Scott Hansen
Professor Alan Labouseur

Lab 1

Crafting a compiler

1.11: The Measure Of Software Similarity (MOSS) [SWA03] 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?

Other attempts to detect plagiarism in computer programming classes is usually based on luck. The instructors responsible for the course may notice that two programs output the same errors or that other aspects of a program look very similar. However, as the size of the class increases, or the number of instructors grading the class increases, this can be increasingly tedious work. MOSS, written by Alex Aiken (he has a lecture series available on youtube about compilers), has an algorithm to automatically detect plagiarism in code.

MOSS is a PERL program that can be executed on a professor's computer and allow examination of similarities between programs after a response is received from the MOSS server. It shows the number of tokens and lines matched between files of student submissions and returns a similarity percentage based on this. The percentage similar is itself not sufficient enough to determine plagiarism, but the user of the tool is able to juxtapose programs and examine for him/herself.

References:

- http://theory.stanford.edu/~aiken/publications/papers/sigmod03.pdf
- https://www.quora.com/How-does-MOSS-Measure-Of-Software-Similarity-Stanford-detect-plagiarism
- http://www3.nd.edu/~kwb/nsf-ufe/1110.pdf

3.1: 1. 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?
T_Keyword(“const”), T_Keyword(“Float”), T_Identifier(id, “payment”), T_Operator(=), T_Float(value,384.00),T_Punctuator(;), T_Keyword(float), T_Identifier(id, “bal”), T_Punctuator(;), T_Keyword(int), T_Identifier(id, “bal”), T_Punctuator(;), T_Keyword(int), T_Identifier(id, month), T_Operator(=), T_Integer(0), T_Punctuator(;), T_Identifier(id, “bal”), T_Operator(=), T_Integer(15000), T_Punctuator(;), T_Keyword(while), T_Punctuator(‘(‘), T_Identifier(“bal”), T_Operator(‘>’), T_Integer(0), T_Punctuator(‘{‘), T_Identifier(printf), T_Punctuator(‘(‘), T_StringLiteral(“Month: %2d Balance” %10.2f\n”), T_Punctuator(‘,’), T_Identifier(“month”), T_Punctuator(‘,’), T_Identifier(“bal”), T_Punctuator(‘)’), T_Punctuator(;), T_Identifier(“bal”), T_Operator(=), T_Identifier(“bal-payment”), T_Operator(+), T_Float(val, 0.015), T_Operator(*), T_Identifier(“bal”), T_Punctuator(;), T_Identifier(“month”), T_Operator(=), T_Operator(+), T_Integer(1), T_Punctuator(;), T_Punctuator(‘}’), T_Punctuator(‘}’), T_EOF()

Extra information must be returned for those tokens that have values such as identifiers and their names and types and their values.

Dragon Book
1.1.4: 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 a compiler uses C as a target language, there may be a few advantages. Firstly, the compiled output would be more portable as it would able to be run on any system that supported C instead of just on a specific architecture for specific machine instructions. The compiled output may also be a lot easier to debug because of the increased readability of the output language. Source-to-source compilation may also allow for legacy code to be run on newer systems.

1.6.1: Exercise 1.6.1 : For the block-structured C code of Fig. 1.13(a), indicate the values assigned to w, x, y, and x.

```
int w, x, y, z;
int i = 4; int j = 5;  //global_i->4; global_j->5
{
  int j = 7; //->local_j = 7
  i = 6; -> //->global_i = 6
  w = i + j; //global_w-> (6 + 7) ->13
}
x = i + j; //global_x -> (6 + 5) -> 11
{
  int i = 8; -> local_i -> 8
  y = i + j; -> global_y -> (8 + 5) -> 13
}
z = i + j; // global_z -> (6 + 5) -> 11
```

“w” is 13, “x” is 11, “y” is 13, “z” is 11