CS558 Programming Languages – Fall 2023 – Suggested Study Question Solutions for Lecture 3a

1. (a) Binding of identifiers occurs for `a` at line 01, `g` and `c` at line 02, `a` at line 03, `d` at line 04 (but *not* line 05!), `h` and `e` at line 06, `f` and `b` at line 09, `j` at line 11, and `b` at line 12.

(b) Uses of identifiers occur for `c` at line 03, `f` and `a` at line 04, `d` (twice!) at line 05, `e` and `d` at line 07, `h` at line 08, `a` and `b` at line 10, `g` and `a` at line 12, and `b` and `a` at line 13.

(c)

01: {}
02: {`a` : line 01}
03: {`a` : line 01, `c`: line 02, `g`: line 02, `f`: line 09 }
04: {`a` : line 03, `c`: line 02, `g`: line 02, `f`: line 09 }
05: {`a` : line 03, `c`: line 02, `d`: line 04, `g`: line 02, `f`: line 09 }
06: {`a` : line 03, `c`: line 02, `d`: line 04, `g`: line 02, `f`: line 09 }
07: {`a` : line 03, `c`: line 02, `d`: line 04, `e`: line 06, `g`: line 02, `f`: line 09, `h`: line 06 }
08: {`a` : line 03, `c`: line 02, `d`: line 04, `g`: line 02, `f`: line 09, `h`: line 06 }
09: {`a` : line 01}
10: {`a` : line 01 `b`: line 09, `g`: line 02, `f`: line 09 }
11: {`a` : line 01 `g`: line 02, `f`: line 09 }
12: {`a` : line 01 `g`: line 02, `f`: line 09, `j`: line 11 }
13: {`a` : line 01 `b`: line 12, `g`: line 02, `f`: line 09, `j`: line 11 }
14: {`a` : line 01 `g`: line 02, `f`: line 09, `j`: line 11 }

(d) The free identifiers of `let b = g(a) in b + a` are `a` and `g`. When we ask for the free identifiers "of a function" we exclude the the parameters and the function name itself (assuming it is allowed to be recursive), so the sole free identifier of `f` is `a`. (If we asked for the "free identifiers of the *body* of `f`," i.e. of the expression `a + b`, the answer would be {`a`, `b`}.) The entire expression has no free variables; such expressions are said to be "closed."

2. With static scoping, the uses of `x` in `set_x` and `print_x` always resolve to the global `x`, so we see `1 1 2 2`. With dynamic scoping, the calls to `set_x` and `print_x` from inside `second` see the local `x` instead, so we see `1 1 2 1`.

3. Here's one simple solution. Function `g` uses a variable (`a`) before it is initialized if-and-only-if control ever reaches the third line of the function, and that occurs if-and-only-if `f` returns.

```
void g() {
  int a;
  f();
  int b = a;  // uninitialized use, but do we ever get here?
}
```

A similar line of reasoning (not always quite so trivial!) can be used to show that essentially every interesting property of programs is undecidable *in general.* (Google "Rice's theorem" for a theoretical exposition.) The consequence is that compilers must necessarily approximate their analyses of program behavior.