CS 538

Midterm Exam

Wednesday, March 29, 2000

11:00 AM — Noon

Instructions

Answer question #1 and any three others. (If you answer more, only the first four will count.) Point values are as indicated. Please try to make your answers neat and coherent. Remember, if we can't read it, it's wrong. Partial credit will be given, so try to put something down for each question (a blank answer always gets 0 points!).

1. (1 point)
   DrScheme is an interactive program development environment for what programming language?
   (a) Jovial
   (b) Spitbol
   (c) Corc
   (d) Scheme
   (e) Oak

2. (a) (13 points)
   Most programming languages, including C, C++ and Java, support the notion of nested visibility. That is, within a series of nested namescopes, the nearest (innermost) definition of an identifier is always used, even if the identifier is also defined in enclosing scopes.

   What are the advantages of supporting nested visibility in a language definition? Are there any disadvantages? If so, how could such disadvantages be remedied?

   (b) (20 points)
   In implementing nested visibility, the nearest definition of an identifier is always used, even if that definition leads to type errors. Thus if we see an array reference `a[1]`, we use the nearest definition of `a`, even if that definition specifies that `a` is something other than an array.

   What if we modify the nested scoping rule so that the nearest definition that is valid is used. Thus we "skip" declarations that can't be right, and look for a definition that makes sense in the context in which the identifier is used. Are there any drawbacks to this modified definition of nested scoping? Does it ever lead to any ambiguities?

3. (a) (15 points)
   Let `L` be a list containing n sublists, where each sublist has the same length. For example, `L` might be `( (1 2 3) (4 5 6) (7 8 9))`.
   Write a Scheme function `(extract L)` that takes a list `L` as described above (n sublists, each the same length) and returns a list containing n+1 sublists. The first sublist contains all the first elements (the `car` values) of the sublists in `L`. The remaining n lists in the answer are the lists in `L`, with the head element removed from each list.
   For example, `(extract '( (1 2 3) (4 5 6) (7 8 9)))` evaluates to `((1 4 7) (2 3) (5 6) (8 9))`.

   (b) (18 points)
   Let us now define a variant of the Scheme `map` function called `map-n`. A call `(map-n F L)` takes a function `F` of n arguments and a list `L` as described in part (a) (`L` contains n sublists, all the same length). `(map-n F L)` produces a list of length m (where each list in `L` has length m). The first value `map-n` produces is `F` applied to the n head of list values in `L`. The second value `map-n` produces is `F` applied to the n "second in list" values in `L`. The last value `map-n` produces is `F` applied to the n values that are the "last in list" values in `L`.
   For example, `(map-n + ( (1 2 3) (4 5 6) (7 8 9)))` evaluates to `(12 15 18)` (since, `(+ 1 4 7)` evaluates to `12`, `(+ 2 5 8)` evaluates to `15` and `(+ 3 6 9)` evaluates to `18`.

   Write a Scheme function that implements `map-n`. You may use `extract` from part (a) if you wish. Note also that the predefined Scheme function `(apply F L)` calls the function `F` with the contents of list `L` as the function's arguments.

4. (a) (13 points)
   Explain what `call/cc` (also known as `call-with-current-continuation`) does.

   (b) (20 points)
   What does the following Scheme expression compute? Why?

```
(let ( (C '()))
   (display (= 10 (call/cc (lambda (k) (set! C k) 10))))
   (C 20)
)
```

5. (a) (15 points)
   Consider the following two Scheme functions:

```
(define f1 (lambda (x) (let  ((a 10))   Body )))

(define f2 (let  ((a 10)) (lambda (x)   Body )))
```

   Define Scheme code that can be substituted for *Body* so that `f1` and `f2` compute different results.

   (b) (18 points)
   What does the following Scheme expression compute? Why?

```
(let ( (+  (lambda (x y) (+ y 2))))
   (let ( (+  (lambda (x y) (+ y x))))
      (+ 5 (+ 2 1))
   )
)
```