COL100 Assignment 5, Part 2

Due date: 8 March 2022

In this second part of Assignment 5, we will extend the simple Python interpreter we developed earlier, to handle a **while** loop.

Our task is to build an interpreter for a simple Python program with a limited syntax, simulating the computer's execution. The input text file consists of a sequence of statements, each statement on a separate line. The informal syntax is indicated below.

Informal Grammar

The input file is a STATEMENT_LIST. A STATEMENT_LIST consists of one or more statements.

A STATEMENT is one of:

- VARIABLE = EXPRESSION
- **while** EXPRESSION :
  STATEMENT_LIST

As in Python, the "**while** EXPRESSION :" occurs on a line by itself, followed by STATEMENT_LIST, with one statement on each line. The STATEMENTS that form the STATEMENT_LIST, are offset by an additional TAB (the '\t' character). The code provided to you counts the number of leading tabs occurring in the line. You can use the tab count to identify the list of STATEMENTs that form the loop body. If you detect a syntax error, print an error message and exit the interpreter.

The conditional EXPRESSION in the **while** must be of boolean type (e.g., "$a > b$"); other expression types (e.g., "$a + b$") are not allowed. Report syntax error if you detect a violation.

The EXPRESSION syntax is unchanged from Assignment 5, Part 1. EXPRESSION is one of:

- TERM
- UNARY_OPERATOR TERM
- TERM BINARY_OPERATOR TERM

BINARY_OPERATOR is one of: $+, -, *, /, >, <, >=, <=, ==$, !=, and, or
UNARY_OPERATOR is one of: $-$, not
TERM is one of: VARIABLE, INTEGER_CONSTANT, True, False
VARIABLE is a sequence of one or more letters
INTEGER_CONSTANT is a sequence of one or more numeric characters ('0' to '9')

Interpreting the *while* loop

Define a class **Instructions** and store the input program as a list of Instructions. Decide on the appropriate attributes (e.g., operation and operands) of this class. The statements you have already handled in Assignment 5, Part 1, should become instructions. A few more instructions would be needed to handle the **while** loops.

Assume that the processor provides the following instructions for your use:

- BLE (branch if less than or equal)
- BLT (branch if less than)
- BE (branch if equal)
- Branch (branch unconditionally)

Implement the loop in terms of these instructions using ideas discussed in class. Your overall strategy to translate the program into instructions should be:

- First determine the loop body for each **while** loop. The **tab** structure should help in this.
- Based on the conditional expression, replace the loop instructions by appropriate branch instructions (BLE/BLT/BE/Branch) in the instruction list.
- Make sure you have the correct *destinations* in the branch instructions. The instruction list should be ready after the previous step, so the list index of the appropriate instruction can be used as the branch destination.
- Now you have the instructions ready. Start interpreting from the first instruction onwards (using the same algorithm as in Assignment 5, Part 1), ensuring that if you are interpreting a branch instruction, the correct destination is followed based on an evaluation of the conditional expression.

Output

When the program completes, print out, as in Assignment 5, Part 1:

- The name and current value of all the variables used in the input program.

- The list of GARBAGE integer objects used in the program but not referred to any more by any variable at the end of the program.

Notes

- Use the code provided to read the input file and count leading tabs in each line.
- Always get simple cases working first before proceeding to more general solutions.
- Handle a simple **while** loop first before handling nested loops.
- Notice that the same branch instruction (e.g., BLE) could be used to handle both a condition (e.g., $a > b$) and its complement $(a <= b)$ with minor changes.
- Try your best! Dont worry if the solution is not 100% general.