CSC 372, Spring 2023
Final Exam Solutions
Tuesday, May 9 **-or-** Thursday, May 11

Problem 1: (6 points) (mean: 5.6, median: 6, 3rd quartile: 6)

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.

Videos have a lot of dimensions, and that makes it hard to pick a best one but I do feel comfortable picking an All-Star team, and here it is, in alphabetical order by title:

AV: Higher-Order Predicates
https://youtu.be/qGO1PiDMGLY

foreach Loop in C#: An Internal Look
https://drive.google.com/file/d/1-b5LYYTjwJH-cwkj3dTyZr9sZ6o1bvdo/view?usp=share_link

Function Decorators!
https://drive.google.com/file/d/1C87DfsCj9Wfg5dmqE3rsVAUaX7d8iXn4/view?usp=sharing

Functors & Applicative Functors in Haskell
https://vimeo.com/823991378?share=copy

Iterators, Iterables, and Making an Iterator in Python
https://drive.google.com/file/d/1eUFh6XheneX72ok64-aBuXAKrtCUCmkw/view?usp=share_link

Julia's Multiple Dispatch: Just a fancy name for Method Overloading or more?
https://youtu.be/vNyO0_DIDio

Rust Borrow Checking
https://youtu.be/cvwHPuRXC-Q

The Dog Programming Language: Some Commands and Example Programs.
https://www.youtube.com/watch?v=B_efVRztfvQ

Union Types in Scala
https://youtu.be/dO9NG_lX9m8

We Don't Need Keywords Where We're Going!
bit.ly/coolaplvideo

Problem 2: (6 points) (mean: 4.7, median: 6, 3rd quartile: 6)

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* `'+'`*. Example:*

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

Solution:

```
plusvals lst = map snd $ filter (\t -> fst t == '+') lst
```

Problem 3: (2 points) (mean: 1.3, median: 1.25, 3rd quartile: 1.5)

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.

Both lines are clauses. The entirety of the first line is a fact; the whole second line is a rule that has four goals: `N > 0`, `M is N - 1`, etc.

Problem 4: (1 point) (mean: 0.9, median: 1, 3rd quartile: 1)

*What's the biggest problem with the following **Prolog** programming problem?*
"Write a predicate `maxint(+List)` that returns the largest integer in `List`."

There's really no idea of a predicate returning a value. `maxint` needs a second argument, perhaps `?Max`, to represent the largest value, either to test it or to instantiate it.

Problem 5: (11 points) (mean: 9.4, median: 10, 3rd quartile: 11)

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.

Example:

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

Solution:

```
wrap_with_each(Word, Wrappers, WrappedWord) :-
    member(W, Wrappers),
    atom_chars(Word, Lets),
    append([W|Lets], [W], NewLets),
    atom_chars(WrappedWord, NewLets).
```

Problem 6: (6 points) (mean: 5.7, median: 6, 3rd quartile: 6)

Write a Prolog predicate `second(?S,?L)` that expresses the relationship that `S` is the second element in the list `L`. `second` must provide [several behaviors; not shown here.]

Solution:

```
second(S,[_,S|_]).
```

Problem 7: (11 points) (mean: 9.4, median: 10.5, 3rd quartile: 11)

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. Example:

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

Solution:

```
prefixes(L, Min, Ps) :-
    findall(P, (append(P,_,L), length(P,Len), Len>=Min), Ps).
```

Problem 8: (12 points) (mean: 9.3, median: 11, 3rd quartile: 12)

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] .
```

This is my solution:

```
find_combo(Nums, Goal, Combo) :- helper(Nums, Goal, 0, Combo).

helper(_, Goal, Goal, []).
helper(Nums, Goal, Sum, [Num|RestOfNums]) :-
    select(Num, Nums, Remaining),
    NewSum is Sum + Num,
    helper(Remaining, Goal, NewSum, RestOfNums).
```

Martin came up with this:

```
find_combo(Ints, Goal, Ints) :-
    sumlist(Ints, Goal), !.

find_combo(Ints, Goal, Combo) :-
    select(_, Ints, Rest),
    find_combo(Rest, Goal, Combo).
```

Note: Problems 9 through 11 were 16 points each. The best two of those three scores were counted.

For the best two of three total, the stats are these mean: 27.8, median: 29.75, 3rd quartile: 31.75. For the per-problem stats for these three problems, zeroes were excluded from the computation.

Problem 9: (16 points) (mean: 13.8, median: 14, 3rd quartile: 16)

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
```

I wrote three different solutions for this problem. Here's the first:

```
(define (vequal? . vals)
    (define (helper vals)
        (if (< (length vals) 2)
            #t
            (let ([v1 (car vals)]
                  [v2 (cadr vals)])
                (and (equal? v1 v2)
                     (helper (cdr vals))))))
    (helper vals))
```

The second:

```
(define (vequal? . vals)
    (= (length vals)
       (apply + (map (lambda (x) (if (equal? x (car vals)) 1 0)) vals))))
```

The third:

```
(define (vequal? . vals)
    (andmap (lambda (x) (equal? x (car vals))) vals))
```

Problem 10: (16 points) (mean: 13.1, median: 16, 3rd quartile: 16)

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*`). Example:

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

Solution:

```
(define (let-vars expr)
    (if (member (car expr) '(let let*))
        (map car (second expr))
        #f))
```

Problem 11: (16 points) (mean: 12.5, median: 13, 3rd quartile: 14)

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. Examples:

```
> (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
```

Solution:

```
(define vars empty)

(define (store name value)
    (if (assoc name vars)
        (displayln "Duplicate")
        (set! vars (cons (cons name value) vars))))

(define (fetch name)
    (let ([val (assoc name vars)])
        (if val
            (displayln (cdr val))
            (displayln "Not found"))))
```

```
(define (names)
    (if (empty? vars)
        (displayln "No names")
        (displayln
            (string-join (map symbol->string
                              (sort (map car vars) symbol<?)) ", "))))
```

Problem 12: (4 points) (mean: 2.7, median: 3.5, 3rd quartile: 4)

*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. Examples:*

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

Solution:

```
(define-syntax-rule
    (++ var)
    (let ([prev var])
       (set! var (add1 var))
        prev))
```

A number of students solved it like this:

```
(define-syntax-rule
    (++ var)
    (begin
        (set! var (add1 var))
        (sub1 var)))
```

I wondered if anybody would do something with ++ or -- for a `macro-extra.txt` submission on assignment 8. Mr. Bush did a ++ with postfix semantics, just like this problem.

Problem 13: (5 points) (1 point each) (mean: 3.3, median: 3.5, 3rd quartile: 4.875)

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?

`x` needs to be treated as a variable name rather than being evaluated as an expression.

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

Tail calls are turned into jumps, effectively eliminating any possibility stack overflow due to too-deep recursion. There's typically a large gain in performance, too.

(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?)

There's no such procedure!

(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.

`(cons 3 4)` creates such a pair. Several students wrote something like `(cons a b)`, but if `b` were to be a list, the resulting pair would be a list.

(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?*

The characters `->` typically indicate a conversion of some sort. The asterisk suffix, such as in `let*`, `list*`, and `for*`, indicates some variation in behavior.

Problem 14: (2 points) (1 point each) (mean: 1.6, median: 2, 3rd quartile: 2)

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

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

Referencing `input` causes a line to be read from standard input. Assigning a value to `output` causes a line (or more) to be written to standard output.

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

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

Any digits at the start of `line` are removed, changing the value held by `line`.

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

```
c = a b
```

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

Because `x` has not been assigned a value and is used in an arithmetic context, its value is considered to be zero, resulting in `5` being assigned to `x`.

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

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

Go the the statement labeled `t`, after evaluating the preceding code, which sets `x` to zero if `x` > 0.

Problem 15: (2 points) (1 point each) (mean: 1.0, median: 1, 3rd quartile: 2)

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

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

Strings in Icon are mutable.

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

As stated, the answer is 39, an integer. In reality, however, addition has higher precedence than concatenation:

```
Icon Evaluator, Version 1.1, ? for help
][ 3 || "4" + 5
   r := "39"  (string)
```

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

String analysis and ordinary computation.

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

It fails!

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

The two focuses were portable software and high-level programming language facilities.

Extra Credit Section (½ point each unless otherwise noted) (mean: 2.6, median: 2.5, 3rd quartile: 3)

(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?

There never was a Part II.

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

John McCarthy

*(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.)*

Solution:

```
oddlen(L) :-
  append([_],Rest,L), append(A,B,Rest), length(A,Len), length(B,Len).
```

Only Mr. Do, Mr. Gullipelli, Mr. Le, and Mr. Nayak got this one correct.

(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.

`time` must be a special form. Consider `(time (length (range n)))`: If `time` is an ordinary procedure, `(length (range n))` will be evaluated as the first argument of time and then `time` will be called with `n`—AFTER the computation of `(length (range n))` has been done! Instead, `time` needs to get the current time, then do the computation, and then get the time again when done and report the difference. Here's a macro that does it:

```
(define-syntax-rule
    (my-time expr)
    (let*
        ([start (current-process-milliseconds)]
         [result expr])
      (printf "Time: ~ams\n" (- (current-process-milliseconds) start))
      expr))
```

Usage:

```
> (define n 10000000)
> (my-time (length (range n)))
Time: 862ms
10000000
> (my-time (length (range n)))
Time: 515ms
10000000
```

Here's the built-in `time` timing the same expression:

```
> (time (length (range n)))
cpu time: 451 real time: 459 gc time: 348
10000000
> (time (length (range n)))
cpu time: 636 real time: 643 gc time: 534
10000000
```

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

Effectively, it lets us write an in-line loop rather than needing to resort to a helper procedure.

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

```
punch = 'HELLO'
```

Mr. Fleming and Mr. Mana correctly speculated that it would cause a punched card with `HELLO` to be produced.

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

See slide 42 in the intro set for a partial list. A great number of students cited SNOBOL4, but it was created at Bell Labs, before Dr. Griswold came to 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`!)*

Dr. Peter Downey, as mentioned in Piazza post 147. I was working for Pete at the time as a member of the lab staff. I recall that he said to me something like, "How about this for the student machine?" and pointed to "lectura" on a list of names.

For that initial round of machine names, we used typeface names. Some others were bocklin, megaron (for the "big" machine—a VAX 8600, rated at 4 MIPS!), caslon, and baskerville. Being not far removed from a teenage boy, I chose "bembo" for my workstation's name and learned from Ralph that Bembo was in fact a popular font for paperback books. See also https://fontsinuse.com/typefaces/72787/lectura.

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

Sargent was a professor in Optical Sciences. See also https://conservancy.umn.edu/handle/11299/107341 and https://conservancy.umn.edu/handle/11299/107340 (Should those links die at some point, Google for `oh256rmg.pdf` and `oh201rmg.pdf`.)

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

He had a desk, but no chair.

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

There were lots of interesting answers for this one...

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

I'll say there was a clear best on this one, by Mr. Nisterenko:

Q: "If you are stuck on an island what language from this course would you chose to help you?"

A: "Prolog—it knows all the facts. Haskell is too lazy, so it wouldn't help. Racket would be scheming your demise."

Statistics

Here are all 54 scores, in descending order:

```
103.00, 102.00, 101.00, 100.50, 100.50, 100.50, 100.00, 100.00, 100.00, 99.00,
98.50, 98.00, 97.00, 97.00, 96.50, 96.50, 96.00, 96.00, 94.50, 93.50, 93.50,
93.50, 93.00, 90.00, 89.50, 88.50, 88.00, 86.50, 85.00, 85.00, 84.50, 84.50,
84.00, 84.00, 84.00, 83.50, 81.50, 80.00, 80.00, 79.00, 78.50, 78.50, 78.50,
77.50, 77.00, 76.00, 72.00, 71.00, 69.00, 66.00, 53.50, 47.00, 45.50, 25.00
```

Mean: 85.23
Median: 87.25
3rd Quartile: 96.88

Here's a table with per-problem statistics. Median/possible shows per-problem median scores divided by possible points, expressed as a percentage.

videos	plusvals	anatomy	maxint	wwe	second	prefixes	find_combo	vequal?	let-vars	store	Best 2	plusplus	Racket S/A	SNOBOL4	Icon	EC	Total
5.6	4.7	1.3	0.9	9.4	5.7	9.4	9.3	13.8	13.1	12.5	27.8	2.7	3.3	1.6	1.0	2.6	85.23
6	6	1.25	1	10	6	10.5	11	14	16	13	29.75	3.5	3.5	2	1	2.5	87.25
6	6	1.5	1	11	6	11	12	16	16	14	31.75	4	4.875	2	2	3	96.875
100.0	100.0	62.5	100.0	90.9	100.0	95.5	91.7	87.5	100.0	81.3	93.0	87.5	70.0	100.0	50.0		

Here's a histogram of all scores: