finl

Manager	Lauren O’Connor, leo2118
System Architect	Robert Cornacchia, rlc2160
Language Güru	Josh Fram, jpf2141
Tester	Paddy Quinn, pmq2101

1. *Describe the language that you plan to implement*

1.1 Motivation

As Wall Street investment firms have increasingly turned to advanced quantitative and algorithmic strategies to enhance returns, the general retail investor has largely been left behind in their ability to access these technologies. At the same time, there are a massive number of data services available to retail investors, including Yahoo Finance, Google Finance, and data provided by brokerage firms. But integrating these extensive data offerings into a program is a challenge, and many of the existing options cost money. To take advantage of the growing world of quantitative investing, retail investors need a language that allows for free and easy access to stock market data. We believe our language can provide a platform that enables writing these programs with ease.

1.2 Language Description

finl (pronounced “final”) is a simple yet powerful language that exploits the power of the Yahoo Finance API and Yahoo’s web query language, YQL. While other languages can certainly send httpRequests to the API, our language will serve to abstract away the complications that come with sending the requests and receiving the response. A built in “stock” data type will allow the user to build an object, and easily and intuitively populate it with any fundamental data that is available on the Yahoo Finance API. A built in “order” data type will facilitate testing of trading strategies. And a built in “account” data type will allow users to keep track of this performance, and integration with .csv import and export functions can allow users to track performance over multiple sessions.

2. *Explain what sorts of programs are meant to be written in your language*

There are a multitude of applications that can be developed in this programming language. An example of a simple program is a program that prints out the earnings announcement date of every stock listed in the user's "account." This would allow the user to easily see which of their holdings have earnings announcements approaching. Another use of the program could be to implement and test a trading strategy. The user would request information on a universe of stocks, compare them based on criteria of their choice, and then place orders based on which stocks they deem attractive. The investment performance is tracked using each of the "stock," "order," and "account" data types. Also, a user could analyze a large universe of stocks by sorting them based on one or more fundamental data points, easily identifying stocks that best fit a set of criteria the user is looking for.

3. *Explain the parts of your language and what they do*

#Loops
```
for 1 to 10 by 2 {
    …
}
 #while loop
x=2 and y=0~{
    …
}
```

#Conditional
```
x = 2? {
     …
} x = 3?? {
    …
} x = 4?? {
    …
} ! {
    …
}
```

#Operators
```
and, or, not, =
```

#Data Types:

int
double
string
array

stock:
- initialize a variable equal to all of the properties of publicly traded stocks.
- ford.price = (current price of the stock) * 1000 shares
- ford.pe = (current pe of ford)
- for a list of all properties see end of document.

order:
- specify stock and how much they would want to buy/sell
- order.value

```
    + - / * mod
    +<< -<< *<< /<<
    > >= < <=
```

#Declarations and Assignment

```
    int x<<3;
    double y<<2.2;
    string
    name<<"john";
    stock f << $F;
    func x returns
    void(arguments)
```

#Functions

```
    func x returns
int(arguments)
    max(), min(),
    avg()
    asort(), dsort()
    buy(stock, int),
    sell(stock, int)
    #int represents
    amount
    export(string
    f),print(string s)
```

#Comments

```
     # I am a comment
```

- order.datePlaced
- order.yield (returns the difference of the order between now and before)
- order.yieldPercentage (returns the percent change between datePlaced and now)

account:

- accounts contain specific pieces of information regarding a user's account
- account value, data opened, performance since open
- account.graph(date opened, present)

4. *Include the source code for an interesting program in your language*

```
func basicStrategy returns int(){

    # Check if the 50-day moving average crosses the
200-day moving average

    stock tesla = $TSLA
    tesla.get50sma()
    tesla.get200sma()
```

```
     #check if moving averages crossed
     tesla.FiftydayMovingAverage >
tesla.TwoHundreddayMovingAverage? {
          order risky_order = buy($TSLA, 1000)

          # A buy order for 1,000 shares of Tesla
     }

     tesla.FiftydayMovingAverage >
tesla.TwoHundreddayMovingAverage? {
           order revenue = sell($TSLA, 1000)
     }

     revenue.value > risky_order.value? {
          print("Some risks are worth taking.")
     }

     risky_order.value < revenue.value? {
          print("Back to the drawing board.")
     }

     int profit = risky_order.value - revenue.value
     return profit
}

func main returns int(string input) {
     basicStrategy();
     return 0;
}

#When the file loads the main input needs to be able
to handle a .csv file as #input. If no input is
specified, a .csv file is created with the same name
as #the .finl. All data, such as orders placed,
buy/sell orders executed, etc., #will be exported to
the .csv file.
```

Stock Data Type Properties (From Yahoo Finance API):

```
{ "query": {
 "count": 1,
 "created": "2015-09-30T01:23:25Z",
 "lang": "en-US",
 "results": {
  "quote": {
   "symbol":
   "Ask":
   "AverageDailyVolume":
   "Bid":
   "AskRealtime":
   "BidRealtime":
   "BookValue":
   "Change_PercentChange":
   "Change":
   "Commission":
   "Currency":
   "ChangeRealtime":
   "AfterHoursChangeRealtime":
   "DividendShare":
   "LastTradeDate":
   "TradeDate":
   "EarningsShare":
 ErrorIndicationreturnedforsymbolch
angedinvalid":
   "EPSEstimateCurrentYear":
   "EPSEstimateNextYear":
   "EPSEstimateNextQuarter":
   "DaysLow":
   "DaysHigh":
   "YearLow":
   "YearHigh":
   "HoldingsGainPercent":
   "AnnualizedGain":
   "HoldingsGain":
"HoldingsGainPercentRealtime":
   "HoldingsGainRealtime":
   "MoreInfo":
   "OrderBookRealtime":
   "MarketCapitalization":
   "MarketCapRealtime":
   "EBITDA":
   "ChangeFromYearLow":
   "PercentChangeFromYearLow":
   "LastTradeRealtimeWithTime":
   "ChangePercentRealtime":
   "ChangeFromYearHigh":
   "PercentChangeFromYearHigh":
   "LastTradeWithTime":
"LastTradePriceOnly":
   "HighLimit":
   "LowLimit":
   "DaysRange":
   "DaysRangeRealtime":
   "FiftydayMovingAverage":
   "TwoHundreddayMovingAverage":
   "ChangeFromTwoHundreddayMoving
Average":
"PercentChangeFromTwoHundreddayM
ovingAverage":
   "ChangeFromFiftydayMovingAverage"
:
   "PercentChangeFromFiftydayMovingA
verage":
   "Name":
   "Notes":
   "Open":
   "PreviousClose":
   "PricePaid":
   "ChangeinPercent":
   "PriceSales":
   "PriceBook":
   "ExDividendDate":
   "PERatio":
   "DividendPayDate":
   "PERatioRealtime":
   "PEGRatio":
   "PriceEPSEstimateCurrentYear":
   "PriceEPSEstimateNextYear":
   "Symbol":
   "SharesOwned":
   "ShortRatio":
   "LastTradeTime":
   "TickerTrend":
   "OneyrTargetPrice":
   "Volume":
   "HoldingsValue":
   "HoldingsValueRealtime":
   "YearRange":
   "DaysValueChange":
   "DaysValueChangeRealtime":
   "StockExchange":
```

	"DividendYield": "PercentChange": } }