0 – Cover Sheet

CS 421 Translator Project

Group 13

Corbin Thaete, Jorge Diaz, Nam Cuong Tran

State of program statement: Until this point, our group has succeeded in completing all aspects of the code and getting it to run without difficulty. Using the g++ compiler on putty, we have been able to compile and execute our code perfectly and preform the required tests. We have implemented extra credit on part B of the project where we have syntax error correction and the ability to turn off tracing messages.

Table of Contents:

Section 0 – page 1

Section 1 – page 2

Section 2 – page 3

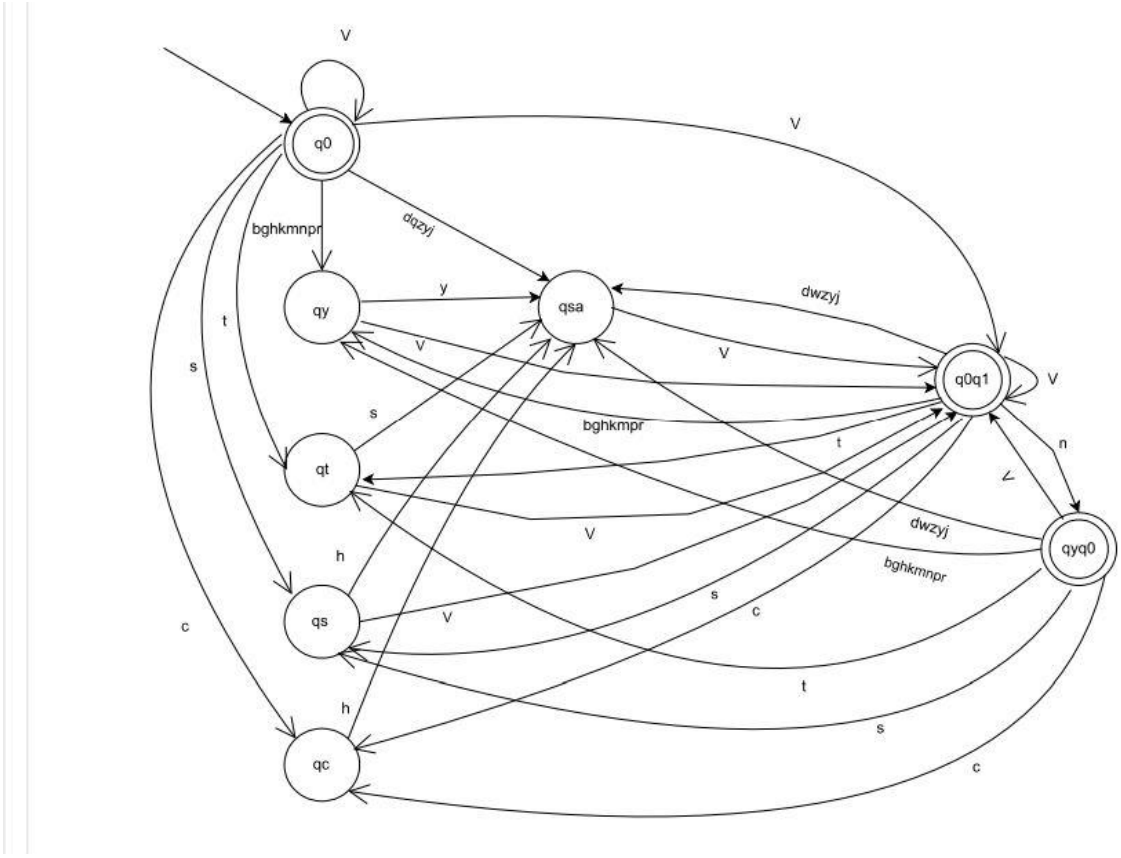
Section 3 – page 15

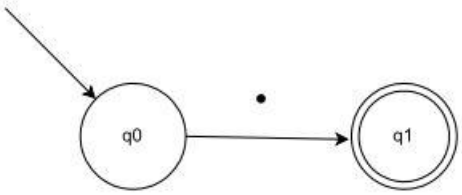
Section 4 – page 36

Section 5 – page 37

Section 6 – page 52

1 – DFA (Final Version and Period DFA)





2 – Scanner Code (Ends on page 14)

```
#include<iostream>
#include<fstream>
#include<string>
#include <algorithm>
#include <map>
#include <vector>
using namespace std;
/* Look for all **'s and complete them */
//=====================================================
// File scanner.cpp written by: Group Number: Nam Cuong Tran, Jorge Diaz, Corbin Thaete
//=====================================================
// --------- Two DFAs ---------------------------------
// WORD DFA
// Done by: Nam Cuong Tran
// RE:  (vowel | vowel n | consonant vowel | consonant vowel n |consonant-pair vowel | conso\
nant-pair vowel n)^+
bool word (string s)
{
  vector<char> vowel;
  vowel.push_back('a');
  vowel.push_back('i');
  vowel.push_back('u');
  vowel.push_back('e');
  vowel.push_back('o');
  vowel.push_back('I');
```

```
vowel.push_back('E');
vector <char> consonant_only;
consonant_only.push_back('d');
consonant_only.push_back('j');
consonant_only.push_back('w');
consonant_only.push_back('y');
consonant_only.push_back('z');
vector <char> consonant_pair_y;
consonant_pair_y.push_back('b');
consonant_pair_y.push_back('g');
consonant_pair_y.push_back('h');
consonant_pair_y.push_back('k');
consonant_pair_y.push_back('m');
consonant_pair_y.push_back('n');
consonant_pair_y.push_back('p');
consonant_pair_y.push_back('r');
int state = 0;
int charpos = 0;
while (s[charpos] != '\0')
 {
   if(state == 0 && count(vowel.begin(), vowel.end(), s[charpos]))
    {
      state = 6;
    }
   else if(state == 6 && s[charpos] == 'n')
    {
      state = 7;
    }
```

```
    else if(state == 0 && count(consonant_only.begin(), consonant_only.end(), s[charpos]))
     {
       state = 5;
     }
    else if(state == 5 && count(vowel.begin(), vowel.end(), s[charpos]))
     {
       state = 6;
     }
    else if(state == 0 && count(consonant_pair_y.begin(), consonant_pair_y.end(), s[charpo\
s]))
     {
       state = 1;
     }
    else if (state == 1 && s[charpos] == 'y')
     {
       state = 5;
     }
    else if (state == 1 && count(vowel.begin(), vowel.end(), s[charpos]))
     {
       state = 6;
     }
    else if( state == 0 && s[charpos] == 't')
     {
       state = 2;
     }
else if(state == 2 && s[charpos] == 's')
     {
       state = 5;
```

```
    }
    else if(state == 2 && count(vowel.begin(), vowel.end(), s[charpos]))
    {
      state = 6;
    }
    else if( state == 0 && s[charpos] == 's')
    {
      state = 2;
    }
    else if(state == 2 && s[charpos] == 'h')
    {
      state = 5;
    }
    else if(state == 2 && count(vowel.begin(), vowel.end(), s[charpos]))
    {
      state = 6;
    }
    else if(state == 0 && s[charpos] == 's')
    {
      state = 3;
    }
else if(state == 3 && s[charpos] == 'h')
    {
      state = 5;
    }
    else if(state == 3 && count(vowel.begin(), vowel.end(), s[charpos]))
    {
      state = 6;
```

```
      }
    else if(state == 0 && s[charpos] == 'c')
      {
        state = 4;
      }
    else if(state == 4 && s[charpos] == 'h')
      {
        state = 5;
      }
    else if(state == 6 && count(vowel.begin(), vowel.end(), s[charpos]))
      {
        state = 6;
      }
    else if(state == 6 && count(consonant_only.begin(), consonant_only.end(), s[charpos]))
      {
        state = 5;
      }
else if(state == 6 && count(consonant_pair_y.begin(), consonant_pair_y.end(), s[charpo\
s]))
      {
        state = 1;
      }
    else if(state == 6 && s[charpos] == 't')
      {
        state = 2;
      }
    else if(state == 6 && s[charpos] == 's')
      {
```

```
        state = 3;
       }
     else if(state == 6 && s[charpos] == 'c')
       {
        state = 4;
       }

     else if(state == 7 && count(vowel.begin(), vowel.end(), s[charpos]))
       {
        state = 6;
       }
else if(state == 7 && count(consonant_only.begin(), consonant_only.end(), s[charpos]))
       {
        state = 5;
       }
     else if(state == 7 && count(consonant_pair_y.begin(), consonant_pair_y.end(), s[charpo\
s]))
       {
        state = 1;
       }
     else if(state == 7 && s[charpos] == 't')
       {
        state = 2;
       }
     else if(state == 7 && s[charpos] == 's')
       {
        state = 3;
       }
```

```
    else if(state == 7 && s[charpos] == 'c')
     {
       state = 4;
     }
    else
      return(false);
    charpos++;
  }//end of while

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

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

// ------ Three  Tables ----------------------------------
// TABLES Done by: Nam Cuong Tran

// ** 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};


// ** Need the reservedwords table to be set up here.
// ** Do not require any file input for this. Hard code the table.
// ** a.out should work without any additional files.
string reverved_words_map(string word)
{
  string type;
  map<string, string> reservedwords_table;
  reservedwords_table.insert(pair<string, string>("masu", "VERB"));
  reservedwords_table.insert(pair<string, string>("masen", "VERBNEG"));
  reservedwords_table.insert(pair<string, string>("mashita", "VERBPAST"));
  reservedwords_table.insert(pair<string, string>("masendeshita", "VERBPASTNEG"));
  reservedwords_table.insert(pair<string, string>("desu", "IS"));
  reservedwords_table.insert(pair<string, string>("deshita", "WAS"));
  reservedwords_table.insert(pair<string, string>("o", "OBJECT"));
  reservedwords_table.insert(pair<string, string>("wa", "SUBJECT"));
  reservedwords_table.insert(pair<string, string>("ni", "DESTINATION"));
  reservedwords_table.insert(pair<string, string>("watashi", "PRONOUN"));
  reservedwords_table.insert(pair<string, string>("anata", "PRONOUN"));
  reservedwords_table.insert(pair<string, string>("kare", "PRONOUN"));
  reservedwords_table.insert(pair<string, string>("kanojo", "PRONOUN"));
  reservedwords_table.insert(pair<string, string>("sore", "PRONOUN"));
  reservedwords_table.insert(pair<string, string>("mata", "CONNECTOR"));
  reservedwords_table.insert(pair<string, string>("soshite", "CONNECTOR"));
  reservedwords_table.insert(pair<string, string>("shikashi", "CONNECTOR"));
  reservedwords_table.insert(pair<string, string>("dakara", "CONNECTOR"));
```

```
  for(int i = 0; i < reservedwords_table.size(); i++)
   {
     if(reservedwords_table[word] != "")
       {
         type = reservedwords_table[word];
         return type;
       }
   }
  return "none";
}
// ------------ Scanner and Driver -----------------------

ifstream fin;  // global stream for reading from the input file
// For the display names of tokens - must be in the same order as the tokentype.
string tokenName[16] = {"ERROR", "WORD1", "WORD2", "PERIOD", "EOFM", "VERB",
"VERBNEG", "VER\
BPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT", "DESTINATION", "PRONOUN",
"CONNECTO\
R"};
// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// Done by: Jorge Diaz
int scanner(tokentype& tt, string& w)
{
  // Grab the next word from the file via fin
  cout << ".....Scanner was called..." << endl;
  fin >> w;  // grab next word from the file via fin
  cout << ">>>>>Word is:" << w << endl;
```

```
if( w == "eofm")
  return EOFM;

// passing token w into word DFA
if(word(w)){
  // checks if token w is a reserved word
  if(reverved_words_map(w) == "none")
    {
      // checks if last character is uppercase
      if(isupper(w.at(w.size()-1)))
        tt = WORD2; // ends in E or I
      else
        tt = WORD1;
    }
    }
  else{
    string type = reverved_words_map(w);
    if (type == "VERB")
      tt = VERB;
    else if(type == "VERBNEG")
      tt = VERBNEG;
    else if(type == "VERBPAST")
      tt = VERBPAST;
    else if(type == "VERBPASTNEG")
      tt = VERBPASTNEG;
    else if(type == "IS")
      tt = IS;
    else if(type == "WAS")
```

```
      tt = WAS;
    else if(type == "OBJECT")
      tt = OBJECT;
    else if(type == "SUBJECT")
      tt = SUBJECT;
    else if(type == "DESTINATION")
      tt = DESTINATION;
    else if(type == "PRONOUN")
      tt = PRONOUN;
    else if(type == "CONNECTOR")
      tt = CONNECTOR;
   }
  }
  // passes token w into PERIOD DFA
  else if(period(w))
   {
    tt = PERIOD;
   }
  // if token doesnt finish in a final state then lexical error
  else{
   cout << ">>>>>Lexical Error: The string is not in my language" << endl;
   tt = ERROR;
  }
  // returns the type
  return tt;
}//the end of scanner
```

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

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

  fin.open(filename.c_str(), fstream::in);

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

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

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

}// end

3 – Original Scanner Test Results

```
[thaet001@empress cs421]$ /cs/recordhw_LK scanner1output.txt
Recording of your homework run will start with the next command prompt.
  WHEN YOU ARE FINISHED: say 'exit' at the command prompt.
  It will end the recording.
Script started, file is ,scanner1output.txt
[thaet001@empress cs421]$ g++ scanner.cpp
[thaet001@empress cs421]$ ./a.out
Enter the input file name: scannertest1.txt
.....Scanner was called...
>>>>>Word is:watashi
Type is:PRONOUN
Word is:watashi
.....Scanner was called...
>>>>>Word is:wa
Type is:SUBJECT
Word is:wa
.....Scanner was called...
>>>>>Word is:rika
Type is:WORD1
Word is:rika
.....Scanner was called...
>>>>>Word is:desu
Type is:IS
Word is:desu
.....Scanner was called...
```

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:sensei

Type is:WORD1

Word is:sensei

.....Scanner was called...

>>>>>Word is:desu

Type is:IS

Word is:desu

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:ryouri

Type is:WORD1

Word is:ryouri

.....Scanner was called...

>>>>>Word is:o

Type is:OBJECT

Word is:o

.....Scanner was called...

>>>>>Word is:yarI

Type is:WORD2

Word is:yarI

.....Scanner was called...

>>>>>Word is:masu

Type is:VERB

Word is:masu

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:gohan

Type is:WORD1

Word is:gohan

.....Scanner was called...

>>>>>Word is:o

Type is:OBJECT

Word is:o

.....Scanner was called...

>>>>>Word is:seito

Type is:WORD1

Word is:seito

.....Scanner was called...

>>>>>Word is:ni

Type is:DESTINATION

Word is:ni

.....Scanner was called...

>>>>>Word is:agE

Type is:WORD2

Word is:agE

.....Scanner was called...

>>>>>Word is:mashita

Type is:VERBPAST

Word is:mashita

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:shikashi

Type is:CONNECTOR

Word is:shikashi

.....Scanner was called...

>>>>>Word is:seito

Type is:WORD1

Word is:seito

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:yorokobI

Type is:WORD2

Word is:yorokobI

.....Scanner was called...

>>>>>Word is:masendeshita

Type is:VERBPASTNEG

Word is:masendeshita

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:dakara

Type is:CONNECTOR

Word is:dakara

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:kanashii

Type is:WORD1

Word is:kanashii

.....Scanner was called...

>>>>>Word is:deshita

Type is:WAS

Word is:deshita

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:soshite

Type is:CONNECTOR

Word is:soshite

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:toire

Type is:WORD1

Word is:toire

.....Scanner was called...

>>>>>Word is:ni

Type is:DESTINATION

Word is:ni

.....Scanner was called...

>>>>>Word is:ikI

Type is:WORD2

Word is:ikI

.....Scanner was called...

>>>>>Word is:mashita

Type is:VERBPAST

Word is:mashita

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:watashi

Type is:PRONOUN

Word is:watashi

.....Scanner was called...

>>>>>Word is:wa

Type is:SUBJECT

Word is:wa

.....Scanner was called...

>>>>>Word is:nakI

Type is:WORD2

Word is:nakI

.....Scanner was called...

>>>>>Word is:mashita

Type is:VERBPAST

Word is:mashita

.....Scanner was called...

>>>>>Word is:.

Type is:PERIOD

Word is:.

.....Scanner was called...

>>>>>Word is:eofm

End of file is encountered.

[thaet001@empress cs421]$ quit

----------Test 2 Results ----------

[thaet001@empress cs421]$ g++ scanner.cpp

[thaet001@empress cs421]$ ./a.out

Enter the input file name: scannertest2.txt

.....Scanner was called...

>>>>>Word is:daigaku

Type is:WORD1

Word is:daigaku

.....Scanner was called...

>>>>>Word is:college

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:college

.....Scanner was called...

>>>>>Word is:kurasu

Type is:WORD1

Word is:kurasu

.....Scanner was called...

>>>>>Word is:class

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:class

.....Scanner was called...

>>>>>Word is:hon

Type is:WORD1

Word is:hon

.....Scanner was called...

>>>>>Word is:book

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:book

.....Scanner was called...

>>>>>Word is:tesuto

Type is:WORD1

Word is:tesuto

.....Scanner was called...

>>>>>Word is:test

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:test

.....Scanner was called...

>>>>>Word is:ie

Type is:WORD1

Word is:ie

.....Scanner was called...

>>>>>Word is:home*

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:home*

.....Scanner was called...

>>>>>Word is:isu

Type is:WORD1

Word is:isu

.....Scanner was called...

>>>>>Word is:chair

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:chair

.....Scanner was called...

>>>>>Word is:seito

Type is:WORD1

Word is:seito

.....Scanner was called...

>>>>>Word is:student

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:student

.....Scanner was called...

>>>>>Word is:sensei

Type is:WORD1

Word is:sensei

.....Scanner was called...

>>>>>Word is:teacher

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:teacher

.....Scanner was called...

>>>>>Word is:tomodachi

Type is:WORD1

Word is:tomodachi

.....Scanner was called...

>>>>>Word is:friend

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:friend

.....Scanner was called...

>>>>>Word is:jidoosha

Type is:WORD1

Word is:jidoosha

.....Scanner was called...

>>>>>Word is:car

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:car

.....Scanner was called...

>>>>>Word is:gyuunyuu

Type is:WORD1

Word is:gyuunyuu

.....Scanner was called...

>>>>>Word is:milk

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:milk

.....Scanner was called...

>>>>>Word is:sukiyaki

Type is:WORD1

Word is:sukiyaki

.....Scanner was called...

>>>>>Word is:tenpura

Type is:WORD1

Word is:tenpura

.....Scanner was called...

>>>>>Word is:sushi

Type is:WORD1

Word is:sushi

.....Scanner was called...

>>>>>Word is:biiru

Type is:WORD1

Word is:biiru

.....Scanner was called...

>>>>>Word is:beer

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:beer

.....Scanner was called...

>>>>>Word is:sake

Type is:WORD1

Word is:sake

.....Scanner was called...

>>>>>Word is:tokyo

Type is:WORD1

Word is:tokyo

.....Scanner was called...

>>>>>Word is:kyuushuu

Type is:WORD1

Word is:kyuushuu

.....Scanner was called...

>>>>>Word is:Osaka

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:Osaka

.....Scanner was called...

>>>>>Word is:choucho

Type is:WORD1

Word is:choucho

.....Scanner was called...

>>>>>Word is:butterfly

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:butterfly

.....Scanner was called...

>>>>>Word is:an

Type is:WORD1

Word is:an

.....Scanner was called...

>>>>>Word is:idea

Type is:WORD1

Word is:idea

.....Scanner was called...

>>>>>Word is:yasashii

Type is:WORD1

Word is:yasashii

.....Scanner was called...

>>>>>Word is:easy

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:easy

.....Scanner was called...

>>>>>Word is:muzukashii

Type is:WORD1

Word is:muzukashii

.....Scanner was called...

>>>>>Word is:difficult

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:difficult

.....Scanner was called...

>>>>>Word is:ureshii

Type is:WORD1

Word is:ureshii

.....Scanner was called...

>>>>>Word is:pleased

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:pleased

.....Scanner was called...

>>>>>Word is:shiawase

Type is:WORD1

Word is:shiawase

.....Scanner was called...

>>>>>Word is:happy

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:happy

.....Scanner was called...

>>>>>Word is:kanashii

Type is:WORD1

Word is:kanashii

.....Scanner was called...

>>>>>Word is:sad

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:sad

.....Scanner was called...

>>>>>Word is:omoi

Type is:WORD1

Word is:omoi

.....Scanner was called...

>>>>>Word is:heavy

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:heavy

.....Scanner was called...

>>>>>Word is:oishii

Type is:WORD1

Word is:oishii

.....Scanner was called...

>>>>>Word is:delicious

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:delicious

.....Scanner was called...

>>>>>Word is:tennen

Type is:WORD1

Word is:tennen

.....Scanner was called...

>>>>>Word is:natural

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:natural

.....Scanner was called...

>>>>>Word is:nakI

Type is:WORD2

Word is:nakI

.....Scanner was called...

>>>>>Word is:cry

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:cry

.....Scanner was called...

>>>>>Word is:ikI

Type is:WORD2

Word is:ikI

.....Scanner was called...

>>>>>Word is:go*

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:go*

.....Scanner was called...

>>>>>Word is:tabE

Type is:WORD2

Word is:tabE

.....Scanner was called...

>>>>>Word is:eat

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:eat

.....Scanner was called...

>>>>>Word is:ukE

Type is:WORD2

Word is:ukE

.....Scanner was called...

>>>>>Word is:take*

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:take*

.....Scanner was called...

>>>>>Word is:kakI

Type is:WORD2

Word is:kakI

.....Scanner was called...

>>>>>Word is:write

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:write

.....Scanner was called...

>>>>>Word is:yomI

Type is:WORD2

Word is:yomI

.....Scanner was called...

>>>>>Word is:read

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:read

.....Scanner was called...

>>>>>Word is:nomI

Type is:WORD2

Word is:nomI

.....Scanner was called...

>>>>>Word is:drink

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:drink

.....Scanner was called...

>>>>>Word is:agE

Type is:WORD2

Word is:agE

.....Scanner was called...

>>>>>Word is:give

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:give

.....Scanner was called...

>>>>>Word is:moraI

Type is:WORD2

Word is:moraI

.....Scanner was called...

>>>>>Word is:receive

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:receive

.....Scanner was called...

>>>>>Word is:butsI

Type is:WORD2

Word is:butsI

.....Scanner was called...

>>>>>Word is:hit

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:hit

.....Scanner was called...

>>>>>Word is:kerI

Type is:WORD2

Word is:kerI

.....Scanner was called...

>>>>>Word is:kick

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:kick

.....Scanner was called...

>>>>>Word is:shaberI

Type is:WORD2

Word is:shaberI

.....Scanner was called...

>>>>>Word is:talk

>>>>>Lexical Error: The string is not in my language

Type is:ERROR

Word is:talk

.....Scanner was called...

>>>>>Word is:eofm

End of file is encountered.

[thaet001@empress cs421]$

4 – Factored Rules with New Non Terminal Names

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

5 – Updated Parser Code for Translation (translator.cpp) Ends on page 47

```
//=================================================
// File parser.cpp written by Group Number: 13
//=================================================
// ----- Four Utility Functions and Globals ----------------------------------
// ** Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
//    to display syntax error messages as specified by me.
// Type of error: match fails
// Done by: Jorge Diaz
// Expect token was missing type or got inserted before the expected token
void syntaxerror1(string saved_t, tokentype tt){
   cout << "SYNTAX ERROR: expected " << tokenName[tt] << " but found " << saved_t << endl;
   exit(1);
}

//tokens, there is no way to choose a path.
// So exit the program as soon as this type of syntax error is found
// Type of error: switch case default error - if we do not see any of the expected
// Done by: Jorge Diaz
void syntaxerror2(string parserFunction, string saved_lex) {
   cout << "SYNTAX ERROR: unexpected " << saved_lex << " found in " << parserFunction <<
endl;
   exit(1);
}

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

```
// Purpose: See the next token without eat it
// Done by: Nam Cuong Tran
tokentype next_token()
{
   // if reach end of file exit
  if(saved_lexeme == "eofm") exit(0);
  if (!token_available)   // if there is no saved token yet
  {
     scanner(saved_token, saved_lexeme);  // call scanner to grab a new token
                                  // 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
  }
  return saved_token;    // return the saved token
}

// Purpose: Eat up the token and release the flag
// Done by: Nam Cuong Tran
bool match(tokentype expected)
{
   // call the function
  tokentype guess = next_token();
  // if reach end of file then exit
  if(guess == EOFM) exit(0);
  if (guess != expected)   // mismatch has occurred with the next token
  {
      // calls a syntax error function here to generate a syntax error message here and do
recovery
       syntaxerror1(saved_lexeme, expected);
```

```
  }
  else  // match has occurred
  {
      token_available = false;   // eat up the token
      cout << "Matched " << tokenName[expected] << endl; // print the the token type
      return true;                    // say there was a match
  }
  return false;
}

// ----- RDP functions - one per non-term ------------------
// Done by: Nam Cuong Tran
// ** 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
// story to start the bnf
void story()
{
  cout << "Processing <story>" << endl;
  // call s()
  s();
}
// 2 <s> ::= [CONNECTOR #getEword#  #gen(CONNECTOR)#] <noun> #getEword#  SUBJECT
#gen(ACTOR)# <after subject>
// s to start bnf
void s()
{
  cout << "Processing <s>" << endl;
   // if connector is next token then match since it is optional
```

```
   if(next_token() == CONNECTOR){

      match(CONNECTOR);
      // call 2 function base on grammar
      getEword();
      gen("CONNECTOR");
   }
   // call noun
   noun();
   // call getEword() base on grammar
   getEword();
   // match subject
   match(SUBJECT);
   // call gen("ACTOR") base on grammar
   gen("ACTOR");
   // call after subject
   after_subject();
}
// 3 <after subject> ::= <verb> #getEword#  #gen(ACTION)# <tense> #gen(TENSE)# PERIOD |
<noun> #getEword# <after noun>
// after subject non ter
void after_subject()
{
  cout << "Processing <after_subject>" << endl;
  switch(next_token())
  {
      // case verb then call verb tense and match period
    case WORD2:
       verb();
```

```
      // call getEword() base on grammar
      getEword();
      gen("ACTION");
      tense();
      // call gen("TENSE") base on grammar
      gen("TENSE");
      match(PERIOD);
      break;
     // case noun then call noun and after_noun functuon
    case WORD1: case PRONOUN:
      noun();
      // call getEword() base on grammar
      getEword();
      after_noun();
      break;
      // otherwise generate syntax error
    default:
      syntaxerror2("after_subject", saved_lexeme);
      break;
  }
}
// 4 <after noun> ::= <be> #gen(DESCRIPTION)# #gen(TENSE)# PERIOD |
// DESTINATION  #gen(TO)# <verb> #getEword#  #gen(ACTION)# <tense> #gen(TENSE)#
PERIOD | OBJECT #gen(OBJECT)# <after object>
void after_noun()
{
  cout << "Processing <after_noun>" << endl;
  switch (next_token())
  {
```

```
  // case be then call be and match period
case IS: case WAS:
   be();
   // call gen("DESCRIPTION") and gen("TENSE")
   gen("DESCRIPTION");
   gen("TENSE");
   match(PERIOD);
   break;
   // case destination then match it and call after_subject
case DESTINATION:
   match(DESTINATION);
   // call gen("TO") and getEword() base on grammar
   gen("TO");
   verb();
   getEword();
   gen("ACTION");
   tense();
   gen("TENSE");
   if(next_token() == PERIOD) match(PERIOD);
   break;
   // case object then match it then call after_object
case OBJECT:
   match(OBJECT);
   gen("OBJECT");
   after_object();
   break;
  // otherwise generate error
default:
```

```
        syntaxerror2("after_noun", saved_lexeme);
        break;
    }
}
// <after object> ::= <verb> #getEword#  #gen(ACTION)# <tense> #gen(TENSE)# PERIOD |
// <noun> #getEword# DESTINATION #gen(TO)# <verb> #getEword#  #gen(ACTION)# <tense>
#gen(TENSE)# PERIOD
void after_object()
{
   cout << "Processing <after_object>" << endl;
   switch (next_token())
   {
    // case verb then call verb tense and period
    case WORD2:
      verb();
      // call getEword() and gen(line) base on grammar
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      if(next_token() == PERIOD) match(PERIOD);
      break;
      // case noun then call noun match(destination) verb tense and period
    case WORD1: case PRONOUN:
      noun();
      // call getEword() and gen(line) base on grammar
      getEword();
      match(DESTINATION);
      gen("TO");
```

```
      verb();
      // call getEword() and gen(line) base on grammar
      getEword();
      gen("ACTION");
      tense();
      gen("TENSE");
      if(next_token() == PERIOD) match(PERIOD);
      break;
      // otherwise generate error
    default:
      syntaxerror2("after_object", saved_lexeme);
      break;
   }
}
void noun()
{
   cout << "Processing <noun>" << endl;
   switch (next_token())
   {
    // word1 then match with word1
    case WORD1:
      match(WORD1);
      break;
    // pronoun then match with pronoun
    case PRONOUN:
      match(PRONOUN);
      break;
      // otherwise generate error
```

```
    default:
      syntaxerror2("noun", saved_lexeme);
      break;
   }
}
void verb()
{
    // if token is word2 then match it
   cout << "Processing <verb>" << endl;
   if(next_token() == WORD2) match(WORD2);
}
void be()
{
   cout << "Processing <be>" << endl;
   switch (next_token())
   {
    // match IS in case IS
   case IS:
      match(IS);
      break;
    // match WAS in case WAS
   case WAS:
      match(WAS);
      break;
      // otherwise generate error
   default:
      syntaxerror2("be", saved_lexeme);
      break;
```

```
  }
}
void tense()
{
   // tense include <tense> := VERBPAST  | VERBPASTNEG | VERB | VERBNEG
  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;
    // if not in tense then generate error
  default:
    syntaxerror2("tense", saved_lexeme);
    break;
  }
}

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

```
// File translator.cpp written by Group Number: 13
//================================================




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

ofstream fout;

// ** Declare Lexicon (i.e. dictionary) that will hold the content of lexicon.txt
// Make sure it is easy and fast to look up the translation.
// Do not change the format or content of lexicon.txt
//  Done by:
map<string, string> english_word_table;

// ** Additions to parser.cpp here:
//    getEword() - using the current saved_lexeme, look up the English word
//                 in Lexicon if it is there -- save the result
//                 in saved_E_word
//  Done by: Nam Cuong Tran

void getEword()
{
   // if there is an input then execute the code
  if(saved_lexeme != "")
  {
    // loop through the table to find the matched english word
    for(int i = 0; i < english_word_table.size(); i++)
```

```
   {
      // if found assign it to saved_E_word
      if(english_word_table[saved_lexeme] != "")
      {
        saved_E_word = english_word_table[saved_lexeme];
      }
      // if not save the original word
      else
      {
        saved_E_word = saved_lexeme;
      }
   }
  }
  else
  {
    // otherwise exit with No input error
    cout << "No input" << endl;
    exit(1);
  }
}

//    gen(line_type) - using the line type,
//                  sends a line of an IR to translated.txt
//                  (saved_E_word or saved_token is used)
//  Done by: Jorge Diaz

void gen(string line){
   //if translated.txt file is closed then pass
```

```
   //we want gen to produce nothing when translated.txt is closed
   //e.g "TENSE: " + VERBPASTNEG
   if(!fout.is_open()){

   }
   // if line is TENSE then write saved_token to translated to translated.txt
   // otherwise, write saved_E_word to translated.txt
   // e.g "CONNECTOR: " + HOWEVER
   else{
      if(line == "TENSE"){
         fout << "TENSE: " + tokenName[saved_token] << endl;
         fout << endl;
      }
      else{
         fout << line + ": " + saved_E_word << endl;
      }
   }

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

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

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

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

// The final test driver to start the translator
// Done by:  Corbin Thaete

int main()
{
  //opens the lexicon.txt file and reads it into Lexicon
  string filename1 = "lexicon.txt";
  fin.open(filename1.c_str(),  fstream::in);
  string word1, word2;
  while(fin >> word1)
    {
       // read each word in and end if reach eofm
     if(word1 == "eofm") break;
     fin >> word2;
     // insert into the map
     english_word_table.insert(pair<string, string>(word1, word2));
    }
  fin.close();   // closes lexicon.txt

  fout.open("translated.txt"); // opens the output file translated.txt
  // if cant open the file then cout
  if(!fout)
  {
    cout << "Error in opening translated.txt" << endl;
  }
```

```
  string filename;
  cout << "Enter the input file name: ";
  cin >> filename;
  fin.open(filename.c_str(),  fstream::in);
  while(true)
   {
      // calls the <story> to start parsing
     story();
     // if reach endof file then break
     if(saved_lexeme == "eofm") break;
   }
  fin.close(); // closes the input file
  fout.close(); //closes traslated.txt
}// end
```

6 – Final Test Results

ACTOR: I/me

DESCRIPTION: rika

TENSE: IS

ACTOR: I/me

DESCRIPTION: teacher

TENSE: IS

ACTOR: rika

OBJECT: meal

ACTION: eat

TENSE: VERB

ACTOR: I/me

OBJECT: test

TO: student

ACTION: give

TENSE: VERBPAST

CONNECTOR: However

ACTOR: student

ACTION: enjoy

TENSE: VERBPASTNEG

CONNECTOR: Therefore

ACTOR: I/me

DESCRIPTION: sad

TENSE: WAS

CONNECTOR: Then

ACTOR: rika

TO: restroom

ACTION: go

TENSE: VERBPAST

ACTOR: rika

ACTION: cry

TENSE: VERBPAST