CS421 Final Project: Compiler
Group 6

Chris Childers
Ly Dung
Lenson Paulo Laca

Section 0: State of the Program

The program is working perfectly…….

Extra Credit Features:

- Enabling/Disabling Trace Messages

  How it works: When program is ran the user is given an option for turning on or off the trace message.

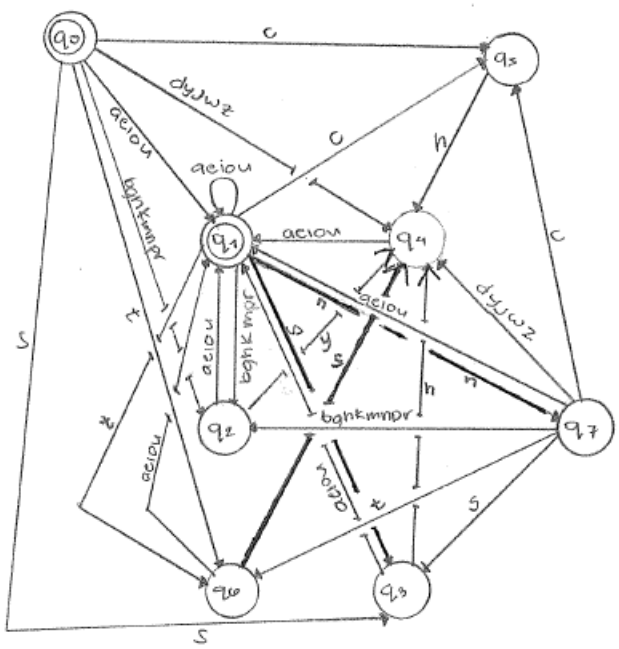
Saving Error messages into a text file

How it works: This feature is integrated into the program when an error occurs it is simultaneously displayed in the screen and written into a text file “error.txt”.

Section 1: DFA

Group 6

Christopher Roberts Childers
Lenson Paulo Laca
Dung Tri Ly



Section 2: Scanner code that match your DFAs

```
#include<iostream>
#include <stdio.h>
#include<fstream>
#include<string>
#include<cstdlib>
#include <ctype.h>

using namespace std;
//Token type for the word in the language
enum tokentype {CONNECTOR, DESTINATION, ERROR, EOFM, IS, OBJECT, PERIOD, PRONOUN,
SUBJECT, WAS, WORD1, WORD2, VERB, VERBNEG, VERBPAST, VERBPASTNEG};
```

```
int scanner(tokentype & , string & );// Looking for the token type for the word in the
language
bool mytoken(string );// Does the word belong to the language?
bool myPeriod(string );// Does the  word is period?
string convertE(string);// Convert japaneses to english
//Purpose: convert japaneses to english
//Algorithm: Looking for correspond english word
// Parameter:String
//Return: string
//Author: Dung Ly
string convertE(string a){
  if(a=="watashi"){
    a="I/me";
  }
  else if(a=="anata"){
    a="you";
  }
  else if(a=="kanojo"){
    a="she/her";
  }
  else if(a=="sore"){
    a="mata";
  }
  else if(a=="mata"){
    a="Also";
  }
  else if(a=="soshite"){
    a="Then";
  }
  else if(a=="shikashi"){
    a="However";
  }
  else if(a=="dakara"){
    a="Therefore";
  }
  return a;
}
//Purpose: Convert tokentype to string
//Algorithm: set the similar string as token type
//Parameter: token type
//Return: string
//Author: Dung Ly
string convertT(tokentype t){
  string a;// Return string for token type
```

```
switch(t){
case CONNECTOR:
  a="CONNECTOR";
  break;
case DESTINATION:
  a="DESTINATION";
  break;
case ERROR:
  a="ERROR";
  break;
case EOFM:
  a="EOFM";
  break;
case IS:
  a="IS";
  break;
case OBJECT:
  a="OBJECT";
  break;
case PERIOD:
  a="PERIOD";
  break;
case PRONOUN:
  a="PRONOUN";
  break;
case SUBJECT:
  a="SUBJECT";
  break;
case WAS:
  a="WAS";
  break;
case WORD1:
  a="WORD1";
  break;
case WORD2:
  a="WORD2";
  break;
case VERB:
  a="VERB";
  break;
case VERBNEG:
  a="VERBNEG";
  break;
case VERBPAST:
```

```
   a="VERBPAST";
   break;
  case VERBPASTNEG:
   a="VERBPASTNEG";
   break;
  default:
   a="Nothing";
     break;
  }
  return a;
}
//Purpose: Looking for token type for a input word
//Algorithm: Calling mytoken function and period function to determined the tokentype of
the word
//Parameter: tokentype , string
//Return: int
//Author:  Dung Ly
int scanner(tokentype & a, string & w)
{
          bool test1=mytoken(w);// Calling token type function
          // the the word is period => return token type
          if(myPeriod(w)==true){
            a=PERIOD;
            return 1;
          }
          // the word belongs to the language=> find it token type
     else if(test1==true)
           {
            // Checking for word 1 and word 2 1st
             if(w[w.length()-1]=='I'||w[w.length()-1]=='E')
              {
                     a=WORD2;
              }
             else
              {
                     a=WORD1;
              }
                    //check if the word  belongs to a reservered word list  or not. Override the
result of a.
                    if (w=="masu")
                            a= VERB;

                    if(w=="masen")
                            a=VERBNEG;
```

```
 if(w=="mashita")
          a=VERBPAST;

 if(w=="masendeshita")
   a= VERBPASTNEG;

 if(w=="desu")
          a= IS;

if(w== "deshita")
  a= WAS;

 if(w== "o")
          a= OBJECT;

 if(w== "wa")
          a=SUBJECT;

 if(w=="ni")
          a=DESTINATION;

 if(w=="watashi")
          a=PRONOUN;

 if(w=="anata")
          a=PRONOUN;

 if(w=="kare")
          a=PRONOUN;

 if(w=="kanojo")
          a=PRONOUN;

  if(w=="sore")
          a=PRONOUN;

if(w== "mata")
          a=CONNECTOR;

 if(w== "soshite")
          a=CONNECTOR;

if(w== "shikashi")
```

```
                        a= CONNECTOR;

                if(w=="dakara")
                        a=CONNECTOR;

                if(w== "eofm")
                 a=EOFM;
            //================================
                return 1;
            }
           // The word is not in the langaue => token type is error
          else if (test1==false)
     {
            a=ERROR;
            cout<<"Lexical Error: "<< w <<" is not valid token"<<endl;
            return 1;

   }
          return 0;
}//the end
//Purpose: Check if the word belogn to the language or not
//Algorithm: Using the language DPA
//Parameter: string
//Return: bool
//Author: Dung Ly
bool mytoken(string s){
      int flag=0;// flag to stop checking if the state is found
      int state = 0;// state the current state
      int charpos = 0;// character position
   // if the word is eofm => end of find return true
          if(s=="eofm")
            return true;
  // Going in to the loop
           while (s[charpos] != '\0')
   {
      if(s[s.length()-1]=='I'||s[s.length()-1]=='E')
                s[s.length()-1]=tolower(s[s.length()-1]);

            bool
list1=s[charpos]=='a'||s[charpos]=='e'||s[charpos]=='i'||s[charpos]=='o'||s[charpos]=='u';
                bool
list2=s[charpos]=='b'||s[charpos]=='g'||s[charpos]=='h'||s[charpos]=='k'||s[charpos]=='m'||
s[charpos]=='n'||s[charpos]=='p'||s[charpos]=='r';
```

```
                bool
list3=s[charpos]=='b'||s[charpos]=='g'||s[charpos]=='h'||s[charpos]=='k'||s[charpos]=='m'||
s[charpos]=='p'||s[charpos]=='r';
                bool
list4=s[charpos]=='d'||s[charpos]=='y'||s[charpos]=='j'||s[charpos]=='w'||s[charpos]=='z';
            bool
list5=s[charpos]=='b'||s[charpos]=='g'||s[charpos]=='h'||s[charpos]=='k'||s[charpos]=='m'||
s[charpos]=='p'||s[charpos]=='r'||s[charpos]=='d'||s[charpos]=='y'|s[charpos]=='z'||s[charp
os]=='w';
                //==========From state 0===============
                if(flag==0&&state==0&&s[charpos]=='c')
                {
                        state=5; // new state
                        flag=1;//  stop go to any other if's
                }

                 else if(flag==0&&state==0&&s[charpos]=='s')
                {
                        state=3;
                        flag=1;

                }

                else if(flag==0&&state==0&&s[charpos]=='t')
                {
                        state=6;
                        flag=1;
                }
                else if(flag==0&&state==0&&list1)
                {
                        state=1;
                        flag=1;
                }

                else if(flag==0&&state==0&&list2)
                {
                        state=2;
                        flag=1;
        }
                else if(flag==0&&state==0&&list4)
                {
                        state=4;
                        flag=1;
                }
```

```
    //============From state 1============

    if(flag==0&&state==1&&s[charpos]=='n')
    {
            state=1;
            flag=1;
    }

    else if(flag==0&&state==1&&s[charpos]=='s')
    {
            state=3;
            flag=1;
    }
    else if(flag==0&&state==1&&s[charpos]=='c')
    {
            state=5;
            flag=1;
    }
    else if(flag==0&&state==1&&s[charpos]=='t')
    {
            state=6;
            flag=1;
    }

    else if(flag==0&&state==1&& list5 )
    {
            state=2;
            flag=1;
    }
    else if(flag==0&&state==1&& list1)
     {
       state=1;
       flag=1;
     }

    //==========From State 2============

if(flag==0&&state==2&& list1)
    {
            state=1;
            flag=1;
    }
    else if(flag==0&&state==2&&s[charpos]=='y')
    {
```

```
            state=4;
            flag=1;
    }
    //=========From State 3===========
 if(flag==0&&state==3&& list1)
    {
            state=1;
            flag=1;
    }
    else if(flag==0&&state==3&&s[charpos]=='h')
    {
            state=4;
            flag=1;
    }
    //=========From State 4===========
if(flag==0&&state==4&& list1)
    {
            state=1;
            flag=1;
    }
    //=========From State 5===========
     if(flag==0&&state==5&& s[charpos]=='h')
    {
            state=4;
            flag=1;
    }
    //=========From State 6===========
     if(flag==0&&state==6&& list1)
    {
            state=1;
            flag=1;
    }
    else if(flag==0&&state==6&&s[charpos]=='s')
    {
            state=4;
            flag=1;
    }
    //=========From State 7===========
if(flag==0&&state==7&& s[charpos]=='s')
    {
            state=3;
            flag=1;
    }
    else if(flag==0&&state==7&& s[charpos]=='t')
```

```
                {
                        state=6;
                        flag=1;
                }
                else if(flag==0&&state==7&& s[charpos]=='c')
                {
                        state=5;
                        flag=1;
                }
                else if(flag==0&&state==7&& list2)
                {
                        state=2;
                        flag=1;
                }
                else if(flag==0&&state==7&& list1)
                {
                        state=1;
                        flag=1;
                }
                else if(flag==0&&state==7&& list4)
                {
                        state=4;
                        flag=1;
                }
                if(flag==0)
                {
                        return false;
                }
                flag=0;// reset the flag
                charpos++;//move to new character in the word
   }//end of while
  if (state == 0||state==1) return(true);  // end in a final state
   else return(false);
}
//Purpose:
//Algorithm:
//Parameter:
//Return:
//Author:
bool myPeriod(string s){
        int state = 0;
        int charpos = 0;
        while (s[charpos] != '\0') {
                if(state==0 && s[charpos]=='.')
```

```
                        state=1;
                else if (state==1)
                        return false;
                charpos++;
        }
        if(state ==1) return (true);
        else return (false);
}
```

Section 3: Scanner test results

Test1:
Enter Input file: test.txt
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :rika
The type: ISThe word :desu
The type: PERIODThe word :.
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :sensei
The type: ISThe word :desu
The type: PERIODThe word :.
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :ryouri
The type: OBJECTThe word :o
The type: WORD2The word :yarI
The type: VERBThe word :masu
The type: PERIODThe word :.
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :gohan
The type: OBJECTThe word :o
The type: WORD1The word :seito
The type: DESTINATIONThe word :ni
The type: WORD2The word :agE
The type: VERBPASTThe word :mashita
The type: PERIODThe word :.
The type: CONNECTORThe word :shikashi
The type: WORD1The word :seito
The type: SUBJECTThe word :wa
The type: WORD2The word :yorokobI
The type: VERBPASTNEGThe word :masendeshita
The type: PERIODThe word :.
The type: CONNECTORThe word :dakara
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :kanashii
The type: WASThe word :deshita
The type: PERIODThe word :.
The type: CONNECTORThe word :soshite
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD1The word :toire

The type: DESTINATIONThe word :ni
The type: WORD2The word :ikI
The type: VERBPASTThe word :mashita
The type: PERIODThe word :.
The type: PRONOUNThe word :watashi
The type: SUBJECTThe word :wa
The type: WORD2The word :nakI
The type: VERBPASTThe word :mashita
The type: PERIODThe word :.
The type: EOFMThe word :eofm

Test2:
Enter Input file: test.txt
The type: WORD1The word :daigaku
==========Lexical Error: college is not valid token==========
The type: ERRORThe word :college
The type: WORD1The word :kurasu
==========Lexical Error: class is not valid token==========
The type: ERRORThe word :class
The type: WORD1The word :hon
==========Lexical Error: book is not valid token==========
The type: ERRORThe word :book
The type: WORD1The word :tesuto
==========Lexical Error: test is not valid token==========
The type: ERRORThe word :test
The type: WORD1The word :ie
==========Lexical Error: home* is not valid token==========
The type: ERRORThe word :home*
The type: WORD1The word :isu
==========Lexical Error: chair is not valid token==========
The type: ERRORThe word :chair
The type: WORD1The word :seito
==========Lexical Error: student is not valid token==========
The type: ERRORThe word :student
The type: WORD1The word :sensei
==========Lexical Error: teacher is not valid token==========
The type: ERRORThe word :teacher
The type: WORD1The word :tomodachi
==========Lexical Error: friend is not valid token==========
The type: ERRORThe word :friend
The type: WORD1The word :jidoosha
==========Lexical Error: car is not valid token==========
The type: ERRORThe word :car

The type: WORD1The word :gyuunyuu
==========Lexical Error: milk is not valid token==========
The type: ERRORThe word :milk
The type: WORD1The word :sukiyaki
The type: WORD1The word :tenpura
The type: WORD1The word :sushi
The type: WORD1The word :biiru
==========Lexical Error: beer is not valid token==========
The type: ERRORThe word :beer
The type: WORD1The word :sake
The type: WORD1The word :tokyo
The type: WORD1The word :kyuushuu
==========Lexical Error: Osaka is not valid token==========
The type: ERRORThe word :Osaka
The type: WORD1The word :choucho
==========Lexical Error: butterfly is not valid token==========
The type: ERRORThe word :butterfly
The type: WORD1The word :an
The type: WORD1The word :idea
The type: WORD1The word :yasashii
==========Lexical Error: easy is not valid token==========
The type: ERRORThe word :easy
The type: WORD1The word :muzukashii
==========Lexical Error: difficult is not valid token==========
The type: ERRORThe word :difficult
The type: WORD1The word :ureshii
==========Lexical Error: pleased is not valid token==========
The type: ERRORThe word :pleased
The type: WORD1The word :shiawase
==========Lexical Error: happy is not valid token==========
The type: ERRORThe word :happy
The type: WORD1The word :kanashii
==========Lexical Error: sad is not valid token==========
The type: ERRORThe word :sad
The type: WORD1The word :omoi
==========Lexical Error: heavy is not valid token==========
The type: ERRORThe word :heavy
The type: WORD1The word :oishii
==========Lexical Error: delicious is not valid token==========
The type: ERRORThe word :delicious
The type: WORD1The word :tennen
==========Lexical Error: natural is not valid token==========
The type: ERRORThe word :natural
The type: WORD2The word :nakI

==========Lexical Error: cry is not valid token==========
The type: ERRORThe word :cry
The type: WORD2The word :ikI
==========Lexical Error: go* is not valid token==========
The type: ERRORThe word :go*
The type: WORD2The word :tabE
==========Lexical Error: eat is not valid token==========
The type: ERRORThe word :eat
The type: WORD2The word :ukE
==========Lexical Error: take* is not valid token==========
The type: ERRORThe word :take*
The type: WORD2The word :kakI
==========Lexical Error: write is not valid token==========
The type: ERRORThe word :write
The type: WORD2The word :yomI
==========Lexical Error: read is not valid token==========
The type: ERRORThe word :read
The type: WORD2The word :nomI
==========Lexical Error: drink is not valid token==========
The type: ERRORThe word :drink
The type: WORD2The word :agE
==========Lexical Error: give is not valid token==========
The type: ERRORThe word :give
The type: WORD2The word :moraI
==========Lexical Error: receive is not valid token==========
The type: ERRORThe word :receive
The type: WORD2The word :butsI
==========Lexical Error: hit is not valid token==========
The type: ERRORThe word :hit
The type: WORD2The word :kerI
==========Lexical Error: kick is not valid token==========
The type: ERRORThe word :kick
The type: WORD2The word :shaberI
==========Lexical Error: talk is not valid token==========
The type: ERRORThe word :talk
The type: EOFMThe word :eofm

Section 4: Factored rules with new non-terminal names

GROUP 6

<story> ::= <s1> {<s1>}

<s1> ::= [CONNECTOR] <noun> SUBJECT <s2>

<s2> ::= <verb> <tense> PERIOD | <noun> <s3>

<s3> ::= <be> PERIOD | DESTINATION <verb> <tense> PERIOD |
OBJECT [<noun> DESTINATION] <verb>
<tense> PERIOD

<noun> ::= WORD1 | PRONOUN

<verb> ::= WORD2

<be> ::= IS | WAS

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

Section 5: Parser code

```
#include<cstdlib>
#include<fstream>
#include<iostream>
#include "ScannerT.cpp"
#include<string>
#include<vector>

using namespace std;

//==================================================
// File parser.cpp written by Group Number: 6
//==================================================

// ** Be sure to put the name of the programmer above each function
// i.e. Done by:

//Part B: Other declarations
tokentype currentToken;  //token in queue
tokentype old;

string inputFile;
string currentWord; //current word read from the file

bool isTokenComparatorFree = false;
bool traceMSG; // turn on or off traceMSGs

ifstream fin;
ofstream fout; //output error messages to a file (error.txt)


//Part B: Functions declarations
tokentype next_token();
bool match(tokentype );
void syntaxerror1(tokentype , tokentype );
void syntaxerror2(tokentype , string );
void story();
void begin_story();
void after_subject();
void be();
void noun();
void tense();
void verb();
void after_noun();
/*
 * main Function
 * Written by: Lenson Paulo Laca, Ly Dung
 * Debug & Fix: Ly Dung
 */
int main()
{
  char traceOpt; //trace input
  bool test =false;
  cout << "Parser Funciton" << endl;
  cout << "Enter Filename of the File you want to parse: " << endl;
  getline(cin,inputFile);

  while(test!=true){
    cout << "KEEP TRACK OF PARSING PROCESS? Y / N: ";
    cin >> traceOpt;

    if(traceOpt == 'Y'|| traceOpt=='y')
      {
        traceMSG = true;
        test=true;
      }
    else if(traceOpt == 'N' || traceOpt=='n')
      {
        traceMSG = false;
        test=true;
      }
  }
```

```
  fin.open(inputFile.c_str());
  fout.open("Errors.txt", ios::out | ios::app);
  trans.open("Translated.txt", ios::out | ios::app);

  story(); //initiates parsing
  cout<<"=====END-Parsering=====<<endl;

  fin.close();
  fout.close();
  trans.close();

  return 0;
}
// Need syntaxerror1 and syntaxerror2 functions (each takes 2 args)
/*
 * syntaxerror1 Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void syntaxerror1(tokentype t, string s)
{
        cout << "SYNTAX ERROR: EXPECTING " << convertT(t) << " BUT FOUND " << s << endl;

        //Keep track of the file opened by writing it to the error file.
        fout << "Input File" << inputFile << ":" <<endl;
        fout << "SYNTAX ERROR: EXPECTING " << convertT(t) << " BUT FOUND " << s << endl;

        exit(-1);
}

/*
 * syntaxerror2 Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void syntaxerror2(string  w, string s)
{
        cout << "SYNTAX ERROR: UNEXPECTED WORD: " << w << " FOUND IN " << s << endl;

        //Keep track of the file opened by writing it to the error file.
        fout << "Filename: " << inputFile << ":" <<endl;
        fout << "SYNTAX ERROR: UNEXPECTED WORD: " << w << " FOUND IN " << s << endl;

        exit(-1);
}
/*
 * match Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
bool match(tokentype t)
{
        if (next_token() != t)
                syntaxerror1(t, currentWord);
        else
                isTokenComparatorFree = false;
        cout << "Matched " << convertT(t) << endl;
                return true;
}
```

```
// ** Need the updated match and next_token (with 2 global vars)
/*
 *next_token Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
tokentype next_token()
{
        string w;

        if (!isTokenComparatorFree)
    {
                fin >> w;
                currentWord = w;
                cout<<"The current word is " << w<<endl;
                scanner(currentToken, w);
                isTokenComparatorFree = true;

                if(traceMSG)
                {
                        cout << "Scanner is being called..." << endl;
                }

                if(currentToken == ERROR)
                {
                        fout << "Filename: " << inputFile << ":" <<endl;
                        fout << "The Word: " << w << " is not found in the language" << endl;
                }
    }
        return currentToken;
}

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

//STORY FUNCTION
//Part B GRAMMAR: <story>::= <begin_story> {<begin_story>}
/*
 * story Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void story()
{
        if(!traceMSG)
        {
                cout << "===============PROCESSING STORY=====================" << endl;
        }

        begin_story();

                while(next_token() != EOFM) //while not reaching end of fille
                {
                switch(next_token())//look ahead for the next tokentype
                        {
                                case CONNECTOR:
                                        begin_story();
                                        break;
                                default:
                                  begin_story();
                                break;
                        }
                }

}
```

```
//BEGIN_STORY FUNCTION
//Part B GRAMMAR: <begin_story>::=[CONNECTOR]<noun>SUBJECT<after_subject>
//Part C GRAMMAR:
<begin_story>::=[[CONNECTOR#getEword##gen#]<noun>#getEword#SUBJECT#gen#<afterSUBJECT>
/*
 * begin_story Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void begin_story()
{
	if(!traceMSG)
	{
		cout << "=====================PROCESSING BEGINNING STORY====================" <<
endl;
	}

	if(next_token() == CONNECTOR)
	{
		match(CONNECTOR);
		getEword();
		gen("CONNECTOR",'A');
	}

	noun(); // process noun
	getEword();
	if(next_token() == SUBJECT) // for syntaxerror1 checking
	{
		match(SUBJECT);
		gen("ACTOR", 'A');
	}
	else
	{
		syntaxerror1(SUBJECT, currentWord);
	}
	after_subject();
}
```

```
//AFTER_SUBJECT FUNCTION
//Part B GRAMMAR: <after_subject>::=<verb><tense> PERIOD | <noun><after_noun>
//Part C GRAMMAR: <after_subject>::=<verb>#getEword##gen#<tense>#gen#PERIOD |
<noun>#getEword#<after_noun>
/*
 * after_subject Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void after_subject()
{
        if(!traceMSG)
        {
                cout << "PROCESSING AFTER SUBJECT" << endl;
        }

        switch(next_token())
        {
                case WORD2:
                        verb();
                        getEword();
                        gen("ACTION", 'A');
                case VERB:
                case VERBNEG:
                case VERBPAST:
                case VERBPASTNEG:
                        tense();
                        gen("TENSE", 'T');
                        if(next_token() == PERIOD)
                        {
                                match(PERIOD);
                                trans<<endl;
                        }
                        else
                        {
                                syntaxerror1(PERIOD, currentWord);
                        }
                        break;

                case WORD1:
                case PRONOUN:
                        noun();
                        getEword();
                        after_noun();
                        break;

                default:
                        syntaxerror2(currentWord, "after_subject");
   }
}
```

```
//AFTER_NOUN FUNCTION
//Part B GRAMMAR: <after_noun>::= <be> PERIOD | DESTINATION <verb><tense> PERIOD| OBJECT [<noun>
DESTINATION ] <verb><tense> PERIOD
//Part C GRAMMAR: <after_noun>::= <be>#gen#PERIOD |
DESTINATION#gen#<verb>#getEword##gen#<tense>#gen#PERIOD|
OBJECT#gen#[<noun>#getEword#DESTINATION#gen#] <verb>#getEword##gen#<tense>#gen#PERIOD
/*
 * after_noun Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void after_noun()
{
	if(!traceMSG)
	{
		cout << "PROCESSING AFTER NOUN" << endl;
	}

	switch(next_token())
	{
		case IS:
		case WAS:
			be();
			gen("DESCRIPTION", 'D');
			if(next_token() == PERIOD)
			{
				match(PERIOD);
				trans << endl;
			}
			else
			{
				syntaxerror1(PERIOD, currentWord);
			}

			break;

		case DESTINATION:
			match(DESTINATION);
			gen("TO", 'A');
			verb();
			getEword();
			  gen("ACTION", 'A');
			tense();
			gen("TENSE",'T');
			if(next_token() == PERIOD)
			{
				match(PERIOD);
				trans << endl;
			}
			else
			{
				syntaxerror1(PERIOD, currentWord);
			}

			break;

		case OBJECT:
			match(OBJECT);
			gen("OBJECT", 'A');
			if(next_token() == WORD1 || next_token() == PRONOUN)
			{
				noun();
				getEword();
				if(next_token() == DESTINATION)
				{
					match(DESTINATION);
					gen("TO", 'A');
				}
				else
				{
					syntaxerror1(DESTINATION, currentWord);
```

```
                        }
                }

                verb();
                getEword();
                gen("ACTION", 'A');
                tense();
                gen("TENSE", 'T');

                if(next_token() == PERIOD)
                {
                        match(PERIOD);
                        trans << endl;
                }
                else
                {
                        syntaxerror1(PERIOD, currentWord);
                }

                break;
                default:
                        syntaxerror2(currentWord, "afterNOUN");
        }
}
/*
 * be Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void be()
{
        if(!traceMSG)
        {
                cout << "PROCESSING BE" << endl;
        }

        switch(next_token())
    {
                case IS:
                        match(IS);
                        break;

                case WAS:
                        match(WAS);
                        break;

                default:
                        syntaxerror2(currentWord, "be");
    }
}
```

```
/*
 * noun Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void noun()
{
	if(!traceMSG)
	{
		cout << "PROCESSING NOUN" << endl;
	}

	switch(next_token())
	{
		case WORD1:
			match(WORD1);
			break;

		case PRONOUN:
			match(PRONOUN);
			break;

		default:
			cout<<currentWord<<" is not connector or noun -none of the alternative
fit"<<endl;
			syntaxerror2(currentWord, "noun");
			break;
	}
}
/*
 * tense Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void tense()
{
	if(!traceMSG)
	{
		cout << "PROCESSING TENSE" << endl;
	}

	switch(next_token())
	{
		case VERB:
			match(VERB);
			break;

		case VERBNEG:
			match(VERBNEG);
			break;

		case VERBPAST:
			match(VERBPAST);
			break;

		case VERBPASTNEG:
			match(VERBPASTNEG);
			break;

		default:
			syntaxerror2(currentWord, "tense");
			break;
	}
}
```

```
/*
 * verb Function
 * Written by: Lenson Paulo Laca
 * Debug & Fix: Ly Dung
 */
void verb()
{
        if(!traceMSG)
        {
                cout << "PROCESSING VERB" << endl;
        }

        switch(next_token())
        {
                case WORD2:
                        match(WORD2);
            break;

                default:
                        syntaxerror2(currentWord, "verb");
                        break;
        }
}
```

Section 6: Parser test results

Test1:
Parser Funciton
Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: N
==========PROCESSING STORY==========
==========PROCESSING BEGINNING STORY==========
The current word is watashi
PROCESSING NOUN
Matched PRONOUN
The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT
The current word is rika
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is desu
PROCESSING BE
Matched IS
DESCRIPTION: rika
TENSE: IS
The current word is .
Matched PERIOD
The current word is watashi
==========PROCESSING BEGINNING STORY==========
PROCESSING NOUN
Matched PRONOUN

The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT
The current word is sensei
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is desu
PROCESSING BE
Matched IS
DESCRIPTION: sensei
TENSE: IS
The current word is .
Matched PERIOD
The current word is rika
==========PROCESSING BEGINNING STORY==========
PROCESSING NOUN
Matched WORD1
The current word is wa
Matched SUBJECT
ACTOR: rika
PROCESSING AFTER SUBJECT
The current word is gohan
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is o
Matched OBJECT
OBJECT: gohan
The current word is tabE
PROCESSING VERB
Matched WORD2
ACTION: tabE
PROCESSING TENSE
The current word is masu
Matched VERB
TENSE: VERB
The current word is .
Matched PERIOD
The current word is watashi
==========PROCESSING BEGINNING STORY==========
PROCESSING NOUN
Matched PRONOUN
The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT

The current word is tesuto
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is o
Matched OBJECT
OBJECT: tesuto
The current word is seito
PROCESSING NOUN
Matched WORD1
The current word is ni
Matched DESTINATION
TO: seito
PROCESSING VERB
The current word is agE
Matched WORD2
ACTION: agE
PROCESSING TENSE
The current word is mashita
Matched VERBPAST
TENSE: VERBPAST
The current word is .
Matched PERIOD
The current word is shikashi
==========PROCESSING BEGINNING STORY==========
Matched CONNECTOR
CONNECTOR: However
PROCESSING NOUN
The current word is seito
Matched WORD1
The current word is wa
Matched SUBJECT
ACTOR: seito
PROCESSING AFTER SUBJECT
The current word is yorokobI
PROCESSING VERB
Matched WORD2
ACTION: yorokobI
PROCESSING TENSE
The current word is masendeshita
Matched VERBPASTNEG
TENSE: VERBPASTNEG
The current word is .
Matched PERIOD
The current word is dakara
==========PROCESSING BEGINNING STORY==========
Matched CONNECTOR
CONNECTOR: Therefore

PROCESSING NOUN
The current word is watashi
Matched PRONOUN
The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT
The current word is kanashii
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is deshita
PROCESSING BE
Matched WAS
DESCRIPTION: kanashii
TENSE: WAS
The current word is .
Matched PERIOD
The current word is soshite
==========PROCESSING BEGINNING STORY==========
Matched CONNECTOR
CONNECTOR: Then
PROCESSING NOUN
The current word is rika
Matched WORD1
The current word is wa
Matched SUBJECT
ACTOR: rika
PROCESSING AFTER SUBJECT
The current word is toire
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is ni
Matched DESTINATION
TO: toire
PROCESSING VERB
The current word is ikI
Matched WORD2
ACTION: ikI
PROCESSING TENSE
The current word is mashita
Matched VERBPAST
TENSE: VERBPAST
The current word is .
Matched PERIOD
The current word is rika
==========PROCESSING BEGINNING STORY==========

PROCESSING NOUN
Matched WORD1
The current word is wa
Matched SUBJECT
ACTOR: rika
PROCESSING AFTER SUBJECT
The current word is nakI
PROCESSING VERB
Matched WORD2
ACTION: nakI
PROCESSING TENSE
The current word is mashita
Matched VERBPAST
TENSE: VERBPAST
The current word is .
Matched PERIOD
The current word is eofm
=====END-Prasering=====
Test2:
Parser Funciton
Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: n
==========PROCESSING STORY==========
==========PROCESSING BEGINNING STORY==========
The current word is soshite
Matched CONNECTOR
CONNECTOR: Then
PROCESSING NOUN
The current word is watashi
Matched PRONOUN
The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT
The current word is rika
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is desu
PROCESSING BE
Matched IS
DESCRIPTION: rika
TENSE: IS
The current word is ne
SYNTAX ERROR: EXPECTING PERIOD BUT FOUND ne
Test3:
Parser Funciton

Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: n
==========PROCESSING STORY==========
==========PROCESSING BEGINNING STORY==========
The current word is dakara
Matched CONNECTOR
CONNECTOR: Therefore
PROCESSING NOUN
The current word is watashi
Matched PRONOUN
The current word is de
SYNTAX ERROR: EXPECTING SUBJECT BUT FOUND de

Test4:
Parser Funciton
Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: n
==========PROCESSING STORY==========
==========PROCESSING BEGINNING STORY==========
The current word is watashi
PROCESSING NOUN
Matched PRONOUN
The current word is wa
Matched SUBJECT
ACTOR: I/me
PROCESSING AFTER SUBJECT
The current word is rika
PROCESSING NOUN
Matched WORD1
PROCESSING AFTER NOUN
The current word is mashita
SYNTAX ERROR: UNEXPECTED WORD: mashita FOUND IN afterNOUN5:

Test5:
Parser Funciton
Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: n
==========PROCESSING STORY==========
==========PROCESSING BEGINNING STORY==========
The current word is wa
PROCESSING NOUN
wa is not connector or noun -none of the alternative fit
SYNTAX ERROR: UNEXPECTED WORD: wa FOUND IN noun
Test 6:
Parser Funciton

```
Enter Filename of the File you want to parse:
test.txt
KEEP TRACK OF PARSING PROCESS? Y / N: n
===============PROCESSING STORY======================
======================PROCESSING BEGINNING STORY====================
The current word is apple
Lexical Error: apple is not valid token
PROCESSING NOUN
apple is not connector or noun -none of the alternative fit
SYNTAX ERROR: UNEXPECTED WORD: apple FOUND IN noun
```

Section 7: Updated Parser code for translation

<story> ::= <S1> { <S1> }

<S1> ::= [CONNECTION # getEword # gen #] <noun> # getEword # SUBJECT # gen

<S2> ::= <verb> # getEword # gen # <tense> # gen # PERIOD | <noun> # getEword # <S3>

<S3> ::= <be> # gen # PERIOD | DESTINATION # gen # <verb> # getEword # gen # <tense> # gen # PERIOD | OBJECT # gen # <noun> # getEword # DESTINATION # gen # <verb> # getEword # gen # <tense> # gen # PERIOD

Section 7: Semantic functions commented with the functionality and the name of the author.

```
/*
 * gen Function
 * Written by: Chris Childers
 * Debug & Fix: Lenson Paulo Laca, Ly Dung
 */
void gen(string name, char type)
{
  switch(type)
    {
    case 'A':
      cout << name << ": " << saved_E_word << endl;
      trans << name << ": " << saved_E_word << endl;
      break;

    case 'T':
      cout << name << ": " << convertT(currentToken) << endl;
      trans << name << ": " <<convertT(currentToken)<< endl;
      break;

    case 'D':
      cout <<  name << ": " << saved_E_word << endl;
      cout << "TENSE" << ": " << convertT(currentToken) << endl;
      trans <<  name << ": " << saved_E_word << endl;
      trans << "TENSE" << ": " << convertT(currentToken) << endl;
      break;
    }
}

/*
 * getEword Function
 * Written by: Chris Childers
 * Debug & Fix: Lenson Paulo Laca, Ly Dung
 */
void getEword()
{
  saved_E_word = convertE(currentWord);
}
```

Section 8: Semantic test results

Translated.txt

ACTOR: I/me
DESCRIPTION: rika
TENSE: IS

ACTOR: I/me
DESCRIPTION: sensei
TENSE: IS

ACTOR: rika
OBJECT: gohan
ACTION: tabE
TENSE: VERB

ACTOR: I/me
OBJECT: tesuto
TO: seito
ACTION: agE
TENSE: VERBPAST

CONNECTOR: However
ACTOR: seito
ACTION: yorokobI
TENSE: VERBPASTNEG

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

CONNECTOR: Then
ACTOR: rika
TO: toire
ACTION: ikI
TENSE: VERBPAST

ACTOR: rika
ACTION: nakI
TENSE: VERBPAST