CSC 413 Project 2 Documentation

Summer 2023

Wing Lee

920558688

CSC-0413-01

https://github.com/csc413-SFSU-Souza/csc413-p2-ayanamesu

- Introduction
- Project Overview

The goal of the project was to create an interpreter for a specific programming language. An interpreter is a software program that reads and executes code written in a particular programming language. The first step was to implement the RunTimeClass, which is a fundamental component of the interpreter. The next phase focused on developing the ByteCode classes. These classes represent individual instructions in the programming language and are responsible for executing them.

- Technical Overview

The objective of the interpreter project was to develop a reliable and efficient interpreter for a specific programming language. Throughout the project, several key components were implemented, tested, and integrated to achieve the desired functionality. To start off we have the RunTimeClass, this class was independent and did not rely on other classes. By prioritizing its development, we ensured that the core functionality could be established early on. The RunTimeClass serves as a foundation for other classes and plays a crucial role in the interpreter's execution. Then we have the ByteCode classes, which are responsible for representing and executing individual instructions in the

programming language. Then the virtual machine this class provide the necessary infrastructure for executing the ByteCode instructions. As the ByteCode classes were implemented, corresponding code was added to the VM classes to ensure their compatibility and enable proper execution.

- ## Summary of Work Completed

The summary of work completed is as follows I first worked on the RunTimeClass implementation because this class is the only independent class that did not rely on other classes. Then I worked on the ByteCode classes starting with the first one ArgsCode and all the way to WriteCode. While I wrote these bytecode classes, I also need to add code to the virtual machine classes. I also can't tell if they work or not so I head onto the Program class and code resolve address and after that I can see if Goto, label, Call and FalseBranch work.

- ## Development Environment

The Java version used is Java 20 SDK
The IDE used is IntelliJ IDEA 2023 1.2 Ultimate Edition

- ## How to Build/Import your Project

I first imported this assignment from GitHub repository and copied the HTTP link and then went to my Terminal and used the command git clone https://github.com/csc413-SFSU-Souza/csc413-p1-ayanamesu.git. Then I began building the project by opening IntelliJ IDEA and using the old project and select files then selecting Import Project from existing resources. Then select the root folder as the source root of the project > Create Project from existing sources > Enter in a project name then keep hitting next > then click finish and once done you should see the C symboles next to the files.

- ## How to Run your Project

To run this project you would first need to make sure the selected testing arguements files all have "DUMP ON" in them. The test arguements are: factorial.dump.cod/factorial.x.cod , fib.x.cod, functionArgsTest.cod. Then you would go to run/debug configurations and edit configuration, first select the "Interpreter class" and then in the program arguements you would enter in either of the 3 test arguements listed above. This is because the program arguements run off of the interpreter class.

- ## Assumption Made

Some assumptions to be made would be that I expected the test arguement to be true and correct and would display the correct results. I also assumed that the project given to me would work properly.

- ## Implementation Discussion

Implementation for this project, first we would start with the RunTimeStack class and work on the methods on the run time stack to test if the rts works we used a main method to verify the stacking results. next we head into the byteCode file and create bytecodes corrosponding to the test arguements which start with ArgsCode all the way down to WriteCode, alphabetically. While implementing these classes we also have to create the Bytecode interface which is a switch arguement that takes in args as the arguement. and inside the switch would be all the bytecode classes. Then we would head to the virtual machine this is where we input our get and sets basically going to the vm and executing the results. Then we would get to a hold as the program would not run without the resolve address inside the program method. To do this I made 2 passes the first one is a hashmap that stores label positions and then a 2nd pass which resolve address for GoTo, Call, and FalseBranch bytecodes. I also made an exception to see if one is not working properly then it would show which one is not working. Initially the program would not run eventually when we fixed the for loop to a while loop the interpreter class started working properly, then it was time to fix the dumping of the bytecodes as there were arguements that were placed wrong. Eventually I got majority of them except the Halt to work properly as you can see there is a "[ ] " under the Pop 3.

- ## Class Diagram

  Please see the documentation folder there is a Diagram png with all the folders that has classes inside of them such as ByteCodes, loaders, virtual machine and the Interpreter.

- ## Project Reflection

In my opinion this project was harder than the previous project, at first glance and while working on it as well. The difficulty was stem from the pdf because the pdf leaves many important tasks that we should do and too many instructions and some of them are very miniscule with no example outputs or examples at all.

Now diving deep into the project what I found most difficult was the dump() part because if I got that wrong then the bytecode classes are all wrong. In the project there are still some small errors such as the Halt dump because its dumping when it should not be dumping. I found that the bytecode classes themselves were pretty self explanatory I got to retouch on switches with ternary conditions.

- ## Project Conclusion/Results

Overall I found this project very difficult. I also went through many phases of emotions through out this project. At some points I had a mental breakdown because I couldn't get the project running, This is caused because of the (For loop) that was made during class. Once the professor helped me resolve that issue I finally had my hopes up and felt better as the project ran as I wanted initially. Before I got help though I spent days debugging and finding out why the project wouldn't run and then I also fixed many bytecodes along the way for factorial.x.cod file then after I got help I finally went and dived into the other test arguements.