#include<iostream>

#include<fstream>

#include<string>

using namespace std;

#include "scanner.cpp"

//Global variables

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

tokentype saved\_token;

string saved\_lexeme;

bool token\_available = false;

//function declarations

void story();

void s();

void noun();

void be();

void verb();

void tense();

void after\_object();

void after\_subject();

void after\_noun();

/\* INSTRUCTION: Complete all \*\* parts.

You may use any method to connect this file to scanner.cpp

that you had written.

e.g. You can copy scanner.cpp here by:

cp ../ScannerFiles/scanner.cpp .

and then append the two files into one:

cat scanner.cpp parser.cpp > myparser.cpp

\*/

// File parser.cpp written by Group Number: 4 - Larry Haskell, Zachary Pownell, Stephen Merten //=================================================

// ----- 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: Lexical Error - given lexeme is not a valid token.

// Done by: \*\*

void syntaxerror1(string lexeme)

{

cout << "Lexical Error: " << lexeme << " is not a valid token." << endl;

exit(1);

}//end of sytaxerror1

// Type of error: Syntax Error - unexpected word found in token.

// Done by: \*\*

void syntaxerror2(string word, tokentype token)

{

cout << "Syntax error: unexpected " << word << " found in " << token << "." << endl;

exit(1);

}//end of syntaxerror2

// \*\* Need the updated match and next\_token with 2 global vars saved\_token and saved\_lexeme

// Purpose: To determine the next token to be read by the scanner

// Done by: Larry Haskell

tokentype next\_token()

{

if(!token\_available)

{

scanner(saved\_token, saved\_lexeme);

cout << "Scanner called using word: " << saved\_lexeme << endl;

token\_available = true;

if(saved\_token == ERROR)

{

syntaxerror1(saved\_lexeme);

}

}

return saved\_token;

}

// Purpose: Checks and eats up the expected token. 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.

// Done by: Larry Haskell

bool match(tokentype expected)

{

if (next\_token() != expected)

{

syntaxerror2(saved\_lexeme, saved\_token);

}

else

{

token\_available = false;

return true;

}

}

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

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

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

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

// Grammar: <story> ::= <s> { <s> } // stay in the loop as long as a possible start of <s> is the next\_token (note it can be CONNECTOR or WORD1 or PRONOUN)

// Done by: Larry Haskell

void story()

{

cout << "Processing <story>" << endl;

s();

while (true && (saved\_lexeme != "eofm"))

{

s();

}

cout << "Successfully parsed story" << endl;

}

//Grammar: <s> ::= [CONNECTOR] <noun> SUBJECT <after subject>

//Done by: \*\*

void s()

{

}

//Grammar: <noun> ::= WORD1 | PRONOUN

//Done by: Zach Pownell

void noun()

{

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

switch(next\_token())

{

case WORD1:

match(WORD1);

break;

case PRONOUN:

match(PRONOUN);

break;

default:

syntaxerror2(saved\_lexeme, saved\_token);

}

}

//Grammar: <verb> ::= WORD2

//Done by: Zach Pownell

void verb()

{

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

match(WORD2);

}

//Grammar: <be> ::= IS | WAS

//Done by: Larry Haskell

void be()

{

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

switch(next\_token())

{

case IS:

match(IS);

break;

case WAS:

match(WAS);

break;

default: //default error

syntaxerror2(saved\_lexeme, "be()");

}

}

//Grammar: <tense> := VERBPAST | VERBPASTNEG | VERB | VERBNEG

//Done by: Zach Pownell

void tense()

{

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

switch(next\_token())

{

case VERBPAST:

match(VERBPAST);

break;

case VERBPASTNEG:

match(VERBPASTNEG);

break;

case VERB:

match (VERB);

break;

case VERBNEG:

match(VERBNEG);

break;

default:

syntaxerror2(saved\_lexeme, saved\_token);

}

}

//Grammer: <after subject> := <verb> <tense> PERIOD | <noun> <after noun>

//Done by: Zach Pownell

void after\_subject()

{

cout << "Processing <after\_subject>" << endl;

switch(next\_token())

{

case WORD1:

case PRONOUN:

verb();

tense();

match(PERIOD);

break;

case WORD2:

noun();

after\_noun();

break;

default:

syntaxerror2(saved\_lexeme, saved\_token);

}

}

//Grammer: <after noun> := <be> PERIOD | DESTINATION <verb> <tense> PERIOD | OBJECT <after object>

//Done by: Larry Haskell

void after\_noun()

{

cout << "Processing <after\_noun>" << endl;

switch (next\_token())

{

case IS:

case WAS:

be();

match(PERIOD);

break;

case DESTINATION:

match (DESTINATION);

verb();

tense();

match(PERIOD);

break;

case OBJECT:

after\_object();

break;

default:

syntaxerror2(saved\_lexeme, saved\_token);

}

}

//Grammar: <after object> := <verb> <tense> PERIOD | <noun> DESTINATION <verb> <tense> PERIOD

//Done by: Zach Pownell

void after\_object()

{

cout << "Processing <after\_object>" << endl;

switch(next\_token())

{

case WORD1:

verb();

tense();

match(PERIOD);

break;

case WORD2:

noun();

match(DESTINATION);

verb();

tense();

match(PERIOD);

break;

}

}

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

// The new test driver to start the parser

// Done by: \*\*

string filename;

int main()

{

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

cin >> filename;

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

// Calls the <story> to start parsing

story();

// Closes the input file

fin.close();

}// end