PAL Syntax Checking
Due Feb 13

Note: There are intermediate deliveries for this assignment as noted in the text.

Description
The program takes a minimal assembly language program, and runs the first steps of a translator (compiler or assembler): removing comments and checking for syntax errors. Your program processes the code, and creates a log file with error messages (if any) and a comments-removed listing of the code.

Details

1. See the separate sheet for the syntax of the Pico Assembly Language (PAL). Additional note: labels follow the same naming rules as identifiers (followed by a colon).
2. High-level design of your main program:
   - open (input) PAL program file and (output) log file
   - for each PAL program line …
     … read line
     … if not blank line and not comment line
       … write line to log file, with sequence number and without any online comment
       … check errors (in order below), stopping at first error on that line
       … write error message (if any) to log file
   - write summary data to log file
   - close files
3. Your program takes a file name as an argument, and appends the extension *.pal* then uses that as the input file name. This file contains (what is supposed to be) a program written in PAL, in plain text form. You may assume that the incoming string follows file name syntax, and that the file exists, is in the right folder, can be opened, and is non-empty.
4. Your program should detect and report on the following errors:
   a. ill-formed label – the first token ends in a colon but doesn't follow the rules for a label name
   b. invalid opcode – the first non-comment string (token) found on a line is not one of the valid opcodes; exception: if the line contains a label, then the opcode should be the second string
   c. too many operands – for the specific opcode, there are more operands than expected (you may assume that any operands on the line are in a valid, comma-separated list for this and the next error)
   d. too few operands – for the specific opcode, there are fewer operands than expected (where possible)
   e. ill-formed operands – an operand name (id or register) is invalid, an immediate value is invalid for octal
   f. wrong operand type – number where id expected, or vice versa
   g. label problems: a branch goes to a non-existent label, or there is a label that is never branched to
5. Detection of an error on a line halts processing on that line (i.e., don't scan for further errors).
6. Make your error messages helpful and informative – remember the bad messages you have had to deal with as a programmer! Suggest corrections where it's easy. Easy example: "too many operands – expected 2 operands for ADD". Hard example: "invalid opcode SAB – should be SUB?".
7. **All processing takes place in one pass across the PAL code. Do not re-scan the code.**
8. You do not need a character-by-character lexical analyzer or tokens. Just use standard input string parsing (possibly built into your language) and string routines (but nothing that trivializes the main thrust of this assignment; if in doubt, ask me first). Use lots of little work routines (e.g., is-valid-id, process-add-op, is-octal-num).
9. Log file. Your program creates a log file in the same directory as the original (input) program. Its name is the same, minus the extension *.pal* and with the extension *.log* added.
   Log file format:

a. heading: some nice title, PAL file name input to the program, log file name, process date (use the system date), your name, CS 3210
b. line-by-line listing of source code lines, numbered sequentially; lines retain their original formatting, so no changes or alignment are needed
c. error messages (if any), are shown below their line of code
d. ending totals: number of lines of code (not blanks and not comment-only lines), number of errors found in each category (but don't list any categories with no errors), and total errors found
e. final statement: this PAL program is / is not valid
f. the log file does not contain: original blank lines in code, original code comments

Sample partial log file. Notice no blank lines or comments from the original code, and "unusual" formatting has not been normalized. Also notice, on line #5, only the first error is reported. Follow this log format.

```
                    <log file heading as given above>


PAL Program Listing:

1.  MOVE        R5, X
2.  ADD                     X,X,Y
3.  SBB         R4, R5, total
    ** error: invalid opcode SBB
4.  MOVE        total, R9
    ** error: ill-formed operand – bad register number
5.  ADD                     123go
    ** error: too few operands for ADD, expected 3 operands
(etc.)

Summary -------------

total errors = 3
    1  invalid opcode
    1  too few operands
    1  ill-formed operand
Processing complete – PAL program is not valid.
```

Development Notes

1. Language. You may use any language for this project. **Email me with your language choice by Friday, Jan 26..** If you use C++, your program should be fully object-oriented (use C if you don't want OO). Do not use any tools such as **lex, regular expressions, or tokenizers** that do the bulk of the work for you – you should be explicitly doing the parsing in your code. Also don't use built-in exception handling – **do error trapping yourself**.
2. Cover memo. Make notes for your cover memo as you work. Your final notes to me should be thorough and not written at the last minute – more specs below.
3. **HW1.1 Due Thursday, Feb 1:** Write the syntax rules for PAL using one of the notations in the book, eg. BNF, Denotational. This will help when you start to create the compiler.
4. **HW1.2 Detail Design due Thursday, Feb 8**. This is NOT code – this is design. Identify inputs, outputs, any algorithms needed, data structures, logic, etc Use English and/or pseudo-code as the

design evolves. I will ask during the 3 weeks what you have finished, what's in progress, what's not started, and what problems you are having or have had. Be prepared to send this information at any point.

5. Develop your code modularly.
   a. First, get a running program that simply produces a log file, without removing comments or blank lines, and without error checking. Use your PAL programs to use as a test case; it will not have blank lines, or comments, or errors.
   b. Next add processing for blank lines and comments; you should get clean log files because it has no errors (and because you are not doing any error checks …).
   c. Then add one error check at a time.
   d. Test each addition as you go, by introducing blank lines, then each type of comment, then each error type into the PAL plain test program I provided. For each error, first try a PAL program that has only that error, in several places and guises. Then try running against PAL code with all previous errors plus the new one (regression testing). Do not try to write code for all errors at once – it's very likely to cost you more time (and frustration!) in the end.
6. Testing. As above and … Once your program handles all you should have one complete test PAL program with. blank lines, comments, and all error types, including some repeated errors with different problems (for example, different operand errors such as "not composed of letters" and "length too long").
7. Create another PAL program from scratch for testing. It may or may not have all errors, but it should have some. Be sure to document them well. In addition, I will create test files based on class discussion.
8. Your final program should be well organized, well documented, modular, polished, yada yada yada Produce a high-quality project that you would be happy to show a prospective employer.
9. This is an individual project. Do not work with classmates, or get help from work or home. This is not an assignment in see-what-you-can-find-on-Google. Please do your own work. Exception: once you have the basic project completed and working fine, you and a classmate may discuss and work on some of the extra credit – just mention your classmate in your cover letter. Come to office hours MTWTh for help, hints and ideas, code walk-throughs, etc. **Don't get stuck making no progress for hours (or days!).**
10. Be sure to scrutinize your output at every step – are you getting what you should be getting?
11. As usual, aim to be done early to allow for problems, or to have time for extra credit.
12. Deadlines and Deliverables:
    a. Tues, Jan 23 in class today: preliminary schedule for entire project
    b. Friday, Jan 26 email me your choice of language
    c. before handing anything in, please double-check these specs carefully!
    d. Tues, Feb. 13 (3 weeks) – hand in these items:
       i. final source code, meeting all specs plus any extra credit
       ii. output logs (3) from my (unaltered) PAL test programs –plain text
       iii. your final 2 PAL testing programs. Please highlight or mark all errors in your PAL programs (and each should have a comment associated with it); example:

```
;;;  calculate total
SUB    Total, R3, R3         ; update total
ADDD   R1, R2                ; invalid opcode
```

       iv. output logs (2) from these (error-filled) PAL testing programs
       v. any extra credit test PAL programs and log files
       vi. final thorough cover memo (about 2 pages), including:
           - your total time – keep track as you go, don't guess or leave this out; nearest quarter hour is fine
           - "how'd it go"
           - briefly discuss whether your initial schedule was a fairly good plan, whether you followed it, and what you would change in general the next time you make a project schedule.

- a discussion of the ins and outs of the first pass of a translator across a program
- a discussion of 3 errors or problems that were not checked about PAL (syntax or semantics) that could be checked, and how hard it would be to add to your program
- a discussion of 3 error-checking issues did not arise using PAL, but that would be present in most high-level-language programs; discuss how they might be handled, and how that would add to the complexity of the translator

<u>Credit Ideas – ONLY after the primary assignment is complete and correct</u>

Do any one or more. Remember to list and discuss extra credit in your cover memo, and you may work with a classmate. You should hand in both the original log files (before extra credit) and enhanced log files. If you created a new small PAL program for any of these enhancements, include that and its log file in your final packet. Note: the code additions should take only about an hour or two (each) to code and test. Extra credit is due with the final project.

1. Check that the END instruction appears exactly once in the program, and is the last instruction in the program.
2. Process all errors in one line of PAL code, instead of stopping at the first error found. Provide multiple error messages in the log file. (Not always possible – why?)
3. Generate a list of identifiers, such as would be used to create a symbol table; give a list of these at the end of your log file. Do not show duplicate identifiers in the list. Registers do not count as identifiers.
4. Collect and report some fairly primitive "style" stats on the code itself, regarding white space and comments. Do these as you process the code (don't make a second pass). Use one of your two enhanced PAL programs, but show a new log file. Pick one or both of …
    a. Count the number of code lines (with or without on-line comments) and blank lines and comment-only lines as you process the PAL code. At the end of the log file, report each count followed by its percentage of the total number of lines.
    b. At a character level, count percentage of comments vs. code (do whatever you want with white space characters, just explain it) and report it in the log file.