Section: 0- Cover sheet with the title, your group number and names.

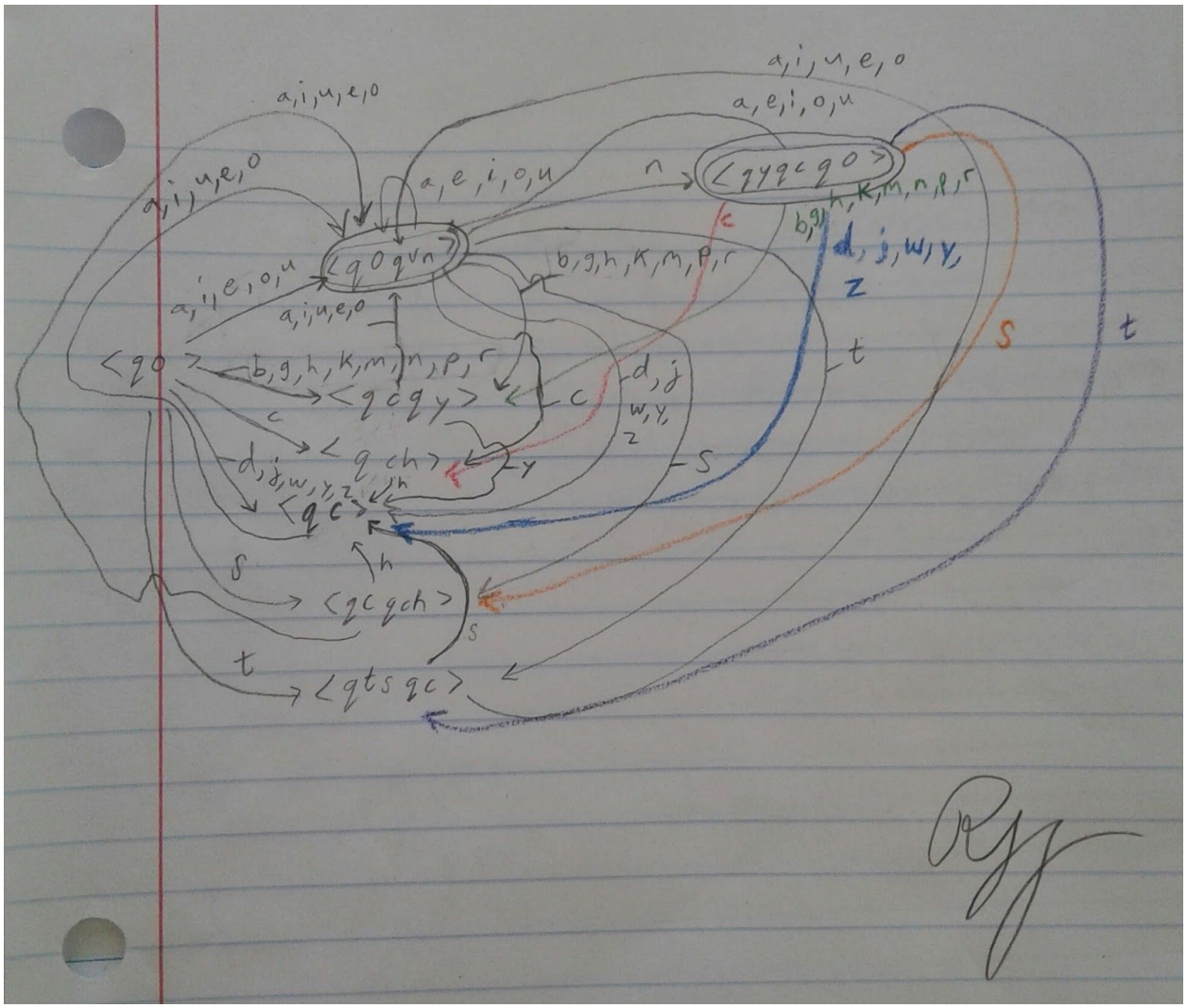
CS421 Report
Group 13

Edgar Cruz
Takuro Iwane
Jesus Rivera

Section 0- State of the program statement

- working perfectly? Yes
- any parts you did not complete? list them. We completed everything
- any bugs? list them. No bugs
- What extra credit features did you implement? Give details. No extra credit features were implemented

Section 1: DFA



Section 2- Scanner Code that match your DFAs

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

//=================================================
// File scanner.cpp written by: Group Number: 13
//=================================================

// ** MYTOKEN DFA to be replaced by the WORD DFA
// ** Done by: Jesus Rivera
// ** RE: (vowel | vowel n| consonant vowel| consonant vowel n| consonant-pair vowel | consonant-pair vowel n)^+
/*
PURPOSE: Checks to see if a string is a Japanese word.
PARAMETER: An s variable of type string.
ALGORITHM: Gets a string and checks it character by character. If it ends
          up on state 6 or 7 return true else false is returned.
*/
bool myword(string s)
{
  int state = 0;
  int charpos = 0;
  //  cout << "Inside of the myword function" << endl;
  while (s[charpos] != '\0')
  {
     if (state == 0 && (s[charpos] == 'a'||s[charpos] == 'e' || s[charpos] == 'E'|| s[charpos] == 'i' || s[charpos] == 'I' || s[charpos]
== 'o' || s[charpos] == 'u'))
     state = 6;
     else
     if (state == 0 && (s[charpos] == 'b'||s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm'||
s[charpos] == 'n' || s[charpos] == 'p'|| s[charpos] == 'r'))
     state = 5;
     else
     if (state == 0 && (s[charpos] == 'c'))
     state = 4;
     else
     if (state == 0 && (s[charpos] == 'd'||s[charpos] == 'j' || s[charpos] == 'w' || s[charpos] == 'y' || s[charpos] == 'z'))
     state = 3;
     else
     if (state == 0 && (s[charpos] == 's'))
              state = 2;
     else
     if (state == 0 && (s[charpos] == 't'))
             state = 1;
     else
     if (state == 6 && (s[charpos] == 'a'||s[charpos] == 'e'||s[charpos] == 'E' || s[charpos] == 'i' || s[charpos] == 'I'|| s[charpos]
== 'o' || s[charpos] == 'u'))
             state = 6;
     else
     if (state == 6  && (s[charpos] == 'n'))
             state = 7;
```

```
    else
    if (state == 6  && (s[charpos] == 't'))
            state = 1;
    else
    if (state == 6 && (s[charpos] == 's'))
            state = 2;
    else
    if (state == 6 && (s[charpos] == 'b'||s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm'||
s[charpos] == 'p' || s[charpos] == 'r'))
      state = 5;
    else
    if (state == 6 && (s[charpos] == 'd'||s[charpos] == 'j' || s[charpos] == 'w' || s[charpos] == 'y' || s[charpos] == 'z'))
            state = 3;
    else
    if (state == 6 && (s[charpos] == 'c'))
            state = 4;
    else
    if (state == 7 && (s[charpos] == 'a'||s[charpos] == 'e'||s[charpos] == 'E'||s[charpos] == 'i' || s[charpos] == 'I'|| s[charpos] ==
'o' || s[charpos] == 'u'))
            state = 6;
    else
    if (state == 7 && (s[charpos] == 't'))
            state = 1;
    else
    if (state == 7 && (s[charpos] == 's'))
      state =2;
    else
    if (state == 7 && (s[charpos] == 'd'||s[charpos] == 'j' || s[charpos] == 'w' || s[charpos] == 'y' || s[charpos] == 'z'))
      state = 3;
    else
    if (state == 7 && (s[charpos] == 'b'||s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm'||s[charpos]
== 'n' || s[charpos] == 'p' || s[charpos] == 'r'))
      state = 5;
    else
    if (state == 7 && (s[charpos] == 'c'))
            state = 4;
    else
    if (state == 1 && (s[charpos] == 'a'||s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u'))
      state = 6;
    else
    if (state == 2 && (s[charpos] == 'a'||s[charpos] == 'e' || s[charpos] == 'E'||s[charpos] == 'i' || s[charpos] == 'I'|| s[charpos]
== 'o' || s[charpos] == 'u'))
      state = 6;
    else
    if (state == 3 && (s[charpos] == 'a'||s[charpos] == 'e' ||s[charpos] == 'E'|| s[charpos] == 'i'||s[charpos] == 'I' || s[charpos]
== 'o' || s[charpos] == 'u'))
      state = 6;
    else
    if (state == 1 && (s[charpos] == 's'))
            state = 3;
    else
    if (state == 2 && (s[charpos] == 'h'))
            state = 3;
    else
```

```
    if (state == 4 && (s[charpos] == 'h'))
            state = 3;
    else
    if (state == 5 && (s[charpos] == 'y'))
            state = 3;
    else
    if (state == 5 && (s[charpos] == 'a' ||s[charpos] == 'e' ||s[charpos] == 'E'|| s[charpos] == 'i'||s[charpos] == 'I' || s[charpos]
== 'o' || s[charpos] == 'u'))
            state = 6;
    else
    {
            //              cout << "I am stuck in state " << state << endl;
              return false;
    }
    charpos++;
   }//end of while

  // where did I end up????
  //cout << "The final state is " << state << endl;
  if (state == 6 || state ==  7)
    return true;  // end in a final state
  else
    return false;
}

// ** Add the PERIOD DFA here
// ** Done by: Jesus Rivera
// ** RE: .^+
/*
PURPOSE: Checks to see if a string is a period.
PARAMETER: An s variable of type string.
ALGORITHM: Gets a string and checks it character by character. If it ends
          up on state 1 (the final state) return true else false is
          returned.
*/
bool period(string s)
{
  int state = 0;
  int charpos = 0;
  //cout << "Inside of the period function" << endl;
  //cout << "What is inside s[charpos] " << s[charpos] << endl;
  while (s[charpos] != '\0')
  {
    if (state == 0 && s[charpos] == '.')
      state = 1;
    else
    if (state == 1 && s[charpos] == '.')
      state = 1;
    else
    {
            //              cout << "I am stuck in state " << state << endl;
      return false;
    }
    charpos++;
```

```
 }//end of while
 //cout << "State in period function is " << state << endl;
 if (state == 1)
    return true;
 else return false;

}

// ** Update the tokentype to be WORD1, WORD2, PERIOD and ERROR.
 enum tokentype {VERB, VERBNEG, VERBPAST, VERBPASTNEG, IS, WAS, OBJECT, SUBJECT, DESTINATION,
PRONOUN, CONNECTOR, WORD1, WORD2, PERIOD, ERROR};
enum englishtype {I, Me, You, He, Him, She, Her, It, Also, Then, However,\
 Therefore
};
// ** Need the lexicon to be set up here (to be used in Part C)
// ** Done by: Takuro

/*
PURPOSE: Checks to see if a string is a lexicon word.
PARAMETER: A variable of type englishtype and a variable of type
          string.
ALGORITHM: Gets a string and compares it with the lexicon words. If the
          string matches one, a type is assigned to it and it returns
          true. If the string doesn't match any of the lexicon words
          false is returned.
*/
bool lexiconWord(englishtype& english, string word)
{
 if(word == "watashi")
   {
     english = I;
     return true;
   }
 else
 if(word == "anata")
 {
    english = You;
    return true;
 }
 else
 if(word == "kare")
 {
    english = He;
    return true;
 }
 else
 if(word == "kanojo")
  {
    english = She;
    return true;
  }
 else
 if(word == "sore")
 {
```

```
    english = It;
    return true;
  }
  else
  if(word == "mata")
  {
    english = Also;
    return true;
  }
  else
  if(word == "soshite")
  {
    english = Then;
    return true;
  }
  else
  if(word == "shikashi")
  {
    english = However;
    return true;
  }
  else
  if(word == "dakara")
  {
    english = Therefore;
    return true;
  }
  else
    return false;
}

// ** Need the reservedwords list to be set up here
// ** Done by: Edgar Cruz
/*
PURPOSE: Checks to see if a string is a reserved word.
PARAMETER: A type variable of type tokentype and a word variable of type
          string.
ALGORITHM: Gets a string and compares it with the reserved words. If the
          string matches one, a type is assigned to it and it returns
          true. If the string doesn't match any of the reserved words
          false is returned.
*/
bool reservedWord(tokentype& type, string word)
{
  //cout << "Inside reservedWord function" << endl;
  if (word == "masu")
  {
    type = VERB;
    return true;
  }
  else
  if (word == "masen")
  {
    type = VERBNEG;
```

```
  return true;
}
else
if (word == "mashita")
{
  type = VERBPAST;
  return true;
}
else
if(word == "masendeshita")
{
  type = VERBPASTNEG;
  return true;
}
else
if (word == "desu")
{
  type = IS;
  return true;
}
else
if (word == "deshita")
{
  type = WAS;
  return true;
}
else
if (word == "o")
{
  type = OBJECT;
  return true;
}
else
if (word == "wa")
{
  type = SUBJECT;
  return true;
}
else
if (word == "ni")
{
  type = DESTINATION;
  return true;
}
else
if (word == "watashi"|| word == "anata"|| word == "kare"|| word == "kanojo" || word == "sore")
{
type = PRONOUN;
return true;
}
else
if (word == "mata" || word == "soshite" || word == "shikashi" || word == "dakara")
  {
  type = CONNECTOR;
```

```
   return true;
   }
  else
  {
    //cout << "not a reserved word" << endl;
  return(false);
  }
}

// ** Do not require any file input for these.
// ** a.out should work without any additional files.

// Scanner processes only one word each time it is called
// ** Done by: Edgar Cruz
/*
PURPOSE: To determine a strings type.
PARAMETER: Provide a tokentype, a string, and an ifstream variable that
          will be passed by reference.
ALGORITHM: A string is grabbed from the file fin. The scanner function then
          checks to see if the string is a Japanese word by passing
          it to the myword function.
          If it is a word, it is then checked to see if it is a
          reserved word by calling the reservedWord function. If it is not
          a reserved word it goes through a while loop to determine
          if it is WORD1 or WORD2. If the string is not a word it is
          tested by calling the period function. If it's not a
          period then ERROR is returned as the tokentype.
*/
void scanner(tokentype& a, string& w, ifstream& fin)
{
  //  cout << ".....Scanner was called..." << endl;
  // ** Grab the next word from the file

      fin >> w;
      // cout << "Word is: " << w << endl;

  if (myword(w))
  {
    //cout << w << " is a word" << endl;

        if (reservedWord(a, w))
              ;
        else
        {
          int pos = 0;
                while (w[pos] != '\0')
                {
                  if (w[pos] == 'I'||w[pos] == 'E')
            a = WORD2;
            else
            a = WORD1;
            pos++;
          }
       }
```

```
  }
  else
  if (period(w))
     a = PERIOD;

  else
  if (w == "eofm")
      a = ERROR;
  else //none of the FAs returned TRUE
  {
       cout << "Lexical Error: " << w << " is not a valid token" << endl;
       a = ERROR;
  }

 /*
 2. Call the token functions one after another (if-then-else)
    And generate a lexical error if both DFAs failed.
    Let the token_type be ERROR in that case.
 3. Make sure WORDs are checked against the reservedwords list
    If not reserved, token_type is WORD1 or WORD2.
 4. Return the token type & string  (pass by reference)
 */
}//the end

// The test driver to call the scanner repeatedly
// ** Done by: Edgar Cruz  **
/*
PURPOSE: Checks to see what tokentype a string is and displays it.
ALGORITHM: An array is created called tokens with 15 different
          tokentypes. Inside of a while loop the scanner fuction is
          called and and 3 arguments are passed to it (a tokentype variable,
          a string variable, and an ifstream variable). Then the word
          and its tokentype are displayed.
*/

int main()
{
  string tokens[15] = {"VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT",
"DESTINATION", "PRONOUN", "CONNECTOR","WORD1","WORD2","PERIOD","ERROR"};
  tokentype thetype;
  string theword;

  /*
1. get the input file name from the user
2. open the input file which contains a story written in Japanese (fin.open).
3. call Scanner repeatedly until the EOF marker is read, and
     each time cout the returned results
     e.g. STRING TOKEN-TYPE
          =====  ============
          watashi PRONOUN  (from the first call)
          wa      SUBJECT  (from the second call)
          gakkou  WORD1
          etc.
  */
```

```
  ifstream fin;
  string userInput;

  cout << "Enter scannertest1 or scannertest2: ";
  getline(cin, userInput);

  fin.open(userInput.c_str());

  while (true)
  {
    scanner(thetype, theword, fin);  // call the scanner
    if(theword == "eofm") break;

    cout << "Word is: " << theword << " "
         << "Token type is: " << tokens[thetype]<< endl << endl;


       // ** display the actual type instead of a number
  }
  // ** close the input file

   fin.close();

}// end
```

Section 3: Scanner Test Results

Script started on Thu 14 Dec 2017 08:31:32 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz076@empress ScannerFiles]$ emacs scanner.lsemacs scanner.[Kg++ scanner.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz076@empress ScannerFiles]$ ./a.out
Enter scannertest1 or scannertest2: scannertest1
Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: rika Token type is: WORD1

Word is: desu Token type is: IS

Word is: . Token type is: PERIOD

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: sensei Token type is: WORD1

Word is: desu Token type is: IS

Word is: . Token type is: PERIOD

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: ryouri Token type is: WORD1

Word is: o Token type is: OBJECT

Word is: yarI Token type is: WORD2

Word is: masu Token type is: VERB

Word is: . Token type is: PERIOD

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: gohan Token type is: WORD1

Word is: o Token type is: OBJECT

Word is: seito Token type is: WORD1

Word is: ni Token type is: DESTINATION

Word is: agE Token type is: WORD2

Word is: mashita Token type is: VERBPAST

Word is: . Token type is: PERIOD

Word is: shikashi Token type is: CONNECTOR

Word is: seito Token type is: WORD1

Word is: wa Token type is: SUBJECT

Word is: yorokobI Token type is: WORD2

Word is: masendeshita Token type is: VERBPASTNEG

Word is: . Token type is: PERIOD

Word is: dakara Token type is: CONNECTOR

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: kanashii Token type is: WORD1

Word is: deshita Token type is: WAS

Word is: . Token type is: PERIOD

Word is: soshite Token type is: CONNECTOR

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: toire Token type is: WORD1

Word is: ni Token type is: DESTINATION

Word is: ikI Token type is: WORD2

Word is: mashita Token type is: VERBPAST

Word is: . Token type is: PERIOD

Word is: watashi Token type is: PRONOUN

Word is: wa Token type is: SUBJECT

Word is: nakI Token type is: WORD2

Word is: mashita Token type is: VERBPAST

Word is: . Token type is: PERIOD

]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz076@empress ScannerFiles]$ exit
exit

Script done on Thu 14 Dec 2017 08:32:35 PM PST

```
Script started on Thu 14 Dec 2017 08:44:39 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz085@empress ScannerFiles]$ g++ scanner.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz085@empress ScannerFiles]$ ./a.out
Enter scannertest1 or scannertest2: scannertest2
Word is: daigaku Token type is: WORD1

Lexical Error: college is not a valid token
Word is: college Token type is: ERROR

Word is: kurasu Token type is: WORD1

Lexical Error: class is not a valid token
Word is: class Token type is: ERROR

Word is: hon Token type is: WORD1

Lexical Error: book is not a valid token
Word is: book Token type is: ERROR

Word is: tesuto Token type is: WORD1

Lexical Error: test is not a valid token
Word is: test Token type is: ERROR

Word is: ie Token type is: WORD1

Lexical Error: home* is not a valid token
Word is: home* Token type is: ERROR

Word is: isu Token type is: WORD1

Lexical Error: chair is not a valid token
Word is: chair Token type is: ERROR

Word is: seito Token type is: WORD1

Lexical Error: student is not a valid token
Word is: student Token type is: ERROR

Word is: sensei Token type is: WORD1

Lexical Error: teacher is not a valid token
Word is: teacher Token type is: ERROR

Word is: tomodachi Token type is: WORD1

Lexical Error: friend is not a valid token
Word is: friend Token type is: ERROR

Word is: jidoosha Token type is: WORD1

Lexical Error: car is not a valid token
Word is: car Token type is: ERROR

Word is: gyuunyuu Token type is: WORD1

Lexical Error: milk is not a valid token
```

Word is: milk Token type is: ERROR

Word is: sukiyaki Token type is: WORD1

Word is: tenpura Token type is: WORD1

Word is: sushi Token type is: WORD1

Word is: biiru Token type is: WORD1

Lexical Error: beer is not a valid token
Word is: beer Token type is: ERROR

Word is: sake Token type is: WORD1

Word is: tokyo Token type is: WORD1

Word is: kyuushuu Token type is: WORD1

Lexical Error: Osaka is not a valid token
Word is: Osaka Token type is: ERROR

Word is: choucho Token type is: WORD1

Lexical Error: butterfly is not a valid token
Word is: butterfly Token type is: ERROR

Word is: an Token type is: WORD1

Word is: idea Token type is: WORD1

Word is: yasashii Token type is: WORD1

Lexical Error: easy is not a valid token
Word is: easy Token type is: ERROR

Word is: muzukashii Token type is: WORD1

Lexical Error: difficult is not a valid token
Word is: difficult Token type is: ERROR

Word is: ureshii Token type is: WORD1

Lexical Error: pleased is not a valid token
Word is: pleased Token type is: ERROR

Word is: shiawase Token type is: WORD1

Lexical Error: happy is not a valid token
Word is: happy Token type is: ERROR

Word is: kanashii Token type is: WORD1

Lexical Error: sad is not a valid token
Word is: sad Token type is: ERROR

Word is: omoi Token type is: WORD1

Lexical Error: heavy is not a valid token
Word is: heavy Token type is: ERROR

Word is: oishii Token type is: WORD1

Lexical Error: delicious is not a valid token
Word is: delicious Token type is: ERROR

Word is: tennen Token type is: WORD1

Lexical Error: natural is not a valid token
Word is: natural Token type is: ERROR

Word is: nakI Token type is: WORD2

Lexical Error: cry is not a valid token
Word is: cry Token type is: ERROR

Word is: ikI Token type is: WORD2

Lexical Error: go* is not a valid token
Word is: go* Token type is: ERROR

Word is: tabE Token type is: WORD2

Lexical Error: eat is not a valid token
Word is: eat Token type is: ERROR

Word is: ukE Token type is: WORD2

Lexical Error: take* is not a valid token
Word is: take* Token type is: ERROR

Word is: kakI Token type is: WORD2

Lexical Error: write is not a valid token
Word is: write Token type is: ERROR

Word is: yomI Token type is: WORD2

Lexical Error: read is not a valid token
Word is: read Token type is: ERROR

Word is: nomI Token type is: WORD2

Lexical Error: drink is not a valid token
Word is: drink Token type is: ERROR

Word is: agE Token type is: WORD2

Lexical Error: give is not a valid token
Word is: give Token type is: ERROR

Word is: moraI Token type is: WORD2

Lexical Error: receive is not a valid token

Word is: receive Token type is: ERROR

Word is: butsI Token type is: WORD2

Lexical Error: hit is not a valid token
Word is: hit Token type is: ERROR

Word is: kerI Token type is: WORD2

Lexical Error: kick is not a valid token
Word is: kick Token type is: ERROR

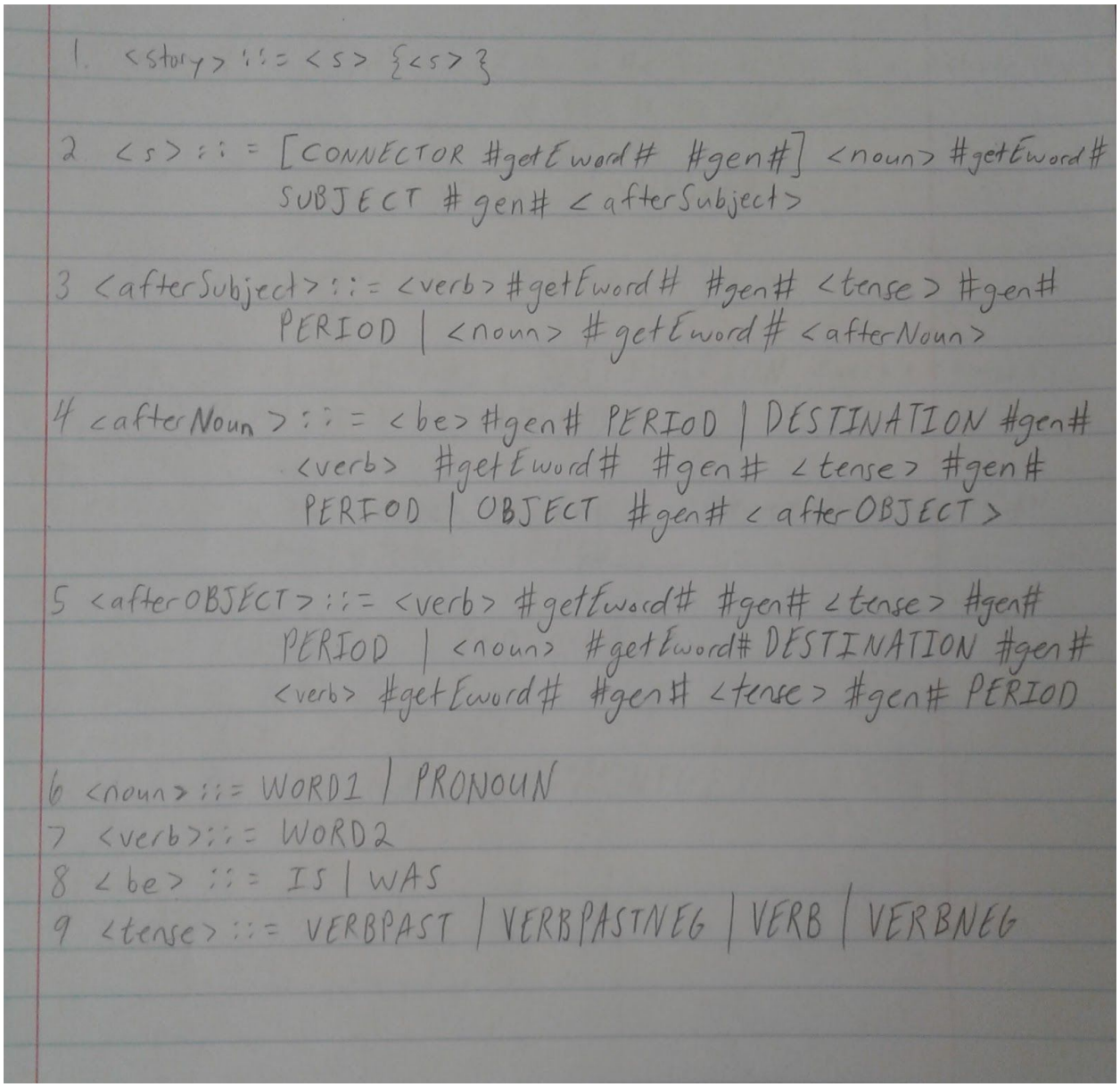
Word is: shaberI Token type is: WORD2

Lexical Error: talk is not a valid token
Word is: talk Token type is: ERROR

]0;cruz085@empress:~/cs421/CS421Progs/ScannerFiles[cruz085@empress ScannerFiles]$ exit
exit

Script done on Thu 14 Dec 2017 08:45:15 PM PST

Section 4: Factored Rules with new non-terminal names



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

2 <s> ::= [CONNECTOR #getEword# #gen#] <noun> #getEword# SUBJECT #gen# <afterSubject>

3 <afterSubject> ::= <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun> #getEword# <afterNoun>

4 <afterNoun> ::= <be> #gen# PERIOD | DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen# PERIOD | OBJECT #gen# <afterOBJECT>

5 <afterOBJECT> ::= <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun> #getEword# DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen# PERIOD

6 <noun> ::= WORD1 | PRONOUN
7 <verb> ::= WORD2
8 <be> ::= IS | WAS
9 <tense> ::= VERBPAST | VERBPASTNEG | VERB | VERBNEG

Section 5: Parser Code

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


tokentype saved_token;//  global buffer for the scanner token
bool token_available; //  global flag indicating whether we have
                      //  saved a token to eat up or not
ifstream fin;
string saved_lexeme;// global variable for the returned word from the scanner.
string savedEnglishWord;
string tokens[15] = {"VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT",
"DESTINATION", "PRONOUN", "CONNECTOR", "WORD1", "WORD2", "PERIOD", "ERROR"};

void s();
void afterSubject();
void afterNoun();
void afterObject();
void noun();
void verb();
void be();
void tense();
bool lexiconWord();



// ** Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
/*
** Done by: Edgar Cruz

PURPOSE: To generate an error message when match() fails.
PARAMETER: An expected variable of type tokentype and a lexeme variable
          of type string.
ALGORITHM: Displays an error message to the screen and exits the program.
*/
void syntaxerror1(tokentype expected, string lexeme)
{
  cout << "SYNTAX ERROR: expected " << tokens[expected] << " but found " << lexeme << endl;
  exit(1);
}

//** Done by: Edgar Cruz
/*
PURPOSE: To generate an error message when afterSubject(), afterNoun(),
```

```
        afterObject(), noun(), be(), or tense() fail.
PARAMETER: A lexeme variable of type string and a parserFunction variable
           of type string.
ALGORITHM: Displays an error message to the screen and exits the program.
*/
void syntaxerror2(string lexeme, string parserFunction)
{
  cout << "SYNTAX ERROR: unexpected " << lexeme << " found in " <<  parserFunction;
  exit(1);
}

//** Done by: Takuro Iwane
/*
PURPOSE: Looks ahead to see what token comes next from the scanner.
ALGORITHM: Checks to see if the global variable, token_available, is false.
           If it's false, scanner is called. The token type returned from
           the scanner is stored in saved_token. The word returned from the scanner
           is stored in lexeme, and fin has the input file that the user
           input. Token_available is set to true. The lexeme is stored in
           saved_lexeme. Returns the saved_token;
*/
tokentype next_token()
{
  string lexeme;
  if(!token_available)// if there is no saved token from the previous lookahead
    {
      cout << "Scanner was called....." << endl;
      scanner(saved_token, lexeme, fin);// call scanner to grab a new token.
                                        // saved_token is a global variable.
      //      cout << "Scanner called using word: " << lexeme << endl;
      if(saved_token == ERROR)
              cout << "Lexical Error" << endl;
      token_available = true;// mark the fact that you have saved it.
                             // token_available is a global variable.
      saved_lexeme = lexeme;
    }
  return saved_token;//return the saved token
}

//** Done by: Takuro Iwane
/*
PURPOSE: Checks and eats up the expected token.
PARAMETER: A tokentype variable called expected.
ALGORITHM: If saved_lexeme is eofm the program will exit. If next_token
           does not match the token type within the variable expected,
           syntaxerror1 is called and the program exits. Otherwise, Matched
           and the token type is displayed to the screen. Token_available
           is set to false and true is returned.
*/
bool match(tokentype expected)
{

  if(saved_lexeme == "eofm")
    exit(1);
```

```
  if(next_token() != expected)
    {
      syntaxerror1(expected, saved_lexeme);
      exit(1);
    }
  else
    {
      cout << "Matched " << tokens[expected] << endl; //display the matched token_type
      token_available = false;
      return 1;
    }
}

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

//** Done by: Edgar Cruz
/*
PURPOSE: <story> starts the parsing.
ALGORITHM: Checks to see if the next_token() matches one of token types. Otherwise,
          the program ends.
*/
//<story>::<s>{<s>}
void story()
{

  cout << "Processing Story" << endl << endl;
  s();
  while(true){
    switch(next_token())
      {
      case CONNECTOR: s();
              break;
      case WORD1: s();
              s();
              break;
      case PRONOUN: s();
              break;
      default:
              exit(1);;

      }// end of switch
  }//  end of loop

}//  end of story

//** Done by: Edgar Cruz
/*
PURPOSE: Continue parsing.
ALGORITHM: If next_token matches CONNECTOR, match(saved_token) is called.
          Otherwise, noun(),
          match(SUBJECT), and afterSubject() called.
*/
//<s>::= [CONNECTOR] <noun> SUBJECT <afterSubject>
```

```
void s()
{
  cout << "\n========Processing <s>========" << endl;
  if(next_token() == CONNECTOR)
    {
      match(saved_token);

    }
  noun();
  match(SUBJECT);
  afterSubject();

}//  end of s

//** Done by: Takuro Iwane
/*
PURPOSE: Next step in the parsing process
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          WORD2, WORD1, and PRONOUN. If the token type that next_token()
          returns doesn't match any of the cases then syntaxerror2 is
          called.
*/
//<afterSubject>::= <verb> <tense> PERIOD | <noun> <afterNoun>
void afterSubject()
{
  cout << "Processing <X>" << endl;
  switch(next_token())
    {
    case WORD2:
      verb();
      tense();
      match(PERIOD);
      break;
    case WORD1:
    case PRONOUN:
      noun();
      afterNoun();

      break;
    default: syntaxerror2(saved_lexeme, "afterSubject");
    }//  end of switch


}//end of afterSubject

//** Done by: Takuro Iwane
/*
PURPOSE: Continues the parsing process.
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          IS, WAS, DESTINATION, and OBJECT. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
```

```
        the cases then syntaxerror2 is called.
*/
//<afterNoun>::= <be> PERIOD | DESTINATION <verb>
//              <tense>  PERIOD | OBJECT  <afterOBJECT>
void afterNoun()
{
  cout << "Processing <Y>" << endl;
  switch(next_token())
    {
    case IS:
    case WAS:
      be();
      match(PERIOD);
      break;
    case DESTINATION:
      match(DESTINATION);
      verb();
      tense();
      match(PERIOD);
      break;
    case OBJECT:
      match(OBJECT);
      afterObject();
      break;
    default:
      syntaxerror2(saved_lexeme, "afterNoun");
    }//  end of switch

}//  end of afterNoun

//** Done by: Jesus Rivera
/*
PURPOSE: To continue the parsing process.
ALGORITHM: next_token() is called in a switch statement. next_token()
        returns a token type and it is compared against the cases
        WORD2, WORD1, and PRONOUN. If it finds a match with a
        case, it will call the functions in it and break. If the
        token type that next_token()returns doesn't match any of
        the cases then syntaxerror2 is called.
*/
//<afterOBJECT>:: <verb> <tense> PERIOD | <noun>
//              DESTINATION <verb> <tense> PERIOD
void afterObject()
{

  cout << "Processing <afterObject>" << endl;
  switch(next_token())
    {
    case WORD2:
      verb();
      tense();
      match(PERIOD);
      break;
    case WORD1:
```

```
    case PRONOUN:
      noun();
      match(DESTINATION);
      verb();
      tense();
      match(PERIOD);
      break;
    default: syntaxerror2(saved_lexeme, "afterObject");
    }//  end of switch

}//  end of afterObject

//** Done by: Jesus Rivera
/*
PURPOSE: Continues the parsing process
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          WORD1 and PRONOUN. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
*/
//<noun>::= WORD1 | PRONOUN
void noun()
{

  cout << "Processing <noun>" << endl;
  switch(next_token())
    {
    case WORD1:
      match(WORD1);
      break;
    case PRONOUN:
      match(PRONOUN);
      break;
    default: syntaxerror2(saved_lexeme, "noun");
    }//  end of switch

}// end of noun

//** Done by: Edgar Cruz
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <verb> to the screen.
          match(WORD2) is called.
*/
//<verb>::= WORD2
void verb()
{
  cout << "Processing <verb>" << endl;
  match(WORD2);
}//  end of verb

//** Done by: Edgar Cruz
```

```
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <be> to the screen.
          next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          IS and WAS. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
*/
//<be>::= IS|WAS
void be()
{

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

}//  end of be

//** Done by: Takuro Iwane
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <tense> to the screen.
          next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          VERBPAST, VERBPASTNEG, VERB, and VERBNEG. If it finds a match with a
          case, it will call the function in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
*/
//<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(saved_lexeme, "tense");
    }//  end of switch

}//  end of tense

// The test driver to start the parser
//** Done by: Jesus Rivera

/*
PURPOSE: To translate a Japanese word to an English word.
ALGORITHM: The user inputs the name of the file that contains
          the Japanese words. Open the input file and open
          the output file that will contain the
          Japanese words translated to English.
*/
int main()
{
  string userInput;
  cout << "Please enter a file name: ";
  getline(cin, userInput);
  //- opens the input file
  fin.open(userInput.c_str());

  //- calls the <story> to start parsing
  story();

  //- closes the input file
  fin.close();


}// end
```

Section 6: Parser Test Results

Test1

```
Script started on Thu 14 Dec 2017 10:12:34 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$
[K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$
[K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$
[K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$ [K[cruz085@empress ParserFiles]$
[K[cruz085@empress ParserFiles]$ g++ parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ./a.out
Please enter a file name: partBtest1
Processing Story


========Processing <s>========
Scanner was called.....
Processing <noun>
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Processing <be>
Matched IS
Scanner was called.....
Matched PERIOD
Scanner was called.....

========Processing <s>========
Processing <noun>
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Processing <be>
Matched IS
Scanner was called.....
Matched PERIOD
Scanner was called.....
```

========Processing <s>========
Processing <noun>
Matched WORD1
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Matched OBJECT
Processing <afterObject>
Scanner was called.....
Processing <verb>
Matched WORD2
Processing <tense>
Scanner was called.....
Matched VERB
Scanner was called.....
Matched PERIOD

========Processing <s>========
Scanner was called.....
Processing <noun>
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Matched OBJECT
Processing <afterObject>
Scanner was called.....
Processing <noun>
Matched WORD1
Scanner was called.....
Matched DESTINATION
Processing <verb>
Scanner was called.....
Matched WORD2
Processing <tense>
Scanner was called.....
Matched VERBPAST
Scanner was called.....
Matched PERIOD
Scanner was called.....

========Processing <s>========
Matched CONNECTOR
Processing <noun>
Scanner was called.....
Matched WORD1
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <verb>
Matched WORD2
Processing <tense>
Scanner was called.....
Matched VERBPASTNEG
Scanner was called.....
Matched PERIOD
Scanner was called.....

========Processing <s>========
Matched CONNECTOR
Processing <noun>
Scanner was called.....
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Processing <be>
Matched WAS
Scanner was called.....
Matched PERIOD
Scanner was called.....

========Processing <s>========
Matched CONNECTOR
Processing <noun>
Scanner was called.....
Matched WORD1
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>

Scanner was called.....
Matched DESTINATION
Processing <verb>
Scanner was called.....
Matched WORD2
Processing <tense>
Scanner was called.....
Matched VERBPAST
Scanner was called.....
Matched PERIOD
Scanner was called.....

========Processing <s>========
Processing <noun>
Matched WORD1
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <verb>
Matched WORD2
Processing <tense>
Scanner was called.....
Matched VERBPAST
Scanner was called.....
Matched PERIOD

========Processing <s>========
Scanner was called.....
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:13:13 PM PST

Test2

```
Script started on Thu 14 Dec 2017 10:14:27 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ex./a.oug++
parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ g++
parser.cexit./a.oug++ parser.c[7P./a.out
Please enter a file name: partBtest2
Processing Story


========Processing <s>========
Scanner was called.....
Matched CONNECTOR
Processing <noun>
Scanner was called.....
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....
Processing <be>
Matched IS
Scanner was called.....
SYNTAX ERROR: expected PERIOD but found ne
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:15:17 PM PST
```

Test 3

Script started on Thu 14 Dec 2017 10:16:08 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ex./a.oug++ parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ g++ parser.cexit./a.out
Please enter a file name: partBtest3
Processing Story

========Processing <s>========
Scanner was called.....
Matched CONNECTOR
Processing <noun>
Scanner was called.....
Matched PRONOUN
Scanner was called.....
SYNTAX ERROR: expected SUBJECT but found de
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ./a.ou[Kexit exit

Script done on Thu 14 Dec 2017 10:17:13 PM PST

Test 4

Script started on Thu 14 Dec 2017 10:17:21 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ex./a.oug++ parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ g++ parser.cexit./a.out
Please enter a file name: partBtest4
Processing Story

========Processing <s>========
Scanner was called.....
Processing <noun>
Matched PRONOUN
Scanner was called.....
Matched SUBJECT
Processing <X>
Scanner was called.....
Processing <noun>
Matched WORD1
Processing <Y>
Scanner was called.....

SYNTAX ERROR: unexpected mashita found in
afterNoun]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:17:59 PM PST

Test 5

Script started on Thu 14 Dec 2017 10:18:03 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ex./a.oug++
parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ g++
parser.cexit./a.out
Please enter a file name: partBtest5
Processing Story

========Processing <s>========
Scanner was called.....
Processing <noun>
SYNTAX ERROR: unexpected wa found in
noun]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:18:33 PM PST

Test 6

Script started on Thu 14 Dec 2017 10:18:41 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ ex./a.oug++
parser.cpp
]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ g++
parser.cexit./a.out
Please enter a file name: ^[[A^[partBtest6
Processing Story

========Processing <s>========
Scanner was called.....
Lexical Error
Processing <noun>
SYNTAX ERROR: unexpected apple found in
noun]0;cruz085@empress:~/cs421/CS421Progs/ParserFiles[cruz085@empress ParserFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:19:14 PM PST

Section 7- Updated Parser Code For Translation

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

//CS421 File translator.cpp

// ** Be sure to put the name of the programmer above each function
// ** Be sure to put the corresponding rule with semantic routines
//    above each function

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

tokentype saved_token;//  global buffer for the scanner token
bool token_available; //  global flag indicating whether we have
                      //  saved a token to eat up or not
ifstream fin;
ofstream fileOutput;
string saved_lexeme;// global variable for the returned word from the scanner.
string savedEnglishWord;
string tokens[15] = {"VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT",
"DESTINATION", "PRONOUN", "CONNECTOR", "WORD1", "WORD2", "PERIOD", "ERROR"};

void s();
void afterSubject();
void afterNoun();
void afterObject();
void noun();
void verb();
void be();
void tense();
bool lexiconWord();
void getEword();
void gen();
```

```
// ** Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
/*
** Done by: Edgar Cruz

PURPOSE: To generate an error message when match() fails.
PARAMETER: An expected variable of type tokentype and a lexeme variable
          of type string.
ALGORITHM: Displays an error message to the screen and exits the program.
 */
void syntaxerror1(tokentype expected, string lexeme)
{
  cout << "SYNTAX ERROR: expected " << tokens[expected] << " but found " << lexeme << endl;
  exit(1);
}

//** Done by: Edgar Cruz
/*
PURPOSE: To generate an error message when afterSubject(), afterNoun(),
        afterObject(), noun(), be(), or tense() fail.
PARAMETER: A lexeme variable of type string and a parserFunction variable
          of type string.
ALGORITHM: Displays an error message to the screen and exits the program.
 */
void syntaxerror2(string lexeme, string parserFunction)
{
  cout << "SYNTAX ERROR: unexpected " << lexeme << " found in " <<  parserFunction;
  exit(1);
}

//** Done by: Takuro Iwane
/*
PURPOSE: Looks ahead to see what token comes next from the scanner.
ALGORITHM: Checks to see if the global variable, token_available, is false.
          If it's false, scanner is called. The token type returned from
          the scanner is stored in saved_token. The word returned from the scanner
          is stored in lexeme, and fin has the input file that the user
          input. Token_available is set to true. The lexeme is stored in
          saved_lexeme. Returns the saved_token;
 */
tokentype next_token()
{
  string lexeme;
  if(!token_available)// if there is no saved token from the previous lookahead
    {
      cout << ".....Scanner was called....." << endl;
      scanner(saved_token, lexeme, fin);// call scanner to grab a new token.
                                  // saved_token is a global variable.
      cout << "Scanner called using word: " << lexeme << endl;
      if(saved_token == ERROR)
            cout << "Lexical Error" << endl;
      token_available = true;// mark the fact that you have saved it.
                        // token_available is a global variable.
      saved_lexeme = lexeme;
```

```
   }
  return saved_token;//return the saved token
}

//** Done by: Takuro Iwane
/*
PURPOSE: Checks and eats up the expected token.
PARAMETER: A tokentype variable called expected.
ALGORITHM: If saved_lexeme is eofm the program will exit. If next_token
          does not match the token type within the variable expected,
          syntaxerror1 is called and the program exits. Otherwise, Matched
          and the token type is displayed to the screen. Token_available
          is set to false and true is returned.
 */
bool match(tokentype expected)
{

  if(saved_lexeme == "eofm")
    exit(1);
  if(next_token() != expected)
    {
      syntaxerror1(expected, saved_lexeme);
      exit(1);
    }
  else
    {
      cout << "Matched " << tokens[expected] << endl; //display the matched token_type
      token_available = false;
      return 1;
    }
}

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

//** Done by: Edgar Cruz
/*
PURPOSE: <story> starts the parsing.
ALGORITHM: Checks to see if the next_token() matches one of token types. Otherwise,
          the program ends.
*/
//<story>::<s>{<s>}
void story()
{

  cout << "Processing Story" << endl << endl;
  s();
  while(true){
    switch(next_token())
      {
      case CONNECTOR: s();
              break;
      case WORD1: s();
              s();
```

```
            break;
      case PRONOUN: s();
            break;
      default:
            exit(1);;

     }// end of switch
  }//  end of loop

}//  end of story

//** Done by: Edgar Cruz
/*
PURPOSE: Continue parsing.
ALGORITHM: If next_token matches CONNECTOR, match(saved_token) is called.
          getEword() and gen() is called. Otherwise, noun(), getEword(),
          match(SUBJECT), gen(), and afterSubject() called.
 */
//<s>::= [CONNECTOR #getEword# #gen#] <noun> #getEword# SUBJECT #gen# <afterSubject>
void s()
{
  cout << "\n========Processing <s>========" << endl;
  if(next_token() == CONNECTOR)
    {
      match(saved_token);
      getEword();
      gen();
    }
  noun();
  getEword();
  match(SUBJECT);
  gen();
  afterSubject();

}//  end of s

//** Done by: Takuro Iwane
/*
PURPOSE: Next step in the parsing process
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          WORD2, WORD1, and PRONOUN. If the token type that next_token()
          returns doesn't match any of the cases then syntaxerror2 is
          called.
 */
//<afterSubject>::= <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun> #getEword# <afterNoun>
void afterSubject()
{
  cout << "Processing <X>" << endl;
  switch(next_token())
    {
    case WORD2:
      verb();
      getEword();
```

```
      gen();
      tense();
      gen();
      match(PERIOD);
      break;
    case WORD1:
    case PRONOUN:
      noun();
      getEword();
      afterNoun();
      //      s2();
      break;
    default: syntaxerror2(saved_lexeme, "afterSubject");// "s1");
    }//  end of switch


}//end of afterSubject

//** Done by: Takuro Iwane
/*
PURPOSE: Continues the parsing process.
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          IS, WAS, DESTINATION, and OBJECT. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
 */
//<afterNoun>::= <be> #gen# PERIOD | DESTINATION #gen# <verb> #getEword# #gen#
//              <tense> #gen# PERIOD | OBJECT #gen# <afterOBJECT>
void afterNoun()
{
  cout << "Processing <Y> <afterNoun>" << endl;
  switch(next_token())
    {
    case IS:
    case WAS:
      be();
      gen();
      match(PERIOD);
      break;
    case DESTINATION:
      match(DESTINATION);
      gen();
      verb();
      getEword();
      gen();
      tense();
      gen();
      match(PERIOD);
      break;
    case OBJECT:
      match(OBJECT);
      gen();
```

```
      afterObject();
      break;
    default:
      syntaxerror2(saved_lexeme, "afterNoun");
    }//  end of switch

}//  end of afterNoun

//** Done by: Jesus Rivera
/*
PURPOSE: To continue the parsing process.
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          WORD2, WORD1, and PRONOUN. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
 */
//<afterOBJECT>:: <verb> #getEword# #gen# <tense> #gen# PERIOD | <noun> #getEword#
//               DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen# PERIOD
void afterObject()
{

  cout << "Processing <afterObject>" << endl;
  switch(next_token())
    {
    case WORD2:
      verb();
      getEword();
      gen();
      tense();
      gen();
      match(PERIOD);
      break;
    case WORD1:
    case PRONOUN:
      noun();
      getEword();
      match(DESTINATION);
      gen();
      verb();
      getEword();
      gen();
      tense();
      gen();
      match(PERIOD);
      break;
    default: syntaxerror2(saved_lexeme, "afterObject");
    }//  end of switch

}//  end of afterObject

//** Done by: Jesus Rivera
/*
```

```
PURPOSE: Continues the parsing process
ALGORITHM: next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          WORD1 and PRONOUN. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
 */
//<noun>::= WORD1 | PRONOUN
void noun()
{

  cout << "Processing <noun>" << endl;
  switch(next_token())
    {
    case WORD1:
      match(WORD1);
      break;
    case PRONOUN:
      match(PRONOUN);
      break;
    default: syntaxerror2(saved_lexeme, "noun");
    }//  end of switch

}// end of noun

//** Done by: Edgar Cruz
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <verb> to the screen.
          match(WORD2) is called.
 */
//<verb>::= WORD2
void verb()
{
  cout << "Processing <verb>" << endl;
  match(WORD2);
}//  end of verb

//** Done by: Edgar Cruz
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <be> to the screen.
          next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          IS and WAS. If it finds a match with a
          case, it will call the functions in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
 */
//<be>::= IS|WAS
void be()
{
```

```
  cout << "Processing <be>" << endl;
  switch(next_token())
    {
    case IS:
      match(IS);
      break;
    case WAS:
      match(WAS);
      break;
    default: syntaxerror2(saved_lexeme, "be");
    }//  end of switch

}//  end of be

//** Done by: Takuro Iwane
/*
PURPOSE: Continues the parsing process
ALGORITHM: Displays processing <tense> to the screen.
          next_token() is called in a switch statement. next_token()
          returns a token type and it is compared against the cases
          VERBPAST, VERBPASTNEG, VERB, and VERBNEG. If it finds a match with a
          case, it will call the function in it and break. If the
          token type that next_token()returns doesn't match any of
          the cases then syntaxerror2 is called.
 */
//<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(saved_lexeme, "tense");
    }//  end of switch

}//  end of tense

//** Done by: Jesus Rivera
/*
PURPOSE: Gets the English Word from the Lexicon
ALGORITHM: Checks to see if the word in saved_lexeme matches the
          Japanes word in " ". When it finds a match, the English
          word for it is stored in a variable called
```

```
        savedEnglishWord. Otherwise, the Japanese word in
        saved_lexeme is stored in savedEnglishWord.
 */
void getEword()
{
  if (saved_lexeme == "watashi")
    savedEnglishWord = "I/me";
  else if (saved_lexeme == "anata")
    savedEnglishWord = "you";
  else if (saved_lexeme == "kare")
    savedEnglishWord = "he/him";
  else if (saved_lexeme == "kanojo")
    savedEnglishWord = "she/her";
  else if (saved_lexeme == "sore")
    savedEnglishWord = "it";
  else if (saved_lexeme == "mata")
    savedEnglishWord = "also";
  else if (saved_lexeme == "soshite")
    savedEnglishWord = "then";
  else if (saved_lexeme == "shikashi")
    savedEnglishWord = "however";
  else if (saved_lexeme == "dakara")
    savedEnglishWord = "therefore";
  else
    savedEnglishWord = saved_lexeme;
}//  end of getEword

//** Done by: Edgar Cruz
/*
PURPOSE: Translates the Japanese word to English
ALGORITHM: The token type in saved_token is put in a switch statement.
        saved_token is compared against the cases CONNECTOR, SUBJECT,
        IS, WAS, OBJECT, DESTINATION, WORD2, VERBPAST,
        VERBPASTNEG, VERB, and VERBNEG. If it finds a match with a
        case, it will display the token type and the English word
        in a text file called translated.txt. Then it will break. If the
        token type in the variable saved_token doesn't match any of
        the cases, then it will go to default and return.
 */
void gen()
{
  switch (saved_token)
  {

  case CONNECTOR:
    fileOutput << "CONNECTOR:   " << savedEnglishWord << endl;
    break;
  case SUBJECT:
    fileOutput << "ACTOR:       " << savedEnglishWord << endl;
    break;
  case IS:
  case WAS:
    fileOutput << "DESCRIPTION: " << savedEnglishWord << endl;
    fileOutput << "TENSE:       " << tokens[saved_token] << endl << endl;
```

```
    break;
  case OBJECT:
    fileOutput << "OBJECT:      " << savedEnglishWord << endl;
    break;
  case DESTINATION:
    fileOutput << "TO:          " << savedEnglishWord << endl;
    break;
  case WORD2:
    fileOutput << "ACTION:      " << savedEnglishWord << endl;
    break;
  case VERBPAST:
  case VERBPASTNEG:
  case VERB:
  case VERBNEG:
    fileOutput << "TENSE:       " << tokens[saved_token] << endl << endl;
    break;

  default:
    return;
  }

}//  end of gen

// The test driver to start the parser
//** Done by: Jesus Rivera

/*
PURPOSE: To translate a Japanese word to an English word.
ALGORITHM: The user inputs the name of the file that contains
          the Japanese words. Open the input file and open
          the output file that will contain the
          Japanese words translated to English.

 */
int main()
{

  string userInput;
  cout << "Please enter a file name: ";
  getline(cin, userInput);

  //- opens the input file
  fin.open(userInput.c_str());

  //- opens the output file translated.txt
  fileOutput.open("translated.txt");

  //- calls the <story> to start parsing
  story();

  //- closes the input file
  fin.close();

  //- closes traslated.txt
```

```
  fileOutput.close();
}// end
```

Section 8- Semantic test results

```
Script started on Thu 14 Dec 2017 10:47:17 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++ translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ ./a.out
Please enter a file name: partBtest1
Processing Story


========Processing <s>========
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: desu
desu is a word
Scanner called using word: desu
Processing <be>
Matched IS
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
```

========Processing <s>========
Processing <noun>
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: sensei
sensei is a word
Scanner called using word: sensei
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: desu
desu is a word
Scanner called using word: desu
Processing <be>
Matched IS
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika

========Processing <s>========
Processing <noun>
Matched WORD1
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: gohan
gohan is a word
Scanner called using word: gohan
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: o
o is a word

```
Scanner called using word: o
Matched OBJECT
Processing <afterObject>
.....Scanner was called.....
Word is: tabE
tabE is a word
Scanner called using word: tabE
Processing <verb>
Matched WORD2
Processing <tense>
.....Scanner was called.....
Word is: masu
masu is a word
Scanner called using word: masu
Matched VERB
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD

========Processing <s>========
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: tesuto
tesuto is a word
Scanner called using word: tesuto
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: o
o is a word
Scanner called using word: o
Matched OBJECT
Processing <afterObject>
.....Scanner was called.....
Word is: seito
seito is a word
Scanner called using word: seito
```

Processing <noun>
Matched WORD1
.....Scanner was called.....
Word is: ni
ni is a word
Scanner called using word: ni
Matched DESTINATION
Processing <verb>
.....Scanner was called.....
Word is: agE
agE is a word
Scanner called using word: agE
Matched WORD2
Processing <tense>
.....Scanner was called.....
Word is: mashita
mashita is a word
Scanner called using word: mashita
Matched VERBPAST
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: shikashi
shikashi is a word
Scanner called using word: shikashi

========Processing <s>========
Matched CONNECTOR
Processing <noun>
.....Scanner was called.....
Word is: seito
seito is a word
Scanner called using word: seito
Matched WORD1
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: yorokobI
yorokobI is a word
Scanner called using word: yorokobI
Processing <verb>
Matched WORD2
Processing <tense>
.....Scanner was called.....

Word is: masendeshita
masendeshita is a word
Scanner called using word: masendeshita
Matched VERBPASTNEG
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: dakara
dakara is a word
Scanner called using word: dakara

========Processing <s>========
Matched CONNECTOR
Processing <noun>
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: kanashii
kanashii is a word
Scanner called using word: kanashii
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: deshita
deshita is a word
Scanner called using word: deshita
Processing <be>
Matched WAS
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: soshite
soshite is a word
Scanner called using word: soshite

========Processing <s>========

Matched CONNECTOR
Processing <noun>
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika
Matched WORD1
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: toire
toire is a word
Scanner called using word: toire
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: ni
ni is a word
Scanner called using word: ni
Matched DESTINATION
Processing <verb>
.....Scanner was called.....
Word is: ikI
ikI is a word
Scanner called using word: ikI
Matched WORD2
Processing <tense>
.....Scanner was called.....
Word is: mashita
mashita is a word
Scanner called using word: mashita
Matched VERBPAST
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika

========Processing <s>========
Processing <noun>
Matched WORD1
.....Scanner was called.....

Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: nakI
nakI is a word
Scanner called using word: nakI
Processing <verb>
Matched WORD2
Processing <tense>
.....Scanner was called.....
Word is: mashita
mashita is a word
Scanner called using word: mashita
Matched VERBPAST
.....Scanner was called.....
Word is: .
Scanner called using word: .
Matched PERIOD

========Processing <s>========
.....Scanner was called.....
Word is: eofm
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:48:33 PM PST

ACTOR: I/me
DESCRIPTION: rika
TENSE: IS

ACTOR: I/me
DESCRIPTION: sensei
TENSE: IS

ACTOR: rika
OBJECT: gohan
ACTION: tabE
TENSE: VERB

ACTOR: I/me
OBJECT: tesuto
TO: seito
ACTION: agE
TENSE: VERBPAST

CONNECTOR: however
ACTOR: seito
ACTION: yorokobI
TENSE: VERBPASTNEG

CONNECTOR: therefore
ACTOR: I/me
DESCRIPTION: kanashii
TENSE: WAS

CONNECTOR: then
ACTOR: rika
TO: toire
ACTION: ikI
TENSE: VERBPAST

ACTOR: rika
ACTION: nakI
TENSE: VERBPAST

Test2

Script started on Thu 14 Dec 2017 10:54:21 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ ex./a.oug++ translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++ translator.cexit./a.out
Please enter a file name: partBtest2
Processing Story

========Processing <s>========
.....Scanner was called.....
Word is: soshite
soshite is a word
Scanner called using word: soshite
Matched CONNECTOR
Processing <noun>
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word

Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: desu
desu is a word
Scanner called using word: desu
Processing <be>
Matched IS
.....Scanner was called.....
Word is: ne
ne is a word
Scanner called using word: ne
SYNTAX ERROR: expected PERIOD but found ne
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:54:47 PM PST

CONNECTOR: then
ACTOR: I/me
DESCRIPTION: rika
TENSE: IS

Test 3

Script started on Thu 14 Dec 2017 10:57:09 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ ex./a.oug++
translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++
translator.cexit./a.out
Please enter a file name: partBtest3
Processing Story

========Processing <s>========
.....Scanner was called.....
Word is: dakara
dakara is a word
Scanner called using word: dakara
Matched CONNECTOR
Processing <noun>
.....Scanner was called.....
Word is: watashi

watashi is a word
Scanner called using word: watashi
Matched PRONOUN
.....Scanner was called.....
Word is: de
de is a word
Scanner called using word: de
SYNTAX ERROR: expected SUBJECT but found de
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 10:57:38 PM PST

CONNECTOR: therefore

Test 4

Script started on Thu 14 Dec 2017 11:03:46 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ ex./a.oug++ translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++ translator.cexit./a.out
Please enter a file name: partBtest4
Processing Story

========Processing <s>========
.....Scanner was called.....
Word is: watashi
watashi is a word
Scanner called using word: watashi
Processing <noun>
Matched PRONOUN
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Matched SUBJECT
Processing <X>
.....Scanner was called.....
Word is: rika
rika is a word
Scanner called using word: rika
Processing <noun>
Matched WORD1
Processing <Y> <afterNoun>
.....Scanner was called.....
Word is: mashita
mashita is a word
Scanner called using word: mashita

SYNTAX ERROR: unexpected mashita found in
afterNoun]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 11:04:18 PM PST

ACTOR: I/me

Test 5

Script started on Thu 14 Dec 2017 11:09:07 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++[[[[Kex./a.oug++
translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++
translator.cexit./a.out
Please enter a file name: parsetBtest5
Processing Story

========Processing <s>========
.....Scanner was called.....
Word is: wa
wa is a word
Scanner called using word: wa
Processing <noun>
SYNTAX ERROR: unexpected wa found in
noun]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 11:09:32 PM PST

Test 6

Script started on Thu 14 Dec 2017 11:11:22 PM PST
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ ex./a.oug++
translator.cpp
]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ g++
translator.cexit./a.out
Please enter a file name: partBtest6
Processing Story

========Processing <s>========
.....Scanner was called.....
Word is: apple
Scanner called using word: apple
Lexical Error
Processing <noun>

SYNTAX ERROR: unexpected apple found in
noun]0;cruz085@empress:~/cs421/CS421Progs/TranslatorFiles[cruz085@empress TranslatorFiles]$ exit
exit

Script done on Thu 14 Dec 2017 11:11:40 PM PST