Final Project

Structure of Programming Languages Fall 2025

December 9th, 2025

Caroline Shantery

https://github.com/cshantery/struct-prog-lang25

The final project requested that we do something to the code we have worked on this semester to demonstrate some mastery of skills. I decided to finish the implementation for the for loop. I started by writing test cases and the test suite for a basic for loop, a for loop with a break, a for loop with a continue, and a nested for loop. Then I ran those tests an watched them fail. From there a started working on the implementation in parser.py and evaluator.py, the implementation works by running init once to initialize the I = 0 statement, then check the condition, then execute the body of the loop, then increment and finally check the condition again.

As discussed in class programming languages are often considered “black boxes” before this class my true understanding of how a programming language works was non existent beyond what I could assume. Each class when we added a feature or built new tests I further understood what was happening behind the scenes when I run a program which was unbelievably helpful in learning to code in general, when learning to implement design patterns and data structures, having an understanding of what the language itself is made up of, how it might be working, provided clarity in what I was attempting to accomplish and learn.

I landed on adding the for loop because I noted that we had included it as a key word but had never implemented it, adding a feature that I use so frequently in my every day life was interesting and I think makes you appreciate the language and features you use mindlessly a little

further. The intended user is myself, I plan to continue to dissect the code written this semester and further understand the inner working of a programming language. I may even attempt to use is at as a guide to implement my own language. Regardless I do plan to use it as a tool to help me chip away at the black box that is computer science. This class took a big chunk out of that box for me, before this class I felt overwhelmed trying to learn about operating systems, and design patterns, and data base design, without having any true understanding of what is happening in a programming language itself.

The interpreter is built on three main aspects, the tokenizer that performs lexical analysis, scanning the source code with regular expression sin order to transform it into organized tokens. The parser, parses the tokens and builds them into an abstract syntax tree, representing the grammar we implemented. The evaluator then will execute the program traversing the AST, and also manages the state through the environment dictionary to handle assignments, function scoping, control flow, and producing the programs output.

I worked in small increments the last week in between studying for finals and classes. I worked in small pieces and continued running the tests I wrote until I found success. If I had more time I would have enjoyed adding more features or creating a more intricate feature myself.

The tests I wrote were integration tests, one example is as follows:

```
print "testing basic for loop...";
sum = 0;
for( i = 0; i < 5; i = i + 1){
   sum = sum + i;
}
assert sum == 10;
```

```
Final Project Feature: For Loop
testing basic for loop...
testing loop with break statment...
testing for loop with continue...
testing nested for loop
for loop tests completed
```

I think the project was a success, It furthered my understanding of the work we did this semester. I also was able to confidently display my knowledge of the material we covered in class and reflect on what I learned and what I would like to continue to learn.

Screen Shots of implementation:

```python
if ast["tag"] == "for":
    _, status = evaluate(ast["init"], environment)
    if status == "exit": return _, "exit"

    while True:
        condition, status = evaluate(ast["condition"], environment)
        if status == "exit": return condition, "exit"

        # break if condition is false
        if not is_truthy(condition):
            break

        # run body
        val, status = evaluate(ast["body"], environment)

        # handle statments
        if status == "return" or status == "exit": return val, status
        if status == "break": break


        # do incriment
        _, status = evaluate(ast["increment"], environment)
        if status == "exit": return _, "exit"

    return None, None
```

```python
def parse_for_statement(tokens):
    for_statement = "for" "(" expression ";" expression ";" expression ")" statement_list
    """

    assert tokens[0]["tag"] == "for"
    tokens = tokens[1:]
    assert tokens[0]["tag"] == "(", f"Expected '(' after for, got{tokens[0]}"
    tokens = tokens[1:]

    #initialize
    init, tokens = parse_expression(tokens)
    assert tokens[0]["tag"] == ";", f"Expected ';' after init, got {tokens[0]}"
    tokens = tokens[1:]

    #condition
    condition, tokens = parse_expression(tokens)
    assert tokens[0]["tag"] == ";", f"Expected ';' after condition, got {tokens[0]}"
    tokens = tokens[1:]

    #incriment
    increment, tokens = parse_expression(tokens)
    assert tokens[0]["tag"] == ")", f"Expected ')' after increment, got {tokens[0]}"
    tokens = tokens[1:]

    #body
    body, tokens = parse_statement_list(tokens)

    return {
        "tag": "for",
        "init": init,
        "condition": condition,
        "increment": increment,
        "body": body
```