Name: \_\_\_\_\_\_\_\_\_\_Joseph Leveille\_\_\_\_\_\_\_\_\_\_\_

Student ID: \_\_\_3955537\_\_\_\_\_\_\_\_\_\_\_

Honor code: I have neither given or received, nor have I tolerated others’ use of unauthorized aid.

Joseph Leveille

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A slight variation: \*All\* questions are 5 points this time!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

0) I should write device drivers in Prolog. **True** or False

*Why?*

Prolog would be a good choice for writing device drivers because…

* It’s relational organization allows quick response times for queries
* There is a guaranteed a response from each query, even if there is some unexpected case.

1) Why is SNOBOL4 really, really good at text handling and strings, generally? ( 3 reasons)

SNOBOL4 is string-based, so it is specifically made with built-in functionality for parsing through strings, whereas many other languages require importing libraries in order to mess with strings effectively.

With its built-in tools, SNOBOL4 can parse through strings with very few lines of code, so it is very efficient with the functions that it has.

One such tool is the pattern data type, which allows SNOBOL4 to search strings very efficiently.

Long story short: SNOBOL4 is made for manipulating and parsing strings.

2) List AND DISCUSS two uses of Prolog.

Expert systems – These are programs where a large number of variables are related through a complicated network. Prolog is great for keeping track of those relationships and using them to find the answer to a problem much easier than another language could.

Proving mathematical theorems – Prolog can be used to prove theorems because it can hold rules to apply to variables, and string together to perform more complicated tasks.

3) What is (\* (+ 3 3) (+ 6 1)) in normal human being terms (The Expression, please)?

(3+3)\*(6+1) = 6\*7 = 42

The product of the sum of three and three with the sum of six and one.

4) What does every language eventually boil down to? What can we never escape? Why?

Machine code is unavoidable because that is the only format of instructions that the hardware can ‘understand’ or act upon. Without creating machine code, the electronics that make up the computer at its lowest level could not perform anything sensible, since they are built to deal with digital signals – only 0s and 1s in a specific format (machine code).

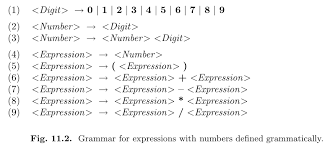
5) How can something be both compiled and interpreted? What impact does this have?

Some languages can be compiled into another format before being interpreted when necessary. This may increase the time it takes to run the program, since two separate steps are happening.

6) Typing: why is it such a big deal? What kinds of decisions do we make about it? Why?

Typing can affect precision of your data in the case of integers vs. doubles vs. floats vs. longs, which may matter a lot, depending on the problems being tackled. This introduces the trade off between memory space & precision that developers of large applications must be conscious of in order to make their product good for its specific purpose. For memory efficiency, it makes the most sense to use integers, or as imprecise of a data type as possible, but calculations will be made more precise by using the more memory-costly data types like long longs or doubles.

7) Add to/change (as needed) this ruleset to support negative numbers **and** numbers after the decimal point.



Between lines 1 & 2:

<Symbol> -> .|-

Between lines 2 & 3:

<Number> -> <Symbol> <Digit>

8) What things do “real” (spoken) languages and programming languages have in common? (2, please)

Grammar = syntax

Both programming languages and spoken languages have formatting specific themselves, just called grammar for spoken or syntax for programming languages. Without properly using the grammar or syntax, the message or instruction may not be communicated effectively, or at all.

There are tons of options of languages, and differently structured families of them. In spoken languages, there are groups of languages that are very similar, like the Romantic languages (or Java-based), which are very distant from languages in other categories, such as Mandarin Chinese (or SNOBOL4).

9) Clojure….:

[Tip: defn defines a new function.]

**(defn fact [n]  
 (if (< n 2)  
 1  
 (\* n (fact(- n 1)))))**

What is this?

A recursive function that calculates n factorial.

Where is the stop (or base case).... [and the other kind of case]?

n < 2 is the check for the base case, so the recursion with stop once n = 1.

Recursion continues decrementing n by 1 from its original value while n > 1.

What does this (in terms of length, and syntax, and so on) tell us about ~~lisp[~~functional languages] and the tradeoffs - good and bad - for it??

This piece of code shows that functional languages can handle recursion quite compactly, but also that parentheses can quickly get difficult to keep track of with these languages. Even in this 4 line function, there are 5 closing parentheses at the end, so organization is very necessary for bigger pieces of code.

10) SNOBOL

**define('rfact(n)') :(rfact\_end)  
rfact rfact = le(n,0) 1 :s(return)  
 rfact = n \* rfact(n - 1) :(return)  
rfact\_end**

What is this?

A recursive function that calculates n factorial.

Where is the stop (or base case).... ?

The base case is when n = 0

What does this (in terms of length, and syntax, and so on) tell us about SNOBOL4 and the tradeoffs - good and bad- for it? (You can, maybe even should? refer to the previous question…)

This shows that SNOBOL4 can perform recursion, as well as comparisons, in a compact way. Even though it is a string based language, it can handle calculations just as easily.

11) Why do SNOBOL4 and Lisp/Clojure/Scala really really do amazing with parsing? At least one really good reason/capability per language group, please. (Note extra space. Good thinking appreciated here - they ARE different and your answer should reflect that!)

SNOBOL4 is string-based, so it is specifically made with built-in functionality for parsing through strings. With these tools (like patterns), SNOBOL4 can parse through strings with very few lines of code, so it is very efficient with the functions that it has.

The functional languages do well with parsing because they can handle arbitrarily sized lists, as well as lists of lists. They are designed to handle lists, so they have the necessary functions built in to search through and handle lists very efficiently.

12) List (yeah, I just did that...again) two places Lisp and its children (like Clojure) get used - and discuss briefly… one should be industry/operational flavored:

* AutoCAD – Lisp was used to develop AutoCAD because it can handle arbitrarily sized sets of data, which is helpful for assemblies of tons of part files.

* Machines that must work reliably, like aircraft/spacecraft – This is because the functional languages work reliably (once you get them to work in the first place) and do not require constant updates to remain usable.

13) Why are there so many (programming) languages?

There are tons of different types of problems to solve with computers that are best approached in a variety of ways, so separate programming languages develop out of those different needs.

14) Explain what the biggest differences (at least 2) between the functional language family members and, say, C or Java?

Parentheses! The functional languages use very different formatting and syntax from C & Java, such as (+ 3 2) instead of the more traditional format 3 + 2. This is a big adjustment for people that are used to C-based languages, as the need for parentheses around each operation or function call can get confusing.

The functional languages are usually interpreted instead of compiled like C & Java, making them slower to load/interpret, but they are very consistent in their outputs because they, by default, do not allow their values to change.

15) Which of the families did you like the most? Why? (For full credit, discuss tradeoffs and demonstrate you understand attributes of the families!)

Prolog (logic family) was my favorite to see because it is unique from the more familiar C-based languages, and it makes sense for use in systems where items are related to each other in a variety of ways. Using Prolog, it is simple to complete problems that require analyzing complicated networks of relationships between variables, which would be much tougher to do with another family of languages. This comes with the trade off that the syntax is vastly different from C-based languages, but that some sense in order to attack problems from a different angle.

***Please be creative with the remaining space…..***