Here is the provided Lex (Flex) program with detailed comments explaining each line of code:

c

CopyEdit

%{

/\*

\* We expand upon the first example by adding recognition of other

\* parts of speech such as verbs, adverbs, conjunctions, prepositions, etc.

\* This program uses Lex (Flex) to recognize parts of speech in a given string.

\*/

#include<stdio.h>

%}

%%

/\* Rule to ignore whitespace characters (tabs) \*/

[\t]+ /\* ignore whitespace \*/ ;

/\* Verbs: if the word is a verb, print it as a verb \*/

is |

am |

are |

were |

was |

be |

being |

been |

do |

does |

did |

will |

would |

should |

can |

could |

has |

have |

had |

go {

printf("%s: is a verb\n", yytext); /\* Print the verb \*/

}

/\* Adverbs: if the word is an adverb, print it as an adverb \*/

very |

shrPly |

gently |

quietly |

calmly |

angrily {

printf ("%s: is an adverb\n", yytext); /\* Print the adverb \*/

}

/\* Prepositions: if the word is a preposition, print it as a preposition \*/

to |

from |

behind |

above |

below |

between {

printf ("%s: is a preposition\n", yytext); /\* Print the preposition \*/

}

/\* Conjunctions: if the word is a conjunction, print it as a conjunction \*/

if |

then |

and |

but |

or {

printf ("%s: is a conjunction\n", yytext); /\* Print the conjunction \*/

}

/\* Adjectives: if the word is an adjective, print it as an adjective \*/

their |

my |

Your |

his |

her |

its {

printf("%s: is an adjective\n", yytext); /\* Print the adjective \*/

}

/\* Pronouns: if the word is a pronoun, print it as a pronoun \*/

I |

YOU |

he |

she |

we |

they {

printf("%s: in a pronoun\n", yytext); /\* Print the pronoun \*/

}

/\* Any alphabetic string (not recognized explicitly as a specific part of speech)

might be a noun \*/

[a-zA-Z]+ {

printf("%s: don't recognize, might be a noun\n", yytext); /\* Print that the word is possibly a noun \*/

}

/\* Default rule for any character or new line, just print the character \*/

.|\n { ECHO; /\* Print any character or new line by default \*/ }

%%

/\* yywrap is required by Lex but not used in this program \*/

int yywrap(void) {

return 1; /\* Return 1 to indicate end of input \*/

}

int main() {

/\* Print a prompt asking for user input \*/

printf("Enter a string :- ");

/\* Call yylex() to begin lexical analysis \*/

yylex();

return 0; /\* Return from main function \*/

}

**Explanation of the Code:**

1. **Declarations (%{ ... %}):**
   * The part between %{ and %} is the C code that is copied into the generated C file. This includes necessary imports like stdio.h.
2. **Lexical Rules (%%):**
   * The patterns defined inside the %% section define the rules for matching different parts of speech. Each pattern is followed by a block of C code to execute when that pattern is matched. For instance, the word is will match the pattern is and print it as a verb.
3. **Whitespace and Default Behavior:**
   * Whitespace ([\t]+) is ignored, and any unmatched character or newline is echoed as the default action (.|\n { ECHO; }).
4. **Lexical Analysis (yylex()):**
   * The yylex() function starts the lexical analysis, scanning the input for words that match the defined rules.
5. **yywrap():**
   * The yywrap() function is a placeholder that Lex calls when it reaches the end of input. It returns 1 to indicate the end of input.
6. **Main Function:**
   * The main() function prompts the user to enter a string, then invokes yylex() to start the lexical analysis.

**How to Run the Program:**

Follow these steps to run the Lex program:

1. **Install Flex (if not installed):**
   * On **Ubuntu/Debian**:

bash

CopyEdit

sudo apt-get install flex

* + On **MacOS** (using Homebrew):

bash

CopyEdit

brew install flex

1. **Create the Lex File:**
   * Save the code above into a file with a .l extension, such as part\_of\_speech.l.
2. **Generate C Code:**
   * Open your terminal and navigate to the directory where the .l file is saved. Run the flex command to generate the C source code:

bash

CopyEdit

flex part\_of\_speech.l

* + This will create a file named lex.yy.c.

1. **Compile the C Code:**
   * Use gcc to compile the generated C file (lex.yy.c) into an executable. You need to link the flex library (-lfl):

bash

CopyEdit

gcc lex.yy.c -o lexer -lfl

1. **Run the Program:**
   * Now, you can run the lexer:

bash

CopyEdit

./lexer

1. **Input:**
   * After running the program, it will prompt you to enter a string. For example:

less

CopyEdit

Enter a string :- I am going to the park quietly.

1. **Expected Output:**  
   The output will be something like:

vbnet

CopyEdit

I: in a pronoun

am: is a verb

going: don't recognize, might be a noun

to: is a preposition

the: don't recognize, might be a noun

park: don't recognize, might be a noun

quietly: is an adverb

This way, the Lex program identifies different parts of speech in a given input string and outputs them accordingly.

Python  
  
Sure! Below is a Python version of the Lexical Analyzer for identifying parts of speech. The program uses regular expressions to match words and classify them into various parts of speech (verb, adverb, preposition, conjunction, adjective, pronoun, and noun).

Here’s the Python code:

python

CopyEdit

import re

# Function to identify the part of speech

def identify\_part\_of\_speech(word):

# Define the parts of speech using regular expressions

verbs = ["is", "am", "are", "were", "was", "be", "being", "been", "do", "does", "did", "will", "would", "should", "can", "could", "has", "have", "had", "go"]

adverbs = ["very", "sharply", "gently", "quietly", "calmly", "angrily"]

prepositions = ["to", "from", "behind", "above", "below", "between"]

conjunctions = ["if", "then", "and", "but", "or"]

adjectives = ["their", "my", "your", "his", "her", "its"]

pronouns = ["I", "YOU", "he", "she", "we", "they"]

# Check if the word is a verb

if word in verbs:

return f"{word}: is a verb"

# Check if the word is an adverb

if word in adverbs:

return f"{word}: is an adverb"

# Check if the word is a preposition

if word in prepositions:

return f"{word}: is a preposition"

# Check if the word is a conjunction

if word in conjunctions:

return f"{word}: is a conjunction"

# Check if the word is an adjective

if word in adjectives:

return f"{word}: is an adjective"

# Check if the word is a pronoun

if word in pronouns:

return f"{word}: is a pronoun"

# If the word is made up of only letters (likely a noun)

if re.match(r'^[a-zA-Z]+$', word):

return f"{word}: don't recognize, might be a noun"

# For any other characters (like punctuation or numbers)

return f"{word}: is an unknown part of speech"

# Function to tokenize input and classify parts of speech

def tokenize(input\_text):

# Split the input string by whitespace or punctuation

words = re.findall(r'\b\w+\b', input\_text) # Use regular expression to match words

# Identify the part of speech for each word

for word in words:

print(identify\_part\_of\_speech(word))

# Main function

if \_\_name\_\_ == "\_\_main\_\_":

input\_text = input("Enter a string: ") # Take input from user

tokenize(input\_text) # Tokenize and classify the parts of speech

**Explanation of the Code:**

1. **Regular Expressions**:
   * The re module is used to match words in the input string. The regular expression r'\b\w+\b' matches all words consisting of alphanumeric characters and underscores, ignoring punctuation marks.
2. **Parts of Speech Classification**:
   * The program checks if the word belongs to one of the predefined lists (verbs, adverbs, prepositions, conjunctions, adjectives, pronouns).
   * If the word does not match any of the categories, the program assumes that it could be a noun (based on the fact that it is a word consisting of letters).
3. **Input and Output**:
   * The program takes a string input from the user, tokenizes it into words, and classifies each word according to its part of speech.

**How to Run It:**

1. Save the code in a file, e.g., part\_of\_speech.py.
2. Open a terminal or command prompt.
3. Run the script by executing:

bash

CopyEdit

python part\_of\_speech.py

1. Enter a sentence when prompted, and the program will classify the parts of speech for each word in the sentence.

**Example:**

For the input:

less

CopyEdit

Enter a string: I am going to the store

The output would be:

vbnet

CopyEdit

I: is a pronoun

am: is a verb

going: don't recognize, might be a noun

to: is a preposition

the: don't recognize, might be a noun

store: don't recognize, might be a noun

This code can be expanded further by adding more parts of speech or more complex logic if needed.