0. Cover Sheet

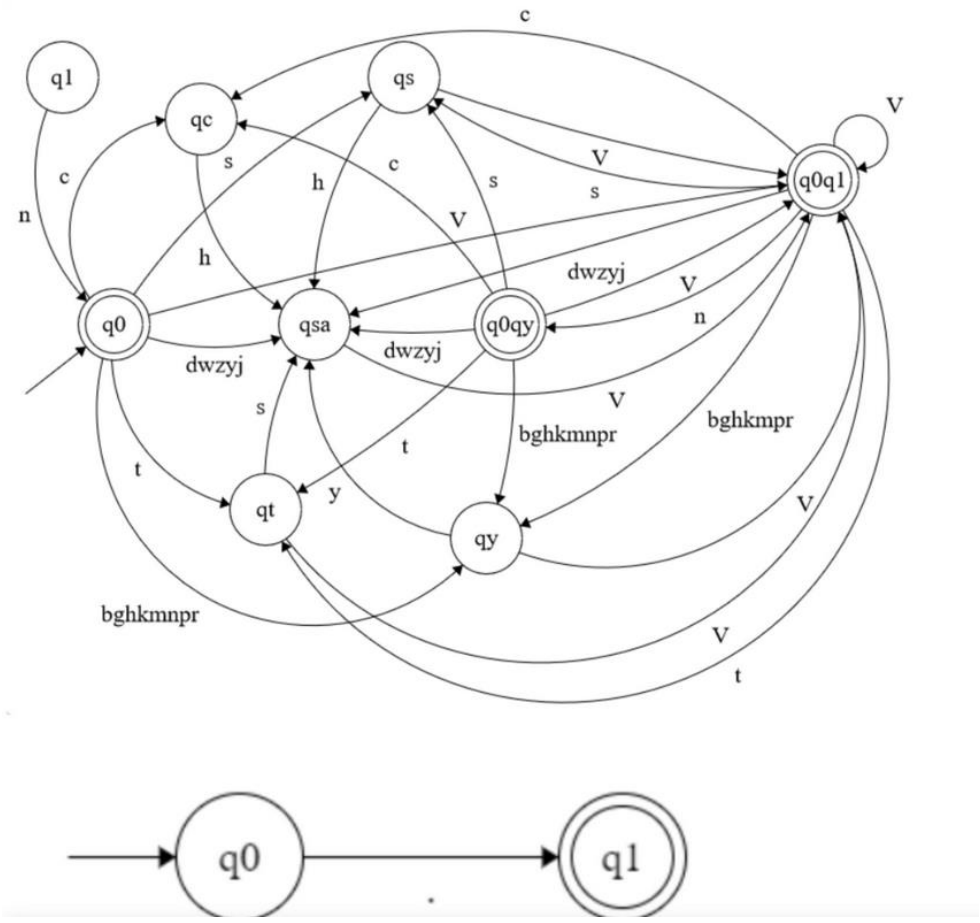
Group 7 Project Report

Names: Emaan Bashir, Morgan Buell, Darshan, Eric Thompson

State of the Program:

Translator *mostly* works, there is a bug sometimes where the description gives us “ERROR” instead for certain words, which we haven’t been able to figure out the cause of. Otherwise it seems to be printing out the correct translations.

1. DFA



2. Scanner.cpp

#include<iostream>

```
#include<fstream>

#include<string>


using namespace std;


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


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

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

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


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


// WORD DFA

// Done by: **Emaan Bashir

// RE: (vowel | vowel n | consonant vowel | consonant vowel n | consonant - pair vowel | consonant - pair vowel n) ^ +


bool word(string s)

{


   string state = "q0";

   int charpos = 0;


   /* replace the following todo the word dfa  ** */
```

```
  while (s[charpos] != '\0')
  {
    if (state == "q0" && s[charpos] == 'c')
      state = "qc";
    else
      if (state == "q0" && s[charpos] == 's')
        state = "qs";
      else
        if (state == "q0" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o'
|| s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
          state = "q0q1";
        else
          if (state == "q0" && (s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos]
== 'y' || s[charpos] == 'j'))
            state = "qsa";
          else
            if (state == "q0" && s[charpos] == 't')
              state = "qt";
            else
              if (state == "q0" && (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 = "qy";
              else
                if (state == "q1" && s[charpos] == 'n')
                  state = "q0";
```

```
                    else
                        if (state == "qc" && s[charpos] == 'h')
                            state = "qsa";
                        else
                            if (state == "qs" && s[charpos] == 'h')
                                state = "qsa";
                            else
                                if (state == "qs" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' ||
s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                    state = "q0q1";
                                else
                                    if (state == "qt" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i'
|| s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                        state = "q0q1";
                                    else
                                        if (state == "qt" && s[charpos] == 's')
                                            state = "qsa";
                                        else
                                            if (state == "qy" && s[charpos] == 'y')
                                                state = "qsa";
                                            else
                                                if (state == "qy" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos]
== 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                                    state = "q0q1";
                                                else
```

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

                                           state = "q0q1";

                                        else

                                           if (state == "q0q1" && s[charpos] == 'c')

                                              state = "qc";

                                           else

                                              if (state == "q0q1" && (s[charpos] == 'a' || s[charpos] == 'e' ||
s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))

                                                 state = "q0q1";

                                              else

                                                 if (state == "q0q1" && s[charpos] == 'n')

                                                    state = "q0qy";

                                                 else

                                                    if (state == "q0q1" && (s[charpos] == 'b' || s[charpos] == 'g'
|| s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' || s[charpos] == 'r'))

                                                       state = "qy";

                                                    else

                                                       if (state == "q0q1" && (s[charpos] == 'd' || s[charpos] ==
'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))

                                                          state = "qsa";

                                                       else

                                                          if (state == "q0q1" && s[charpos] == 's')

                                                             state = "qs";

                                                          else

                                                             if (state == "q0q1" && s[charpos] == 't')
```

```
                                                    state = "qt";

                                                else

                                                    if (state == "q0qy" && s[charpos] == 'c')

                                                        state = "qc";

                                                    else

                                                        if (state == "q0q1" && (s[charpos] == 'a' ||
s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' ||
s[charpos] == 'E'))

                                                            state = "q0q1";

                                                        else

                                                            if (state == "q0q1" && s[charpos] == 'n')

                                                                state = "q0qy";

                                                            else

                                                                if (state == "q0q1" && (s[charpos] == 'b' ||
s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' ||
s[charpos] == 'r'))

                                                                    state = "qy";

                                                                else

                                                                    if (state == "q0q1" && (s[charpos] == 'd'
|| s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))

                                                                        state = "qsa";

                                                                    else

                                                                        if (state == "q0q1" && s[charpos] == 's')

                                                                            state = "qs";

                                                                        else

                                                                            if (state == "q0q1" && s[charpos] ==
't')
```

```
                                                    state = "qt";

                                                else

                                                    if (state == "q0qy" && s[charpos]
== 'c')

                                                        state == "qc";

                                                    else

                                                        if (state == "q0q1" &&
(s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))

                                                            state = "q0q1";

                                                        else

                                                            if (state == "q0q1" &&
s[charpos] == 'n')

                                                                state = "q0qy";

                                                            else

                                                                if (state == "q0q1" &&
(s[charpos] == 'b' || s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' ||
s[charpos] == 'p' || s[charpos] == 'r'))

                                                                    state = "qy";

                                                                else

                                                                    if (state == "q0q1" &&
(s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))

                                                                        state = "qsa";

                                                                    else

                                                                        if (state == "q0q1" &&
s[charpos] == 's')

                                                                            state = "qs";

                                                                        else
```

```
                                                                    if (state == "q0q1" &&
s[charpos] == 't')
                                                                       state = "qt";
                                                                    else
                                                                       if (state == "q0qy"
&& s[charpos] == 'c')
                                                                          state = "qc";
                                                                       else
                                                                          if (state ==
"q0qy" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] ==
'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                                                             state = "q0q1";
                                                                          else
                                                                             if (state ==
"q0qy" && s[charpos] == 's')
                                                                                state = "qs";
                                                                             else
                                                                                if (state ==
"q0qy" && s[charpos] == 't')
                                                                                   state =
"qt";
                                                                                else
                                                                                   if (state ==
"q0qy" && (s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'j' || s[charpos] ==
'y'))
                                                                                      state =
"qsa";
                                                                                   else
```

```
return(false);
      charpos++;
   }//end of while
  // where did I end up????
   if (state == "q0" || state == "q0q1" || state == "q0qy") return(true);  // end in a final state
   else { return(false); }

}



// PERIOD DFA
// Done by: Morgan Buell and Darshan
// RE: .
bool period(string s)
{  // complete this
   string state = "q0"; // establishes the string state
   int charpos = 0;

   while (s[charpos != '\0']) { //while loop loops through the input file until it hits the end
      if (state == "q0" && s[charpos] == '.') { //if statement chekcs whether or not the scanner picks up a period
         state == "PERIOD"; //once scanner picks up period, it changes state to period
         return true; //returns "true" when the state is equal to period
      }
```

```
        else { //if scanner doesn't pick up a period return false

            return false;

        }


    }

}


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


// TABLES Done by: Eric Thompson


// ** Update the tokentype to be WORD1, WORD2, PERIOD, ERROR, EOFM, etc.

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


// ** For the display names of tokens - must be in the same order as the tokentype.

string tokenName[30] = { "Error", "Word1", "Word2", "Period", "Eofm", "Verb", "VerbNeg", "VerbPast",
"VerbPastNeg", "Is", "Was", "Object", "Subject", "Destination", "Pronoun", "Connector" };


// ** Need the reservedwords table to be set up here.

string reservedwords[30] = { "masu", "masen", "mashita", "masendeshita", "desu", "deshita", "o", "wa",
"ni", "watashi", "anata", "kare", "kanojo", "sore", "mata", "soshite", "shikashi", "dakara", "eofm" };

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

// ** Do not require any file input for this. Hard code the table.

// ** a.out should work without any additional files.
```

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

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

// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// ** Done by: Eric Thompson
int scanner(tokentype& tt, string& w)
{

        // ** Grab the next word from the file via fin
        // 1. If it is eofm, return right now.
        fin >> w;

        if (w == "eofm") //Return if we've hit the end of file
        {
                tt = 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.
  */
if (word(w))
{
        //Don't need to do anything yet
}
else if (period(w)) //No more processing needed, we can set the type and return
{
        tt = PERIOD;
        return -1;
}
else //Neither a word nor period, must be an error
{
        tt = ERROR;
        return -1;
}

/**
 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.
*/
```

```
//Checking if reserved word
int i = 0;//Keep track of the index so we can get the enum in the matching enums array
for (string s : reservedwords)
{
	if (s == w)		//the string in reservedwords matches our word
	{
		tt = reservedwordsenums[i];	//Set to the matching enum type
		return 0;
	}
	i++;	//Increment index
}

//Checking if WORD1 or WORD2
char lastLetter = w.back();		//Getting the last char of our word to compare

if ((lastLetter == 'I') | (lastLetter == 'E'))  //Word 1 if ending in capital I or E
{
	tt = WORD1;
}
else if (islower(lastLetter))		//Word 2 if a lowercase letter
{
	tt = WORD2;
}
else //Otherwise must be an error
```

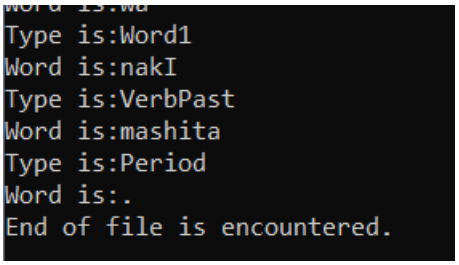
```
	{
		tt = ERROR;
		return -1;
	}

	//4. Return the token type & string  (pass by reference)
	return 0;
}//the end of scanner

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

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

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

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

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

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

}// end
```

3. Scanner Test Results

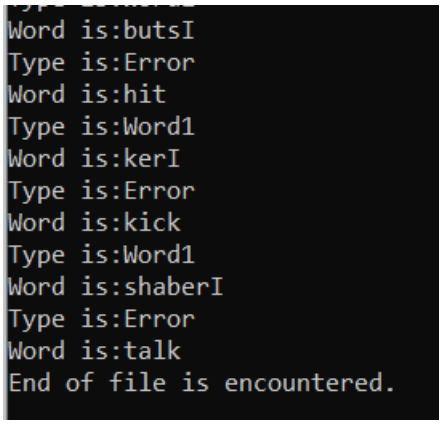
Test1

```
Enter the input file name: test1
Type is:Pronoun
Word is:watashi
Type is:Subject
Word is:wa
Type is:Word2
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:Word2
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:Word2
Word is:ryouri
Type is:Object
Word is:o
Type is:Word1
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:Word2
Word is:gohan
Type is:Object
Word is:o
Type is:Word2
Word is:seito
Type is:Destination
Word is:ni
Type is:Word1
Word is:agE
Type is:VerbPast
Word is:mashita
Type is:Period
Word is:.
Type is:Connector
Word is:shikashi
Type is:Word2
Word is:seito
Type is:Subject
Word is:wa
Type is:Word1
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:Word2
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:Word2
Word is:toire
Type is:Destination
Word is:ni
Type is:Word1
Word is:ikI
Type is:VerbPast
Word is:mashita
Type is:Period
Word is:.
Type is:Pronoun
Word is:watashi
Type is:Subject
```


Type is:Word1
Word is:nakI
Type is:VerbPast
Word is:mashita
Type is:Period
Word is:.
End of file is encountered.

Test2

```
Enter the input file name: test2
Type is:Word2
Word is:daigaku
Type is:Error
Word is:college
Type is:Word2
Word is:kurasu
Type is:Error
Word is:class
Type is:Word2
Word is:hon
Type is:Error
Word is:book
Type is:Word2
Word is:tesuto
Type is:Error
Word is:test
Type is:Word2
Word is:ie
Type is:Error
Word is:home*
Type is:Word2
Word is:isu
Type is:Error
Word is:chair
Type is:Word2
Word is:seito
Type is:Error
Word is:student
Type is:Word2
Word is:sensei
Type is:Error
Word is:teacher
Type is:Word2
Word is:tomodachi
Type is:Error
Word is:friend
Type is:Word2
Word is:jidoosha
Type is:Error
Word is:car
Type is:Word2
Word is:gyuunyuu
Type is:Error
Word is:milk
Type is:Word2
Word is:sukiyaki
Type is:Error
Word is:tenpura
Type is:Word2
Word is:sushi
Type is:Word2
Word is:biiru
Type is:Error
Word is:beer
Type is:Word2
Word is:sake
Type is:Word2
Word is:tokyo
Type is:Word2
Word is:kyuushuu
Type is:Error
Word is:Osaka
Type is:Word2
Word is:choucho
Type is:Error
Word is:butterfly
Type is:Word2
Word is:an
Type is:Word2
Word is:idea
Type is:Word2
Word is:yasashii
Type is:Error
Word is:easy
Type is:Word2
Word is:muzukashii
Type is:Error
Word is:difficult
Type is:Word2
Word is:ureshii
Type is:Error
Word is:pleased
Type is:Word2
Word is:shiawase
Type is:Error
Word is:happy
Type is:Word2
Word is:kanashii
Type is:Error
Word is:sad
Type is:Word2
Word is:omoi
Type is:Error
Word is:heavy
Type is:Word2
Word is:oishii
Type is:Error
Word is:delicious
Type is:Error
Word is:tennen
Type is:Error
Word is:natural
Type is:Word1
Word is:nakI
Type is:Error
Word is:cry
Type is:Word1
Word is:ikI
Type is:Error
Word is:go*
Type is:Word1
Word is:tabE
Type is:Error
Word is:eat
Type is:Word1
Word is:ukE
Type is:Error
Word is:take*
Type is:Word1
Word is:kakI
Type is:Error
Word is:write
Type is:Word1
Word is:yomI
Type is:Error
Word is:read
Type is:Word1
Word is:nomI
Type is:Error
Word is:drink
Type is:Word1
Word is:agE
Type is:Error
Word is:give
Type is:Word1
Word is:moraI
Type is:Error
Word is:receive
Type is:Word1
Word is:butsI
Type is:Error
```



4. Factored Rules w/ New Non-terminal Names and Semantic Routines

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

<afterSubject> ::= <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD | <noun> #getEword# <afterNoun>

<afterNoun> ::= <be> #gen(DESCRIPTION)# #gen(TENSE)# PERIOD | DESTINATION #gen(TO)# <verb> #getEword# #gen(ACTION)# <tense> #gen(TENSE)# PERIOD | OBJECT #gen(OBJECT)# <afterObject>

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

5. Parser Code w/ Translator.cpp

```
#include<iostream>

#include<fstream>

#include<string>

#include<iomanip>

#include <cstdlib>

#include <map>


using namespace std;
```

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


// TABLES Done by: Eric Thompson


// ** Update the tokentype to be WORD1, WORD2, PERIOD, ERROR, EOFM, etc.

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


// ** For the display names of tokens - must be in the same order as the tokentype.

string tokenName[30] = { "ERROR", "WORD1", "WORD2", "PERIOD", "EOFM", "VERB", "VERBNEG",
"VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT", "DESTINATION", "PRONOUN",
"CONNECTOR" };


// ** Need the reservedwords table to be set up here.

string reservedwords[30] = { "masu", "masen", "mashita", "masendeshita", "desu", "deshita", "o", "wa",
"ni", "watashi", "anata", "kare", "kanojo", "sore", "mata", "soshite", "shikashi", "dakara", "eofm" };

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

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

// ** Do not require any file input for this. Hard code the table.

// ** a.out should work without any additional files.

tokentype  saved_token;     // global buffer for the token the scanner returned.


bool   token_available = false;          // global flag indicating whether
```

```
                        // we have saved a token to eat up or not
string saved_lexeme;
string translated = "";

void AfterObject();
bool match(tokentype t);
tokentype next_token();
void syntaxerror2(tokentype expectedToken_Type, tokentype foundSaved_lexeme);

/* Look for all **'s and complete them */
//=====================================================
// Translator
//=====================================================
string saved_E_word;

//Store the dictionary key/value pairs for translation lookup
map<string, string> dict;

//ifstream for reading in files
ifstream fin;

//Initializes the map with whatever filename is passed in
//Written by: Eric Thompson
void initMap(string filename)
```

```
{
    //Stores the key/value pairs
    string key;
    string value;

    //Open the ifstream
    fin.open(filename);

    //Loop until fin hits the end of the file
    while (!fin.eof())
    {
        //Read in the key, then the value
        fin >> key;
        fin >> value;

        //cout << key << " | " << value << endl; //Temp cout for debugging, can be removed
        //Pass in the key/value pair
        dict[key] = value;
    }

    //Close the ifstream
    fin.close();
}
```

```
//Done By* Darshan and Morgan*

void gen(string line_type)

{

   string translated; //Used for verb tense



   int i = 0; //Counter while going through each reserved word

   for (string s : reservedwords)

   {

      if (s == saved_lexeme) //Check if we have a match with our reserved word

      {

         //cout << "-----Reserved word found " << saved_lexeme << " | " << s << " | " <<
reservedwordsnames[i] << endl;

         translated = reservedwordsnames[i]; //Get the associated name for that reserved word

      }

      i++; //Increment

   }

   //Check what the line type is, cout depending on it

   if (line_type == "CONNECTOR")

      cout << "CONNECTOR: " << saved_E_word << endl;

   else if (line_type == "OBJECT")

      cout << "OBJECT: " << saved_E_word << endl;

   else if (line_type == "ACTION")

      cout << "ACTION: " << saved_E_word << endl;

   else if (line_type == "ACTOR")

      cout << "ACTOR: " << saved_E_word << endl;
```

```
    else if (line_type == "TO")
        cout << "TO: " << saved_E_word << endl;
    else if (line_type == "DESCRIPTION")
        cout << "DESCRIPTION: " << saved_E_word << endl;
    else if (line_type == "TENSE")
        cout << "TENSE: " << translated << endl;

    else
        cout << "ERROR" << endl;
}


//  Done by: Emaan Bashir
// ** 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
void getEword()
{
    //For every element in the dictionary
    for (auto& word : dict)
    {
        if (word.first == saved_lexeme) //If the key matches our saved lexeme
        {
```

```
        //cout << "EQUAL: " << word.first << " AND " << saved_lexeme << endl;
        if (word.second != "") //Removes some weird false positives we were getting
        {
           saved_E_word = word.second; //Set saved E word to the value
           return;
        }

      }
   }
   //cout << "NOT EQUAL " << saved_lexeme << endl;
   saved_E_word = saved_lexeme; //If there was no match we set saved E word to the saved lexeme
}

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

// File scanner.cpp written by: Group Number: 7

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

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

// WORD DFA

// Done by: **Emaan Bashir

// RE: (vowel | vowel n | consonant vowel | consonant vowel n | consonant - pair vowel | consonant - pair vowel n) ^ +

bool word(string s)

{
```

```
   string state = "q0";
   int charpos = 0;

   /* replace the following todo the word dfa  ** */
   while (s[charpos] != '\0')
   {
      if (state == "q0" && s[charpos] == 'c')
         state = "qc";
      else
         if (state == "q0" && s[charpos] == 's')
            state = "qs";
         else
            if (state == "q0" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o'
|| s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
               state = "q0q1";
            else
               if (state == "q0" && (s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos]
== 'y' || s[charpos] == 'j'))
                  state = "qsa";
               else
                  if (state == "q0" && s[charpos] == 't')
                     state = "qt";
                  else
                     if (state == "q0" && (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 = "qy";
            else
                if (state == "q1" && s[charpos] == 'n')
                    state = "q0";
                else
                    if (state == "qc" && s[charpos] == 'h')
                        state = "qsa";
                    else
                        if (state == "qs" && s[charpos] == 'h')
                            state = "qsa";
                        else
                            if (state == "qs" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' ||
s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                state = "q0q1";
                            else
                                if (state == "qt" && (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i'
|| s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))
                                    state = "q0q1";
                                else
                                    if (state == "qt" && s[charpos] == 's')
                                        state = "qsa";
                                    else
                                        if (state == "qy" && s[charpos] == 'y')
                                            state = "qsa";
```

```
                                    else

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

                                            state = "q0q1";

                                        else

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

                                                state = "q0q1";

                                            else

                                                if (state == "q0q1" && s[charpos] == 'c')

                                                    state = "qc";

                                                else

                                                    if (state == "q0q1" && (s[charpos] == 'a' || s[charpos] == 'e' ||
s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' || s[charpos] == 'E'))

                                                        state = "q0q1";

                                                    else

                                                        if (state == "q0q1" && s[charpos] == 'n')

                                                            state = "q0qy";

                                                        else

                                                            if (state == "q0q1" && (s[charpos] == 'b' || s[charpos] == 'g'
|| s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' || s[charpos] == 'r'))

                                                                state = "qy";

                                                            else

                                                                if (state == "q0q1" && (s[charpos] == 'd' || s[charpos] ==
'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))

                                                                    state = "qsa";

                                                                else
```

```
                                                    if (state == "q0q1" && s[charpos] == 's')
                                                       state = "qs";
                                                    else
                                                       if (state == "q0q1" && s[charpos] == 't')
                                                          state = "qt";
                                                       else
                                                          if (state == "q0qy" && s[charpos] == 'c')
                                                          {
                                                          }
                                                          else
                                                             if (state == "q0q1" && (s[charpos] == 'a' ||
s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' || s[charpos] == 'I' ||
s[charpos] == 'E'))
                                                                state = "q0q1";
                                                             else
                                                                if (state == "q0q1" && s[charpos] == 'n')
                                                                   state = "q0qy";
                                                                else
                                                                   if (state == "q0q1" && (s[charpos] == 'b' ||
s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' || s[charpos] == 'p' ||
s[charpos] == 'r'))
                                                                      state = "qy";
                                                                   else
                                                                      if (state == "q0q1" && (s[charpos] == 'd'
|| s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))
                                                                         state = "qsa";
                                                                      else
```

```
                                                    if (state == "q0q1" && s[charpos] == 's')
                                                       state = "qs";
                                                    else
                                                       if (state == "q0q1" && s[charpos] ==
't')
                                                          state = "qt";
                                                       else
                                                          if (state == "q0qy" && s[charpos]
== 'c')
                                                          {
                                                          }
                                                          else if (state == "q0q1" &&
(s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
                                                             state = "q0q1";

                                                          else
                                                             if (state == "q0q1" && s[charpos]
== 'n')
                                                                state = "q0qy";
                                                             else
                                                                if (state == "q0q1" &&
(s[charpos] == 'b' || s[charpos] == 'g' || s[charpos] == 'h' || s[charpos] == 'k' || s[charpos] == 'm' ||
s[charpos] == 'p' || s[charpos] == 'r'))
                                                                   state = "qy";
                                                                else
                                                                   if (state == "q0q1" &&
(s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'y' || s[charpos] == 'j'))
```

```
                                                    state = "qsa";
                                                else
                                                    if (state == "q0q1" &&
s[charpos] == 's')
                                                        state = "qs";
                                                    else
                                                        if (state == "q0q1" &&
s[charpos] == 't')
                                                            state = "qt";
                                                        else
                                                            if (state == "q0qy" &&
s[charpos] == 'c')
                                                                state = "qc";
                                                            else
                                                                if (state == "q0qy"
&& (s[charpos] == 'a' || s[charpos] == 'e' || s[charpos] == 'i' || s[charpos] == 'o' || s[charpos] == 'u' ||
s[charpos] == 'I' || s[charpos] == 'E'))
                                                                    state = "q0q1";
                                                                else
                                                                    if (state ==
"q0qy" && s[charpos] == 's')
                                                                        state = "qs";
                                                                    else
                                                                        if (state ==
"q0qy" && s[charpos] == 't')
                                                                            state = "qt";
                                                                        else
```

```
                                                                                                    if (state ==
"q0qy" && (s[charpos] == 'd' || s[charpos] == 'w' || s[charpos] == 'z' || s[charpos] == 'j' || s[charpos] ==
'y'))

                                                                                                       state =
"qsa";

                                                                                                   else


return(false);

       charpos++;

   }//end of while

  // where did I end up????

   if (state == "q0" || state == "q0q1" || state == "q0qy") return(true);  // end in a final state

   else { return(false); }


}




// PERIOD DFA

// Done by: Morgan Buell and Darshan

bool period(string s)

{  // complete this



   string state = "q0"; // establishes the string state

   int charpos = 0;
```

```
   while (s[charpos != '\0']) { //while loop loops through the input file until it hits the end
      if (state == "q0" && s[charpos] == '.') { //if statement chekcs whether or not the scanner picks up a period
         state == "PERIOD"; //once scanner picks up period, it changes state to period
         return true; //returns "true" when the state is equal to period
      }
      else { //if scanner doesn't pick up a period return false
         return false;
      }

   }
}

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

// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// ** Done by: Eric Thompson
int scanner(tokentype& tt, string& w)
{

   // ** Grab the next word from the file via fin
   // 1. If it is eofm, return right now.
```

```
fin >> w;


if (w == "eofm") //Return if we've hit the end of file
{
   tt = 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.
*/
if (word(w))
{
   //Don't need to do anything yet
}
else if (period(w)) //No more processing needed, we can set the type and return
{
   tt = PERIOD;
   return -1;
}
else //Neither a word nor period, must be an error
{
```

```
    tt = ERROR;
    cout << "Lexical error: " << w << " is not a valid token." << endl;
    return -1;
}

/**
        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.
*/
//Checking if reserved word
int i = 0;    //Keep track of the index so we can get the enum in the matching enums array
for (string s : reservedwords)
{
    if (s == w)    //the string in reservedwords matches our word
    {
        //cout << "Set " << s << " to " << w << endl;
        tt = reservedwordsenums[i];    //Set to the matching enum type
        return 0;
    }
    i++;    //Increment index
}
```

```
   //Checking if WORD1 or WORD2
   char lastLetter = w.back();    //Getting the last char of our word to compare

   if ((lastLetter == 'I') | (lastLetter == 'E'))    //Word 2 if ending in capital I or E
   {
      tt = WORD2;
   }
   else if (islower(lastLetter))    //Word 1 if a lowercase letter
   {
      tt = WORD1;
   }
   else //Otherwise must be an error
   {
      tt = ERROR;
      cout << "Lexical error: " << w << " is not a valid token." << endl;
      return -1;
   }

   //4. Return the token type & string  (pass by reference)
   return 0;
}//the end of scanner

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

```
//================================================


// ----- 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: Error 1: Match fail, unexpected token type found

// Done by: Eric Thompson

void syntaxerror1(tokentype expectedToken_Type, tokentype foundSaved_lexeme)

{

   if (foundSaved_lexeme == EOFM)

   {

      cout << "End of file encountered" << endl;

      exit(1);

   }

   cout << "SYNTAX ERROR: expected " << tokenName[expectedToken_Type] << " but found " <<
tokenName[foundSaved_lexeme] << endl;

   exit(1);

}

// Type of error: Error 2, Switch default, unexpected token found in parser function

// Done by: Eric Thompson

void syntaxerror2(tokentype unexpectedSaved_lexeme, string parserFunction)

{

   if (unexpectedSaved_lexeme == EOFM)
```

```
   {
      cout << "End of file encountered" << endl;
      exit(1);
   }
   cout << "SYNTAX ERROR: unexpected " << tokenName[unexpectedSaved_lexeme] << " found in " << parserFunction << endl;
   exit(1);
}
// ** Need the updated match and next_token with 2 global vars
// saved_token and saved_lexeme

// Purpose: **Save the string returned from the scanner in a globally accessible saved_lexeme
// Done by: Emaan Bashir
tokentype next_token() {
   if (!token_available)   // if there is no saved token yet
   {

      scanner(saved_token, saved_lexeme);    // call scanner to grab a new token
      //cout << "Scanner called using word: " << saved_lexeme << endl;
      // saved_token is the token type and
// saved_lexeme is the word that is read in
      token_available = true;                        // mark that fact that you have saved it

      if (saved_token == ERROR)
      {
```

```
            //syntaxerror1(saved_lexeme, saved_token);
        }
    }
    translated += dict[saved_lexeme] + " ";
    return saved_token;    // return the saved token
}

// Purpose: Checks and consumes expected token
// Compares next_token() and the expected token if they're different
// it will generate a syntax
// Done by: Morgan Buell
bool match(tokentype expected)
{
    if (next_token() != expected)
    {
        syntaxerror1(next_token(), saved_token);
    }
    else
    {
        token_available = false;
        //cout << "Matched " << tokenName[expected] << endl;
        return true;
    }
```

```
}

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

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

// Grammar: <be> ::= IS | WAS
// Done by: Emaan Bashir
void be()
{
   //cout << "Processing <" << "be>" << endl;
   //cout << tokenName[next_token()] << endl;
   switch (next_token()) {
   case IS:
      match(IS);
      break;
   case WAS:
      match(WAS);
      break;
   default:
      syntaxerror2(next_token(), "be");
   }
```

```
}

// Grammar: <noun> ::= WORD1 | PRONOUN
// Done by: Eric Thompson
void Noun()
{
   //cout << "Processing <" << "Noun>" << endl;
   //cout << tokenName[next_token()] << endl;
   switch (next_token())
   {
   case WORD1:
      match(WORD1);
      break;
   case PRONOUN:
      match(PRONOUN);
      break;
   default:    //Invalid
      syntaxerror2(next_token(), "Noun");
   }
}

// Grammar: <noun> ::= WORD2
// Done by: Eric Thompson
void Verb()
```

```
{
   //cout << "Processing <" << "Verb>" << endl;
   //cout << tokenName[next_token()] << endl;
   match(WORD2);
}

//Grammar: <Tense> ::= VERBPAST | VERBPASTNEG | VERB | VERBNEG
//Done by: Morgan Buell
void Tense()
{
   //cout << "Processing <" << "Tense>" << endl;
   //cout << tokenName[next_token()] << 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(next_token(), "Tense");
   }
}

// Grammar: <afterNoun> ::= <be> gen(DESCRIPTION) gen(TENSE) PERIOD | DESTINATION gen(TO)
<verb> getEword gen(ACTION) <tense> gen(TENSE) PERIOD | OBJECT gen(OBJECT) <afterObject>
// Done by: Emaan Bashir
void AfterNoun()
{
   //cout << "Processing <" << "AfterNoun>" << endl;
   //cout << tokenName[next_token()] << endl;
   switch (next_token()) {
   case IS:
   case WAS:
      be();
      gen("DESCRIPTION");
      gen("TENSE");
      match(PERIOD);
      break;
```

```
    case DESTINATION:
        match(DESTINATION);
        gen("TO");
        Verb();
        getEword();
        gen("ACTION");
        Tense();
        gen("TENSE");
        match(PERIOD);
        break;

    case OBJECT:
        match(OBJECT);
        gen("OBJECT");
        AfterObject();
        break;

    default:
        syntaxerror2(next_token(), "AfterNoun");
    }
}

// Grammar: <afterSubject> ::= <verb> getEword gen(VERB) <tense> gen(TENSE) PERIOD | <noun>
getEword <afterNoun>

// Done by: Eric Thompson
```

```
void AfterSubject()
{
   //cout << "Processing <" << "AfterSubject>" << endl;
   //cout << tokenName[next_token()] << endl;
   switch (next_token())
   {
   case WORD2:
      //Call Verb then Tense RDP
      Verb();
      getEword();
      gen("VERB");
      Tense();
      gen("TENSE");
      //Match PERIOD
      match(PERIOD);
      break;
   case WORD1: //Fall through
   case PRONOUN:
      //Call Noun and AfterNoun RDP
      Noun();
      getEword();
      AfterNoun();
      break;
   default:    //Invalid
```

```
        syntaxerror2(next_token(), "AfterSubject");
    }
}


//Grammar <AfterObject> ::= <verb> getEword gen(ACTION) <tense> gen(TENSE) PERIOD | <noun>
getEword gen(ACTION) DESTINATION gen(TO) <verb> getEword <tense> gen(TENSE) PERIOD

//Done by: Morgan Buell

void AfterObject()

{
    //cout << "Processing <" << "AfterObject>" << endl;
    //cout << tokenName[next_token()] << endl;
    switch (next_token())
    {
    case WORD2:
        Verb();
        getEword();
        gen("ACTION");
        Tense();
        gen("TENSE");
        match(PERIOD);
        break;
    case WORD1:
    case PRONOUN:
        Noun();
        getEword();
```

```
        gen("ACTION");
        match(DESTINATION);
        gen("TO");
        Verb();
        getEword();
        gen("ACTION");
        Tense();
        gen("TENSE");
        match(PERIOD);
        break;
    default:
        syntaxerror2(next_token(), "AfterObject");
    }
}



// Grammar: <s> ::= [CONNECTOR getEword gen(CONNECTOR)] <noun> getEword [SUBJECT]
gen(ACTOR) <afterSubject>
// Done by: Eric Thompson
void s()
{
    //cout << "Processing <" << "s>" << endl;
    //cout << tokenName[next_token()] << endl;
    //Optionally match the CONNECTOR
    if (next_token() == CONNECTOR)
```

```
    {
        match(CONNECTOR);
        getEword();
        gen("CONNECTOR");
    }
    //Call Noun RDP
    Noun();
    getEword();
    //Match SUBJECT
    match(SUBJECT);
    gen("ACTOR");
    //Call after subject RDP
    AfterSubject();
}
// Grammar: <story> ::= <s> {<s>}
// Done by: Darshan and Morgan
void story()
{



    s();
    cout << endl;

    while (true)//Repeatable part
```

```
    {


        switch (next_token())//check the next_token
        {


        case CONNECTOR:
        case WORD1:
        case PRONOUN:
            s();//recursive call
            cout << endl;
            break;


        default:
            if (saved_lexeme != "eofm") //if statement to stop at eofm
                syntaxerror2(next_token(), "story");
            return;




        }
    }
}


string filename;
```

```
//---------- Driver ---------------------------


// The new test driver to start the parser
// Done by:  **
int main()
{
   cout << "Group 7 Translator" << endl;
   cout << "Enter the name of the lexicon to be read in: ";
   string lexicon;
   cin >> lexicon;
   initMap(lexicon);


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


   //** calls the <story> to start parsing
   while (true)
   {
      story();
      // the arguments
      if (saved_lexeme == "eofm") break;  // stop now
      cout << endl;
   }
```

```
   //** closes the input file

   fin.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

Test1

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test1.txt
ACTOR: I/me
DESCRIPTION: rika
TENSE: IS

ACTOR: I/me
DESCRIPTION: teacher
TENSE: IS

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

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

CONNECTOR: However
ACTOR: student
ERROR
TENSE: VERBPASTNEG

CONNECTOR: Therefore
ACTOR: I/me
DESCRIPTION: sad
TENSE: WAS

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

ACTOR: rika
ERROR
TENSE: VERBPAST

End of file encountered
```

Test2

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test2.txt
CONNECTOR: Then
ACTOR: I/me
DESCRIPTION: rika
TENSE: IS
SYNTAX ERROR: expected WORD1 but found WORD1
```

Test3

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test3.txt
CONNECTOR: Therefore
SYNTAX ERROR: expected WORD1 but found WORD1
```

Test4

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test4.txt
ACTOR: I/me
SYNTAX ERROR: unexpected VERBPAST found in AfterNoun
```

Test5

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test5.txt
SYNTAX ERROR: unexpected SUBJECT found in Noun
```

Test6

```
Group 7 Translator
Enter the name of the lexicon to be read in: lexicon.txt
Enter the input file name: test6.txt
Lexical error: apple is not a valid token.
SYNTAX ERROR: unexpected ERROR found in Noun
```