Montana University Bozeman (CSCI 468)

Compilers Capstone

Professor Carson Gross

Gregory Hill, team member 1
5-6-2022

Section 1: Program

Source Link

Section 2: Teamwork

This project was completed with contributions from two team members, myself, and my team member. I was designated the primary coder of the project, and my team member was designated the code tester of the project. My primary contribution to the project was the coding of the compiler. This included the implementation of parsing, evaluating, and compiling Catscript code. As the primary coder, I worked about ninety percent of the hours spent on this project. My team member's primary contribution was generating three tests and the Catscript documentation. These tests were made to ensure the correct implementation of my code, and the documentation is an outline, as a getting started guide, of the Catscript programming language. As the code tester my team member worked about ten percent of the hours spent.

Section 3: Design Pattern

In this project, the design pattern of memoization was used. This pattern is in the file CatscriptType.java located within the "parser" package. The exact file path is as follows: "src/main/java/edu/montana/csci/csci468/parser/CatscriptType.java." Within this file memoization is implemented on the method "getListType."

The purpose of using this pattern is to eliminate the need to initialize a new list type every time the getListType method is called. Without the use of memoization, when the method is called, a list type must be initialized, creating the scenario where every time the program needs to access the type of a list, the program must spend resources and additional time creating the initialization.

When memoization is implemented in this method, it stores the already initialized types into a HashMap that can later be referenced. Now, when the getListType method is called, if the desired type has already been initialized, it exists on the HashMap and can be accessed and returned. Using the HashMap to store initialized types results in the program not having to spend additional compute resources and additional time to create a new list type every time the method is called, improving the speed and efficiency of the program.

Section 4: Technical Writing

Catscript Documentation Link

Section 5: UML

UML Diagram Link

Section 6: Design Trade-Offs

The most important design trade-off made in the creation of the compiler was the decision to use recursive descent to hand write the parser instead of using a parser generator. Using a parser generator would have saved some development time, as implementing the parser by hand is more time consuming, but it also would have drastically decreased the understandability of the program.

The main goal during the creation of this project, was to not only correctly implement the program and create a compiler, but also have the programmer understand how the compiler works. Using recursive descent in the parser section of the program allowed the coder to work step by step through the parsing processes, forcing them to understand what is happening instead of just using generated code. In addition, the generated code from a parser generator is most often unreadable and exceedingly difficult to digest, while the code made using recursive descent is much simpler and the flow of the program is much easier to understand.

Using recursive descent instead of a parser generator requires more work from the programmer initially, but because of that additional work, the rest of the compiler will be easier to implement because of the better understanding of the program obtained.

Section 7: Software Development Life Cycle Model

The development of this project was modeled around test driven development. Prior to implementation of the parser, evaluator, and compiler sections of the program, many tests were written to ensure the program is being implemented correctly.

This model of development greatly helped in the implementation of the program in this project. Because all tests were written prior to implementation I had a much better understanding what needed to be completed. Without these tests, I would have been struggling to implement one working feature of the program, most likely trying to get everything to work at once. With the tests to start from, implementing parsing, evaluating, and compiling became a step-by-step process that was easy to follow along.

Test driven development allowed me to work on this project with a better understanding of each method, each class, and how the compiler worked, most likely allowing me to complete this project earlier and with more knowledge of compilers.