CS3733: Software Engineering – B00
Final Exam

Question	Poss. Points	Grade
1	15	
2	10	
3	20	
4	15	
5	24	
6	16	
	Total	
Extra Credit	3	

Name:

Answer all questions in the spaces provided on this exam. If you need additional space, get blank paper from the professor. Clearly label all answers on additional sheets with their corresponding question number.

1. (15 points) For each of the following sets of data and operations, indicate whether they would be most appropriately grouped into a class or into a subsystem.

   (a) A filesystem containing *files* and *directories*, with operations for finding paths to files, finding files containing certain data, and moving files between directories.

   (b) Employee records containing *name*, *position*, *salary*, *office*, and *extension*, with operations for assigning offices, computing wages based on hours worked, and asking whether an employee has a particular position.

   (c) Hash tables with the operations of associating data with keys, retrieving data, and running some function on every data associated ith a given key.

2. (10 points) State two advantages of the visitor pattern over the interpreter pattern for implementing software designs.

3. (20 points) In class, we discussed a very simple model of search engines in which the engine searched a set of URLs for desired keywords. This model is too simplistic because real engines don't search the web afresh on each query. Instead, they maintain data on which words appear in which documents and return URLs based on that data (this is why search engines sometimes return stale links). Rather than maintain this information on all words, the engine selects certain words as "interesting", and only matches those words (this prevents searches from considering words like "the", "an", etc). A more realistic data model and operations for search engines would be as follows:

   Data Model:

   - A URL is a string.
   - A word is an identifier.
   - The interesting-words is a set of words.
   - A query consists of a set of words and a list of URLs that satisfy the query (assume all queries are "and"-queries for this problem). The URLs are in decreasing order based on the number of matches they contain for the words in the query.
   - A match-entry indicates how many times a word appears in a document. It consists of an interesting word, a URL, and a number.
   - A match-table is a set of match-entries.
   - A seed-database is a set of URLs (these URLs have been used to create the match-table and can be used to re-create the tables in case the search engine crashes).

   Operations:

   - *create-query : list-of-words* $\rightarrow$ *query*
     Creates a new query from words entered by the user but does not search the engine.
   - *search-on-query : query* $\rightarrow$ *query*
     Creates a new query which updates the URLs from the input query to reflect the words in the input query based on the match table (this supports adding new words to a query and refining the search).
   - *update-query : list-of-words query* $\rightarrow$ *query*
     Adds additional words to an existing (already searched) query and refines the satisfying URLs according to the new words.

   (Questions on next page)

Answer the following questions on this data model and set of operations:

(a) (10 points) State three (plausible) invariants on this design.

(b) (10 points) For each operation, propose either a pre-condition or a post-condition for it (state whether each is a pre- or post-condition).

4. (15 points) While writing a simple adventure game containing characters from fairy tales, you develop the following definitions:

```
(define charI (interface () kiss))

(define character
  (class* object% (charI) ()
    (public (kiss (lambda () this)))
    (sequence (super-init))))

(define Prince
  (class* character () ()
    (sequence (super-init))))

(define Princess
  (class* character () ()
    (sequence (super-init))))

(define Frog
  (class* character () ()
    (override
      (kiss (lambda () (make-object Prince))))
    (sequence (super-init))))
```

Your friend wishes to extend the system in two ways: (1) add a new kind of character (Wizards) and (2) add a new operation *carry*, through which characters can carry a single object through the game. Your friend proposes the following code for these extensions and claims that it will never generate a run-time error.

```
(define charcarryI (interface (charI) carry))

(define Prince/Carry
  (class* Prince (charcarryI) ()
    (private (carrying #f))
    (public (carry (lambda (item) (set! carrying item))))
    (sequence (super-init))))

(define Princess/Carry
  (class* Princess (charcarryI) ()
    (private (carrying #f))
    (public (carry (lambda (item) (set! carrying item))))
    (sequence (super-init))))

(define Frog/Carry
  (class* Frog (charcarryI) ()
    (private (carrying #f))
    (public (carry (lambda (item) (set! carrying item))))
    (sequence (super-init))))

(define Wizard
  (class* character (charcarryI) ()
    (private (carrying #f))
    (public (carry (lambda (item) (set! carrying item))))
    (sequence (super-init))))
```

Is your friend's claim that this code will never yield a run-time error correct? Either explain why it is correct or provide a scheme expression that will yield an error and state what specific error will arise.

Extra Credit (3 points): Your friend's code duplicates a lot of code across the class extensions. How could you have written this code, *without modifying your original code*, to share the duplicated code across the extensions?

5. (24 points) Your company is providing a service to help people stay updated on the scores of their favorite sports teams. Your initial data model says that a game consists of two teams and their scores, that a game-roster is a set of games, and that a watch-set is a of teams that a particular person wants to track.

   Your initial implementation captures each team and its score in a list and uses a structure to group teams into matches, as follows:

```
(define-struct game (team1 team2))

(define sample-roster
  (list (make-game (list 'braves 3) (list 'cubs 5))
        (make-game (list 'yanks 4) (list 'mets 6))
        (make-game (list 'astros 10) (list 'padres 3))))
```

   You write the following code to display scores for games involving teams that a person wants to track:

```
;; display-watches : game-roster list-of-symbol → void
;; displays score of each game in which one of the teams
;; in the game is in the list of teams to watch.
(define display-watches
  (lambda (todays-games watch-teams)
    (cond [(empty? todays-games) void]
          [else
           (let ([game (first todays-games)])
             (if (or (member (first (game-team1 game)) watch-teams)
                     (member (first (game-team2 game)) watch-teams))
                 (display-score (first (game-team1 game))
                                (first (rest (game-team1 game)))
                                (first (game-team2 game))
                                (first (rest (game-team2 game))))
                 void) ;; no else clause for the if expression
             (display-watches (rest todays-games) watch-teams))])))

;; display-score : symbol number symbol number → void
;; prints out the team names and score for a particular game
(define display-score
  (lambda (team1 score1 team2 score2)
    (printf "~a : ~a ~n~n" team1 score1)
    (printf "~a : ~a ~n" team2 score2)))
```

   Users of your service ask to also see statistics about the games (such as batting averages for players in baseball, yards rushed in football, or set points in tennis). You need to extend your software to support these statistics. You have already decided to change the representation of each team in a game to a class that contains the team's name, score, and statistics, with all needed getters.

   (a) (12 points) Which portions of the above code will need to change when you switch to the class-based representation for each team? (draw boxes around the expressions that need to change; your boxes should contain only expressions that need to change).

(b) (12 points) How could you have written the original code so that you would not need to edit it to make this change now? (You may either rewrite the code or describe how the revised code would differ from the original, but be precise!)

6. (16 pts) This class has been discussing component-based software construction, in which programmers package code into components (units, in our terms) with well-defined imports and exports. For this question, assume all component code is distributed in compiled form (aka object code). Thus, programmers using components must treat them as black boxes which they cannot access other than through the exports. In open-source software development, in contrast, programmers provide the full source code for their products, with no expectations that users will treat the code as black boxes. Both approaches have strong proponents in the computing community.

   (a) (8 pts) State two advantages of each model (component-based design and open-source development) from the perspective of a programmer who *uses* the provided software to develop other software.

   (b) (8 pts) State two advantages of each model (component-based design and open-source development) from the perspective of a programmer who *develops* the provided software.