Lab One - Making Tokens with your Lexer

Murray Coueslant
murray.coueslant1@marist.edu

September 6, 2019

1 The Dragon Book

1.1 Problem 1.1.4

1.1.1 Problem

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?

1.1.2 Solution

Besides the fact that the C language is one of the most popular languages on the entire planet, it also has a massive amount of great machine code generating compilers already in existence for the language. In addition to this, it is a relatively simple language with a total of six token types. This means that translating into C should in theory be easier since there are fewer options to choose from when creating C code from other code.

An advantage of 'cross-compiling' into C is the vast number of options for a machine language compiler for the language. The compilers which have already been created for C are full of optimisations, and therefore it may be more beneficial to write a compiler which translates to C and then compile from C to ML using another compiler (much like the Typescript -> Javascript translation).

1.2 Problem 1.6.1

1.2.1 Problem

For the block-structured C code of Fig. 1.13(a), indicate the values assigned to w, x, y, and z.

```
 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;
```

1.2.2 Solution

The above C code has three blocks, one at the top level and two on an equivalent level below it. The variables w, x, y, and z are declared in the top level scope. Alongside them, the variables i and j are also declared and given values 4 and 5. So by the time line 2 has finished executing, the variables have values:

- w - null
- x - null
- y - null
- z - null
- i - 4
- j - 5

Then we reach line 3, which includes a redefinition of the variable j, in that block's scope. The instance of j from the top level block retains its value of 5; the new instance receives a value of 7. We then have a reassignment of the variable i, which will now carry a value of 6. We then see the first assignment of w, it is given a value of i + j. In this case, it is given the value of 6 + 7, due to the redefinition and reassignment in the prior two lines. Following the end of the block on line 6, the variables have the following values:

- w - 13
- x - null
- y - null
- z - null
- i - 6
- j (top level) - 5
- j (block scope) - 7

The moment we leave that block and move onto line 7, the block scoped instance of j disappears. Line 7 is an assignment to x with the value i + j, which in this case is 6 + 5. After line 7, the variables have the following values:

- w - 13

- x - 11
- y - null
- z - null
- i - 6
- j - 5

We then enter the second to last part of the program, another block scope. In this block, i is redefined to have a value of 8. This creates a new instance of i in the block's scope. y is then assigned the value i + j. Which in this case is 8 + 5. The variables now have values:

- w - 13
- x - 11
- y - 13
- z - null
- i (top level) - 6
- i (block scope) - 8
- j - 5

The final line of the program is an assignment to z, with the top level values of i and j. This would give z the value 6 + 5, making it the same as x. The variables now look like:

- w - 13
- x - 11
- y - 13
- z - 11
- i - 6
- j - 5

2 Crafting A Compiler

2.1 Problem 1.11

2.1.1 Problem

The Measure Of Software Similarity (MOSS) tool can detect similarity of programs written in a variety of modern programming languages. Its main application has been in detecting similarity of programs submitted in computer science classes, where such similarity may indicate plagiarism (students, beware!). In theory, detecting equivalence of two programs is undecidable, but MOSS does a very good job of finding similarity in spite of that limitation. Investigate the techniques MOSS uses to find similarity. How does MOSS differ from other approaches for detecting possible plagiarism?

2.1.2 Solution

MOSS uses many different techniques to detect similarities in documents. Techniques such as hashing and winnowing which are described in detail in `http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf` are used to detect and provide a 'similarity' score for files. Obviously, the software cannot know whether or not plagiarism has actually taken place, but it can point you towards where to investigate.

MOSS is interesting because it does not seem to only use a tokenization approach (like a lexer), whereas other source code similarity detection programs do. MOSS uses a much more robust method which involves using substrings and position-agnostic hashes to check for the similarity between two bodies of text. Because of this, MOSS is able to provide a much more reliable similarity score which is a lot more resistant to small changes in positioning or order in texts. Tokenization takes place, but the MOSS software has the ability to fingerprint a document as well, which is an added bonus when checking collections of documents that may not all be source code.

2.2 Problem 3.1

2.2.1 Problem

Assume the following text is presented to a C scanner:

```
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;
    }
}
```

What token sequence is produced? For which tokens must extra information be returned in addition to the token code?

2.2.2 Solution

The following token sequence is produced when a C scanner is fed the above program:

1. Identifier - main
2. Special Symbol - left parenthesis (
3. Special Symbol - right parenthesis)
4. Special Symbol - left brace
5. Keyword - const
6. Keyword - float
7. Identifier - payment
8. Operator - assignment
9. Constant - 384.00

10. Keyword - float
11. Identifier - bal
12. Keyword - int
13. Identifier - month
14. Operator - assignment
15. Constant - 0
16. Identifier - bal
17. Operator - assignment
18. Constant 15000
19. Keyword - while
20. Special Symbol - left parenthesis (
21. Identifier - bal
22. Operator - greater than
23. Constant - 0
24. Special Symbol - right parenthesis)
25. Special Symbol - left brace
26. Keyword - printf
27. Special Symbol - left parenthesis (
28. String - `"Month: %2d Balance: %10.2f\n"`
29. Identifier - month
30. Identifier - bal
31. Special Symbol - right parenthesis)
32. Identifier - bal
33. Operator - assignment
34. Identifier - bal
35. Operator - subtraction
36. Identifier - payment
37. Operator - addition
38. Constant - 0.015
39. Operator - multiplication
40. Identifier - bal
41. Identifier - month
42. Operator - assignment
43. Identifier - month
44. Operator - addition

45. Constant - 1
46. Special Symbol - right brace
47. Special Symbol - right brace

In this case, the tokens which would require extra information along with the token identifier are any "Identifier" tokens, as well as any of the "Constant" tokens.