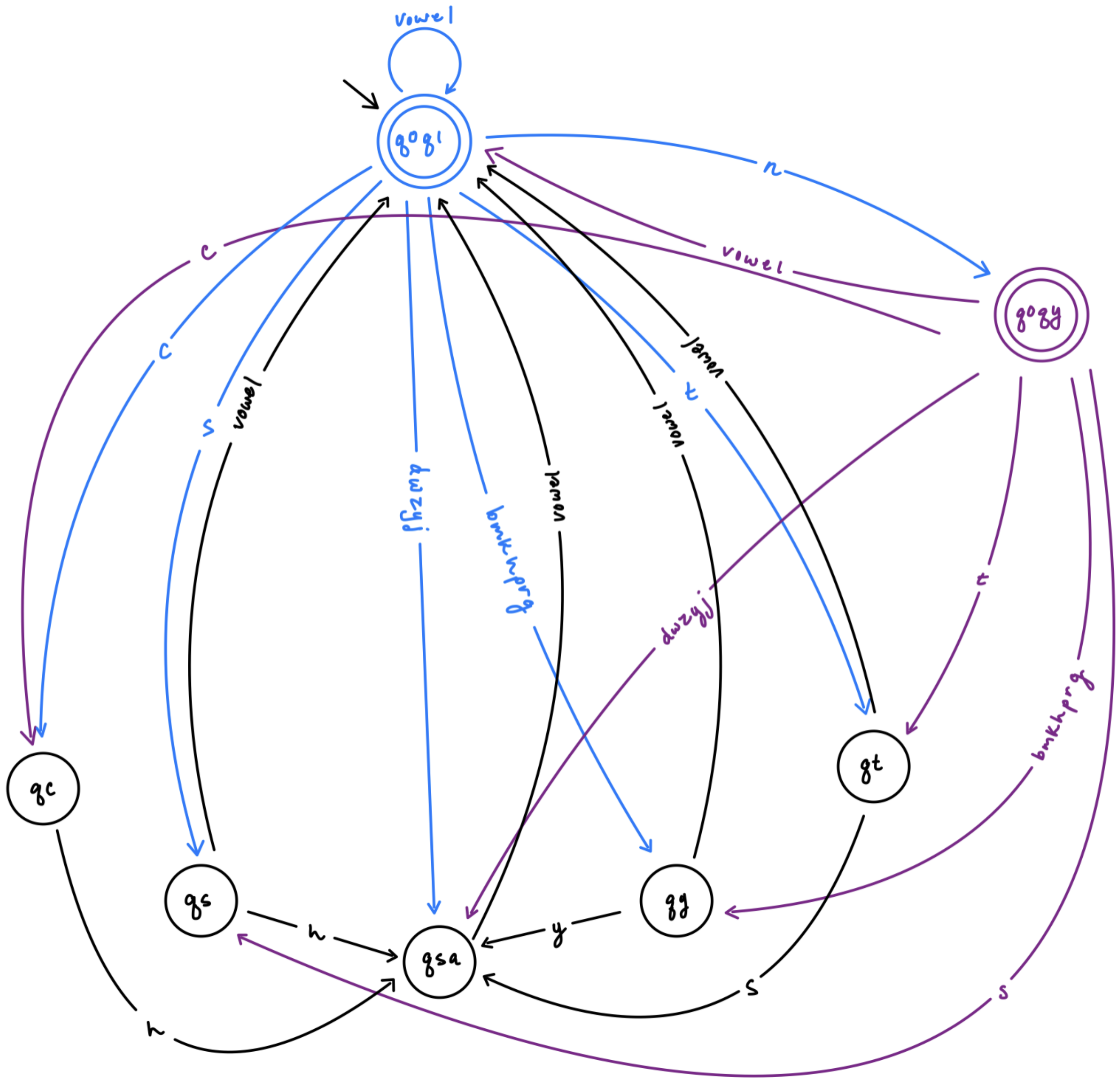
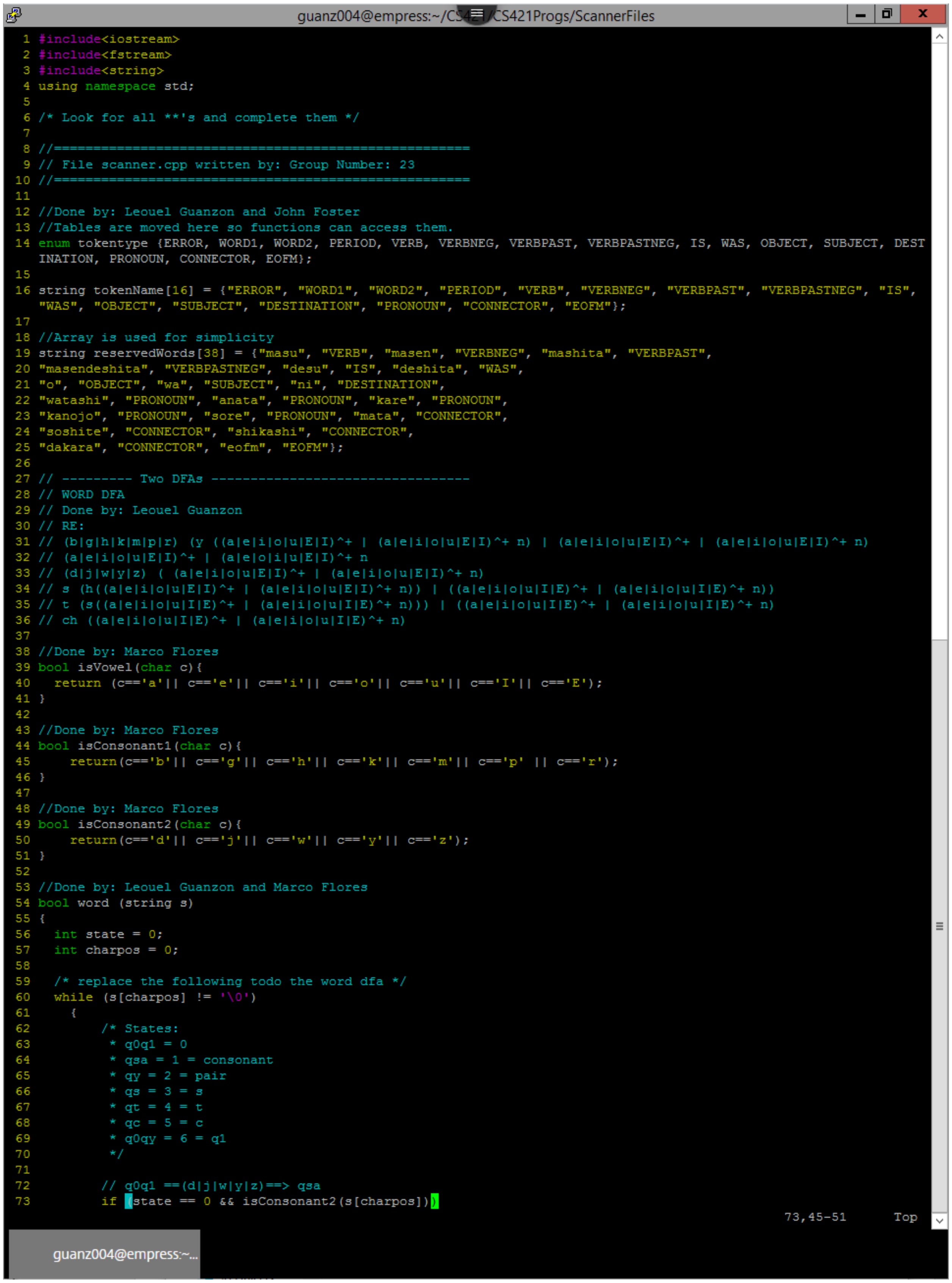
Japanese Scanner/Parser/Translator Project

Group 23
Leouel Guanzon
Marco Flores
John Foster

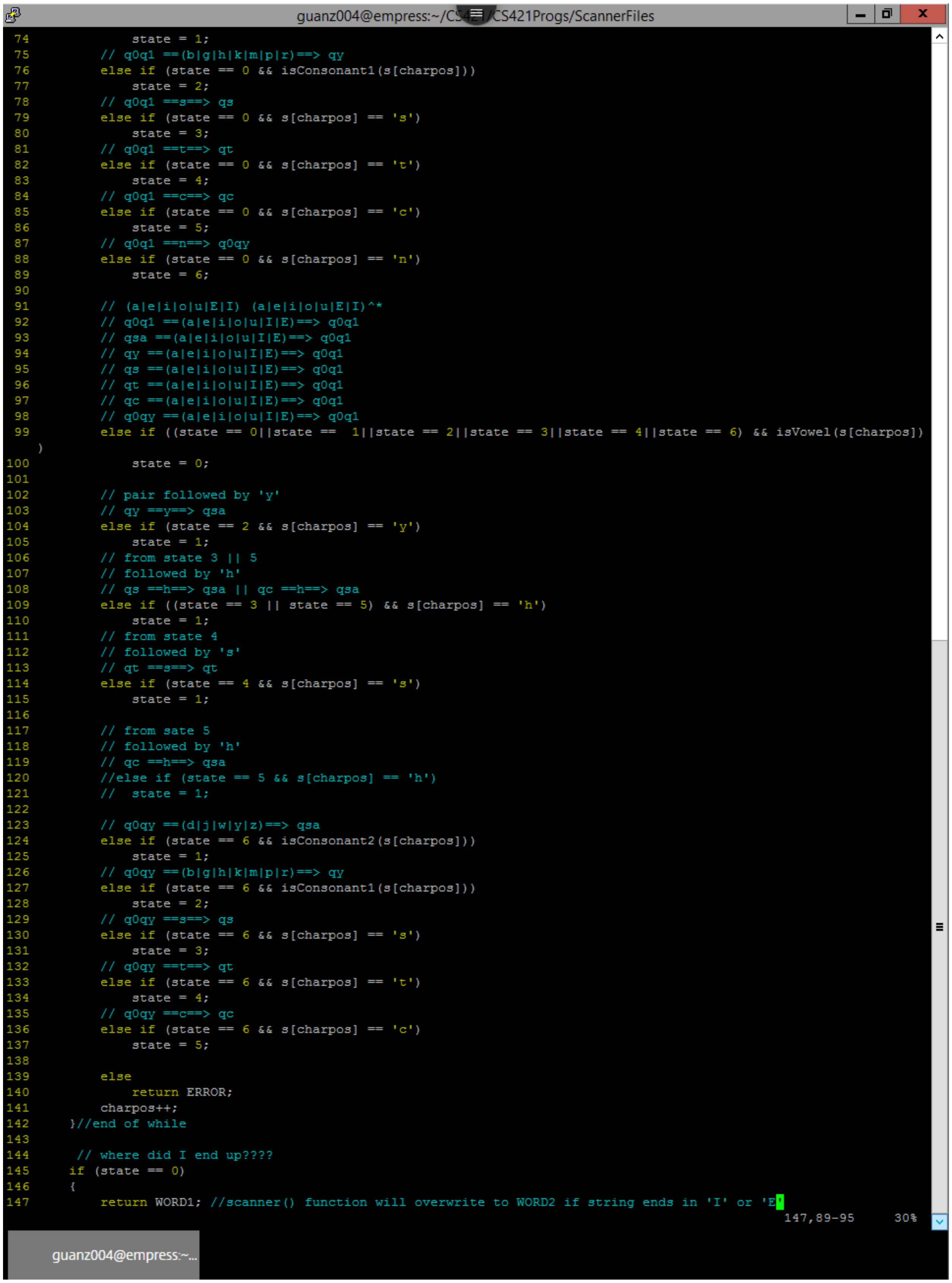
1 – DFA


vowel
q0q1
n
vowel
q0qy
c
c
s
vowel
dwzyj
bmkhprg
vowel
vowel
t
vowel
dwzyj
t
bmkhprg
qc
qs
qsa
qg
qt
h
y
s
h
s

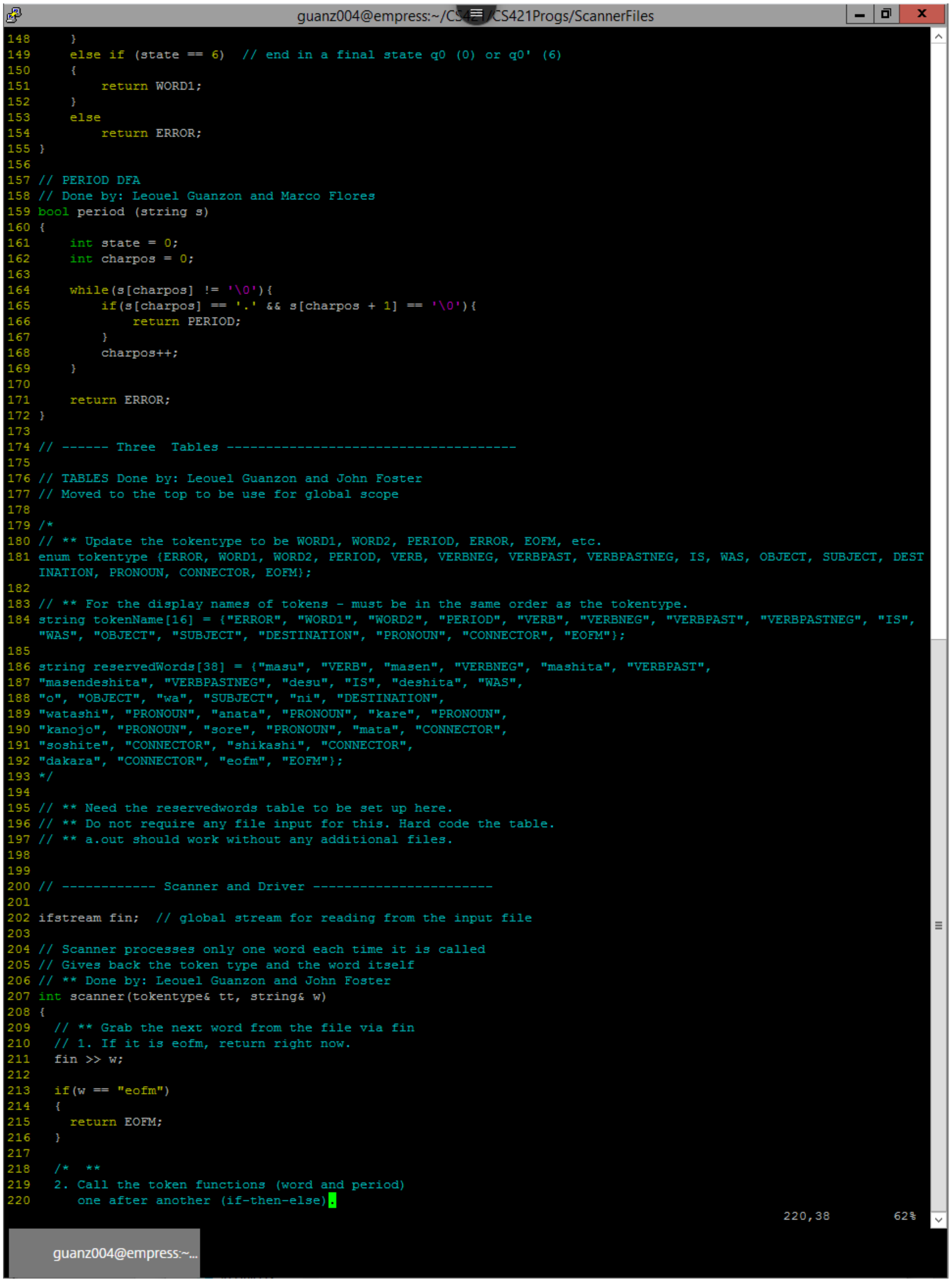
2 – Scanner.cpp



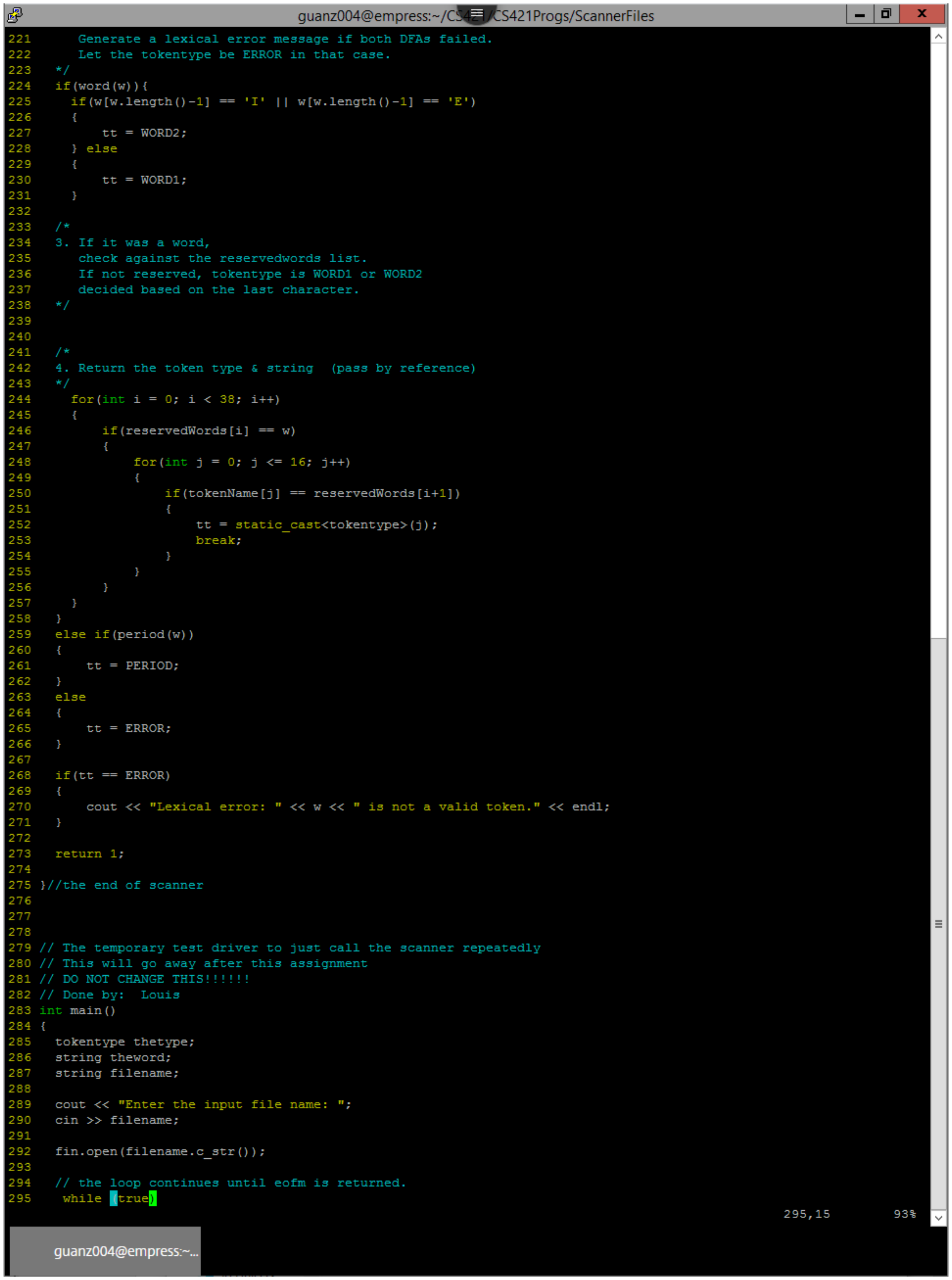
```cpp
#include<iostream>
#include<fstream>
#include<string>
using namespace std;

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

//=====================================================
// File scanner.cpp written by: Group Number: 23
//=====================================================

//Done by: Leouel Guanzon and John Foster
//Tables are moved here so functions can access them.
enum tokentype {ERROR, WORD1, WORD2, PERIOD, VERB, VERBNEG, VERBPAST, VERBPASTNEG, IS, WAS, OBJECT, SUBJECT, DESTINATION, PRONOUN, CONNECTOR, EOFM};

string tokenName[16] = {"ERROR", "WORD1", "WORD2", "PERIOD", "VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT", "DESTINATION", "PRONOUN", "CONNECTOR", "EOFM"};

//Array is used for simplicity
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"};

// --------- Two DFAs ---------------------------------
// WORD DFA
// Done by: Leouel Guanzon
// RE:
// (b|g|h|k|m|p|r) (y ((a|e|i|o|u|E|I)^+ | (a|e|i|o|u|E|I)^+ n) | (a|e|i|o|u|E|I)^+ | (a|e|i|o|u|E|I)^+ n)
// (a|e|i|o|u|E|I)^+ | (a|e|o|i|u|E|I)^+ n
// (d|j|w|y|z) ( (a|e|i|o|u|E|I)^+ | (a|e|i|o|u|E|I)^+ n)
// s (h((a|e|i|o|u|E|I)^+ | (a|e|i|o|u|E|I)^+ n)) | ((a|e|i|o|u|I|E)^+ | (a|e|i|o|u|I|E)^+ n))
// t (s((a|e|i|o|u|I|E)^+ | (a|e|i|o|u|I|E)^+ n))) | ((a|e|i|o|u|I|E)^+ | (a|e|i|o|u|I|E)^+ n)
// ch ((a|e|i|o|u|I|E)^+ | (a|e|i|o|u|I|E)^+ n)

//Done by: Marco Flores
bool isVowel(char c){
  return (c=='a'|| c=='e'|| c=='i'|| c=='o'|| c=='u'|| c=='I'|| c=='E');
}

//Done by: Marco Flores
bool isConsonant1(char c){
    return(c=='b'|| c=='g'|| c=='h'|| c=='k'|| c=='m'|| c=='p' || c=='r');
}

//Done by: Marco Flores
bool isConsonant2(char c){
    return(c=='d'|| c=='j'|| c=='w'|| c=='y'|| c=='z');
}

//Done by: Leouel Guanzon and Marco Flores
bool word (string s)
{
  int state = 0;
  int charpos = 0;

  /* replace the following todo the word dfa */
  while (s[charpos] != '\0')
    {
        /* States:
         * q0q1 = 0
         * qsa = 1 = consonant
         * qy = 2 = pair
         * qs = 3 = s
         * qt = 4 = t
         * qc = 5 = c
         * q0qy = 6 = q1
         */

        // q0q1 ==(d|j|w|y|z)==> qsa
        if (state == 0 && isConsonant2(s[charpos]))
```



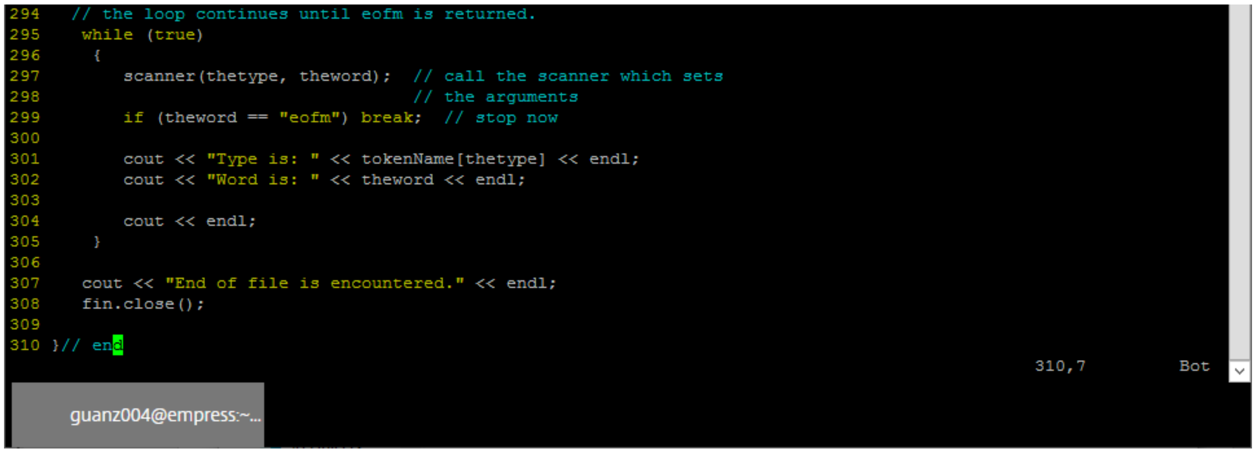
```
            state = 1;
        // q0q1 ==(b|g|h|k|m|p|r)==> qy
        else if (state == 0 && isConsonant1(s[charpos]))
            state = 2;
        // q0q1 ==s==> qs
        else if (state == 0 && s[charpos] == 's')
            state = 3;
        // q0q1 ==t==> qt
        else if (state == 0 && s[charpos] == 't')
            state = 4;
        // q0q1 ==c==> qc
        else if (state == 0 && s[charpos] == 'c')
            state = 5;
        // q0q1 ==n==> q0qy
        else if (state == 0 && s[charpos] == 'n')
            state = 6;

        // (a|e|i|o|u|E|I) (a|e|i|o|u|E|I)^*
        // q0q1 ==(a|e|i|o|u|I|E)==> q0q1
        // qsa ==(a|e|i|o|u|I|E)==> q0q1
        // qy ==(a|e|i|o|u|I|E)==> q0q1
        // qs ==(a|e|i|o|u|I|E)==> q0q1
        // qt ==(a|e|i|o|u|I|E)==> q0q1
        // qc ==(a|e|i|o|u|I|E)==> q0q1
        // q0qy ==(a|e|i|o|u|I|E)==> q0q1
        else if ((state == 0||state ==  1||state == 2||state == 3||state == 4||state == 6) && isVowel(s[charpos]))
            state = 0;

        // pair followed by 'y'
        // qy ==y==> qsa
        else if (state == 2 && s[charpos] == 'y')
            state = 1;
        // from state 3 || 5
        // followed by 'h'
        // qs ==h==> qsa || qc ==h==> qsa
        else if ((state == 3 || state == 5) && s[charpos] == 'h')
            state = 1;
        // from state 4
        // followed by 's'
        // qt ==s==> qt
        else if (state == 4 && s[charpos] == 's')
            state = 1;

        // from sate 5
        // followed by 'h'
        // qc ==h==> qsa
        //else if (state == 5 && s[charpos] == 'h')
        //  state = 1;

        // q0qy ==(d|j|w|y|z)==> qsa
        else if (state == 6 && isConsonant2(s[charpos]))
            state = 1;
        // q0qy ==(b|g|h|k|m|p|r)==> qy
        else if (state == 6 && isConsonant1(s[charpos]))
            state = 2;
        // q0qy ==s==> qs
        else if (state == 6 && s[charpos] == 's')
            state = 3;
        // q0qy ==t==> qt
        else if (state == 6 && s[charpos] == 't')
            state = 4;
        // q0qy ==c==> qc
        else if (state == 6 && s[charpos] == 'c')
            state = 5;

        else
            return ERROR;
        charpos++;
    }//end of while

     // where did I end up????
    if (state == 0)
    {
        return WORD1; //scanner() function will overwrite to WORD2 if string ends in 'I' or 'E'
                                                                                     147,89-95      30%
```



```
    }
    else if (state == 6)  // end in a final state q0 (0) or q0' (6)
    {
        return WORD1;
    }
    else
        return ERROR;
}

// PERIOD DFA
// Done by: Leouel Guanzon and Marco Flores
bool period (string s)
{
    int state = 0;
    int charpos = 0;

    while(s[charpos] != '\0'){
        if(s[charpos] == '.' && s[charpos + 1] == '\0'){
            return PERIOD;
        }
        charpos++;
    }

    return ERROR;
}

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

// TABLES Done by: Leouel Guanzon and John Foster
// Moved to the top to be use for global scope

/*
// ** Update the tokentype to be WORD1, WORD2, PERIOD, ERROR, EOFM, etc.
enum tokentype {ERROR, WORD1, WORD2, PERIOD, VERB, VERBNEG, VERBPAST, VERBPASTNEG, IS, WAS, OBJECT, SUBJECT, DESTINATION, PRONOUN, CONNECTOR, EOFM};

// ** For the display names of tokens - must be in the same order as the tokentype.
string tokenName[16] = {"ERROR", "WORD1", "WORD2", "PERIOD", "VERB", "VERBNEG", "VERBPAST", "VERBPASTNEG", "IS", "WAS", "OBJECT", "SUBJECT", "DESTINATION", "PRONOUN", "CONNECTOR", "EOFM"};

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"};
*/

// ** 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.


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

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

// Scanner processes only one word each time it is called
// Gives back the token type and the word itself
// ** Done by: Leouel Guanzon and John Foster
int scanner(tokentype& tt, string& w)
{
  // ** Grab the next word from the file via fin
  // 1. If it is eofm, return right now.
  fin >> w;

  if(w == "eofm")
  {
    return EOFM;
  }

  /*  **
  2. Call the token functions (word and period)
     one after another (if-then-else).
                                                    220,38         62%
```

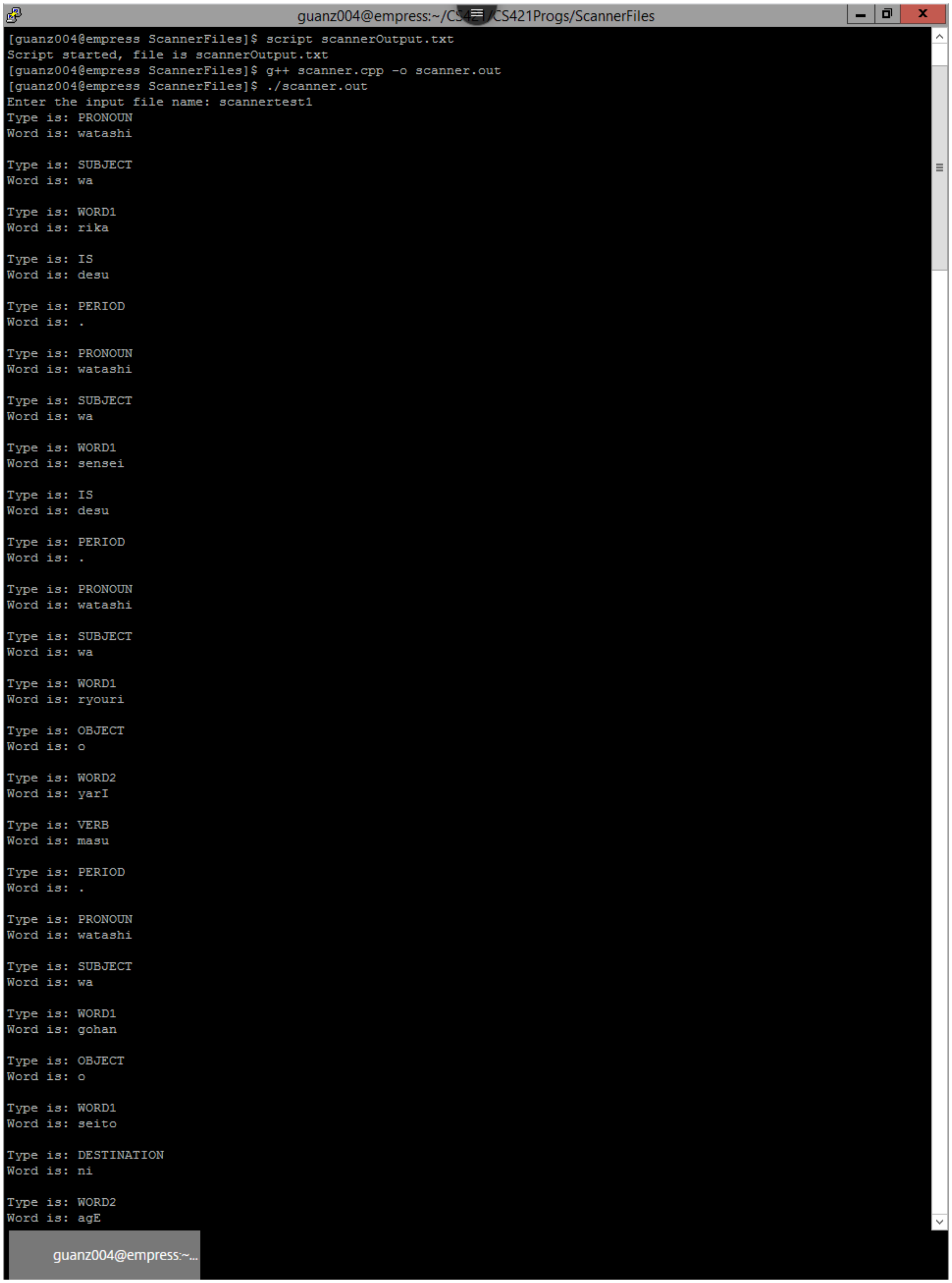


```
     Generate a lexical error message if both DFAs failed.
     Let the tokentype be ERROR in that case.
  */
  if(word(w)){
    if(w[w.length()-1] == 'I' || w[w.length()-1] == 'E')
    {
        tt = WORD2;
    } else
    {
        tt = WORD1;
    }

  /*
  3. If it was a word,
     check against the reservedwords list.
     If not reserved, tokentype is WORD1 or WORD2
     decided based on the last character.
  */


  /*
  4. Return the token type & string  (pass by reference)
  */
    for(int i = 0; i < 38; i++)
    {
        if(reservedWords[i] == w)
        {
            for(int j = 0; j <= 16; j++)
            {
                if(tokenName[j] == reservedWords[i+1])
                {
                    tt = static_cast<tokentype>(j);
                    break;
                }
            }
        }
    }
  }
  else if(period(w))
  {
      tt = PERIOD;
  }
  else
  {
      tt = ERROR;
  }

  if(tt == ERROR)
  {
      cout << "Lexical error: " << w << " is not a valid token." << endl;
  }

  return 1;

}//the end of scanner



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

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

  fin.open(filename.c_str());

  // the loop continues until eofm is returned.
   while (true)
                                                      295,15        93%
```

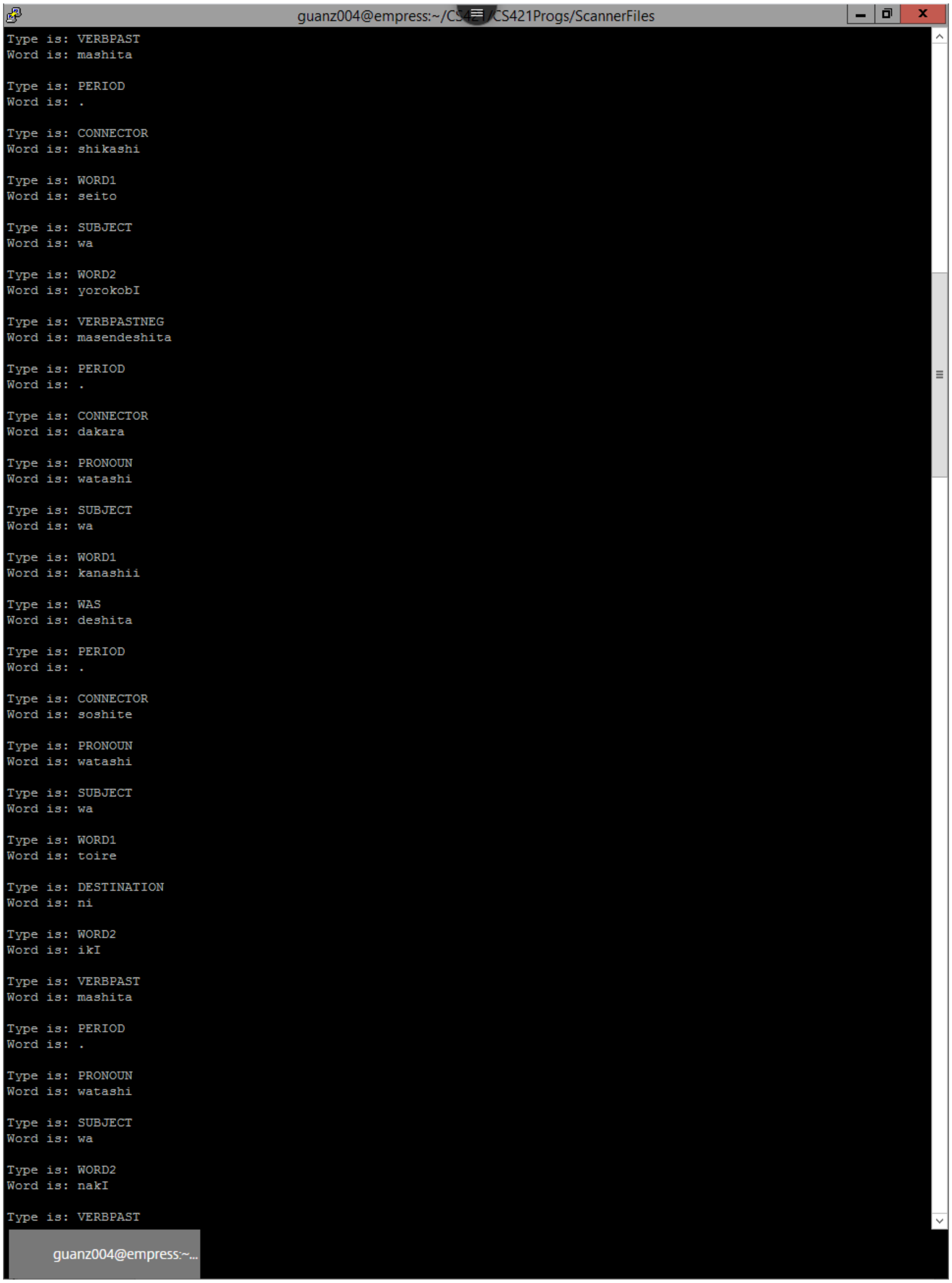


```
  // the loop continues until eofm is returned.
   while (true)
    {
       scanner(thetype, theword);  // call the scanner which sets
                                   // the arguments
       if (theword == "eofm") break;  // stop now

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

       cout << endl;
    }

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

}// end
                                                      310,7          Bot
```

3 – Original Scanner test results



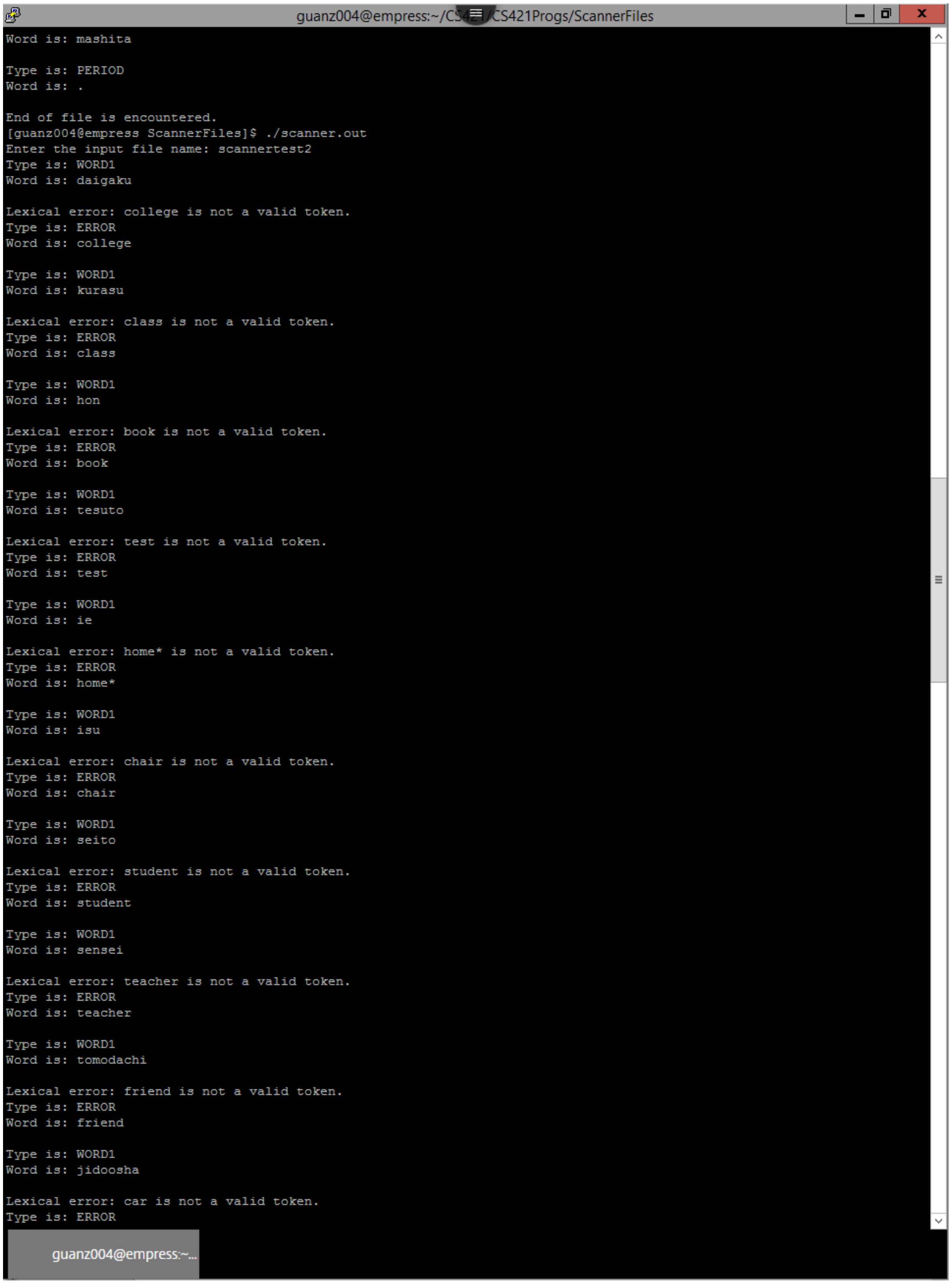
```
[guanz004@empress ScannerFiles]$ script scannerOutput.txt
Script started, file is scannerOutput.txt
[guanz004@empress ScannerFiles]$ g++ scanner.cpp -o scanner.out
[guanz004@empress ScannerFiles]$ ./scanner.out
Enter the input file name: scannertest1
Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: rika

Type is: IS
Word is: desu

Type is: PERIOD
Word is: .

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: sensei

Type is: IS
Word is: desu

Type is: PERIOD
Word is: .

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: ryouri

Type is: OBJECT
Word is: o

Type is: WORD2
Word is: yarI

Type is: VERB
Word is: masu

Type is: PERIOD
Word is: .

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: gohan

Type is: OBJECT
Word is: o

Type is: WORD1
Word is: seito

Type is: DESTINATION
Word is: ni

Type is: WORD2
Word is: agE
```



guanz004@empress:~/CS4 /CS421Progs/ScannerFiles

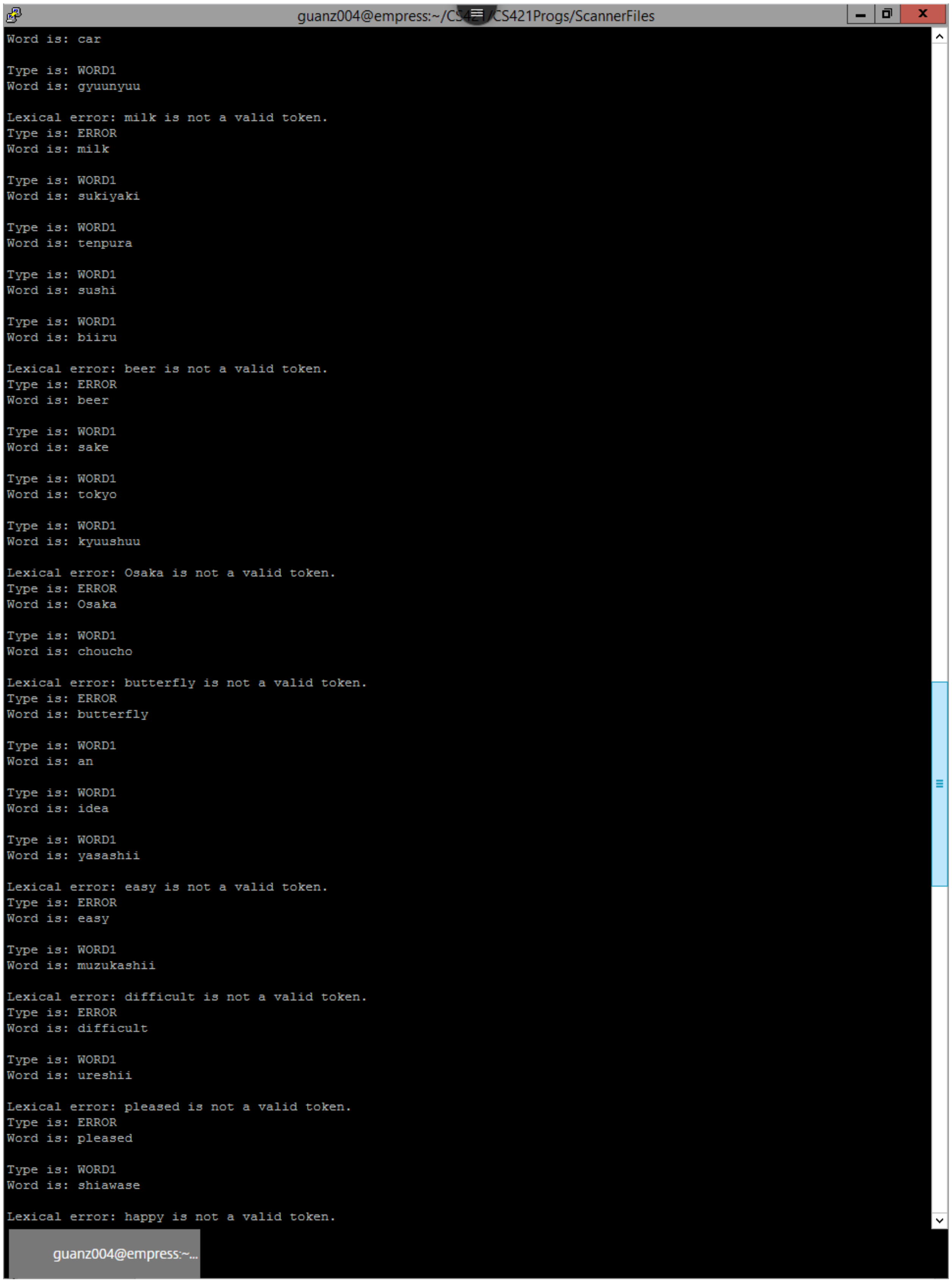
```
Type is: VERBPAST
Word is: mashita

Type is: PERIOD
Word is: .

Type is: CONNECTOR
Word is: shikashi

Type is: WORD1
Word is: seito

Type is: SUBJECT
Word is: wa

Type is: WORD2
Word is: yorokobI

Type is: VERBPASTNEG
Word is: masendeshita

Type is: PERIOD
Word is: .

Type is: CONNECTOR
Word is: dakara

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: kanashii

Type is: WAS
Word is: deshita

Type is: PERIOD
Word is: .

Type is: CONNECTOR
Word is: soshite

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD1
Word is: toire

Type is: DESTINATION
Word is: ni

Type is: WORD2
Word is: ikI

Type is: VERBPAST
Word is: mashita

Type is: PERIOD
Word is: .

Type is: PRONOUN
Word is: watashi

Type is: SUBJECT
Word is: wa

Type is: WORD2
Word is: nakI

Type is: VERBPAST
```

guanz004@empress:~...


guanz004@empress:~/CS421/CS421Progs/ScannerFiles

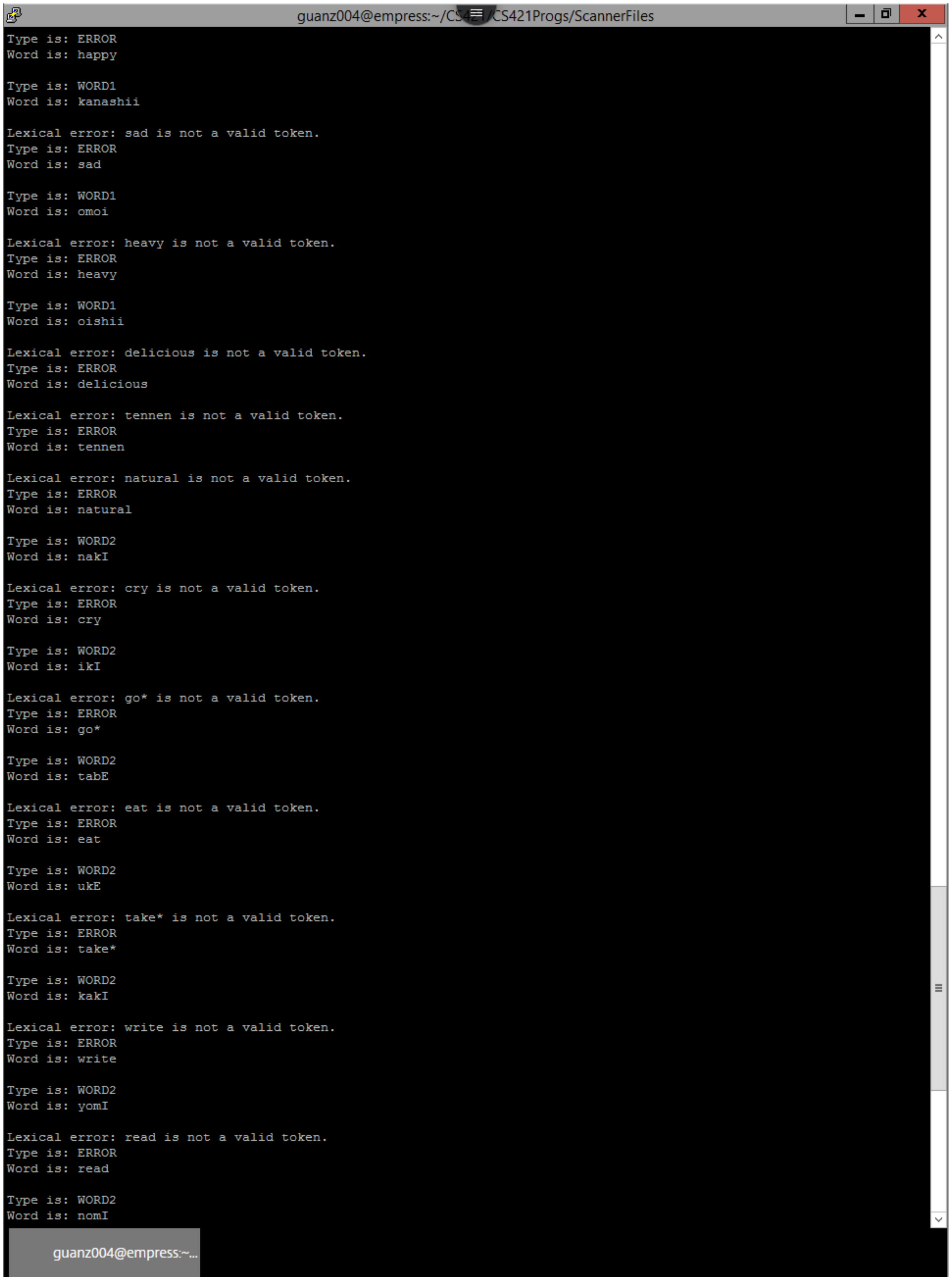
```
Word is: mashita

Type is: PERIOD
Word is: .

End of file is encountered.
[guanz004@empress ScannerFiles]$ ./scanner.out
Enter the input file name: scannertest2
Type is: WORD1
Word is: daigaku

Lexical error: college is not a valid token.
Type is: ERROR
Word is: college

Type is: WORD1
Word is: kurasu

Lexical error: class is not a valid token.
Type is: ERROR
Word is: class

Type is: WORD1
Word is: hon

Lexical error: book is not a valid token.
Type is: ERROR
Word is: book

Type is: WORD1
Word is: tesuto

Lexical error: test is not a valid token.
Type is: ERROR
Word is: test

Type is: WORD1
Word is: ie

Lexical error: home* is not a valid token.
Type is: ERROR
Word is: home*

Type is: WORD1
Word is: isu

Lexical error: chair is not a valid token.
Type is: ERROR
Word is: chair

Type is: WORD1
Word is: seito

Lexical error: student is not a valid token.
Type is: ERROR
Word is: student

Type is: WORD1
Word is: sensei

Lexical error: teacher is not a valid token.
Type is: ERROR
Word is: teacher

Type is: WORD1
Word is: tomodachi

Lexical error: friend is not a valid token.
Type is: ERROR
Word is: friend

Type is: WORD1
Word is: jidoosha

Lexical error: car is not a valid token.
Type is: ERROR
```

guanz004@empress:~...


guanz004@empress:~/CS4 /CS421Progs/ScannerFiles

```
Word is: car

Type is: WORD1
Word is: gyuunyuu

Lexical error: milk is not a valid token.
Type is: ERROR
Word is: milk

Type is: WORD1
Word is: sukiyaki

Type is: WORD1
Word is: tenpura

Type is: WORD1
Word is: sushi

Type is: WORD1
Word is: biiru

Lexical error: beer is not a valid token.
Type is: ERROR
Word is: beer

Type is: WORD1
Word is: sake

Type is: WORD1
Word is: tokyo

Type is: WORD1
Word is: kyuushuu

Lexical error: Osaka is not a valid token.
Type is: ERROR
Word is: Osaka

Type is: WORD1
Word is: choucho

Lexical error: butterfly is not a valid token.
Type is: ERROR
Word is: butterfly

Type is: WORD1
Word is: an

Type is: WORD1
Word is: idea

Type is: WORD1
Word is: yasashii

Lexical error: easy is not a valid token.
Type is: ERROR
Word is: easy

Type is: WORD1
Word is: muzukashii

Lexical error: difficult is not a valid token.
Type is: ERROR
Word is: difficult

Type is: WORD1
Word is: ureshii

Lexical error: pleased is not a valid token.
Type is: ERROR
Word is: pleased

Type is: WORD1
Word is: shiawase

Lexical error: happy is not a valid token.
```

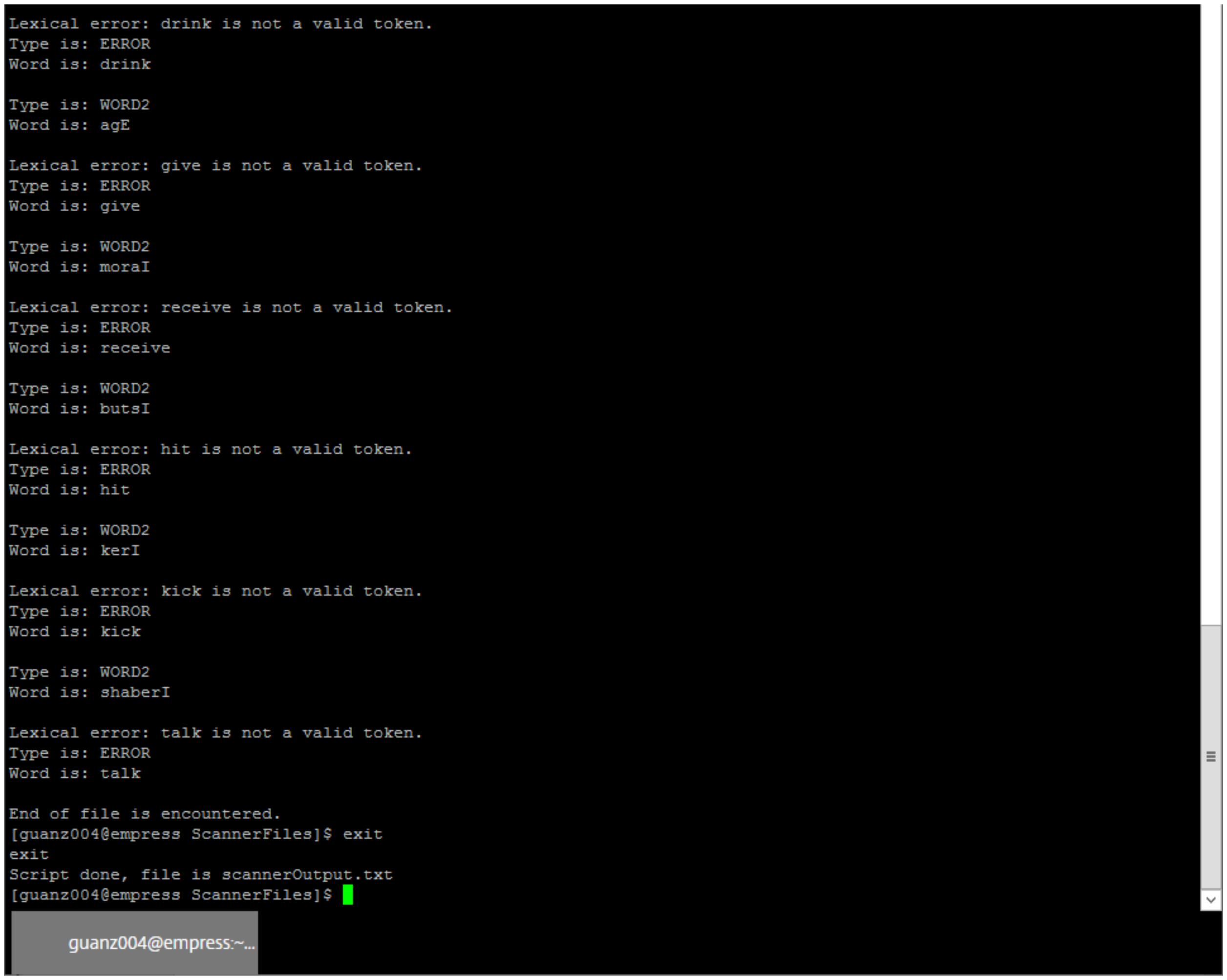
guanz004@empress:~...



guanz004@empress:~/CS421/CS421Progs/ScannerFiles

```
Type is: ERROR
Word is: happy

Type is: WORD1
Word is: kanashii

Lexical error: sad is not a valid token.
Type is: ERROR
Word is: sad

Type is: WORD1
Word is: omoi

Lexical error: heavy is not a valid token.
Type is: ERROR
Word is: heavy

Type is: WORD1
Word is: oishii

Lexical error: delicious is not a valid token.
Type is: ERROR
Word is: delicious

Lexical error: tennen is not a valid token.
Type is: ERROR
Word is: tennen

Lexical error: natural is not a valid token.
Type is: ERROR
Word is: natural

Type is: WORD2
Word is: nakI

Lexical error: cry is not a valid token.
Type is: ERROR
Word is: cry

Type is: WORD2
Word is: ikI

Lexical error: go* is not a valid token.
Type is: ERROR
Word is: go*

Type is: WORD2
Word is: tabE

Lexical error: eat is not a valid token.
Type is: ERROR
Word is: eat

Type is: WORD2
Word is: ukE

Lexical error: take* is not a valid token.
Type is: ERROR
Word is: take*

Type is: WORD2
Word is: kakI

Lexical error: write is not a valid token.
Type is: ERROR
Word is: write

Type is: WORD2
Word is: yomI

Lexical error: read is not a valid token.
Type is: ERROR
Word is: read

Type is: WORD2
Word is: nomI
```

guanz004@empress:~...



```
Lexical error: drink is not a valid token.
Type is: ERROR
Word is: drink

Type is: WORD2
Word is: agE

Lexical error: give is not a valid token.
Type is: ERROR
Word is: give

Type is: WORD2
Word is: moraI

Lexical error: receive is not a valid token.
Type is: ERROR
Word is: receive

Type is: WORD2
Word is: butsI

Lexical error: hit is not a valid token.
Type is: ERROR
Word is: hit

Type is: WORD2
Word is: kerI

Lexical error: kick is not a valid token.
Type is: ERROR
Word is: kick

Type is: WORD2
Word is: shaberI

Lexical error: talk is not a valid token.
Type is: ERROR
Word is: talk

End of file is encountered.
[guanz004@empress ScannerFiles]$ exit
exit
Script done, file is scannerOutput.txt
[guanz004@empress ScannerFiles]$
```

4 – Factored Rules

```
1 <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)

2 <s> ::=  [CONNECTOR] <noun> SUBJECT  <verb> <tense> PERIOD
3 <s> ::=  [CONNECTOR] <noun> SUBJECT  <noun> <be>    PERIOD
4 <s> ::=  [CONNECTOR] <noun> SUBJECT  <noun> DESTINATION  <verb> <tense> PERIOD
5 <s> ::=  [CONNECTOR] <noun> SUBJECT  <noun> OBJECT <verb> <tense> PERIOD
6 <s> ::=  [CONNECTOR] <noun> SUBJECT  <noun> OBJECT <noun> DESTINATION <verb> <tense> PERIOD
```

// Refer to Left Factoring file to make these into
// one rule until things start to differ.

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

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

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

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

7 <noun> ::= WORD1 | PRONOUN

8 <verb> ::= WORD2

9 <be> ::=   IS | WAS

10 <tense> := VERBPAST  | VERBPASTNEG | VERB | VERBNEG
```

5 – Updated Parser code for Translation