COSC4315: A Basic Interpreter of Python Programs

1 Introduction

You will create a Python interpreter written in C++ that can evaluate simple variable assignments with integer arithmetic expressions. You do not have to consider infinite integers or real numbers.

The input Python program will have simple variables (no lists or arrays) with simple assigment, function definitions, if/else control statement, function calls, following Python syntax and evaluation semantics. For this homework consider functions without parameters, but be prepared for function parameters in a future homework.

2 Input and output

The input is a regular source code Python file (e.g. example.py).

Input example 1

```
# File:    t1.py
# Content: test source code

# simple function f
def f():
  x=1
  y=2
  if(x==1):
    t= x+y
  return t


x=1+3*100/2+f()
y=x*100+10*3.1416
print("x=",x)
print("y=",y)
```

Output example

```
-bash-4.2$ mypython t1.py
x= 151
y= 15131.416
```

3 Program input and output specification, main call

The main program should be called **mypython**. The output should be written to the console without any extra characters, but the TAs can redirect it to create some output file. Call syntax at the OS prompt:

```
mypython <file.py>
```

4 Requirements

- Your interpreter should behave in the same way as Python.
- Only one statement per line. Treat line number as instruction address.
- Only integer numbers as input (no decimals!).
- No nesting of functions calls: only 1 level. In the next edition there may be function calls nested up to 3 levels.
- No nested if() statements. In a future homework there may be control statements nested up to 3 levels.
- One variable assigment per line with full expression on that line. Do not break an arithmetic expression into multiple lines as it will complicate your parser and will make testing more difficult.
- Variable and function names using only letters and digits
- Arithmetic expression combining integers, variable names and function calls. Spaces may separate tokens.
- Local and global variables. Functions required to update global variables or create local variables if necessary
- Arithmetic operators: $+ - */$. No parenthesis, other operators or function calls inside expression (yet).
- Variable assignment with mutation. You can assume the expression is integer.
- basic if/else with one comparison (unlikely we would use and/or in future homework)
- functions return only one value (avoid tuples)
- Default Python indentation: 3 spaces
- Support comments with #
- No Python data structures (lists, arrays).
- No OOP constructs like class or method calls.
- No advanced features like I/O, threads, multiple files
- Correctness is the most important requirement: TEST your program with many expressions. Your program should not crash or produce exceptions.

5 Evaluation: Python program interpretation requirements

- Use the Python 3.* interpreter as reference. Test your program comparing with the Python interpreter.
- Allocate variables statically (easier) or dynamically (harder). Deallocate them at the end of the program (easier) or with scoping (harder).
- Output with **print()** statement, following default Python format. Notice Python 2 allows print without parenthesis, but Python 3 does not. Avoid including extra output, if not necessary.
- Catch errors at runtime by default (dynamically). You should identify the error in a specific manner when feasible instead of just displaying an error message.
- Show all the output you want in a trace file, similar to log files. This is essential to debug your program at runtime.
- Optional: Catch errors at parse time, even if it means making multiple passes. Do not worry about minimizing number of passes.

6 Programming requirements

- Must work on our Linux server
- Interpreter must be programmed in GNU C++. Using other C++ compilers is feasible, but discouraged since it is difficult to debug source code. TAs will not test your programs with other C++ compilers (please do not insist).
- Modifying the existing Python open source code is not allowed because it defeats the purpose of learning the basics of programming a lanaguage interpreter. Also, includes many more features than those required in this homework. Finally, our language specification simplifies building the interpreter.
- You can use STL or any c++ libaries or you can develop your own C++ classes. You are required to disclose any code you downloaded/copied.
- You can develop your own scanner/parser or you can use lex, yacc, flex, bison, and so on.
- Create a README file with instructions to compile. Makefile encouraged.
- Your program must compile/run from the command line. There must not be any dependency with your IDE.
- Your mypython program should work in interactive mode or reading an input source code file, like the Python interpreter.