Name: Dawson Demien

Student ID: 4001332

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

(sign)

Dawson Demien

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

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

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

*Why?*

**The reason you should not write device drivers in Prolog is because Prolog is primarily a query language. Therefore, the language’s logic is built on relationships, facts, and rules. This would be horrible for writing device drivers because the language describes the logic of what is happening but does not describe how it is happening. Therefore, bugs would be incredibly annoying, and it would be hard to create software that thinks for someone’s machine. Rather the software would say what it is happening. Prolog can talk good, but it can’t think as well (in lay-man’s terms).**

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

**SNOBOL allows people to easily Search, Change, and Utilize string variables in a few lines of code. This is because SNOBOL is very good at pattern-matching. This means SNOBOL4 can perform tasks such as parsing strings, searching for a certain substring within a string, and even finding substrings based on the surrounding text. The reason it is so good with strings is because it is a string-oriented symbolic language, utilizes pattern-matching, and was purposefully created by Bell Labs for the sole reason of manipulating strings.**

2) List AND DISCUSS two uses of Prolog.

1. **Natural Language Processing: This is the interaction between computers and the human language. It is mainly concerned with how to program a computer to process and analyze large amounts of natural data. The goal tends to be a computer that can “understand” the contents of a document including the context of nuances. In completing this goal, a software can accurately grab data from a document and categorize it. On a dumbed down scale this would be similar to when Prolog is capable of creating the nuances of relations between variables such as like(mary, john). Meaning when someone queries Mary they can find that she likes John.**
2. **Type Systems: This is when a program creates a logical system comprising of a set of rules that assigns a property called a type to variables, expressions, or functions. The purpose of a type system is to reduce bugs be defining interfaces between the different parts of the computer program, and then checking that the parts have been connected in a consistent way. There are also other purposes such as expressing business rules. On a dumbed down scale this would be like creating a family tree in Prolog, where there are relations that are checked for consistency, and at which can be queried.**

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

**This equation would be: (3+3) \* (6+1)**

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

**Every language eventually boils down to machine code or bytes. We can never escape machine code (bytes/memory) because it is what every computer uses to work correctly. It is what a computer uses to pull inputs and push outputs to the user. Every computer has memory and registers; therefore, for a computer to work correctly it must be in touch with these registers and memory to execute tasks. If someone wants to execute code, then that code needs some way to either be compiled or interpreted to machine code/bytes. Overall, every language becomes bytes, and we can never escape either the interpretation or compilation into machine code.**

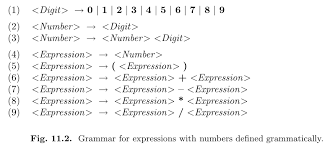
5) How can something be both compiled and interpreted? What impact does this have?

**The way in which a language can be both interpreted and compiled is by first compiling the code and then interpreting that code. An example would be Java at which is first compiled to binary byte code and then interpreted by the JVM (Java Virtual Machine). This allows the virtual machine to be small and efficient, running nearly as fast as the CPU running native compiled code. The impact is greater efficiency and speed.**

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

**Typing is a big deal because it defines memory and tells the compiler/interpreter how the data will be used. The typing dictates the operations that can be done to it, the meaning of the data, and the way the values are stored. The reason why we make these decisions is because each different type of data has different memory and bytes relevant to the data type. Furthermore, one cannot simply perform mathematical functions on strings or characters. Without typing there would be plenty of errors to go around.**

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



**<NDigit> -> -9 | -8 | -7 | -6 | -5 | -4 | -3 | -2 | -1**

**<Number> -> <NDigit>**

**<Number> -> <NDigit><Number>**

**<Number> -> <Number>.<Digit> //Having number here first means number in front of decimal can be either positive or negative, but needs digit after decimal because decimal places cannot be negative**

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

1. **Syntax: set of rules that govern how language elements are combined to form a valid expression. In natural language the syntax would be grammar, which informs people on how to formulate a correct/coherent sentence (nouns/verbs). Syntax in a programming language would be symbols that are needed to complete a line of code such as parentheses around an if statement.**
2. **Semantics: This is what the language evaluates to. Both coding languages and natural languages evaluate to something. Natural languages evaluate to feelings, thoughts, and questions/ answers. Whereas a programming language evaluates expressions to CPU instructions to comprise the flow of a program.**

9) Clojure….:

[Tip: defn defines a new function.]

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

What is this?

**This function calculates the factorial of the number that is passed in.**

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

**The base case is when the number passed in is less than 2.**

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

**Lisp is great at recursive calls. We can calculate the factorial of a number with a simple three-line function. The downside to this is how confusing the syntax is. There are a lot of parentheses and the order in which the logic is within the parentheses is backwards. The syntax if Lisp can be hard to grasp and cause a big learning curve, but once someone understands it then they will be capable of producing very simple recursive calls that are small and efficient.**

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?

**This also calculates the factorial of a number.**   
Where is the stop (or base case).... ?

**The base case is when the number n is less then or equal to 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…)

**I hate this language lol. But seriously, objectively the language was also capable of producing a recursive function within only a few lines of code just as Lisp was able to. This is good considering that when one wants to produce a recursive method, it should be concise and efficient. At which this method is, but the syntax is God awful. The syntax is worse than lisp because here it really does seem like one is trying to calculate math within the realm of strings. One must assign a variable to every line of code for it to work correctly which does not match up the best with mathematical computations. This leaves for a learning curve as well and could possibly make a simple recursive method seem like much more than it really is. Overall, there is a concise recursive method present that is probably efficient, but to get there one must learn absolutely atrocious syntax. The tradeoffs are horrible syntax for concise code.**

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!)

**Well as I have answered before for SNOBOL4, it is very good at parsing because of the fact that it was built as a string-oriented language that excels in pattern-matching. The pattern-matching is exactly what allows for easy string parsing within SNOBOL4.**

**From here on down they are all functional programming languages which helps incredibly with parsing in that fact alone.**

**Lisp is good at string parsing because it has very simple grammar. The longer the grammar the more one must do to parse a string. Therefore, lisp is so good at parsing because the grammar is simple and short.**

**Clojure is good at string parsing because the annoying symbols that come with parsing are filtered out which makes it very friendly. It takes out all of the useless syntax symbols involved in parsing that one may have when parsing in say Java. There are built in map and filter functions that help take care of a lot of the useless syntax symbols one may have to deal with in other languages.**

**Scala is good at string parsing because allows for the programmer to implement combinator functions. These take simple functions as input and then combines them together for a more complicated output function. This becomes very useful when implementing parsers.**

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:

* **Artificial Intelligence: Lisp and its children are used in AI. Lisp is great at tracking a problem one does not know how to solve yet. This is perfect considering it characterizes AI perfectly. Furthermore, lisp supports the implementation of software that computes with symbols, and this is the very core of Lisp (computing with symbols/ symbolic expressions). Furthermore, lisp is great at recursion which is extremely needed in AI considering AI is all about learning on itself and recursive calls.**

* **Bioinformatics and computational biology. Lisp is used here because it has unparalleled support for homoiconicity, domain-specific languages, extensible macros, and error handling. Lisp makes it easy to create extensible macros which facilitates the creation of modularized extensions to help bioinformaticians easily create plug-ins for their software. This, in turn, paves the way for creating enterprise-level, fault-tolerant domain-specific languages in any research area or specialization.**

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

**There are so many different programming languages because each language is created as an answer for a problem in mind. A prime example would be at Bell Labs where they created SNOBOL as a way for them to manipulate symbolic string data better. Another example would be C programming which was created for the sole purpose of building Unix on.**

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

**One of the biggest differences between the functional language family and C and Java is that functional languages have immutable data. This means that the data is not directly changed and if someone wants to save the manipulated data, they must save it as a new variable; whereas, in Java or C the data is mutable, and the manipulations of data are applied right to the same variable. Another difference is that functional programming follows a declarative programming model and C/Java follows an imperative programming model. This means that functional languages say “what they want” and C/Java says how to get what they want.**

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

**The family of programming I liked the most is the declarative programming family. Therefore, of all the languages in this class it would be Prolog. For some reason I genuinely enjoy creating rules and querying a set of data/code. The tradeoffs of this language would be that one can easily ask a set of data a question and get back what they asked for, but data manipulation is a whole other issue. Declarative languages do not have the full power of other programming languages. Obviously declarative languages cannot address the “how” and therefore needs help from other languages to do this. This is because people often run out of luck when querying data and eventually run into trying to address “how” something is to be done. The example I am thinking of is SQL utilizing Javascript. Therefore, a tradeoff of Declarative languages is that even though it can help one get to a the answer of “what” they will eventually need help from another programming language to address the “how.”**

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

Have a wonderful Christmas break! Thank you for an enjoyable last fall semester!

-Dawson