Project #3

Introduction to Bison: McCabe Cyclomatic Complexity

CpSc 8270: Language Translation

Computer Science Division, Clemson University

Brian Malloy, PhD

September 27, 2017

Due Date:

In order to receive credit for this assignment, your solution must meet the requirements specified in this document and be submitted, using the `handin` facility, by 8 AM, Friday, October 6^{th}, 2017. The handin close date is set at three days after the due date. If you submit after the due date but before the handin close date there will be a ten point deduction. No submissions will be accepted after the handin close date and no submissions will be accepted by email.

How to Submit:

To submit your solution, copy the README file to the project directory and fill in your information. Then do make clean on the project directory, compress the directory using tar or zip, and then submit your compressed directory using the handin command.

Project Specification:

The purpose of this project is to help you to become familiar with Python 2.7.2, and with Bison, a parser generator. All input must be read by the flex scanner that has been provided for you, and all cyclomatic complexity computation must be computed in the provided Python 2.7.2 parser, with possible modifications to the flex scanner. Thus, an added benefit of this project is that it will provide a gentle introduction to the Python parser that you will be using for the rest of the semester.

The project entails computing cyclomatic complexity for each function in a Python program; include Python constructs that are back ported from 3x into 2.7.2. Print the complexity measure as a three tuple where the items in the tuple are: (1) the line number of the beginning of the function; (2) the name of the function, quoted to indicate that it's a string; and (3) the cyclomatic complexity of the function.

Your oracle for the project is the mccabe *flake* module in Python, and your answers should exactly match those provided by the flake module.

You can install the flake mccabe module using one of the following commands:

```
$ pip install mccabe
$ pip install --user mccabe
```

And you can execute the module on a Python program called main.py as follows:

```
$ python -m mccabe main.py
$ python -m mccabe --min 5 main.py
```

Not All Equal:

Cyclomatic complexity tools will typically report different results and sometimes these results are drastically different. To compare complexity results you might consider installing radon and executing it on some python test cases:

```
pip install radon
radon cc -s main.py
```

Methodology:

Your approach to complexity computation must entail using the scanner and parser for Python 2.7.2 that has been placed in the course repository. you cannot use any other tool, such as grep or Python itself to compute complexity. To accomplish the computation, you should study the productions in parse.y and try to find the ones that will be involved in McCabe complexity computation.

Testing: An advantage in using the Python parser that we have provided is that it will accept all valid Python programs, which will make it easier for you to write test cases: you can validate them against any Python 2.7.2 parser and compare the results with your parser. Also, by matching your output to the output of the flake module and using the provided test script, test.py, it will also be easy for you to write test cases.

Two test scripts have been provided in the parser directory of the project repository:

- alltest.py – if you run this test script it will execute the test cases provided by the Python developers, with 1537 test cases Passed, and 3 test cases Failed.
- test.py – this test script will possibly compile your program, run the test cases in the directory cases, and compare the results from your program with the results in the file with prefix the same as the test case and suffix of .out. For example, a test case, x.py is provided in the cases directory, with corresponding expected output in x.out. You must augment the test case in cases with at least 3 additional test cases that adequately test your complexity tool.

List of Tasks:

1. Modify the scanner and parser provided in the project 3 directory to build a tool to compute McCabe's complexity. The metrics result should exactly match those provided by the flake8 mccabe module.
2. Add test cases to the directory cases to adequately test your tool.

Web Pages:

```
The mccabe module for python from flake8:
https://pypi.python.org/pypi/mccabe

The radon module:
https://radon.readthedocs.io/en/latest/commandline.html
https://radon.readthedocs.io/en/latest/intro.html#
```