaxerror2(saved_lexeme, "afterObject");
    }
}

//Grammar: <noun> ::= WORD1 | PRONOUN
//Done by: Esai Delgado
void noun()
{
  if(display_trace)
    cout << "Processing <noun>" << endl;
  switch(next_token())
    {
    case WORD1: //if upcoming WORD1
      match(WORD1);
      break;
    case PRONOUN: //if upcoming PRONOUN
```

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

//Grammar: <verb> ::= WORD2
//Done by: Paolo Rimando
void verb()
{
  if(display_trace)
    cout << "Processing <verb>" << endl;
  switch(next_token())
    {
    case WORD2:
      match(WORD2);
      break;
    default:
      syntaxerror2(saved_lexeme, "verb");
    }
}

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

//Grammer: <tense> ::= VERBPAST | VERBPASTNEG | VERB | VERBNEG
```

```
//Done by: Esai Delgado
void tense()
{
  if(display_trace)
    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(saved_lexeme, "tense");
    }
}

//=================================================
// File translator.cpp written by Group Number: **11
//=================================================

// ----- 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: Paolo Rimando
string lexicon[50][2];

// ** 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: Ralph Lira
void getEword()
{
  for(int x = 0; x < 50; x++)
    {
      if(lexicon[x][0] == saved_lexeme)
        {
          saved_E_word = lexicon[x][1];
              return;
        }
    }
  saved_E_word = saved_lexeme;
}
//    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: Esai Delgado
void gen(string line_type)
{
  if(line_type == "TENSE")//saved_token is generated for TENSE only
    {
      fout << line_type << ": " << tokenName[saved_token] << endl << endl;
    }
  else//saved_E_word is generated
    {
      fout << line_type << ": " << saved_E_word << endl;
    }
}

// ----- 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.


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

// The final test driver to start the translator
// Done by: Paolo Rimando
int main()
{
  //** opens the lexicon.txt file and reads it into Lexicon
```

```
//** closes lexicon.txt
//ifstream fin;
fin.open("lexicon.txt");

string word;
for(int i = 0; i < 50; i++)
{
  for(int j = 0; j < 2; j++)
    {
        fin >> word;
        if(fin.eof())
          break;
        lexicon[i][j] = word;
    }
}
fin.close();

//** opens the output file translated.txt
//ofstream fout;
fout.open("translated.txt");

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


//EC - tracing messages
char userin=0;
while(userin!= -1)
 {
  cout << "Display trace messages? (y/n): " << endl;
  cin.clear();
  cin.ignore(256,'\n');
  cin >> userin;
  switch(userin)
   {
  case 'Y':
  case 'y':
    //display_trace already true
    userin= -1;
    break;
  case 'N':
    case 'n':
      display_trace = false; // bool display_trace declared above scanner function
      userin= -1;
      break;
    default:
      cout << "Invalid input. ";
     }//end switch
   }//end of while

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

**6 - Final test results**

**Test 1 Results**

```
Script started on Wed 11 Dec 2019 09:39:44 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
g++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ ./a.out
Enter the input file name: partCtest1
Display trace messages? (y/n):
y
Processing <story>

Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: desu
Processing <be>
Matched IS
Scanner called using word: .
Matched PERIOD
Scanner called using word: watashi
Processing <s>
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: sensei
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: desu
Processing <be>
Matched IS
Scanner called using word: .
```

```
Matched PERIOD
Scanner called using word: rika
Processing <s>
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: gohan
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: o
Matched OBJECT
Processing <after_object>
Scanner called using word: tabE
Processing <verb>
Matched WORD2
Processing <tense>
Scanner called using word: masu
Matched VERB
Scanner called using word: .
Matched PERIOD
Scanner called using word: watashi
Processing <s>
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: tesuto
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: o
Matched OBJECT
Processing <after_object>
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
Scanner called using word: shikashi
Processing <s>
Matched CONNECTOR
Processing <noun>
Scanner called using word: seito
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: yorokobI
Processing <verb>
Matched WORD2
Processing <tense>
Scanner called using word: masendeshita
Matched VERBPASTNEG
Scanner called using word: .
Matched PERIOD
Scanner called using word: dakara
Processing <s>
Matched CONNECTOR
Processing <noun>
Scanner called using word: watashi
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: kanashii
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: deshita
Processing <be>
Matched WAS
Scanner called using word: .
Matched PERIOD
Scanner called using word: soshite
```

```
Processing <s>
Matched CONNECTOR
Processing <noun>
Scanner called using word: rika
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: toire
Processing <noun>
Matched WORD1
Processing <after_noun>
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
Scanner called using word: rika
Processing <s>
Processing <noun>
Matched WORD1
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: nakI
Processing <verb>
Matched WORD2
Processing <tense>
Scanner called using word: mashita
Matched VERBPAST
Scanner called using word: .
Matched PERIOD
Scanner called using word: eofm

Successfully parsed <story>.
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:40:15 PM PST
```

**Test 2 Results**

```
Script started on Wed 11 Dec 2019 09:40:22 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
ex./a.oug++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ g++
translator.cexitg++ translator.c[K ./a.out
Enter the input file name: partCtest2
Display trace messages? (y/n):
y
Processing <story>

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 <after_subject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: desu
Processing <be>
Matched IS
Scanner called using word: ne

SYNTAX ERROR: expected PERIOD but found ne
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:40:44 PM PST
```

## Test 3 Results

```
Script started on Wed 11 Dec 2019 09:40:49 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
ex./a.oug++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ g++
translator.cexitg++ translator.c[K  ./a.out
Enter the input file name: partCtest3
Display trace messages? (y/n):
y
Processing <story>

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;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:41:31 PM PST
```

**Test 4 Results**

```
Script started on Wed 11 Dec 2019 09:41:36 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
ex./a.oug++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ g++
translator.cexit./a.oug++ translator.c[11P./a.out
Enter the input file name: partCtest4
Display trace messages? (y/n):
y
Processing <story>

Processing <s>
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
Scanner called using word: wa
Matched SUBJECT
Processing <after_subject>
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <after_noun>
Scanner called using word: mashita

SYNTAX ERROR: unexpected mashita found in afterNoun
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:41:57 PM PST
```

**Test 5 Results**

```
Script started on Wed 11 Dec 2019 09:42:03 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
ex./a.oug++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ g++
translator.cexit./a.out
Enter the input file name: partCtest5
Display trace messages? (y/n):
y
Processing <story>

Processing <s>
Scanner called using word: wa
Processing <noun>

SYNTAX ERROR: unexpected wa found in noun
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:42:25 PM PST
```

**Test 6 Results**

```
Script started on Wed 11 Dec 2019 09:42:32 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
ex./a.oug++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ g++
translator.cexit./a.out
Enter the input file name: partCtest6
Display trace messages? (y/n):
y
Processing <story>

Processing <s>
Scanner called using word: apple

LEXICAL ERROR: apple is not a valid token
Processing <noun>

SYNTAX ERROR: unexpected apple found in noun
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit
Script done on Wed 11 Dec 2019 09:42:51 PM PST
```

**Extra Credit Test 1**

```
Script started on Wed 11 Dec 2019 09:57:40 PM PST
]0;lira012@empress:~/CS421Progs/TranslatorFiles [?1034h[lira012@empress TranslatorFiles]$
g++ translator.cpp
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ ./a.out
Enter the input file name: partCtest1
Display trace messages? (y/n):
n
Processing <story>


Successfully parsed <story>.
]0;lira012@empress:~/CS421Progs/TranslatorFiles [lira012@empress TranslatorFiles]$ exit
exit

Script done on Wed 11 Dec 2019 09:58:04 PM PST
```