CS421 Japanese to English Translator

Group #8

Member A: Thomas Griffin

Member B: Jan Ocampo

Member C: Josh Wilber

***State Of Program***

The translator is working almost as intended and we completed the functions that were necessary. The program compiles and runs but there are some semantic routines that are not functioning properly. These cases include:  
  
TO: student  
ACTION: give

TO: student  
ACTION: student

TO: restroom  
ACTION: restroom

CONNECTOR: I/me  
Actor: However

*Contributions*Jan and Josh were responsible for creating most of the grammar rules, routines, and functions while Thomas contributed by performing extensive debugging and main methods.  
  
*Table of Contents*  
 - *DFA Page No. 3*

*- Scanner Code Page No. 4*

*- Scanner Output Page No. 12*

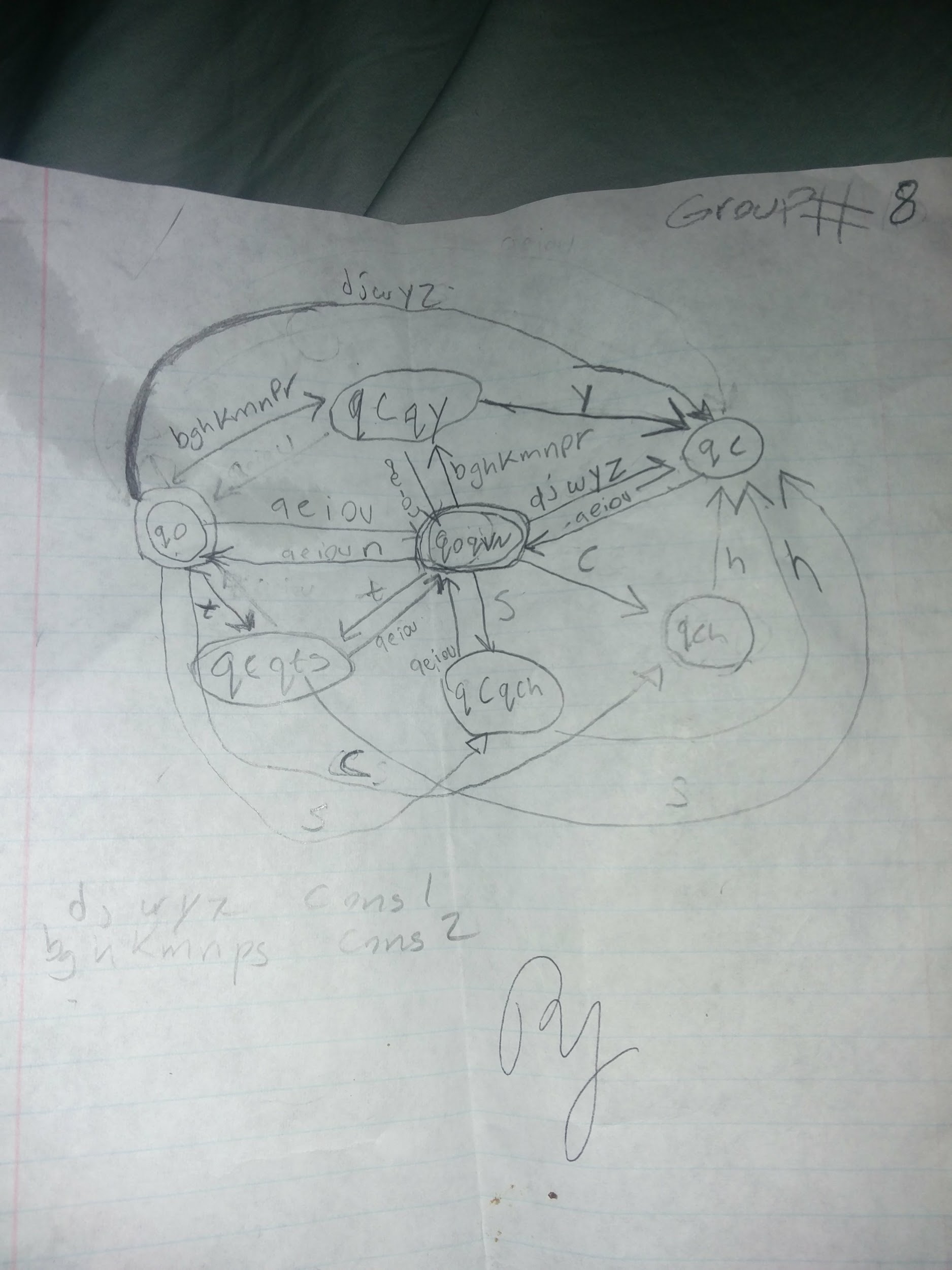
*- Grammar Rules with Non Terminals Page No. 16*

*- Parser Code Page No. 17*

*- Parser Output Page No. 29*

*- Updated Grammar Rules Page No. 28  
- Updated Parser Code Page No. 39*

*- Updated Parser Output Page No. 56*

***DFA ***

***Scanner Code***

#ifndef SCANNER\_H

#define SCANNER\_H

#include<iostream>

#include<fstream>

#include<string>

// enum

enum token\_type {ERROR, WORD1, WORD2, PERIOD, RESERVED, CONNECTOR, DESTINATION, VERB, VERBPAST, VERBPASTNEG, VERBNEG, IS, WAS, PRONOUN, OBJECT, SUBJECT, NOUN, EOFM};

// Token array

extern std::string tokens[5];

// reserved string

extern std::string reservedWords[38];

//File stream to open file

extern std::ifstream fin;

// Checks if the character is a vowel

bool vowel(char v);

bool cons1(char c);

bool cons2(char c);

bool periodDfa(std::string c);

bool wordDfa(std::string s);

int checkReserved(std::string s);

void scanner(token\_type& a, std::string& w, int& resIndex);

#endif

***// END SCANNER.H***

#include<iostream>

#include<fstream>

#include<string>

#include<iomanip>

#include"scanner.h"

using namespace std;

// Token array

std::string tokens[5] = {"ERROR", "WORD1", "WORD2", "PERIOD"};

// reserved string

std::string reservedWords[38]= {"masu", "VERB", "masen", "VERBNEG", "mashita", "VERBPAST", "masendeshita",

"VERBPASTNEG", "desu", "IS", "deshita", "WAS", "o", "OBJECT", "wa", "SUBJECT", "ni", "DESTINATION",

"watashi","PRONOUN", "anata", "PRONOUN", "kare", "PRONOUN", "kanojo", "PRONOUN", "sore", "PRONOUN",

"mata", "CONNECTOR", "soshite", "CONNECTOR", "shikashi", "CONNECTOR", "dakara", "CONNECTOR", "eofm"

"EOFM"}; ;

//File stream to open file

std::ifstream fin;

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

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

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

bool vowel(char v)

{

return (v == 'a' || v == 'e' || v == 'i' || v == 'o' || v == 'u' || v == 'I' || v == 'E');

}

bool cons1(char c)

{

return (c == 'd' || c == 'j' || c == 'w' || c == 'y' || c == 'z');

}

bool cons2(char c)

{

return (c == 'b' || c == 'g' || c == 'h' || c == 'k' || c == 'm' || c == 'n' || c == 'p' || c == 'r');

}

// \*\* Add the PERIOD DFA here

// \*\* Done by: Thomas Griffin

bool periodDfa(string c)

{

int state = 0;

if(c == ".")

state = 1;

return state ==1;

}

token\_type stringToToken(string s)

{

if(s =="CONNECTOR")

return CONNECTOR;

else if(s =="VERB")

return VERB;

else if(s =="VERBPASTNEG")

return VERBPASTNEG;

else if(s =="VERBNEG")

return VERBNEG;

else if(s =="SUBJECT")

return SUBJECT;

else if(s =="VERBPAST")

return VERBPAST;

else if(s =="DESTINATION")

return DESTINATION;

else if(s =="IS")

return IS;

else if(s =="WAS")

return WAS;

else if(s =="PRONOUN")

return PRONOUN;

else if(s =="NOUN")

return NOUN;

else if(s =="OBJECT")

return OBJECT;

else if(s == "eofm")

return EOFM;

else

return ERROR;

}

// \*\* MYTOKEN DFA to be replaced by the WORD DFA

//state 0 = q0, state 1 = qoqvn, state 2 = qcqy, state 3 = qc,

//state 4 = qch, state 5 = qcqch, state 6 = qcqts

// \*\* Done by: Jan Ocampo

// \*\* RE:

bool wordDfa(string s)

{

int state = 0;

int charpos = 0;

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

{

if(state == 0 && vowel(s[charpos]))

state = 1;

else if(state == 0 && cons1(s[charpos]))

state = 3;

else if(state == 0 && cons2(s[charpos]))

state = 2;

else if(state == 0 && s[charpos] == 't')

state = 6;

else if(state == 0 && s[charpos] == 'c')

state = 4;

else if(state == 0 && s[charpos] == 's')

state = 5;

else if(state == 1 && (vowel(s[charpos]) || s[charpos] == 'n' ))

state = 0;

else if(state == 1 && cons2(s[charpos]))

state = 2;

else if(state == 1 && cons1(s[charpos]))

state = 3;

else if(state == 1 && s[charpos] == 'c')

state = 4;

else if(state == 1 && s[charpos] == 's')

state = 5;

else if(state == 1 && s[charpos] == 't')

state = 6;

else if(state == 2 && vowel(s[charpos]))

state = 1;

else if(state == 2 && s[charpos] == 'y')

state = 3;

else if(state == 3 && vowel(s[charpos]))

state = 1;

else if(state == 4 && s[charpos] == 'h')

state = 3;

else if(state == 5 && vowel(s[charpos]))

state = 1;

else if(state == 5 && s[charpos] == 'h')

state = 3;

else if(state == 6 && vowel(s[charpos]))

state = 1;

else if(state = 6 && s[charpos] == 's')

state = 3;

else return false;

charpos++;

}//end of while

// where did I end up????

return (state == 0 || state == 1);

}

// \*\* Update the tokentype to be WORD1, WORD2, PERIOD and ERROR.

//enum tokentype {ERROR, WORD1, WORD2, PERIOD, RESERVED };

//list of japanese words and their english counterparts, if a japanese word is at index n its

//english counterpart is at n+1. Japanese words are in the odd indices while english words are

//in the even indices.

//list of reserved words in japanese. If the japanese word is at n then its part of speech is stored

//at n+1. Japanese words are at odd indices while their parts of speech are at even indices.

int checkReserved(string s)

{

int i = 0;

for(i =0; i < 38; i+=2)

{

if (reservedWords[i] == s)

return i + 1;

}

return 0;

}

// \*\* Need the lexicon to be set up here (to be used in Part C)

// \*\* Need the reservedwords list to be set up here

// \*\* Do not require any file input for these.

//ifstream fin;

///string tokens[5] = {"ERROR", "WORD1", "WORD2", "PERIOD"};

// Scanner processes only one word each time it is called

// \*\* Done by: Jan Ocampo

void scanner(token\_type& a, string& w, int& resIndex)

{

// \*\* Grab the next word from the file

fin >> w;

if(w == "eofm")

{

a = EOFM;

return;

}

/\*

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)

\*/

if(wordDfa(w))

{

resIndex = checkReserved(w);

if(resIndex)

a = stringToToken(reservedWords[resIndex]);

else if(w[w.length()-1] <= 'Z')

a = WORD2;

else

a = WORD1;

}

else if(periodDfa(w))

{

a = PERIOD;

}

else

{

cout << "Lexical error encounter: "<< w << " is not a valid word\n";

a = ERROR;

}

}//the end

/\*

// The test driver to call the scanner repeatedly

// \*\* Done by: Josh Wilber

int main()

{

tokentype thetype;

string theword;

string fname;

setfill(' ');

cout << "Enter the name of your input file: ";

cin >> fname;

int resIndex = 0;

fin.open(fname.c\_str());

if(!fin)

{

cout << "Error opening file!\n";

return 0;

}

cout << setw(14) << left << "String" << setw(13) << "TOKEN-TYPE" << setw(14) << "\n=====" << setw(14) << "===========" <<endl;

/\*

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.

while (true)

{

scanner(thetype, theword, resIndex); // call the scanner

cout << setw(14) << theword;

if (theword == "eofm")

break;

if(thetype == RESERVED)

cout <<setw(14) << reservedWords[resIndex];

else

cout << setw(14) << tokens[thetype];

cout << endl;

// \*\* display the actual type instead of a number

}

// \*\* close the input file

}\*/

***Scanner Output***

Script started on Wed 13 Dec 2017 09:14:51 PM PST

]0;griff073@empress:~/cs421/rikaFiles/CS421Progs/ScannerFiles[griff073@empress ScannerFiles]$ ./a.out

Enter the name of your input file: scannertest1

String TOKEN-TYPE

===== ===========

watashi PRONOUN

wa SUBJECT

rika WORD1

desu IS

. PERIOD

watashi PRONOUN

wa SUBJECT

sensei WORD1

desu IS

. PERIOD

watashi PRONOUN

wa SUBJECT

ryouri WORD1

o OBJECT

yarI WORD2

masu VERB

. PERIOD

watashi PRONOUN

wa SUBJECT

gohan WORD1

o OBJECT

seito WORD1

ni DESTINATION

agE WORD2

mashita VERBPAST

. PERIOD

shikashi CONNECTOR

seito WORD1

wa SUBJECT

yorokobI WORD2

masendeshita VERBPASTNEG

. PERIOD

dakara CONNECTOR

watashi PRONOUN

wa SUBJECT

kanashii WORD1

deshita WAS

. PERIOD

soshite CONNECTOR

watashi PRONOUN

wa SUBJECT

toire WORD1

ni DESTINATION

ikI WORD2

mashita VERBPAST

. PERIOD

watashi PRONOUN

wa SUBJECT

nakI WORD2

mashita VERBPAST

. PERIOD

eofm

]0;griff073@empress:~/cs421/rikaFiles/CS421Progs/ScannerFiles[griff073@empress ScannerFiles]$ ./a.out

Enter the name of your input file: scannertest2

String TOKEN-TYPE

===== ===========

daigaku WORD1

college Lexical Error: college is not a valid lexical token

ERROR

kurasu WORD1

class Lexical Error: class is not a valid lexical token

ERROR

hon WORD1

book Lexical Error: book is not a valid lexical token

ERROR

tesuto WORD1

test Lexical Error: test is not a valid lexical token

ERROR

ie WORD1

home\* Lexical Error: home\* is not a valid lexical token

ERROR

isu WORD1

chair Lexical Error: chair is not a valid lexical token

ERROR

seito WORD1

student Lexical Error: student is not a valid lexical token

ERROR

sensei WORD1

teacher Lexical Error: teacher is not a valid lexical token

ERROR

tomodachi WORD1

friend Lexical Error: friend is not a valid lexical token

ERROR

jidoosha WORD1

car Lexical Error: car is not a valid lexical token

ERROR

gyuunyuu WORD1

milk Lexical Error: milk is not a valid lexical token

ERROR

sukiyaki WORD1

tenpura WORD1

sushi WORD1

biiru WORD1

beer Lexical Error: beer is not a valid lexical token

ERROR

sake WORD1

tokyo WORD1

kyuushuu WORD1

Osaka Lexical Error: Osaka is not a valid lexical token

ERROR

choucho WORD1

butterfly Lexical Error: butterfly is not a valid lexical token

ERROR

an WORD1

idea WORD1

yasashii WORD1

easy Lexical Error: easy is not a valid lexical token

ERROR

muzukashii WORD1

difficult Lexical Error: difficult is not a valid lexical token

ERROR

ureshii WORD1

pleased Lexical Error: pleased is not a valid lexical token

ERROR

shiawase WORD1

happy Lexical Error: happy is not a valid lexical token

ERROR

kanashii WORD1

sad Lexical Error: sad is not a valid lexical token

ERROR

omoi WORD1

heavy Lexical Error: heavy is not a valid lexical token

ERROR

oishii WORD1

delicious Lexical Error: delicious is not a valid lexical token

ERROR

tennen WORD1

natural Lexical Error: natural is not a valid lexical token

ERROR

nakI WORD2

cry Lexical Error: cry is not a valid lexical token

ERROR

ikI WORD2

go\* Lexical Error: go\* is not a valid lexical token

ERROR

tabE WORD2

eat Lexical Error: eat is not a valid lexical token

ERROR

ukE WORD2

take\* Lexical Error: take\* is not a valid lexical token

ERROR

kakI WORD2

write Lexical Error: write is not a valid lexical token

ERROR

yomI WORD2

read Lexical Error: read is not a valid lexical token

ERROR

nomI WORD2

drink Lexical Error: drink is not a valid lexical token

ERROR

agE WORD2

give Lexical Error: give is not a valid lexical token

ERROR

moraI WORD2

receive Lexical Error: receive is not a valid lexical token

ERROR

butsI WORD2

hit Lexical Error: hit is not a valid lexical token

ERROR

kerI WORD2

kick Lexical Error: kick is not a valid lexical token

ERROR

shaberI WORD2

talk Lexical Error: talk is not a valid lexical token

ERROR

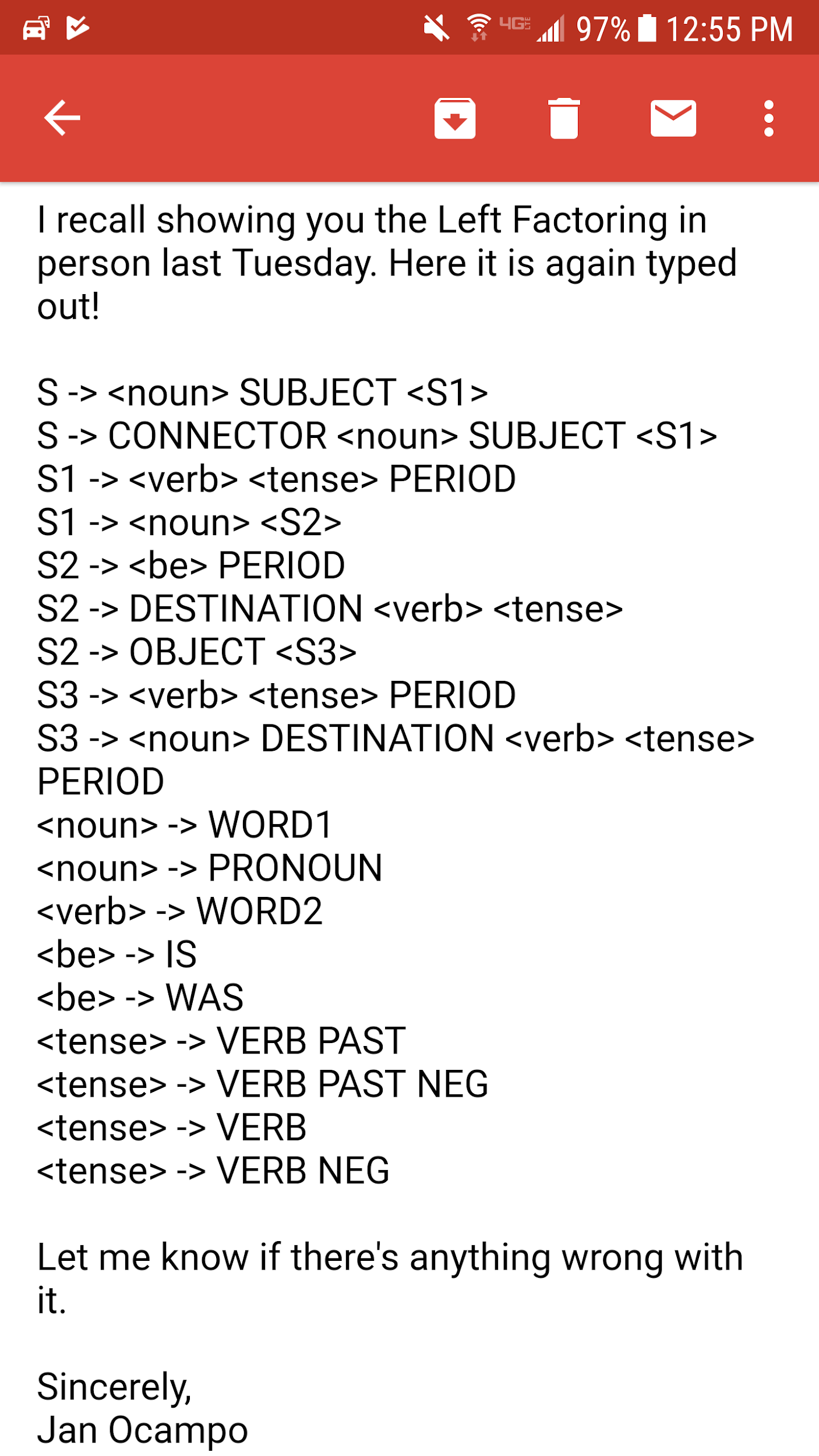
eofm

]0;griff073@empress:~/cs421/rikaFiles/CS421Progs/ScannerFiles[griff073@empress ScannerFiles]$ exit

exit

Script done on Wed 13 Dec 2017 09:15:58 PM PST

**// END SCANNER OUTPUT**

***Grammar Rules with Non Terminals***

***Parser Code***

#include<iostream>

#include<fstream>

#include<string>

#include<cstdlib>

#include "scanner.h"

using namespace std;

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

// File parser.cpp written by Group Number: \*\*

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

// \*\* Be sure to put the name of the programmer above each function

// i.e. Done by:

// \*\* Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)

// \*\* Need the updated match and next\_token (with 2 global vars)

// \*\* Make each non-terminal into a function here

// \*\* Be sure to put the corresponding grammar rule above each function

//Global Variables

// Token array

//File stream to open file

bool token\_available; // global flag indicating whether

// we have saved a token to eat up or not

void noun();

void rule\_s();

void rule\_s1();

void rule\_s2();

void rule\_s3();

void verb();

void tense();

void be();

token\_type saved\_token; // global buffer for the scanner token

string tokenToString(token\_type t);

int resIndex = 0;

//syntax\_error1(token\_type token, string saved\_lexeme)

//Takes in 2 args, the token type and the saved lexeme

//Only called if match fails

void syntax\_error1(token\_type token, string saved\_lexeme)

{

cout << "SYNTAX ERROR: expected " << tokenToString(token) << " but found " << saved\_lexeme << "." << endl;

cout << "Halting parser...." << endl;

exit(1);

}

//syntax\_error2(string saved\_lexeme, string parser\_function)

//Takes in 2 args, the saved lexeme and the parser function that the error occurred in

//Only called if switch case fails in a parser function when goes to default

void syntax\_error2(string saved\_lexeme, string parser\_function)

{

cout << "SYNTAX ERROR: unexpected " << saved\_lexeme << " found in " << parser\_function << "." << endl;

cout << "Halting parser...." << endl;

exit(1);

}

// next\_token(void)

// Looks ahead to see what token comes next from the scanner.

// HOW: checks first to see if the token\_available flag is false.

// If so, saved\_token gets scanner() result.

// and the flag is set true.

// Thus a token is grabbed but is not eaten up.

// Returns the saved\_token

//

token\_type next\_token()

{

string lexeme;

if (!token\_available) // if there is no saved token from previous lookahead

{

cout << "Scanner was called . . . " << endl;

scanner(saved\_token, lexeme, resIndex); // call scanner to grab a new token

token\_available = true; // mark that fact that you have saved it

}

return saved\_token; // return the saved token

}

string tokenToString(token\_type t)

{

switch (t) {

case ERROR:

return "ERROR";

case WORD1:

return "WORD1";

case WORD2:

return "WORD2";

case PERIOD:

return "PERIOD";

case RESERVED:

return "RESERVED";

case CONNECTOR:

return "CONNECTOR";

case DESTINATION:

return "DESTINATION";

case VERB:

return "VERB";

case VERBPAST:

return "VERBPAST";

case VERBPASTNEG:

return "VERBPASTNEG";

case VERBNEG:

return "VERBNEG";

case IS:

return "IS";

case WAS:

return "WAS";

case PRONOUN:

return "PRONOUN";

case OBJECT:

return "OBJECT";

case SUBJECT:

return "SUBJECT";

case NOUN:

return "NOUN";

}

}

//match(expected)

// Checks and eats up the expected token.

// HOW: checks to see if expected is different from next\_token()

// and if so, generates a syntax error and handles the error

// else token\_available becomes false (eat up) and returns true.

bool match(token\_type expected)

{

if (next\_token() != expected) // mismatch has occurred with the next token

{

// generate a syntax error message here

// do error handling here if any

syntax\_error1(expected, tokenToString(next\_token()));

}

else // match has occurred

{

cout << "MATCHED " << tokenToString(expected) << endl;

token\_available = false; // eat up the token

return true; // say there was a match

}

}

void story()

{

cout << "Processing story. . ." << endl;

rule\_s();

while(true)

{

cout << "\n======= Processing <s> =======\n" << endl;

switch(next\_token())

{

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

match(SUBJECT); //Make sure SUBJECT is found/matched

rule\_s1(); //Process <s1>

break;

case CONNECTOR: //If next word is CONNECTOR

match(CONNECTOR);//Make sure CONNECTOR is found/matched

noun(); //Process noun

match(SUBJECT); //Make sure subject is found/matched

rule\_s1(); //Process <s1>

break;

case EOFM:

cout << "END OF FILE REACHED" << endl;

exit(1);

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s"); //If next\_token is not what was expected, syntax error

}

}

}

//Rule <s>: Has 2 choices

//<s> -> <noun> SUBJECT <s1>

//<s> -> CONNECTOR <noun> SUBJECT <s1>

void rule\_s()

{

cout << "\n======= Processing <s> =======\n" << endl;

switch(next\_token()) //Look ahead for next token

{

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

match(SUBJECT); //Make sure SUBJECT is found/matched

rule\_s1(); //Process <s1>

break;

case CONNECTOR: //If next word is CONNECTOR

match(CONNECTOR);//Make sure CONNECTOR is found/matched

noun(); //Process noun

match(SUBJECT); //Make sure subject is found/matched

rule\_s1(); //Process <s1>

break;

case EOFM:

cout << "END OF FILE REACHED" << endl;

exit(1);

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s

//Rule <s1>: Has 2 choices

//<s1> -> <verb> <tense> PERIOD

//<s1> -> <noun> <s2>

void rule\_s1()

{

cout << "Processing <rule\_s1>" << endl;

switch(next\_token()) //Look ahead for next token

{

case WORD2:

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

rule\_s2(); //Process <s2>

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s1"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s1

//Rule <s2>: Has 3 choices

//<s2> -> <be> PERIOD

//<s2> -> DESTINATION <verb> <tense>

//<s2> -> OBJECT <s3>

void rule\_s2()

{

cout << "Processing <rule\_s2>" << endl;

switch(next\_token())

{

case IS:

case WAS:

be(); //Process be

match(PERIOD); //Make sure PERIOD is found/matched

break;

case DESTINATION:

match(DESTINATION); //Make sure DESTINATION is found/matched

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case OBJECT:

match(OBJECT); //Make sure OBJECT is found/matched

rule\_s3(); //Process rule\_s3

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s2"); //If next\_token is not what was expected, syntax error

}//End switch statement

}//End rule\_s2

//Rule <s3>: Has 2 choices

//<s3> -> <verb> <tense> PERIOD

//<s3> -> <noun> DESTINATION <verb> <tense> PERIOD

void rule\_s3()

{

cout << "Processing <rule\_s3>" << endl;

switch(next\_token())

{

case WORD2:

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case NOUN:

case WORD1:

case PRONOUN:

noun(); //Process noun

match(DESTINATION); //Make sure DESTINATION is found/matched

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s3"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s3

//Rule <noun>: Has 2 options

//<noun> -> WORD1

//<noun> -> PRONOUN

void noun()

{

cout << "Processing <noun>" << endl;

switch(next\_token())

{

case WORD1:

match(WORD1); //Make sure WORD1 is found/matched

break;

case PRONOUN:

match(PRONOUN); //Make sure PRONOUN is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "noun"); //If next\_token is not what was expected, syntax error

}//End switch

}//End noun()

//Rule <verb>: Has 1 option

//<verb> -> WORD2

void verb()

{

cout << "Processing <verb>" << endl;

switch(next\_token())

{

case WORD2:

match(WORD2); //Make sure WORD2 is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "verb"); //If next\_token is not what was expected, syntax error

}//End switch

}//End verb()

//Rule <be>: Has 2 options

//<be> -> IS

//<be> -> WAS

void be()

{

cout << "Processing <be>" << endl;

switch(next\_token())

{

case IS:

match(IS); //Make sure IS is found/matched

break;

case WAS:

match(WAS); //Make sure WAS is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "be"); //If next\_token is not what was expected, syntax error

}//End switch

}//End be()

//Rule <tense>: Has 4 options

//<tense> -> VERBPAST

//<tense> -> VERBPASTNEG

//<tense> -> VERB

//<tense> -> VERBNEG

void tense()

{

cout << "Processing <tense>" << endl;

switch(next\_token())

{

case VERBPAST:

match(VERBPAST); //Make sure VERBPAST is found/matched

break;

case VERBPASTNEG:

match(VERBPASTNEG); //Make sure VERBPASTNEG is found/matched

break;

case VERB:

match(VERB); //Make sure VERB is found/matched

break;

case VERBNEG:

match(VERBNEG); //Make sure VERBNEG is found/matched

default:

syntax\_error2(tokenToString(saved\_token), "tense"); //If next\_token is not what was expected, syntax error

}//End switch

}//End tense()

// The test driver to start the parser

// Done by: Thomas Griffin

int main()

{

string fileName;

//Asks user for name of file to open

cout << "Enter the name of the file to open: ";

cin >> fileName;

cout << "Opening file " << fileName << "...\n";

fin.open(fileName.c\_str());

//Checks if file is open successfully

//Terminates if not open

if(!fin)

{

cerr << "Unable to open file " << fileName << "...\n";

exit(1);

}

story();

//- opens the input file

//- calls the <story> to start parsing

//- closes the input file

//Closes the file

fin.close();

}// end

//\*\* should require no other input files!

***Parser Output***

Script started on Tue 28 Nov 2017 10:50:54 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: test

Opening file test...

Processing story. . .

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

MATCHED IS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

MATCHED IS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

MATCHED OBJECT

Processing <rule\_s3>

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 <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

MATCHED OBJECT

Processing <rule\_s3>

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

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

Scanner was called . . .

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED WORD1

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <verb>

MATCHED WORD2

Processing <tense>

Scanner was called . . .

MATCHED VERBPASTNEG

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED PRONOUN

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

MATCHED WAS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED WORD1

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

MATCHED DESTINATION

Processing <verb>

Scanner was called . . .

MATCHED WORD2

Processing <tense>

Scanner was called . . .

MATCHED VERBPAST

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <verb>

MATCHED WORD2

Processing <tense>

Scanner was called . . .

MATCHED VERBPAST

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

END OF FILE REACHED

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 10:51:20 AM PST

**// End Parser Output 1**

Script started on Tue 28 Nov 2017 11:05:11 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest2

Opening file partBtest2...

Processing story. . .

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

Scanner was called . . .

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED PRONOUN

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

MATCHED IS

Scanner was called . . .

SYNTAX ERROR: expected PERIOD but found WORD1.

Halting parser....

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 11:05:33 AM PST

**// End Parser Output 2**

Script started on Tue 28 Nov 2017 11:05:47 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest3

Opening file 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 WORD1.

Halting parser....

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 11:06:14 AM PST

**// End Parser Output 3**

Script started on Tue 28 Nov 2017 11:08:01 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest4

Opening file partBtest4...

Processing story. . .

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

SYNTAX ERROR: unexpected VERBPAST found in rule\_s2.

Halting parser....

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 11:08:26 AM PST

**// End Parser Output 4**

Script started on Tue 28 Nov 2017 11:08:43 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest5

Opening file partBtest5...

Processing story. . .

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

Scanner was called . . .

SYNTAX ERROR: unexpected SUBJECT found in rule\_s.

Halting parser....

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 11:09:19 AM PST

**// End Parser Output 5**

Script started on Tue 28 Nov 2017 11:13:00 AM PST

]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest6

Opening file partBtest6...

Processing story. . .

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

Scanner was called . . .

Lexical error encounter: apple is not a valid word

SYNTAX ERROR: unexpected ERROR found in rule\_s.

Halting parser....

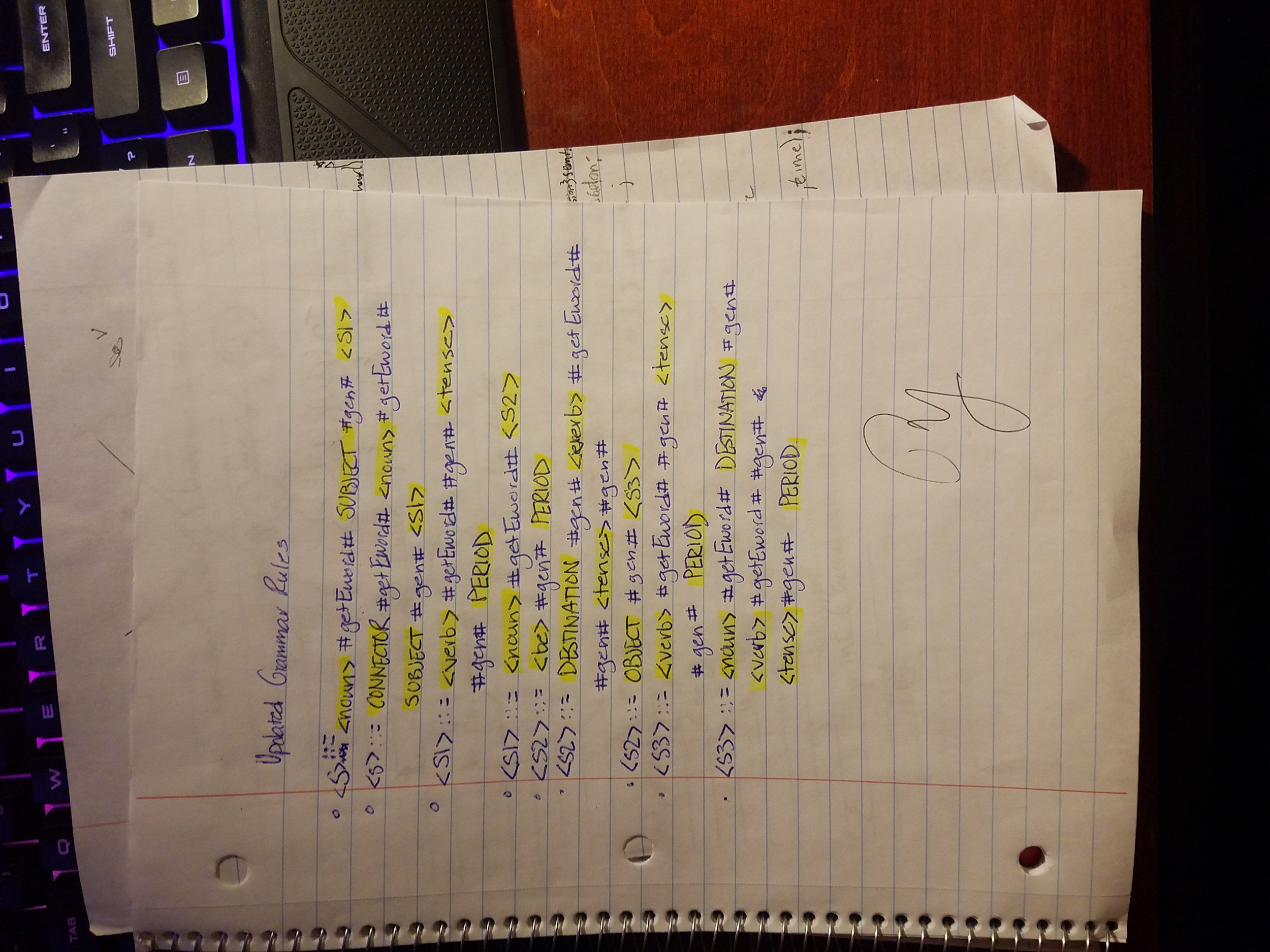
]0;griff073@empress:~/cs421/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Tue 28 Nov 2017 11:13:21 AM PST

**// End Parser Output 6**

***Updated Grammar Rules***

******

***Updated Parser Code (Translator)***

#include<iostream>

#include<fstream>

#include<string>

#include<cstdlib>

#include "scanner.h"

using namespace std;

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

// File parser.cpp written by Group Number: 8

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

// \*\* Be sure to put the name of the programmer above each function

// i.e. Done by:

// \*\* Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)

// \*\* Need the updated match and next\_token (with 2 global vars)

// \*\* Make each non-terminal into a function here

// \*\* Be sure to put the corresponding grammar rule above each function

//File stream to open file

//Global variables

token\_type saved\_token; // global buffer for the scanner token

string tokenToString(token\_type t);

int resIndex = 0;

bool token\_available; // global flag indicating whether

// we have saved a token to eat up or not

string nouns[40] = {"watashi", "I/me", "daigaku", "college", "kurasu", "class", "hon", "book", "tesuto", "test", "ie", "home", "isu", "chair", "seito", "student", "sensei", "teacher",

"tomodachi", "friend", "jidoosha", "car", "gyuunyuu", "milk", "biiru", "beer", "choucho", "butterfly", "ryouri", "cooking", "toire", "restroom", "anata", "you",

"kare", "he/him", "kanojo", "she/her", "sore", "it"};

string descriptions [16] = {"yazashii", "easy", "muzukashii", "difficult", "ureshii", "pleased", "shiawase", "happy", "kanashii",

"sad", "omoi", "heavy", "oishii", "delicious", "tennen", "natural"};

string verbs [26] = {"nakI" , "cry", "ikI", "go", "tabE", "eat", "ukE", "take", "kakI", "write", "yomI", "read", "nomI", "drink",

"agE", "give", "moraI", "receive", "butsI", "hit", "kerI", "kick", "shaberI", "talk", "yarI", "do"};

string connectors[8] = {"mata", "Also", "soshite", "Then", "shikashi", "However", "dakara", "Therefore"};

string saved\_E\_word; //Saved English word

string word; //Global string to be passed to scanner

ofstream output;

//Prototypes

void getEword();

void gen();

void noun();

void rule\_s();

void rule\_s1();

void rule\_s2();

void rule\_s3();

void verb();

void tense();

void be();

//GetEWord: Coded by Joshua Wilber

//Word is the actual word token that was scanned

//How to convert the token into a string to compare with string array?

void getEword()

{

int i;

switch (saved\_token)

{

case WORD1:

case PRONOUN:

for(i = 0; i < 40; i+=2 )

{

if(nouns[i] == word)

{

saved\_E\_word = nouns[i+1]; //Returns translated word

return;

}

}

if(saved\_token == WORD1)

saved\_E\_word = "WORD1";

else if(saved\_token == PRONOUN)

saved\_E\_word = "PRONOUN";

break;

case WORD2:

for(i = 0; i < 26; i+=2)

{

if(word == verbs[i])

{

saved\_E\_word = verbs[i+1]; //Returns translated word

return;

}

}

saved\_E\_word = "WORD2";

break;

case CONNECTOR:

for(i = 0; i < 8; i+=2)

{

if(word == connectors[i])

{

saved\_E\_word = connectors[i+1]; //Returns translated word

return;

}

}

saved\_E\_word = "CONNECTOR";

break;

case IS:

case WAS:

for(i= 0; i < 16; i+=2)

{

if(word == descriptions[i])

{

saved\_E\_word = descriptions[i+1]; //Returns translated word

return;

}

}

if(saved\_token == IS)

saved\_E\_word = "IS";

else if(saved\_token == WAS)

saved\_E\_word = "WAS";

break;

}

}

//Generator: Coded by Jan OCampo

//Provides an Internal Representation for each line

void gen() {

switch(saved\_token)

{

// S

case CONNECTOR:

cout << "CONNECTOR: " << saved\_E\_word << endl;

output << "CONNECTOR: " << saved\_E\_word << endl;

break;

case SUBJECT:

cout << "ACTOR: " << saved\_E\_word << endl;

output << "ACTOR: " << saved\_E\_word << endl;

break;

// S2 Non terminals

case OBJECT:

cout << "OBJECT: " << saved\_E\_word << endl;

output << "OBJECT: " << saved\_E\_word << endl;

break;

case DESTINATION:

cout << "TO: " << saved\_E\_word << endl;

output << "TO: " << saved\_E\_word << endl;

break;

// <verb>

case WORD2:

cout << "ACTION: " << saved\_E\_word << endl;

output << "ACTION: " << saved\_E\_word << endl;

break;

// <be>

case IS:

case WAS:

cout << "DESCRIPTION: " << saved\_E\_word << endl;

output << "DESCRIPTION: " << saved\_E\_word << endl;

cout << "TENSE: " << tokenToString(saved\_token) << endl; // second part is not right we need to find out what tense pulls

output << "TENSE: " << tokenToString(saved\_token) << endl;

break;

// <tense>

case VERB:

case VERBPAST:

case VERBNEG:

case VERBPASTNEG:

cout << "TENSE: " << tokenToString(saved\_token) << endl;

output << "TENSE: " << tokenToString(saved\_token)<< endl;

break;

}

}

//Coded by Thomas Griffin

//syntax\_error1(token\_type token, string saved\_lexeme)

//Takes in 2 args, the token type and the saved lexeme

//Only called if match fails

void syntax\_error1(token\_type token, string saved\_lexeme)

{

cout << "SYNTAX ERROR: expected " << tokenToString(token) << " but found " << saved\_lexeme << "." << endl;

cout << "Halting parser...." << endl;

exit(1);

}

//Coded by Thomas Griffin

//syntax\_error2(string saved\_lexeme, string parser\_function)

//Takes in 2 args, the saved lexeme and the parser function that the error occurred in

//Only called if switch case fails in a parser function when goes to default

void syntax\_error2(string saved\_lexeme, string parser\_function)

{

cout << "SYNTAX ERROR: unexpected " << word << " found in " << parser\_function << "." << endl;

cout << "Halting parser...." << endl;

exit(1);

}

//Coded by: Jan OCampo

// next\_token(void)

// Looks ahead to see what token comes next from the scanner.

// HOW: checks first to see if the token\_available flag is false.

// If so, saved\_token gets scanner() result.

// and the flag is set true.

// Thus a token is grabbed but is not eaten up.

// Returns the saved\_token

//

token\_type next\_token()

{

string lexeme;

if (!token\_available) // if there is no saved token from previous lookahead

{

cout << "Scanner was called . . . " << endl;

scanner(saved\_token, word, resIndex); // call scanner to grab a new token

token\_available = true; // mark that fact that you have saved it

}

return saved\_token; // return the saved token

}

string tokenToString(token\_type t)

{

switch (t) {

case ERROR:

return "ERROR";

case WORD1:

return "WORD1";

case WORD2:

return "WORD2";

case PERIOD:

return "PERIOD";

case RESERVED:

return "RESERVED";

case CONNECTOR:

return "CONNECTOR";

case DESTINATION:

return "DESTINATION";

case VERB:

return "VERB";

case VERBPAST:

return "VERBPAST";

case VERBPASTNEG:

return "VERBPASTNEG";

case VERBNEG:

return "VERBNEG";

case IS:

return "IS";

case WAS:

return "WAS";

case PRONOUN:

return "PRONOUN";

case OBJECT:

return "OBJECT";

case SUBJECT:

return "SUBJECT";

case NOUN:

return "NOUN";

}

}

//match(expected)

// Checks and eats up the expected token.

// HOW: checks to see if expected is different from next\_token()

// and if so, generates a syntax error and handles the error

// else token\_available becomes false (eat up) and returns true.

bool match(token\_type expected)

{

//cout << saved\_E\_word << endl;

if (next\_token() != expected) // mismatch has occurred with the next token

{

// generate a syntax error message here

// do error handling here if any

syntax\_error1(expected, tokenToString(next\_token()));

}

else // match has occurred

{

gen();

cout << "MATCHED " << tokenToString(expected) << endl;

token\_available = false; // eat up the token

return true; // say there was a match

}

}

//Coded by Thomas Griffin

//<story> -> <s> {<s>}

void story()

{

cout << "Processing story. . ." << endl;

rule\_s();

while(true)

{

cout << "\n======= Processing <s> =======\n" << endl;

output << "\n";

switch(next\_token())

{

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

match(SUBJECT); //Make sure SUBJECT is found/matched

rule\_s1(); //Process <s1>

break;

case CONNECTOR: //If next word is CONNECTOR

match(CONNECTOR);//Make sure CONNECTOR is found/matched

noun(); //Process noun

match(SUBJECT); //Make sure subject is found/matched

rule\_s1(); //Process <s1>

break;

case EOFM:

cout << "END OF FILE REACHED" << endl;

exit(1);

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s"); //If next\_token is not what was expected, syntax error

}

}

}

//Coded by: Joshua Wilber

//Rule <s>: Has 2 choices

//<s> -> <noun> #getEword# SUBJECT #gen# <s1>

//<s> -> CONNECTOR #getEword# #gen# <noun> #getEword# SUBJECT #gen# <s1>

void rule\_s()

{

cout << "\n======= Processing <s> =======\n" << endl;

switch(next\_token()) //Look ahead for next token

{

//<noun>

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

match(SUBJECT); //Make sure SUBJECT is found/matched

rule\_s1(); //Process <s1>

break;

case CONNECTOR: //If next word is CONNECTOR

getEword();

match(CONNECTOR);//Make sure CONNECTOR is found/matched

noun(); //Process noun

match(SUBJECT); //Make sure subject is found/matched

rule\_s1(); //Process <s1>

break;

case EOFM:

cout << "END OF FILE REACHED" << endl;

exit(1);

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s

//Coded by: Jan OCampo

//Rule <s1>: Has 2 choices

//<s1> -> <verb> #getEword$ #gen# <tense> #gen# PERIOD

//<s1> -> <noun> #getEword# <s2>

void rule\_s1()

{

cout << "Processing <rule\_s1>" << endl;

switch(next\_token()) //Look ahead for next token

{

case WORD2:

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case WORD1:

case PRONOUN: //If next word is <noun>

noun(); //Process noun

rule\_s2(); //Process <s2>

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s1"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s1

//Coded by: Joshua Wilber

//Rule <s2>: Has 3 choices

//<s2> -> <be> #gen# PERIOD

//<s2> -> DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen#

//<s2> -> OBJECT #gen# <s3>

void rule\_s2()

{

cout << "Processing <rule\_s2>" << endl;

switch(next\_token())

{

case IS:

case WAS:

be(); //Process be \*Possibly cause multiple errors

match(PERIOD); //Make sure PERIOD is found/matched

break;

case DESTINATION:

match(DESTINATION); //Make sure DESTINATION is found/matched

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case OBJECT:

match(OBJECT); //Make sure OBJECT is found/matched

rule\_s3(); //Process rule\_s3

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s2"); //If next\_token is not what was expected, syntax error

}//End switch statement

}//End rule\_s2

//Coded by Thomas Griffin

//Rule <s3>: Has 2 choices

//<s3> -> <verb> #getEword# #gen# <tense> #gen# PERIOD

//<s3> -> <noun> #getEword# DESTINATION #gen# <verb> #getEword# #gen# <tense> #gen# PERIOD

void rule\_s3()

{

cout << "Processing <rule\_s3>" << endl;

switch(next\_token())

{

case WORD2:

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

case NOUN:

case WORD1:

case PRONOUN:

noun(); //Process noun

match(DESTINATION); //Make sure DESTINATION is found/matched

verb(); //Process verb

tense(); //Process tense

match(PERIOD); //Make sure PERIOD is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "rule\_s3"); //If next\_token is not what was expected, syntax error

}//End switch

}//End rule\_s3

//Coded by: Jan OCampo

//Rule <noun>: Has 2 options

//<noun> -> WORD1

//<noun> -> PRONOUN

void noun()

{

cout << "Processing <noun>" << endl;

getEword();

switch(next\_token())

{

case WORD1:

match(WORD1); //Make sure WORD1 is found/matched

break;

case PRONOUN:

match(PRONOUN); //Make sure PRONOUN is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "noun"); //If next\_token is not what was expected, syntax error

}//End switch

}//End noun()

//Coded by: Joshua Wilber

//Rule <verb>: Has 1 option

//<verb> -> WORD2

void verb()

{

cout << "Processing <verb>" << endl;

getEword();

switch(next\_token())

{

case WORD2:

match(WORD2); //Make sure WORD2 is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "verb"); //If next\_token is not what was expected, syntax error

}//End switch

}//End verb()

//Coded by Thomas Griffin

//Rule <be>: Has 2 options

//<be> -> IS

//<be> -> WAS

void be()

{

cout << "Processing <be>" << endl;

switch(next\_token())

{

case IS:

match(IS); //Make sure IS is found/matched

break;

case WAS:

match(WAS); //Make sure WAS is found/matched

break;

default:

syntax\_error2(tokenToString(saved\_token), "be"); //If next\_token is not what was expected, syntax error

}//End switch

}//End be()

//Coded by: Thomas Griffin

//Rule <tense>: Has 4 options

//<tense> -> VERBPAST

//<tense> -> VERBPASTNEG

//<tense> -> VERB

//<tense> -> VERBNEG

void tense()

{

cout << "Processing <tense>" << endl;

switch(next\_token())

{

case VERBPAST:

match(VERBPAST); //Make sure VERBPAST is found/matched

break;

case VERBPASTNEG:

match(VERBPASTNEG); //Make sure VERBPASTNEG is found/matched

break;

case VERB:

match(VERB); //Make sure VERB is found/matched

break;

case VERBNEG:

match(VERBNEG); //Make sure VERBNEG is found/matched

default:

syntax\_error2(tokenToString(saved\_token), "tense"); //If next\_token is not what was expected, syntax error

}//End switch

}//End tense()

// The test driver to start the parser

// Coded by: Joshua Wilber

int main()

{

string fileName;

output.open("translated.txt");

//Asks user for name of file to open

cout << "Enter the name of the file to open: ";

cin >> fileName;

cout << "Opening file " << fileName << "...\n";

fin.open(fileName.c\_str());

//Checks if file is open successfully

//Terminates if not open

if(!fin)

{

cerr << "Unable to open file " << fileName << "...\n";

exit(1);

}

story();

//- opens the input file

//- calls the <story> to start parsing

//- closes the input file

//Closes the files

fin.close();

output.close();

}// end

//\*\* should require no other input files!

***Updated Parser Output (Translator)***

Script started on Thu 14 Dec 2017 05:02:34 PM PST

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest1

Opening file partBtest1...

Processing story. . .

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

ACTOR: I/me

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

DESCRIPTION: WORD1

TENSE: IS

MATCHED IS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

ACTOR: I/me

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

DESCRIPTION: teacher

TENSE: IS

MATCHED IS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Scanner was called . . .

ACTOR: WORD1

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

OBJECT: WORD1

MATCHED OBJECT

Processing <rule\_s3>

Scanner was called . . .

Processing <verb>

ACTION: eat

MATCHED WORD2

Processing <tense>

Scanner was called . . .

TENSE: VERB

MATCHED VERB

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

ACTOR: I/me

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

OBJECT: test

MATCHED OBJECT

Processing <rule\_s3>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Scanner was called . . .

TO: student

MATCHED DESTINATION

Processing <verb>

Scanner was called . . .

ACTION: student

MATCHED WORD2

Processing <tense>

Scanner was called . . .

TENSE: VERBPAST

MATCHED VERBPAST

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

CONNECTOR: student

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED WORD1

Scanner was called . . .

ACTOR: However

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <verb>

ACTION: WORD2

MATCHED WORD2

Processing <tense>

Scanner was called . . .

TENSE: VERBPASTNEG

MATCHED VERBPASTNEG

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

CONNECTOR: WORD2

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED PRONOUN

Scanner was called . . .

ACTOR: Therefore

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

DESCRIPTION: WORD1

TENSE: WAS

MATCHED WAS

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

CONNECTOR: WORD1

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED WORD1

Scanner was called . . .

ACTOR: Then

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

TO: restroom

MATCHED DESTINATION

Processing <verb>

Scanner was called . . .

ACTION: restroom

MATCHED WORD2

Processing <tense>

Scanner was called . . .

TENSE: VERBPAST

MATCHED VERBPAST

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Scanner was called . . .

ACTOR: WORD1

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <verb>

ACTION: cry

MATCHED WORD2

Processing <tense>

Scanner was called . . .

TENSE: VERBPAST

MATCHED VERBPAST

Scanner was called . . .

MATCHED PERIOD

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

Scanner was called . . .

END OF FILE REACHED

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest2

Opening file partBtest2...

Processing story. . .

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

Scanner was called . . .

CONNECTOR: Then

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED PRONOUN

Scanner was called . . .

ACTOR: Then

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

Processing <be>

DESCRIPTION: WORD1

TENSE: IS

MATCHED IS

Scanner was called . . .

SYNTAX ERROR: expected PERIOD but found WORD1.

Halting parser....

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest3

Opening file partBtest3...

Processing story. . .

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

Scanner was called . . .

CONNECTOR: Therefore

MATCHED CONNECTOR

Processing <noun>

Scanner was called . . .

MATCHED PRONOUN

Scanner was called . . .

SYNTAX ERROR: expected SUBJECT but found WORD1.

Halting parser....

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest4

Opening file partBtest4...

Processing story. . .

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

Scanner was called . . .

Processing <noun>

MATCHED PRONOUN

Scanner was called . . .

ACTOR: I/me

MATCHED SUBJECT

Processing <rule\_s1>

Scanner was called . . .

Processing <noun>

MATCHED WORD1

Processing <rule\_s2>

Scanner was called . . .

SYNTAX ERROR: unexpected VERBPAST found in rule\_s2.

Halting parser....

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest5

Opening file partBtest5...

Processing story. . .

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

Scanner was called . . .

SYNTAX ERROR: unexpected SUBJECT found in rule\_s.

Halting parser....

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ ./a.out

Enter the name of the file to open: partBtest6

Opening file partBtest6...

Processing story. . .

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

Scanner was called . . .

Lexical error encounter: apple is not a valid word

SYNTAX ERROR: unexpected ERROR found in rule\_s.

Halting parser....

]0;griff073@empress:~/debug/newdebug[griff073@empress newdebug]$ exit

exit

Script done on Thu 14 Dec 2017 05:04:28 PM PST

**// END FIRST TEST**

ACTOR: I/me

DESCRIPTION: WORD1

TENSE: IS

ACTOR: I/me

DESCRIPTION: teacher

TENSE: IS

ACTOR: WORD1

OBJECT: WORD1

ACTION: eat

TENSE: VERB

ACTOR: I/me

OBJECT: test

TO: student

ACTION: student

TENSE: VERBPAST

CONNECTOR: student

ACTOR: However

ACTION: WORD2

TENSE: VERBPASTNEG

CONNECTOR: WORD2

ACTOR: Therefore

DESCRIPTION: WORD1

TENSE: WAS

CONNECTOR: WORD1

ACTOR: Then

TO: restroom

ACTION: restroom

TENSE: VERBPAST

ACTOR: WORD1

ACTION: cry

TENSE: VERBPAST

**// END TEST 2**