

# **Compilers Project**

Joshua Anderson

Gianforte School of Computing, Montana State University

CSCI 468: Compilers

Carson Gross

Spring 2022

## **I. Program**

See /capstone/portfolio/source.zip file for all source code and tests.

## **II. Teamwork**

The team consisted of myself, Philip Gales, and Jesse Russell. Majority of the project was done by myself throughout the semester though. The part where the group came in was with the testing phase. We each chose one of the others' compilers and created test cases to ensure they functioned correctly. We also each documented the functionality of the compilers and had each other look over those documents. The team communicated through Discord and worked together to make sure we all had solid final projects to submit in the end. The entire project took over eighty hours to complete and the group portion was at most six hours.

## **III. Design Pattern**

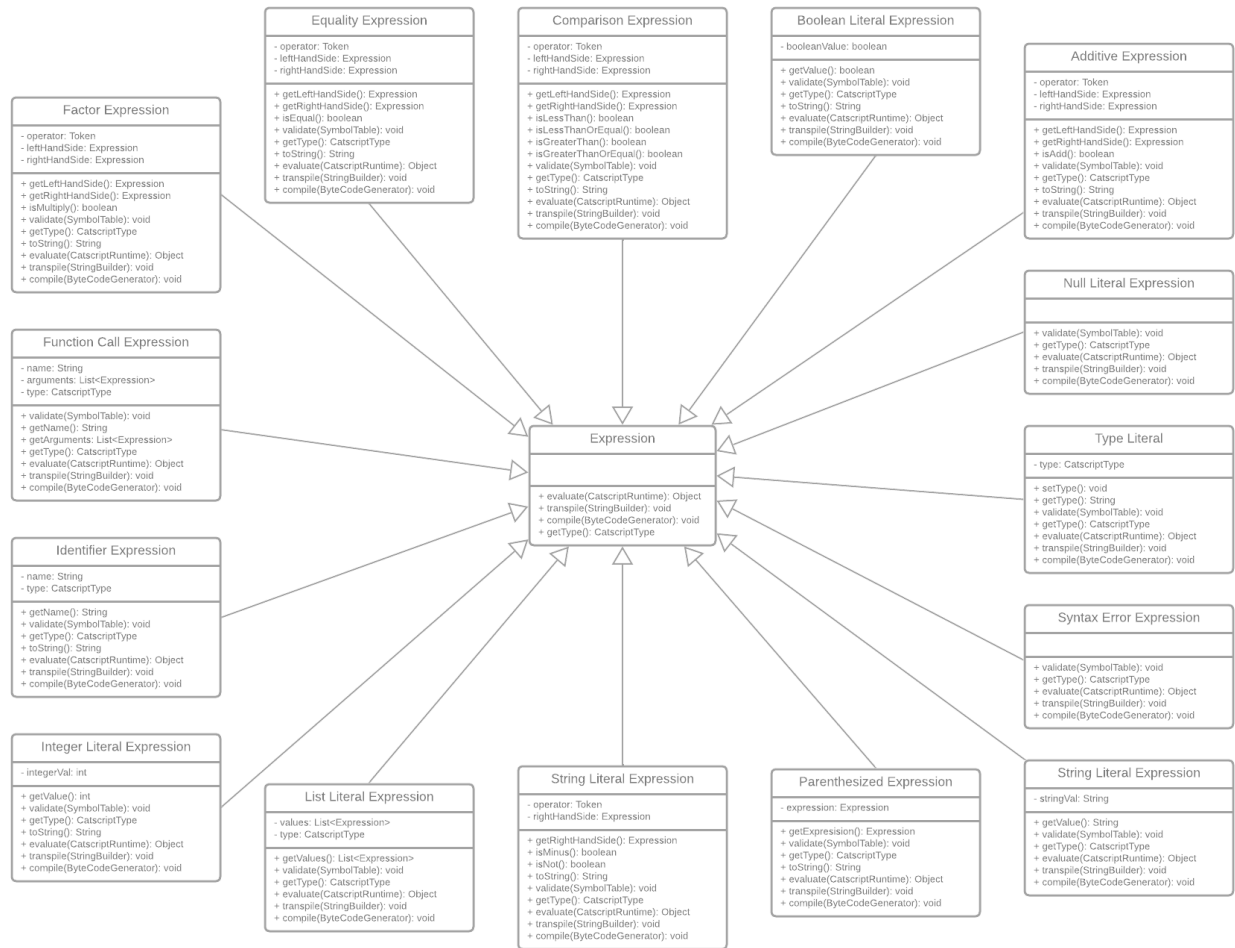
The design pattern that was used for this project was the Memoization Pattern. This design pattern ensures that for the same inputs, a specific method won't execute more than once by storing the results in a Hash Map. This was implemented in the CatscriptType class on the getListType method in order to reduce the amount of times a new ListType is created. The Memoization Pattern helps reduce redundancy and increases efficiency which is why it was chosen for this project.

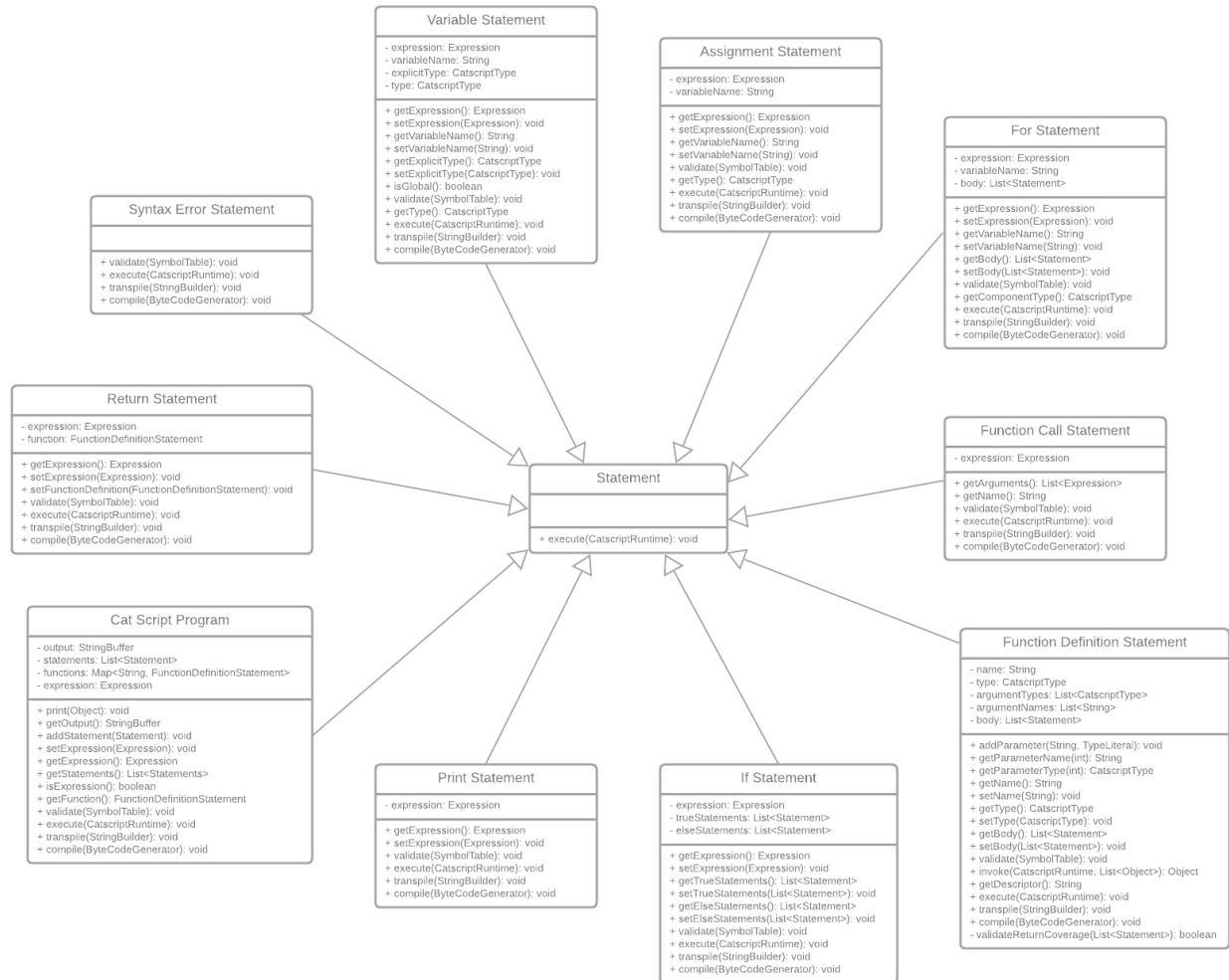
## **IV. Technical Documentation**

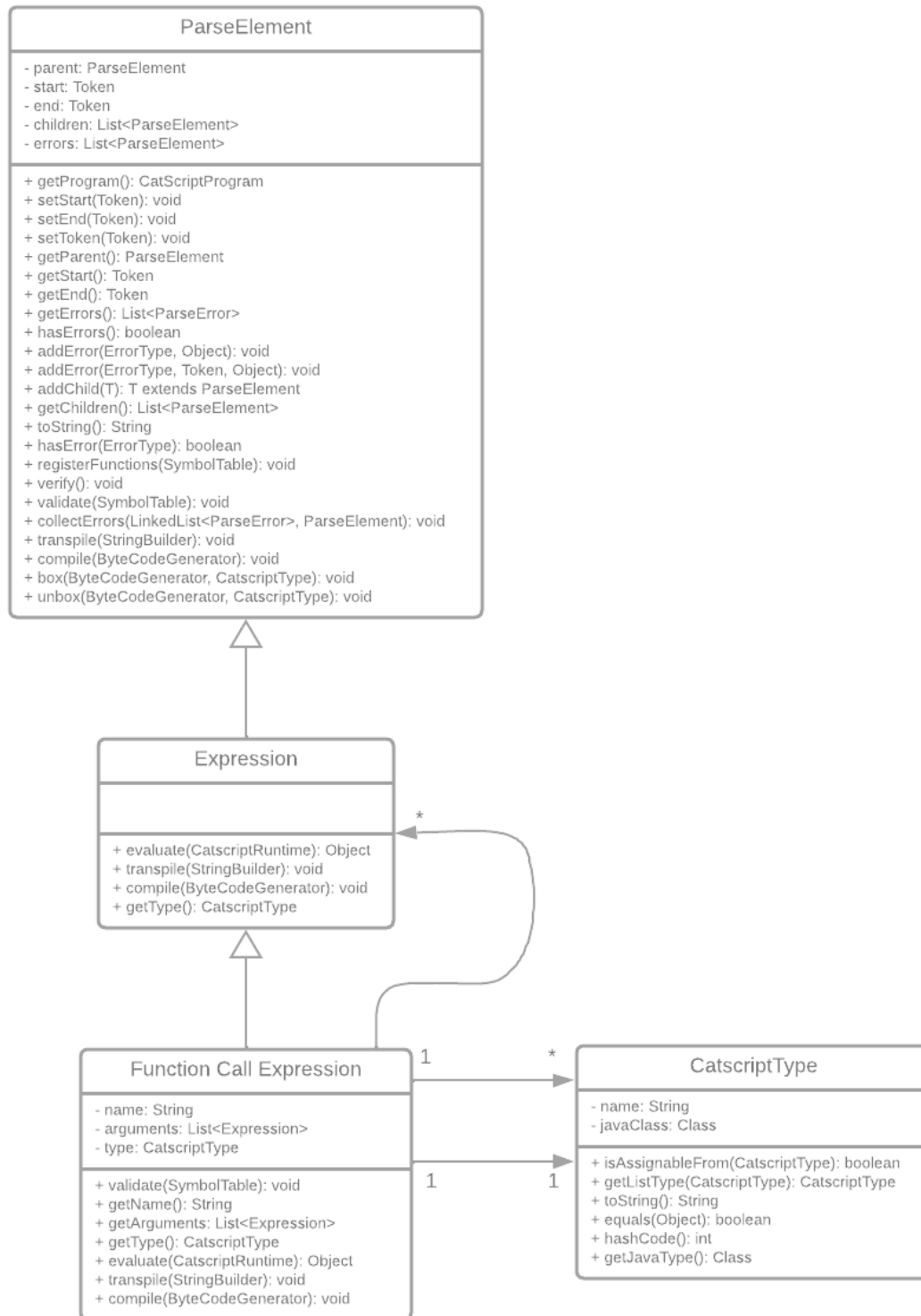
See /capstone/portfolio/Catscript.md for the technical documentation.

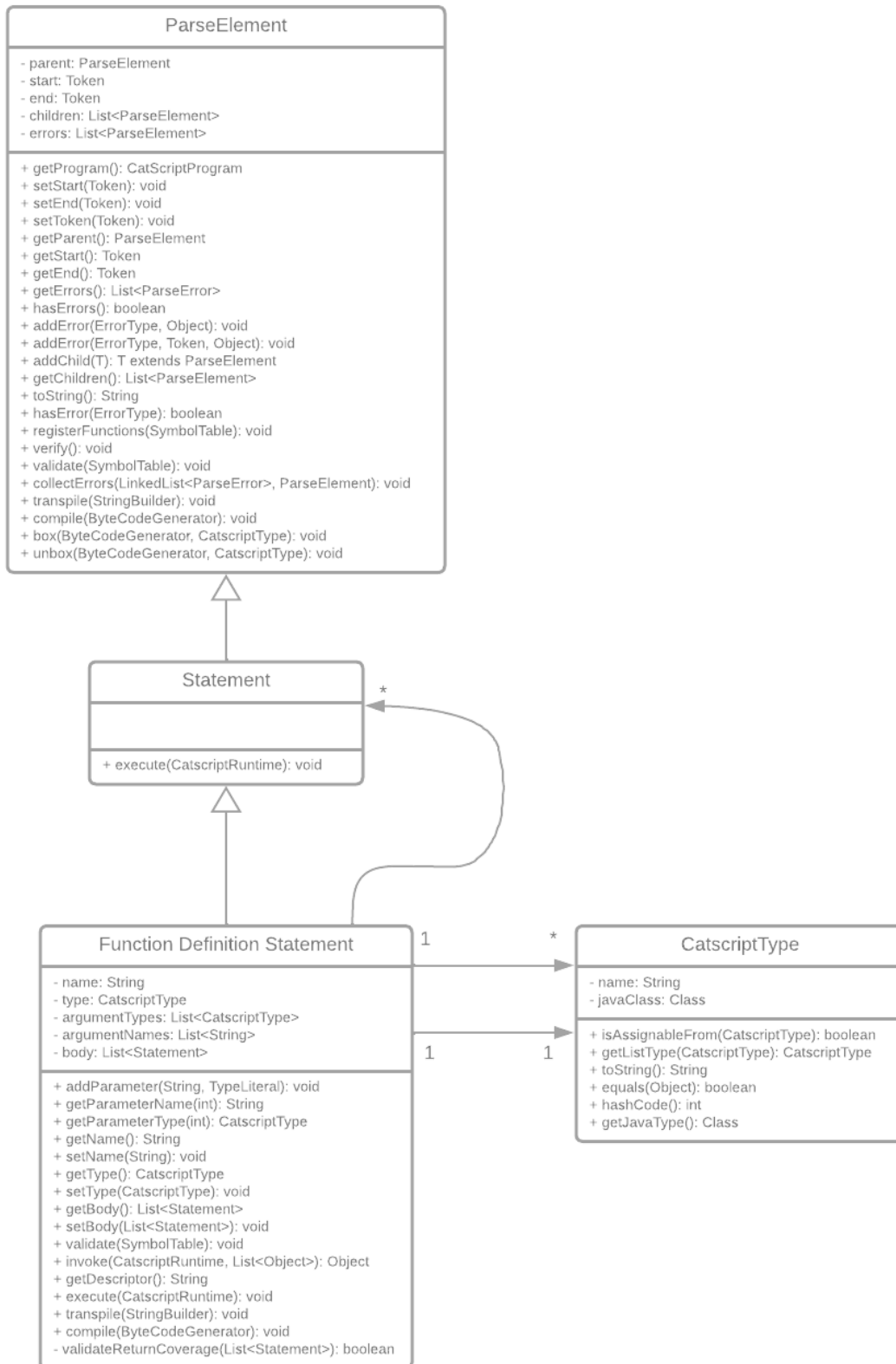
## V. UML

Figure 1

*All Expression Types Class Diagram*

**Figure 2***All Statement Types Class Diagram*

**Figure 3***Expression Class Diagram*

**Figure 4***Statement Class Diagram*

## VI. Trade-Offs

Throughout the design process of this project, there were many times where decisions had to be made that lead to different trade-offs. Here are some of those decisions:

- 1. Recursive Descent vs. Parser Generator :** The first choice I had to make was the approach to the compiler. Both Recursive Descent and Parser Generator have been used throughout the development of all programming languages. Parser Generators are more commonly taught in the academia world, but Recursive Descent finds itself more apparent in the industry world nowadays. I ultimately decided on the Recursive Descent method because it's extremely efficient and expresses the natural recursive nature of context-free grammars more obviously.
- 2. Test Driven vs. Scrum:** The other decision I had was what the life cycle approach was to be in order to most effectively accomplish my goals. Of the many potential methods, I determined that either Test Driven Development or Scrum would be best for the desired outcomes. Ultimately, I ended up on Test Driven Development due to the more stable approach it took compared to Scrum. With the Scrum life cycle approach, you have to work on different parts of the project all the time which wasn't something that sounded appetizing especially when I was learning how to do everything as I went. Test Driven Development has a very clear beginning to end approach that also allows you to ensure that the functionalities that you want to be a part of your program are accomplished effectively. It also means that the code is well documented since the tests can show very clear use cases of everything. It's because of those reasons that I chose Test Driven Development over Scrum for this project.

## **VII. Test Driven Development Life Cycle Approach**

To best ensure that I had a consistent and organized approach with a clear beginning to end process, I decided to use the Test Driven Development life cycle approach. This framework emphasizes testing your code over and over again in order to accomplish your goals. It does an excellent job at interweaving coding, testing, and designing all together. Some benefits of using the Test Driven Development life cycle approach are that it helps eliminate unnecessary coding by reducing the amount of debugging and design time, it is extremely easier to refactor code as long as tests remain passing, and it makes your code well documented since each test shows specific use cases. There were also some downfalls to using the Test Driven Development life cycle approach. If the tests had bugs in them, then your code would have bugs in them. If requirements changed, then all your tests would have to change to fit the requirements. This can be really time consuming. But even with these downfalls, I ultimately decided that the Test Driven Development life cycle approach was the one I wanted to use.