Your last name ______________________________

Last name of classmates beside you (or "wall"/ "aisle") On my left: ____________ On my right: _______________

CSC 372 Final Exam

Tuesday, May 9 **-or-** Thursday, May 11
2023

READ THIS FIRST

Read this page now but do not turn this page until you are told to do so. Go ahead and fill in the boxes above with your last name and the last names of any classmates sitting beside you.

<u>This is a 110-minute exam</u> with a total of 100 points of regular questions and an extra credit section.

<u>The last five minutes of the exam is a "seatbelts required" period</u>, to avoid distractions for those who are still working. If you finish before the "seatbelts required" period starts, you may turn in your exam and leave. If not, you must stay quietly seated—no "packing up"— until time is up for all.

<u>You are allowed no reference materials whatsoever.</u>

If you have a question, raise your hand. We will come to you. DO NOT leave your seat.

If you have a question that can be safely resolved with a minor assumption, like the name of a function or the order of function arguments, state the assumption and proceed.

Feel free to use abbreviations, like "`otw`" for "`otherwise`".

It's fine to use helper predicates or procedures unless they are specifically prohibited or a specific form for the function or predicate is specified.

<u>Don't make a programming problem hard</u> by assuming that it needs to do more than is specifically mentioned in the write-up or that the solution that comes to mind is "too easy."

<u>If you're stuck on a problem, please ask for a hint.</u> Try to avoid leaving a problem completely blank—that's a sure zero.

<u>It is better to put forth a solution that violates stated restrictions than to leave it blank</u>—a solution with violations may still be worth partial credit.

When told to begin, double-check that your name is at the top of this page, and then **<u>put your initials in the lower right-hand corner of the top side of each sheet, checking to be sure you have all 16 sheets.</u>**

<u>BE SURE to enter your last name on the sign-out log when turning in your completed exam.</u>

Problem 1: (6 points)

Cite three things about programming languages you learned by watching your classmates' video projects. Each of the three should be about a different language and have a bit of depth, as described in the Piazza post that announced this problem.

Problem 2: (6 points)

Without writing any recursive code, write a Haskell function `plusvals lst` of type `[(Char, a)] -> [a]`, that returns a list of the second elements of the tuples whose first element is `'+'`. Examples:

```
> plusvals [('+',5),('-',7),('+',2)]
[5,2]

> plusvals [('a',5),('b',7),('c',2)]
[]

> plusvals [('*','t'),('+','h'),('+','i'),('/','s')]
"hi"
```

My solution uses the `fst` and `snd` functions, which return the first and second elements of 2-tuples, respectively.

Problem 3: (2 points)

Here is the procedure for a Prolog predicate named `printN`:

```
printN(0).
printN(N) :- N > 0, M is N - 1, printN(M), writeln(N).
```

Circle and label one example of each of these four elements: clause, fact, goal, rule.

Problem 4: (1 point)

What's the biggest problem with the following **Prolog** programming problem?

"Write a predicate `maxint(+List)` that returns the largest integer in `List`."

Problem 5: (11 points)

Write a Prolog predicate `wrap_with_each(+Word, +Chars, -Wrapped)` that "wraps" `Word` with each of the characters in `Chars`, instantiating `Wrapped` to each result in turn.

Assume that `Word` is an atom and that `Chars` is a list of one-character atoms.

My solution uses `atom_chars(?Atom, ?CharList)`.

Examples:

```
?- wrap_with_each(ate, [d,s], W).
W = dated ;
W = sates.

?- wrap_with_each(ee, [p,d,s,e,z],W).
W = peep ;
W = deed ;
W = sees ;
W = eeee ;
W = zeez.
```

If called with an empty list of characters, `wrap_with_each` fails:

```
?- wrap_with_each(test, [], W).
false.
```

Problem 6: (6 points)

Write a Prolog predicate `second(?S,?L)` that expresses the relationship that `S` is the second element in the list `L`. `second` must provide all of the following behaviors:

`second` can get the second element from a list, failing if there are less than two elements:

```
?- second(X,[3,4,5]).
X = 4.

?- second(X, [3]).
false.
```

`second` can test whether the second element in a list has a specific value:

```
?- second(a, [t,a,c,k]).
true.

?- second(x, [t,a,c,k]).
false.
```

`second` can establish a constraint for the second element in a list that is more fully specified by subsequent goals.

```
?- second(7,L), L=[X,X].
L = [7, 7],
X = 7.

?- second(7,L), length(L,5), last(L,end).
L = [_1704, 7, _1716, _1722, end].
```

Problem 7: (11 points)

Write a Prolog predicate `prefixes(+List,+Min,-Prefixes)` that instantiates `Prefixes` to a list that contains the prefixes of `List`that are at least `Min` elements long.

Examples:

```
?- prefixes([a,b,c,d,e],2,Ps).
Ps = [[a, b], [a, b, c], [a, b, c, d], [a, b, c, d, e]].

?- prefixes([a,b,c],3,Ps).
Ps = [[a, b, c]].

?- prefixes([a,b,c],4,Ps).
Ps = [].

?- prefixes([],0,Ps).
Ps = [[]].
```

My solution uses `findall`.

Problem 8: (12 points)

This problem is like the pit-crossing example in the slides and `connect.pl` on assignment 6, although greatly simplified.

Write a Prolog predicate `find_combo(+Ints, +Goal, -Combo)` that finds a combination of one or more values from the list `Ints` whose sum is `Goal`. Assume that `Ints` contains only integers.

Examples:

```
?- find_combo([3,1,5,2],10,Combo).
Combo = [3, 5, 2] .

?- find_combo([3,1,5],4,Combo).
Combo = [3, 1] .
```

If there is no combination of values from `Ints` whose sum is `Goal`, `find_combo` fails:

```
?- find_combo([3,1,5],7,Combo).
false.
```

There may be many combinations of values that satisfy a given call to `find_combo`. Only the first satisfying combination is requested in the examples above but it is fine for your solution to produce all possible combinations.

Remember that the predicate `select(?E, ?L, ?R)` will select, one by one, each element in the list `L` and instantiate `R` to the remaining values:

```
?- select(E, [3,1,5], R).
E = 3,
R = [1, 5] ;

E = 1,
R = [3, 5] ;

...
```

8

IMPORTANT

Three Racket programming problems, `vequal?`, `let-vars`, and `store` follow. Each is worth 16 points. We will grade all three problems and count your best two scores of the three.

Problem 9: (16 points)

Write a Racket procedure `vequal?` that is like a variadic `equal?`—it returns `#t` if all of its arguments are equal to each other, as determined by `equal?`.

Examples:

```
> (vequal? 'a 'a 'a)
#t

> (vequal? 3 3 3 4)
#f
```

If `vequal?` is called with less than two values it returns `#t`:

```
> (vequal?)
#t
> (vequal? 7)
#t
```

Note: your solution will surely need to use the `equal?` procedure. Something like `(apply = '(a a a))` might cross your mind but although = is variadic, it only works with numbers.

Problem 10: (16 points)

This problem is similar to `pinfo` on assignment 7. You are to write a Racket procedure `let-vars` that takes a Racket expression and, if it is a `let` or `let*` form, return a list of the variables bound by the `let` (or `let*`). Examples:

```
> (let-vars '(let ([x 3][a 5]) (< x (* a 2))))
'(x a)

> (let-vars '(let* ([args (drop-right args 0)]
                    [num-args (length args)]
                    [either (if (empty? args) 3
                                (add1 num-args))]) 'result))
'(args num-args either)

> (let-vars '(let () 3))
'()
```

If the form is not a `let` or `let*`, `let-vars` returns `#f`:

```
> (let-vars '(lettuce ([i 0]) (add1 i)))
#f
> (let-vars '(define j 3))
#f
```

Assume the forms are valid. You won't see something like `(let-vars '(let a 3 b 4))`

DO NOT worry about `let`s nested inside `let`s; only analyze the top-level expression:

```
> (let-vars '(let ([a 4]) (let ([b 5]) (+ a b))))
'(a)  ; NOTE: does *not* reflect the nested let that binds b!
```

Reminder: You must handle both `let` and `let*`!

You may write your solution below, or on the next page.

(Space for solution for `let-vars`.)

For reference:

```
> (let-vars '(let ([x 3][a 5]) (< x (* a 2))))
'(x a)

> (let-vars '(let* ([args (drop-right args 0)]
                    [num-args (length args)]
                    [either (if (empty? args) 3
                                (add1 num-args))]) 'result))
'(args num-args either)

> (let-vars '(let () 3))
'()
```

Problem 11: (16 points)

For this problem you are to write three simple Racket procedures that store and fetch values associated with names. The examples below show running `store.rkt` and then making calls to the three procedures.

The `store` procedure associates a name with a value. `fetch` prints the value associated with a specified name. Names are assumed to be symbols; values can be of any type. Examples:

```
% rk/i store.rkt
> (store 'x 5)

> (store 'color "yellow")

> (fetch 'color)
yellow

> (fetch 'x)
5
```

A `store` with a name that's already been used, or a `fetch` with an unknown name, produce errors:

```
> (store 'x 10)
Duplicate

> (fetch 'z)
Not found
```

Finally, `names` prints a sorted, comma-separated list of names for values that have been stored:

```
> (names)
color, x
```

<u>There is no persistence between runs</u>. If we reload `store.rkt`, we see that previously created names are gone:

```
% rk/i store.rkt
> (fetch 'x)
Not found

> (names)
No names
```

Important: Note that `(names)` prints `"No names"` if no `store`s have yet been done.

There is space for your solution on the next page.

(Space for solution for `store/fetch/names`.)

For reference:

```
> (names)
No names
> (store 'a "this is a")
> (store 'a 7)
Duplicate
> (fetch 'a)
this is a
> (fetch 'b)
Not found
> (store 'c 10)
> (names)
a, c
```

For `names`, my solution uses `string-join`:

```
> (string-join '("a" "b" "c") ", ")
"a, b, c"
```

Problem 12: (4 points)

Write a Racket **macro** that mimics the postfix form of Java's ++ operator. Recall that we've learned that "the value of `i++` is `i`" (whatever `i` is), and that the increment of `i` is a side-effect.

Here's the ++ macro in operation:

```
> (define i 7)
> (++ i)
7
> i
8
> (define j (++ i))
> j
8
> i
9
```

Half the value of this problem is having `(++ `*`var`*`)` produce the correct value; the other half is having the correct side-effect.

Problem 13: (5 points) (1 point each)

The following questions and problems are related to Racket.

(1) What is it about `(define x 7)` that requires `define` to be a special form?

(2) What's the very-important promise that Scheme and Racket make about tail-recursive calls in procedures?

(3) What built-in Racket procedure produces the type of a value? (In other words, what is the Racket analog for Python's `type(...)` function?)

(4) Write a Racket expression that creates a pair that is not a list. That is, create an `x` such that `(pair? x)` is true but `(list? x)` is false.

(5) In a Racket procedure name like `x->y`, what does `->` typically indicate? Similarly, if a the name of a procedure or special form ends with asterisk, like `x*`, what does that indicate?

Problem 14: (2 points) (1 point each)

The following questions about SNOBOL4 are worth one point each. You may answer as many as you want but the maximum score on this problem is two points.

(1) What's significant about the names `input` and `output`?

(2) If successful, what's the side effect of the following pattern match statement?

```
loop line span(&digits) =
```

(3) Write a statement that concatenates the values of `a` and `b`, and assigns the result to `c`.

(4) If the very first line in a program is `x = x + 5`, what happens?

(5) In the following statement, what does `:(t)` mean?

```
x = gt(x,0) 0    :(t)
```

Problem 15: (2 points) (1 point each)

The following questions about Icon are worth one point each. You may answer as many as you want but the maximum score on this problem is two points.

(1) Aside from one-based indexing in Icon, what's a very significant difference between strings in Icon and Python?

(2) Assuming that + and `||` are of equal precedence and left associative, what value is produced by the following Icon expression? `3 || "4" + 5`

(3) What two modes of computation in SNOBOL4 does Icon's string scanning facility endeavor to unify?

(4) What does `write(read()[10])` do if a line having only five characters is read?

(5) Originally, the Icon project had two separate research focuses. Name either of them.

Extra Credit Section (½ point each unless otherwise noted)

(1) The first paper on Lisp was "Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I". What was especially notable about Part II?

(2) Who is regarded as the creator of Lisp?

(3) Write a Prolog predicate `oddlen(+L)` that succeeds iff the list `L` has a odd number of elements. **Restrictions**: Your implementation may have only one clause and use only the predicates `append` and `length`. It may use the `[E1, E2, ..., EN]` list syntax, but not the `[E1, E2, ... | Tail]` form. (In short, like `append.pl` on assignment 6.)

(4) In the Racket slides we saw some examples of `(time` *`expr`*`)`, to evaluate `expr` and report the time spent (and more). Is `time` a special form? Justify your answer.

(5) What's the basic capability provided by Racket's "named-`let`"?

(6) Speculate: What does the following SNOBOL4 code do?

```
    punch = 'HELLO'
```

(7) Name three programming languages that have been created at The University of Arizona.

(8) Almost 40 years ago the name "lectura" was chosen for the department's instructional timesharing system, a VAX-11/785. Who suggested that name? (Hint: It wasn't `whm`!)

(9) UA professor Murray Sargent III was instrumental in attracting Ralph Griswold to The University of Arizona. In what department was Sargent a professor?

(10) On his first day here, what was notably lacking from Ralph Griswold's office?

(11) What do you think you will most remember from 372? (1 point)

(12) Write a good extra credit question related to the course material and answer it. (1 point)