Compilers — Project Part 1
P_0 interpreter

Due: Thursday, 26 Sep 2013, 13:30

For this project, work in groups of 2 or 3.

1 P_0

Over this semester, you will build a compiler for a subset of Python. We start this project with a smaller subset of Python that we'll call P_0. P_0 consists of integers, variables, arithmetic expressions, assignment statements, **`print`** statements, and calls to the `input` function. Read the sections of the Python Reference Manual (`http://docs.python.org/reference/`) that apply to P_0:

- Integers are *plain integers* (i.e., 32-bit integers) as defined in Section 3.2. You do not need to implement integer overflow.
- Integer literals are as defined in Section 5.2.2 and 2.4.4. You do not need to implement long integer literals (because they are not 32 bit).
- The arithmetic operators are as defined in Sections 5.5–5.8 These include the usual unary and binary arithmetic and bitwise operators.
- Evaluation order is left-to-right (Section 5.14)
- The `input` function is in Section 2.1. The `input` function should read only integer constants, and return an integer.
- The **`print`** statement is as defined in Section 6.6, but accepts only a single integer expression as its argument, not a tuple. The extended "**`print`** chevron" statement should not be supported.
- Assignment statements as as defined in Section 6.2, but can assign to only one variable, not to a tuple.

Note that there are no control-flow statements or booleans in P_0. All values should be plain integers.

2 Interpreter

Create a Python script named `interp.py` that takes the name of a file as a command-line argument. The file should contain the text of a P_0 program and interprets that program, printing the output of the program to the standard output (stdout). Your interpreter should be a recursive function over the Python ASTs from the `compiler.ast` module and patterned after the `num_nodes` function in the lecture notes. You are not allowed to use Python's built-in `eval` function. For example, given a file `test1.py` with the following contents:

```
x = - input()
print x + input()
```

running `python interp.py test1.py` should read two integers (e.g., 99 and 55, below) and subtract the first from the second.

```
$ python interp.py test1.py
99
55
-44
```

If a program is not a legal P_0 program, your interpreter should print an error message and exit gracefully. It should not just crash.

3 Testing

When developing your interpreter, you should use test-driven development. Write test cases—that is, P_0 input files—that exercise each feature of your interpreter. Writing thorough tests before you write the interpreter itself will help ensure that you cover all the cases correctly. Be sure to include both legal programs and illegal programs. Since P_0 is a subset of standard Python, for legal programs you can compare your output against the standard Python interpreter.

4 Submission

When submitting the assignment, include the names of all members of the pair in each file of your submission. Only one member of the pair needs to submit the code. If both submit, the later submission will be graded. Submit both your source code and test cases. Code that does not parse or that fails because it calls undefined functions or tries to read undefined variables will be given a 0.

Both your code and tests will be graded. Be sure they are readable and well-documented.