# LISP Term Paper

Christopher Koepke

CS 4080.03

May 13, 2022

My chosen language was LISP. Me and my teammate (Ismael Garcia) originally had three preferred languages to learn: RUBY, PERL, and LISP. I was personally interested in LISP because of its recommended use in Artificial Intelligence. RUBY and PERL honestly seemed like an easy topic to present about, but now I know that is not really the case; although I still feel that I would have had an easier time learning about the other two programming languages. The actual work on my presentation did not seriously start until about three weeks before our presentation date. My partner did not start working on the presentation until two days before the presentation, because of some "medical" issue he had. Those three weeks were very stressful and led me to cram as much information as possible in the allotted time with the assumption that I would be presenting by myself. This led to what I believed would be a disjointed presentation on my chosen language. If my teammate did not state that he would work on the rest of the presentation before we were to present, then I would have finished everything and reworked it to flow more smoothly. In hindsight, I could have spent more time on this presentation, but a combination of other classwork, personal issues/engagements, and some procrastination is what led to the final product. Overall, I tried the best that I could with my allotted time.

When I first started to learn about LISP, I was completely lost. Not because I was not invested in learning about it, but because it was so different than what I was used to seeing in other programming languages, specifically Java and C++. The two programming languages that I was taught in my community college (Java/C++) were very similar. I did not give much thought to the differences between them, other than the syntax used and how to use the built-in functions/libraries. Java was my first introduction to programming and was difficult to learn. C++ was next and was easier to learn because it was similar to Java. Those are the only two languages that I know how to write code for. I was familiar with the names of other programming languages, such as Python, COBOL, BASIC, but have never attempted to learn how to write code in those languages. What made LISP so difficult for me was on how the entire language was built around the notation of lambda calculus. From my understanding, Lambda Calculus is the notation of binding an abstraction to a variable. The table below is what I used to try to understand the syntax of LISP through lambda calculus notation.

*Rules of the simplest form of Lambda Calculus*

| Syntax | Name | Description |
|---|---|---|
| $x$ | Variable | A character or string representing a parameter or mathematical/logical value. |
| $(\lambda x.M)$ | Abstraction | Function definition ($M$ is a lambda term). The variable $x$ becomes bound in the expression. |
| $(M\ N)$ | Application | Applying a function to an argument. $M$ and $N$ are lambda terms. |

*Extracted from: https://en.wikipedia.org/wiki/Lambda_calculus*

From the above “rules” of lambda calculus, we might get an expression such as: (λx.λy.(λz.(λx.z x) (λy.z y)) (x y)). I still do not completely understand how to interpret the expression shown, but I do understand that each abstraction is tied to an application, and it reminds me of prefix notation (+ab), which means (a + b). As stated previously, these expressions are what heavily influenced LISP, which is an acronym for LISt Processor. The language’s full name ties easily into its main data structure, list.

Lists are very important to the structure of the LISP programming language, this is because in LISP, everything is a list; or at least that is a common misconception of the language. The reason I have stated that is because I had a very difficult time trying to find concrete information about LISP and some sources agreed with that statement and some disagreed. Instead of learning about LISP, I would get information pertaining to some of the more common “dialects” of LISP, such as Common LISP, Scheme, Racket. There are around thirty-five different dialects of LISP (List of Lisp-family programming languages, 2022). I could have gone to the library or bought a book to learn about LISP, but I instead used the greatest tool mankind every created, Google. Joking aside, internet resources have always been my desired medium for gathering information, both for ease of access, potential vastness of information, and if relevant, different perspectives. I only try to gather information from credible sources, such as higher education institutions, publications, etc., but this method did not produce the desired information. Because LISP is not commonly taught and/or because of the many different dialects, I am not satisfied that all of the information that I collected into my presentation is strictly a feature of LISP or just a common derivative of it.

LISP is an expression-oriented programming language. This means that each statement in the language is an expression, which must return a result. It is hard for me to put into a concrete and detailed explanation but what this means is that any function, assignment, operation, etc. must return a value. This also ties into its main data structure, list, and even further, its design around lambda calculus. Symbolic expressions (S-expression) are defined as “syntax that mirrors the internal representation of code and data” (Lisp (programming language), 2022). I still cannot visualize what an S-expression is but do understand what an expression is and how it works. From my understanding, an expression is basically a list where the function/operation is first, and its arguments preceded the function/operation. To me, it feels like I am just learning how to drive a car and know how to use it but do not know how it works internally. One interesting aspect of LISP that I found is that since the source code of LISP is made up of lists, the programming language itself can manipulate its own source code. This allows programmers to create macros, which can be used to create new syntax in LISP. I am not familiar with how macros work in programming languages but have used them in video games, where they are used to perform some repetitive tasks, which may or may not be what I am trying to connect it to in relation to LISP. Regardless of my limited knowledge of macros, just the idea of creating new syntax in the language is very interesting.

Since its development in 1959 by John McCarthy, it has been used to pioneer many of the ideas in computer science that we still use to this day, such as: Trees, Conditionals, Dynamic Typing, Recursion, and many others. I am sure that it is only because of how old it is that the language was used to pioneer many of these ideas. It does show however that the language itself is influential. What is more impressive to me is the fact that it is routinely recommended as the preferred language for A.I. research. A.I. research is stated to have truly started around 1956 as an academic field in Dartmouth College. It is not easy to tie the fact that LISP is the preferred language other than its relative closeness in time. Modern A.I. was stated to be inspired by the thinking that human intelligence was just the process of mechanical manipulation of symbols (History of artificial intelligence, 2022). This led me to think that the underlying structure of LISP, S-expressions, and lambda syntax, is why it is the preferred language to use. This topic might not be needed in my evaluation of this language, and might even just be rambling on my part, but it did truly make me question why this language was so highly recommended. Some questions that I had were why a more "modern" language was not recommended, or even why a completely new language was not created just for this field of research. I still cannot answer these questions completely but can connect to the reason why it might be used, symbolic programming. During my research for this term paper, I have learned that new programming languages were created for A.I. research but not for "true" A.I. research but for machine learning; not sure if the two are different but wanted to point it out.

Since I am familiar with C++, pointers are something that I was taught about; although not as thoroughly as what was taught in this class. LISP uses pointers, but not in the way that I am used to. Because LISP is basically a language made up of lists, and lambda calculus uses paring in its syntax, S-expressions can be thought of as a list of pairs, to my understanding. Each element points to the next element. An example I can think of is a variable name pointing to a value and that value pointing to NIL, that satisfies the pairing requirement. I state in my presentation that I consider this to be a language of pseudo-pointers because of the indirect way that we can create pointers in LISP. I understand that my explanation is lacking, and that LISP does use pointers, but I could not figure out how to make an example that is easy to understand. Doing more research while writing this paper led me to a site that shows a LISP program using pointer syntax that is similar to C/C++ but is not used in a direct way. It defines a foreign variable to hold the address of a foreign callable. To me, it doesn't look like it is built-in syntax of LISP and do not really understand what is going on but do know the concept of a pointer holding a memory address. This is the only example of pointers that I could find specifically for LISP and not for one of its dialects. The one I showed in my presentation is for Common LISP.

*This is the syntax for the pointer of the program*

```
int(*FuncPtr1)(int, int) ;
int (*FuncPtr)() ;
```

The rest of the program I believe is used to make the callable object, which I guess is what is used to return an address. They also used two examples, one for Win32 and one for Linux (How do you pass a pointer to a Lisp function to a calling program?, n.d.). This is the struggle I had while doing research for this programming language. The website of the program can be found in the references section.

For my demo program, I decided it would be fun to create a sudoku solver program. I understood the logic of the algorithm and thought it would be easy to implement; I was wrong. If I did this in Java or C++, it would have been easier and quicker to do, but I am sure that was one of the main points to this semester long assignment. The handler and print functions were easy to set up. My stress came about when trying to setup the actual logic of the algorithm. The first issue I had was in the differences between "let" and "setq" keywords for LISP. "let" creates local variables and "setq" sets the value of existing variables. I tried to use "let" and "setq" for assigning variables to my array but would keep getting errors when trying to compile it.

Eventually I got it to work using “setf.” After going through the presentation and writeup of this paper, I began to think that maybe all of my errors were results of the complier that I was trying to use at that time. I still cannot figure out how to get any of the compliers that I had downloaded to run correctly. I can run my program on an online complier, but my partner was able to run his from the windows command line, not sure how he got it to run. The other issue I had was in trying to use file input to set my sudoku board to solve. I also tried to write the results to an output file. After trying to get the various compliers to run on my machine, I decided to just hard code a test sudoku board, just to see if my program will work. I was running out of time to try and rewrite my code to get the file input to work so I just kept the test board in my program for the demo and continued to work on my presentation slides. By Sunday of the week before my presentation date, I decided that I had written enough for my portion of the presentation and that my program would be enough just to demonstrate the language. About an hour after I decided that I was done, my partner notified me that he would begin working on his portion of the presentation. I would have liked to have had finished the slides and demo program by Saturday to allow time for rewriting, restructuring, recoding, and reviewing but my partner did not get started on it until Sunday afternoon; By that time I was convinced that I would be presenting by myself.

This semester long assignment was one of the hardest assignments that I have had in my academic career. It was more than just a “little” difficult in trying to learn a new programming language, one that is nearly alien to me. I had a friend in class presenting the PERL programming language and was able to follow along relatively easily and understood the syntax and how to use it. It was not because the presentation was well structured and informative, it was because of how similar it was to programming languages that I was already familiar with. In contrast, learning LISP was like learning calculus for the first time in high school, which for me was in 2005, but I had someone guiding me through it whereas I had to learn LISP without that guidance. For me, I can see two main issues that I had with this language. First was the fact that nearly all information I had found was for programming languages that were derivatives of LISP, such as CommonLISP, Scheme, Racket, etc.. The second problem was more personal, I could not understand the “flow” of the language. Java was difficult for me because it was my first introduction into coding, but not so difficult that I could not understand what was happening. I remember seeing many posts about how LISP is a nightmare of parentheses, with a common rewriting of its acronym to something like Lots of Irritating Superfluous Parentheses or Lost in Stupid Parentheses. I feel the same as these humorous rewriting of the acronym is but at the same time, I have more respect for people who know how to use this language and its derivatives. I am also interested in how a program in LISP can rewrite its own source code and maybe delve deeper into why it is highly recommended for A.I. research. Overall, I am happy that this assignment is done, but I am also more curious as to why other languages were created beyond the simple “because I wanted to” answer.

Hopefully this paper satisfies your requirements for the term paper assignment, and I wanted to thank you for your lectures and your help. What I learned in this class has made me more informed about the intricacies of programming languages and why it is important to understand why a PL was designed and how it might be better for a given task.

## REFERENCES

Wikipedia Contributors. (2022, 10 May). *History of artificial intelligence*. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/History_of_artificial_intelligence

Unknown Author. (n.d.). *How do you pass a pointer to a Lisp function to a calling program?* Retrieved from LispWorks: http://www.lispworks.com/kb/404d1bad7fe771f3802568c400540b50.html

Wikipedia Contributors. (2022, April 7). *Lisp (programming language)*. Retrieved from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/Lisp_(programming_language)

Wikipedia Contributors. (2022, April 27). *List of Lisp-family programming languages*. Retrieved May 10, 2022, from Wikipedia, The Free Encyclopedia: https://en.wikipedia.org/wiki/List_of_Lisp-family_programming_languages