CS 164, Fall 2006 **CS 164: Homework #5** **P. N. Hilfinger**

Due: Monday, 20 November 2006

General instructions about homework. Check out the homework framework with the command:

```
svn checkout svn+ssh://cs164-tb@HOST/_hw/LOGIN
```

where *LOGIN* is your instructional login. If you've already done this, use `svn update` from within your working copy of the homework directory to update with a `hw5` subdirectory. Fill in the skeleton file(s) in that subdirectory and commit it to hand in homework.

1. The Algol 68 language introduced an expression called the *case conformity clause*. Here's one version of it:

`case` I `=` E_0 `in` T_1`:` E_1`;` T_2`:` E_2`; ...;` T_n`:` E_n`; esac`

where the E_i are expressions (i.e., with values), I is an identifier, and the T_i are types. The idea here is that the program first evaluates E_0, and assigns I its value. If the dynamic type of I is T_i (or a subtype of it) for some i, the program evaluates E_i and yields its value as the value of the entire clause. If more than one T_i fits, the program chooses one arbitrarily and evaluates it. The problem is come up with a typing rule for this expression. That is, we want to know what goes above the line in

$$\frac{}{O \vdash \text{case } I = E_0 \text{ in } T_1\text{: } E_1;\, T_2\text{: } E_2;\, \ldots;\, T_n\text{: } E_n;\, \text{esac} : T_0}$$

to make a sound rule. There is no need to know the rest of this language to do this.

2. In Java, the following is legal:

```
String[] Y;
Object[] X;
...
X = Y;
```

That is, an array of T_1 may be assigned to a variable of type array-of-T_2 as long as T_1 is a subtype of T_2. As it turns out, this rule is unsound in the sense that because of it, certain type errors can only be discovered at execution time, requiring a (somewhat) expensive check that slows down some operations. Give an example of how this can happen (by which I mean an actual Java program).

3. I produced the following program using `gcc -S foo.c`.

```
.globl f
    .type   f, @function
f:
    pushl   %ebp
    movl    %esp, %ebp
    subl    $16, %esp
    movl    $0, -4(%ebp)
    movl    $0, -8(%ebp)
    jmp     .L2
.L3:
    movl    -8(%ebp), %eax
    sall    $2, %eax
    addl    8(%ebp), %eax
    movl    (%eax), %eax
    addl    %eax, -4(%ebp)
    incl    -8(%ebp)
.L2:
    movl    -8(%ebp), %eax
    cmpl    12(%ebp), %eax
    jl      .L3
    movl    -4(%ebp), %eax
    leave
    ret
```

Produce a plausible definition (in C) of function `f`, one that might have produced this output. The function does return a value.