

CS2135, A02

Final Exam

Name:

Problem	Points	Score
1	20	
2	25	
3	25	
4	30	
Total		

You have 50 minutes to complete the problems on the following pages. There should be sufficient space provided for your answers. You do not need to show templates, but you may receive partial credit if you do. You also do not need to show test cases or examples of data models, but you may develop them if they will help you write the programs.

Your programs may contain only the following Scheme syntax:

define define-struct cond else lambda local begin set!

and the following primitive operations:

*empty? cons? cons first rest car cdr list map filter length member
number? + - * / = < > <= >= zero? min max
symbol? symbol=? equal?
boolean? **and or not**
printf read*

and the functions introduced by **define-struct**.

You may, of course, use whatever constants are necessary.

1. (20 points) Consider the following program:

```
;; fetch-two-inputs : string (alpha beta → gamma) → gamma
;; prompts user for two inputs and invokes provided function
(define (fetch-two-inputs prompt callback)
  (begin
    (printf prompt)
    (let ([in1 (read)]
          [in2 (read)])
      (let ([alist (list prompt in1 in2)])
        (callback in1 in2))))))

(define (number-match alist command anum)
  (cond [(symbol=? command 'get-larger) (filter (lambda (elt1) (> elt1 anum)) alist)]
        [(symbol=? command 'get-smaller) (filter (lambda (elt2) (< elt2 anum)) alist)]
        [(symbol=? command 'count-equal) (length (filter (lambda (elt3) (= elt3 anum)) alist))])

(define (numbers-app)
  (let ([alist (list 6 0 2 5 1 8 3 1 2)])
    (fetch-two-inputs "Enter a command and a number "
                      (lambda (command num)
                        (number-match alist command num)))))

(numbers-app)
```

(a) (12 points) Write down all of the closures (including their environments) that get created when running this program, assuming that the user inputs `'get-smaller` and 5 as the two inputs. [Do not copy the function bodies in the closures – `(lambda (params) ...)` is enough].

(b) (4 points) What result would the above code yield under static scoping and why (assuming user enters `get-smaller` and 5)?

(c) (4 points) What result would the above code yield under dynamic scoping and why (assuming user enters `get-smaller` and 5)?

2. (25 points) The following code provides the core of a blackjack game program that you want to release on the web (blackjack is a card game: the goal is to get as close as possible to 21 points without going over, where your points are the sum of numbers on your cards). To do this, you would need to (a) convert *prompt-card-script* to *prompt-card-web* (b) move all calls to *prompt-card-script* into script position, replacing calls to *prompt-card-script* with calls to *prompt-card-web*.

```
:: Assume all unprovided functions are primitives, not scripts.
:: prompt-card-script : number → symbol
:: prompts user for whether they want to draw a new card
(define (prompt-card-script curr-points)
  (begin
    (printf "You have ~a points. Do you want another card? " curr-points)
    (read)))
:: blackjack : list[card] → string
:: plays blackjack and returns string with final results
(define (blackjack cards)
  (cond [(> (points cards) 21) "Sorry, you lose "]
        [else (cond [(symbol=? (prompt-card-script (points cards)) 'y)
                      (blackjack (cons (draw-new-card) cards))]
                    [else (check-if-win (points cards))])]))
(blackjack empty)
```

- (a) (10 points) What is the contract on *prompt-card-web*?

- (b) (5 points) Edit *prompt-card-script* to become *prompt-card-web*. You can mark up the code above rather than write out all of the new code.

- (c) (10 points) Rewrite the *blackjack* code so that all calls to *prompt-card-script* are in script position.

3. (30 points) Your company is developing the code for a computer-based voting system. The voting system must allow creating ballots for different districts and tallying votes both within districts and across all districts.

(a) (15 points) Your company has decided to implement candidates as objects. The following interaction shows how these objects should behave:

```
> (define george-boston (make-candidate-obj 'george 'boston))
> (define george-worcester (make-candidate-obj 'george 'worcester))
> (george-boston 'cast-vote)
> (george-worcester 'cast-vote)
> (george-worcester 'cast-vote)
> (george-boston 'local-count)
george has 1 votes in district boston
> (george-boston 'total-count)
george has 3 votes across all districts
> (george-worcester 'local-count)
george has 2 votes in district worcester
> (george-worcester 'total-count)
george has 3 votes across all districts
```

The following code is a partial implementation of the candidate object. Finish the code so that it would produce the interaction shown above. Your finished code should not contain any free variables. You may not use global variables. Mark your edits on the code fragment (do not rewrite the code).

```
(define make-candidate-obj
```

```
(lambda (service)
  (cond [(symbol=? service 'get-name) cand-name]
        [(symbol=? service 'cast-vote)
         (begin
          (set! local-votes (+ 1 local-votes))
          (set! total-votes (+ 1 total-votes)))]
        [(symbol=? service 'local-count)
         (printf "~a has ~s votes in district ~a~n" cand-name local-votes district)]
        [(symbol=? service 'total-count)
         (printf "~a has ~a votes across all districts~n" cand-name total-votes)]))
```

- (b) (15 points) Using the completed *make-candidate-obj*, the company proposes the following code for ballot objects:

```
(define (find-candidate name alocobjs)
  (first (filter (lambda (candidate) (symbol=? name (candidate 'get-name)))
                alocobjs)))
(define make-ballot-obj
  (lambda (district candidate-names)
    (local ([define candidates (map (lambda (name) (make-candidate-obj name district))
                                    candidate-names)])
      (lambda (service)
        (cond [(symbol=? service 'vote)
                (lambda (for-candidate)
                  (let ([candidate (find-candidate for-candidate candidates)])
                    (candidate 'cast-vote)))]
              [(symbol=? service 'candidate-tally)
                (lambda (for-candidate)
                  (let ([candidate (find-candidate for-candidate candidates)])
                    (begin
                     (candidate 'local-count)
                     (candidate 'total-count)))))]))))))
```

This code allows the following interaction in which al has the wrong number of total votes.

```
> (define boston-ballot (make-ballot-obj 'boston (list 'george 'al 'ralph)))
> (define worcester-ballot (make-ballot-obj 'worcester (list 'george 'al 'ralph)))
> ((boston-ballot 'vote) 'george)
> ((boston-ballot 'vote) 'al)
> ((worcester-ballot 'vote) 'ralph)
> ((boston-ballot 'candidate-tally) 'al)
al has 1 votes in district boston
al has 3 votes across all districts
```

- i. (5 points) Explain why the code produces this unexpected (and incorrect) answer.
- ii. (10 points) In the space below, rewrite *make-candidate-obj* up to the **lambda** (*service*) ... so that the ballot program could behave correctly. You may change the contracts on *make-candidate-obj* and/or *make-ballot-obj*; in other words, your code must allow the spirit of the existing interactions, but may require the company to change how they call these two functions. You do **not** need to show the edits *make-ballot-obj* as part of your answer.

4. (25 points) We've decided to add classes and objects to Curly. Assume that classes can have any number of services, but that services do not require additional arguments/parameters. The following example shows the extended syntax and how we could use it to return a circle object with radius 10.

```
{with x=5
  {do {with circle={class radius {service area {pi * {radius * radius}}}
      {service perimeter {pi * {radius * 2}}}}}
    {do {with y=10 {do {object {var circle} {var y}}}}}}}}
```

- (a) (15 points) Here is the existing data definition for Curly expressions. Extend the definition and provide the necessary define-structs to include classes and objects according to the above syntax.

An expr is one of

- a number,
- (*make-var symbol*)
- (*make-proc symbol expr*)
- (*make-plus expr expr*)
- (*make-mult expr expr*)
- (*make-with symbol expr expr*)
- (*make-call expr expr*)

- (b) (5 points) The Curly interpreter with delayed substitution attaches environments to *make-proc* expressions, but not to *make-plus* expressions. Why aren't environments needed on *make-plus* expressions?

- (c) (5 points) Assuming we want our Curly interpreter to use environments and retain static scoping, will either classes and/or objects require environments? Justify your answer.