CS421 Project Japanese-English Translator

This project will demonstrate to you that what you learned in CS421 is useful in writing a natural language processor.

You will write a small New-Japanese to English translator by completing the scanner (Part A) and the parser (Part B) parts for New-Japanese.

You will also do simple translation (Part C). The lexicon will be used for this purpose.

Project Part A - The Scanner

- **Must email a correct DFA to me before working on the scanner. Due WEEK8 Monday 10/16.**
- **Preliminary executable is due WEEK10 Monday 10/30.**
- **All required files are in /cs/cs421LK/CS421Progs/ScannerFiles on Empress. Must use them so that Parser can be implemented later without difficulties.**
- **See @@@@ for common errors people have made.**

Your **tokens' regular expressions** are as follows. The capitalized names are the actual token types to be returned to the parser.

Words:	**returned token type**
(vowel | vowel **n** | consonant vowel | consonant vowel **n** | consonant-pair vowel | consonant-pair vowel **n**)^+	
if it ends in a lower case vowel or 'n'	**WORD1**
if it ends in I | E	**WORD2**

Where consonants are:

b **d** g h **j** k m n p r s t **w y z** **(note that d,j,w,y,z do not start consonant pairs)**

consonant pairs are:

by
gy
hy
ky
my
ny
py
ry
ch **(note that c is not listed as a consonant)**
sh
ts

and vowels are:

a, i, u, e, o (but I and E uppercase are allowed)

Punctuation:	**returned token type**
.	**PERIOD** (for end of a sentence)

Sample words are:		**showing the parts:**
aoi	WORD1 i ending	(a-o-i)
okashii	WORD1 i ending	(o-ka-shi-i)
agE	WORD2 E ending	(a-gE)
takI	WORD2 I ending	(ta-ta-kI) -- (ends in capital **i**)
nyuuryoku	WORD1 u ending	(nyu-u-ryo-ku)
panda	WORD1 a ending	(pan-da)
hon	WORD1 n ending	(hon)

http://cgi.csusm.edu/ryoshii/japaneseDemo/aiueo.html (for sounds/moras)

Reserved words: **returned token type**

(English translation are found below the words for your convenience)
These are a subset of Words.

Verb Markers:

masu	**VERB**
masen	**VERBNEG**
mashita	**VERBPAST**
masendeshita	**VERBPASTNEG**
desu	**IS**
deshita	**WAS**

Particles:

o	**OBJECT**
wa	**SUBJECT**
ni	**DESTINATION**

http://cgi.csusm.edu/ryoshii/japaneseDemo/syntax.html (for Japanese particles)

watashi, anata, kare, kanojo, sore (I/me) (you) (he/him) (she/her) (it)	**PRONOUN**
mata (Also)	**CONNECTOR**
soshite (Then)	**CONNECTOR**
shikashi (However)	**CONNECTOR**
dakara (Therefore)	**CONNECTOR**
eofm	**EOFM** (end of file marker)

Assumptions to make your job easier:

You may assume that each token is preceded and followed by blanks.
e.g. watashi wa gakkou ni ikI mashita .

You may assume that all words are in **lower case** except for **I** and **E** in some cases.
You may also assume that there is an **eofm marker** at the end of the input file.

Sample of text that will be read in from file:

watashi wa rika desu .
watashi wa sensei desu .
rika wa gohan o tabE masu .
watashi wa tesuto o seito ni agE mashita .
shikashi seito wa yorokobI masendeshita .
dakara watashi wa kanashii deshita .
soshite rika wa toire ni ikI mashita .
rika wa nakI mashita .
eofm

STEP 1: NFA –to- DFA due on/before Week 8 Monday 10/16:

Using my NFA, you must draw the DFA for Words.
(Yes, there are **only two DFA's** for this project: one for WORD and another for PERIOD (a simple DFA that accepts the character '.') because **we will not build the "the last character check" into the DFAs).**

STEP 2: What your scanner should do (see scanner.cpp):

Your scanner should modify the HW2 program to process the above tokens
instead of the ones for HW2.

You must write a function for each DFA.
You must make sure the state numbers are the same as those in your final DFAs.

void Scanner
1. Get the next string up to a blank or end of line
2. Call the two token functions one after another (if-then-else)
   And generate a lexical error (***) if both DFAs failed and
   let the token_type be ERROR in that case.
3. For non-word tokens (i.e. period)
   return the token type with the actual token string.
4. Make sure WORDs are checked against the reserved words table (**)
   If not reserved, token_type is WORD1 or WORD2.
   based on the last character and return the actual string.
   e.g. WORD1 plus "seito"
   e.g. WORD2 plus "tabE"
4. Return the token type & string **(pass by reference)**

(**)**When a WORD string is found**, your scanner should go through a table of
reserved words to see if it is there. **(use the content of the reservedwords.txt file to set this table up)**
If it is a **reserved word**
  return the token type and the actual string.
  e.g. PRONOUN plus "watashi"

Please do not read from any file to prepare the reserved words table.
I should be able to test your program without relying on any of your input files.

(***) Errors to be generated must use the exact language below: @@@@
If a string does not fit ANY token description, generate:
"Lexical error: *string* is not a valid token"
e.g. "Lexical error: apple is not a valid token"

Scanner Test Driver: (DO NOT CHANGE MY CODE)
void main Your test driver **for now**
1. gets the input file name from the user
2. opens the input file which contains a story written in Japanese (fin.open).
3. **calls Scanner** repeatedly until the EOF marker is read, and
   each time cout the returned results as follows:

e.g. STRING TOKEN-TYPE

watashi	PRONOUN	(from the first call)
wa	SUBJECT	(from the second call)
rika	WORD1	
etc.		

Required Comments:@@@@

For each function in the Scanner, you must comment it with **at least** the following info:

1. the **regular expression** (RE) for the token
2. the **programmer's name**

Required Testing:

Test cases are available in **scannertest1 and scannertest2.** Their outputs are in **correctoutput** files.

Must Divide the Work as follows:

- **Member A and B: NFA to DFA conversion**
- **Member C: Code the functions for DFAs**
- **Member A and B: Scanner()**

Everyone must come to a meeting physically to put together the work and help debug each other's code. If a member is failing to do his/her work, other members should take over the task to complete the program so that it can be used for the next part of the project. You will get to report who should get the credit for the task in a file called ScannerWork.txt.

What to Submit on/before Week 10 Monday 10/30:

You will have to submit your program (only the executable) to be tested by me before you can move onto the parser.

/cs/submitIt_LK G# executable (.out created on empress @@@@)
/cs/submitIt_LK G# scannerWork.txt (filled out honestly)
where G# is your designated group number (for example, G12)

And 72 hours later
/cs/checkIt_LK G# (to get my feedback)

[5pts toward the project] for contributing members only if both the DFA and these files were received before the due dates and corrected right away as requested.

The final version of the DFAs should go into your project report. Original scanner.cpp and test results should go into your project report. So keep them in a safe place!!!

End of Part A.

PART B will be Parsing. Your parser will replace your test driver.