StockX

Jesse Van Marter jjv2121
Ravinarayanan Lakshmanan rl2857
Ricardo Martinez rmm2222
Sophie Lucy sjl2185

Introduction

The financial industry has been a perfect space for the use of programming languages because of the heavy reliance on mathematical calculation and accuracy involved in gaining successful returns. Technology has allowed people to be more precise in their trades and experimental in their strategies. Thanks to the amount of historical as well as real time stock market data publically available, there are infinite possibilities to the application of a financial programming language. By using such resources, we intend to build a language that can be used by beginner investors and extended to experienced individuals.

Language Description

StockX is a financial based language intended to give users an easy, abstract way to test and execute trading strategies. The language will feature an "account" data type which can hold money, risk preferences, portfolios, and the ability to make trades. "Portfolio" is another StockX specific datatype that will contain a collection of stocks, details of when they were bought and sold and can also have portfolio specific risk preferences.. "Stock" datatypes will be able to be filled with technical and fundamental information about a specific stock. This will allow users the option of using stocks historical data to backtest strategies or to keep track of trades in 'real time'. "Order" types will be executed by an account on a specific stock into a portfolio the account owns. This, combined with risk preferences on the account and portfolio levels, will allow users to manage capital at an account level while still keeping track of the performance and returns of each portfolio. StockX is meant to have the ability to be leveraged realtime account tracking and execution (letting users have a transparent view of what strategies are working for them and at the same time assuring them their risk preferences are followed on an account and portfolio level) and also be used to optimize strategies and risk preferences on historical data.

Syntax

Comments -

```
/* Comment */
// This is a single line comment
/* Multi-line comments
  can be written
  like this */
```

Operators -

- **Assignment** `= += -= *= /=`
- **Arithmetic** `+ - * / % ++ --`

- **Relational** `== != <> <= >=`
- **Conditional** `and or not`

Primitive Types -

- `int i = 100;`
- `float y = 100.00;`
- `string s = “StockX”;`
- `array arr = [ “StockX”, “is”, “a”, “financial”, “language” ];`
- Financial data structures
    - A variable of type `stock` the has the properties associated with a stock.
      ```
      stock stk = SYMBOL;
      ```
    - A variable of type `portfolio`, a collection of `stocks` and information of the orders associated with the portfolio
      ```
      portfolio port = PORTFOLIO;
      ```
    - A variable of type `order` which contains the details of the order being placed by the `account` holder. E.g.: Buy, Sell etc.
      ```
      order ord = ORDER;
      ```
    - A variable of type `account` that holds user information and portfolios
      ```
      account acc = ACCOUNT;
      ```

Functions -

```
function fun(arguments) returns (datatype)
{
    //The definition goes here
};
```

Code Sample -

The following is a code snippet that matches two orders accordingly performs the buy or sell between two account holders.

```
/*
  buyTWTR -  An order placed by user U1 to buy 100 shares of Twitter
  sellTWTR - An order placed by user U2 to sell 100 shares of Twitter
*/

function matchOrders(order buyTWTR, order sellTWTR) returns(boolean)
{
      /*
```

```
                Check the quantity property of the orders to see if they
                match (in this case, 100)
        */
        If[ buyTWTR.quantity == sellTWTR.quantity ]
        {
            //Fulfill the orders and update the quantity
            buyTWTR.quantity = buy( TWTR, buyTWTR.quantity );
            sellTWTR.quantity = sell( TWTR, sellTWTR.quantity );
        };
        return true;
};

Function main() returns()
{
        orderStatus = matchOrders( buyTWTR, sellTWTR );
        if( orderStatus )
        Return 0;
};
```

In the above sample code snippet, `buy()` and `sell()` are functions that accept a variable of type `stock` and the quantity being traded between the two account holders.