
This is a 75 minute, CLOSED notes, books, etc. exam.
ASK if anything is not clear.
WORK INDIVIDUALLY.
Strategy: Scan the entire exam first. Work on the easier ones before the harder ones. Don't waste too much time on any one problem. Show all work on the space provided. Write your name on each page. Check to make sure you have 7 pages.

You may not use `set!` in your Racket programs unless otherwise indicated.

Question	Points	Score
1	5	
2	5	
3	5	
4	5	
5	5	
6	10	
7	15	
8	5	
9	10	
10	5	
11	10	
12	10	
13	10	
Total:	100	

1. (5 points) Select **all** of the following true statements about macros.
   A. Text substitution macros are powerful, but may result in inadvertent variable capture.
   B. Syntactic macros work by restructuring abstract syntax trees.
   C. Macros in C are hygienic.
   D. Syntactic macros work at the "preprocessor" stage, before tokenization.
   E. Macros are not used in Racket, since it is easy to restructure lists without relying on macros.

2. (5 points) Select **all** of the following true statements about Louden & Lambert's language design criteria.
   A. Macros in Racket are an example of *extensibility*.
   B. A language with poor *regularity* might either make similar things look different, or make different things look similar.
   C. The design goal of *efficiency* refers only to making the programmer more efficient, not to making more efficient compiled code.
   D. C is a good example of a language that provides strong *security*, since it catches errors like writing past the end of an array.
   E. C may be considered to be an efficient language from the perspective of the compiler, since it produces efficient executables, but less efficient from the perspective of the programmer, since fewer details are handled automatically.

3. (5 points) Select **all** of the following true statements about parsers and tokenizers.
   A. The job of a parser is to take in tokens and produce an abstract syntax tree (AST).
   B. Abstract syntax trees are sometimes called "lexemes".
   C. Tokens may be thought of as the "words" of a language.
   D. Once an abstract syntax tree is produced, it is only useful for evaluation by an interpreter; for efficiency, compilers typically work directly from the stream of tokens.
   E. All languages with support for macros include a "macro expander" piece before tokenization.

4. (5 points) Select **all** of the following true statements about JavaScript.
   A. JavaScript lacks classes, and hence cannot be considered an object-oriented programming language.
   B. Inside of a function that is not part of an object, `this` will refer to the global object, unless it was invoked with the keyword `new`, in which case it refers to a new object being created.
   C. Functions in JavaScript are *closures*; that is, they remember variables from the scope where they were created.
   D. JavaScript is a purely functional programming language.
   E. Every object has a `prototype` field, which is another object; if you add properties to this object, then they are available to all objects created with the same constructor.

5. (5 points) Select **all** of the following true statements about programming contracts.
   A. Programming contracts are useful for establishing *blame* when things go wrong.
   B. If a precondition is not satisfied, then the library writer is at fault for any errors.
   C. Contracts may only be specified on modules.
   D. Specifying a contract for a function is more efficient than specifying one for a module, since it avoids repeated checks on the contract for a recursive call.
   E. In Racket, the `->i` operator is used to specify contracts on functions where the validity of the arguments or result depend on each other.

6. (10 points) Consider the following Racket code:

```
(struct account (balance))

(define new-account (account 0))

(define (balance acc)
  (account-balance acc))

(define (deposit acc amt)
  (account (+ (account-balance acc) amt)))
```

Write a `provide` statement that makes the following guarantees:

- The `balance` function takes in an `account` and returns a number.
- The `deposit` function takes in an `account` and a positive number, and returns an `account`.

7. (15 points) Write a `largest-elem` function in Racket that takes a list of numbers and returns the largest element in the list. (You may **not** use the `max` function). If the list is empty, raise an error.

```
;; Sample usage
(largest-elem '(9 0 42 1 6)) ;; evaluates to 42
(largest-elem '(-2 -8 -865)) ;; evaluates to -2
```

8. (5 points) Consider the following Racket definitions:

```
(define-syntax-rule (if-mac c thn els)
  (if c thn els))

(define (if-fun c thn els)
  (if c thn els))
```

While these two definitions seem similar, there are some important differences. Write a short example that illustrates the difference between `if-fun` and `if-mac`. (Hint: consider the difference between how the arguments to a macro and the arguments to a function are evaluated).

9. (10 points) **In a tail-recursive style**, write a `mult-all` function that takes in a list of numbers and returns the product. (You will need to create a helper function in order to solve this problem).

```
;; Sample usage
(mult-all '(2 3 5)) ;; evaluates to 30
```

10. (5 points) Did you put your name on every page of the exam?

11. (10 points) Consider the following JavaScript program:

```
function Car(make, model, year) {
  this.make = make;
  this.model = model;
  this.year = year;
  this.honk = function() { console.log("honk!"); }
}
Car.prototype.honk = function() { console.log("Meep!"); }

var car1 = new Car("Chevy", "Nova");
var car2 = new Car("Tesla", "Model S", 2014);
var car3 = Car("Ford", "Explorer", 2001); // Forgot to call "new"

car1.honk();
delete car2.honk;
car1.honk();
car2.honk();
car3.honk();
console.log(model);
```

(a) Select **all** of the following true statements.

A. This code throws an error when creating `car1`, since an argument is missing.

B. This code throws an error when trying to create `car3`, since the programmer forgot to call `new`

C. This code throws an error when calling `car3.honk()`, since `honk` is not defined for the object `car3`.

D. This code throws an error when calling `car3.honk()`, since `car3` is not defined.

E. This code throws an error when calling `console.log(model)`, because `model` is not defined.

(b) Comment out any lines of the program that cause an error. Now what is the output of this program?

12. (10 points) Consider the following program:

```
function mystery(n) {
  var curPos = 2, lst = [], i;
  for (i=curPos; i<n; i++) lst[i]=i;
  return function() {
    var x;
    while (lst[curPos] === undefined) {
      if (curPos > n) throw new Error("Nothing left");
      curPos++;
    }
    x = curPos;
    while (x < n) {
      delete lst[x];
      x += curPos;
    }
    return curPos;
  }
}

var next = mystery(25);
for (i=0; i<5; i++) {
  console.log(next());
}
```

What is the output of this program? **NOTE:** After the while loop, the value of `lst` will be

```
[ undefined, undefined, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14,
  15, 16, 17, 18, 19, 20, 21, 22, 23, 24 ]
```

For 2 points extra credit (HARD, NO PARTIAL CREDIT), modify this program to return a pair of functions, where the first function gets the next number, and the second function tests whether there are any numbers left.

13. (10 points) The lambda calculus has the following expressions:

$e ::=$		*Expressions*
	x	Variables
	$(\lambda x.e)$	Functions
	$e\ e$	Function application

From this base, we can define a number of additional constructs, such as booleans and conditional statements:

$$\begin{array}{ll} let & \texttt{true} = (\lambda x.(\lambda y.x)) \\ & \texttt{false} = (\lambda x.(\lambda y.y)) \\ & \texttt{if} = (\lambda cond.(\lambda then.(\lambda else.cond\ then\ else))) \end{array}$$

Showing your work, show the evaluation of the following program:

```
if false false true
```