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>
```

Assumptions:

- Your interpreter should behave in the same way as Python.
- Only integer numbers as input (no decimals!).
- Function calls nested up to 3 levels
- Control statements nested up to 3 levels
- do not break an arithmetic expression into multiple lines as it will complicate your parser and will make testing more difficult.

4 Syntax Requirements

- Arithmetic expression combining integers and variable names. Spaces may separate tokens.
- Functions required te create local variables if necessary
- Arithmetic operators: $+ - */$. No parenthesis (yet).
- Local and global variables
- Variable assignment with mutation
- basic if/else with one comparison
- functions return only one value
- Default indentation: 2-3 spaces.
- Support comments with #
- No data strutures (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.
- Breaking a number into a list of nodes. Each node will store the number of digits specified in the parameters. Notice it is acceptable to "align" digits after reading the entire number so that that the rightmost node (end) has all the digits.

5 Evaluation: Python program interpretation requirements

- Use the Python interpreter as reference.
- Output with **print** statement, following default Python format.
- Catch errors at runtime by default (dynamically)
- Optional: Catch errors at parse time, even if it means making multiple 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).
- You can use STL or develop your own C++ classes.
- 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.