Linter problem 29

20 points

A "linter" is a programming tool generally used to help find issues with code that might otherwise go missed. For example, some code might compile without errors, but have subtle bugs that could be caught via static analysis (like linting).

You are tasked with writing a simple linter that checks a new programming language, BALONEYPLUSPLUS, for the following code issues:

Lint Error Code	Message	Explanation
10	Variable declared but not used	A variable declaration statement exists but the variable is not referenced elsewhere
20	Unexpected indentation	A style check, improves readability of code
30	Trailing whitespace	A style check, lines should end with a line break, not a space or tab character
40	Func declaration without documentation	Encourages the programmer to add an explanatory comment to their functions

See the next page for the complete BALONEYPLUSPLUS Language Specification

Here is a valid code example

```
VAR i
VAR j
FOR $i = 1 TO 5 DO:
    PRINT SUM( $i , $j )
NEXT

# IF statement with correct indentation
IF SUM( 1, 1 ) != 2 THEN:
    PRINT "Something is seriously wrong"
ENDIF

# SUM returns the arithimetic total  of its two parameters
FUNC SUM( a, b )
    RETURN $a + $b
ENDFUNC

# Note required comment above "SUM" function declaration
```

Tip

Keep in mind that comments are ignored by the compiler, so from a language standpoint a variable appearing in a comment is not actually an instance of that variable being used.

Input

The input file is a single text document, the source code of a valid BALONEYPLUSPLUS program.

```
VAR flag
VAR result

$result = 0

FOR $i = 0 TO 10 DO:
   $result = $result + $i
NEXT

PRINT $result   (there are trailing spaces here)
```

Output

Your linter will scan the program for the listed lint error checks, and report the following information:

- Line number where the lint error occurred
- The lint error code
- The lint Message (from the table above, must match exactly including case)

For example, this is expected output for the above input example:

```
1:10 Variable declared but not used
7:20 Unexpected indentation
10:30 Trailing whitespace
```

Report lint errors in sorted order. Sort by line number (ascending). If one line has multiple lint errors, sort by lint error code (ascending).

Linter (continued) - BALONEYPLUSPLUS Language Spec

General Guidelines

- The language is case sensitive
- Variables must be declared with `VAR foo`, and then used like `$foo`.
- Variables must be surrounded by whitespace, "MYFUNC($i)" is invalid but "MYFUNC($i)" is correct.
- Nested loops and conditionals are not allowed.

Conditional Statements

```
IF <condition> THEN:
    <statements>
    <...>
ENDIF
```

Note each statement in the inner block is indented 4 spaces.

Loops

Valid `FOR` loops:

```
FOR $z = 1 TO 10 DO:
    <statements>
NEXT

VAR num
$num = 10
FOR $z = 1 TO $num DO:
    <statements>
NEXT
```

Note the inner block is indented 4 spaces.

Whitespace Before Statement

In this language, all lines should begin with a non-whitespace character, unless indentation is required as already specified. For example, this is correct:

```
# some comment
VAR xyz
```

This is not correct:

```
# some comment
 VAR xyz
```

Empty lines (lines without any non-space characters) are ignored.

Comments

Comments begin with the symbol # and indicate the remainder of the line is a comment. Comments are discarded by the compiler before building the application.

Examples:

```
# this is a comment
VAR xyz # this is also a comment
```

Function Declarations

Syntax:

```
FUNC <name>( <arg1>, <arg2>, ... )
    <statements>
ENDFUNC
```

Example:

```
FUNC myfunc( a, foo )
    VAR b
    $b = $a + $foo
    RETURN $b
ENDFUNC

FUNC limitOne( j )
    RETURNIF $j == 0 : 0
    RETURNIF $j > 0 : 1
ENDFUNC
```

To encourage documentation, the language requires that function declarations are preceeded with an explanatory comment. The comment must begin with the function name and it must be written on the line immediately above the function declaration. Example:

```
# myfunc adds a and foo and returns the result
FUNC myfunc( a, foo )
...
```