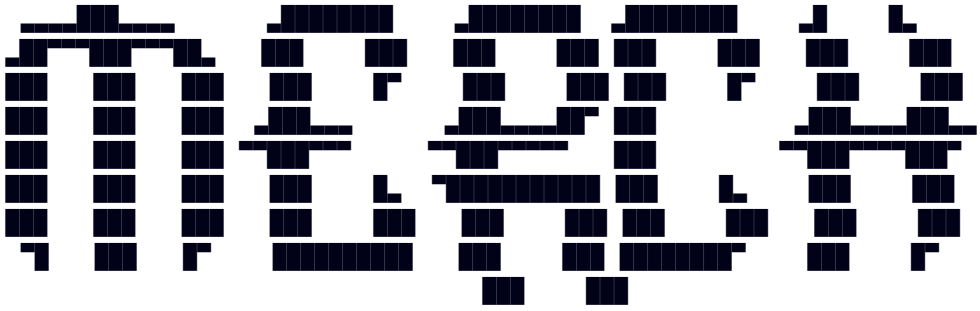

Language Tutorial and Reference Manual

Compilers - CSC 492 - Fall 2018

Dr. Paul Hinker

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

Table of Contents

Introduction

MERCH is an easy to use language to help make getting information on the stock market easier. The design of MERCH is to allow easy programming for individuals without a programming background, taking away the more technical aspects of a language. MERCH supports basic operations such as getting the moving average for specified stocks, and getting a price of a stock on a specified date. The language also supports comparative and conditional operations to decide whether or not certain lines should be run. Additionally, methods for repositioning in the program allow for lines to be repeated, and an await operator to set a time period between when these lines are repeated.

Installation

For development the MERCH language comes with a user dev environment script. This script will make sure that the correct installation of python exists on the user system, and setup any required environment variables. After installation, the user will then be able to work on new additions to the MERCH compiler.

On the user side of things all that is required for installation is pulling down the merch.exe from our gitlab https://gitlab.mcs.sdsmt.edu/7308815/compilersFA18. We have an executable for both Windows 10 and Ubuntu 16+ in the executable folder on gitlab. Both these executables do not require any additional dependencies, how to run them can be found in the next section.

Running Merch

Once you have pulled your desired executable from the gitlab navigate to the folder it is located. In windows you can run the .exe from powershell and in Ubuntu it can be run from the compiler. The command to run the compiler should like this:

For windows 10: ./merch.exe ./yourmerchprogram.merch

For Ubuntu: ./merch ./yourmerchprogram.merch

Where the second argument is your MERCH file. A MERCH file can be written in any text editor just make sure to save it as .merch.

Creating the Executable

The creation of the MERCH compiler executable is different for both Windows and Ubuntu. I'll start with Ubuntu first because it's the easiest. In the main directory of source code for the

MERCH compiler is a bash script called “installer.sh” run this and the executable for Ubuntu will be created.

To create the executable for windows you must first run the makefile in linux and then copy over the testingMain.py, progInfo.py, ParserV2Parser.py, ParserV2Listener.py, ParserV2Lexer.py, and operations.py files to a windows folder. You will also need pyinstaller for python 2.7 this can be installed using the following command in powershell:

py -2 -m pip install pyinstaller

After that is done you run the following command in the folder that contains the merch compiler files:

path to: \pyinstaller.exe --onefile .\testingMain.py .\progInfo.py .\ParserV2Parser.py .\ParserV2Listener.py .\ParserV2Lexer.py .\operations.py

The start of the command is the path to where the pyinstaller was installed this should be in your Python27\Scripts folder. After the command is run the .exe will be located in the newly created dist folder.

Getting Started

The best way to get started writing in MERCH is to write a quick program to perform basic operations to get a feel for the structure of a MERCH program. Start out by opening a text file and writing:

```
Get 30 Day Moving Average for MSFT
```

Save the file as firstProgram.merch. This produces a quick program to get a moving average for Microsoft. Run this file with the MERCH interpreter to see it’s output (See Running Merch). The statement can be changed up to create different operations. “30 Day Moving Average” can be replaced with other operations such as “Closing Price” or “Opening Price”. The next step is to write a multi line program using variables. Change the file to include the following:

```
MyStocks are MSFT, AAPL, AMZN
Get High Price for MyStocks
Get Low Price for MyStocks
```

This creates a new program that gives the most recent high and low prices for Microsoft, Apple, and Amazon. Experiment with this to create additional variables (similar to the MyStocks line) to get a feel of how to use variables. This exercise introduces the basics of MERCH and will allow a swift start with the language.

Structure of a statement

A statement can be a simple or compound expression that produces some output. A statement can be used to perform various operations on a stock, or a collection of stocks. The structure of a statement is as follows:

Get 30 Day Moving Average for MSFT, AMZN, GOOGL

This specific statement outputs the moving average for Microsoft, Amazon, and Google over the last 30 days. A statement will begin with the keyword “Get” followed by the name of an operation to be performed, such as “30 Day Moving Average” as seen above. The operation is followed by the “For” keyword which is then followed by the stock (ticker) symbols of the company, or companies, to have the operation performed on. Additionally, these stocks can be replaced by a variable name. The structure for declaring variables can be seen below.

Variables

A variable can be used to store the names of one, or several stocks to avoid having to re write the name of a stock repeatedly throughout a program. The two structures to declare variables are as follows:

MyStocks are AAPL, GOOGL, AMZN
Microsoft is MSFT

This allows any operation used on “MyStocks” to perform the operation on Apple, Google, and Amazon, and any operation used on “Microsoft” to perform the operation on Microsoft. Variables can only be defined as the stock (ticker) symbols of a stock or stocks. Using the company name will result in an error. A demonstration of how the above examples can be used can be seen below:

Get Opening Price for Mystocks
Get Closing Price for Microsoft

The For Statement

The “For” statement is used to specify which stocks an operation will operate on. “For” is typically used in combination with the “Get” statement. An example of its usage can be seen above in the Structure of a Statement section.

The Use Statement

The “Use” statement is used to jump to a specified point in the program as defined by a tag. “Use” can be used in conjunction with “if” to check a condition and determine whether or not it should jump to the tag.

The Get Statement

The “Get” statement is used to specify an operator to be used. “Get” is typically the first statement used in a line to specify an operation to be performed, used in combination with “For”. An example of its usage can be seen above in the Structure of a Statement section.

The Wait Statement

The “Wait” statement is used to wait a specified amount of time before performing the next operation. An example of its usage can be seen below:

```
Wait 1 hour 4 minutes 20 seconds
```

This statement is useful in combination with the “Use” statement to create a loop in which will produce an output every specified amount of time. An example of it’s usage as a timer can be seen below:

```
@ResetTimer
Get Price For WMT
Wait 15 minutes
Use ResetTimer
```

This program will get the price of Wal-Mart every 15 minutes.

The Are Statement

The “Are” statement is used as an assignment operation. Its primary use is to assign stocks to a stock list. The location of the operator will normally be between a stock list and a user’s designated stocks. An example would be:

```
MyStocks are AAPL, GOOGL, AMZN
```

This statement assigns Apple, Google, and Amazon stocks to the stock list MyStocks. The operation takes place from right to left.

The @ Symbol (Tags)

The @ symbol will set up a tag for the program to be able to jump to in the event of wanting to redo operations already written out. Upon jumping to the tag the program will run everything in the .merch file below the @ symbol location.

```
@runFromHere
Wait 9 days
Get 8 day moving Average for mystocks
use runFromHere
```

The tag defines the place the code will run from, and will run the command wait. After the wait has been resolved it will re-run get moving average for mystocks.

The If Statement

The “If” statement is used in conjunction with the greater than, less than, and equal to statements to determine if a command should be executed.

See greater than, less than, and equal to statements for examples.

The Greater Than Statement

The “Greater Than” statement is used as a comparison operation. For this statement it compares whether one value is larger than another. The result of the operation is a boolean value of true or false. An example would be:

```
If Get 30 day Moving average for MSFT is greater than 100 write Hello World
```

The Less Than Statement

The “Less Than” statement is used as a comparison operator to compare whether or not the result of an operation is less than a given value. In example, the following statement would write “Microsoft is down” to the terminal if the 30 day moving average for Microsoft was below 100.

```
If Get Moving Average For MSFT Is Less Than 100 Write Microsoft is down
```

The Equal to Statement

The “Equal To” statement is used as a comparison operator to compare whether or not the result of an operation is equal to a given value. In example, the following statement would write “Microsoft is at 105” if the 20 day moving average of Microsoft was 105

```
If Get Moving Average For MSFT Is Equal To 105 Write Microsoft is at 105
```

The Write Statement

The Write statement is used for displaying user defined print statements.

```
Write This is an output statement.
```

This line of code will output to the console: “This is an output statement.”.

Each Write command will be printed on a new line. Additionally, a user will be able to print user defined variables to the console as shown in the following code block.

Keyword Naming Convention

Keywords are all lowercase, with the exception of the first character can be lower or capital case.

Operations

Below are the current operations that are supported by MERCH.

The Moving Average Statement

The “Moving Average” statement can be used to pull the average price of a stock over a certain number of days. The moving average statement can be run with a default number of days or a specified number of days. If used as default, the moving average will be calculated for the past 30 days. Examples of the Moving Average operation being used both as default and as specified can be seen below:

```
Get Moving Average for MSFT
Get 25 day Moving Average for AAPL
```

The first statement will do a 30 day moving average, and the second will do a 25 day.

Percent Change

The “Percent Change” statement can be used to calculate the percent change in the price of specified stocks. Percent change is calculated as the percent change of the moving average of the past 5 days in relation to the moving average of the past 20 days. An example of the percent change operation can be seen below:

Get Percent change for Mystocks

Where Mystocks is a variable containing stocks

High Price

Gets the high price for the current stock market day, or the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the high price for the past specified number of days. An example for the usage of this operation is as follows:

Get High Price for Mystocks
Get 15 day High Price for Mystocks

Where Mystocks is a variable containing stocks.

Low Price

Gets the low price for the current stock market day, or the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the low price for the past specified number of days. An example for the usage of this operation is as follows:

Get Low Price for Mystocks
Get 20 day Low Price for Mystocks

Where Mystocks is a variable containing stocks.

Price

Gets the current price for the current stock market day, or the closing price for the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the closing price for the past specified number of days. An example for the usage of this operation is as follows:

Get Price for Mystocks
Get 30 day Price for Mystocks

Where Mystocks is a variable containing stocks

Volume

Gets the volume for the current stock market day, or the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the volume for the past specified number of days. An example for the usage of this operation is as follows:

Get Volume for Mystocks
Get 5 day Volume for Mystocks

Where Mystocks is a variable containing stocks

Closing Price

Gets the closing price for the current stock market day, or the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the closing price for the past specified number of days. An example for the usage of this operation is as follows:

Get Closing Price for Mystocks
Get 12 Closing Price for Mystocks

Where Mystocks is a variable containing stocks

Opening Price

Gets the opening price for the current stock market day, or the previous stock market day if the market is closed. Additionally, the day keyword can be added to print the opening price for the past specified number of days. An example for the usage of this operation is as follows:

Get Opening Price for Mystocks
Get 3 Opening Price for Mystocks

Where Mystocks is a variable containing stocks

Results

After execution of an operation command, the result will be printed to the screen.

Character Input and Output

All character input is done through a MERCH program file. MERCH uses python for interpretation, which means it accepts utf-8 characters. Output is printed to the console by using the Write statement or by using one of the operations that will print its results to to the screen such as Moving Average. See Write or Operations.

Variable Scope

The variables within a file are accessible throughout the entire file. However the variables are not accessible outside the file.