

Programming Languages G22.2110 Summer 2007

Midterm

6/21/2007

1. Syntax (20 = 10 + 10 points)

Consider the following EBNF grammar with start symbol $\langle s \rangle$:

```
 $\langle s \rangle ::= "*" \langle s \rangle \mid \langle s \rangle "[" \langle s \rangle "]" \mid \langle id \rangle$   
 $\langle id \rangle ::= "a" \mid "b" \mid \dots \mid "z"$ 
```

1a. (10 points) Show both parse trees for the expression `*p[i]`.

1b. (10 points) Change the grammar so `[...]` has higher precedence than `*`.

2. Functional rewriting (20 = 10 + 10 points)

2a. (10 points) Consider the following Scheme function:

```
(define (a N L) (if (null? L) L (cons (+ N (car L)) (a N (cdr L)))))
```

Show how Scheme evaluates the expression `(cons 2 (a 4 '()))` by rewriting it.

2b. (10 points) Consider the following Scheme function:

```
(define (add-or-mul op x y) ((if (eq? "add" op) + *) x y))
```

Show how Scheme evaluates the expression `(add-or-mul "mul" -2 3)` by rewriting it.

3. Recursion (20 = 5 + 5 + 5 + 5 points)

3a. (5 points) Briefly define “mutual recursion”.

3b. (5 points) What is the difference between `let`, `let*`, and `letrec` in Scheme?

3c. (5 points) Give an example for mutually recursive functions in Scheme.

3d. (5 points) Give an example for mutually recursive types in C.

4. Scoping (20 = 5 + 15 points)
Consider the following Python program:

```
def f(x):
    a = x
    def g():
        def h():
            print a
        h()
    def i(z):
        a = z
        g()
    i("f/i")
f("main/f")
```

- 4a. (5 points) What does this program print, using static scoping?
- 4b. (15 points) Show the frames on the stack when `h` has just been called. For each frame, show the static and dynamic links.
5. Scheme (20 = 6 + 14 points)

- 5a. (6 points) Write a Scheme function “`make-pair`” that takes two arguments, and returns a list that contains these two arguments. For example,

```
(make-pair 2 3) ⇒ (2 3)
(make-pair "hello" "world") ⇒ ("hello" "world")
```

- 5b. (14 points) Write a Scheme function “`zip-lists`” that takes two list arguments, and returns a list of pairs: the pair of the first element of both lists, the pair of the second element of both lists, etc. For example,

```
(zip-lists '(1 2) '(3 4)) ⇒ ((1 3) (2 4))
(zip-lists '(97 98 99) '("a" "b" "c")) ⇒ ((97 "a") (98 "b") (99 "c"))
```

You can assume that the input is always well-formed, i.e., both arguments are lists, and they have the same length.

<http://www.cs.nyu.edu/courses/summer07/G22.2110-001/midterm.pdf>

Total points: 100.