MERCH

Language Tutorial and Reference Manual

Compilers - CSC 492 - Fall 2018

Dr. Paul Hinker

Ryan McCaskell, Lyndon Engel, Jeff Ross,
Joe Ceritelli, and Jimmy Hinker

Table of Contents

Project Plan

Team Meetings

Team meeting during the course of development are at 8pm every Monday. Each member will be expected to state the successes and failures from the previous week of development so that appropriate changes to the development plans can be made. Weekly meetings are the time for higher level planning and development so that the next week can be used for implementation or research. Additionally, code reviews and merges into appropriate branches will be conducted at the end of each meeting if applicable.

Additional team meetings may be scheduled throughout the course of development in order to help on crunch weeks where a particular checkpoint must be achieved.

Schedule

08/24/18 - Team MERCH name and group make up sent to Dr. Hinker
08/27/18 - First Team meeting, research for DSL ideas
09/03/18 - DSL agreed upon by group
09/10/18 - Translator, general format of semantics and syntax, python backend agreed upon
09/17/18 - Complete white paper for Review
09/21/18 - White paper to Dr. Hinker
09/25/18 - First basic MERCH test programs created to test against
10/02/18 - ANTLR java example program compiling
10/03/18 - Port of needed code from soft eng platform uploaded to repo
10/06/18 - Version 1 of language in repo, very basic simple 1 line of code
10/13/18 - Get basic tree navigation working in python
10/14/18 - Version 2 of language in repo, better naming conventions and more expandable
10/19/18 - Language parsing mostly complete through ANTLR
10/19/18 - Development environment installer complete
10/19/18 - Complete Language Tutorial and Reference for Review
10/22/18 - Language Tutorial and Reference to Dr. Hinker
10/24/18 - More complete test suite completed to account for all proposed functionality
10/29/18 - Complete the implementation for all operation functions
11/03/18 - Long workday, Implement tree walker, all dev branches merged into feature
11/04/18 - Use, and words complete. Get operations complete.
11/13/18 - Testing procedures updated. Merge of all get operations and tree parsing functions get, tag, nap, wrt, wrds, comet, use.
11/16/18 - Bug fixes for get, nap and parser fixed.

11/20/18 - Testing reorganization and scripting. File documentation and code reorganization. Master Merge.
11/27/18 - Data pulling optimization, Clean up output, Documentation, Bug fix for variables
12/01/18 - Documentation, practice for presentation, implement If in language
12/03/18 - Documentation, presentation

MVP

MVP will be the product turned into Dr. Hinker at the end of the semester. The MVP will include an easy to run MERCH interpreter that takes in a MERCH file and executes the code. The MVP features of a MERCH interpreter will include the ability to make comments, loop, delay execution, manually print comments to the console, set variables that contain lists of stocks, run basic data analysis functions which output data to the console and perform basic conditional operations. The MVP basic data analysis functions will include Moving Average, Percent Increase, Get High Price, Get Low Price, Get Open Price, Get Close Price, and Get Volume.

Software Development Plan

The software development plan is an Agile style development. A formal sprint system is not used, however a design for MERCH was agreed upon at the beginning of development, and weekly standup meetings dictated the next week's feature development. Standup meetings are held every Monday at 8pm. It is the responsibility of each member to dictate the successes and failures of the previous week. Based on the progress made of each team member, new feature tasks were assigned to each member weekly.

Software Requirements

The requirements of MERCH will be as follows:

1. Linux and Windows capable
2. Simple to run and operate
3. User is not reliant on external libraries or programs
4. Interpreted program files allow commenting
5. Interpreted program files allow looping
6. Interpreted program files allow conditionals
7. Interpreted program files allow variables
8. Interpreted program files allow basic stock analysis functionality
9. Interpreted program files allow user to print custom messages to console
10. Gracefully handles errors

Delivery Schedule

The delivery schedule will be as follows:

- September 21 - White Paper

- October 22 - Tutorial and Reference Manual
- December 3 - Presentation of MERCH
- December 3 - Final Paper and MVP MERCH language

Acceptance and Certification Testing

For acceptance and certification testing of the software requirements of MERCH, within the repository will be a folder named test_programs that contains all blackbox tests. Each software requirement above that relates directly to the operation of the software will have one or more tests that test the functionality of the requirement. Before the files are then merged with feature, two developers review and sign off on the codes segments for one last check. Then for one final assurance before the feature branch is merged with master, all developers do a final check off.

Code Maintenance

Code maintenance is conducted through gitlab hosted by the South Dakota School of Mines and Technology. Code will be controlled by having a Master, Feature and Developer branches. The Test Plan section of this document covers the acceptance testing for each branch tier. Developers create branches off of the feature branch and make a single new feature for the MERCH interpreter. Once a new feature meets the testing requirements, a merge request is created to merge into the feature branch. After the feature branch has made significant progress and is stable a master merge may occur. A merge into the feature branch will require approval of the entire development team and testing as approved by the person in charge of Testing and Verification.

Documents Maintenance

Documents are shared to the development team through Google Drive to allow for easier simultaneous editors. Final drafts of documentation is placed into the repository for public access.

Final Report Schedule

It is expected that the final document sections will be completed as followed:

- Language White Paper - Prior to 09/21/18
- Language Tutorial - Prior to 10/22/18
- Language Reference Manual - Prior to 10/22/18
- Code Listing - Prior to 12/03/18
- Project Plan - Prior to 10/01/18
- Language Evolution Document - Prior to 12/03/18
- Translator / Compiler Architecture - Prior to 11/01/18
- Development Environment and Run-Time System - Prior to 12/03/18
- Test Plan and Scripts - Prior to 10/01/18

- Conclusions - Prior to 12/03/18

Language Evolution Document

Future Expandability

Keywords
To follow the general goal of this product, the language can be expanded to include more names for commands. In example, the keyword “get” could be run with the names “get”, “run”, “execute”, or any name desired. This would assist in keeping the language simple so that certain keywords stick better with different people.

Features for advanced users
Additionally, support could be added to allow more advanced users to extend the program to include more stock handling functionality. This change would require a moderate amount of work, but would not be impossible, and would provide significant functionality to the language. This would require providing some kind of API to be used to retrieve information from the stock market. Functions already exist for easy access to data, so creating a way to pull information would be easy, but actually implementing new functionality into the program through an API would present the real challenge.

Variables
Another item that could be used to expand the system would be to add new types of variables outside of lists of stocks that are currently implemented. Adding other types of variables could make items like if statements more dynamic, and make print statements more customized. For example if one were to find the moving average of aapl, and wanted to see if it was greater than another company, they could do so with this. With the current outline of the system, adding this in would not be difficult.

System Design to allow for expandability

With the current set up of the language, adding new items such as new commands to parse data is not difficult and can be done fairly easily. In fact, most features that could be used to extend the language are fairly easy to implement with the current set up. Although there are exceptions with items like adding support for users to add their own commands as discussed above. The design of the language is fairly simple in that it is driven by the keywords at the beginning of a statement, and is further specified with items in the rest of the statement, with some throwaway terms to just make the statement look similar to English. The statements are parsed in the language by the first term, so adding something new is as simple as adding a new function to handle anything found with a certain keyword, making the language fairly dynamic to add to.

Translator Architecture

This code will run with python and ANTLR4, it will have an executable so nothing should need to be installed

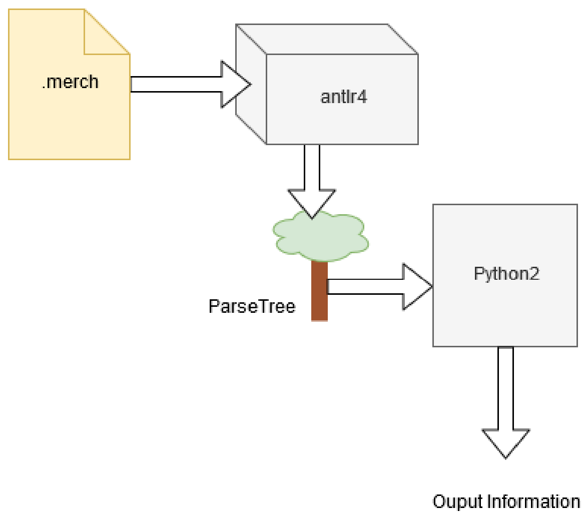
Interpretation

The DSL (MERCH) uses a natural language interface, as to allow inexperienced programmers to use this to quickly and repeatedly pull information from the NASDAQ. The system architecture will need to run as an interpreter to allow this. Running as an interpreter will help with error management.

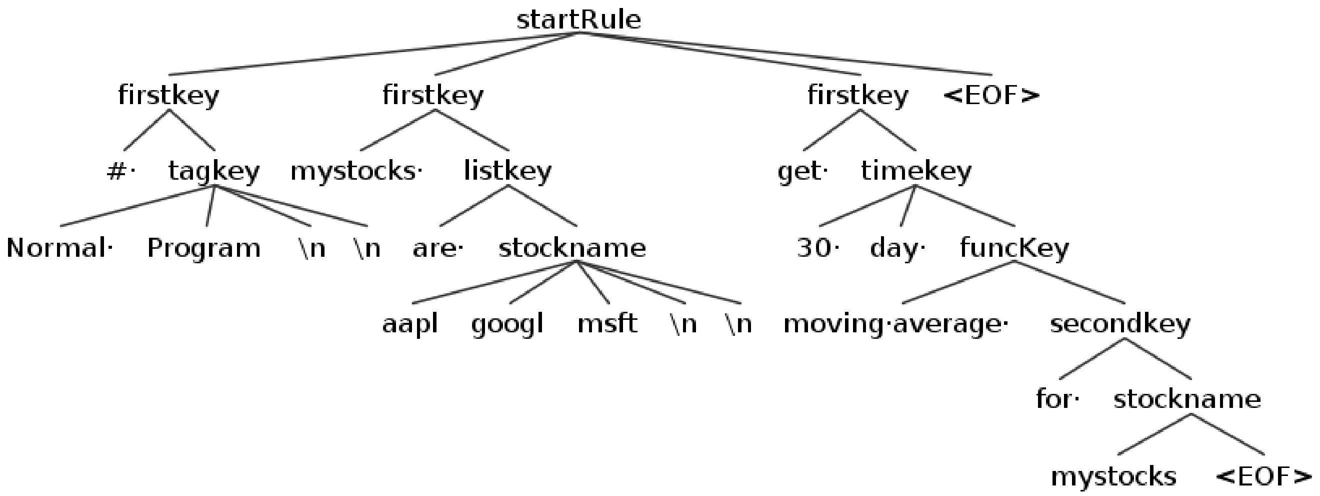
Tokenizing and Parsing

The file that is input to antlr will be parsed and tokenized into a parse tree for python to navigate and spit out the users information.

Antlr takes a .merch file and separates it into lexar rules and parser rules. It takes each line individually and looks for a way to complete the "*start rule"* which is defined as 0 - infin number of newlines, and then a firstkey which is broken down further into each of the individual parser rules. Each parser rule is made up of lexar rules and more parser rules.



This is an example of the parse tree built by Antlr4. Everything is built off of startRule, which is looking for firstkey, and then keywords off of that.



After Antlr4 establishes the tree it passes it to the python code which uses that to pull information from the NASDAQ for the user to see. We built a switch statement to account for each possibility off of the startRule.

Development Environment and Run-Time System

OS Requirements

The MERCH compiler is supported on Windows 10 and Ubuntu 16 or greater operating systems. The executable contains all required dependency, so nothing is required to install.

For development there is a bash script called “devinstaller” that will set up all required libraries and dependency in linux for a user to to begin development on the MERCH compiler.

Hardware Requirements

OS: Windows 10 or Ubuntu 16 +
Processor: Intel Core i3 or better
Memory: 2GB or more
Storage: 50MB

The Runtime System

The MERCH compiler uses antlr4 runtime for python2. This runtime system is built into the merch executable file, so the user doesn't have to jump through any hoops figuring out how to download python and antlr4.

Use of Existing Libraries

Antlr 4.7.1 - Used to generate the listener, lexer, and parser into python files.

Python 2.7 - Used to run the MERCH compiler

- Requests - used for pulling stock information from NASDAQ

Java 10 - Needed to run Antlr

IDE Requirements to write a program in MERCH

Any text editor will work for creating a MERCH program. Just be sure to save the file with the ".merch" extension.

Test Plan and Scripts

Types of Tests

We do a form of black box testing, ad-hoc testing, system testing and exploratory testing.

When Testing is Done

While developing code the developer should be testing to make sure the logic is correct. Black box testing is performed before a merge to make sure all the tests in our testing directory are passing. The black box tests are repeated on the feature branch to confirm that the merge was integrated successfully

Testing Responsibilities

Testing is the responsibility of the developer on the branches off of feature. The system tester will run the test_programs after the branch is merged.

Branches that require testing

Any branches off of feature are required to be tested against the tests in the test_programs directory. Feature must be tested after a merge.

Code Reviews

Code reviews are performed by at least two people that have not worked on the branch. Code reviews are then performed once again before merging into master.

Requirements for Product Release Branch

All code that goes into the product branch needs to be fully tested and approved by all of the developers on the team

Requirements for Features Branch

All code that goes into the Feature Branch needs to be tested by the developer and needs to pass all of the black box tests. There will be a code review before the code is merged into feature. During this time there may be ad-hoc testing that occurs if someone spots a potential issue

Developer Branches

Developer branches need to be tested by the developer(s), the black box tests related to the code changes need to pass. A code review of the changes must be made before the branch is merged into feature.

Code Listing

https://gitlab.mcs.sdsmt.edu/7308815/compilersFA18

Conclusions

What was learned

To start with, nearly every aspect of the material learned in class was used at some point to progress with the development of MERCH. Some of the more notable highlights include a large increase to the overall knowledge of regex and the ability to parse and analyze strings into tokenize-able pieces of information. Although past classes and assignments have had some work with regex and the systems behind it, a program like this requires a much deeper understanding that can be carried over in to a broad range of applications.

Additionally, the planning that is required to make a good clean language was more than any of the group expected. There are so many technical aspects that go into making a good language, and each step is just as important as the previous.

Lastly, project without design guidance allowed for more opportunity to be creative. Initially this freedom could be quite frustrating as there are a lot of possibilities for a DSL. This freedom is what allowed our team to make a lot of academic and creative progress this semester in regards to languages and their interpretation at a compiler level.

What could have been done differently

Obtain user stories from potential users outside of the project. Although we believe that we have made a good product, more market and individual research could have been conducted to make the product even better and more focused to user needs. Additionally, having a more concrete sprint schedule would have benefited the team to make more concrete goals as we were developing.

Was the scope of each person reasonably allocated

Project Manager - Joe Ceritelli

- Planned and managed the teams trajectory as the project was developed.
- Established team meetings and had a list of things to go over for each next sprint.
- Managed the merging of master and feature branches.
- Had Git locked down for when there were problems within the workings of Git

Language Guru - Jeff Ross

- Had a handle of python and the resulting parse tree from antlr4 to help and guide other developers when they had problems.
- When a big bug came up Jeff was quickest to to jump on and and fix the problem

- Established a coding practice for other developers to follow for easier to read code
- Had some great breakthroughs on the way functions were moving information around

System Architect - Jimmy Hinker

- Kept the parser up to date
- Designed and developed majority of the parser functionality
- Quickly and accurately provided debugging help in regards to the parser and its functionality
- Was concise and accurate when discussing fixes for bugs

Verification and Validation - Ryan McCaskell

- Had tests built for all the corners of the project
- Had tests built before the feature branch was complete to allow developers to test as they completed features.
- Quickly understood the pitfalls within the system causing system failure, creating concise testing
- Was available at all times for questions about testing, and anything else not understood
- Built automatic testing to run multiple things through quickly and efficiently

Systems Integrator - Lyndon Engel

- Developed the Dev installer for the group to use to avoid different environments and allow developers to avoid different bugs between systems.
- Built and adjusted the .exe for the language to be portable with just the use of a USB
- Configured compiler to accept MERCH files or direct input.
- Was always available for the code reviews, to comment and establish that the code would not break upon merging.

Is the project expandable for the future

MERCH language is built to be expanded. Expanding the code base python2.7 and Antlr4 is needed, as well as understanding of the two languages. Adding more functionality requires the developer to add keywords into the Antlr4 file, and adding functions to the Merch.py file. The MERCH file is functionally built so it should enable the developer to add without messing up other previously built functions. The long spin up to understand how things are working will be mainly done in the understanding of python and Antlr4.

With more resources what could have changed

With more time/resources/team members it would have been nice to create additional natural feeling sentences for users, so that a wider variety of communication methods could be used. Additionally, it would be nice to allow for a modular method of operations that a user can use. Although we were able to make several useful operations for a base user, many types of analytical methods of analyzing stocks exist, and providing a method for users to build or import operations outside of the language could help encourage its use. If a skilled user were to want additional functionality, currently we would have to hard code it into the language.

Additionally, adding a wider variety of data types and operations that can be performed on. This would allow users a greater ability to perform mathematical operations on obtained data and data that the user wishes to make comparisons on.