Thomas Williams
CSC402
12/14/25

Final Report

For this project, I implemented a programming language called Cat ASCII Description Language or **“**CADL” for short. CADL is a domain-specific language designed to describe cats using traits such as mood, ears, body type, and other characteristics, and then render those descriptions as ASCII cat art. Instead of being a general-purpose language focused on computation, CADL is built specifically for describing and visualizing cats, making it fundamentally different from traditional C-like languages. A CADL program consists of declarations that define cats and their traits, along with statements that control when and how those cats are drawn.

Although the concept of the language is intentionally playful, the implementation follows the same structure as a full programming language as discussed throughout the course. CADL includes a lexer, parser, grammar rules, abstract syntax tree construction, and an interpreter aided by a ascii rendering file. The grammar defines how cat declarations, traits, expressions, and statements are written. Source code is first tokenized by the lexer, then parsed into an AST using the grammar rules, and finally executed by the interpreter with the ascii renderer to produce ASCII output. The language supports features such as cat assignment, conditionals, loops, and functions, allowing cat traits to be dynamically modified at runtime before rendering. Overall, CADL demonstrates how a complete language implementation pipeline can be adapted to a domain-specific application while still supporting meaningful programming constructs. Below is an example program for CADL.

```
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;
```

Implementation:

The implementation of CADL follows the standard interpreter pipeline presented throughout the course (as mentioned above it consists of a lexer, parser, etc). The system is implemented in Python and is structured as a collection of files, each responsible for a distinct phase of execution within the pipeline. This design makes the flow of source code through the language easy to trace, from tokenization, through the other various phases, and finally to the ASCII output. The frontend of the language is handled by the lexer and parser. The lexer first tokenizes the source code and converts it into a stream of tokens which represent keywords, operators, and even punctuation which is specific to CADL. These tokens get passed to the parser which is driven by the CADL defined grammar. This then constructs an abstract syntax tree which can be printed out on its own to see the hierarchical structure of the program. The AST developed by the parser passes on to the interpreter. The runtime behavior for CADL is also handled within the interpreter phase as well as the symbol table.

The symbol table handles any related scoping to the program and is based very closely on the scoped symbol table system used in the Cuppa3 interpreter shown in class. Cat objects get stored simply as dictionaries. By keeping the symbol table implementation this way, the complexity of trait updates and cat-specific behavior is handled within the interpreter logic. The interpreter traverses the abstract syntax tree and executes nodes according to their type. This includes evaluating expressions, handling control flow such as conditionals and loops, and executing function calls with proper scope management by pushing a new scope to the symbol table. This then gets popped once the function returns allowing CADL to support parameter passing and local state while still operating on shared cat objects stored in outer scopes. WHen all said is done and a draw statement is encountered the interpreter retrieves the cat object from the symbol table and passes its current trait dictionary to the ASCII rendering component.

Challenges:

The most challenging part of implementing CADL was the ASCII rendering system. While generating ASCII output may appear simple, managing consistent formatting while supporting multiple traits quickly became complex. Each trait affects a different part of the output, and combining them in a way that always produced a readable and well-aligned cat required careful handling. This became more difficult when introducing moods, since moods act as higher-level preset, I ran into an issue of it overriding certain traits while still allowing other custom traits to remain in effect. Balancing the overrides of mood while still supporting user-defined traits required additional logic and iteration within the interpreter, since the interpreter ultimately drives the rendering calls.

Examples of CADL:

The following examples are used to demonstrate that CADL works as intended and supports the core features of the language. Each example focuses on a different part of the implementation, starting with basic cat declarations and drawing, then showing how different moods affect the ASCII output, and finally demonstrating runtime behavior using control flow and trait updates.

Together, these examples show how CADL handles trait assignment, mood presets, and dynamic execution while producing ASCII output.

```
cat Miso {
    mood = "happy";
}

draw Miso;
```

```
    ^   ^
= ( ^w^ ) =
Miso
```

```
cat Sleepy  { mood = "sleepy"; }
cat Happy   { mood = "happy"; }
cat Angry   { mood = "angry"; }
cat Loving  { mood = "loving"; }
cat Curious { mood = "curious"; }
cat Excited { mood = "excited"; }
cat Sad     { mood = "sad"; }

draw Sleepy;
draw Happy;
draw Angry;
draw Loving;
draw Curious;
draw Excited;
draw Sad;
```

```
   /\_/\_
- ( -.- ) -
Sleepy
    ^   ^
= ( ^w^ ) =
Happy
  (o   o)
~ ( ¬x¬ ) ~
Angry
   /\ /\
= ( *3* ) =
Loving
    ^   ^
= ( o.o ) =
Curious
  /\   /\
= ( OoO ) =
Excited
   /\_/\_
- ( u_u ) -
Sad
```

```
cat Miso {
    mood = "excited";
    ears = "pointy";
    tail = "curled";
}

while (Miso.mood != "sleepy") {
    draw Miso;
    Miso.mood = "sleepy"; // change condition
    Miso.mouth = "None";
    Miso.tail = "fluffy";
}

draw Miso;
```

```
   /\   /\
= ( OoO ) =
       ~~)
Miso
   /\_/\_
- ( -.- ) -
       ~~>
Miso
```

Overall CADL was a project I really enjoyed doing as it allowed me to combine two things I love, coding and cats. Working through the full interpreter pipeline and turning it into a domain-specific language really helped me better understand the ideas and concepts covered in the course lectures, especially how all the different pieces of a language implementation fit together. While the language itself is intentionally simple and playful, the underlying implementation reflects the structure of a real programming language. CADL successfully met its goals of allowing users to describe cats through a program and see those descriptions come to life as ASCII art, and it was especially satisfying to experiment with different traits and watch the cats change right before your eyes.