Your last name	Last name of classmates beside you (or "wall"/ "aisle")
_____	On my left: _____ On my right: _____

CSC 372 Final Exam

Thursday, December 15, 2022

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.

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

The last five minutes of the exam is a "seatbelts required" period, 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.

Aside from a sheet of paper with a list of things you learned from your classmates' videos, you are allowed no reference materials whatsoever.

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`" and "`init`" for "`initialize`"

It's fine to use helper functions/methods/predicates unless they are specifically prohibited or a specific form for the code is specified.

Don't make a programming problem hard 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."

It is better to put forth a solution that violates stated restrictions than to leave it blank—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 **put your initials in the lower right-hand corner of the top side of each sheet, checking to be sure you have all eight sheets.**

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

PLEASE, PLEASE, PLEASE RAISE YOUR HAND AND ASK FOR A HINT IF YOU'RE STUMPED ON A PROBLEM!

2

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: (10 points)

The following program is in Tcl and was adapted from `rosettacode.org/wiki/Ordered_words#Tcl`. Read through the code and then for one point each, briefly answer the questions that follow below the code.

```
 1      # Return a list of the ordered words (of maximal length) in a list
 2      proc chooseOrderedWords list {
 3          set maxlen 0
 4          set orderedOfMaxlen {}
 5          foreach word $list {
 6              # Condition to determine whether a word is ordered;
 7              # are its characters in sorted order?
 8              if {$word eq [join [lsort [split $word ""]] ""]} {
 9                  set wordlen [string length $word]
10                  if {$wordlen > $maxlen} {
11                      set maxlen $wordlen
12                      set orderedOfMaxlen {}
13                  }
14                  if {$wordlen == $maxlen} {
15                      lappend orderedOfMaxlen $word
16                  }
17              }
18          }
19          return $orderedOfMaxlen
20      }
21      # Get the dictionary and print the ordered words from it
22      set t [http::geturl "http://cs.arizona.edu/classes/cs372/fall22/web2"]
23
24      set result [chooseOrderedWords [http::data $t]]
25
26      puts "Longest ordered words:"
27      for {set i 0} {$i < [llength $result]} {incr i} {
28          puts [lindex $result $i]
29          }
```

(1) What does the syntax for procedure definitions seem to be?

(2) How does one apparently create and initialize a variable?

(3) What appears to be the rule about when a variable name needs a sigil?

(4) What purpose do square brackets (`[` and `]`) seem to serve?

(5) At line 8, `eq` is used to test equality but at line 14, == is used to test for equality. Why are two different operators used?

(6) What does the `{}` in line 4 apparently represent?

(7) Roughly translate line 5 (`foreach...`) into Ruby code that uses an iterator.

(8) Roughly translate line 28 (`puts [lindex $result $i]`) into Ruby.

For one point each, **make up to four more significant observations** about the code, but maximum score for the problem is 10.

4

Problem 3: (11 points)

Write a Haskell function `avgvows` of type `[String] -> Double` that returns the average (mean) number of vowels per word in a body of text, ignoring one-letter words. Assume all letters are lower-case.

Examples:

```
> avgvows ["what a", "treat"]
1.5

> avgvows ["ouch oops", "my bad"]
1.25

> avgvows ["my my"]
0.0
```

Note that the argument is a LIST of `String`s!

Assume there is at least one word in the body of text. **Assume** that `3 / 2` produces `1.5`.

5

Problem 4: (8 points)

Write a Prolog predicate `all_atoms(+L)` that succeeds iff the list `L` is not empty and all its values are atoms.

Examples:

```
?- all_atoms([just,a,test]).
true.

?- all_atoms([just,1,test]).
false.

?- all_atoms([]).
false.
```

Note that `all_atoms` fails if given an empty list.

I wrote two very different solutions for this problem. One of them uses `findall/3` and the other does not.

6

Problem 5: (20 points)

This is the Prolog problem like `connect` and pit-crossing that you were told to expect.

Write a Prolog predicate `makepath/1` that takes a list of "layers" and prints a sequencing of **ALL** of them that has a continuous path of asterisks from top to bottom. Here's an example of a path through five layers:

```
?- makepath(['---*-','--*--','----*','--*--', '---*-']).
---*-
--*--
--*--
---*-
----*
true.
```

Layers consist of only dashes and asterisks. Assume all layers are the same length. There will be zero or more asterisks in each layer.

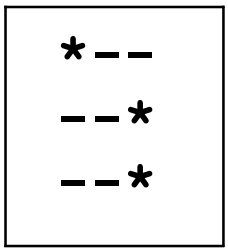
For the path from one layer to the next to be valid there must be an asterisk in the same position in both layers, or within one position/place to the left or right.

Here's an example with multiple asterisks in some layers:

```
?- makepath(['-*-*-','---*-','**--*','--*--']).
-*-*-
**--*
--*--
---*-
true.
```

Here's a case where no path exists. Note that **it prints the number of layers that were provided.**

```
?- makepath(['*--','--*','--*']).
No sequencing for those 3 layers
true.
```



Unlike `connect` and pit-crossing, where there could be leftover cables or planks, `makepath` requires ALL layers to be used.

There is space for your solution on the next page.

7

For reference:

```
?- makepath(['-*-*-','--*--','---*-']).
-*-*-
--*--
---*-
true.

?- makepath(['*--','--*','--*']).
No sequencing for those 3 layers
true.
```

Some possibly useful predicates:

```
select(?Element, ?List, ?Remaining)
atom_chars(?Atom, ?CharList)
nth0(?Index, ?List, ?Elem)
```

Remember:

- ALL layers must be used!
- If a layer has an asterisk in position N and the following layer has an asterisk in position N-1, N, or N+1, there is a path between those two layers.
- `makepath` produces printed output. (Just like I did in `connect.pl`, my solution has a helper predicate to find a suitable sequence, and a helper to print a sequence. In turn, my `makepath` uses those two helpers.)

Problem 6: (15 points)

Write a Ruby program `hmt.rb` that, based on command-line arguments, prints the "head", "tail" and/or "middle" of its input lines, possibly multiple times. This problem was inspired by the `head` and `tail` UNIX commands.

The command

```
$ ruby hmt.rb ht 5 < x.txt
```

specifies that the "head" (`h`) and "tail" (`t`) of the input are to be printed. This causes the first five lines and then the last five lines of `x.txt` to be printed; a total of ten lines.

The command

```
$ ruby hmt.rb tth 20 < x.txt
```

causes the last twenty lines of `x.txt` to be printed twice, followed by the first twenty lines; a total of sixty lines.

Recall that the UNIX `seq` command, can be used to print numbers from 1 through some N:

```
$ seq 4
1
2
3
4
$
```

Let's use `seq` to experiment with `hmt.rb`:

```
$ seq 6 | ruby hmt.rb thm 2
5
6
1
2
3
4
```

```
$ seq 5 | ruby hmt.rb ttt 1
5
5
5
```

```
$ seq 6 | ruby hmt.rb m 4
2
3
4
5
```

Think of "middle" as being "middle-ish". My version of `hmt.rb` says the middle two lines of a five-line file are the second and third lines but if your version produces the third and fourth lines, that's fine!

If the number of lines to print is not specified, it **defaults to three**. Example:

```
$ seq 1000 | ruby hmt.rb tt
998
999
1000
998
999
1000
```

(continued on next page)

9

Important: Don't worry about any error handling!

- Assume that the first argument is a non-empty string consisting of any combination of `h`s, `m`s, and `t`s.
- Assume that if a second argument is present, it is a positive integer no greater than the number of input lines.

For reference:

```
$ seq 6 | ruby hmt.rb thm 2
5
6
1
2
3
4
```

```
$ seq 5 | ruby hmt.rb ttt 1
5
5
5
```

```
$ seq 6 | ruby hmt.rb hh
1
2
3
1
2
3
```

Implementation notes:

- Use `STDIN.readlines` to read the entire input and produce an array of all the lines.
- `puts a` where `a` is an array, will print the elements of `a`, one per line.
- `String` has an iterator `each_char` that calls its block with each character in turn.

Problem 7: (9 points)

Write a Ruby method `lc_occurs(s,n)` that returns a string of the lower-case letters that occur **n** or more times in the string `s`. Characters in the result are in alphabetical order.

Examples:

```
>> lc_occurs("just a test", 1)
=> "aejstu"

>> lc_occurs("just a test", 2)
=> "st"

>> lc_occurs("just a test", 3)
=> "t"

>> lc_occurs("just a test", 4)
=> ""

>> lc_occurs("",0)
=> "abcdefghijklmnopqrstuvwxyz"
```

RESTRICTION: You may not use 'for' or 'while' statements in your solution.

Implementation notes:

- The expression `'a'..'z'` produces a `Range` consisting of the lower-case letters.
- `Range` includes `Enumerable` and has an `each` iterator.
- `String` has an iterator `each_char` that calls its block with each character in turn.

Problem 8: (12 points) **11**

In this problem you are to implement a Ruby class named `IntRange` that represents a non-empty range of integers.

Let's make an `IntRange` that represents the integers from 3 through 7:

```
>> r = IntRange.new(3, 7)
```

The `inspect` method produces a space-separated string of the integers along with the total number of values:

```
>> r.inspect
=> "3 4 5 6 7 (5 values)"
```

The `size` and `bounds` methods produce the number of values and the lower and upper ends, respectively:

```
>> r.size
=> 5
>> r.bounds
=> [3, 7]
```

Ranges can be subscripted and like Icon, indices are one-based. Negative indexing is not supported. Out of bounds indices produce `nil`.

```
>> r[1]
=> 3

>> r[5]
=> 7

>> [r[0], r[6]]
=> [nil, nil]
```

Subscripting like `r[start,len]` is not supported.

An `IntRange` can have as little as one value but **the constructor assumes that first <= last**.

```
>> r2 = IntRange.new(-100,-100)

>> r2.inspect
=> "-100 (1 values)"

>> r2[1]
=> -100

>> r2[2]
=> nil
```

There is space for your solution and a note about extra credit on the next page.

12

For reference:

```
>> r = IntRange.new(3, 7)

>> r.inspect
=> "3 4 5 6 7 (5 values)"

>> r.size
=> 5

>> r.bounds
=> [3, 7]

>> r[1]
=> 3

>> r[5]
=> 7

>> [r[0], r[6]]
=> [nil, nil]
```

Earn a point of extra-credit by not having `return` appear in your code!

Problem 9: (5 points) (one point each unless otherwise indicated)

The following questions and problems are related to Ruby.

(1) What are two examples of sigils in Ruby and what does each indicate?

(2) A novice programmer might simply understand that the following loop terminates when end of file is reached on standard input. What's a more thorough explanation of what happens when end of file is reached on standard input?

```
while line = STDIN.gets
    puts line
end
```

(3) For any Ruby Integer `n`, I'd like `n.big?` to return a "truthy" value if `n` is greater than 1000 and a "falsey" value if not. Write Ruby code to fulfill my wish.

(4) What is the essence of the duck typing mindset?

(5) If you only remember one thing about Ruby, what will it be?

Problem 10: (2 points)

Write a SNOBOL4 program `twice.sno` that prompts for the user for a number and then prints twice that number, like this:

```
$ snobol4 twice.sno
Number?
7
Twice 7 is 14
```

For a point of extra credit, make it loop until the user hits ^D (end-of-file) to terminate it.

Problem 11: (2 points)

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 2 points.

(1) Write two distinctly different Icon expressions that fail to produce a value.

(2) Write an expression that produces the length of `s`, where `s` is an Icon string.

(3) Write an expression that produces the sum of the first and last elements of `L`, where `L` is a list of strings that represent integers. If `L` is `["7", "45", "8"]` the sum would be `15`. Assume that `L` has at least one element.

(4) What is a major aspect of Icon that Ralph Griswold was disappointed with?

(5) Regarding language design, what was Ralph Griswold's opinion about whether a language should be easy to implement?

15

Extra Credit Section (½ point each unless otherwise noted)

(1) With respect to programming languages, the term "mixin" was said to have been borrowed from the product line of a Somerville, Massachusetts business. What was the primary product of that business?

(2) What's the language in the SNOBOL family that was developed between SNOBOL4 and Icon and that "had everything".

(3) Brooks Law says that "Adding manpower to a late software project makes it _____."

(4) To the best of `whm`'s knowledge, what was the size of the RATSNO (Rational SNOBOL4) user community at its peak?

(5) Write the Prolog predicate `member/2` in terms of `nth0/3`.

(6) Write a SNOBOL4 pattern that would match decimal fractions like `123.45`, `0.789`, and `1000.1`.

(7) Among Haskell, Icon, Java, Prolog, Python, Ruby, and SNOBOL4, which are the three oldest languages?

(8) What's relatively unique about when Ruby was created?

(9) Is Java a compiled language or an interpreted language? Briefly justify your answer. (**1 point**)

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