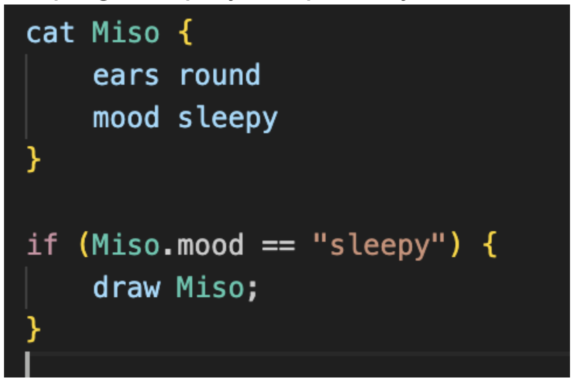
Thomas Williams
CSC402
11/15/2025

Project Proposal

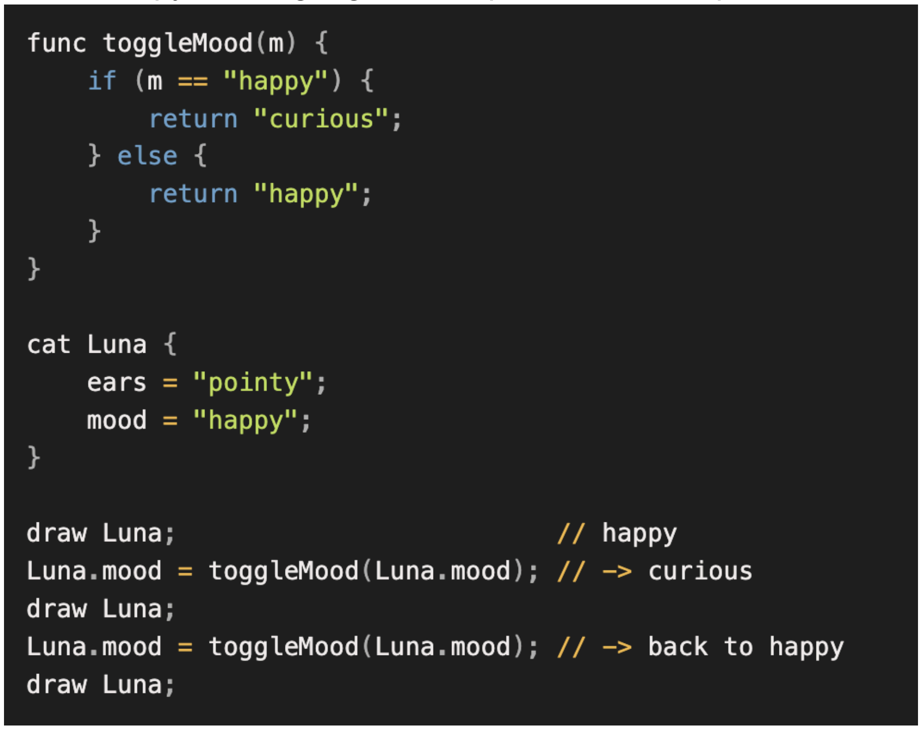
For this project I would like to create a programming language implementation called Cat ASCII Description Language, or “CADL” for short. The way CADL would work is you write a program that describes a cat. This would involve features like traits, moods, and behaviors. The language then turns said program/code into ASCII cat art. Unlike the C-like language which is more of a general purpose programming language handling computations, conditionals, and regular programming logic. CADL would be more like a domain specific language as it will be built specifically to describe cats and generate the following ASCII cat representation as its output. Making it different from general computing. The programming language implementation design of CADL would include tools and techniques learned throughout the course by utilizing a lexer, parser, grammar rules, AST construction, and an interpreter. Although the purpose of the language sounds simple underneath it will still be a complete structure built using what we learned. The grammar will define how cat declarations, traits, and statements are written. The lexer will tokenize the source code which will then go to the parser which as we know will use the grammar rules and build the AST structure. Finally with the use of our interpreter we will get the ASCII output.

Using the C-like language implementer from lectures CADL will be built using this as a framework but it changes the purpose of that system completely redirecting it into a domain specific language. Instead of handling the general computation like the C-like implementer it will use the same core pipeline to interpret the cat description programs fed to it. To go deeper into this you can declare cats, give them specific traits like ears and mood, and then use statements and logic to update those traits or draw the cat in ASCII form. Through modification of the lexer, parser, and new AST structure a big part of it will be adding new grammar rules and AST nodes. The structure will have a similar feel but every construct is tied back to the idea of programming a cat. Down below is an example of some acceptable programs that use cat declarations, attributes, one shows a function call. This shows the complexity of what the grammar will be and how it is not as simple as CADL may look from the outside as it behaves like a real interpreted language rather than just a plug and play template system.


```
cat Miso {
    ears round
    mood sleepy
}

if (Miso.mood == "sleepy") {
    draw Miso;
}
```

This example will confirm Miso is a valid cat with valid traits, check the conditional, and since the mood is sleepy the language will output the ASCII representation of Miso.



```
func toggleMood(m) {
    if (m == "happy") {
        return "curious";
    } else {
        return "happy";
    }
}

cat Luna {
    ears = "pointy";
    mood = "happy";
}

draw Luna;                            // happy
Luna.mood = toggleMood(Luna.mood); // -> curious
draw Luna;
Luna.mood = toggleMood(Luna.mood); // -> back to happy
draw Luna;
```

This example shows CADL's ability to support functions, parameter passing, return values, and runtime evaluation of expressions. The toggleMood function takes the current mood as input and returns a different mood based on conditional logic. By repeatedly calling the function and reassigning the cat's mood between draw statements, the program demonstrates dynamic trait updates and function-driven behavior.

When it comes to a timeline for CADL: (Starting as soon as the proposal is approved)
Week1: Plan on designing the full grammar, token list and planning out the AST node types needed for cat declarations, attribute access, draw statements, and function calls.
Week2: Modify the lexer to support CADL tokens, implement new parser rules, and ensure the AST builds correctly. This will also contain tests for the lexer as well to ensure proper tokenization.
Week3: The implementation of a symbol table for cat objects will be addressed, I will add runtime support for accessing and modifying cat traits and create the interpreter logic.
Week4: Build the ASCII rendering part of the system using trait values and then run various programs to ensure there are no errors, debug, and cleanup.

The resources needed for this language implementation are past lecture notes and program examples to use as a framework and basic structure. Python as this is what the CADL system will be designed using. ASCII templates to represent the ears, moods and other traits that will be mapped to the rendering part.