finL

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

Table of Contents

1. Introduction

1.1 Background

As Wall Street investment firms have increasingly turned to advanced quantitative and algorithmic strategies to enhance returns, the general retail investor (i.e. an isolated individual buying and selling stocks, not affiliated with a financial firm) has largely been left behind in their ability to access these technologies. Investment firms pour millions of dollars into advanced technology designed to algorithmically test trading strategies and execute trades with extreme precision with regards to timing. Any individual attempting to buy and sell stocks independent of these financial firms is at a severe disadvantage. 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 as an individual who likely has little or no computer science knowledge, utilizing these platforms is a daunting task, and presents a steep learning curve.

1.2 Goals

Our goal was to make quantitative and algorithmic trading and strategizing more accessible to the general retail investor. Integrating the aforementioned data offerings into a program is a challenge, especially for retail investors unfamiliar with the technological understanding of how to query a database or how to programmatically test trading strategies. In addition, 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, by implementing syntax that reads naturally for a financially literate individual, and by abstracting away the hassle of calling the Yahoo Finance YQL database.

2. Language Tutorial

2.1 Setup and Compilation

2.1.1 Environment Setup

Because finl compiles to Java, any machine running the finl compiler should be installed with Java 7 or above. To utilize the full financial functionality of finl, the user should be connected to the internet, because queries are made via an http get request to the Yahoo Finance YQL database, particularly to populate the stock data type.

2.1.2 How to Compile and Run

Running a Single File (input.finl)

```
$ make clean
$ make
$ ./finlc input.finl
$ ./finl.sh input
```

To Include a Command-Line CSV Portfolio

```
$ ./finl.sh input my_portfolio
```

Options

the finl compiler provides two options: -a and -s

```
$ ./finlc -a input.finl     # prints the abstract syntax tree to the console
$ ./finlc -s input.finl     # prints the semantically analyzed syntax tree to the console
```

Regression Testing

```
$ ./regression_test.sh
```

2.1.3 Program Structure

finL does not require a 'main' function. finL programs are a list of statements and static function declarations, both of which must be followed by semi-colons. Statements can come before or after function declarations. However, functions cannot be called until after they are defined, so it is best practice to define all functions at the top of the program . Likewise, it is best style to import which portfolio you will be using at the top of your program, otherwise a default portfolio will be used. Declarations also must be made before, and separate from, assignments. Below is a very simple program illustrating program structure.

```
portfolio “file_name”;

function int add(int x, int y){
	return x+y;
};

function int subtract(int x, int y){
	return x-y;
};

int two;
int one;
two << 2;
one << 1;
int result;
result << subtract(two, one);
print result; # prints 1
```

2.2 Basic Types and Syntax

2.2.1 Primitive Types

finL supports the primitive types int and float. Primitive types can be declared and assigned as followed. Assignments must follow declarations, they cannot be on the same line.

```
# Primitive declarations and assignments
int our_grade;
float value;
our_grade << 100;
value << 231.7;
```

2.2.2 Non-Primitive Types

finL supports strings, stocks, and orders. Sample declarations and assignments are provided. To access an attribute of these datatypes, use the [] punctionation, also shown below. For more detailed information about these data types, please refer to section 3.2.2.

```
# Non-primitive declarations and assignments
string this_project;
this_project << “fire”;

stock i_love_apple;
i_love_apple << @AAPL;

print @NFLX[price];

order buyApple;
buyApple << 10 of @AAPL;
```

2.2.3 Functions, Operators, and Keywords

User-Defined Functions

Functions can be defined anywhere, but they can only be called after they have been defined. The function must include the return type, although void can be used. Let's take the following function declaration as an example:

```
# this function takes in  floats and returns their average
function float avg(float x, float y){
       float sum;
       sum << x+y;
       return sum/2.0;
};
```

Operators & Keywords

finL provides many built-in operators for ease of use. Common arithmetic operators are addition (+), subtraction(-), division(/), and multiplication (*). Additional operators provided are modulus (%), which gets the remainder from the division of two numbers, and power (**), which raises the first argument to the power of the next argument.

Logical operators include and, or, & not. Comparison operators include greater than (>) or equal to (>=), less than (<) or equal to(<=), and equality operators include equals (=). To use not equals, enclose an equality expression in parentheses and precede it with the not keyword (i.e. not(x=y)).

One useful keyword is print, which prints both primitive or non-primitive data-types to the console, including a terminating new-line character.To print the current portfolio, simply use the print keyword with no argument (i.e. print ;).Valuable keywords that finL provides also include buy and sell, which execute an order. They can be used in two ways, as shown below. Portfolio is a keyword in finl that switches which portfolio you are currently using, by exporting the current portfolio to a CSV and importing the next portfolio from a CSV. For instance, below, we switch the portfolio to my_portfolio_name.

```
order appleOrder;
appleOrder << 10 of myStock;

buy appleOrder;
sell 10 of @AMZN;

portfolio "my_portfolio_name";
```

2.2.4 Control

Conditional Statements

finL supports conditional statements via the ? and ! keywords. ? acts as an if statement and ! as an else. ? must be followed by an expression which evaluates to a zero (false) or a non-zero (true) number (all comparison and logical operators evaluate to 1 if true or 0 if false). Only the last body of a conditional statement should be terminated with a semi-colon, as illustrated below.

```
int grade;
grade = 93;


grade = 93 ?
       {print “exact guess”;};

grade > 90 ?
       {print “LETS GO”;}
! {print “ya hate to see that”; };

# this will print:
# exact guess
# LETS GO
```

Loops

finL supports two types of loops. The while loop contains a conditional and a body. The body will continue to execute until the conditional is true.

```
while x=5 { print “here”; x=1};
# prints “here” once
```

The while loop can also be utilized to execute a block a set number of times, such as a for loop would do. How to do that is demonstrated below.

```
int i;
i<<0;
while i<5 {
     print i;
     i+<<1;
};
# executes 5 times
```

When loops are a bit more complicated, but are a key feature of finl. A when loop takes a conditional that evaluates to 0 or 1 and periodically checks it. This conditional should utilize at least one stock attribute and another stock attribute or a number. The when loop is executed in a separate thread so the body of the program can continue executing. When the loop evaluates to true, it executes its body once and terminates. This program periodically checks the when loop conditional. When the 50 day moving average moves above the 200 day moving average,the buy is executed. Regardless of if the conditional is satisfied, the program will execute “sell 20 of chill”

because it happens in a separate thread.

```
stock chill;
chill << @NFLX;
order netflixOrder;
netflixOrder << 70 of chill;

when chill[priceMA50] > chill[priceMA200] {
     buy netflixOrder;
}
sell 20 of chill;
```

2.3 Example Programs

2.3.1 When Loops

The following program prepares a buy order of 1000 shares of Tesla stock. When the Tesla 200 day moving average moves above the Tesla 50 day moving average, the order is executed.

```
function void tesla() {
       stock stk;
       stk << @TSLA; # set stock variable equal to Tesla stock
       print stk;
       order tesla_order;
       tesla_order << 1000 of stk; # set order to 1000 shares of Tesla
       print tesla_order;
       when stk[priceMA200] > stk[priceMA50] {
              buy tesla_order;
              print;
       };
       print; # print portfolio
};

tesla();
```

2.3.2 Switching Portfolios

The following two programs illustrate how to work with different portfolios. The first snippet creates a new portfolio named demo, since no CSV with that name currently exists in the directory. The portfolio is populated and printed. At the end of the program, the current portfolio is always exported to a CSV file, so there now exists in the directory "demo_holdings.csv" and "demo_orders.csv."

```
portfolio "demo";

function void f() {
        stock stk1;
        stk1 << @FB;

        stock stk2;
        stk2 << @AAPL;

        order order0;
        order0 << 20 of stk1;
        order order1;
        order1 << 10 of stk1;

        order order2;
        order2 << 10 of stk2;

        order order3;
        order3 << 10 of @NFLX;

        buy order1;
        sell order2;
        buy order3;
};

f();
print;
```

This program imports the demo_holdings CSV that we just created, prints it, and makes more changes to the portfolio, which is again exported at the end of the program. The previous files are overwritten with the updated portfolio information.

```
portfolio “demo”;

print;

stock stk1;
stk1 << @DPZ;

order order1;
order1 << 100 of stk1;

stk1[price] < 50 ?
        {buy order1;}
! {sell order1;};
```

```
print;
```

3. Language Reference Manual

3.1 Lexical Conventions

3.1.1 Tokens

There are five types of tokens in finL. They are: identifiers, reserved words, whitespace, operators, and separators.

3.1.2 Identifiers

Identifiers are any sequence of letters, underscores, and digits that signify the name of a variable or function. Identifiers in finL are case-sensitive, must begin with a letter, and cannot contain any punctuation besides underscores.

3.1.3 Comments

 # <comment>

Only single-line comments are supported in finL. Comments are indicated with a single # symbol, e.g.:

```
#this is a one line comment
```

3.1.4 Whitespace

All whitespace (spaces, tabs, new lines) in finL is ignored.

3.1.5 Separators

finL uses the following characters as separators:

`{}` – code block separator
`()` – grouping separator and parameter list separator
`;` – statement delimiter

3.1.6 Reserved Words & Symbols

Types & Null (see 3.2 Types)

`int`
`float`
`string`
`stock`
`order`

Boolean Logic Operators (see 3.3 Expressions)

- `and`
- `or`
- `not`

Control Flow & Loops (see 3.4 Control)

- `?`
- `!`
- `while`
- `when`

Functions & Related (see 3.5 Functions)

- `function`
- `return`
- `void`
- `of`
- `portfolio`
- `print`
- `buy`
- `sell`

3.2 Types

3.2.1 Primitive

int

```
int   fetty;
fetty  <<  1738;
```

signed 64 bit type
a string of numeric characters without a decimal point, and an optional sign character
an integer literal may be expressed in decimal (base 10)

float

```
float x ;
x  << 34.55;
```

signed 64 bit type
a string of numeric characters that can be before and/or after a decimal point, with an optional sign character
a float literal may be expressed in decimal (base 10)

3.2.2 Non-Primitive

string

```
string  x;
x  <<  "hello, its me";
```

a finite sequence of ASCII characters, enclosed in double quotes

stock

stock apple;
apple << @AAPL;

wrapper type that contains the stock ticker and all data from Yahoo Finance YQL database table yahoo.finance.quotes
to access an individual attribute, use the [] punctionation, i.e. my_stock[ebitda]

attributes include:

bookValuePerShare	ebitda	eps	marketCap
pe	peg	priceBook	priceSales
revenue	roe	sharesFloat	sharesOutstanding
epsEstimateCurrentYear	quote	eps	price
epsEstimateNextQuarter	priceOpen	pricePrevClose	priceMA200
epsEstimateNextYear	priceMA50	priceDayHigh	priceDayLow
oneYearTargetPrice	bid	ask	avgVolume
changeFromYearHigh	change	changePercent	changeFromMA200
changeFromYearLow	dividend	annualYield	changeFromMA50
annualYieldPercent	exDivDate_String	payDate_String	

to initialize a stock, follow the @ symbol with the appropriate ticker (i.e. Google's ticker is GOOG, Netflix's ticker is NFLX)

order

order buyApple;
buyApple << 10 of @AAPL;

buy or sell order including which stock and number of shares
utilize the buy and sell keywords to execute an order
orders must be associated with a portfolio to be executed (if no portfolio is specified, an order is associated with the default portfolio)

3.2.3 Casting

finL does not support casting.

3.3 Expressions

3.3.1 Declaration & Assignment

Declarations

<type> <identifier>;

examples:

```
int x;
stock stk;
string name;
```

Assignments

<identifier> << <value>;

identifiers must be declared in a line previous to when they are assigned
you cannot declare and initialize a variable in the same line

examples:

```
int x;
float dollars;
x<<3;
dollars<<2.2;
```

stock

<identifier> << @<ticker-name>;

the @ symbol should be followed by the appropriate stock ticker symbol

order

<identifier> << <number-of-shares> of <stock>

a stock variable or an @ sign followed with a ticker can be used after 'of'
the number of shares should be positive, a buy or sell is indicated by using the appropriate keyword

3.3.2 Arithmetic Operators

<value> + <value>

finL supports the following arithmetic operators, in order of precedence:

power (**)
multiplication(*) division(/) modulus(%)
addition(+) subtraction(-)

3.3.3 Relational Operators

<value> <= <value>

finL supports the following relational operators:

< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)
= (equal to)
*not: to utilize not equal, use the not keyword in conjuction with the = operator
i.e.: not (x=y)

all return a result of 0 if the condition is false or 1 if the condition is true
relational operators all support integers, floats, stock attributes, and strings (strings are compared lexographically)

3.3.4 Logic Operators

<expression> or <expression>

finL supports the following boolean operators as reserved keywords:

and
logical intersection of two expressions
example: 0 and 1 evaluates to 0 (false)

or
logical union of two expressions
example: 1 or 0 evaluates to 1 (true)

not (<expression>)
logical negation of an expression
must include parentheses around argument
example: not (1=1) evaluates to 0 (false)

Any non-zero number is considered true, and zero is considered false

3.3.5 Shorthand Operators

<value> +<< 2

finL also supports shorthand operators (for ints and floats) that combine arithmetic and assignment, for the following operations:

addition (+<<), subtraction (-<<), multiplication (*<<), and division (/<<)

examples:

```
x<<x+3; # can also be written x+<<3;
x<<x*4; # can also be written x*<<4;
```

3.3.6 Precedence

Highest Precedence	`[]`
	`**`
	`* / %`
	`+ -`
	`< > <= >= =`
	`and or not`
Lowest Precedence	`<< +<< -<< *<< /<<`

3.4 Control

3.4.1 Conditionals

1<0 ? {...}
! {...};

The last conditional statement body must terminate with a semi-colon. Boolean true and false is not used, instead zero is false and any non-zero number is true.

<expression>?

<boolean expression> ?{ … } ;

if conditionals are represented with a single question mark (?)
? must be preceded by a boolean expression, and a block that is executed if the condition is non-zero

example:

```
x = 2 ? { print “X is 2!”;};
```

<expression>!

! { … };

denotes an else statement
must occur following a ?
it does not include a boolean expression, but it is still followed by a block

example:

```
x=4 ? { … }
! { print “X is 2!”;};
```

3.4.2 Loops

while

while <expression> { … };

perform iterations of a loop until the boolean condition (evaluated at the start of each iteration) is false
ability to iterate for a certain number of times (i.e. a for loop), as displayed in the second example below

example:

```
while not(x=1) { print “yes”; x=y; };
```

for-loop example:

```
int x;
x << 0;
while x<10 {
	print x;
	x+<<2;
};
# executes 5 times
```

NOTE: while loops should not be used to continually query the YQL database; use a when loop

when

when <boolean expression>

a key control loop in finL
starts a new thread to support being able to place multiple limit orders
the when loop checks the conditional periodically
when the loop has a boolean expression that evaluates to non-zero, it executes its body once and terminates
rest of program continues on to execute
when loops should have expressions with at least one argument as a stock attribute, and the other as a number or stock attribute
both arguments cannot be numbers because an infinite loop will be created

example:

```
stock chill;
chill << @NFLX;
order netflixOrder;
```

```
netflixOrder << 70 of chill;

when chill[priceMA200] >
       chill[priceMA50] {
       buy netflixOrder;
}
sell 20 of chill;
```

In this program, when the 50 day moving average moves above the 200 day moving average,the buy is executed. Regardless of if the conditional is satisfied, the program will execute "sell 20 of chill" because it happens in a separate thread.

3.4.3 Scope

finL has function scope, similar to Javascript, which means that the scope changes inside functions.

3.5 Functions

3.5.1 Reserved Words

function

function <return-type> <identifier> (<optional-parameters>){ ....} ;

a reserved word used for user-defined functions that takes 0 or more parameters, and returns an expression or void

example:

```
function void tester(int testParam) { ... } ;
```

return

return <expression>;

return *expression* to the caller from the function block
cannot be used outside of function declarations

void

return void;

a reserved word returned to the caller from the function block that indicates the function does not return a value

of

myOrder << 10 of @GOOG;

keyword that is reserved for use in order assignment, i.e. to buy '10 shares of Google' as above

portfolio

portfolio <name-string>

keyword that exports the current portfolio to a csv file
imports the specified portfolio from a csv file
only the portfolio name needs to be given, not the csv file name (i.e. demo_holdings.csv would be passed just as "demo")
if no portfolio specified at program start, a default is used
at program end, the current portfolio is automatically exported to a csv file

buy

buy <order>

keyword that executes a buy specified by the order
associated with whatever the current portfolio is

sell

sell <order>

keyword that executes a sell specified by the order
associated with whatever the current portfolio is

print

print "hello professor edwards";
print ;

keyword that prints to the console, including a terminating new-line character
print can be ued with any of the following: int, float, string, stock, stock attributes, and order
to print the current portfolio, use the print keyword with no following arguments, i.e.: print ;

3.5.2 Function Definition

function <return-type> <identifier> (<optional-parameters>) {…} ;

example:

```
function int buy_ten(stock myStock) {
	buy 10 of myStock;
	return 1;
} ;
```

4. Project Plan

4.1 Project Management

4.1.1 Planning

The concept behind the finance language came as a suggestion from Josh, who was the most familiar working with financial concepts and had the most knowledge of algorithmic trading. We tried to meet weekly, or bi-weekly, in order to work towards the next-closest milestone, and increasingly more frequently as that milestone approached. Conveniently, the four of us are in various groups together for other classes, so scheduling frequent meetings was not a problem.

4.1.2 Specification

The specification initially outlined in our project proposal and LRM was sketched out by Josh and Rob, since they are most comfortable with financial concepts, whereas Lauren and Paddy were not familiar at all with financial terms and data. This was actually a perfect blend, having half the group with a deep understanding for the material, and the other half seeing the material with a fresh perspective. As we began developing, Paddy and Lauren were able to identify the lack in logistical feasibility and usability for certain constructs coming from a purely programmatic point of view, and Josh and Rob were able to alter our original ideas to fit with a more financial-focused model.

4.1.3 Development

In the development of the "Hello World" milestone, it became apparent to us that we worked much more effectively when physically in the same place, rather than having four secluded individuals pushing and pulling disparate code. The four of us met regularly and actually used an HDMI cord to program on a television screen so that we could each be aware at all times what point we were at in the compiler, and so we could easily show each other code snippets. This led to much more effective programming threads, as Rob could create test scripts based on what new features Paddy was implementing, as an example. We also utilized Git as a distributed version control system in order to work more cohesively.

4.1.4 Testing

As aforementioned, the testing strategy was to develop unit tests, specific to each feature as we added that feature to the compiler. We tried to implement both positive and negative tests; that is, we would purposely make some tests fail in order to make sure our compiler was only compiling proper code. We implemented regression testing in order to make certain that previous tests did not fail upon development of new features, and we implemented a batch of more in-depth test scripts after our language had been fleshed out in order to test more complex programs.

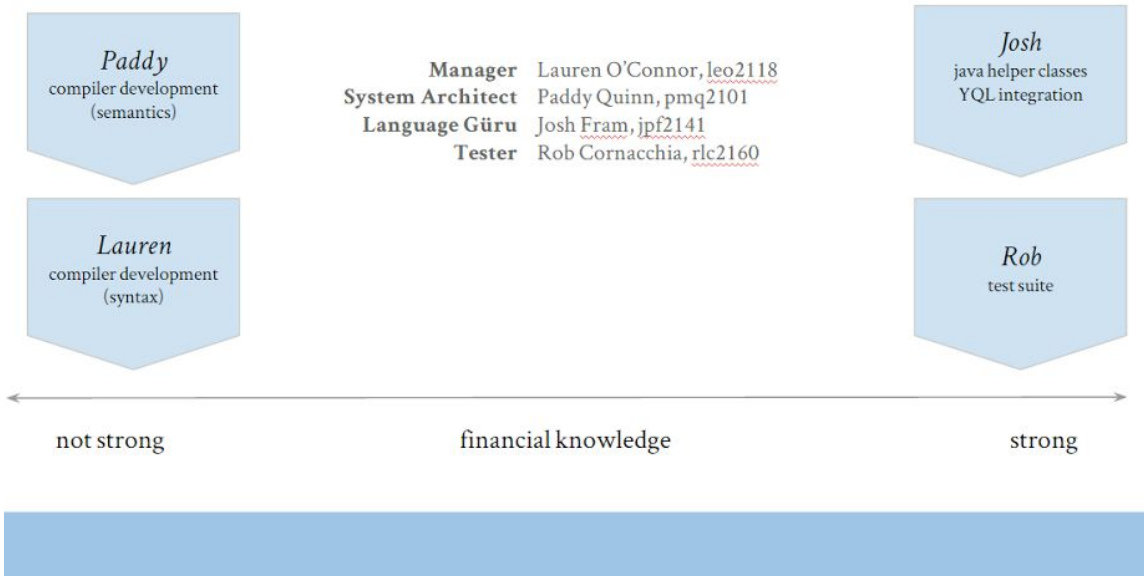
4.2 Project Timeline

Below is a schedule of when we completed each required milestone, as well as a list of progress markers as a guideline for where we were in between each milestone

Milestone	**Date**
Language Idea Formulated	Sept 28th
Project ProposalCompleted	Sept 30th
LRM Completed	Oct 26th
Syntactic Analysis Started	Nov 5th
Hello World and other Small Programs Completed	Nov 12th
Regression Testing Added	Nov 15th
'Hello World' Demo	Nov 16th
Semantic Analysis Started	Nov 30th
Basic Types, Operators, Functions Implemented	Dec 16th
Final Demo Program Finalized	Dec 20th
Final Presentation	Dec 21st
Final Report Finished	Dec 22nd

4.3 Roles and Responsibilities

The responsibilities were divided up according to role. While everyone contributed to the syntactic and semantic analysis, Paddy (as system architect) took the lead in developing the compiler, in particular specializing in the semantic analysis. Lauren, in addition to leading executive decisions about the language specification, lent most of her programming time to helping Paddy build the compiler, specializing in the lexical and syntactic analysis. As Josh and Rob were much more versed in financial concepts, they focused on integrating the API as well as guiding Paddy and Lauren on the more financial aspects of the language, such as stock attributes and portfolios. Because Josh introduced the group to the Yahoo Finance API, he took the lead in implementing the necessary Java libraries to query the database and manipulate stock data. Rob created the bulk of the test suite, writing simple positive and negative test files for each new feature we implemented.



4.4 Development Environment

As mentioned, the project was source-controlled using Git, and we developed primarily on the OS X platform. The compiler code was all developed using the Sublime IDE, whereas the necessary Java code was implemented using the Eclipse IDE (there were definitely some issues that had to be resolved merging the two). All scripts (testing suite and compiling OCaml/Java) were also written in Sublime.

5. Language Evolution

5.1 Summary

There were three main factors that affected the evolution of our language from the time of conception in the LRM to the final product: aesthetics and usability, time sensitivity, and feasibility. The biggest driver of change was trying to make our language easy to use, in part by making it aesthetically pleasing and natural for our users, which our first iteration didn't quite keep in mind. For instance, we changed the if conditional from ?(expression) to expression ? which is much more natural because it reads like an actual yes/no question. Similarly we got rid of the percent data type because we thought it was confusing for the user and didn't add any signficant value. Time sensitivity was also a big factor, because we were a bit too optimistic about how much we could actually implement over the course of the semester. In the end, we tried to narrow in on our most crucial and most interesting features. Feasibility was also an issue, because some things we specified in our LRM, as we started to implement code, we realized would not be feasible and we had to make alterations accordingly.

5.2 Types

5.2.1 Primitive

Originally, we had included a percent type in our primitive data types. It would be declared as a percent out of 100 (i.e. 50 for 50%), but then be treated as a proper percent (0.5) in calculations. As we implemented the language we realized that this might confuse the user, and that it didn't add anything beneficial to the program, so we discarded it.

5.2.2 Non-Primitive

Likewise, we also had a currency type which kept the currency amount and currency denomination, and users could convert between denominations. Sketching out the types, having this as its own structure didn't seem quite natural, especially since users would most likely keep track of money in float variables. It also didn't seem to have much benefit for the user besides the conversion. If we were to continue working on the project, we would likely implement conversion as a reserved keyword instead, as a more aesthetic option and for more ease of use.

We also originally had portfolio as its own data type. We kept the portfolio idea but implemented it a bit more creatively. Instead of declaring a type, the user can import and export portfolios from csv files, users can switch and print portfolios very easily, and orders are automatically associated with the current portfolio for ease of use. We think the portfolio usage as it stands now definitely helps the user grasp how portfolios are used more conceptually, almost as a state that you are in, rather than an object to be directly manipulated.

The order and stock data types were also much more complicated when we originally sketched them out, but as the language evolved, we were able to make them much more succinct and easy to conceptualize and use, which is important for our target user, someone who may not have much of a CS background. Instead of associated functions, we have the buy and sell keywords, and instead of toString() methods, we take care of 'pretty-printing' the stocks, orders, and portfolios, as two examples.

5.3 Expressions

Many of our expressions remained the same, although for not equals, we decided to switch to a more logical and aesthetic way (not = looks inconsistent) by simply having the user create an equality check and then preceding it with the not operator.

We also had not thought fully through our comparison operators and particularly, which data types they would be applicable to. In the end, to try and prevent confusion and memorization of which operators work for which data type, we tried to implement comparison for all the types, for example by comparing strings lexographically. The only types that cannot compare are stocks and orders, but only because you compare their values, not the entire type itself.

With regards to declaration and assignment, we luckily though of much more aesthetic ways of implementing assignment for non-primitive data types. In our LRM, our stock data type had a very clean assignment (stock stk << @AAPL), but we originally wanted to populate the order data type by declaring it and then assigning each of it's attributes, which is time-consuming, inconsistent, and not very feasible. Instead, we came upon the very intuitive declaration of "10 of @AAPL," for example, which reads just like an order would in English ("10 shares of Apple").

5.4 Control

Our goal for control was to keep it to as few options as possible, so that the user could have a wide range of functionality, but wouldn't be overwhelmed with having to learn how to use too many features.

5.4.1 Conditionals

Originally because of our misunderstanding of lexical analysis, we believed that having the question mark in front of the conditional expression would be easier to implement, albeit a bit more confusing and much less aesthetically pleasing. Once we realized the feasibility of having it at the end of the conditional, we switched it to be more aesthetic and natural to the reader, as now it reads like a yes or no question (i.e. is x equal to 2).

5.4.2 Loops

Originally we included just for loops, because both for and while loops can be implemented using the other. However, we realized that using a while loop to execute a for loop is much more intuitive than using a for loop to execute a while loop (i.e. for(; x!=0;)), so we switched over. We also realized quickly that our 'index' keyword would not work (inside a for loop, the keyword index would keep track of the current iteration), because of nested loops, so that was a good example of us having to drop a feature due to feasibility issues.

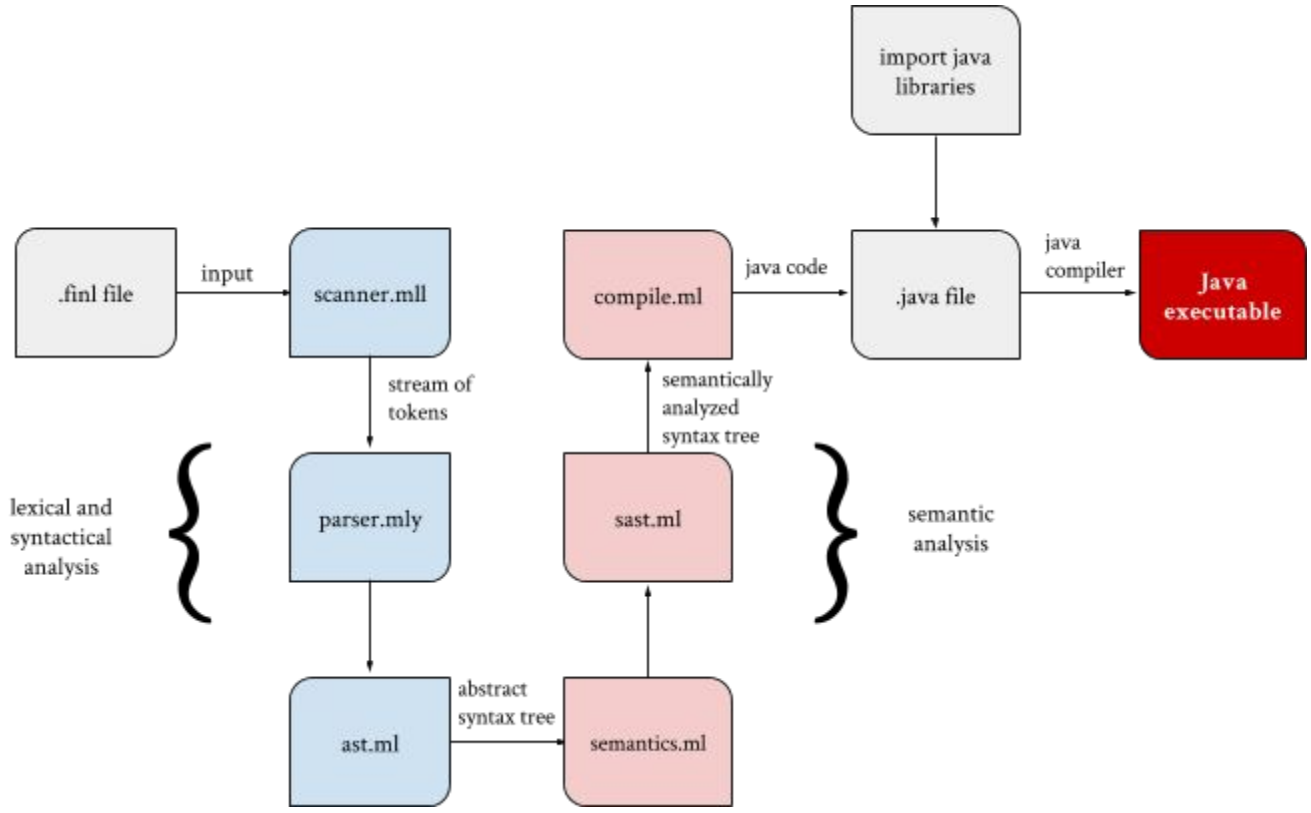
5.5 Functions

We decided not to have any built-in functions, because in our LRM we had, for instance, export as a keyword, convert as a built-in function, and it was not very consistent. To create an easier interface for the user, and for consistency, we decided to implement all the built-in functions as keywords.

We got rid of the use keyword due to our refactoring of the portfolio type. Now, instead of use, the user uses the keyword portfolio to either import or switch portfolios. We also ended up getting rid of the export keyword and instead having the program automatically export to a csv file at the end of every program, again to make it easier for the user.

6. Translator Architecture

6.1 Diagram



6.2 Lexical Analysis

Scanner
Implented using ocamllex, the scanner takes a .finl program and produces a stream of tokens, providing basic lexical analysis. It is here that whitespace and comments are discarded, and that programs with invalid tokens are caught.

6.3 Syntactic Analysis

Parser and AST
Implemented using ocamlyacc, the parser takes the stream of tokens given by the scanner, and uses them to generate an AST, defined in the OCaml ast file. It is here that programs with invalid syntax (i.e. when tokens are in an invalid order) are caught.

6.4 Semantic Analysis

Semantics and SAST
Implemented using OCaml, the semantics analyzer parses through the AST and converts it to an SAST, after checking that all items in the AST are semantically valid. It is here that any programs with meaning-errors are caught (i.e. using an undeclared variable, trying to add a string to a float).

6.5 Java Code Generation

Compile and Java Libraries
Implemented using OCaml, the compile file takes in the SAST and parses it to generate corresponding Java code. All function definitions are declared static and placed outside of the Java main function, and all statements are placed inside the Java main function. The Java libraries are used to query the YQL database and utilize CSV files, as well as define and implement more complex data types such as stock and order, utilizing the respective data.

7. Test Plan

7.1 Test Suite

7.1.1 Unit Testing

In order to immediately test features, we used unit testing, both positive and negative. Some small programs are specifically designed to fail in order to make sure certain things are forbidden in finl and are caught in our compiler rather than in the Java compiler. This allowed us to test features as they were implemented. These tests can be found in the test_suite folder, or in section 9, below.

7.1.2 Regression Testing

We utilized regression testing in order to ensure that as we added features, none of our modifications caused new bugs, which would be reflected in the failure of prior, previously successful tests (and this happened frequently). Each unit test has a corresponding expected output file, and the actual output upon running the test is evaluated against the expected output from the corresponding file. To better visualize failures, successful tests are printed as green, and failed tests are printed as red. The shell script to run the regression test is in the src file, labeled regression_test.sh.

7.2 Implementation

We utilized several sources of testing. To unit test, we compiled and ran a single file as one would normally do in finl. To regression test, we used a shell script regression_test.sh, which evaluates all test outputs to their expected outputs, prints green for successful matches, and prints red for failed tests. We also utilized the menhir --interpret --interpret-show-cst option towards the beginning of development for immediate syntactical testing. And finally, to encourage a more focused and in-depth testing strategy, we implemented a -a option and a -s option for .finl files, which print the AST and SAST respectively.

7.3 Sample Test Scripts

7.3.1 Unit Tests

Lexical and Syntactic Example Test
This simple test was written immediately after the syntax for variable declarations was implemented. It tests that both the lexical and syntactic analysis of variable declarations is valid.

vdecl_test.finl

```
int x_1;
string w__;
print "success";
```

Semantic Example Test

The following unit test was written immediately after the semantic analysis for binary operator type-checking was implemented. To make sure type mismatches were being caught, we wrote this test, which should (and does) fail because adding an integer to a string is forbidden in finl. Most of the semantic tests are designed to fail.

binop_type_mismatch_test.finl

```
function int main() {
	int x;
	x << 1 + "string";
};
```

7.3.2 Regression Testing

The following is a snippet of the results from our regression testing script. As you can see, many of our old unit tests pass, regardless of adding new functionality. The test multiple_return_test.finl failed because we had not yet implemented semantic checking for multiple returns. Because the actual output for this test was not the same as the expected output, the test fails, which also lets us know that the regression test is working and not just passing every test we give it.

```
Compiling int_float_op_test.finl ...
Running int_float_op_test ...
int_float_op_test passed.
Compiling main_test.finl ...
Running main_test ...
main_test passed.
Compiling mod_test.finl ...
Running mod_test ...
mod_test passed.
Compiling mult_assign_test.finl ...
Running mult_assign_test ...
mult_assign_test passed.
Compiling multiple_params_test.finl ...
Running multiple_params_test ...
multiple_params_test passed.
Compiling multiple_return_test.finl ...
Running multiple_return_test ...
multiple_return_test failed.
Compiling negative_statement_test.finl ...
Running negative_statement_test ...
negative_statement_test passed.
Compiling negatives_test.finl ...
Running negatives_test ...
negatives_test passed.
Compiling nested_if_test.finl ...
Running nested_if_test ...
nested_if_test passed.
Compiling no_return_test.finl ...
Running no_return_test ...
no_return_test passed.
Compiling not_test.finl ...
Running not_test ...
not_test passed.
Compiling operator_test.finl ...
Running operator_test ...
```

In order to visualize successes and failures better, we sent all error messages to a .log file, so they wouldn't print to the console. However, if you run a program individually, the error messages still print, so the user gets a clear message telling them what they did wrong. This is demonstrated below, where the user defined two function parameters with the same name, which is not allowed in finl.

```
dyn-160-39-252-96:src padraic$ ./finlc ../test_suite/bad_formals_test.finl
Fatal error: exception Semantics.Except("Formal parameter 'x' is already defined!")
```

8. Conclusions

8.1 Paddy Quinn

The biggest surprise for me was definitely how useful OCaml is, especially in terms of language design. After just the simple programs from Homework 1, I was dreading the thought of having to build an extensive, robust compiler using OCaml. But as I started to become comfortable writing in it, I realized how much more efficient OCaml is than Java, particularly because of its pattern-matching capabilities, and its natural proclivity for recursive functions. Probably the next biggest takeaway from this project is the importance of communication in group work. I hadn't done many group programming projects prior to this semester, and having one of the first be this compiler was definitely a good learning experience; due to all the files being so interconnected, the littlest change could break the compiler, so it was extremely important for us all to be in sync with what each other was doing.

8.2 Rob Cornacchia

Before this project, testing seemed like a chore that took away from writing other, more pertinent, code. However, I've come to understand not only how important test code is, but also how much time it saves. When working with such a large codebase, every new line of code has the potential to wreak havoc on the multilayered compilation process or cause shift/reduce conflicts. By writing a thorough test suite, we could code with confidence that we were moving in the right direction (or at least know when we lit the computer on fire). Writing tests for a new feature of the language before implementing that feature helped direct our efforts and streamline the development process.

8.3 Lauren O'Connor

A bit prior to assigning roles at the beginning of the semester, I think most of us viewed the other roles as having a heavier workload than the manager, more of a figurehead position, myself included. However, it became apparent to me how significant my role as a leader was as the project progressed. As we started developing, the rest of the group would turn to me with errors they couldn't fix, to ask what work they could do next, or to prompt me to make a decision on language specification. I think this was really vital because having someone assign tasks made the group much more effective and cohesive, and having one person executively decide on the specifics of the language led to a more consistent end product. I'm really glad I ended up in this role because I think it was a valuable, and practical, experience in managing a development product.

Likewise, I think it was a very valuable experience to work on such a big programming project from semester beginning to end. Usually for classes, we write small programming projects and then never think about them again, which isn't really a realistic representation of how projects work in 'the real world.' Your code has to be readable not only to you but to everyone, and it's important to do everything 'the right way' instead of the easy way, because taking shortcuts could mean more work down the line. Working with git was also valuable, because I haven't worked too thoroughly with any version control systems before, and I'll definitely have to do that moving forward. Having such a big project with a reasonably long timeline (rather than a small algorithm due in two weeks time) definitely offered me practical experience.

8.4 Josh Fram

With regards to the non-technical aspects of the project, I realized just how important effectively delegating work can be. By playing to our individual strengths and weaknesses, we were able to be much more productive than if we had simply tried to split every aspect of the project evenly among us. For instance, I was happy to take over writing most of the Java libraries because I am very familiar with Java, and I am very comfortable with the concepts underlying the library classes (stocks, portfolios, etc.). Because of this, object oriented design and coding process was significantly easier for me than it would have been for Lauren, Rob, or Paddy. It was also good to have someone that is aware of how the end goals of the language coalesce with the functionality and language design work in a role that made it easy to combine the two. By dividing the labor appropriately, we kept a good balance of financial versus technical viewpoints, and we were more effective because each portion of the project was developed by whomever was best suited for it.

On the technical side of things, I gained even more of an appreciation for good object oriented design. The Java Libraries for finL are huge and complicated, and without a design that made sense, using them (and maintaining them in the future) would have been a very painstaking process. The amount of data available from the Yahoo Finance API is incredible, and writing libraries and a programming language in a way that made it possible to nimbly deal with all the available data proved to be a big help as we moved towards the end phases of our project.

9. Full Code Listing

9.1 Scanner

scanner.mll

```
{ open Parser }

let letter = ['a'-'z' 'A'-'Z']
let digit = ['0'-'9']
let ticker = '@'letter+
let variable = letter(letter|digit|'_')*
let access = letter(letter|digit)*

rule token = parse
       [' ' '\t' '\r' '\n']     { token lexbuf }
       | '#'                    { comment lexbuf }
       | '('                    { LPAREN }
       | ')'                    { RPAREN }
       | '{'                    { LBRACE }
       | '}'                    { RBRACE }
       | ';'                    { SEMI }
       | ','                    { COMMA }
       | '+'                    { PLUS }
       | '-'                    { MINUS }
       | '*'                    { TIMES }
       | '/'                    { DIVIDE }
       | '%'                    { MOD }
       | "**"                   { POWER }
       | '<'                    { LT }
       | "<="                   { LEQ }
       | '>'                    { GT }
       | ">="                   { GEQ }
       | '='                    { EQ }
       | "<<"                   { ASSIGN }
       | "+<<"                  { AASSIGN }
       | "-<<"                  { SASSIGN }
       | "*<<"                  { MASSIGN }
       | "/<<"                  { DASSIGN }
       | "and"                  { AND }
       | "or"                   { OR }
       | "not"                  { NOT }
       | "of"                             { OF }
       | "buy"                            { BUY }
       | "sell"                           { SELL }
       | "print"                          { PRINT }
       | '?'                    { IF }
       | '!'                    { ELSE }
       | "while"                { WHILE }
```

```
        | "when"                { WHEN }
        | "int"                 { INTD }
        | "float"               { FLOATD }
        (*| "array"               { ARRAY }*)
        | "string"              { STRINGD }
        | "stock"               { STOCK }
        | "order"               { ORDER }
        | "portfolio"           { PF }
        | "function"            { FUNC }
        | "return"              { RETURN }
        | "void"                { VOID }
        | ticker as t                { TICK(t) }
        | variable as var      { VAR(var) }
        | '['access*']' as a     { ACCESS(a) }
        | digit+ as i            { INT(int_of_string i) }
        | digit*'.'digit+ as flt { FLOAT(float_of_string flt) }
        | '"'('\\'_|[^'"'])*'"' as str   { STRING(str) }
        | eof                         { EOF }

and comment = parse
        '\n' { token lexbuf }
        | eof { EOF }
        | _  { comment lexbuf }
```

9.2 Parser

parser.mly

```
%{ open Ast %}


%token SEMI LPAREN RPAREN LBRACE RBRACE COMMA
%token PLUS MINUS TIMES DIVIDE POWER MOD
%token ASSIGN AASSIGN SASSIGN MASSIGN DASSIGN
%token EQ GEQ GT LEQ LT
%token RETURN WHILE WHEN IF ELSE VOID
%token AND OR NOT
%token BUY SELL PRINT
%token INTD STRINGD FLOATD /*ARRAY*/ STOCK ORDER PF FUNC OF
%token <string> TICK
%token <string> ACCESS
%token <int> INT
%token <float> FLOAT
%token <string> STRING
%token <string> VAR
%token EOF

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN AASSIGN SASSIGN MASSIGN DASSIGN
```

```
%left AND OR
%left EQ
%left GEQ GT LEQ LT
%left PLUS MINUS
%left TIMES DIVIDE MOD
%left POWER

%start program
%type <Ast.program> program


%%
program:
  line_list EOF { { lines = List.rev $1 } }

line_list:
  /* nothing */ { [] }
  | line_list line { $2 :: $1 }

line:
  statement { Stmt($1) }
  | fdecl { Fdecl($1) }

statement:
  expression SEMI { Expr($1) }
  | WHILE expression LBRACE statement_list RBRACE SEMI { While($2, $4) }
  | WHEN access_expression LBRACE statement_list RBRACE SEMI { When($2, $4) }
  | expression IF LBRACE statement_list RBRACE %prec NOELSE SEMI { If($1, $4, []) }
  | expression IF LBRACE statement_list RBRACE ELSE LBRACE statement_list RBRACE SEMI {
If($1, $4, $8) }
  | vdecl SEMI { Vdecl($1) }
  | BUY order SEMI { Buy($2) }
  | SELL order SEMI { Sell($2) }
  | PRINT expression_option SEMI { Print($2) }
  | RETURN expression_option SEMI { Ret($2) }
  | PF STRING SEMI { Portfolio($2) }

access_expression:
  access EQ access { ($1, Equal, $3) }
  | access EQ number { ($1, Equal, $3) }
  | number EQ access { ($1, Equal, $3) }
  | access LT access { ($1, Less, $3) }
  | access LT number { ($1, Less, $3) }
  | number LT access { ($1, Less, $3) }
  | access LEQ access { ($1, Leq, $3) }
  | access LEQ number { ($1, Leq, $3) }
  | number LEQ access { ($1, Leq, $3) }
  | access GT access { ($1, Greater, $3) }
  | access GT number { ($1, Greater, $3) }
```

```
  | number GT access { ($1, Greater, $3) }
  | access GEQ access { ($1, Geq, $3) }
  | access GEQ number { ($1, Geq, $3) }
  | number GEQ access { ($1, Geq, $3) }

number:
  INT { Int($1) }
  | FLOAT { Float($1) }

access:
  stock ACCESS { Access($1, $2) }

order:
  VAR { Var($1) }
  | INT OF stock { Order($1, $3) }

stock:
  VAR { Var($1) }
  | TICK { Stock($1) }

expression_option:
  /* nothing */ { Noexpr }
  | expression { $1 }

expression:
  number { $1 }
  | STRING  { String($1) }
  | TICK { Stock($1) }
  | VAR  { Var($1) }
  | INT OF stock { Order($1, $3) }
  | MINUS expression { Unop(Neg, $2) }
  | NOT LPAREN expression RPAREN { Unop(Not, $3) }
  | stock ACCESS { Access($1, $2) }
  | expression PLUS expression  { Binop($1, Add, $3) }
  | expression MINUS expression  { Binop($1, Sub, $3) }
  | expression TIMES expression { Binop($1, Mult, $3 ) }
  | expression DIVIDE expression { Binop($1, Div, $3) }
  | expression EQ expression { Binop($1, Equal, $3) }
  | expression LT expression { Binop($1, Less, $3) }
  | expression LEQ expression { Binop($1, Leq, $3) }
  | expression GT expression { Binop($1, Greater, $3) }
  | expression GEQ expression { Binop($1, Geq, $3) }
  | expression MOD expression { Binop($1, Mod, $3) }
  | expression POWER expression { Binop($1, Pow, $3) }
  | expression AND expression { Binop($1, And, $3) }
  | expression OR expression { Binop($1, Or, $3) }
  | VAR ASSIGN expression  { Assign($1, $3) }
  | VAR AASSIGN expression { Aassign($1, $3) }
  | VAR SASSIGN expression { Sassign($1, $3) }
```

```
  | VAR MASSIGN expression { Massign($1, $3) }
  | VAR DASSIGN expression { Dassign($1, $3) }
  | VAR LPAREN args RPAREN { Call($1, $3) }
  | LPAREN expression RPAREN { $2 }

args:
  /* no arguments */ { [] }
  | arg_list { List.rev $1 }

arg_list:
  expression { [$1] }
  | arg_list COMMA expression { $3 :: $1 }

vdecl:
  dtype VAR { { dtype = $1; vname = $2 } }

vdtype:
  VOID { Voidtype }
  | dtype { $1 }

dtype:
  INTD { Inttype }
  | STRINGD { Stringtype }
  | FLOATD { Floattype }
  | STOCK { Stocktype }
  | ORDER  { Ordertype }

fdecl:
  FUNC vdtype VAR LPAREN params RPAREN
  LBRACE statement_list RBRACE SEMI
  {
    {
      rtype = $2;
      name = $3;
      formals = $5;
      body = List.rev $8;
    }
  }

params:
  /* no parameters */ { [] }
  | param_list { List.rev $1 }

param_list:
  vdecl   { [$1] }
  | param_list COMMA vdecl { $3 :: $1 }

statement_list:
  /* nothing */ { [] }
```

```
  | statement_list statement { $2 :: $1 }
```

9.3 AST

ast.ml

```
type binop = Add | Mult | Div | Equal | Neq | Less | Leq | Greater | Geq | Mod | Pow | And |
Or | Sub
type unop = Neg | Not

type data_type =
  Inttype
  | Stringtype
  | Floattype
  | Stocktype
  | Ordertype
  | Voidtype

type var_decl = {
  dtype : data_type;
  vname : string;
}

type expression =
  Int of int
  | String of string
  | Float of float
  | Stock of string
  | Order of int * expression
  | Var of string
  | Unop of unop * expression
  | Access of expression * string
  | Binop of expression * binop * expression
  | Assign of string * expression
  | Aassign of string * expression
  | Sassign of string * expression
  | Massign of string * expression
  | Dassign of string * expression
  | Call of string * expression list
  | Noexpr

type statement =
  Expr of expression
  | While of expression * statement list
  | When of (expression * binop * expression) * statement list
  | If of expression * statement list * statement list
  | Vdecl of var_decl
  | Ret of expression
  | Buy of expression
  | Sell of expression
  | Print of expression
  | Portfolio of string

type func_decl = {
  rtype : data_type;
  name : string;
  formals : var_decl list;
  body : statement list;
```

```
}

type line =
  Stmt of statement
  | Fdecl of func_decl

type program = {
    lines : line list;
}

let string_of_binop = function
  Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | Mod -> "%"
  | Pow -> ", "
  | And -> "&&"
  | Or -> "||"

let string_of_unop = function
  Neg -> "-"
  | Not -> "!"

let string_of_data_type = function
  Inttype -> "int"
  | Stringtype -> "string"
  | Floattype -> "float"
  | Stocktype -> "stock"
  | Ordertype -> "order"
  | Voidtype -> "void"

let rec string_of_expression = function
  Int(i) -> "Int(" ^ string_of_int i ^ ")"
  | String(s) -> "String(" ^ s ^ ")"
  | Float(f) -> "Float(" ^ string_of_float f ^ ")"
  | Stock(stk) -> "Stock(" ^ stk ^ ")"
  | Order(i, e) -> "Order(" ^ string_of_int i ^ ", " ^ string_of_expression e ^ ")"
  | Var(v) -> "Var(" ^ v ^ ")"
  | Unop(op, e) -> "Unop(" ^ string_of_unop op ^ " " ^ string_of_expression e ^ ")"
  | Binop(e1, o, e2) -> "Binop(" ^ string_of_expression e1 ^ " " ^ string_of_binop o ^ " " ^
string_of_expression e2 ^ ")"
  | Access(e, s) -> "Access(" ^ string_of_expression e ^ " -> " ^ s ^ ")"
  | Assign(a, e) -> "Assign(" ^ a ^ " = " ^ string_of_expression e ^ ")"
  | Aassign(aa, e) -> "Aassign(" ^ aa ^ " = " ^ string_of_expression e ^ ")"
  | Sassign(sa, e) -> "Sassign(" ^ sa ^ " = " ^ string_of_expression e ^ ")"
  | Massign(ma, e) -> "Massign(" ^ ma ^ " = " ^ string_of_expression e ^ ")"
  | Dassign(da, e) -> "Dassign(" ^ da ^ " = " ^ string_of_expression e ^ ")"
  | Call(c, el) -> c ^ "(" ^ String.concat ", " (List.map string_of_expression el) ^ ")"
  | Noexpr -> "noexpr"

let string_of_vdecl (vdecl: var_decl) =
  "vdecl{" ^ vdecl.vname ^ " -> " ^ string_of_data_type vdecl.dtype ^ "}"
```

```
let rec string_of_statement = function
  Expr(e) -> "expression{" ^ string_of_expression e ^ "}"
  | If(expr, slst, slst2) -> "if{\n(" ^
                             string_of_expression expr ^
                             ") statementlist{\nstatement{" ^
                             String.concat "}\nstatement{" (List.map string_of_statement
                             slst)^"}\n}\nelse{\nstatementlist{ statement{" ^
                             String.concat "}\nstatement{" (List.map string_of_statement
                             slst2) ^ "\n}\n}\n}"
  | While(expr, slst) -> "while{\n(" ^ string_of_expression expr ^ ")
                             statementlist{\nstatement{" ^
                             String.concat"}\nstatement{"(List.map string_of_statement slst)
^
                             "}\n}\n}"
  | When((e1, op, e2), slst) -> "when{\n(" ^
                                string_of_expression e1 ^
                                ", " ^
                                string_of_binop op ^
                                ", " ^
                                string_of_expression e2 ^
                                ") statementlist{\nstatement{" ^
                                String.concat "}\nstatement{" (List.map string_of_statement
                                slst) ^ "}\n}\n}"
  | Vdecl(v) -> string_of_vdecl v
  | Ret(r) -> "return{" ^ string_of_expression r ^ "}"
  | Buy(b) -> "buy{" ^ string_of_expression b ^ "}"
  | Sell(s) -> "sell{" ^ string_of_expression s ^ "}"
  | Print(e) -> "print{\nexpression{" ^ string_of_expression e ^ "}\n}"
  | Portfolio(str) -> "portfolio{" ^ str ^ "}"

let string_of_fdecl (fdecl: func_decl) =
  "name{" ^
  fdecl.name ^
  "} rtype{" ^
  string_of_data_type fdecl.rtype ^
  "} formals{" ^
  String.concat ", " (List.map string_of_vdecl fdecl.formals) ^
  "}\nbody{\nstatement{" ^
  String.concat "}\nstatement{" (List.map string_of_statement fdecl.body) ^
  "}\n}"

let string_of_line = function
  Stmt(s) -> "statement{" ^ string_of_statement s ^ "}"
  | Fdecl(f) -> "fdecl{\n" ^ string_of_fdecl f ^ "\n}"

let string_of_program (prog: program) =
  "program{\nline{\n" ^ String.concat "\n}\nline{\n" (List.map string_of_line prog.lines) ^
"\n}\n}\n"
```

9.4 Semantic Analyzer

semantics.ml

```
open Ast
open Sast
open String
```

```
exception Except of string

type scope = {
	scope_name : string;
	scope_rtype : Ast.data_type ;
}

type environment = {
	function_table : Sast.sfunc_decl list;
	symbol_table : Ast.var_decl list;
	checked_statements : Sast.sstatement list;
	env_scope : scope;
	returned : bool;
}

let root_env = {
	function_table = [];
	symbol_table = [];
	checked_statements = [];
	env_scope = { scope_name = "reserved"; scope_rtype = Voidtype; };
	returned = false;
}

let name_to_sname env name =
	let sname =
		if name = "reserved" then (raise (Except(
			"'reserved' is a reserved function name!")))
			(* reserved_test.finl *)
		else
			if name = "main" then ("reserved")
			else name
	in
	(try ignore (List.find (fun f -> f.sname = sname) env.function_table);
	raise (Except("Function '" ^ name ^ "' is already defined!"))
	(* overwrite_function_test.finl *)
	with Not_found -> sname)

let formal_to_sformal sformals (formal: Ast.var_decl) =
	let found = List.exists (fun sf -> formal.vname = sf.vname) sformals in
	if found then raise (Except("Formal parameter '" ^ formal.vname ^
	"' is already defined!")) (* bad_formals_test.finl *)
	else formal :: sformals

let analyze_vdecl env (vdecl: Ast.var_decl) =
	let found = List.exists (fun symbol -> symbol.vname = vdecl.vname) env.symbol_table
	in
	if found then raise (Except("Variable '" ^ vdecl.vname ^ "' is already defined!"))
	(* redeclare_variable_test.finl *)
	else vdecl
```

```
let check_for_main name =
	let new_name =
		if name = "main" then "reserved"
		else name
	in new_name

let check_for_reserved sname =
	let new_name =
		if sname = "reserved" then "main"
		else sname
	in new_name

let check_assign env var (sexpression: Sast.sexpression) =
	(try let vdecl = List.find (fun s -> s.vname = var) env.symbol_table in
		let dtype = vdecl.dtype in
		let etype = sexpression.sdtype in
		let sametype = dtype = etype in
			if sametype then (sexpression)
			else let dstring = Ast.string_of_data_type dtype
				and estring = Ast.string_of_data_type etype in
				raise (Except("Symbol '" ^ var ^ "' is of type '" ^ dstring ^
				"', not of type '" ^ estring ^ "'."))
				(* assign_type_mismatch_test.finl *)
	with Not_found -> raise (Except("Symbol '" ^ var ^ "' not initialized!")))
	(* uninitialized_assignment_test.finl *)

let var_to_svar env name =
	(try let var = List.find (fun s -> s.vname = name) env.symbol_table in
		{ sexpr = Svar(name); sdtype = var.dtype; }
	with Not_found -> raise (Except("Symbol '" ^ name ^ "' is uninitialized!")))
	(* uninitialized_variable_test.finl *)

let check_string_binop (sexpr1: Sast.sexpression) (op: Ast.binop) (sexpr2: Sast.sexpression)
=
	match op with
		Add -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Stringtype; }
		| Equal -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
		| Less -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
		| Leq -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
		| Greater -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
		| Geq -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
		| _ -> raise (Except("Operator '" ^ Ast.string_of_binop op ^
			"' is not supported for strings!"))

let check_number_binop (sexpr1: Sast.sexpression) (op: Ast.binop) (sexpr2: Sast.sexpression)
=
	match op with
		Equal -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
```

```
            | Less -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | Leq -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | Greater -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | Geq -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | And -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | Or -> { sexpr = Sbinop(sexpr1, op, sexpr2); sdtype = Inttype; }
            | _ -> (let typ = match sexpr1.sdtype with
                                        Floattype -> Floattype
                                        | _ -> let is_int = sexpr2.sdtype =
                                               Inttype in
                                               if is_int then (Inttype) else
                                               Floattype
                                               in { sexpr = Sbinop(sexpr1, op,
                                               sexpr2); sdtype = typ; })

let binop_to_sbinop (sexpr1: Sast.sexpression) (op: Ast.binop) (sexpr2: Sast.sexpression) =
       let type1 = sexpr1.sdtype
       and type2 = sexpr2.sdtype in
       match (type1, type2) with
              (Stringtype, Stringtype) -> check_string_binop sexpr1 op sexpr2
              | (t, Stringtype) -> raise (Except("Cannot do binary operation on type '" ^
                                   Ast.string_of_data_type t ^ "' and type 'string'!"))
              | (Stringtype, t) -> raise (Except(
                                  "Cannot do binary operation on type 'string' and type '"
                                   ^ Ast.string_of_data_type t ^ "'."))
              | (Stocktype, t) -> raise (Except(
                                  "Type 'stock' does not support binary operations!."))
              | (t, Stocktype) -> raise (Except(
                                  "Type 'stock' does not support binary operations!."))
              | (Ordertype, t) -> raise (Except(
                                  "Type 'order' does not support binary operations!."))
              | (t, Ordertype) -> raise (Except(
                                  "Type 'order' does not support binary operations!."))
              | (_, _) ->    check_number_binop sexpr1 op sexpr2

let unop_to_sunop env (u: Ast.unop) (se: Sast.sexpression) =
              let typ = se.sdtype in
              match typ with
                     Inttype -> { sexpr = Sunop(u, se); sdtype = Inttype; }
                     | Floattype -> { sexpr = Sunop(u, se); sdtype = Floattype; }
                     | t -> raise (Except("Unary operations are not supported for type '" ^
                            Ast.string_of_data_type typ ^ "'!"))

let split_access acc =
       sub acc 1 ((length acc) - 2)

let rec expression_to_sexpression env (expression: Ast.expression) =
       match expression with
              Int(i) -> { sexpr = Sint(i); sdtype = Inttype; }
```

```
| String(s) -> { sexpr = Sstring(s); sdtype = Stringtype; }
| Float(f) ->  { sexpr = Sfloat(f); sdtype = Floattype; }
| Stock(stk) -> { sexpr = Sstock(stk); sdtype = Stocktype; }
| Order(i, ord) -> let new_ord = (expression_to_sexpression env ord) in
                   if new_ord.sdtype = Stocktype then
                   ({ sexpr = Sorder(i, new_ord); sdtype = Ordertype; })
                   else raise (Except("Invalid order!"))

| Var(v) -> var_to_svar env v

| Unop(op, e) -> unop_to_sunop env op (expression_to_sexpression env e)

| Binop(e1, o, e2) -> let se1 = expression_to_sexpression env e1 in
                      let se2 = expression_to_sexpression env e2 in
                      binop_to_sbinop se1 o se2

| Access(e, s) -> let acc = split_access s and checked_expression =
                  expression_to_sexpression env e in
                  { sexpr = Saccess(checked_expression, acc);
                  sdtype = Floattype; }

| Assign(a, e) -> let sexpression = expression_to_sexpression env e in
                  let checked_sexpression = check_assign env a sexpression in
                  { sexpr = Sassign(a, checked_sexpression);
                  sdtype = checked_sexpression.sdtype; }

| Aassign(aa, e) -> let sexpression = expression_to_sexpression env e in
                    let checked_sexpression = check_assign env aa sexpression
                    in
                    let typ = checked_sexpression.sdtype in
                    if typ <> Inttype && typ <> Floattype &&
                    typ <> Stringtype then
                    (raise (Except("Add assignment is undefined for type '" ^
                    Ast.string_of_data_type typ ^ "'!")))
                    else { sexpr = Saassign(aa, checked_sexpression);
                    sdtype = checked_sexpression.sdtype; }

| Sassign(sa, e) -> let sexpression = expression_to_sexpression env e in
                    let checked_sexpression = check_assign env sa sexpression
                    in
                    let typ = checked_sexpression.sdtype in
                    if typ <> Inttype && typ <> Floattype then
                    (raise (Except("Add assignment is undefined for type '" ^
                    Ast.string_of_data_type typ ^ "'!")))
                    else { sexpr = Ssassign(sa, checked_sexpression);
                    sdtype = checked_sexpression.sdtype; }

| Massign(ma, e) -> let sexpression = expression_to_sexpression env e in
                    let checked_sexpression = check_assign env ma sexpression
```

```
                                in let typ = checked_sexpression.sdtype in
                                if typ <> Inttype && typ <> Floattype then
                                (raise (Except("Add assignment is undefined for type '" ^
                                Ast.string_of_data_type typ ^ "'!")))
                                else { sexpr = Smassign(ma, checked_sexpression);
                                sdtype = checked_sexpression.sdtype; }

            | Dassign(da, e) -> let sexpression = expression_to_sexpression env e in
                                let checked_sexpression = check_assign env da sexpression
                                in let typ = checked_sexpression.sdtype in
                                if typ <> Inttype && typ <> Floattype then
                                (raise (Except("Add assignment is undefined for type '" ^
                                Ast.string_of_data_type typ ^ "'!")))
                                else { sexpr = Sdassign(da, checked_sexpression);
                                sdtype = checked_sexpression.sdtype; }


            | Call(c, el) -> let sname = check_for_main c in (* no_return_test.finl *)
                            (try let func = List.find (fun f -> f.sname = sname)
                             env.function_table in
                             (try let new_el = List.map2
                             (fun f e -> let sexpr = expression_to_sexpression env e in
                             if f.dtype <> sexpr.sdtype then (raise
                             (Except("Function parameter type mismatch!")))
                             (* parameter_type_mismatch_test.finl *)
                             else sexpr) func.sformals el in
                             { sexpr = Scall(sname, new_el); sdtype = func.srtype; }
                             with Invalid_argument _ -> raise
                             (Except("Wrong argument length to function '" ^
                             check_for_reserved func.sname ^ "'.")))
                             (* arg_length_test.finl *)
                             with Not_found -> raise
                             (Except("Function '" ^ c ^ "' not found!")))
                             (* uninitialized_call_test.finl *)
            | Noexpr -> { sexpr = Snoexpr; sdtype = Voidtype; }

let check_estatement (sexpr: Sast.sexpression) =
       match sexpr.sexpr with
              Sassign(a, e) -> sexpr
              | Saassign(aa, e1) -> sexpr
              | Ssassign(sa, e2) -> sexpr
              | Smassign(ma, e3) -> sexpr
              | Sdassign(da, e4) -> sexpr
              | Scall(c, el) -> sexpr
              | _ -> raise (Except("Not a statement!")) (* statement_test.finl *)

let check_return env (sexpression: Sast.sexpression) =
       let scope = env.env_scope in
       let fname = scope.scope_name in
```

```
        let within_func = fname <> "reserved" in
        if within_func then
                (let rtype = scope.scope_rtype in
                let etype = sexpression.sdtype in
                let sametype = rtype = etype in
                if sametype then (sexpression)
                else raise (Except("Function '" ^ fname ^ "' returns type '" ^
                 Ast.string_of_data_type rtype ^ "', not type '" ^
                 Ast.string_of_data_type etype ^ "'."))) (* bad_return_test.finl *)
        else raise (Except("Return statements cannot be used outside of functions!"))
        (* outside_return_test.finl *)

let rec statement_to_sstatement env (statement: Ast.statement) =
        if env.returned then (raise (Except("Unreachable statement!")))
        else match statement with
                If(ex, sl, els) -> let checked_expression = expression_to_sexpression env ex
                                   in let typ = checked_expression.sdtype in
                                   if typ <> Inttype && typ <> Floattype then
                                   (raise (Except(
                                   "If expressions only take numerical types!")))
                                   else let if_env = { function_table = env.function_table;
                                                       symbol_table = env.symbol_table;
                                                       checked_statements = [];
                                                       env_scope = env.env_scope;
                                                       returned = false; }
                                    in let else_env = if_env in
                                    let if_env =
                                    List.fold_left statement_to_sstatement if_env sl in
                                    let else_env =
                                    List.fold_left statement_to_sstatement else_env els in
                                    let checked_statement =
                                    Sif(checked_expression, if_env.checked_statements,
                                    else_env.checked_statements) in
                                    let new_env = { function_table = env.function_table;
                                                    symbol_table = env.symbol_table;
                                                    checked_statements = checked_statement ::
                                                    env.checked_statements;
                                                    env_scope = env.env_scope;
                                                    returned = if_env.returned &&
                                                    else_env.returned; }
                                                in new_env

                | While(ex, sl) -> let checked_expression = expression_to_sexpression env ex
                                   in let typ = checked_expression.sdtype in
                                   if typ <> Inttype && typ <> Floattype then (raise
                                   (Except("While expressions only take numerical types!")))
                                   else let while_env = { function_table =
                                                          env.function_table;
                                                          symbol_table = env.symbol_table;
```

```
                                                checked_statements = [];
                                                env_scope = env.env_scope;
                                                returned = false; }
                                in let while_env =
                                List.fold_left statement_to_sstatement while_env sl in
                                let checked_statement =
                                Swhile(checked_expression, while_env.checked_statements) in
                                let new_env = { function_table = env.function_table;
                                                symbol_table = env.symbol_table;
                                                checked_statements = checked_statement ::
                                                env.checked_statements;
                                                env_scope = env.env_scope;
                                                returned = while_env.returned; }
                                 in new_env
| When((e1, op, e2), sl) -> let checked_expression1 = expression_to_sexpression env e1
                            and checked_expression2 = expression_to_sexpression env e2
                            in let when_env = { function_table = env.function_table;
                                                symbol_table = env.symbol_table;
                                                checked_statements = [];
                                                env_scope = env.env_scope;
                                                returned = false; }
                           in
                           let when_env =
                           List.fold_left statement_to_sstatement when_env sl in
                           if when_env.returned then (raise
                           (Except("When loops should not return!"))
                           else let checked_statement =
                           Swhen((checked_expression1, op, checked_expression2),
                           when_env.checked_statements) in
                           let new_env = { function_table = env.function_table;
                                           symbol_table = env.symbol_table;
                                           checked_statements = checked_statement ::
                                           env.checked_statements;
                                           env_scope = env.env_scope;
                                           returned = false; }
                           in new_env

             | Expr(e) -> let checked_expression = expression_to_sexpression env e in
                          let checked_statement =
                          Sexpr(check_estatement checked_expression) in
                          let new_env = { function_table = env.function_table;
                                          symbol_table = env.symbol_table;
                                          checked_statements = checked_statement ::
                                          env.checked_statements;
                                          env_scope = env.env_scope;
                                          returned = false; }
                          in new_env
             | Vdecl(v) -> let checked_vdecl = analyze_vdecl env v in
                           let checked_statement = Svdecl(checked_vdecl) in
```

```
                                let new_env = { function_table = env.function_table;
                                                symbol_table = checked_vdecl ::
                                                env.symbol_table;
                                                checked_statements = checked_statement ::
                                                env.checked_statements;
                                                env_scope = env.env_scope;
                                                returned = false; }
                                in new_env
| Ret(r) -> let checked_expression = check_return env (expression_to_sexpression env r) in
      (* break out of scope when return is found FLAG IN ENV?? *)
    let checked_statement = Sret(checked_expression) in
      let new_env = { function_table = env.function_table;
                      symbol_table = env.symbol_table;
                      checked_statements = checked_statement :: env.checked_statements;
                      env_scope = env.env_scope;
                      returned = true; }
      in new_env
| Buy(b) -> let checked_expression = expression_to_sexpression env b in
      if checked_expression.sdtype <> Ordertype then
           (raise (Except("'buy' keyword takes type 'order'!")))
      else let checked_statement = Sbuy(checked_expression) in
      let new_env = { function_table = env.function_table;
                      symbol_table = env.symbol_table;
                      checked_statements = checked_statement :: env.checked_statements;
                      env_scope = env.env_scope;
                      returned = false; }
      in new_env
| Sell(s) -> let checked_expression = expression_to_sexpression env s in
       if checked_expression.sdtype <> Ordertype then
              (raise (Except("'buy' keyword takes type 'order'!")))
      else let checked_statement = Ssell(checked_expression) in
      let new_env = { function_table = env.function_table;
                      symbol_table = env.symbol_table;
                      checked_statements = checked_statement :: env.checked_statements;
                      env_scope = env.env_scope;
                      returned = false; }
      in new_env
| Print(ex) -> let checked_expression = expression_to_sexpression env ex in
      if checked_expression.sdtype = Voidtype && checked_expression.sexpr <> Snoexpr then
             (raise (Except("Cannot print 'void' type!")))
      else let checked_statement = Sprint(checked_expression) in
      let new_env = { function_table = env.function_table;
                        symbol_table = env.symbol_table;
                        checked_statements = checked_statement :: env.checked_statements;
                        env_scope = env.env_scope;
                        returned = false; }
      in new_env
| Portfolio(str) -> let checked_portfolio = Sportfolio(str) in
      let new_env = { function_table = env.function_table;
```

```
                        symbol_table = env.symbol_table;
                        checked_statements = checked_portfolio :: env.checked_statements;
                        env_scope = env.env_scope;
                        returned = false; }
        in new_env

let fdecl_to_sfdecl env (fdecl: Ast.func_decl) = (* multiple_return_test.finl NEEDS FIX *)
        let checked_name = name_to_sname env fdecl.name in
        let checked_formals = List.fold_left formal_to_sformal [] fdecl.formals in
        let func_env = { function_table = env.function_table;
                        symbol_table = checked_formals;
                        checked_statements = [];
                        env_scope = { scope_name = fdecl.name; scope_rtype = fdecl.rtype; };
                        returned = false; }
        in
        let func_env = List.fold_left statement_to_sstatement func_env fdecl.body in
        let void = fdecl.rtype = Voidtype
        and returns = func_env.returned in
        if (void && not returns) || returns then
                (let sfdecl = { sname = checked_name;
                                        sformals = List.rev checked_formals;
                                        sbody = List.rev func_env.checked_statements;
                                        srtype = fdecl.rtype; }
                in
                let new_env = { function_table = sfdecl :: env.function_table;
                                symbol_table = env.symbol_table;
                                checked_statements = env.checked_statements;
                                env_scope ={ scope_name= "reserved"; scope_rtype = Voidtype; };
                                returned = false; }
                in new_env)
        else (raise (Except("Function '" ^ fdecl.name ^
            "' does not have a return statement!"))) (* no_return_test.finl *)

let analyze_line env (line: Ast.line) =
        match line with
                Stmt(s) -> statement_to_sstatement env s
                | Fdecl(f) -> fdecl_to_sfdecl env f

let analyze (program: Ast.program) =
        let new_env = List.fold_left analyze_line root_env program.lines in
        let sprogram = { sfunc_decls = new_env.function_table;
                        sstatements = List.rev new_env.checked_statements; }
        in sprogram
```

9.5 SAST

sast.ml

```
open Ast
```

```
type expr =
  Sint of int
  | Sstring of string
  | Sfloat of float
  | Sstock of string
  | Sorder of int * sexpression
  | Svar of string
  | Sunop of Ast.unop * sexpression
  | Sbinop of sexpression * Ast.binop * sexpression
  | Saccess of sexpression * string
  | Sassign of string * sexpression
  | Saassign of string * sexpression
  | Ssassign of string * sexpression
  | Smassign of string * sexpression
  | Sdassign of string * sexpression
  | Scall of string * sexpression list
  | Snoexpr
and sexpression = {
  sexpr : expr;
  sdtype : Ast.data_type;
}

type sstatement =
  Sexpr of sexpression
  | Sif of sexpression * sstatement list * sstatement list
  | Swhile of sexpression * sstatement list
  | Swhen of (sexpression * Ast.binop * sexpression) * sstatement list
  | Svdecl of Ast.var_decl
  | Sret of sexpression
  | Sbuy of sexpression
  | Ssell of sexpression
  | Sprint of sexpression
  | Sportfolio of string

type sfunc_decl = {
  srtype : Ast.data_type;
  sname : string;
  sformals : Ast.var_decl list;
  sbody : sstatement list;
}

type sprogram = {
  sfunc_decls : sfunc_decl list;
  sstatements : sstatement list;
}

let rec string_of_sexpression (sexpr: sexpression) =
  let expr = match sexpr.sexpr with
   Sint(i) -> "Sint(" ^ string_of_int i ^ ")"
```

```
    | Sstring(s) -> "Sstring(" ^ s ^ ")"
    | Sfloat(f) -> "Sfloat(" ^ string_of_float f ^ ")"
    | Sstock (stk) -> "Sstock(" ^ stk ^ ")"
    | Sorder (i, ord) -> "Sorder(" ^ string_of_int i ^ " of " ^ string_of_sexpression ord ^
                         ")"
    | Svar(v) -> "Svar(" ^ v ^ ")"
    | Sunop(op, se) -> "Sunop(" ^ Ast.string_of_unop op ^ " " ^ string_of_sexpression se ^
                       ")"
    | Sbinop(e1, o, e2) -> "Sbinop(" ^ string_of_sexpression e1 ^ " " ^ Ast.string_of_binop o
                           ^ " " ^ string_of_sexpression e2 ^ ")"
    | Saccess(e, s) -> "Saccess(" ^ string_of_sexpression e ^ " -> " ^ s ^ ")"
    | Sassign(a, e) -> "Sassign(" ^ a ^ " = " ^ string_of_sexpression e ^ ")"
    | Saassign(aa, e) -> "Saassign(" ^ aa ^ " = " ^ string_of_sexpression e ^ ")"
    | Ssassign(sa, e) -> "Ssassign(" ^ sa ^ " = " ^ string_of_sexpression e ^ ")"
    | Smassign(ma, e) -> "Smassign(" ^ ma ^ " = " ^ string_of_sexpression e ^ ")"
    | Sdassign(da, e) -> "Sdassign(" ^ da ^ " = " ^ string_of_sexpression e ^ ")"
    | Scall(c, el) -> c ^ "(" ^ String.concat ", " (List.map string_of_sexpression el) ^ ")"
    | Snoexpr -> ""
  in expr ^ " -> " ^ Ast.string_of_data_type sexpr.sdtype

let rec string_of_sstatement = function
  Sexpr(e) -> "sexpression{" ^ string_of_sexpression e ^ "}"
  | Sif(sexpr, ssl, ssl2) -> "sif{ sstatement{" ^ String.concat "} sstatement{"
                             (List.map string_of_sstatement ssl) ^ "} else{ sstatement{" ^
                             String.concat "} sstatement{"
                             (List.map string_of_sstatement ssl2) ^ "}}"
  | Swhile(sexpr, ssl) -> "swhile{ sstatement{" ^ String.concat "} sstatement{"
                          (List.map string_of_sstatement ssl) ^ "}}"
  | Swhen((se1, op, se2), ssl) -> "swhen{ (" ^ string_of_sexpression se1 ^ " " ^
                                  Ast.string_of_binop op ^ " " ^ string_of_sexpression se2 ^
                                  ") sstatement{" ^ String.concat "} sstatement{"
                                  (List.map string_of_sstatement ssl) ^ "}}"
  | Svdecl(v) -> Ast.string_of_vdecl v
  | Sret(r) -> "sreturn{ sexpression{" ^ string_of_sexpression r ^ "}}"
  | Sbuy(b) -> "sbuy{ sexpression{" ^ string_of_sexpression b ^ "}}"
  | Ssell(s) -> "ssell{ sexpression{" ^ string_of_sexpression s ^ "}}"
  | Sprint(p) -> "sprint{ sexpression{" ^ string_of_sexpression p ^ "}}"
  | Sportfolio(str) -> "sportfolio{" ^ str ^ "}"

let string_of_sfdecl (sfdecl: sfunc_decl) =
  "sname{" ^
  sfdecl.sname ^
  "} srtype{" ^
  Ast.string_of_data_type sfdecl.srtype ^
  "} sformals{" ^
  String.concat ", " (List.map Ast.string_of_vdecl sfdecl.sformals) ^
  "}\nsbody{\nsstatement{" ^
  String.concat "}\nsstatement{" (List.map string_of_sstatement sfdecl.sbody) ^
  "}\n}"
```

```
let string_of_sprogram (sprog: sprogram) =
  "sprogram{\nsfunc_decls{\nsfunc_decl{\n" ^
  String.concat "\n}\nsfunc_decl{\n" (List.map string_of_sfdecl sprog.sfunc_decls) ^
  "\n}\n}\nsstatements{\nsstatement{" ^
  String.concat "}\nsstatement{" (List.map string_of_sstatement sprog.sstatements) ^
  "}\n}\n}\n"
```

9.6 Compiler

compile.ml

```
open Ast
open Sast
open String

let num_whens = ref 0

let compile_dtype = function
  Inttype -> "int"
  | Stringtype -> "String"
  | Floattype -> "double"
  | Stocktype -> "FinlStock"
  | Ordertype -> "FinlOrder"
  | Voidtype -> "void"

let compile_vdecl (vdecl: Ast.var_decl) =
  compile_dtype vdecl.dtype ^ " " ^ vdecl.vname

let boolean_to_sexpr bool_string =
  "FinlLib.boolean_to_int(" ^ bool_string ^ ")"

let sexpr_to_boolean sexpr_string =
  "FinlLib.num_to_boolean(" ^ sexpr_string ^ ")"

let compile_compare sexpr_string1 op_string sexpr_string2 is_string_compare =
  let str = if is_string_compare then ("FinlLib.compare_strings(" ^ sexpr_string1 ^ ", \"" ^
                                       op_string ^ "\", " ^ sexpr_string2 ^ ")")
            else sexpr_string1 ^ " " ^ op_string ^ " " ^ sexpr_string2
  in boolean_to_sexpr str

let string_of_stock ticker =
  let len = length ticker in
  let new_ticker = sub ticker 1 (len - 1) in
  "new FinlStock(\"" ^ new_ticker ^ "\")"

let string_of_order amt ord =
  let new_ord = match ord.sexpr with
    Svar(var) -> var
    | Sstock(stk) -> string_of_stock stk
```

```
    | _ -> "" (* parser should not allow any other tokens *)
  in "new FinlOrder(" ^ string_of_int amt ^ ", " ^ new_ord ^ ")"

let rec compile_sexpression (sexpr: Sast.sexpression) =
  match sexpr.sexpr with
    Sstring(str) -> str
    | Sint(i) -> string_of_int i
    | Sfloat(f) -> string_of_float f
    | Sstock(stk) -> string_of_stock stk
    | Sorder(i, ord) -> string_of_order i ord
    | Sunop(op, expr) -> (let unop = Ast.string_of_unop op in
                          match op with
                           Neg -> unop ^ compile_sexpression expr
                           | Not -> boolean_to_sexpr
                                    (unop ^ sexpr_to_boolean (compile_sexpression expr)))
    | Sbinop(expr1, op, expr2) -> (match op with
                                    Pow -> "Math.pow(" ^ compile_sexpression expr1 ^
                                           Ast.string_of_binop op ^
                                           compile_sexpression expr2 ^ ")"
                                    | Equal -> compile_compare (compile_sexpression expr1)
                                               (Ast.string_of_binop op)
                                               (compile_sexpression expr2)
                                               (if expr1.sdtype = Stringtype then (true)
                                               else false)
                                    | Less -> compile_compare (compile_sexpression expr1)
                                              (Ast.string_of_binop op)
                                              (compile_sexpression expr2)
                                              (if expr1.sdtype = Stringtype then (true)
                                              else false)
                                    | Leq -> compile_compare (compile_sexpression expr1)
                                             (Ast.string_of_binop op)
                                             (compile_sexpression expr2)
                                             (if expr1.sdtype = Stringtype then (true)
                                             else false)
                                    | Greater -> compile_compare (compile_sexpression expr1)
                                                 (Ast.string_of_binop op)
                                                 (compile_sexpression expr2)
                                                 (if expr1.sdtype = Stringtype then (true)
                                                 else false)
                                    | Geq -> compile_compare (compile_sexpression expr1)
                                             (Ast.string_of_binop op)
                                             (compile_sexpression expr2)
                                             (if expr1.sdtype = Stringtype then (true)
                                             else false)
                                    | And -> boolean_to_sexpr
                                             (sexpr_to_boolean (compile_sexpression expr1) ^
                                             " " ^ Ast.string_of_binop op ^ " " ^
                                             sexpr_to_boolean (compile_sexpression expr2))
                                    | Or -> boolean_to_sexpr
```

```
                                        (sexpr_to_boolean (compile_sexpression expr1) ^
                                        " " ^ Ast.string_of_binop op ^ " " ^
                                        sexpr_to_boolean (compile_sexpression expr2))
                                    | _ -> compile_sexpression expr1 ^ " " ^
Ast.string_of_binop op ^ " " ^ compile_sexpression expr2)
    | Saccess(expr, str) -> compile_sexpression expr ^ ".getRequest(\"" ^ str ^ "\")"
    | Sassign(var, expr) -> var ^ " = " ^ compile_sexpression expr
    | Saassign(avar, aexpr) -> avar ^ " += " ^ compile_sexpression aexpr
    | Ssassign(svar, sexpr) -> svar ^ " -= " ^ compile_sexpression sexpr
    | Smassign(mvar, mexpr) -> mvar ^ " *= " ^ compile_sexpression mexpr
    | Sdassign(dvar, dexpr) -> dvar ^ " /= " ^ compile_sexpression dexpr
    | Svar(str) -> str
    | Scall(name, exprlst) -> name ^ "(" ^ String.concat ", "
                              (List.map compile_sexpression exprlst) ^ ")"
    | Snoexpr -> ""

let rec compile_sstatement = function
  Sexpr(expr) -> compile_sexpression expr ^ ";"
  | Sif(e, sl, sl2) -> let els = if (List.length sl2) = 0 then ("")
                       else "else {" ^ String.concat "\n" (List.map compile_sstatement sl2)
                       ^ "}" in
                       "if (" ^
                       sexpr_to_boolean (compile_sexpression e) ^
                       ") {\n" ^
                       String.concat "\n" (List.map compile_sstatement sl) ^
                       "\n} " ^
                       els
  | Swhile(e, sl) -> "while (" ^
                     sexpr_to_boolean (compile_sexpression e) ^
                     ") {\n" ^
                     String.concat "\n" (List.map compile_sstatement sl) ^
                     "\n}"
  | Swhen((e1, op, e2), sl) -> num_whens := !num_whens + 1;
                               "Thread when" ^
                               string_of_int !num_whens ^
                               " = new Thread(new Runnable()
                               {\npublic void run(){\nwhile (true) {\n if (" ^
                               compile_sexpression e1 ^
                               " " ^
                               Ast.string_of_binop op ^
                               " " ^
                               compile_sexpression e2 ^
                               ") {\n" ^
                               String.concat "\n" (List.map compile_sstatement sl) ^
                               "\nbreak;\n}" ^
                               "\ntry{ Thread.sleep(10000); } " ^
                               "catch (InterruptedException ie) " ^
                               "{ System.out.println(\"Program execution interrupted!\");
                               }\n}\n}\n});" ^
```

```
                              "\nwhen" ^
                              string_of_int !num_whens ^
                              ".start();"
  | Svdecl(v) -> compile_vdecl v ^ ";"
  | Sret(r) -> "return " ^ compile_sexpression r ^ ";"
  | Sbuy(b) -> "portfolio.buy(" ^ compile_sexpression b ^ ");"
  | Ssell(s) -> "portfolio.sell(" ^ compile_sexpression s ^ ");"
  | Sprint(e) -> (match e.sdtype with
                    Stocktype -> compile_sexpression e ^ ".printStock();"
                    | Ordertype -> compile_sexpression e ^ ".printOrder();"
                    | Voidtype -> "portfolio.printHoldings();"
                    | _ -> "System.out.println(" ^ compile_sexpression e ^ ");")
  | Sportfolio(str) -> "portfolio = portfolio.switchWith(" ^ str ^ ");"

let compile_sfdecl (func: Sast.sfunc_decl) =
  "public static " ^
  compile_dtype func.srtype ^
  " " ^
  func.sname ^
  "(" ^
  String.concat ", " (List.map compile_vdecl func.sformals) ^
  ") {\n" ^
  String.concat "\n" (List.map compile_sstatement func.sbody) ^
  "\n}"

let compile (sprogram: Sast.sprogram) (filename: string) =
  "import java.lang.Math;\n" ^
  "import bin.*;\n" ^
  "\npublic class " ^
  filename ^
  " {\n" ^
  "\npublic static FinlPortfolio portfolio;\n\n" ^
  String.concat "\n" (List.map compile_sfdecl sprogram.sfunc_decls) ^
  "\npublic static void main(String[] args) {\n" ^
  "if (args.length > 0) {\n" ^
  "portfolio = new FinlPortfolio(args[0]);\n" ^
  "portfolio.csvPortfolioBuilder(); }\n" ^
  "else { portfolio = new FinlPortfolio(\"default\"); }\n" ^
  String.concat "\n" (List.map compile_sstatement sprogram.sstatements) ^
  "\nportfolio.csvExport();\n}\n}"
```

9.7 Top-Level

finlc.ml

```
open Printf
open String
```

```
type action = Ast | Sast | Compile

let _ =
       let action =
              if Array.length Sys.argv = 3 then
      List.assoc Sys.argv.(1) [ ("-a", Ast); ("-s", Sast); ("-c", Compile)]
    else Compile in
    let file_index = (Array.length Sys.argv) - 1 in
      let read_file = open_in Sys.argv.(file_index) in
      let lexbuf = Lexing.from_channel read_file in
      let program = Parser.program Scanner.token lexbuf in
      match action with
        Ast -> print_string (Ast.string_of_program program)
        | _ ->
          let checked_program = Semantics.analyze program in
          (match action with
            Sast -> print_string (Sast.string_of_sprogram checked_program)
            | _ ->  close_in read_file;
                    let last_slash = try rindex Sys.argv.(1) '/' with Not_found -> -1 in
                    let start = last_slash + 1 in
                    let name = sub Sys.argv.(file_index) start ((length
                        Sys.argv.(file_index)) - 5 - start) in
                    let write_file = open_out (name ^ ".java") in
                    let compiled_program = Compile.compile checked_program name in
                    fprintf write_file "%s" compiled_program;
                    close_out write_file;
                    ignore (Sys.command ("javac " ^ name ^ ".java")))
              (*else (*Compile in*) print_endline "Please specify exactly 1 filename."*)
```

9.8 Scripts

9.8.1 Makefile

Makefile

```
.PHONY : make
make :
       javac -cp ../YahooFinanceAPI-2.0.0.jar bin/*
       ocamlc -c ast.ml
       ocamlc -c sast.ml
       ocamlyacc -v parser.mly
       ocamlc -c parser.mli
       ocamlc -c parser.ml
       ocamllex scanner.mll
       ocamlc -c scanner.ml
       ocamlc -c semantics.ml
       ocamlc -c compile.ml
       ocamlc -c finlc.ml
       ocamlc -o finlc ast.cmo sast.cmo parser.cmo scanner.cmo compile.cmo semantics.cmo
finlc.cmo

.PHONY : clean
clean :
       rm -f finlc parser.ml parser.mli scanner.ml bin/*.class \
       *.cmo *.cmi *.output *.class *.java *.log *.csv
```

9.8.2 Regression Test

regression_test.sh

```
#!/bin/bash

NC='\033[0m'
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'

for file in ../test_suite/*.finl
do
        FILENAME=${file##*/}
        printf "${YELLOW}Compiling $FILENAME ...${NC}\n"
        EXECUTABLE="${FILENAME%.*}"
        ./finlc $file > "$EXECUTABLE.log" 2>&1
        printf "${YELLOW}Running $EXECUTABLE ...${NC}\n"
        ./finl.sh $EXECUTABLE >> "$EXECUTABLE.log" 2>&1
        DIFF=$(diff $EXECUTABLE.log ../test_suite/$EXECUTABLE.out)
        if [ "$DIFF" = "" ]
                then printf "${GREEN}$EXECUTABLE passed.${NC}\n"
        else printf "${RED}$EXECUTABLE failed.${NC}\n"
        fi
done
```

9.8.3 Running finl Programs

finl.sh

```
#!/bin/sh
java -cp .:../YahooFinanceAPI-2.0.0.jar $1 $2
```

9.9 Java Libraries

FinlLib.java

```
package bin;

public class FinlLib {

        public static boolean num_to_boolean(int x) {
                if (x != 0) { return true; } else { return false; }
        }

        public static boolean num_to_boolean(double x) {
                if (x != 0) { return true; } else { return false; }
        }

        public static int boolean_to_int(boolean b) {
                if (b) { return 1; } else { return 0; }
        }

        public static boolean compare_strings(String s1, String op, String s2) {
                if (s1.equals(s2)) {
                        switch (op) {
                                case "==":return true;
                                case "<":return false;
```

```
                                case "<=":return true;
                                case ">":return false;
                                case ">=":return true;
                        }
                } else {
        switch (op) {
                case "==":return false;
                case "<": if (s1.compareTo(s2) < 0) { return false; } else { return true; }
                case "<=": if (s1.compareTo(s2) < 0) { return false; } else { return true; }
                case ">":if (s1.compareTo(s2) > 0) { return false; } else { return true; }
                case ">=":if (s1.compareTo(s2) > 0) { return false; } else { return true; }
        }
       }
      return false;
     }
}
```

FinlOrder.java

```
package bin;


public class FinlOrder {

          public int size                                   = 0;
          public FinlStock stock                = null;
          public double sharePrice      = 0.0;
          public java.util.Date date = null;
          private String type;
          private boolean execute               = false;

          ////////////////////////////////////////
          //////////*FinlOrder Constructors*//////////
          ////////////////////////////////////////

          /* automatically executes */
          public FinlOrder(int size, FinlStock stock) {
                    this.size = size;
                    this.stock = stock;
                    if(!this.stock.valid) {               //stock is not a valid stock
                              this.execute = true;        //order cannot be executed
                    }
          }

          public FinlOrder() {        //constructor for building from PDF

          }

          public void printOrder() {
                    if(this.execute) {
                              System.out.println("\n\n" + this.stock.companyName
                                                  + " (" + this.type.toUpperCase() + " Order Executed)"
                                                  + "\n_____________________________________\n");
                              System.out.println("Symbol:\t\t\t" + this.stock.symbol);
                              System.out.println("Order Size Value:\t" + this.size);
                              System.out.format("Execution Price:\t$%.2f\n", this.sharePrice);
                              System.out.println("Trade Date:\t\t" + this.date.toString()
                              + "\n_____________________________________\n\n");
                    }
                    else {
                              System.out.println("\n\n" + this.stock.companyName
                                                  + "(Order Not Yet Executed)"
                                                  + "\n_____________________________________\n");
                              System.out.println("Symbol:\t\t\t" + this.stock.symbol);
                              System.out.println("Order Size Value:\t" + this.size);
                              System.out.println("Order Type:\t\tOrder Not Yet Executed");
                              System.out.println("Execution Price:\tOrder Not Yet Executed");
                              System.out.println("Trade Date:\t\tOrder Not Yet Executed"
                                                  + "\n_____________________________________\n\n");
                    }
```

```
                System.out.flush();
        }

        //getters and setters//
        public String getType() {
                return this.type;
        }

        public void setType(String type) {
                this.type = type;
        }

        //gets whether the order has been executed
        public boolean getExecute() {
                return this.execute;
        }
        public void setExecute(boolean execute) {
                this.execute = execute;
        }
}
```

FinlPortfolio.java

```
package bin;


import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.text.DateFormat;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;


public class FinlPortfolio {

        DateFormat dateFormat                 = new SimpleDateFormat("yyyy/MM/dd HH:mm:ss");

        private ArrayList<FinlOrder> orders;            //list of all orders
        private ArrayList<Holding> holdings;            //list of all positions

        private double accountValue = 0.0;
        public String portfolioName = "defaultPortfolio";
        private String csvName = this.portfolioName;

        public FinlPortfolio(String name) {
                this.setPortfolioName(name);
                orders = new ArrayList<FinlOrder>();            //list of all orders
                holdings = new ArrayList<Holding>();            //list of all positions
        }

        public void buy(FinlOrder order) {
                if(order.getExecute() == true) { //order has been executed
                        if(order.stock.valid)
                                System.err.println(order.stock.symbol.toUpperCase()
                                                + " Order already executed! Nothing Done.\n");
                        return;                                 //do nothing
                }
                order.setType("buy");
                order.stock.refresh();       //refresh stock information
                order.sharePrice = order.stock.finlQuote.ask.doubleValue();
                order.date = new Date();
                order.setExecute(true);
                orders.add(order);
                new FinlPortfolio.Holding(order);
        }

        public void sell(FinlOrder order) {
```

```
            if(order.getExecute() == true) { //order has been executed
                    if(order.stock.valid)
                            System.err.println(order.stock.symbol.toUpperCase()
                                            + "Order already executed! Nothing Done.\n");
                    return;                                 //do nothing
            }
            order.setType("sell");
            order.stock.refresh();
            order.sharePrice = order.stock.finlQuote.bid.doubleValue();
            order.date = new Date();
            order.setExecute(true);
            orders.add(order);
            new FinlPortfolio.Holding(order);
    }


    private void setPortfolioName(String name) {
            this.portfolioName = name;
            this.csvName = this.portfolioName;
    }

    public void updateCompositions() {
            for(int i = 0; i < holdings.size(); i++) {
                    Holding listStock = holdings.get(i);
                    listStock.percentOfPortfolio
                            = Math.abs(listStock.positionValue/accountValue);
            }
            this.updatePNL();
    }

    public FinlPortfolio switchWith(String csvName) {
            this.csvExport();
            FinlPortfolio test2 = new FinlPortfolio(csvName);
            test2.csvPortfolioBuilder();
            return test2;
    }

    public void updatePNL() {
            for(int i = 0; i < this.holdings.size(); i++) {
                    Holding listStock = this.holdings.get(i);
                    listStock.stock.refresh();              //refresh stock info
                    double currentCost =
                   listStock.stock.finlQuote.price.doubleValue()*listStock.positionShares;
                    double executionCost = listStock.avgPrice * listStock.positionShares;
                    listStock.pnl = currentCost - executionCost;
                    listStock.positionValue = currentCost;
            }
    }

    public void printOrders() {
            for(int i = 0; i < orders.size(); i++) {
                    orders.get(i).printOrder();
            }
    }

    public void printHoldings() {
            this.updatePNL();
            System.out.println("\n\nPortfolio Name:\t" + this.portfolioName
                            + "\n _____________________________________________________");
            System.out.format("|Account Value\t|\tPositions\t|\tTrades\t|\n|  $"
                            + "%.2f" + "\t|\t"
                            + this.holdings.size() + "\t\t|\t" + this.orders.size() +
                            "\t|\n -------------------------------------------------------"
                            + "\n\n\n---------\nHoldings:\n---------\n",
                            this.accountValue);

            for(int i = 0; i < holdings.size(); i++) {
                    Holding listStock = holdings.get(i);
                    System.out.println(listStock.stock.companyName
                                    + "\n_____________________________");
                    System.out.println("Symbol:\t\t" + listStock.stock.symbol);
                    System.out.println("Total Shares:\t" + listStock.positionShares);
                    System.out.format("Total Value:\t$%.2f\n", listStock.positionValue);
                    System.out.format("Average Price:\t$%.2f\n", listStock.avgPrice);
                    System.out.format("Position P&L:\t$%.2f\n", listStock.pnl);
                    System.out.format("Weight:\t\t%.2f%%\n", listStock.percentOfPortfolio*100);
                    System.out.println("Last Trade:\t" + listStock.lastOrder.toString()
```

```
                        + "\n_____________________________\n\n");
                }
                //System.out.flush();
        }

        public void csvPortfolioBuilder() {
                csvPortfolioBuilder(this.portfolioName);
        }

        public void csvPortfolioBuilder(String name) {
                this.csvName = name;
                this.accountValue = 0.0;
                String csvHoldings = name + "_holdings.csv";
                String csvOrders = name + "_orders.csv";

                BufferedReader holdingReader = null;
                BufferedReader orderReader = null;

                try {
                        holdingReader = new BufferedReader(new FileReader(csvHoldings));
                        orderReader = new BufferedReader(new FileReader(csvOrders));


                        holdingImporter(this, holdingReader);
                        orderImporter(this, orderReader);
                } catch (FileNotFoundException e) {
                        System.err.println(".csv File Does Not Exist!");
                } catch (IOException e) {
                        System.err.println(".csv File Does Not Exist!");
                } catch (NumberFormatException e) {
                        System.err.println("Date Exception in .csv Importing");
                } catch (ParseException e) {
                        System.err.println("Date Exception in .csv Importing");
                } finally {
                        if (holdingReader != null) {
                                try {
                                        holdingReader.close();
                                } catch (IOException e) {
                                        System.err.println("IO Exception in CSV Importing");
                                }
                        }
                        if (orderReader != null) {
                                try {
                                        orderReader.close();
                                } catch (IOException e) {
                                        System.err.println("IO Exception in CSV Importing");
                                }
                        }
                }
        }

        @SuppressWarnings("deprecation")
        private BufferedReader holdingImporter(FinlPortfolio importedPortfolio,
                        BufferedReader holdingReader) throws NumberFormatException, IOException, ParseException {

                int lineNum = 1;
                String line = "";
                String cvsSplitBy = ",";

                //build Portfolio: Holdings
                while ((line = holdingReader.readLine()) != null) {

                        if(!(lineNum++ < 5)) {                //if the line is a header line, do this

                                //create new holding object to put into Portfolio
                                Holding holdingObject = new Holding();
                                String[] arrayHolding = line.split(cvsSplitBy);
                        // use comma as separator

                                String weightString = arrayHolding[5];
                        //we multiplied by 100 to get to %, so we divide to get regular double for consistency
                        double weight =
                       Double.parseDouble(weightString.substring(0, weightString.length()-1))/100;
                        holdingObject.stock = new FinlStock(arrayHolding[0]);               //stock
                        holdingObject.positionShares = Integer.parseInt(arrayHolding[1]);    //shares
                        holdingObject.positionValue = dollarToDouble(arrayHolding[2]);       //value
                        holdingObject.avgPrice = dollarToDouble(arrayHolding[3]);  //average price
```

```
                    holdingObject.pnl = dollarToDouble(arrayHolding[4]);                //p&l

                    //calculations above^
                    holdingObject.percentOfPortfolio = weight;                          //weight
                    holdingObject.lastOrder = new Date(arrayHolding[6]);
                    //this.accountValue += holdingObject.positionValue;
                    importedPortfolio.holdings.add(holdingObject);    //add the holding object to the holdings
                                                                      //list

                    }        //end holdings object
                    if(lineNum == 3) {
                             String[] arrayHolding = line.split(cvsSplitBy);
                    // use comma as separator
                             this.accountValue = dollarToDouble(arrayHolding[1]);
                    }
          }         //end holdings lister
          return holdingReader;
}

@SuppressWarnings("deprecation")
private BufferedReader orderImporter(FinlPortfolio importedPortfolio,
                    BufferedReader orderReader) throws NumberFormatException, IOException, ParseException {

          int lineNum = 1;
          String line = "";
          String cvsSplitBy = ",";

          while ((line = orderReader.readLine()) != null) {
                    if(!(lineNum++ < 2)) {                //if the line is a header line, do this

                             //create new order Object to put into portfolio
                             FinlOrder orderObject = new FinlOrder();

          String[] arrayOrder = line.split(cvsSplitBy);
          orderObject.stock = new FinlStock(arrayOrder[0]);                  //stock ticker
          orderObject.setType(arrayOrder[1]);                                //order type
          orderObject.size = Integer.parseInt(arrayOrder[2]);                //order size
          orderObject.sharePrice = dollarToDouble(arrayOrder[3]);            //execution price
          orderObject.date = new Date(arrayOrder[4]);
          orderObject.setExecute(true);
          importedPortfolio.orders.add(orderObject);
                    }
          }
          return orderReader;
}

public double dollarToDouble(String s){
          String dollarRemoved = s.substring(1);
          double dollarRemovedDouble = Double.parseDouble(dollarRemoved);
          return dollarRemovedDouble;
}

public void csvExport(){
          try {
                    this.updatePNL();
                    String fileName = this.csvName;
                    //System.out.println(fileName);
                    csvOrdersExport(fileName);
                    csvHoldingsExport(fileName);

          } catch (IOException e) {
                    System.err.println("\n\nSomething is wrong with the file"
                                        + "export\n\n");
                    e.printStackTrace();
          }
}

private void csvHoldingsExport(String fileName) throws IOException {
          fileName += "_holdings.csv";

          FileWriter writer = new FileWriter(fileName);

          writer.append("Portfolio Name: "); writer.append(",");
          writer.append(this.portfolioName); writer.append("\n");
          writer.append("Account Value: "); writer.append(",");
          writer.append("$" + Double.toString(this.accountValue));
          writer.append("\n\n");
```

```
            writer.append("Stock Name"); writer.append(",");
            writer.append("Total Shares"); writer.append(",");
            writer.append("Total Value"); writer.append(",");
            writer.append("Average Price"); writer.append(",");
            writer.append("Position P & L"); writer.append(",");
            writer.append("Percent of Portfolio"); writer.append(",");
            writer.append("Last Execution Date"); writer.append("\n");

            for(int i = 0; i < holdings.size(); i++) {
                    Holding listHolding = holdings.get(i);
                    writer.append(listHolding.stock.symbol);
                    writer.append(",");
                    writer.append(Integer.toString(listHolding.positionShares));
                    writer.append(",");
                    writer.append("$" + Double.toString(listHolding.positionValue));
                    writer.append(",");
                    writer.append("$" + Double.toString(listHolding.avgPrice));
                    writer.append(",");
                    writer.append("$" + Double.toString(listHolding.pnl));
                    writer.append(",");
                    writer.append(Double.toString((listHolding.percentOfPortfolio)*100) + "%");
                    writer.append(",");
                    writer.append(listHolding.lastOrder.toString());
                    writer.append("\n");
            }
            writer.flush();
            writer.close();
    }

    private void csvOrdersExport(String fileName) throws IOException {
            fileName += "_orders.csv";

            FileWriter writer = new FileWriter(fileName);

            writer.append("Stock Name"); writer.append(",");
            writer.append("Order Type"); writer.append(",");
            writer.append("Order Size"); writer.append(",");
            writer.append("Execution Price"); writer.append(",");
            writer.append("Execution Date");
            writer.append("\n");

            for(int i = 0; i < orders.size(); i++) {
                    FinlOrder listOrder = orders.get(i);
                    writer.append(listOrder.stock.symbol);
                    writer.append(",");
                    writer.append(listOrder.getType());
                    writer.append(",");
                    writer.append(Integer.toString(listOrder.size));
                    writer.append(",");
                    writer.append("$" + Double.toString(listOrder.sharePrice));
                    writer.append(",");
                    writer.append(listOrder.date.toString());
                    writer.append("\n");
            }
            writer.flush();
            writer.close();
    }

    class Holding {

            public int positionShares;
            public double positionValue;
            public double percentOfPortfolio;
            public double avgPrice;
            public double pnl;
            public Date lastOrder;
            public FinlStock stock;

            private Holding(FinlOrder order) {
                    Holding position = checkHoldings(order);
                    if(position == null)         //if the portfolio doesnt contain the stock being ordered:
                            generateNewHolding(order);
                    else {              //if the portfolio contains the stock being ordered:
                            position.addToHolding(order);
                    }
                    updateCompositions();
```

```
                }

                private Holding() { //used when importing from the csv

                }

                private void addToHolding(FinlOrder order) {
                        int tempSize = Math.abs(this.positionShares);
                        double tempAvg = this.avgPrice;


                        if(order.getType().equals("sell"))
                                this.positionShares -= order.size;
                        else if (order.getType().equals("buy"))
                                this.positionShares = this.positionShares + order.size;
                        else System.err.println("Order Type Not Set");

                        double weightedAverage = (((tempAvg * tempSize)
                                        + (order.sharePrice * order.size)))/(tempSize + order.size);
                        this.avgPrice = weightedAverage;                //and new order's price
                        this.pnl = (this.positionShares * order.sharePrice) //difference between value of position
                        - (this.positionShares * this.avgPrice); //now and what we paid for it
                        this.lastOrder = order.date;
                        this.stock = order.stock;
                        this.positionValue = this.avgPrice * this.positionShares;

                        if(order.getType().equals("buy"))
                                accountValue += (order.size * order.sharePrice);
                        else if (order.getType().equals("sell"))
                                accountValue += (order.size * order.sharePrice);

                        this.percentOfPortfolio = Math.abs(this.positionValue/accountValue);

                        //no need to add to the list, order already exists and we are just modifying
                }       //end addToHoldings()

                private void generateNewHolding(FinlOrder order) {
                        if(order.getType().equals("sell"))
                                this.positionShares -= order.size;
                        else if (order.getType().equals("buy"))
                                this.positionShares += order.size;
                        else System.err.println("Order Type Not Set");

                        this.avgPrice = order.sharePrice;
                        this.pnl = 0;
                        this.lastOrder = order.date;
                        this.stock = order.stock;
                        this.positionValue = this.avgPrice * this.positionShares;

                        if(order.getType().equals("buy"))
                                accountValue += (order.size * order.sharePrice);
                        else if (order.getType().equals("sell"))
                                accountValue += (order.size * order.sharePrice);

                        this.percentOfPortfolio = Math.abs(this.positionValue/accountValue);
                        holdings.add(this); //add this new holding to the list
                }       //end generateNewHolding()

                //checks if the stock is in the portfolio
                private Holding checkHoldings(FinlOrder order) {
                        for(int i = 0; i < holdings.size(); i++) {
                                Holding listStock = holdings.get(i);
                                if(listStock.stock.symbol.equals(order.stock.symbol)) {
                                        return listStock;  //stock is in the portfolio
                                }
                        }
                        return null;       //stock is not in the portfolio
                }
        }       //end Holdings class
}
```

FinlStock.java

```
package bin;


import java.io.IOException;
import java.io.PrintStream;
import java.math.BigDecimal;
import java.util.Calendar;

import yahoofinance.YahooFinance;
import yahoofinance.quotes.stock.StockDividend;
import yahoofinance.quotes.stock.StockQuote;
import yahoofinance.quotes.stock.StockStats;

public class FinlStock {

	public String symbol;
	public String companyName;
	public boolean valid = true; //valid stock

	private yahoofinance.Stock stock;
	public FinlQuote finlQuote;
	public FinlFundamentals finlFundamentals;
	public FinlDividend finlDividend;

	//////////////////////////////////////////
	//////////*FinlStock Constructor*///////////
	///////////Gets all sub-objects/////////////
	public FinlStock(String ticker) {
		setStock(ticker);
		if(this.stock.getName().equals("N/A")) {
			System.err.println(ticker + " Not Found!");
			this.valid = false;
		}

		this.finlQuote                                          = new FinlStock.FinlQuote(stock);
		this.finlFundamentals                           = new FinlStock.FinlFundamentals(stock);
		this.finlDividend                                       = new FinlStock.FinlDividend(stock);
	}

	private void setStock(String ticker) {
		try{
			this.symbol = ticker;

			PrintStream defaultOutputStream = System.out;	//save output stream
			System.setOut(null);                            //redirect console output
			this.stock = YahooFinance.get(this.symbol);	//assign stock w/ suppressed output
			this.companyName = this.stock.getName();
			System.setOut(defaultOutputStream);             //reset output stream to default

		} catch (IOException IOE) {
			IOE.printStackTrace();
		}
	}

	public FinlStock refresh() {
		return new FinlStock(this.symbol);
	}

	public void printStock() {
		this.stock.print();
	}

	/* Ocaml request parser */
	public double getRequest(String request) {
		if(!this.valid) {
```

```
                    System.err.println("Stock Is Not Valid!");
                    return 0;
          }
          double result;
          try {
                    result = this.finlFundamentals.fundamentalCheck(request);
                    if(result != 0)
                              return result;
                    result = this.finlQuote.quoteCheck(request);
                    if(result != 0)
                              return result;
                    result = this.finlDividend.dividendCheck(request);
                    if(result != 0)
                              return result;
          } catch (NullPointerException NTE) {
                    System.err.println("Error!");
                    NTE.printStackTrace();
                    return 0.0;
          }
          return result;
}


//////////////////////////////////////////
/////////*Fundamentals SubClass*///////////
//////////////////////////////////////////
public class FinlFundamentals {

          /*Fundamentals Object*/
          private StockStats fundamentals;

          /*Fundamentals*/
          public BigDecimal   bookValuePerShare;
          public BigDecimal   ebitda;
          public BigDecimal   eps;
          public BigDecimal   marketCap;
          public BigDecimal   pe;
          public BigDecimal   peg;
          public BigDecimal   priceBook;
          public BigDecimal   priceSales;
          public BigDecimal   revenue;
          public BigDecimal   roe;
          public BigDecimal   sharesFloat;
          public BigDecimal   sharesOutstanding;

          /*Estimates*/
          public BigDecimal   epsEstimateCurrentYear;
          public BigDecimal   epsEstimateNextQuarter;
          public BigDecimal   epsEstimateNextYear;
          public BigDecimal   oneYearTargetPrice;

          //////////////////////////////////////////
          //////////////*Constructors*/////////////////
          //////////////////////////////////////////

          public FinlFundamentals(yahoofinance.Stock stock) {
                    fundamentals = stock.getStats();
                    this.populateStatistics();
                    this.populateEstimates();
          }         //end constructor FinlQuote(String symbol)

          private void populateStatistics() {
                    bookValuePerShare   = this.fundamentals.getBookValuePerShare();
                    ebitda              = this.fundamentals.getEBITDA();
                    eps                 = this.fundamentals.getEps();
```

```
                marketCap          = this.fundamentals.getMarketCap();
                pe                 = this.fundamentals.getPe();
                peg                = this.fundamentals.getPeg();
                priceBook          = this.fundamentals.getPriceBook();
                priceSales         = this.fundamentals.getPriceSales();
                revenue            = this.fundamentals.getRevenue();
                roe                = this.fundamentals.getROE();
                sharesFloat        = BigDecimal.valueOf(this.fundamentals.getSharesFloat());
                sharesOutstanding  = BigDecimal.valueOf(this.fundamentals.getSharesOutstanding());

        }       //end populateStatistics()

        private void populateEstimates() {
                epsEstimateCurrentYear       = this.fundamentals.getEpsEstimateCurrentYear();
                epsEstimateNextQuarter       = this.fundamentals.getEpsEstimateNextQuarter();
                epsEstimateNextYear= this.fundamentals.getEpsEstimateNextYear();
                oneYearTargetPrice           = this.fundamentals.getOneYearTargetPrice();
        }       //end populateEstimates()

        private double fundamentalCheck(String request) {
                if(request.equalsIgnoreCase("bookValuePerShare"))
                        return this.bookValuePerShare.doubleValue();
                else if(request.equalsIgnoreCase("ebitda"))
                        return this.ebitda.doubleValue();
                else if(request.equalsIgnoreCase("eps"))
                        return this.eps.doubleValue();
                else if(request.equalsIgnoreCase("marketCap"))
                        return this.marketCap.doubleValue();
                else if(request.equalsIgnoreCase("pe"))
                        return this.pe.doubleValue();
                else if(request.equalsIgnoreCase("peg"))
                        return this.peg.doubleValue();
                else if(request.equalsIgnoreCase("priceBook"))
                        return this.priceBook.doubleValue();
                else if(request.equalsIgnoreCase("priceSales"))
                        return this.priceSales.doubleValue();
                else if(request.equalsIgnoreCase("revenue"))
                        return this.revenue.doubleValue();
                else if(request.equalsIgnoreCase("roe"))
                        return this.roe.doubleValue();
                else if(request.equalsIgnoreCase("sharesFloat"))
                        return this.sharesFloat.doubleValue();
                else if(request.equalsIgnoreCase("sharesOutstanding"))
                        return this.sharesOutstanding.doubleValue();
                else if(request.equalsIgnoreCase("epsEstimateCurrentYear"))
                        return this.epsEstimateCurrentYear.doubleValue();
                else if(request.equalsIgnoreCase("epsEstimateNextQuarter"))
                        return this.epsEstimateNextQuarter.doubleValue();
                else if(request.equalsIgnoreCase("epsEstimateNextYear"))
                        return this.epsEstimateNextYear.doubleValue();
                else if(request.equalsIgnoreCase("oneYearTargetPrice"))
                        return this.oneYearTargetPrice.doubleValue();
                else return 0.0;
        }       //end fundamentalsCheck()
}

//////////////////////////////////////////
////////////*Quotes SubClass*/////////////
//////////////////////////////////////////
public class FinlQuote {

        /*Quote & Prices*/
        private StockQuote  quote;
        /*////Prices////*/
        public BigDecimal   price;
```

```
public BigDecimal   priceOpen;
public BigDecimal   pricePrevClose;
public BigDecimal   priceMA200;
public BigDecimal   priceMA50;
public BigDecimal   priceDayHigh;
public BigDecimal   priceDayLow;
public BigDecimal   bid;
public BigDecimal   ask;
public BigDecimal   avgVolume;
/*Price Movement*/
public BigDecimal   change;             //change from current price to previous close
public BigDecimal   changePercent;      //change from current price to previous close, in percent
public BigDecimal   changeFromMA200;
public BigDecimal   changeFromMA50;
public BigDecimal   changeFromYearHigh;
public BigDecimal   changeFromYearLow;


//////////////////////////////////////////
//////////////*Constructors*/////////////////
//////////////////////////////////////////

public FinlQuote(yahoofinance.Stock stock) {
          quote = stock.getQuote();
          this.populatePrice();
          this.populateMovement();
}         //end constructor FinlQuote(String symbol)

//////////////////////////////////////////
//////////////*Populate Methods*/////////////
//////////////////////////////////////////
private void populatePrice() {
          price                         = this.quote.getPrice();
          priceOpen                     = this.quote.getOpen();
          pricePrevClose                = this.quote.getPreviousClose();
          priceMA200                    = this.quote.getPriceAvg200();
          priceMA50                     = this.quote.getPriceAvg50();
          priceDayHigh                  = this.quote.getDayHigh();
          priceDayLow                   = this.quote.getDayLow();
          bid                           = this.quote.getBid();
          ask                           = this.quote.getAsk();
          avgVolume                     = BigDecimal.valueOf(this.quote.getAvgVolume());
}         //end populatePrice()

private void populateMovement() {
          change                        = this.quote.getChange();
          changePercent                 = this.quote.getChangeInPercent();
          changeFromMA200              = this.quote.getChangeFromAvg200();
          changeFromMA50                = this.quote.getChangeFromAvg50();
          changeFromYearHigh  = this.quote.getChangeFromYearHigh();
          changeFromYearLow           = this.quote.getChangeFromYearLow();
}         //end populateMovement()

public double quoteCheck(String request) {
          if(request.equalsIgnoreCase("price"))
                    return this.price.doubleValue();
          else if(request.equalsIgnoreCase("priceOpen"))
                    return this.priceOpen.doubleValue();
          else if(request.equalsIgnoreCase("pricePrevClose"))
                    return this.pricePrevClose.doubleValue();
          else if(request.equalsIgnoreCase("priceMA200"))
                    return this.priceMA200.doubleValue();
          else if(request.equalsIgnoreCase("priceMA50"))
                    return this.priceMA50.doubleValue();
          else if(request.equalsIgnoreCase("priceDayHigh"))
```

```
                                        return this.priceDayHigh.doubleValue();
                                else if(request.equalsIgnoreCase("priceDayLow"))
                                        return this.priceDayLow.doubleValue();
                                else if(request.equalsIgnoreCase("bid"))
                                        return this.bid.doubleValue();
                                else if(request.equalsIgnoreCase("avgVolume"))
                                        return this.avgVolume.doubleValue();
                                else if(request.equalsIgnoreCase("change"))
                                        return this.change.doubleValue();
                                else if(request.equalsIgnoreCase("changePercent"))
                                        return this.changePercent.doubleValue();
                                else if(request.equalsIgnoreCase("changeFromMA200"))
                                        return this.changeFromMA200.doubleValue();
                                else if(request.equalsIgnoreCase("changeFromMA50"))
                                        return this.changeFromMA50.doubleValue();
                                else if(request.equalsIgnoreCase("changeFromYearHigh"))
                                        return this.changeFromYearHigh.doubleValue();
                                else if(request.equalsIgnoreCase("changeFromYearLow"))
                                        return this.changeFromYearLow.doubleValue();
                                else return 0.0;
                        }       //end quoteCheck()
                }       //end FinlQuote subclass

                ////////////////////////////////////////////
                ////////////*Dividend SubClass*/////////////
                ////////////////////////////////////////////
                public class FinlDividend {

                        /*Dividend Object*/
                        private StockDividend dividend;

                        /*Dividend Data*/
                        public BigDecimal   annualYield;
                        public BigDecimal   annualYieldPercent;
                        public Calendar     exDivDate;
                        public Calendar     payDate;
                        public String               exDivDate_String;
                        public String               payDate_String;

                        /*Pass a Stock Object*/
                        public FinlDividend(yahoofinance.Stock stock) {
                                dividend                                = stock.getDividend();

                                annualYield                             = this.dividend.getAnnualYield();
                                annualYieldPercent          = this.dividend.getAnnualYieldPercent();
                                exDivDate                               = this.dividend.getExDate();
                                payDate                                 = this.dividend.getPayDate();
                        }       //end constructor FinlDividend(Stock stock)

                        public double dividendCheck(String request) {
                                if(request.equalsIgnoreCase("annualYield"))
                                        return this.annualYield.doubleValue();
                                else if(request.equalsIgnoreCase("annualYieldPercent"))
                                        return this.annualYieldPercent.doubleValue();
//                              else if(request.equalsIgnoreCase("payDate"))
//                                      return this.payDate.toString();
//                              else if(request.equalsIgnoreCase("exDivDate"))
//                                      return this.exDivDate.toString();
                                else return 0.0;
                        }       //end dividendCheck()
                }       //end FinlDividend subclass
}       //End FinlStock.java
```

9.10 Test Suite

access_test.finl

```
stock stk;
stk << @TSLA;
float s;
s << stk[price];
s << @AAPL[price];
print "yay!";
```

add_assign.finl

```
function int main(int x) {
      return x +<< 1;
};

print main(1);
string y;
y << "string";
y +<< "string";
print y;
```

and_test.finl

```
print 1 and 1;
print 1.0 and 0;
print 1 and 0.1;
```

arg_length_test.finl

```
function int main(){
      return 1;
};

main(1);
```

arith_test.finl

```
function int main() {
      int i;
      i << 3 + 3 * 8 / 6;
      print i + 3;
```

```
      print 3;
      return 0;
};

main();
```

assign_function_test.finl

```
function string main(string x){
      return x + x;
};

string x;
x << main("string");
print x;
```

assign_type_mismatch_test.finl

```
int x;
#x << "string";
#x +<< "string";
#x -<< "string";
#x *<< "string";
x /<< "string";
```

assign_void_test.finl

```
function void main() {
      print "main";
};

string x;
x << main();
```

bad_buy_sell_test.finl

```
int a;
a << 1;
buy a; # comment this out to test bad sells
sell a;
```

bad_formals_test.finl

```
function int main(int x, int x) {
      return 0;
};
```

bad_mod_test.finl

```
"s" % 1;
```

bad_order_test.finl

```
order o;
string s;
s << "23";
o << 10 of s;
```

bad_return_test.finl

```
function int main() {
      return 1;
};

function string mainn() {
      return 1;
};
```

bad_type_assign_test.finl

```
string x;
x << "string";
x -<< "string";
stock s;
s << @AAPL;
s *<< @TSLA;
```

binop_type_mismatch_test.finl

```
function int main() {
      int x;
      x << 1 + "string";
};
```

buy_sell_test.finl

```
buy 10 of @TSLA;
order o;
o << 101 of @AAPL;
sell o;
print "heyo!";
```

define_order_test.finl

```
order o1;
o1 << 10 of @AAPL;
stock s1;
s1 << @TSLA;
o1 << 100 of s1;
print "this should work";
```

define_stock_test.finl

```
stock apple;
apple << @AAPL;
print "success";
```

div_assign_test.finl

```
function int main(int x){
      return x /<< 2;
};

print main(4);
```

float_test.finl

```
function float main(float f) {
      return f;
};

print main(3.14);
```

hello_world_test.finl

```
function float main(float f) {
      return f;
};
```

```
print main(3.14);
```

if_decl_test.finl

```
int x;
x << 1;
1 < 2 ? {
      int x;
};
```

if_else_return_test.finl

```
function int ret() {
      1 > 2 ? {
            return 1;
      } ! {
            return 0;
      };
};

print ret();

function int ret2() {
      1 > 2 ? {
            return 1;
      };
      return 0;
};

print ret2();
```

if_else_test.finl

```
1 > 2 ? {
      print "1 is more than 2";
} ! {
      print "1 is less than 2";
};
```

if_return_test.finl

```
function int main() {
      1 > 2 ? {
```

```
            return 1;
      };
      return 0;
};

print main();
```

if_test.finl

```
int x;
x << 1;
x < 2 and x > 0.0 ? {
      print "yay";
};
```

int_float_op_test.finl

```
print 6 + 3.0;
print 6 - 3.0;
print 6 * 3.0;
print 6 / 3.0;
print 6 > 3.0;
print 6 < 3.0;
print 6 = 3.0;
print 6 >= 3.0;
print 6 <= 3.0;
```

main_test.finl

```
function int main(){
      return 0;
};

function int main(){
      return 0;
};
```

mod_test.finl

```
print 10 % 2;
print 7 % 2;
print 2 % 1.3;
#print 2.3 % 1;
```

```
print 2.0 % 1.3;
```

mult_assign_test.finl

```
int x;
x << 1;
x *<< 5;
print x;
```

multiple_params_test.finl

```
function int main(int x, string y) {
      print x;
      print y;
      return 0;
};

main(3, "string");
```

multiple_return_test.finl

```
function int main() {
      return 0;
      return 1;
};

main();
```

negative_statement_test.finl

```
-4;
```

negatives_test.finl

```
int x;
x << -1;
float y;
y << -1.4;
print x;
print y;
```

nested_if_test.finl

```
1 < 2 ? {
      print "yes";
      1 > 2.0 ? {
            print "no!!!";
      };
};
```

no_return_test.finl

```
function int main(){};

main();
```

not_test.finl

```
not (0) ? {
      print "dope";
};

not (1 = 0.0) ? {
      print "dopedope";
};
```

operator_test.finl

```
function int main() {

  print 1+1;
  print 3-1;
  print 2*1;
  print 6/3;
  return 0;
};

main();
```

or_test.finl

```
print 1 or 0;
print 0.0 or 0;
```

outside_return_test.finl

```
return 1;
```

overwrite_function_test.finl

```
function int m() {
      print 1;
      return 0;
};

function int m() {
      print 2;
      return 0;
};

m();
```

parameter_type_mismatch_test.finl

```
function int main(int x) {
      return x;
};

main("string");
```

pow_test.finl

```
print 3**0;
print 3**2;
print 3**2.0;
print 3.0**2;
print 3.0**2.0;
print_void_test.finl

function void main(){};
print main();
```

redeclare_variable_test.finl

```
int x;
string x;
```

reserved_test.finl

```
function int reserved(){
      return 0;
};
```

return_from_void_test.finl

```
function void main() {
      return "1\n";
};

print main();
```

return_void_test.finl

```
function void main() {
      print "hey!";
      return;
};

main();
print "spalding";
```

statement_test.finl

```
function int main() {
      3 + 4;
};
```

string_op_test.finl

```
print "string" + "string";
#print "string" - "string";
#print "string" * "string";
#print "string" / "string";
print "a_string" > "string";
print "a_string" < "string";
print "a_string" = "string";
print "string" = "string";
print "a_string" >= "string";
print "a_string" <= "string";
#print "string" % "string";
#print "string" ** "string";
```

sub_assign_test.finl

```
function int main(int x){
      return x -<< 1;
};

print main(1);
```

uninitialized_assignment_test.finl

```
#x << 3;
#x +<< 3;
#x -<< 3;
#x *<< 3;
x /<< 3;
```

uninitialized_call_test.finl

```
main();
```

uninitialized_variable_test.finl

```
int y;
y << 2;
y << z;
```

vdecl_test.finl

```
int x_1;
string w__;
print "success";
```

void_function_test.finl

```
function void main() {
      print "main";
};

main();
```

when_num_test.finl

```
when @TSLA[price] > 0 {};
```

```
when @TSLA[price] < 1000000.1 {};

print "hey";
```

when_test.finl

```
when @TSLA[priceMA200] > @TSLA[priceMA50] {
      print;
};
```

while_test.finl

```
int x;
x << 0;
while (x < 5) {
      print x;
      x +<< 1;
};
```