Lab 1 – Making Tokens

Ryan Munger
Ryan.Munger1@marist.edu

February 13, 2025

1 Crafting a Compiler

1.1 Exercise 1.11 - MOSS

Measure of Software Similarity (MOSS) is a plagiarism/code similarity detection tool. Most plagiarism checkers function (like Turnitin) can identify offenders by performing a simple check for similar/matching strings in a document. As a computer science student, I have heard time and time again: "bro just change the variable names the professor won't know." **They will, in fact know.** Aside from visual inspection (did you really think you could slip that past the recipient of a doctorate?), we can detect such cheating with MOSS. Instead of looking at the actual text, MOSS examines the meaning of the code. MOSS uses tokenization and fingerprinting to detect code similarity. It is whitespace ignorant, position independent, and suppresses noise. Therefore, you cannot trick it by re-ordering functions or blocks. By comparing tokens and distinctive code fingerprints we are able to compare two programs based off of their meaning/approach. I doubt that this could be used for simple classes such as Intro to Programming due to the small size and obvious approach to the assignments.

1.2 Exercise 3.1 - Token Sequence

```
main(){
    const float payment = 384.00;
    float bal;
    int month = 0;
    bal = 15000;
    while (bal > 0) {
        printf("Month: %2d Balance: %10.2f\n", month, bal);
        bal = bal - payment + 0.015 * bal;
        month = month + 1;
    }
}
```

Produced Token Sequence:
<id, main><symbol, OPEN_PAREN><symbol, CLOSE_PAREN><symbol, OPEN_BRACE>

<keyword, CONSTANT><keyword, F_TYPE><id, payment><symbol, ASSIGN_OP><digit, 3><digit, 8><digit, 4><symbol, PERIOD><digit, 0><digit, 0><symbol, SEMICOL>

<keyword, F_TYPE><id, bal><symbol, SEMICOL>

<keyword, int><id, month><symbol, ASSIGN_OP><digit, 0><symbol, SEMICOL>

<id, bal><symbol, ASSIGN_OP><digit, 1><digit, 5><digit, 0><digit, 0><digit, 0><symbol, SEMICOL>

<keyword, while><symbol, OPEN_PAREN><id, bal><symbol, GREATER><digit, 0><symbol, CLOSE_PAREN><symbol, OPEN_BRACE>

<keyword printf><symbol, OPEN_PAREN><string, Month: %2d Balance: %10.2f
n><symbol, COMMA><id, month><symbol, COMMA><id, bal><symbol, CLOSE_PAREN>
<symbol, SEMICOL>

<id, bal><symbol, ASSIGN_OP><id, bal><symbol, SUBTRACT><id, payment><symbol, ADD><digit, 0><symbol, PERIOD><digit, 0><digit, 1><digit, 5><symbol, MULT> <symbol, SEMICOL>

<id, month><symbol, ASSIGN_OP><id, month><symbol, ADD><digit, 1><symbol, SEMICOL>

<symbol, close_block>
<symbol, close_block>

Well... that's that. Phew. I didn't break up the chars in the charlist (string) like we do in our compiler to save some space (and effort!). Some tokens require additional information: identifiers, digits, and strings. I would also add the position in which they were found.

Why is the lexer mean? *Apparently generating a token of appreciation for the programmer's work caused issues down the line...*

---

2 Dragon Book

2.1 Exercise 1.1.4 - C as a Target Language

A compiler that translates a high-level language into another high-level language is called a source-to-source translator. What advantages are there to using C as a target language for a compiler?

If I were to utilize a source-source translator, I would definitely choose one that has C as its target language. C excels in portability (who needs a JVM), support, and efficiency. It would be difficult to find a system or environment that does not support C. Additionally, with all of C's libraries and amazing compiler optimizations, translating your code to C could possibly make it more efficient. I personally use a lot of languages that are interoperable with C and can call C functions directly such as Go, Rust, and Zig. Zig can actually compile to C as an intermediate language. Way cooler than the typescript to javascript transpilation (because I said so).

2.2 Exercise 1.6.1 - Variables in Block-Structured Code

```
1  int w , x , y , z ;
2  int i = 4; int j = 5;
3  {   int j = 7;
4      i = 6;
5      w = i + j;
6  }
7  x = i + j;
8  {   int i = 8;
9      y = i + j;
10 }
11 z = i + j;
```

Indicate the values assigned to w, x, y, and z.

Line 1: w, x, y, z initialized. No values.
Line 2: i is 4 and j is 5.
Line 3: j is 7. (In this block)
Line 4: i is 6. (Globally)
Line 5: w is (i + j), (6 + 7), 13.
Line 6: :(
Line 7: x is (i + j), (6 + 5), 11.
Line 8: i is 8 (In this block).
Line 9: y is (i + j), (8 + 5), 13.
Line 10: :(
Line 11: z is (i + j), (6 + 5), 11.

Final: w = 13, x = 11, y = 13, z = 11.

I am going to use my brain as the target language for my next compiler and run some programs VERY locally. I'll keep you posted.
