CS 342 — Principles of Programming Languages May Day, 1991

Test

Please write your name here:

Before you begin, please take a moment to look over the entire test so that you can budget your time.

In this test the names "SASL, "CLU", etc. refer to the languages implemented in the textbook's interpreters. The "real" languages will always be referred to by the names "real SASL", "real CLU" etc.

Some global functions and clusters for the CLU and SASL questions are given on the next page.

Answers to essay questions will be judged on their completeness and clarity. However, please do not feel obligated to fill up all the empty space on the page. Conversely, do not feel limited by the space on a page, since you may continue an answer on the back of a page if you wish.

problem	subject	points	score
1	SASL: find the answer	25	
2	SASL vs. Scheme	20	
3	Information Hiding in CLU	25	
4	Program `equal?` test on lists in CLU	10	
5	Program Stacks in CLU	20	
Total		100	

This exam (only) counts for 15% of your final grade.

The following SASL functions are taken from the text and reproduced here for your reference. They are assumed to be in the global environment in all the SASL questions.

```
(set force (lambda (x)
   (if (list? x)  ; apply list? to every component
     (if (force (car x)) (force (cdr x)) '())
     'T)))

(set first-n (lambda (n l)
   (if (null? l) '()
     (if (= n 0) '()
       (cons (car l) (first-n (- n 1) (cdr l)))))))

(set mapcar (lambda (f l)
     (if (null? l) '()
      (cons (f (car l)) (mapcar f (cdr l))))))
```

Assume that the following CLU cluster is in the global environment for all of the following CLU questions.

```
(cluster List
    ; Exports: nil, null?, cons, car, cdr, rplaca, rplacd
    (rep type a d)
    (define nil () (List 0 0 0))
    (define null? (l) (= (type l) 0))
    (define cons (item l) (List 1 item l))
    (define car (l) (a l))
    (define cdr (l) (d l))
    (define rplaca (l a) (set-a l a))
    (define rplacd (l d) (set-d l d))
)
```

1. For each of the following SASL programs, give the result, if any of the *last* expression in the program. If the last expression denotes an infinite list, write down its first four elements, using "..." to denote thunks that are still suspended. If the last expression denotes an infinite loop or an infinite list, say so. (These expressions are to be evaluated in an environment where the functions on the previous page are defined.)

   Please show any work if you want partial credit.

   (a) (10 points)

   ```
   (set bread-line (lambda () (bread-line)))
   (set kgb (lambda (b c a) (if b c a)))
   (set marx 'truth)
   (kgb (= marx 'truth) ((lambda (x) 27) 32) (bread-line))
   ```

   (b) (5 points)

   ```
   (set long-march (cons 1 (cons 2 long-march)))
   (set b (first-n 4 long-march))
   (force b)
   b
   ```

   (c) (10 points)

   ```
   (set factory (lambda (l1 l2)
                  (cons (+ (car l1) (car l2))
                        (factory (cdr l1) (cdr l2)))))
   (set production (cons 0 (mapcar (lambda (x) (+ x 2)) production)))
   (set five-year-plan (factory production (cdr (cdr production))))
   (set c (first-n 4 five-year-plan))
   (force c)
   c
   ```

2. (20 points) Briefly describe the differences in the way expressions are evaluated in SASL as opposed to Scheme. Also give an example to illustrate each difference.

3. (25 points) Briefly define information hiding. Describe how our version of CLU aids information hiding. Also give examples to illustrate your description.

4. (10 points) Using the `List` cluster, write a CLU function `equal?` that takes two lists as arguments. This function should behave as follows: `(equal?  l1 l2)` returns 0 (i.e., false) if and only if `l1` and `l2` have different lengths or the i^{th} element of `l1` is not equal to the i^{th} element of `l2`; otherwise `(equal?  l1 l2)` returns 1 (i.e., true). Use `=` to compare the elements of the list for equality.

5. (20 points) Fill in the missing code in the bodies of the operations in the following CLU cluster definition. Stacks are to be respresented as lists. (The abstraction function for this representation is described as follows. If the list `r` is the result of `(elems s)`, where `s` is a Stack, Then `r` represents the empty stack if `(List$null? r)` is true, otherwise `r` represents the stack whose top element is `(List$car r)`, whose "second from the top" element (if any) is `(List$car (List$cdr r))`, etc.)

```
(cluster Stack
    ; Export: new, push, pop, top, isEmpty
    (rep elems)
    (define new ()
        ; TYPE: -> Stack
        ; EFFECT: return a new empty stack


    )
    (define push (s i)
        ; TYPE: Stack * Int -> ()
        ; MODIFIES: s
        ; EFFECT: make i the top element of s


    )
    (define pop (s)
        ; TYPE: Stack -> ()
        ; REQUIRES: s is not empty
        ; MODIFIES: s
        ; EFFECT: make the top element of s the second-from-the-top element.


     )
    (define top (s)
        ; TYPE: Stack -> Int
        ; REQUIRES: s is not empty
        ; EFFECT: return the top element of s


     )
     (define isEmpty (s)
        ; TYPE: Stack -> Bool
        ; EFFECT: return 1 if s is empty, return 0 otherwise


     )
)
```