CMPILER Problem Set: Creating a Lexical Analyzer or Scanner Instructor: Neil Patrick Del Gallego

[100 pts] General Instructions: From techniques you have picked up in class, a scanner or lexical analyzer converts a stream of input into a list of tokens. In this problem set, you are tasked to create your own lexical analyzer for recognizing valid tokens, following the lexical rules of **MIPS64**, a reduced instruction set for CPU.

Restrictions
In this hands-on activity, you are not allowed to use any scanner or parsing libraries. This includes ANTLR, Java/C++/C# RegEx utilities, and the like. Only your chosen programming language, native string operations such as concat, format, split, and similar, and a UI library (optional) is allowed. You must write your own scanner from scratch. Using any restricted library will automatically earn a 0 for this exam score.

MIPS64 Lexical Rules
• General Purpose Registers (GPR) – There are 32 registers for MIPS64, which starts at 0 and ends at 31. The register format can either have a prefix 'R' or a prefix '$.' Examples are: R0, R1, R2, R30 or $1, $2, $3, $31.
• Double Precision Floating Point Registers (FPR) – There are 32 FPR registers for MIPS64, which starts at 0 and ends at 31. The FPR format can either have a prefix 'F', or a prefix of '$F'. Examples are: F1, F2, F30 or $F1, $F2, $F30.
• Support for the following keywords: DADDU, DADDIU, DMULT, DMULTU.
• Delimiters are "," "<whitespace>", "\n"

A reference MIPS64 code is shown below. Note that you only need to implement the lexical rules declared above!

```
    lw    $t0, 1060($gp)    # fetch background
    andi  $t1, $t0, 0xff00    # isolate blue
    sll   $t1, $t1, 2         # times 2
    andi  $t1, $t1, 0xff00    # get rid of overflow
    lui   $t2, 0xffff         # $t2 = 0xffff0000
    ori   $t2, $t2, 0x00ff    # $t2 = 0xffff00ff
    and   $t0, $t0, $t2       # get rid of old value of blue
    or    $t0, $t0, $t1       # new value
    sw    $t0, 1060($gp)     # background = ...
```

Input
The input will be the following:
- A text file containing **n** lines of MIPS64 instructions. Each line contains **C** characters that follows the prescribed format of MIPS64.
- All inputs are assumed to be in uppercase format.

Output
- A text file containing **n** lines where each line contains **K** token labels of the following: **KEYWORD GPR FPR.** Delimiters are not printed.
- If the set of characters cannot be recognized, then print **ERROR**.
- For example, if the input is "**DADDU R5, R0, R30"**, then the output of the lexical analyzer should be the following token labels in order: **KEYWORD GPR GPR GPR.**

Since this is simply a scanner/lexical analyzer, ordering of tokens may be jumbled and should not be your concern. Consider this example: **DADDU DADDIU $5 DADDIU DMULTU.** The output should then be: **KEYWORD KEYWORD GPR KEYWORD KEYWORD** even if this code is a syntax error in MIPS64 assembler.

Sample Test Case
The following test cases you may use are given in the table below:

Input	Output
DADDU R5, R0, R30 DADDU, DADDIU, DMULT, DMULTU DADDIU $F2, $F30 $F2, $F30 $F2, $F30 $F2, $F30 INT X = 5 $F2 INT DADDIU	KEYWORD GPR GPR GPR KEYWORD KEYWORD KEYWORD KEYWORD KEYWORD FPR FPR FPR FPR FPR FPR FPR FPR ERROR ERROR ERROR ERROR FPR ERROR KEYWORD

Grading Scheme

The **actual** test case will be different from the sample provided. It will consist of 15 test cases, each worth 4 points each. 4 points are given if the actual result yields the expected result.

Deductions will be given if the following requirements were not met:

Lexical Analysis
Does not use NFA/DFA/finite state accepter for recognizing tokens. Brute-force method.
No evidence of using tokens in code.

60 pts will be for the test cases. 40 pts will be for the PPT submission.

Submission Details
You may prepare some of the requirements in advance.

The **actual** test case will be given as a timed quiz in Canvas. You are to run the given test case and submit it online. After you are done running the program using the test case provided, submit a GDrive URL containing the following:

- SOURCE – Contains your source code. Add a README.txt that has your name and instructions how to run your program. Also indicate the entry class file, where the main function is located. Alternative can be a Github link.
- PPT – A PowerPoint showing the following set of slides.
    - Title page – Your name + section + a video demo in MP4 format. The video contains a recording of your running program, showing the **actual** test case file as input, and showing the output listing down the results in detail. Should be in MP4 file. Recording must be seamless and not cut. Please ensure that your video is embedded in your PPT.
    - Lexical analysis – Discussion of your lexical analyzer implementation which contains the following. Provide code snippets wherever necessary:
        - Reading the text input. How do you create tokens?
        - Show implementation of maximal munch algorithm.
        - Show your DFA implementation. You may provide a DFA diagram or table.
        - How do you deal with unrecognizable tokens?
- FORM AND OUTPUT –
    - Text file produced by your program which should have a similar format as discussed in the specs.
    - Academic honesty agreement form declaration. See below.

APPENDIX A: Academic Honesty Form

==========Copy and paste the following section below and sign the form. Save in PDF file================

ACADEMIC HONESTY AGREEMENT

I am answering this exam myself and to the best of my ability, without any assistance from other persons, materials, or resources that I am not allowed to access during the exam period. I declare that the software is fully written by me and without any assistance from my peers. I am fully aware and hereby agree to the clause that violation of this agreement is considered cheating and will result in a 0.0 for this course.

[illegible]

MARK LOUIS SANCHEZ LIM
Signature over Printed Name

==END==