Final Exam, CS-415: Principles of Programming Languages Spring 2020, Iyengar

Due: **Noon Thursday 5/7/20.**

Describe at least two advantages and two disadvantages of the programming language you chose for your case study. (10)

C Language:

**Advantages: 1. Very fast and efficient compared to many other popular languages**

**2. Manual Memory Management allows for proprietary data storage and handling that isn't possible in many other popular languages**

**Disadvantages: 1. Weak typing, manual memory management, etc. can make the language difficult to debug**

**2. Manual Memory Management poses a major security risk as, without the proper security procedures in place (in OS), a program could read/alter memory used by another program**

What is meant by functional language? (10)

**Functional programming, as opposed to other paradigms like object oriented languages, focus on using functions to accomplish their tasks. Strictly speaking, any of these functions should only have one output for any one input. Mathematical functions are a common example of such a thing. There aren't many languages that are purely functional, Haskell likely being the most popular of them. Other languages that support other paradigms like C++, C#, and Java, can also be considered functional languages.**

What is meant by non-mutative language? (10)

**Mutation in programming is when a variable is changed after it's been defined. A non-mutative language doesn't allow that to happen. Lisp and Prolog would be examples of that as once a list is defined, it can't be changed. It can be used, and different list based off of the first can be returned, but the original list remains intact.**

What is meant by declarative language? (10)

**Delarative languages abstract much of how something is done allowing a programmer to just saw what should be done. You don't care how it gets done, just that it is. SQL for example, has a lot of networking and file IO under the hood that goes into a simple SQL query. I don't have to tell it how to do it, I just ask for the query and I get it. If I wanted to do a similar database handling in strict Java, I would have to write all of the file/network IO myself, expressly telling it how to do so. By this logic, declarative languages tend to be higher level languages.**

Problem solving is nothing but symbol manipulation. Explain. (5)

**At the most basic layer, look at the number 1. The bar with a hat and flat base consistently mean "One" to you and me, but it really represents the quantity of a single thing. That symbol could be totally different though. Other languages with different numerical characters are a good example of that. If we agree on what a symbol represents, then it has meaning.**

**With that in mind, take a look at a simple equation: "1+2". That is simply 3 symbols. We all agreed that 1 represents a single thing. "+" represents the concept of addition. "2" represents the quantity of one more than one. We interpret these symbols to have a meaning that is not inherent to the symbols themselves, because we preemptively agreed on their meaning. Show those symbols to an alien and they wouldn't have a clue what they meant. Language as a whole fits this. Every character, and every word, and every sentence structure, only has a meaning because we agreed on it.**

**Take this idea to programming languages, and there are so many levels to it. Not only does each language have it's own symbolic structure and syntax, but think about how that stuff is stored in memory. We don't have to use 2's complement to store numbers. We could use some arbitrary random choice of binary representation if we wanted to. It would change the algorithms that manipulate those numbers, only because those algorithms are symbolic in themselves. Addition in 2's complement only works as addition because we defined addition's meaning. The same can be said about any algorithm in any language. It reaches all the way down to how the hardware works, all the way up to how we communicate our ideas to others.**

What is a proposition? What is a predicate (how is it different from a proposition)? (5)

**A proposition is a statement that has some overarching truth value. "Adam Stammer is 20 feet tall." is an example of this because that statement is definitively false; I am actually 6'4", not 20", tall.**

**Predicates are like propositions, but they use undefined variables that must be substituted before they can be evaluated. Building off my previous example, let's say the "P(x) = Adam Stammer is X feet tall." We don't know if this is true or false because we don't know what x is. That means that this is not a proposition. If we substitute in "2", then P(2) means "Adam Stammer is 2 feet tall." This statement can be evaluated, as there are no unknown variables, making it a proposition, that we know evaluates to false because the statement is not true.**

Describe the three core primitive operations in Lisp? (10)

if by that you mean the list primitive functions, I think it would be the following.

**car – gets the first item from a list**

**cdr – gets everything except the first item from a list**

**cons – concatenates two lists together**

**With these functions you can constructive and destructively build any list you wanted.**

**If instead you're looking for the classifications of primitives they would be the following:**

**Function – evaluate a given input to generate a given output. Mathemeatical operations (+, *, -, etc) would be an example of this. Each of the list operations listed above would also be considered function operations.**

**Predicate – Predicates on the other, as described above, take some input and return a boolean true or false(nil). Equality checks (>, <, equal, etc) would be the most simple of these. listp was also one that came in handy in our assignments.**

**Macros – these perform many of the same things as functions but do so syntactically differently. Cond is an example of this as it uses its own syntax while still taking in inputs and giving outputs. Defun allows you to define your own functions, but defun itself is a macro.**

Write a prolog predicate to **count** the number of *positive* values in a list of numbers. (20)

```
countpositives([],0).

countpositives(List, R) :- List = [Head | Rest], (>( Head, 0) ->
countpositives(Rest, T), R is T + 1  ;  countpositives(Rest, T), R = T).
```

Write a lisp function(s) that takes a list of items of the form – '(a b c d e f g) and returns a list of pair of lists the form – '((a c e g) (b d f)). (20).

**The first list (odds) always has an extra period in the second to last location. So '(a b c d e f g) actually evaluates to '((a c e . g) (b d f)). I think I know which line is doing it but I'm not sure how to fix it. I tried a lot of stuff and it didn't really help.**

```
(defun listPair (myList)
      (if myList ; if the list isn't empty
            (if (cdr (cdr myList)) ; if the list is length 2 or more
                        (list
                              (cons (car myList) (car (listPair (cdr (cdr
myList))))) ; combine the first item with the rest of the odd items
                              (cons (car (cdr myList)) (car (cdr (listPair
(cdr (cdr myList)))))) ; combine the second ite mwith rest of the even items
                        )

                  (list (car myList) (cdr myList)) ; return the first and
second items split
                  ;i think this line is causing the extra period but I'm not
sure
            )
            '(() ()) ; if the list is empty (base case)
      )
)
```