CSE 130: Midterm Solutions

Sean O’Rourke, Fox Harrell

February 26, 2004

1

1 o, 2 p, 3 e, 4 b, 5 g, 6 k, 7 h, 8 c, 9 q.

2

Name equivalence: two variables’ types are (strict) name equivalent if they are declared in the same statement, or if they have the same non-constructed type identifier. Structural equivalence: two variables’ types are structurally equivalent if:

- they are the same primitive type;
- both are array or pointer types whose base types are structurally equivalent; or
- both are record types whose corresponding members are structurally equivalent.

In this example, since both “feet” and “miles” are aliases for “xs:integer”, “feet” and “miles” are structurally equivalent. However, they are not name equivalent. A language using structural equivalence could cause confusion by allowing variables of types “feet” and “miles” to be used interchangeably, even though they have very different meanings.

3

C: Richie; Fortran: Backus; Algol 60: Backus, McCarthy, et al.; CPL: Strachey; C++: Stroustrup; Pascal: Wirth; Simula: Nygaard, Dahl.

4

Algol 60 featured: Lexical scope, block structure, call-by-name, and recursive procedures. It also had a formal syntax, written in BNF.
The parse tree:

- S—NP VP
- NP—N—time
- VP—V PP
- V—flies
- PP—P NP
- P—like
- NP—Det N
- Det—an
- N—arrow

The other interpretation of the sentence uses "flies" as a noun and "like" as a verb. One acceptable solution consists of adding the following rules to the grammar:

- NP ::= adj N
- VP ::= V NP
- N ::= flies
- V ::= like
- adj ::= time

6

e.g.

int x = 1; // decl (1)

void f() {
    int x = 2; // decl (2)
    g();
}

void g() {
    printf("%d ", x);
}
int main() {
  g();
  f();
}

With dynamic scope, variable bindings are determined by looking up the call stack. With lexical scope, variable bindings are determined by looking outward in the lexical environment (program text) in which the use appears. So in the example above, the program prints “1 1” using lexical scope, because x is bound to declaration (1), the only declaration visible from g(). Using dynamic scope, the program prints “1 2”, because when g() is called from f(), declaration (2) of x appears first in g()’s dynamic scope when called from f().