COSC4315: A Basic Interpreter of Python Programs
Functional Code

1 Introduction

You will create a Python interpreter written in C++ that can evaluate variable assignments with arithmetic expressions computed with functions. Functions include the primitive functions to define lambda calculus. Functions can be recursive. You do not have to consider infinite integers, strings or real numbers.

The input Python program will have variable assigment, function definitions, if/else control statement, function calls, following Python syntax and evaluation semantics.

2 Input and output

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

Input example 1

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

# next
def s(x):
 y= x+1
 return y

# predecessor(), pred()
def p(x):
  if (x>0):
    return x-1
  else:
    return 0

def recadd(x,y):
 if y==0:
   return x
 else:
   return add(1,recadd(x,p(y)))


def add(x,y):
  z=x+y
  return z

def multiply(x,y):
  z=x*y
  return z


def multefficient(x,y):
  if x!=0:
    if y!=0:
      z=x*y
      return z
  return 0

def dup(x):
  z=2*x
  return z


def divide(a,b):
  if b!=0:
    c=a/b
    return c
  else:
    return -1

def f(n):
 if n==1:
   p=1
 else:
   p= f(n-1)*n
 return p

# sum 1..n
def sumnat(n):
 if n==0:
   s=1
 else:
   s= n + sumnat(n-1)
 return s

x1=add(multiply(1,3),multiply(4,6))
y1=1
y2=succ(0)
x2=multiply(add(y1,y1),add(2,2))
x3=dup(x1)
x4 =add(x1,x2)
xyz= add(multiply(2,4),6)
y=divide(5,2)
print("x1=",x1)
print("x2=",x2)
print("x3=",x3)
print("x4=",x4)
print("y1=",y1)
print("y=",y)

# recursive
n=10
s=f(n)
print("factorial(",n,")=",s);
s=sumnat(10)
print("sum naturals(",n,")=",s)
```

To see the output simply run these Python statements.

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. Other than extra spaces, your output should match the installed Python interpreter. Call syntax at the OS prompt to compile and run:

```
g++ -std=c++11 *.cpp -o mypython
./mypython <file.py>
```

4 Requirements

- Your interpreter should work in the same way as the Python interpreter. That is, test your interpreter comparing results.
- No system calls (i.e. calling Python interpreter itself via shell tricks).
- Only one statement per line.
- Only integer numbers as input (no decimals!). For the result of division, return floor(a/b). Since in C++ any fractional part of the answer is discarded , making the result an integer if both operands are integers(the division return an real in python3).
- nested if/else statements. Up to 2 nested levels. else required. elif not required. if statement can be recursively used in if statement and else statement.
- 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 (no underscore), like textbook.
- Local and global variables. Functions required to update global variables or create local variables if necessary. And local variables can have the same name as global variables (consider scope).
- Operations: Addition, subtraction, multiplication, division. Notice division may return a real number, but you have to truncate it and return an integer.
- Warning: Variable assignment with mutation.
- Data types: You can assume the expression has integers.
- basic if/else with one comparison
- functions return only one value (avoid lists/tuples of multiple values)
- Support comments with #. The input script can have comments.
- Not required:

  while/for loops
  Lists
  Python data structures (lists, arrays, objects).
  OOP constructs like class or method calls.
  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.
- Choose data structure to store instructions and data structure to store variables. In your documentation explain them.
- 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

- Program to be submitted in 2 phases. Phase 1: working on simple terst cases PASS/FAIL. Phase 2: all test cases, graded.
- Must work on Linux cloud 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 like Visual C++ (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 or C++ libaries or you can develop your own C++ classes. You are required to disclose any code you download/copy.
- 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.