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.

Note (submission guidelines):

- Please provide 2 test cases for which you have make your program run and corresponding time complexity .....in comments
- Provide the answer of each test case you have run .....in comments
- We will be providing test cases (same as given in example) on piazza .... make necessary changes in it and remember the changes you have made..... you have to make same changes to hidden test cases provided at time of demo
- Write a general comment stating the exception you have handled in your program (like invalid syntax of while loop) and approach you have followed
- You have lot of time left after major, take advantage of it.
- instruction class has instruction type and an execute function - which depending on the instr type executes the instruction and changes the program counter, one must not use external functions to do but use the class's functions

Instructions:
All programs are to be written in Python. Functions must be documented with proper comments stating

- the purpose of the function
- the types and meanings of the input arguments
- any conditions on the input arguments (input specifications)
- the type and meaning of the return values
- conditions on the outputs (output specifications)

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.

Example1

Examples are self explanatory

```
a = 10
b = 1
while a > b:
    ##input_prg_2
    a = a - 1
c = 1
```

- instruction list:

0	1	2	3	4	5
a = 10	b = 1	BLE a,b, 5	a = a-1	branch 2	c = 1

- Data list:

0	1	2	3	4	5	6	7	8	9	10	11	12
10	a,2	1	b,2	9	8	7	6	5	4	3	2	c,2

Example2

Examples are self explanatory

```
i = 0
while i < 3:
    j = 1
    while j < 2:
        x = i + j
        j = j + 1
    i = i + 1
```

```
y = 0
```

- instruction list:

0	1	2	3	4	5	6	7	8	9
i = 0	BLE 3,i, 9	j = 1	BLE 2,j, 7	x = i + j	j = j + 1	branch 3	i = i + 1	branch 1	y = 0

- Data list:

0	1	2	3	4	5	6	7
0	i,2	3	1	j,5	2	x,2	y,0

Output

Print out Instruction list .

- Instruction list should contain minimal number of Branch statements
- **Hint - Identify pattern in above example**

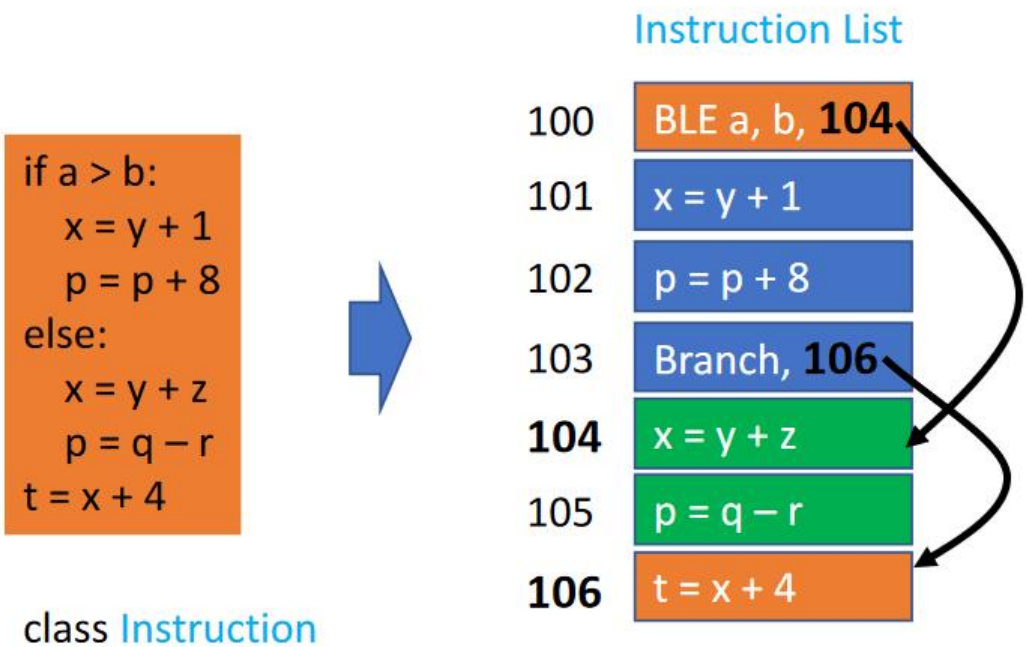
Also, **After each iterations of while loop** and when the **program completes,** print out, as in Assignment 5, Part 1:

- 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.
- Snippet of class instruction is also given
- After instruction list is created, then you need to execute/run it using the algorithm in assignment 5 part I. Only difference is that for new branch instruction they need to jump to corresponding instruction instead of updating any variable.
- 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.
- You can see in pic below, each statement in the program will be converted to respective instruction class object which will have opcode, operands etc as attributes.Example provided is for if-else statement, you should take idea from this and implement for while loop.

- Try your best! Do not worry if the solution is not 100% general.
- Keep checking piazza and assignment for query resolution and submission guidelines.




```
class Instruction
    "instruction"
    def __init__ (self, ...)
        self. ... =
        self. ... =
```