Compilers Construction 2004

p5 q 7
AM

Solution Notes

[Syllabus: "survey of execution mechanisms"]

(a) This does *lex* → *syn* → *trn* → *cg* resulting in object code which (after linking) runs directly on the target machine

(b) This does *lex* → *syn* → *trn* and writes the machine-code-like byte code to a file to be reloaded (or simply leaves it in memory). At run-time this is interpreted with a loop

```
    for (;;) switch (*pc++)
    {
case opcode1: <code for opcode1>; continue;
case opcode2: <code for opcode2>; continue;
    }
```

(c) At compile time this does *lex* → *syn* and leaves the tree in memory (or via a file as above). At run-time the interpreter does essentially `eval(tree, empty)` where

```
eval(n,env) = n        /* if n is an integer */
eval(x,env) = lookup(x,env)
eval(e+e',env) = eval(e,env) + eval(e',env)
etc.
```

(d) At compile time this does nothing. All the lexing, parsing must be done at run-time each time a statement must be executed. This may be simplest done by re-parsing the phrase starting on the current line, and then interpreting it as in (c).

(e)
Malformed syntax: at compile time in (a),(b),(c) only
undeclared variable: at compile time in (a),(b) only [but note that other answers are possible if justified]
type error: at compile time in (a),(b) only [but note that other answers are possible if justified]
division by zero: at run-time in all cases [except a compiler might notice division by *literal* zero.]