Your Name______________________________

Foundations of Computer Science
Exam 1

Lynn Andrea Stein
Franklin W. Olin College of Engineering

revised 1

Your Name___________________________

This exam is intended as a two hour sit-down examination. However, you may self-administer it at any time between 10am on Tuesday October 5 and 4pm on Friday October 8, 2004. Those who wish to take the exam during Tuesday's class period may do so. Regardless of when you complete the exam, you may not discuss it with anyone until after all exams have been turned in at 4pm on Friday, October 8. (Exams should be turned in to Holly Bennett in OC360 (or thereabouts).

The exam is intended to be completed in a single sitting. You may take more than two hours to complete the examination, but you should not consider this an unlimited-time exam. (Taking four hours would be fine, though presumably unnecessary; taking 20 hours would not. Exercise reasonable judgement.) In particular, anything you can't solve within a reasonable amount of time is not likely to be worth an excessive effort.

This exam is closed book. You are not permitted to use any materials, or to consult with any people, beyond the exam itself or the course instructor. (I will be on campus Tuesday through Thursday and should be available by phone, email, and IM for the duration of the exam interval. Please exercise reasonable discretion and don't call outside of the hours of 8am-10:30pm.)

You may take reasonable breaks during the exam, but you are expected to honor the spirit of the single sitting administration. If possible, avoid mealtimes, conversations, phone calls, IMs, and other interpersonal interactions, though you may get up and walk around, have a snack, etc.

It is perfectly acceptable – even preferable – to hand write your answers in this exam booklet or on blank paper that you provide. If you wish to type your exam, you may use a computer but (a) you must not use resources on the computer other than your word processor (b) you should avoid checking email during the exam, except if that is your means of contacting me , and then only to read my email (c) you should not IM with people other than me.

Whether your exam is typed or hand written, each problem should be clearly identified, separated from other problems, and legible. Any extra pages should be stapled *in order* to the back of this exam and, on the problem page in this booklet, you should write "see attached page (#)." You may also continue solutions on the backs of pages or on additional pages, but these should also be clearly labeled and the exam book should note where the solution can be found.

After you have finished this exam, in the space provided on the final page or on an attached piece of paper, please write out the phrase "I have neither given nor received unauthorized assistance during the completion of this work. I agree not to discuss this exam in any way until after 4pm on October 8." Please sign your name to indicate that you have abided by all rules and conducted yourself according to the Olin College Honor Code. If you cannot write out this phrase and sign your name to it, please explain. [1]

[1] This text courtesy of Professor Sarah Spence.

Your Name____________________________

1. Linear Data Types

For this question, you may use any computer language of your choice provided that it's one that I know or can infer from your code. You may also use any reasonable pseudocode notation.

A. Complete the following definition of the *QUEUE* abstract data type:

make-empty-queue: returns a queue containing no elements.
queue-empty? Q: returns boolean true if queue is empty, false otherwise
enqueue elt Q: returns a queue containing all of the elements in Q as well as elt as the most recently added item
dequeue Q: returns a queue containing ____

____

____

front Q: returns ____

____

B. What is returned by
front enqueue a **dequeue dequeue enqueue** b **enqueue** c **enqueue** d **enqueue** e **enqueue** f **dequeue enqueue** g **enqueue** h **make-empty-queue**
?

____

Draw a picture of the queue on which this **front** operation is performed. (You should not include its history, just its current state.)

C. Complete the following definition of the *STACK* abstract data type. The small roman numerals indicate where additional operations should be supplied:

make-empty-stack: returns a stack containing no elements.
stack-empty? stack: returns boolean true if stack is empty, false otherwise

i. ____

____

____

ii. ____

____

____

iii. ____

____

____

D. Implement the queue abstract data type using two stacks, **smoke** and **mirrors**. In other words, assume that you have two stacks, **smoke** and **mirrors**, (defined according to your abstraction) and use them and their operations to define the five queue operations. You should not need to use any additional machinery.

You may answer this question on the back of the preceding page or on a separate piece of paper, but please clearly label it and indicate here where your answer may be found.

(Reminder: For this question, you may use any computer language of your choice provided that it's one that I know or can infer from your code. You may also use any reasonable pseudocode notation.)

Your Name____________________________

2 Regular Expressions and Finite State Machines

A. Write a regular expression to describe the set of strings over alphabet {*a, b, c*} that contains at least one *a*.

____

B. Write a regular expression to describe the set of strings over alphabet {*a, b, c*} that contains at least one *a* and at least one *b*.

____

C. Give an English description of the language of the following regular expression:

((01) U (10))*

____

____

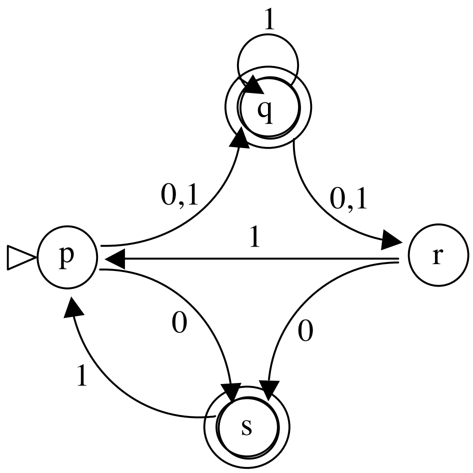
D. Draw a nondeterministic finite state machine corresponding to the following regular expression:

((a U b)(c*)(a U b)*) U a*c

(this question continues on the next page)

Your Name______________________________

E. Consider the following automaton.



In what state(s) will the automaton be after receipt of each character in the string 1010101?

□	
1	
0	
1	
0	
1	
0	
1	

Does this automaton accept this string? Explain.

3 Lisp programming

The following code defines part of the implementation of a *SET* data type in terms of Lisp lists.

```
;; the one and only empty set Ø, also known as {}
(define the-empty-set '())

;; is the set empty?
(define (set-empty? set)
  (null? set))              ;; or (eq? set the-empty-set)

;; selectors to extract elements:
(define set-first car)
(define set-rest cdr)
```

We also need a way to add elements to a set, but let's leave that for part B.

A. Complete the following definition of the set-member? function:

```
;; is elt a member of set?  Check recursively....

(define (set-member? elt set)

  (cond ((set-empty? set)[          ])

        ((eqv? elt (set-first set)) #t)     ;; eqv? is like eq?

        (else [                                              ])))
```

Remember, good code preserves data abstractions.

(this question continues on the next page)

B. Constructing sets poses some interesting questions. For example, we might choose the simplest definition of insertion:

```
;; add an element to a set
(define set-insert cons)
```

Alternately, we might choose a definition that preserves the property that an element cannot be in a set more than once:

```
;; add an element to a set only if it's not already there
(define (set-insert elt set)
  (if (set-member? elt set)       ;; if elt's already a member
      set                         ;; just return set
      (cons elt set)))            ;; else add elt to set
```

Briefly describe the benefits and costs of the member-verifying version of *set-insert*. Indicate under what circumstances you would prefer one version over the other. Also indicate any implications that you see for other set operations. (Your answer should be a few sentences long, but not an essay!)

____

____

____

____

____

____

____

____

____

____

(this question continues on the next page)

C. The following additional definitions complete the *SET* data type implementation.

```
;; set1 union set2 just glues the lists together.
;; Note that it doesn't eliminate duplicates, though.
(define set-union append)

;; set1 intersection set2 produces a set containing
;; only those elements that belong to both sets.
(define (set-intersection set1 set2)
  (filter (lambda (elt) (set-member? elt set2)) set1))

;; set1 minus set2 produces a set containing those elements
;; of set1 not also present in set2.
(define (set-difference set1 set2)
  (filter (lambda (elt) (not (set-member? elt set2))) set1))
```

Complete the definitions of *append* and *filter*

```
;; (append L1 L2) returns a new list that contains the elements
;; of L1 and L2 together.  For example,
;;  (append '(a b) '(c d)) returns (a b c d)   and
;;  (append '(1 (2 3) 4) '((5 6) 7)) returns (1 (2 3) 4 (5 6) 7)

(define (append L1 L2)

  (cond ((null? L1) [          ])

        (else (cons (car L1) (append (cdr L1) L2)))))


;; (filter tst lst) takes
;;    a test procedure that in turn returns a boolean
;;        - see set-intersection or set-difference, above -
;;    and a list.
;; (filter tst lst) returns a list containing those elements
;;    of lst that pass the test

(define (filter tst lst)

  (cond ((null? lst) '())
        ([                        ] (cons (car lst)

                                        (filter tst (cdr lst))))

        (else (filter tst (cdr lst)))))
```

Your Name____________________________

4 Pumping Lemma

The Pumping Lemma says:

For every Regular Language L,

There exists a constant n (generally corresponding to the number of states in L's FSM)

And for every string w in the language L with length > n

The string w can be split into xyz

with $|xy| \leq n$ (there are at most n characters in xy)

and $|y| > 0$ (y isn't the empty string)

so that xy^iz is also in L for all values of i

Using the pumping lemma, prove that the following is *not* a regular language:

The language containing all strings of 2^k 1s, i.e., □ 1, 11, 1111, 11111111, etc., but not 111 or 111111.

(Hint: Assume that the language is regular and that there is some constant n for which the Pumping Lemma holds. Show that this would mean that a string not actually in the language would have to be there.)

(You may use the backs of pages or additional attached pages if necessary. However, indicate here where your answer can be found. Note that a (succinct) correct answer *can* be fit into the space on this page.)

This is the end of the required portion of the exam. The problem on the next page is an optional bonus (extra credit) problem. However, you MUST fill in the honor code declaration on the final page of this examination booklet. Also, please make certain that you have put your name on every page of this exam booklet and any attached pages.

5 Bonus Problem (Extra Credit; Optional):

We can use the set data type of problem 3 to build a power set generator. The following scheme procedure takes a set as argument and returns the set of all possible subsets of that set. (This would be useful, for example, if we were converting a nondeterministic FSM to a deterministic FSM….)

```
1(define (all-possible-subsets set)
2. (cond ((set-empty? set) aps-base-case)
3.       (else
4.        (let ((all-but-first (all-possible-subsets (set-rest set))))
5.          (let ((all-with-first (map (lambda (p)
6.                                          (set-insert (set-first set)
7.                                                      p))
8.                                   all-but-first)))
9.            (set-union all-but-first all-with-first))))))
```

where *map* is

```
10. (define (map proc lst)
11.   (cond ((null? lst) '())
12.         (else (cons (proc (car lst)) (map proc (cdr lst))))))
```

A. What should the value of *aps-base-case* be, i.e., what is the base case of the *all-possible-subsets* recursion?

B. What are the values of *all-but-first* and *all-with-first* after the evaluations of the lets, i.e., when line 9 is about to be evaluated in the invocation of

```
(all-possible-subsets '(a b c))
```

all-but-first: ____

all-with-first: ____

C. What is the order of growth of this procedure (in terms of □ and the size of *set*)? Explain your answer in terms of the scheme code. Use line numbers. (You may answer this question on the back of a page or on a separate sheet, but indicate here where I can find it.)

6 Honor Code Declaration

Please write out and sign the honor code declaration from the instructions on page 2 of this exam in the space below or provide an explanation here as to why you cannot do so.