Uniform General Algorithmic (UNIGA)

Financial Trading Language

Project Report

Leon Wu (llw2107@columbia.edu)
Jiahua Ni (jn2173@columbia.edu)
Jian Pan (jp2472@columbia.edu)
Yang Sha (ys2280@columbia.edu)
Yu Song (ys2310@columbia.edu)

May 7, 2007

Columbia University in the City of New York

Table of Content

1. Introduction

1.1 Introduction

Today many financial firms have their own trading software tools to facilitate investors' investment process. These tools often differ from each other and usually take a long time to learn. They are also not easy to be customized and very expensive. Many institutional and individual investors have their own custom-designed investment strategies. They want to implement their investment strategies using some easy-to-use software with low cost. However, designing financial trading software from scratch requires professional knowledge and can be costly and time-consuming. The UNIGA Language provides an easy and efficient way for people to design their own investment plan. The language allows users to write a program that can automatically trade financial instruments including Stock, Options, Bonds, and Mutual Funds etc. using pre-defined trading strategy that defines trading rules such as set-price, sell-price, comparisons, quantity and other elements.

1.2 Overview of the Language

UNIGA is a high-level scripting language. Script programming languages enable programmers to specify trading operations intuitively. Although not as comprehensive as the more well known scripting languages such as Perl or Python, the built-in keywords make the language more intuitive and easy to use. The user is able to design trading software in the form of a program. The translator will then output a Java source file that can be edited and compiled into Java byte-code.

1.3 Goals

UNIGA Language is a language that enables the users to perform various kinds of trading operations, mainly consisted of selling and buying but with more easy-to-use powerful features, using an interactive coding process. Thus, again, UNIGA Language is meant to be easy to use, portable, powerful, versatile and flexible.

1.3.1 Ease-to-use

UNIGA Language is a clean, intuitive, and easy-to-learn language which allows users to create their own buying and selling strategies by writing a few lines of code. UNIGA uses a well-defined set of basic syntaxes similar to Perl or PHP and this makes UNIGA programmer-friendly.

1.3.2 Portability

Since Java is a platform-independent portable language thanks to its own interpreter, UNIGA Language is also portable language because it converts user-created-program into Java code. So, you can execute the program on any platform where Java Virtual Machine is installed.

1.3.3 Powerful

UNIGA language enables the users to perform various trading transactions, for example buying, selling, and comparing stock prices, with just few lines of code.

This powerfulness allows work to be more and efficient and productive as well as providing a clear outline of what was used to achieve the final result.

1.3.4 Versatile
Users can target almost every financial market all over the world from New York, Tokyo, London, and Paris to Sydney. Also, users can deal with several financial commodities such as stock, futures, securities, options and so on.

1.3.5 Flexible
UNIGA Language allows users to add their own functions or even import and other libraries in order to use new functions that the users need.

1.4 Basic Language Features

1.4.1 Statement
An UNIGA Language statement represents a complete instruction. Statements can contain reserved words, operators, and punctuation marks, and always end in a semicolon. Examples are shown in Sample Code section.

1.4.2 Data Types
We have defined our data type as follows:
Numeric: double (to represent date, time, price, trade volume)
Boolean: true/false Boolean values.

1.4.3 Reserved Words
The basic vocabulary of UNIGA Language consists of a set of pre-defined words, which we call reserved words. Reserved words each have a specific meaning or purpose. Mainly, this can be classified into two subsets.

1.4.3.1 Basic Reserved Words

close	Last traded price of a stock
date	Date of the close of a stock
open	First traded price of a stock
high	Highest traded price of a stock
low	Lowest traded price of a stock
market	Get market price of stock
volume	Number of shares or contracts traded in a stock

1.4.3.2 Control Reserved Words

while	Used for continuous execution of an action, stops when the preliminary condition disqualifies
for	For loop
if-else	Conditional execution clause

1.4.4 Expression and Operators
In UNIGA Language, an expression is any combination of reserved words and operators that represent a value.

1.4.4.1 Mathematical Operators

Math Operator	Meaning
+	Addition
-	Minus
*	Multiplication
/	Division

1.4.4.2 Relational Operators

Relational Operator	Meaning
<	Less than
>	Greater than
=	Assign
==	Equal to
~	Not Equal

1.4.4.3 Logical Operators

Relational Operator	Meaning
&	Logical AND
|	Logical OR

1.4.5 Punctuation Marks
There are a number of punctuation marks to establish statements, define parameters, delimit words, and establish order of precedence.

Symbol	Name	Description
;	Semicolon	Ends a statement
()	Parentheses	Group values and forces them to be calculated first
,	Comma	Separates each parameter or input
:	Colon	Used in declaration statements to begin the list of inputs or variables
“ ”	Quotation Marks	Defines a text string
[]	Square Brackets	Used to specify dates
{}	Curly Brackets	Used as modifier, to reference a value from a previous bar

1.4.6 Built in functions
In order to separate our language’s program from data, we’ve defined the following built in functions for our language:

print/println: Console output functions, function print prints variable values, characters and strings.

error(): Standard error handler

average(String stockID **):** Average stock price of current date

sum(): Total asset value of a portfolio.

holdings(): List current detailed stock holdings of a portfolio

pl(): Profit and loss of a portfolio.

1.4.7 User defined functions

In order to support user defined function, we defined the following means to define and declare a function:

```
return_type func_name(para1,para2);
```

1.5 Sample Code

Sample code to perform operation in UNIGA Language:

1.5.1 buy

```
buy Symbol Shares Stop Limit;
buy “MSFT” 1000 23.5 28.7;
```

1.5.2 sell

```
sell Symbol Shares Stop Limit;
sell “MSFT” 1000 23.5 28.7;
```

1.5.3 if-else

```
if close > high “MSFT” {2};
else buy “MSFT” 500 0 0;
```

1.5.4 while

```
while(i<5){
   println i;
   i=i+1;
}
```

1.6. Optional Features

Because traders and investors tender to make mistakes when typing digits, it would be nice if there are functionalities that automatically detect such misbehaviors and return error messages according to the mistakes users made. This idea is hard to implement as the compiler or interpreter is not easy to determine the range of user input.

2. Language Tutorial

2.1 “Hello World”

HelloWorld.uniga:

```
main(){
  print “Hello World”;
}
```

The above program will print the “Hello World” on the screen. There is another function “*println()*” also take charge of printing a string or expression.

Since UNIGA will not generate an executable file, the only way to execute the above program is:

```
java Main HelloWorld.uniga
```

The suffix of any executable UNIGA file must be “.uniga”.

2.2 Stock orders

Two built-in functions are responsible for buying/selling stocks: a) buy, and b) sell. The stop price and limit price for the specific stock should be provided, however, if the customer does not need to set the stop or limit price in advance, the stop price or limit price should be 0.

The format of buy/sell is:

buy symbol number-of-shares stop-price limit-price
and
sell symbol number-of-shares stop-price limit-price

StockOrders.uniga

```
main(){
   buy “MSFT” 1000 0 0;
   sell “INTC” 550 0 18;
}
```

The above program executes the following orders: 1) buy 1000 shares of MSFT (the symbol of Microsoft), without setting stop price and limit price, and 2) sell 550 shares of INTC (the symbol of Intel), with $18 as the limit price.

These 2 orders will be recorded into the file /data/ORDERS.xml, and after the transaction is executed, customer’s portfolio, which is recorded by /data/PROTFOLIO.xml will be updated.

2.3 Check the prices

The historical prices can be displayed by using the keywords: “market”, “high”, “low”, “open”, “close”, “volume”, and “average”.

DailyHighAndLow.uniga

```
main(){
  println high “MSFT” {2};
  println low “MSFT” {2};
}
```

The *DailyHighAndLow.uniga* prints out the high and low prices of MSFT (symbol of Microsoft) 2 days ago.

And if the customer needs the current price, he can use the function “market”.

```
double currentPrice=market “MSFT”;
```

You can also use it in the if or for statement.

```
while(market “MSFT” > 28.72){
        buy “MSFT” 1000 0 0;}
```

The above program will keep checking the price for Microsoft, and whenever the price is larger than 28.72, an order is filed.

The function “average” is a little bit different. It outputs the average price between today’s high price and low price, within the effective trading hours. In a valid order, the stop and limit prices should be between daily high and low, the average function is very helpful for users to identify the price range.

The format of this function is:

```
double avgPrice=average(“MSFT”);
```

The return type is double.

2.4 Check the portfolio

There are several built-in functions for checking the portfolio. They are “pl”, and “sum”. “pl()” will return today’s profit loss, while “sum()” will return the amount of your total asset. The way to use these 2 functions are:

```
double profitLoss=pl();
```

and

```
double assetSum=sum();
```

2.5 User defined function

UNIGA supports you to define your own function. A complete function consists of declaration and body. For example, to create a function buying Google stock with market price, you can write it like this:

```
double buyGoogle(double share, double stop, double limit){
   if share<1 then {println “incorrect order”; return 0;}
   buy “GOOG” share stop limit;
   return 1;
}
```

And inside the “main()”, you can call this function:

```
main(){
  while(Market “GOOG” > 412.0){
     if buyGoogle(1000, 0, 415.2) then println “filed”;
  }
```

2.6 A sample program

Below is a comprehensive program that executes a certain trading strategy mocking the real trading situations, and at the same time testing all aspects of our language construct.

Strategy.uniga source file

```
main()
{       double s;
        s = sum();
        print "Sum of portfolio: ";
        print s;
        println "----------------------";

        double five_high;
        five_high=high "MSFT" {5} + high "MSFT" {4} + high "MSFT" {3} + high
"MSFT" {2} + high "MSFT" {1};
        five_high=five_high/5;

        double five_low;
        five_low=low "MSFT" {5} + low "MSFT" {4} + low "MSFT" {3} + low "MSFT"
{2} + low "MSFT" {1};
        five_low=five_low/5;

        double five_average;
        five_average=five_high+five_low;
        five_average=five_average/2;

        double tenfive_high=high "MSFT" {10} + high "MSFT" {9} + high "MSFT" {8}
+ high "MSFT" {7} + high "MSFT" {6};
        tenfive_high=tenfive_high/5;
        double tenfive_low=low "MSFT" {10} + low "MSFT" {9} + low "MSFT" {8} +
low "MSFT" {7} + low "MSFT" {6};
        tenfive_low=tenfive_low/5;
        double ten_average=tenfive_high+tenfive_low;
        ten_average=ten_average/10;
        ten_average=ten_average+five_average;
        ten_average=ten_average/2;

        double current_price=market "MSFT";
        double limit,stop;
        double buyshare=0,sellshare=0;

        while(current_price > five_average & buyshare < 1900){
                if open "MSFT" {0}>five_average then {
                        limit= open "MSFT" {0};
                        buy "MSFT" 500 five_low limit;

                }
                else {
                        stop=open "MSFT" {0};
                        buy "MSFT" 500 stop five_high;
                }
                buyshare=buyshare+500;

                current_price=market "MSFT";
        }
```

Strategy.uniga source

```
        if current_price < five_average then {
                if current_price < ten_average then {
                        sell "MSFT" 1000 current_price five_high;
                }
                                    else {
                        sell "MSFT" 500 ten_average five_high;
                }
        }

        double delta_average,delta;
        if open "MSFT" {0} - close "MSFT" {1} > 0 then{
                delta=open "MSFT" {0} - close "MSFT" {1};
        }
        else {
                delta=close "MSFT" {1} - open "MSFT" {0};
        }



        if ten_average > five_average then {
                delta_average=ten_average-five_average;
                stop=open "MSFT" {0} - delta_average;
                limit= market "MSFT" + delta;
                sell "MSFT" 1000 stop limit;
        }
        else {
                delta_average=five_average-ten_average;
                stop=open "MSFT" {0} - delta;
                limit= market "MSFT" + delta_average;
                buy "MSFT" 1000 stop limit;
        }


        s = sum();
        print "Sum of portfolio: ";
        print s;
        println " ";
        println "---------------------";

        double r;
        r = pl();
        print "Profit and loss: ";
        print r;
        println " ";
        println "---------------------";


}
```

Output results

```
Sum of portfolio: 552555.0----------------------
------------------------------
Date: 4/25/2007
Order Type: buy
Stock ID: MSFT
Amount: 500.0
Stop Price: 27.6
Limit Price: 28.55
Filled Status: 1
Filled Price: 28.55
------------------------------

------------------------------
Date: 4/25/2007
Order Type: buy
Stock ID: MSFT
Amount: 500.0
Stop Price: 27.6
Limit Price: 28.55
Filled Status: 1
Filled Price: 28.55
------------------------------

------------------------------
Date: 4/25/2007
Order Type: buy
Stock ID: MSFT
Amount: 500.0
Stop Price: 27.6
Limit Price: 28.55
Filled Status: 1
Filled Price: 28.55
------------------------------

------------------------------
Date: 4/25/2007
Order Type: buy
Stock ID: MSFT
Amount: 500.0
Stop Price: 27.6
Limit Price: 28.55
Filled Status: 1
Filled Price: 28.55
------------------------------

------------------------------
Date: 4/25/2007
Order Type: buy
Stock ID: MSFT
Amount: 1000.0
Stop Price: 26.570000000000004
Limit Price: 40.0961
Filled Status: 0
Filled Price: 0.0
------------------------------

Sum of portfolio: 552915.0
----------------------
Profit and loss: 407550.0
----------------------
```

3. Language Reference Manual

3.1 Lexicon convention

3.1.1 Identifiers
Identifiers consist of a sequence of one or more uppercase or lowercase alphabetic characters, (digits 0 to 9). The first character of an identifier should be a letter and cannot be a number. Identifiers are case sensitive, upper case and lower case letters are treated differently. Keywords are not identifiers.

3.1.2 Keywords
The following identifiers are reserved as keywords:

boolean double date void
if else for break
while function return market
open close volume high
low sell buy

3.1.3 Numbers
A number consists of digits, decimal point ".” All numbers in our language are implemented as double precision floating point numbers by default.

3.1.4 String literals
Strings are sequences of zero or more characters. String literals are character strings surrounded by quotation marks (" "). String literals can include any valid character, including white-space characters and character escape sequences.

3.1.5 Operators
An operator is a token that specifies an operation on at least one operand, and yields some result (a value, designator, side effect, or some combination). Operands are expressions or constants (a form of expression).
Operators in UNIGA are:

+, -, *, /
>, <, = =, =, &, |

3.2 Data Types
We have defined our data type as follows:
Numeric: double (to represent date, time, price, trade volume)
True/False: Boolean values, there's no array for this data type.

3.3 Declarations

3.3.1 General type declaration
The general syntax of a declaration is as follows:

declaration:

type-specifier init-declarator-list(opt);
init-declarator-list:
declarator
init_declarator-list , declarator

Type specifiers are: double, boolean

3.4 Functions

3.4.1 Function Calls
A function call is a primary expression, usually a function identifier followed by parentheses, which is used to invoke a function. The parentheses contain a (possibly empty) comma-separated list of expressions that are the arguments to the function.

3.4.2 Functions Types
A function has the derived type "function returning type". The type can be any data type except array types or function types. If the function returns no value, its type is "function returning void ", sometimes called a void function. Functions can be introduced into a program in one of two ways:

1. A function definition can create a function designator, define its parameters and their type, define the type of its return value, and supply the body of the function.

2. A function declaration announces the properties of a function defined elsewhere.

3.4.3 Function Definitions
A function definition includes the code for the function. Function definitions can appear in any order, and in one source file or several, although a function cannot be split between files. Function definitions cannot be nested.
A function definition has the following syntax:

function-definition:
return-type declarator declaration-list(opt)
compound-statement
declaration-specifiers

The declaration-specifiers (type-qualifier, and type- specifier) can be listed in any order.
Type specifiers are: double, boolean

Example:
```
double Max( x, y ) {
  if x > y then
    return x;
  else
```

```
   return y;
}
main () {
  buy "MSFT" 100 0 0;
  buy "MSFT" Max(100, 50) 0 0;
}
```

3.4.4 Function Declarations

For all functions if the function definition is located after the calling function in the source code, the function must be declared before calling it.

3.4.5 Function Parameters and Arguments

UNIGA functions exchange information by means of parameters and arguments. The term parameter refers to any declaration within the parentheses following the function name in a function declaration or definition; the term argument refers to any expression within the parentheses of a function call.

The following rules apply to parameters and arguments of UNIGA functions:

- Except for functions with variable-length argument lists, the number of arguments in a function call must be the same as the number of parameters in the function definition. This number can be zero.
- The maximum number of arguments (and corresponding parameters) is 50 for a single function.
- Arguments are separated by commas. However, the comma is not an operator in this context, and the arguments can be evaluated by the compiler in any order. There is, however, a sequence point before the actual call.
- Arguments are passed by value; that is, when a function is called, the parameter receives a copy of the argument's value, not its address. This rule applies to all scalar values, structures, and unions passed as arguments.
- Modifying a parameter does not modify the corresponding argument passed by the function call.

3.4.6 Function invocation and return

Function call

A function call can be a single statement followed by a “;”.

Return statement

The return statement is used to return from the function at the point the return statement is specified Return statement can also return a value. The return statement is ended by a “;”.

3.4.7 Built-In functions

3.4.7.1 print/println

Console output functions, function print prints variable values, characters and strings. Function println prints entire line with return character.

Sample: *print "calculating portfolio profit and loss"*
println "calculating daily profit"

3.4.7.2 error()
Standard error handler

3.4.7.3 average(String stockID **)**
Average stock price of current date

3.4.7.4 sum()
Total asset value of a portfolio.

3.4.7.5 holdings()
List current detailed stock holdings of a portfolio

3.4.7.6 pl()
Profit and loss of a portfolio.

3.5 Expressions and Operators

3.5.1 Primary expressions
Simple expressions are called primary expressions; they denote values. Primary expressions include previously declared identifiers, constants, string literals, and parenthesized expressions.

Primary expressions have the following syntax:

primary-expression:
identifier
constant
string-literal
parenthesized expression

3.5.1.1 Identifier
An identifier is a primary expression provided it is declared as designating an object or a function.

3.5.1.2 Constant
A constant is a primary expression. Its type depends on its form (in UNIGA, it's Boolean or double)

3.5.1.3 String Literals
A string literal is a primary expression

3.5.1.4 Parenthesized Expressions
An expression within parentheses has the same type and value as the expression without parentheses would have. Any expression can be delimited by parentheses to change the precedence of its operators.

3.5.2 Postfix Operators
Postfix expressions include function calls and postfix increment and decrement expressions. The operators in postfix expressions have left-to-right associativity.

Postfix expressions have the following syntax:

postfix-expression:
function-call

3.5.2.1 Function Calls
Function calls have the following syntax:

function-call:
postfix-expression (argument-expression-list(opt))
argument-expression-list(opt):
assignment-expression
argument-expression-list(opt), assignment expression

A function call is a postfix expression consisting of a function designator followed by parentheses.

The order of evaluation of any expressions in the function parameter list is undefined, but there is a sequence point before the actual call. The parentheses can contain a list of arguments (separated by commas) or can be empty.

3.6 Binary Operators
The binary operators are categorized as follows:
- Multiplicative operators: multiplication (*), and division (/)
- Additive operators: addition (+) and subtraction (-)
- Relational operators: less than (<), greater than (>)
- Equality operators: equality (==)
- Logical operators: AND (&) and OR (|)

3.6.1 Multiplicative operators:
The multiplicative operators are *, /. Operands must have arithmetic type. Operands are converted, if necessary, according to the usual arithmetic conversion rules.
The * operator performs multiplication.
The / operator performs division.

3.6.2 Additive operators:

The additive operators + and - perform addition and subtraction. Operands are converted, if necessary, according to the usual arithmetic conversion rules.

3.6.3 Relational operators:

The relational operators compare two operands and produce a result of type int. The result is 0 if the relation is false, and 1 if it is true. The operators are: less than (<), greater than (>). Both operands must have an arithmetic type.
The relational operators associate from left to right.

3.6.4 Equality operators:

The equality operator: equal (==) produces a result of type double, so that the result of following statement is 1 if both operands have the same value, and 0 if they do not:

Exp1 == Exp2;

3.6.5 Logical operators:

The logical operators are AND (&) and OR (|). These operators guarantee left-to-right evaluation. The result of the expression (of type double) is either 0 (false) or 1 (true). The operands need not have the same type, but both types must be scalar. If the compiler can make an evaluation by examining only the left operand, the right operand is not evaluated.

3.7 Assignment Operators

Assignments result in the value of the target variable after the assignment. They can be used as sub-expressions in larger expressions.

Assignment expressions have two operands: a modifiable value on the left and an expression on the right. A simple assignment consists of the equal sign (=) between two operands:

Exp1 = Exp2;

The value of expression Exp2 is assigned to Exp1. The type is the type of Exp1, and the result is the value of Exp1 after completion of the operation.

3.8 Statements

This section describes the following kinds of statements in the UNIGA programming language.

Statements are executed in the sequence in which they appear in a function body. UNIGA supports the following types of statements:

- Compound statements
- Expression statements
- Null statements
- Selection statements
- Iteration statements

- Break statements
- Trading statements

3.8.1 Compound Statements
A compound statement, or block, allows a sequence of statements to be treated as a single statement.

A compound statement begins with a left brace, contains optional declarations followed optionally by statements, and ends with a right brace, as shown in the following example: compound-statement:

{declaration-list}? {statement-list}?
declaration-list:
declaration
| *declaration-list declaration*
statement-list:
statement
| *statement-list statement*

Example:

```
{
 x=2;
 y=3;
 if x > y then
   return x;
 else
   return y;
}
```

Block declarations are local to the block, and, for the rest of the block, they supersede other declarations of the same name in outer scopes.

3.8.2 Null Statement
A null statement is used to provide a null operation in situations where the grammar of the language requires a statement, but the program requires no work to be done. The null statement consists of a semicolon:
;
The null statement is useful with the if, while, and for statements. The most common use of this statement is in loop operations in which all the loop activity is performed by the test portion of the loop.

3.8.3 Selection Statement
The if statement has the following syntax:

if expression

then
statement
else(opt)
else-statement(opt)

The statement following the control expression is executed if the value of the control expression is true (non-zero). An if statement can be written with an optional else clause that is executed if the control expression is false (0).

3.8.4 Iteration Statement

An iteration statement, or loop, repeatedly executes a statement, known as the loop body, until the controlling expression is false (0). The control expression must have a scalar type. Iteration statement in UNIGA includes the following:

- The while statement evaluates the control expression before executing the loop body
- The for statement executes the loop body based on the evaluation of the second of three expressions

3.8.4.1 The while Statement

The while statement evaluates a control expression before each execution of the loop body. If the control expression is true (non-zero), the loop body is executed. If the control expression is false (0), the while statement terminates. The while statement has the following syntax:

while (expression)
statement

3.8.4.2 The for Statement

The for statement evaluates three expressions and executes the loop body until the second controlling expression evaluates to false (0). The for statement is useful for executing a loop body a specified number of times. The for statement has the following syntax:

for (expression-1(opt) ;
expression-2(opt) ; expression-3(opt))
statement

The for statement executes the loop body zero or more times. Semicolons (;) are used to separate the control expressions. A for statement executes the following steps:

- expression-1 is evaluated once before the first iteration of the loop. This expression usually specifies the initial values for variables used in the loop.
- expression-2 is any scalar expression that determines whether to terminate the loop. expression-2 is evaluated before each loop iteration. If the expression is true (nonzero), the loop body is executed. If the expression is false (0), execution of the for statement terminates.
- expression-3 is evaluated after each iteration.

- The for statement executes until expression-2 is false (0), or until a jump statement, such as break or goto, terminates execution of the loop.

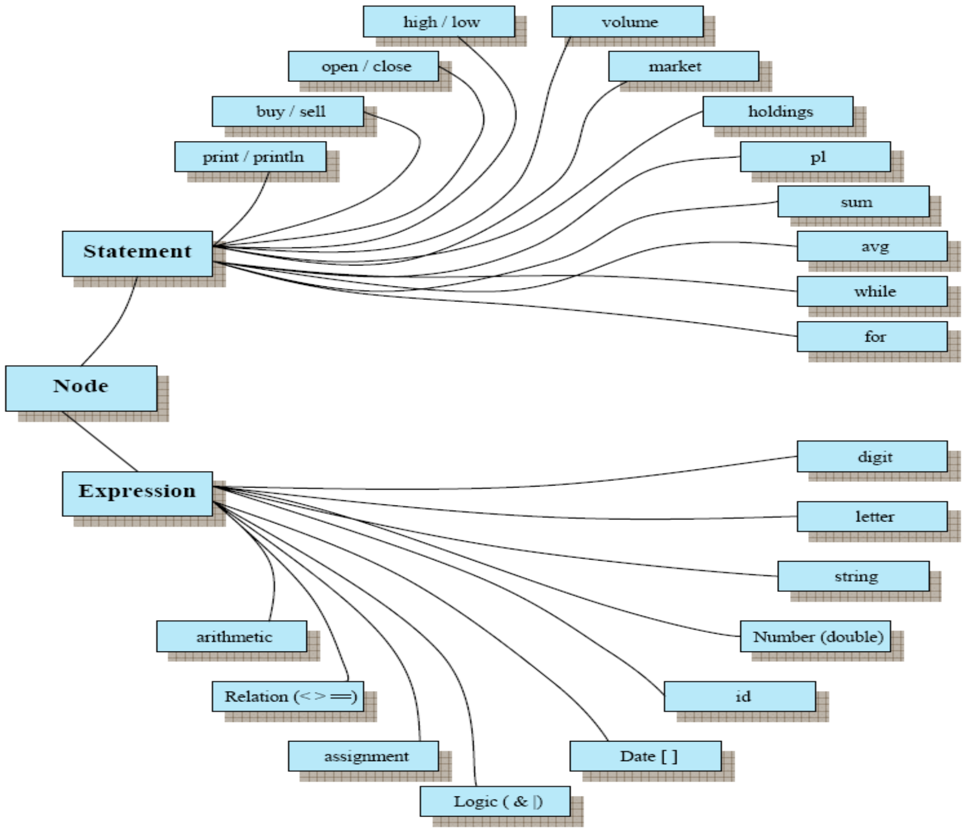
3.8.5 Break Statement
The break statement causes the termination of the enclosing while and for. The control passes to the statement following the terminated statement.

3.8.6 Return Statement
The return statement cause program control to return to the caller. An optional expression following the return keyword will cause the function to return the value of the expression to the caller. If required, the expression is converted, as if by assignment, to the type of the function in which it appears.

3.8.7 Trading Statement
UNIGA provides standard trading statements which are easy to use.

buy/sell stock_identifier expression1 expression2

stock_identifier is of type string, and is generally the trading stock identifier. expression1 is used to refer the volume to buy/sell. expression2 is used to refer the price at which to buy/sell.



3.9 References
[1] C Reference Manual, Dennis M.Ritchie

4. Project Plan

4.1 Project Processes Overview

The entire UNIGA project is divided into four phases:

- Project planning
- Project specification
- Project development (implementation)
- Testing

The specific processes involved in each phase are explained in the section below.

4.1.1 Project planning

Out of five members of our project team, four of us have been in a same team working on an Advanced Software Engineering course project one semester earlier. Having that experience on our back, we were more experienced in planning and carrying out a project's execution which are of large size and involves multiple members' teamwork. Right from the beginning, we set out to put three things in place:

1. ***Setting up the project development framework***: this involves deciding on the Java development language, the SVN source code version control system on CUNIX, and deciding on the common coding style and coding convention

2. ***Setting up incremental development approach***: we decided on abiding a incremental development approach (which proved to have worked well in our previous semester advanced software engineering projects) where we will first implement a small functional core for our UNIGA compiler as early as possible, and then expand on the language's basic functionalities by adding system built-in functions later on in the development stage. The functional core included basic language control flow implementation, function declaration, variable declaration and value assignment, data input and output. The system built-in functions included functions for reading in XML stock data, output XML portfolio data, portfolio profit and loss calculation, print to console function.

3. ***Assigning duty to each member of the team according to their skill sets and personal preferences***. Recollecting from past project experiences, we've learned that a group works out best if each team member's skill set is communicated thoroughly at the very beginning of the project, and the member is assigned with the portion of the work he feels comfortable and enjoys doing.

4. ***Set up responsibility reporting mechanism, and attain schedule milestone with best effort***: we realized it will be crucial that a there's a responsibility reporting mechanism in place throughout the project. This mechanism will set up multiple milestones in the entire project's schedule, guaranteeing all members attain their promised payload at each milestone's checkpoint. The mechanism also tries to define the actions that can be taken in case a member's workload was unable to be

delivered. Trying to attain schedule milestone with best effort provides two obvious advantages: first, small delays in earlier stages might propagate and become bigger delays into later stages, the delays might grow exponentially and eventually grow out of control, the best way to eliminate this is to abide the schedule strictly; second, incremental progresses will boost entire team morale and leave time buffer for unexpected delays that might happen later.

With the above project prerequisite already in place, we went on to specify the requirements to out specific project. We identified the topic of our compiler project to be a financial trading language based on two considerations: the first one is there's no prevalent language at the present time providing UNIGA's function which makes our implementation still unique; second, the scope and complexity involved in implementing this topic would allow us to finish the development in the given time frame within this semester. Having decided our UNIGA language target application, we further translated the application requirements into the specifications stated in the next section.

4.1.2 Project specification

After analyzing our project's target application: to provide an easy to learn high level language for executing trading strategies, we set out our language specifications accordingly:

We decided that in order for a language to function properly, the most bare essential language constructs such to assign variable values, compare values, control program logic flows must be in place, these include: arithmetic operators (+, -, *, /), relational operators (>, <, ==), logical operators (&, |), selection statements (if … else), and iteration statements (while( ), for( )). The data types was originally to include integer, floating point numbers, double precision numbers, and date variables, Boolean variables. But later, in order to make reduce complexity, we decided to only support the double precision type variable, which represents the stock price, volume and date. Only when representing date, the date value is differentiated from the rest data by being quoted in square brackets: [].

The application specific language construct include the following:

1. We created the "buy & sell" operator to support the action of buying and selling a particular stock, the exemplar grammar is as:" *buy/sell stock_identifier expression1 expression2*".

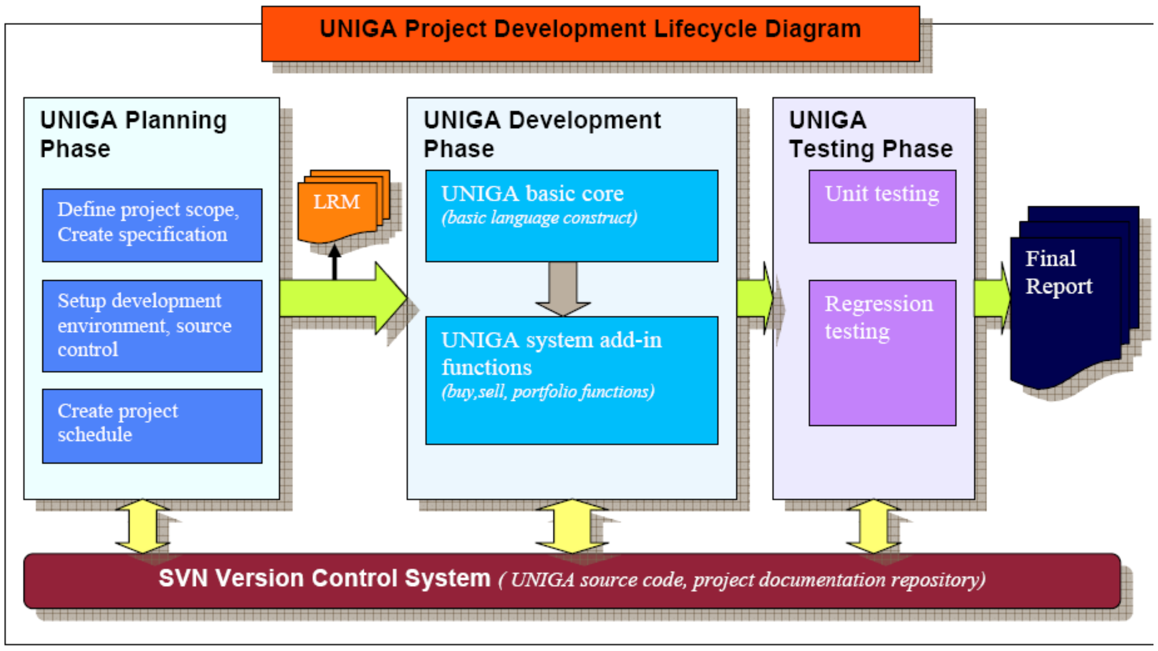
2. We defined that all input stock prices data at the beginning of the day are fed in as inputs in the format of XML files. We would create a group of system built-in functions to both read in these data and to output the data also as an XML file after the program finishes execution. We would also create a group of functions to calculate the profit/loss for a portfolio of stocks. These system built-in functions basically perform the most common functions that the user is likely to call on a daily operational basis.

3. We would give the user the language construct to define their own functions, this is also essential in today's any given functional languages, and we do support user functions declaration as well.

4.1.3 Project development

The project development was basically a strictly carried out plan according to our set out project schedule. In addition to abiding to the project schedules, we also enforced weekly meetings on every Monday noon before class to summarize previous week's progress and to determine the upcoming week's task; through out the week, team members communicate regularly, usually daily to report their development problems or to notify the team if a newer version of project file has been updated into the version control system; and then finally, usually on each Thursday afternoon of the week, we would have our TA meetings with our TA Neesha to determine our progress relative to what the project current is demanding, because the TA meeting happens during the middle of the week, it's another timeline correction we can have to make sure our week's effort on the project is well planned out.

The entire project development is done in an incremental way, listed as the graphical development phase chart below:


UNIGA Project Development Lifecycle Diagram
UNIGA Planning Phase
Define project scope, Create specification
Setup development environment, source control
Create project schedule
LRM
UNIGA Development Phase
UNIGA basic core
(basic language construct)
UNIGA system add-in functions
(buy,sell, portfolio functions)
UNIGA Testing Phase
Unit testing
Regression testing
Final Report
SVN Version Control System (UNIGA source code, project documentation repository)

The specific time line of our project development and role assignment for each team member is explained in section 4.3 and section 4.4 respectively.

4.1.4 Testing

Project testing was involved throughout the project in two different scales. One is testing done during the phase of development, this included testing whether simple UNIGA code snippets was compiled as expected after parser, lexer and walker have been created. Testing at this stage were more of a tool to correct obvious errors existent in the ANTLR language files. As the UNIGA basic core and system add-in functions were finished, system level tests were deigned and carried out to verify whether the compiler as an entirety qualifies to the expected performance measures demanded by the initial design. In the UNIGA testing phase, two kinds of tests were designed and implemented, unit testing and regression testing. Details of these two testing mechanisms and results are elaborated in section 6 of this report.

4.2 Programming Style Guide

Language	Programming Style Specifications
Java	Names representing packages should be in all lower case.
	Names representing types must be nouns and written in mixed case starting with upper case.
	Variable names must be in mixed case starting with lower case.
	Names representing constants (final variables) must be all uppercase using underscore to separate words.
	Names representing methods must be verbs and written in mixed case starting with lower case.
	Iterator variables should be called ***i, j, k*** etc
	Abbreviations in names should be avoided.
	Exception classes should be suffixed with ***Exception***.
	Classes should be declared in individual files with the file name matching the class name. Secondary private classes can be declared as inner classes and reside in the file of the class they belong to.
	The incompleteness of split lines must be made obvious
	The ***package*** statement must be the first statement of the file. All files should belong to a specific package
	The ***import*** statements must follow the ***package*** statement. ***import*** statements should be sorted with the most fundamental packages first, and grouped with associated packages together and one blank line between groups.
	Method modifiers should be given in the following order: ***<access> static abstract synchronized <unusual> final native*** The <*access*> modifier (if present) must be the first modifier.
	Variables should be initialized where they are declared and they should be declared in the smallest scope possible.
	Class variables should never be declared public.
	Basic indentation should be 2

4.3 Project Timeline

Date	Progress
January 17	Team Forming
January 22	Brain Storming
January 24	Determine a topic

January 25	Subversion source control created
January 29	Start to sketch language outline
Feb 5	Finish Language Proposal
Feb 7	Start writing LRM and Final Report
Feb 7	**Proposal delivered**
Feb 12	Finish outlining LRM and Final Report
Feb 19	Initial sample code complete
Feb 28	Finish all .g file and move to test phase
March 5	Test script completes
March 5	**Language Reference Manual (LRM) delivered**
March 25	Built-in functions and Java implementation files developed
April 1	Makefile completed
April 8	Split of grammar files into ParserLexer.g and Walker.g completed
April 12	System demo to TA
April 18	All Java files have been completed and tested
April 19	System demo to TA
April 26	**Final demo delievered**
May 2	Final Report completed
May 7	**Final report delivered**

4.4 Roles and Responsibilities

Member	Functional Part	Responsibility Details
Jiahua Ni Team Leader	Front end	Writing Parser, Lexer, Walker, language specification, testing cases
Leon Wu	Back end	Major system built-in functions, Source Control setup, development framework setup, specifications
Jian Pan	Back end, testing	Walker, Unit testing, Regression testing cases, LRM, final report
Yang Sha	Front end	Writing Parser, Lexer, Walker, language specification, built-in functions
Yu Song	Testing	Unit testing cases, Perl modules batch testing scripts, LRM, final report, regression test

4.5 Software Development Environment

UNIGA language was developed mainly by using Java. *ParserLexer.g* and *Walker.g* are ANTLR format and then be translated to Java codes. Besides these, other files are all written in Java code.

4.5.1 Operating Systems

Our development was based on Java and ANTLR and thus we could use both UNIX and Win32 environment. At home, we mainly use Win32 as the platform since it provides lots of JDK tools. For file version control, we made use of SVN system which is installed in UNIX machines in school.

4.5.2 Java 1.5

Java is a simple , object-oriented, network-savvy, interpreted, robust, secure, architecture neural, portable, high-performance, multithreaded, dynamic language.
Java is most suitable language for our project as it provides almost all the functionalities as standard library and thus significantly shortened both our coding and debugging time.

4.5.3 Perl 5.8.5

Perl took an active role in testing. It was obvious that, to test every detail, typing command line each time could be very time-consuming and inefficient, so we decided to use a script language to make this procedure automatically. Since Perl is superior to other languages in terms of regular expression presentation and it's execution speed, we chose Perl to serve this role. The detail is shown in test section.

4.5.4 ANTLR 2.7.5

ANTLR, which stands for Another Tool for Language Recognition, is a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing Java, C#, C++, or Python actions. ANTLR provides excellent support for tree construction, tree walking, and translation.

4.5.5 Subversion (SVN)

Subversion is an open source application for revision control. Also commonly referred to as svn or SVN, Subversion is designed specifically to be a modern replacement for CVS.

4.5.6 Putty 0.56

Putty is a free SSH, Telnet, rlogin software, and raw TCP client. It was originally available only for Windows, but is now also available on various Unix platforms.
We used this to connect to CUNIX machines to check out or add latest files and folders into SVN share folder. This tool is also recommended and provided by CUIT.

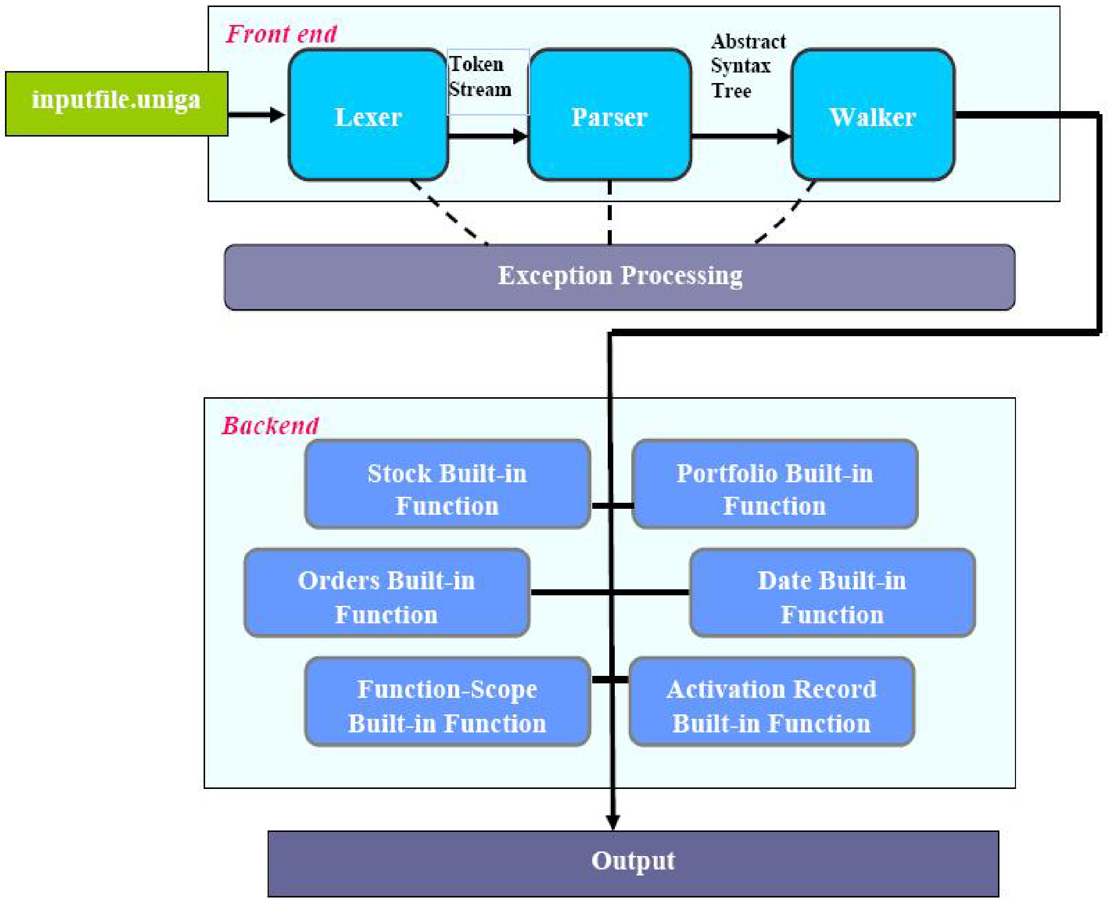
4.5.7 WinSCP 3.7.4

WinSCP (Windows Secure copy) is an open source SFTP client for Microsoft Windows. Its main function is secure file transfer between a local and a remote computer. Beyond this, WinSCP offers basic file manager functionality. It uses Secure Shell (SSH) and supports the legacy SCP protocol in addition to SFTP. This tool was extremely useful for us to securely download the newest files from SVN shared folder to each team member's local Win32 machine. This tool is also recommended and provided by CUIT.

5. Architectural Design

5.1 Architecture Diagram

Below is the architecture diagram.



The UNIGA language translator consists of these components:

- **Lexer (ParserLexer.g file)**: the lexer's role is to recognize strings and characters in the input UNIGA file and translate them into a stream of tokens. White spaces, comments and those characters or strings that are not defined in the lexer will be eliminated later on when constructing the Abstract Syntax Tree.

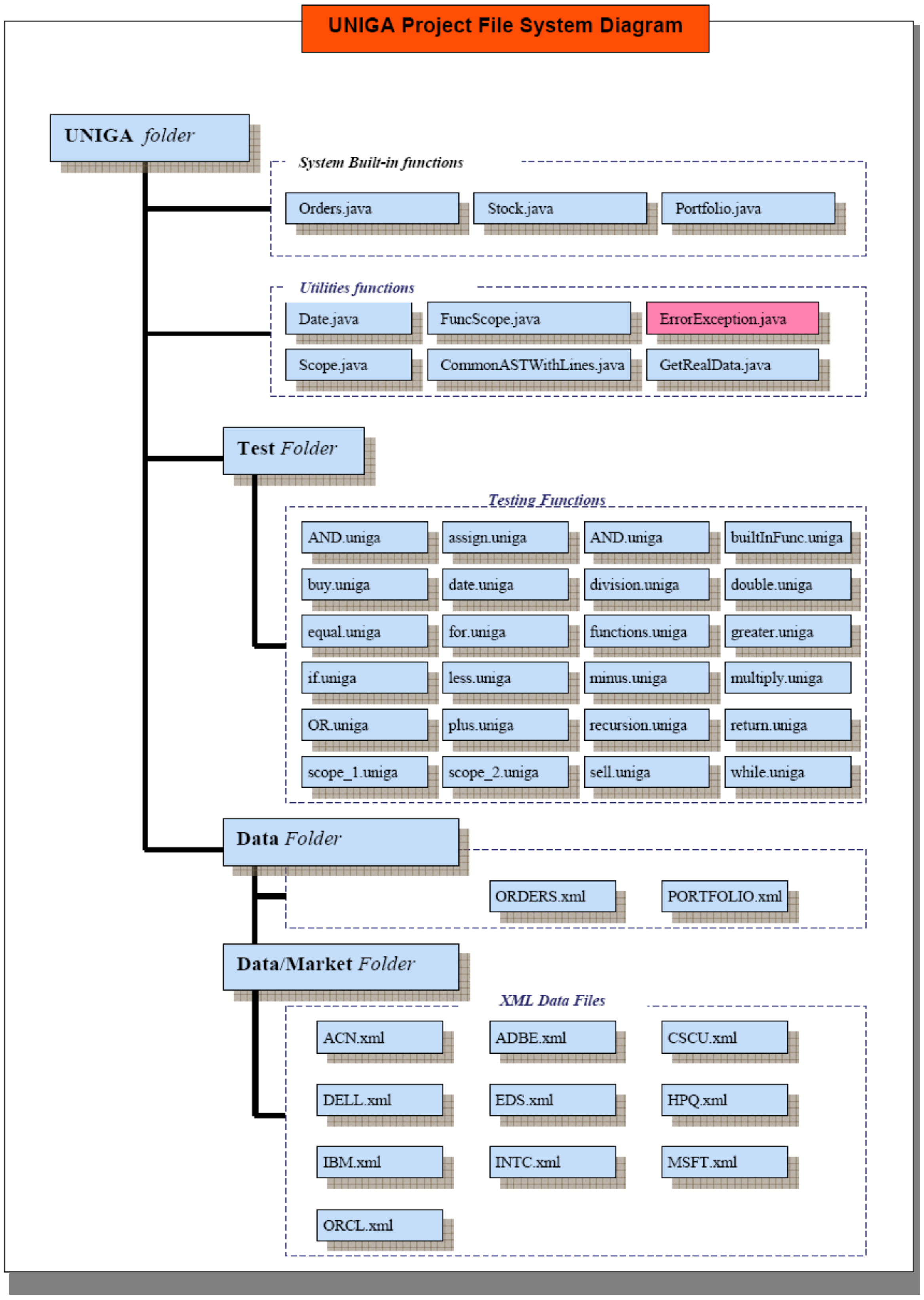
- **Parser (ParserLexer.g file):** the parser parses the input program file as a stream of tokens, check for grammar errors, does a syntax analysis of the program file, and create an Abstract Syntax Tree (AST) which represents the semantic structure of the program. In the parsing process, rules such as left associativity and operator precedence are of extreme importance as they resolve language ambiguity issues that may exist in parsing the program.

- **Tree Walker (Walker.g file)**: the tree walker walks through the Abstract Syntax Tree (AST) to form an intermediate representation of the program in the form of Intermediate Representation (IR) classes. Walking through the AST lets the compiler understand the semantics of the program, and enables the translated IR classes to be formed to be carried out by ANTLR java engine to perform the program logic.

- **Input/Output:** The application program gets market information stored in XML format as data input. The application program creates the portfolio data as XML formatted file. At the end of each day, an updated portfolio data file is created in XML formatted file. There will be specific system built-in functions created to perform these tasks respectively.

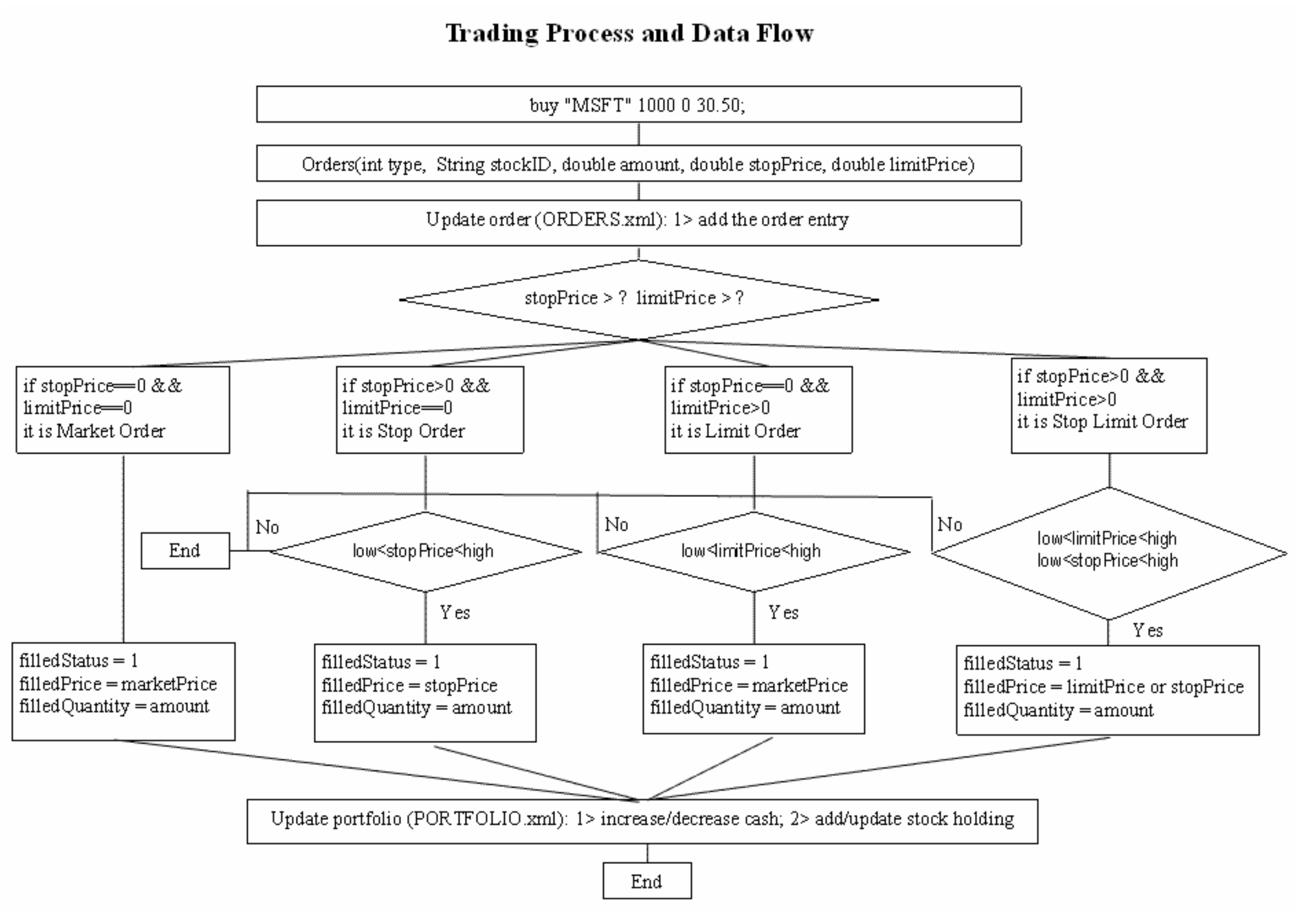
- **Exception handling:** exception classes are created to handle exceptions, and an exception line reporter will report which line the exception was thrown displayed at the console.

5.2 File System Diagram
Below is the file system structure.



5.3 Trading Process and Data Flow

Below is the diagram of trading process and data flow.


Trading Process and Data Flow
buy "MSFT" 1000 0 30.50;
Orders(int type, String stockID, double amount, double stopPrice, double limitPrice)
Update order (ORDERS.xml): 1> add the order entry
stopPrice > ? limitPrice > ?
if stopPrice==0 && limitPrice==0 it is Market Order
if stopPrice>0 && limitPrice==0 it is Stop Order
if stopPrice==0 && limitPrice>0 it is Limit Order
if stopPrice>0 && limitPrice>0 it is Stop Limit Order
End
No
low<stopPrice<high
No
low<limitPrice<high
No
low<limitPrice<high low<stopPrice<high
Yes
Yes
Yes
filledStatus = 1 filledPrice = marketPrice filledQuantity = amount
filledStatus = 1 filledPrice = stopPrice filledQuantity = amount
filledStatus = 1 filledPrice = marketPrice filledQuantity = amount
filledStatus = 1 filledPrice = limitPrice or stopPrice filledQuantity = amount
Update portfolio (PORTFOLIO.xml): 1> increase/decrease cash; 2> add/update stock holding
End

6. Test Plan

6.1 Testing Overview and Goals

Testing in our project is done on two levels: unit testing and regression testing. Detailed explanation of the unit testing and regression testing approach will be provided in the sections below. Unit testing was done for each new language construct added into the ANTLR paser, lexer, and walker files, while regression testing was done for all language constructs in the ANTLR grammar files when a new language was added in.

Testing Goals: The goal of unit testing is to ensure individual language constructs gets parsed correctly in the compiler. The goal of regression is to guarantee newly added language constructs doesn't break existing language elements. With unit testing and regression testing carried throughout the development cycle, the UNIGA compiler will ensure a comprehensive level of stability when developments are done.

Automation Utilized: Unit testing was mostly done manually to debug new language construct errors; the level of automation used was mostly using MAKEFILE to automate the compile and execution of single unit test cases. Regression testing was executed by conducting batch unit test cases which was automated by using Perl scripts.

All the test codes are listed in the Appendix.

6.2 Unit Testing

Unit testing is conducted to test the correctness of individual language constructs in the UNIGA grammar. Language constructs being tested for example include: assignment operators, arithmetic operators, iteration statements (for loop, while loop)… etc.

Our test script reads test cases one by one from "*test*" directory and then executes them in the same order as read in. Sound results are written into *sound_test_result.log* with execution date, time, each test case's name., and their output On the other hand, abnormal results are recorded in *bad_test_result.log* , also with the execution date, time, and each test case name, by redirecting stderr stream. Instead of normal outputs, in *bad_test_result.log,* you can check the error messages returned from system.

There are 37 unit test cases in our test suite which covers all of our language's major constructs. 31 of them are designed to test each single language elements such as "add", "assign", "built-in functions", and so on. The other 5 are nested combinations of single unit cases that are considered important to test. The last test case is a real situation code under the assumption that clients will place their order using this kind of format.

There are three scenarios in our unit test plan. First scenario is to make sure our grammar definitions is right. Second scenario is to ensure it pops out error messages or appropriate messages when it should. Three is to test the combinations of language constructs to verify they work properly as a whole system.

The first scenario, test each single unit or two units to ensure that our UNIGA grammar parsers everything correctly. Each test case is read by Perl program and be executed on the fly.

Second scenario is to test our grammar's error handling ability. This basically requires an exception is thrown if a test case is written using the incorrect grammar.

The last scenario is to check if our grammar as a whole works properly or not. This means, if correct grammar and incorrect grammar are present in a unit testing file, the correct part get executed and the incorrect part get parsed and exception be thrown.If anything abnormal happens, the error will be written into error log using stderr stream by Perl program.

unit_test.pm: the perl module file that has implementation of tester.

```
#------------------------------------------------------------------
# COMS 4115 test script introduced by Yu Song ys2310@columbia.edu
#------------------------------------------------------------------
#!/usr/local/gnu/bin/perl
#use FindBin;
#use lib "$FindBin::Bin/../lib";
my $test_dir = "./test";

sub create_unit_test_log {
#     open(OUT1, ">sound_test_result.log"); # open ouput stream
#     open(OUT2, ">bad_test_result.log");   # open ouput stream

      opendir IN, $test_dir or die "Couldn't open $test_dir: $!";
      my @files = map { "$test_dir/$_" } grep { /\.uniga$/ }
      readdir(IN);
      closedir IN;

      ($Second, $Minute, $Hour, $Day, $Month, $Year, $WeekDay, $DayOfYear, $IsDST) =
localtime(time);
      $Year += 1900;
      $date = "$Hour:$Minute:$Second, $Month/$Day/$Year";
      `echo "Last update ($date)" >sound_test_result.log`;
      `echo "Last update ($date)" >bad_test_result.log`;

      $i = 1;
      foreach $file (@files) {
#     sound log
              `echo >>sound_test_result.log`;
              `echo +++++++++ test case $i +++++++++ >>sound_test_result.log`;
              `echo $file >>sound_test_result.log`;
              `java Main $file 1>>sound_test_result.log 2>>null`;
              print OUT1 "\n";


#     bad log

              `echo >>bad_test_result.log`;

              `echo +++++++++ test case $i +++++++++ >>bad_test_result.log`;

              `echo $file >>bad_test_result.log`;
              `java Main $file 2>>bad_test_result.log`;
              print OUT2 "\n";

              $i++;
      }
#     close (OUT1);
#     close (OUT2);
}
```

uniga.pl: the base perl program that calls the function in *unit_test.pm* and make use of it.

```
#------------------------------------------------------------------
# COMS 4115 test script introduced by Yu Song ys2310@columbia.edu
#------------------------------------------------------------------
#!/usr/local/gnu/bin/perl
use unit_test;

create_unit_test_log();
```

Examples of individual unit test cases are listed below:

buy.uniga

```
main()
{
        buy "MSFT" 1000 0 28.5;
}
```

sell.uniga

```
main()
{
        println "sell 1000 shares Microsoft without setting stop and limit prices";
        sell "MSFT" 1000 0 0;
        println "sell 500 shares Microsoft with stop price 28.5 and limit price 28.8";
        sell "MSFT" 500 28.5 28.8;
}
```

assign.uniga

```
main()
{
        double a=1;
        if a==1 then{
                return 1;
        } else { return 0; }

}
```

for.uniga

```
main()
{
        double a=0;
        double i;
        for (i=0;i<5;i=i+1) {
                a=a+i;
                println a;
                a=a;
        }
        if a==10 then{
                return 1;
        } else {
                return 0;
        }
}
```

while.uniga

```
main()
{
        double a,i=0;
        while (i<5) {
                a=a+i;
                i=i+1;
        }
        if a==10 then{
                return 1;
        } else {
                return 0;
        }
}
```

return.uniga

```
double testReturn(double i){
  return i+1;
}
main()
{
       double i=0;
       if testReturn(i)-i>0 then print "correct return";
       else print "incorrect return";
}
```

portfolio.uniga

```
main()
{
       double s;
       s = sum();
       print "Sum of portfolio: ";
       print s;
       println " ";
       println "----------------------";

       buy "MSFT" 500 10.00 0;
       buy "MSFT" 500 28.50 0;
       sell "INTC" 500 0 0;
       buy "HPQ" 500 0 0;

       s = sum();
       print "Sum of portfolio: ";
       print s;
       println " ";
       println "----------------------";

       double r;
       r = pl();
       print "Profit and loss: ";
       print r;
       println " ";
       println "----------------------";

}
```

6.3 Regression Testing

Regression testing is conducted to ensure newly added UNIGA language constructs don't break existing language components. As the number of language elements we need to do regression testing increased following our advancing development cycle, we used Perl scripts to automate the regression task.

First ***regression.pl*** executes all the unit test cases again, stores the results in ***regression_temp.log*** file. Next, compare this new log with the original ***bad_test_result.log*** and if any regression bugs happened, it will be explicitly recorded in ***regression_test_result.log***. ***Rgression_test_result.log*** stores all the regression tests' results so far with the test-date.

regression.pl: this program reports the results of regression tests.

```
#-------------------------------------------------------------------
# COMS 4115 test script introduced by Yu Song ys2310@columbia.edu
#-------------------------------------------------------------------
#!/usr/local/gnu/bin/perl
#use FindBin;
#use lib "$FindBin::Bin/../lib";
my $test_dir = "./test";

        opendir IN, $test_dir or die "Couldn't open $test_dir: $!";
        my @files = map { "$test_dir/$_" } grep { /\.uniga$/ }
        readdir(IN);
        closedir IN;

        ($Second, $Minute, $Hour, $Day, $Month, $Year, $WeekDay, $DayOfYear, $IsDST) = localtime(time);
        $Year += 1900;
        $date = "$Hour:$Minute:$Second, $Month/$Day/$Year";
        `echo "Last update ($date)" >regression_temp.log`;
        `echo $date >>regression_test_result.log`;


        $i = 1;
        foreach $file (@files) {
#       regression log

                `echo >>regression_temp.log`;

                `echo +++++++++ test case $i +++++++++ >>regression_temp.log`;

                `echo $file >>regression_temp.log`;
                `java Main $file 2>>regression_temp.log`;
                $i++;
        }

        open(IN1, "<bad_test_result.log");  # open ouput stream
        open(IN2, "<regression_temp.log");  # open ouput stream
        @a = ();
        @b = ();

        while(<IN1>) {
                push(@a, $_);
        }
        while(<IN2>) {
                push(@b, $_);
        }
        close (IN1);
        close (IN2);

        foreach $file (@files) {
                $temp1 = "";
                $temp2 = "";

                foreach $a (@a) {
                        chomp $file;
                        chomp $a;
                        if ($a eq $file) {
                                $temp1 = $a;
                        }
                        elsif ($a =~ /\+/) {
                                if ($temp1 eq "") {
                                        # nothing
                                } else {
                                        last;
                                }
                        } else {
                                if ($temp1 eq "") {
                                        # nothing
                                } else {
                                        $temp1 = $temp1." ".$a;
                                }
                        }
                }#end of foreach @a
                foreach $b (@b) {
                        chomp $b;
                        if ($b eq $file) {
                                $temp2 = $b;
                        }
                        elsif ($b =~ /\+/) {
                                if ($temp2 eq "") {
                                        # nothing
                                } else {
                                        last;
                                }
                        } else {
                                if ($temp2 eq "") {
                                        # nothing
                                } else {
                                        $temp2 = $temp2." ".$b;
                                }
                        }
                }#end of foreach @b
#               print $temp1."\n";
#               print $temp2."\n";
#               `touch regression_test_result.log`;
                if ( length($temp2) > length($temp1) ) {
                        print $file." regression test FAILED\n";
                        `echo "++++++++++++++++++++++++++++++++++++++++++++++" >>regression_test_result.log`;
                        `echo $file." regression test FAILED" >>regression_test_result.log`;
                        `echo "++++++++++++++++++++++++++++++++++++++++++++++" >>regression_test_result.log`;
                } else {
                        print $file." regression test past\n";
                        `echo $file." regression test past">>regression_test_result.log`;
                }#end of if
        }
        `echo "-------------------------------------------------------\n">>regression_test_result.log`;
```

7. Lessons Learned

Jiahua Ni:

Use CVN to control all the files, but everyone should backup the current version before making any significant changes. There was one time we found the current version in CVN didn't work properly, however, we failed to backup the previous one and had to spend much time on recovering previous one manually.

Grammar file should be generated before the tree parser file. We got the error by Antlr "grammar file should be the last" when we split the grammar file with the tree parser file. Furthermore, Antlr doesn't continue on generating. We had to combine these two files again and then take much effort on finding the problem.

Start from a small working grammar core and add other components step by step. Thus, you can easily find out what component in the grammar caused an error rather than having to go through hundreds of lines to find the bug. For example, we didn't implement for/while/if-else/function/variable scope at first, and we start from a quite small working grammar core.

We were confused with expression and statement at first, and put the "open/close/high/low" clause in the statement part. Thus, we were not able to get the values of these clauses and assigned to some variable. Actually, those clauses which might be assigned to some variable should be put in the expression part.

When there are errors in the programs after AST tree is built, it is better for the interpreter to give the error information which includes the exact line number. However, the default function getLine() in CommonAST class always returns 0. Then, we derive a subclass of CommonAST class and use ASTFactory to handle this situation.

It is better to confirm the most part of the grammar as well as the tree parser before doing the back-end part. Thus, we are quite sure what is needed in the back-end part.

Yang Sha:

First of all, I learnt how to build up a large system from ground. I could hardly understand the grammars introduced on the paper and lecture notes, until implementing a small sample on my own computer. And by studying the codes and running the demo, I started to build up a very simple grammar file of UNIGA, which only dealt with "int" data format and "if" statement.

As the time went by, our progress seemed pretty fluently. However, one day, when we move a new folder onto the SVN server, the system cracked down. The reason was due to the lack of space on CUNIX server, where we deployed the SVN server. Since none of us kept the latest version, we almost lost all the files modified in the past month. It is, actually, a perfect case on disaster recovery. From then on, we kept a backup on another server as a backup, and all the daily progress were recorded.

Another important lesson I learnt is to do the testing as early as possible, and at the end of each development stage, there should be enough time arranged for unit testing and regression testing. Sometimes one of us forgot to do the testing before uploading the file, and when the next person modified the file, it was pretty hard to find out the bugs inside the codes written by others. Spending half of the time for testing seemed to be a waste of time during the development; however, it can accelerate the overall progress tremendously.

Jian Pan:

Although we gained experiences on handling large scale development projects in Advanced Software Engineering last semester, there were still many things learned by doing the PLT project in this course.

First, add at least a week more to your originally planned schedule when trying to write your first parser, lexer, walker, and also for getting yourself used to ANTLR. Although ANTLR has already automated many things for you in building up a well functioning compiler, there is always a learning curve at the beginning, especially for building a compiler. The beginning part may seem slow sometimes but I guess that's just the way it is, after you've come up with your first working parser, lexer, walker that can parse a simple assignment expression and get out the correct result, you may start to feel more comfortable and start to be on the fast track. So be patient and add a week's more time to your schedule. Also start from a small core and increment functionalities gradually. In our project, we started from variable declaration, assignment and moved scope and function-scope to the last, because scoping and function scoping is more complex, being well versed in debugging with ANTLR grammar will make debugging the later much easier.

Second, for testing, although it seemed to belong to rather late stage of the development process, but try to start early because if in case debugging the parser, lexer and walker takes up more time than expected, you might find that in the end you won't have that extra week to devote to testing, which is just as important to the overall project. The way we did testing, was we had two persons assigned to writing the unit testing cases for different language constructs very early on, almost during the middle of our entire project's schedule, once the framework is there, if the language constructs changes, we just modify the unit cases accordingly and we can unit test our newest changes. Testing isn't what you do at the last minute, it's what you would do along the way of development, if you've used your composed test cases to verify your language development, it's more likely at the end that your entire language construct will pass many of the unit tests. The above principle also applies to regression testing, which would benefit you to discover potential bugs along the way instead of finding out something at the very end.

Thirdly, about team work principles and communications. The general things to keep in mind are: know your team members and assign task to their expertise accordingly. Because four members of our group have worked in a project team previously, we had little difficulty assessing this. Also, sketch out a schedule and try your best to reach every milestone, single milestone delays may propagate and grow into delays you can't handle. We managed to keep our schedules on track for almost all milestones by sending reminder emails to every team member and requesting replies to assess progress. Also frequent communication among team members help to identify common problems early, instead of having last minute surprises. Having a modularized task approach also helped a lot and seems to be a good approach. We made sure every team member in our team has a good understanding of the parser, lexer and walker, and then dispatch each member to work on either improving the front end (parser, lexer, walker), or construct add-in functions/ testing parts. But understanding the mechanism of the parser, lexer and walker is the common base, each member could do much better on their section of the task after they've had a good understanding of this part.

Overall, I've learned a lot from this course and the project, and getting into the details of designing all the constructs of a language would make me understand much better the elements that would impact a program's performance.

Yu Song:

Prior to starting our project, we decided what we'd exactly like to make and each individual's role. Everybody in our team had a brainstorm, and originally there were about 30 ideas. We discussed very carefully and finally narrowed down to the one gained consensus of all of our five.

As such, we finished our proposal, LRM, final report template and basic functionality supported by UNIGA language by March. Since each of us was clear what s/he should do, with little interference, we could concentrate on our tasks.

Also, choosing a team leader is a really crucial part while doing a project as a team. This is because, at least for me, I felt that our team leader was the one look over the whole project progress and, when it stagnates, spur us on. Our team leader was nice in terms of managing team members as well. We gathered together to have a meeting at least once a week. When we were becoming dull during late time, she was also the one who reproached us for the slack.

As summary, I would say

One.   Start brainstorming ASAP

Two.   Choose a responsible leader

Three. Decide each member's role and avoid interference

Four. Meet regularly and keep making progress

Leon Wu:

One thing we did really well was effective time management. In the first week of the semester, we decided group organization and one hour weekly meeting time on every Monday right before the class. The weekly meeting helped us a lot. Even when there is no immediate deliverable due in the near weeks, we still have the meeting to discuss and do brainstorming for the project. Our steady pace throughout the semester helped us to work out the prototype far earlier than any other team and nearly completion three weeks earlier than final deadline. There was no last minute rush work in the project.

We tried to work efficiently as a group. The Columbia CUNIX servers are equipped with Subversion source control. Any student with CUID can login and use it, although most people don't know that it is provided for free. In the second week of the semester, I managed to set up Subversion source code control repository and granted team members permission to work on it. I also wrote a Subversion user manual, which is a cheat sheet of various SVN commands because other team members were new to Subversion. Soon, we were able to exchange source code, documents and other materials via Subversion. Although some glitches and versioning issues arose from Subversion due to its system design, overall we saved a lot of time and worked much more efficiently.

Development is tedious. We tried to simplify the development process with the help of effective development framework. Development involves multiple steps. The most repetitive work during development includes development environment login/set up, compilation and testing. At the initial stage of the coding, we worked out a development framework. We classified the development environments to Windows XP and Columbia CUNIX Unix/Linux server. We standardized the development environments and a simple README was written. It included the commands to set up two different environments such as CLASSPATH setup, steps to do compilation, steps to run the UNIGA test programs. With the clear instructions, any team member can quickly get into coding and testing mode whenever a Windows PC or CUNIX SSH connection is available. Soon, I also added Makefile, which further simplified the compilation process.

UNIGA, acronym of our team member's last character of last names, is designed to be a financial trading language with flexibility. During the system architecture design, we discussed and debated how we can make the implementation agile and make the system easy to be ported to other system. We decided to use XML file format to hold domain specific information such as Portfolio (cash and stock holdings), Stock (historic and current stock pricing information), and Orders (records of all submitted order transactions). There are only small amount of standard Java interface calls in the Antlr walker.g file. We developed the implementation Java files in a modularized fashion so that it is easy to customize the system when using different data source such as relational database or web services.

Overall, I think the project has been an exciting exploration and learning experience. Our team members put their best effort onto it. Although there are still a lot of places to be improved, we believe our UNIGA is a good prototype financial trading language.

8. Appendix

Following are complete code listings for the UNIGA project.

ParserLexer.g

```
//////////////////////////////////////////////////////////////////////
/*
 * Parser
 * Author: Jiahua Ni, Yang Sha, Jian Pan
 */
//////////////////////////////////////////////////////////////////////
class SimpParser extends Parser;

options
{
	exportVocab=SimpParser;
	buildAST = true;
	k = 2;
}

tokens
{
	STATEMENT;
	DECLS;
	SUBPROG;
	FUNCDEF;
	FOREXPR;
	FUNC_CALL;
	ARG_LIST;
	EXPR_LIST;
}

program:  functions EOF!;

functions: (function)* main;

main: "main"^ LPAREN! RPAREN! subprogram;

function: ret ID  LPAREN! decls RPAREN! subprogram
 {#function = #([FUNCDEF,"FUNCDEF"], function);};

decls : (declo  (COMMA! declo )*
        | ){ #decls = #([DECLS, "DECLS"], #decls); } ;

declo : ("double"^)  ID
       ;

subprogram : LBRACE! (stmt)* RBRACE!
             {#subprogram = #([SUBPROG,"SUBPROG"], subprogram);};

ret: "double" | "void";

stmt   :( bool
	| buystmt
	| varstmt
	| sellstmt
	| ptstmt
	| ptlnstmt
	| returnstmt
	| "break") SEMI!
	| subprogram
	| whilestmt
	| ifstmt
	| forstmt
	| holdingstmt
	;

ptstmt:  "print"^ (STRING | expr);

ptlnstmt: "println"^ (STRING | expr);

buystmt: "buy"^  STRING NUMBER expr1 expr2;

sellstmt:"sell"^ STRING NUMBER expr1 expr2;
```

```
highstmt:"high"^ STRING LBRACE! NUMBER RBRACE!;

lowstmt:"low"^ STRING LBRACE! NUMBER RBRACE!;

openstmt:"open"^ STRING LBRACE! expr RBRACE!;

closestmt:"close"^ STRING LBRACE! NUMBER RBRACE!;

volumnstmt: "volume"^ STRING LBRACE! NUMBER RBRACE!;

marketstmt: "market"^ STRING;

holdingstmt: "holdings" SEMI!;

holdingtypestmt: "holdingStockType" LPAREN! RPAREN!;

plstmt: "pl" LPAREN! RPAREN!;

sumstmt: "sum" LPAREN! RPAREN!;

avgstmt: "average"^ LPAREN! STRING RPAREN!;

whilestmt: "while"^ LPAREN! bool RPAREN! subprogram ;

forstmt : "for"^ LPAREN! forexpr SEMI!
                       forexpr SEMI!
                       forexpr RPAREN! stmt;

forexpr : (((bool)|/*nothing*/)) {#forexpr = #([FOREXPR, "FOREXPR"], forexpr); };


returnstmt : "return"^ (bool)?;

dateexpr:"date"^ LSQ! NUMBER RSQ!;

/* Function related definition */
func_call_stmt : ID LPAREN!  expr_list RPAREN!
                    {#func_call_stmt = #([FUNC_CALL,"FUNC_CALL"],
func_call_stmt);}
                    ;

expr_list
             : (((bool) (COMMA! (bool))*)
             |/*nothing*/)
             ;

ifstmt:"if"^ bool "then"! stmt
             (options {greedy=true;} : "else"! stmt)?;

varstmt
      : ("double"^) args
      ;

args
      : arg (COMMA! arg )*
      ;

arg   : ID (ASSIGN^ bool)?
      ;


bool:  join(OR^join)*
             ;

join:  equality(AND^equality)*;

equality: rel(EQ^rel)*;

rel:   expr((LT^|GT^)expr)*;

expr : (ID ASSIGN^ expr1) | expr1;
```

```
expr1 : expr2 ( (PLUS^ | MINUS^) expr2 )* ;

expr2 : expr3 ( (TIMES^ | DIV^) expr3 )* ;

expr3
       : ID
       | "("! expr ")"!
       | NUMBER
       | "true"
       | "false"
       | openstmt
       | closestmt
       | highstmt
       | lowstmt
       | volumnstmt
       | dateexpr
       | func_call_stmt
       | plstmt
       | sumstmt
       | avgstmt
       | marketstmt
       | holdingtypestmt
       ;



//////////////////////////////////////////////////////////////////////
/*
 * Lexer
 * Author: Jiahua Ni, Yang Sha
 */
//////////////////////////////////////////////////////////////////////
class SimpLexer extends Lexer;

options {
       exportVocab=SimpParser;
       testLiterals = false;
       k = 2;
       charVocabulary = '\3'..'\377';
       }

/*Expression*/
PLUS :              '+' ;
MINUS :      '-' ;
TIMES :      '*' ;
DIV :        '/' ;
ASSIGN :     '=' ;
LPAREN :     '(' ;
RPAREN :     ')' ;
LSQ:         '[' ;
RSQ:         ']' ;
SEMI :              ';' ;
EQ:          "==";
COLON:              ':';
LT:          '<';
GT:          '>';
AND:         '&';
OR:          '|';
LBRACE:      '{';
RBRACE:      '}';
COMMA:       ',';

protected LETTER:   ('a'..'z' | 'A'..'Z') ;
protected DIGIT :   '0'..'9' ;
ID options { testLiterals = true; }: LETTER (LETTER | DIGIT | '_')* ;
NUMBER :                  (DIGIT)+ ('.' (DIGIT)+ )?;
STRING :                  '"'! (~('"' | '\n' ) | ('"'! '"'))* '"'!;
WS :                      ( ' ' | '\t' | '\n' { newline(); } |
'\r' ){ $setType(Token.SKIP); } ;
```

Waker.g

```
//////////////////////////////////////////////////////////////////////
/*
 * Author: Yang Sha, Jiahua Ni, Jian Pan
 * Walker
 */
//////////////////////////////////////////////////////////////////////
class SimpWalker extends TreeParser;
options {
      importVocab=SimpParser;
}

{

      Scope scope;
      static int RETURN_TRUE=1;
      static int BREAK_TRUE=2;
      String s=null;
}

program
{

}     : (function)* main;


function {double return_type=0;}: #(FUNCDEF return_type = ret ID decls
subprogram:.) {FuncScope.functionDefine(#ID.getText(), #subprogram,
return_type);};

ret returns [double r=0]: ("void" {r=0;}) | ("double" {r=1;});

decls : #(DECLS args);

args : arg args| ;

arg : #("double" ID) {FuncScope.registerArgument(#ID.getText());}
      ;

main  {scope = ActivationRecord.create(null);double r;}: #("main" r=subprogram);

funexecute returns [double r=0]: #(SUBPROG  r=stmts){if(r==RETURN_TRUE) return
RETURN_TRUE;};

subprogram  returns [double r=0]: #(SUBPROG {scope.enter_scope();}r=stmts) {if
(r==RETURN_TRUE) return RETURN_TRUE; scope.leave_scope();};

stmts returns [double r= 0 ]: (r=stmt {
                                        if(r==RETURN_TRUE) return RETURN_TRUE;
                                        if(r==BREAK_TRUE) return BREAK_TRUE;}
                              )*;

stmt   returns [double r=0]
{
 double a, b, c;
 double price, lprice;
}
      : r= subprogram {
                    if(r==RETURN_TRUE) return RETURN_TRUE;
                    if(r==BREAK_TRUE) return BREAK_TRUE;
                    }
      | #("if" a=pred:expr {
      AST thenpart = pred.getNextSibling();
      AST elsepart = thenpart.getNextSibling();

      if (a != 0) r = stmt(thenpart);
      else if (elsepart != null) r = stmt(elsepart);
```

```
        else r = 0;
        }
        )
        |#("double" (assignvalue)*)
        |#("holdings" {Portfolio port=new Portfolio();port.holdings();}
       )
        |#("while" while_expr:. loop_body:.
                {

                        while (expr(#while_expr)!=0)
                        {
                                r=stmt(#loop_body);
                                if(r==RETURN_TRUE) return RETURN_TRUE;
                                if(r==BREAK_TRUE) break;
                        }
                }
                )

        |#("for" {a=1;}
                                #(FOREXPR e1:. {if(#e1!=null){this.stmt((#e1));}})
                                #(FOREXPR e2:. {if(#e2!=null){this.stmt((#e2));}})
                                #(FOREXPR e3:.) body:.
                {
                        while(a!=0)
                        {
                                if((#body)!=null){
                                        r=stmt((#body));
                                        if(r==RETURN_TRUE) return RETURN_TRUE;
                                        if(r==BREAK_TRUE) break;
                                }
                                if((#e3)!=null){
                                        stmt((#e3));
                                }
                                if((#e2)!=null){
                                        a=expr((#e2));
                                }
                        }
                }
          )
        |"break"{r=BREAK_TRUE;}
        | #("return"  (exp:.)? {if(#exp==null)FuncScope.addReturnValue(null);
                                                  else FuncScope.addReturnValue(new
Double(expr(#exp)));
                                        return RETURN_TRUE;}
                )
        | #("print"
        ( s:STRING { System.out.print(#s.getText()); }
                | a=expr { System.out.print(a); } ) )
     | #("println"
     ( z:STRING { System.out.println(#z.getText()); }
         | a=expr { System.out.println(a); } ) )
        | #("buy"
        ( m:STRING p:NUMBER price=bprice:expr lprice=qprice:expr{
                try{
         if(price<0) {System.out.println("Line:"+#m.getLine()+" Invalid Stop
                        Price."); System.exit(0);}
         if(lprice<0){System.out.println("Line:"+#m.getLine()+" Invalid Limit
                        Price."); System.exit(0);}
             Orders myOrder = new Orders(0,#m.getText(),expr(p), price, lprice);
                        }catch(Exception e){System.out.println("error opening
file");}} ))
        | #("sell"
        ( n:STRING q:NUMBER price=sprice:expr lprice=zprice:expr{
                try{
                  if(price<0) {
                     System.out.println("Line:"+#n.getLine()+" Invalid Stop
                                                             Price.");
                     System.exit(0);}
                  if(lprice<0){
                     System.out.println("Line:"+#n.getLine()+" Invalid Limit
                                                             Price.");
                     System.exit(0);}
                Orders myOrder = new Orders(1,#n.getText(),expr(q), price, lprice);
                  }catch(Exception e){
                        System.out.println("error opening file");}}
        ))

        |r=expr
        ;

assignvalue{double a;}:
        #(ASSIGN ID
                        {if(scope.check_in_present_scope(#ID.getText())==true)
                        {System.out.println("Line:"+#ID.getLine()+" Variable "
                        +#ID.getText()+ " already defined");System.exit(0);}}

                        a=expr {
                                scope.add_in_present_scope(#ID.getText(),a);

                        }
                )

        |ID {if(scope.check_in_present_scope(#ID.getText())==true)
                        {
                        System.out.println("Line:"+#ID.getLine()+" Variable "
                        +#ID.getText()+ " already defined");
                                System.exit(0);
                        }


                        scope.add_in_present_scope(#ID.getText(),0);

                };


expr returns [double r=0;]
{ double a,b; }
        : #(ASSIGN ID
                {

                  if(scope.get_variable(#ID.getText())==null)
                        {
                          System.out.println("Line:"+#ID.getLine()+" The
                                variable "+#ID.getText()+" does not exist");
                          System.exit(0);}}
                          a=expr {scope.add_or_modify(#ID.getText(),a);
                  }
            )

        | #("open" (openid:STRING {
                AST barAgoPart = openid.getNextSibling();

                if (barAgoPart!= null) a = expr(barAgoPart);
                else a = 0;
                Stock record = new Stock(#openid.getText());
                r=record.getStockPrice(2,expr(barAgoPart));


                }
                ))

         | #("close" (closeid:STRING {
                AST barAgoPart = closeid.getNextSibling();

                if (barAgoPart!= null) a = expr(barAgoPart);
                else a = 0;
                Stock record = new Stock(#closeid.getText());
                r=record.getStockPrice(3,expr(barAgoPart));


                }
                ))
      | #("holdingStockType" {Portfolio typeAmt=new Portfolio();
      r=typeAmt.holdingStockType();}
      )
  | #("pl" {Portfolio plIt=new Portfolio(); r=plIt.pl();}
      )
  | #("sum" {Portfolio sumIt = new Portfolio(); r=sumIt.sum();}
      )
  | #("average"
     (symbolID: STRING {Stock avgIt = new Stock(#symbolID.getText());
    r=avgIt.average(#symbolID.getText());})
      )
  | #("market"
       (symID: STRING {try{
                        GetRealData marketData = new GetRealData();
                        r=marketData.displayItems(#symID.getText());
                        }catch(
                             Exception e)
                             {System.out.println("Line:"+#symID.getLine()+"
                             market data unavailable.");
                          }}
       )
     )
     | #("high" (highid:STRING {
          AST barAgoPart = highid.getNextSibling();

          if (barAgoPart!= null) a = expr(barAgoPart);
          else a = 0;
          Stock record = new Stock(#highid.getText());
          r=record.getStockPrice(0,expr(barAgoPart));


          }
          ))

     | #("low" (lowid:STRING {
          AST barAgoPart = lowid.getNextSibling();

          if (barAgoPart!= null) a = expr(barAgoPart);
          else a = 0;
          Stock record = new Stock(#lowid.getText());
          r=record.getStockPrice(1,expr(barAgoPart));


          }
          ))
     | #("volume" (volumnid:STRING {
          AST barAgoPart = volumnid.getNextSibling();

          if (barAgoPart!= null) a = expr(barAgoPart);
          else a = 0;
          Stock record = new Stock(#volumnid.getText());
          r=record.getStockPrice(4,expr(barAgoPart));


          }
          ))


    | #("date" NUMBER{
                      a =Double.valueOf(#NUMBER.getText()).doubleValue();
                      if((a>20080000)
                           ||(a<19000000)){System.out.println(
                           "Line:"+#NUMBER.getLine()+" Date invalid");
                           System.exit(0);
                           }
                      Date newdate=new Date(a);
                      r=newdate.getunigadate();
                      }
          )

    | #(PLUS a=expr b=expr {
                        try{
                            r=a+b;
                        }catch(NumberFormatException e)
```

```
                            {System.out.println("Line:"+#PLUS.getLine()+"
                            invalid variables");
                            System.exit(0);

                            }
                      }
   )
 | #(MINUS a=expr b=expr { r = a - b; } )
 | #(TIMES a=expr b=expr { r = a * b; } )
 | #(DIV a=expr b=expr {
                      if(b==0.0){
                            System.out.println("Line:"+#DIV.getLine()+"
                            Divident cannot be zero");
                            System.exit(0);
                      }
                      r = a / b; }
 )
 | #(EQ a=expr b=expr { r =(a==b)?1:0; } )
 | #(LT a=expr b=expr { r =(a<b)?1:0; } )
 | #(GT a=expr b=expr { r =(a>b)?1:0; } )
 | #(AND a=expr b=expr { if((a!=0.0) && (b!=0.0))r=1;else r=0; } )
 | #(OR a=expr b=expr { if((a==0.0) && (b==0.0))r=0;else r=1; } )
 | #(FUNC_CALL ID (r=expr {FuncScope.setArgument(r);})*){
               r=0;
               scope = ActivationRecord.create(scope);
               scope.enter_scope();
               r=FuncScope.funcCall(this, #ID.getText(), scope);
               scope.leave_scope();
               scope = ActivationRecord.remove(scope);}

 | ID {
               Double t=scope.get_variable(#ID.getText());
               if(t==null)
               {
                     System.out.println("Line:"+#ID.getLine()+
                          " Variable "+#ID.getText() +" not found");
                     System.exit(1);
               }
               else
               {
                     r = t.doubleValue();
               }
        }
 | "true" {r=1;}
 | "false" {r=0;}
 | #(NUMBER { r =Double.valueOf(#NUMBER.getText()).doubleValue();})
 ;
```

Main.java

```
/*
 * Author: Leon Wu, Yang Sha
 * Main program for UNIGA
 */
import java.io.*;
import antlr.CommonAST;
import antlr.collections.AST;
import antlr.ASTFactory;
import antlr.RecognitionException;
import antlr.TokenStreamException;
import antlr.TokenStreamIOException;

class Main {

    public static void main( String[] args ) {

        if ( args.length >= 1 )
            execFile( args[args.length-1] );
        else
            System.out.println(“input file name”);

        System.exit( 0 );
    }

    public static void execFile( String filename ) {

        try
        {
            InputStream input = ( null != filename ) ?
                (InputStream) new FileInputStream( filename ) :
                (InputStream) System.in;

            SimpLexer lexer = new SimpLexer( input );

            SimpParser parser = new SimpParser( lexer );

         ASTFactory factory = new ASTFactory();
         factory.setASTNodeClass(CommonASTWithLines.class);
         parser.setASTFactory(factory);

            parser.program();

         //CommonAST tree = (CommonAST)parser.getAST();

         AST tree=parser.getAST();

        System.out.println("------AST tree------\n"+tree.toStringList()+"\n-----
        End of AST-----\n");

            SimpWalker walker = new SimpWalker();

         walker.program(tree);

            // Walker caller code here TO BE ADDED

        } catch( IOException e ) {

            System.err.println( "Error in I/O: " + e );

        } catch( RecognitionException e ) {

            System.err.println( "Error in Recognition: " + e );

        } catch( TokenStreamException e ) {

            System.err.println( "Error in Token stream: " + e );
            } catch( RuntimeException e ) {

                System.err.println( "Error in Runtime: " + e );

                e.printStackTrace();

        } catch( Exception e ) {

         System.out.println("error!!");

            System.err.println( "Error: " + e );

                  e.printStackTrace();
        }
    }

}
```

Orders.java

```
/*
 * Author: Leon Wu
 * File name: Orders.java
 * Process order information
 */

import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.xml.sax.*;
import org.w3c.dom.*;

public class Orders{

    String filename = new String("data/ORDERS.xml");
    String[] orderType = {"buy", "sell"};

    public Orders(){
    }

    public Orders(int type,  String stockID, double amount, double stopPrice,
double limitPrice){
            try
                  {
                        DocumentBuilderFactory docBuilderFactory =
                        DocumentBuilderFactory.newInstance();
                        DocumentBuilder docBuilder =
                        docBuilderFactory.newDocumentBuilder();
                       Document document = docBuilder.parse(new File(filename));
                        document.getDocumentElement().normalize();

                        // get the root of the document
                        Element root = document.getDocumentElement();

                        // build a new order
                        Element newDate = document.createElement( "Date" );
                        newDate.appendChild(
                                    document.createTextNode( getDate() ) );

                        Element newType = document.createElement( "Type" );
                        newType.appendChild(
                                    document.createTextNode( orderType[type] ) );

                        Element newID = document.createElement( "ID" );
                        newID.appendChild( document.createTextNode( stockID ) );
                        Element newAmount = document.createElement( "Amount" );

                        newAmount.appendChild( document.createTextNode( Double.t
                        oString(amount) ) );

                        Element newStop = document.createElement( "Stop" );
                        Text stopText =
                        document.createTextNode( Double.toString(stopPrice) );
                        newStop.appendChild( stopText );

                        Element newLimit = document.createElement( "Limit" );
                        Text limitText =
                        document.createTextNode( Double.toString(limitPrice) );
                        newLimit.appendChild( limitText );

     /* --------------------------Order Fullfilment Logic-------------------------
-- */
                        // determine fill status and fill price
                        String filledStatus =  "0";
                        double filledPrice = 0.0;

// market order: automaticly filled at market price = (high + low)/2 with
rounding to 2 digit decimal
                        if (stopPrice==0 && limitPrice==0){
                              filledStatus = "1";
                              Stock sk = new Stock(stockID);

                              filledPrice = Math.rint(100.0 *
                                   ((sk.getStockPrice(0,0)+sk.getStockPrice(1,0)
                                   )/2)) / 100.0;

                        }

            // limit order: fill if low < limit price < high, filled price = limit
price
                        if (stopPrice==0 && limitPrice>0){
                              Stock sk = new Stock(stockID);
                        if (limitPrice > sk.getStockPrice(1,0) && limitPrice <
                                         sk.getStockPrice(0,0)){
                                    // filled
                                    filledStatus = "1";
                                    filledPrice = limitPrice;
                              } else {
                                    // not filled
                                    filledStatus = "0";
                                    filledPrice = 0.0;
                              }
                        }

            // stop order: fill if low < stop price < high, filled price = stop
price
                        if (stopPrice>0 && limitPrice==0){
                              Stock sk = new Stock(stockID);
                              if (stopPrice > sk.getStockPrice(1,0) && stopPrice
<

                                 sk.getStockPrice(0,0)){
                                    // filled
                                    filledStatus = "1";
                                    filledPrice = stopPrice;
                              } else {
                                    // not filled
                                    filledStatus = "0";
                                    filledPrice = 0.0;
                              }
                        }

                        // stop limit order: limit price take precedence
                        if (stopPrice>0 && limitPrice>0){
                              Stock sk = new Stock(stockID);
                        if (limitPrice > sk.getStockPrice(1,0) && limitPrice <
                              sk.getStockPrice(0,0)){
                                    // filled
                                    filledStatus = "1";
                                    filledPrice = limitPrice;
                              }else if (stopPrice > sk.getStockPrice(1,0) &&
                                   stopPrice < sk.getStockPrice(0,0)){
                                    // filled
                                    filledStatus = "1";
                                    filledPrice = stopPrice;
                              }else{
                                    // not filled
                                    filledStatus = "0";
                                    filledPrice = 0.0;
                              }
                        }
        /* ------------------------End of Order Fullfilment Logic----------------
------- */

                       Element newFilledStatus =
                           document.createElement( "FilledStatus" );

                       newFilledStatus.appendChild( document.createTextNode(fille
                       dStatus) );

                       Element newFilledPrice =
                          document.createElement( "FilledPrice" );

                       newFilledPrice.appendChild( document.createTextNode( Doubl
                       e.toString(filledPrice)) );

                        Element newOrder = document.createElement( "Record" );
                        newOrder.appendChild( newDate );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newType );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newID );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newAmount );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newStop );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newLimit );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newFilledStatus );
                        newOrder.appendChild(document.createTextNode("\n"));
                        newOrder.appendChild( newFilledPrice );
                        newOrder.appendChild(document.createTextNode("\n"));

                        // Add the new order to the root
                        root.appendChild(newOrder);
                        root.appendChild(document.createTextNode("\n"));
                        document.getDocumentElement().normalize();

                        // Write out the XML
                       TransformerFactory tf = TransformerFactory.newInstance();
                        Transformer transformer = tf.newTransformer();
                        DOMSource source = new DOMSource(document);
                        transformer.setOutputProperty(OutputKeys.ENCODING,"UTF-
8");

                        transformer.setOutputProperty(OutputKeys.INDENT,"yes");
                        PrintWriter pw = new PrintWriter(new
                                     FileOutputStream(filename));
                        StreamResult result = new StreamResult(pw);
                        transformer.transform(source, result);
                        pw.close();

                        // Print out order details
                        System.out.println("------------------------------");
                        System.out.println("Date:
                        "+newDate.getFirstChild().getNodeValue());
                        System.out.println("Order Type:
                        "+newType.getFirstChild().getNodeValue());
                        System.out.println("Stock ID:
                        "+newID.getFirstChild().getNodeValue());
                        System.out.println("Amount:
                        "+newAmount.getFirstChild().getNodeValue());
                        System.out.println("Stop Price:
                        "+newStop.getFirstChild().getNodeValue());
                        System.out.println("Limit Price:
                        "+newLimit.getFirstChild().getNodeValue());
                        System.out.println("Filled Status:
                        "+newFilledStatus.getFirstChild().getNodeValue());
                        System.out.println("Filled Price:
                        "+newFilledPrice.getFirstChild().getNodeValue());
                        System.out.println("------------------------------\n");

                        // Update Portfolio
                        if (filledStatus.equals("1")){
                              Portfolio po = new Portfolio();
                              po.newOrder(getDate(), type, stockID, amount,
                                                      filledPrice);
                        }

                  } catch( Exception e ) {
                        e.printStackTrace();
                  }
    }

    protected String getDate(){
            Calendar cal = new GregorianCalendar();
            int day = cal.get(Calendar.DAY_OF_MONTH);
            int month = cal.get(Calendar.MONTH)+1;
            int year = cal.get(Calendar.YEAR);
            return month + "/" + day + "/" + year;
    }

}
```

Stock.java

```
/*
 * Author: Leon Wu
 * File name: Stock.java
 * Read stock information (open, high, low, close, volume)
 * from XML file based on given stockid and date
 */

import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import org.w3c.dom.*;
import org.xml.sax.*;

public class Stock {

     static int day, month, year;
     String filename = "";

     public Stock(String ID){
            filename = "data/market/"+ID+".xml";
     }

     // getStockPrice returns the price info for days ago
     // typeID: 0=high; 1=low; 2=open; 3=close; 4=volume
     public double getStockPrice(int typeID, double days){
```

```
String dateValue = "";
String openValue = "";
String highValue = "";
String lowValue = "";
String closeValue = "";
String volumeValue = "";
double returnValue = 0;

if (typeID<0 || typeID>=5) return -1;

String theDate = getDate((int)days);


try {

      DocumentBuilderFactory docBuilderFactory =
      DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder =
         docBuilderFactory.newDocumentBuilder();
      Document doc = docBuilder.parse(new File(filename));
      doc.getDocumentElement().normalize();

      // get the root of the document
      Element root = doc.getDocumentElement();

      NodeList listOfRecords = doc.getElementsByTagName("Record");

      for(int s=0; s<listOfRecords.getLength() ; s++){

      Node firstRecordNode = listOfRecords.item(s);

      if(firstRecordNode.getNodeType() == Node.ELEMENT_NODE){

      Element firstRecordElement = (Element)firstRecordNode;

      NodeList dateList =
              firstRecordElement.getElementsByTagName("Date");
      Element dateElement = (Element)dateList.item(0);
      dateValue = dateElement.getFirstChild().getNodeValue().trim();

      NodeList openList =
              firstRecordElement.getElementsByTagName("Open");
      Element openElement = (Element)openList.item(0);
      openValue = openElement.getFirstChild().getNodeValue().trim();

      NodeList highList =
              firstRecordElement.getElementsByTagName("High");
      Element highElement = (Element)highList.item(0);
      highValue = highElement.getFirstChild().getNodeValue().trim();

      NodeList lowList = firstRecordElement.getElementsByTagName("Low");
      Element lowElement = (Element)lowList.item(0);
      lowValue = lowElement.getFirstChild().getNodeValue().trim();

      NodeList closeList =
              firstRecordElement.getElementsByTagName("Close");
      Element closeElement = (Element)closeList.item(0);
      closeValue = closeElement.getFirstChild().getNodeValue().trim();

      NodeList volumeList =

      firstRecordElement.getElementsByTagName("Volume");
      Element volumeElement = (Element)volumeList.item(0);
      volumeValue = volumeElement.getFirstChild().getNodeValue().trim();

      if (theDate.equals(dateValue)) {
            switch (typeID)     {
                  case 0:
                        returnValue = Double.valueOf(highValue);
                        break;
                        case 1:
                                returnValue = Double.valueOf(lowValue);
                                break;
                        case 2:
                                returnValue = Double.valueOf(openValue);
                                break;
                        case 3:
                                returnValue = Double.valueOf(closeValue);
                                break;
                        case 4:
                                returnValue = Double.valueOf(volumeValue);
                                break;
                        default:
                                returnValue = -1;
                                break;
                        }
                }
        }
}

     }catch (SAXParseException err) {
          System.out.println ("** Parsing error" + ", line "
          + err.getLineNumber () + ", uri " + err.getSystemId ());
          System.out.println(" " + err.getMessage ());
     }catch (SAXException e) {
          Exception x = e.getException ();
          ((x == null) ? e : x).printStackTrace ();
     }catch (Throwable t) {
          t.printStackTrace ();
     }

        return returnValue;
}

   // get date for prior n days
private String getDate(int days){
        // if number of days (ago) is negative, use current date
        if (days<0) days=0;

        Calendar cal = new GregorianCalendar();
        day = cal.get(Calendar.DAY_OF_MONTH);
        month = cal.get(Calendar.MONTH)+1;
        year = cal.get(Calendar.YEAR);
        if(day-days<=0){
                if(month-1<=0){
                        year-=1;
                        month=12;
                        day=31;
                }
                else{
                        month-=1;
                        day=getLastDayOfMonth(month,year);
                }
        }
        else{
                day-=days;
        }
        return month+"/"+day+"/"+year;
}

   // get days of last month
private int getLastDayOfMonth(int month, int year){
        int numDays = 0;
        switch (month) {
                case 1:
                case 3:
                case 5:
                case 7:
                case 8:
                case 10:
                case 12:
                        numDays = 31;
                        break;
                    case 4:
                    case 6:
                    case 9:
                    case 11:
                        numDays = 30;
                        break;
                    case 2:
                        if (isLeap(year))
                            numDays = 29;
                        else
                            numDays = 28;
                        break;
                    default:
                        numDays = -1;
                        break;
                }
            return numDays;
    }

        // is leap year or not
    private boolean isLeap(int year){
            boolean leap;
            if (year % 400 == 0){
                    leap = true;
            }
            else if (year % 100 == 0){
                    leap = false;
            }
            else if (year % 4 == 0){
                    leap = true;
            }
            else{
                    leap = false;
            }
            return leap;
    }

        // average price of a stock for current date
        public double average(String ID){
            double avg = 0.0;
            Stock sk = new Stock(ID);
            avg = Math.rint(100.0 *
                    ((sk.getStockPrice(0,0)+sk.getStockPrice(1,0))/2)) / 100.0;
            return avg;
        }

}
```

Portfolio.java

```
/*
 * Author: Leon Wu
 * File name: Portfolio.java
 * Process portfolio
 */
import java.io.*;
import java.util.*;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.xml.sax.*;
import org.w3c.dom.*;
```

```
public class Portfolio {

     String filename = new String("data/PORTFOLIO.xml");

        public Portfolio(){
        }

        // new filled order processing
 public void newOrder(String orderDate, int type, String stockID, double amount,
 double filledPrice){

            double tempAmount = 0.0;
            boolean stockOwned = false;
            String dateValue = "";
            String idValue = "";
            String amountValue = "";
            try
                   {
                          DocumentBuilderFactory docBuilderFactory =
                          DocumentBuilderFactory.newInstance();

                          DocumentBuilder docBuilder =
                          docBuilderFactory.newDocumentBuilder();
                          Document doc = docBuilder.parse(new File(filename));
                          doc.getDocumentElement().normalize();

                          // get the root of the document
                          Element root = doc.getDocumentElement();

                          NodeList listOfRecords =
                                 doc.getElementsByTagName("Record");

                          for(int s=0; s<listOfRecords.getLength() ; s++){

                                 Node firstRecordNode = listOfRecords.item(s);

                                 if(firstRecordNode.getNodeType() ==
                                      Node.ELEMENT_NODE){

                                 Element firstRecordElement =
                                         (Element)firstRecordNode;

                                 NodeList dateList =
                                   firstRecordElement.getElementsByTagName("Date");

                                 Element dateElement = (Element)dateList.item(0);
                                 dateValue =
                                 dateElement.getFirstChild().getNodeValue().trim();

                                 NodeList idList =
                                 firstRecordElement.getElementsByTagName("ID");

                                 Element idElement = (Element)idList.item(0);
                                 idValue =
                                           idElement.getFirstChild().getNodeValue()
                                           .trim();

                                 NodeList amountList =
                                 firstRecordElement.getElementsByTagName("Amount");

                                 Element amountElement =
                                         (Element)amountList.item(0);
                                 amountValue =
                                      amountElement.getFirstChild().getNodeValue().
                                      trim();

                                        if (idValue.equals("CASH")) {
                                               if (type==0){
                                                      // buy order
                                               tempAmount =
                                               Double.valueOf(amountValue) -
                                               filledPrice*amount;

                                               } else if (type==1){
                                                      // sell order
                                                     tempAmount =
                                                          Double.valueOf(amountValue)
                                                          + filledPrice*amount;

                                               }



                                               dateElement.getFirstChild().setNodeV
                                               alue(orderDate);

                                              amountElement.getFirstChild().setNodeV
                                              alue(Double.toString(tempAmount));
                                        }

                                        if (idValue.equals(stockID)) {
                                               // this stockID is alreay in the
                 holdings

                                               stockOwned = true;

                                               if (type==0){
                                                      // buy order
                                               tempAmount =
                                               Double.valueOf(amountValue) + amount;
                                               } else if (type==1){
                                                      // sell order
                                               tempAmount =
                                               Double.valueOf(amountValue) - amount;
                                               }

                                           dateElement.getFirstChild().setNodeValue
                                           (orderDate);

                                              amountElement.getFirstChild().setNodeV
                                              alue(Double.toString(tempAmount));
                                        }
                                 }
                          }

                          // if the stockID is new, add a new entry to portfolio
                          if (!stockOwned){

                                 Element newDate = doc.createElement( "Date" );
                              newDate.appendChild( doc.createTextNode( orderDate ) );

                                 Element newID = doc.createElement( "ID" );
                                newID.appendChild( doc.createTextNode( stockID ) );

                                 Element newAmount = doc.createElement( "Amount" );
                                 if (type==0){
                                        // buy order
                                        tempAmount = amount;
                                 } else if (type==1){
                                        // sell order
                                        tempAmount = 0 - amount;
                                 }

                               newAmount.appendChild( doc.createTextNode(
                                 Double.toString(tempAmount) ) );

                                 Element newOrder = doc.createElement( "Record" );
                                 newOrder.appendChild( newDate );
                                 newOrder.appendChild(doc.createTextNode("\n"));
                                 newOrder.appendChild( newID );
                                 newOrder.appendChild(doc.createTextNode("\n"));
                                 newOrder.appendChild( newAmount );
```

```
                                newOrder.appendChild(doc.createTextNode("\n"));

                                // Add the new order to the root
                                root.appendChild(newOrder);
                                root.appendChild(doc.createTextNode("\n"));
                                doc.getDocumentElement().normalize();
                        }

                        // Write out the XML
                        TransformerFactory tf = TransformerFactory.newInstance();
                        Transformer transformer = tf.newTransformer();
                        DOMSource source = new DOMSource(doc);
                        transformer.setOutputProperty
                                     (OutputKeys.ENCODING,"UTF-8");
                        transformer.setOutputProperty(OutputKeys.INDENT,"yes");
                        PrintWriter pw = new PrintWriter(new
                                 FileOutputStream(filename));
                        StreamResult result = new StreamResult(pw);
                        transformer.transform(source, result);
                        pw.close();

                } catch( Exception e ) {
                        e.printStackTrace();
                }

         }

        public double holdingStockType() {
         try{
             DocumentBuilderFactory docBuilderFactory =
             DocumentBuilderFactory.newInstance();
              DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
              Document doc = docBuilder.parse(new File(filename));
              doc.getDocumentElement().normalize();

              // get the root of the document
              Element root = doc.getDocumentElement();

              NodeList listOfRecords = doc.getElementsByTagName("Record");
              // CASH is not a kind of stock
              return listOfRecords.getLength()-1;
         }catch( Exception e) {
              e.printStackTrace();
              return 0;
         }
        }
         // print portfolio holdings
         public void holdings() {
                String dateValue = "";
                String idValue = "";
                String amountValue = "";
                try
                        {

                            DocumentBuilderFactory docBuilderFactory =
                            DocumentBuilderFactory.newInstance();
                            DocumentBuilder docBuilder =
                                       docBuilderFactory.newDocumentBuilder();
                            Document doc = docBuilder.parse(new File(filename));
                            doc.getDocumentElement().normalize();

                              // get the root of the document
                             Element root = doc.getDocumentElement();

                             NodeList listOfRecords =
                                        doc.getElementsByTagName("Record");

                                  System.out.println("Holdings");
                                  System.out.println("-----------------------------
-");
```

```
                        for(int s=0; s<listOfRecords.getLength() ; s++){

                              Node firstRecordNode = listOfRecords.item(s);

                              if(firstRecordNode.getNodeType() ==
                                                Node.ELEMENT_NODE){

                              Element firstRecordElement =
                                          (Element)firstRecordNode;

                                NodeList dateList =
                                firstRecordElement.getElementsByTagName("Date");

                                Element dateElement = (Element)dateList.item(0);
                                     dateValue =
                                     dateElement.getFirstChild().getNodeValue().
                                     trim();

                                     NodeList idList =
                                     firstRecordElement.getElementsByTagName(
                                                                         "ID");
                                    Element idElement = (Element)idList.item(0);
                                     idValue =
                                     idElement.getFirstChild().getNodeValue().tr
                                     im();

                                     NodeList amountList =
                                     firstRecordElement.getElementsByTagName(
                                                                     "Amount");
                                      Element amountElement =
                                         (Element)amountList.item(0);
                                     amountValue =
                                     amountElement.getFirstChild().getNodeValue(
                                     ).trim();

                                    // Print out holding details
                                    System.out.println("Date: "+dateValue);
                                    System.out.println("Stock ID: "+idValue);
                                    System.out.println("Amount: "+amountValue);
                              }
                        }

                              System.out.println("-----------------------------
-\n");

                  } catch( Exception e ) {
                        e.printStackTrace();
                  }
      }

      // total value of portfolio
      public double sum(){
            double tempAmount = 0.0;
            String dateValue = "";
            String idValue = "";
            String amountValue = "";

            try
                  {
                        DocumentBuilderFactory docBuilderFactory =
                        DocumentBuilderFactory.newInstance();
                        DocumentBuilder docBuilder =
                        docBuilderFactory.newDocumentBuilder();
                        Document doc = docBuilder.parse(new File(filename));
                        doc.getDocumentElement().normalize();

                        // get the root of the document
                        Element root = doc.getDocumentElement();

                        NodeList listOfRecords =
                                  doc.getElementsByTagName("Record");
```

```
                    for(int s=0; s<listOfRecords.getLength() ; s++){

                        Node firstRecordNode = listOfRecords.item(s);

                        if(firstRecordNode.getNodeType() ==
                                                      Node.ELEMENT_NODE){

                        Element firstRecordElement =
                                   (Element)firstRecordNode;

                            NodeList dateList =
                            firstRecordElement.getElementsByTagName("Date
                            ");

                            Element dateElement =
                                (Element)dateList.item(0);
                            dateValue = dateElement.getFirstChild().
                                        getNodeValue().trim();

                            NodeList idList =
                            firstRecordElement.getElementsByTagName("ID");

                            Element idElement = (Element)idList.item(0);
                            idValue = idElement.getFirstChild().
                                       getNodeValue().trim();
                            NodeList amountList = firstRecordElement.
                                     getElementsByTagName("Amount");
                            Element amountElement =
                                     (Element)amountList.item(0);
                            amountValue = amountElement.getFirstChild().
                                     getNodeValue().trim();

                             if (idValue.equals("CASH")) {
                                 tempAmount +=
                                      Double.valueOf(amountValue);
                             } else {
                                 Stock sk = new Stock(idValue);
                                 tempAmount +=
                                 Double.valueOf(amountValue)*sk.
                                 getStockPrice(3, 0);
                                   }
                        }
                    }

                } catch( Exception e ) {
                    e.printStackTrace();
                }
              return tempAmount;
    }

      // total value of portfolio
    public double sumup(String file){
        double tempAmount = 0.0;
        String dateValue = "";
        String idValue = "";
        String amountValue = "";
        try
                {
                   DocumentBuilderFactory docBuilderFactory =
                   DocumentBuilderFactory.newInstance();
                    DocumentBuilder docBuilder =
                    docBuilderFactory.newDocumentBuilder();
                    Document doc = docBuilder.parse(new File(file));
                    doc.getDocumentElement().normalize();

                    // get the root of the document
                    Element root = doc.getDocumentElement();

                    NodeList listOfRecords =
                               doc.getElementsByTagName("Record");
                        for(int s=0; s<listOfRecords.getLength() ; s++){

                            Node firstRecordNode = listOfRecords.item(s);

                            if(firstRecordNode.getNodeType() ==
                                                  Node.ELEMENT_NODE){

                            Element firstRecordElement =
                                                (Element)firstRecordNode;

                            NodeList dateList =
                              firstRecordElement.getElementsByTagName("Date");
                            Element dateElement = (Element)dateList.item(0);
                            dateValue =
                            dateElement.getFirstChild().getNodeValue().trim();

                            NodeList idList =
                                 firstRecordElement.getElementsByTagName("ID");
                            Element idElement = (Element)idList.item(0);
                            idValue =
                            idElement.getFirstChild().getNodeValue().trim();

                            NodeList amountList =
                            firstRecordElement.getElementsByTagName("Amount");
                            Element amountElement =
                                      (Element)amountList.item(0);
                            amountValue =
                                 amountElement.getFirstChild().getNodeValue().
                                 trim();

                            if (idValue.equals("CASH")) {
                                   tempAmount += Double.valueOf(amountValue);
                                   }
                            else {
                                   Stock sk = new Stock(idValue);
                                   tempAmount +=
                                      Double.valueOf(amountValue)*
                                                 sk.getStockPrice(3, 0);
                                            }
                            }
                        }

                } catch( Exception e ) {
                        e.printStackTrace();
                }
              return tempAmount;
      }

      // profit and loss
      public double pl(){
              double tempAmount = 0.0;
          String file1 = new String("data/PORTFOLIO_BAK.xml");
              String file2 = new String("data/PORTFOLIO.xml");
              tempAmount = sumup(file2) - sumup(file1);
              tempAmount = Math.rint(100.0 * tempAmount) / 100.0;
              return tempAmount;
      }

}
```

Date.java

```
/*
```

```
 * Author: Jiahua Ni
 * File name: Date.java
 */
import java.io.*;

public class Date{

   double date;
   int year,month,day;

   Date(double inputdate){
     day=(int)inputdate%100;
     year=(int)inputdate/10000;
     month=(int)(inputdate-year*10000)/100;

     if(day>31){System.out.println("Day invalid");System.exit(0);}
     switch(month){
            case 1: {date=day;break;}
            case 2: {date=day+31;break;}
            case 3: {if(year%4==0)date=day+60;else date=day+59;break;}
            case 4:{if(year%4==0)date=day+91;else date=day+90;break;}
            case 5:{if(year%4==0)date=day+121;else date=day+120;break;}
            case 6:{if(year%4==0)date=day+152;else date=day+151;break;}
            case 7:{if(year%4==0)date=day+182;else date=day+182;break;}
            case 8:{if(year%4==0)date=day+213;else date=day+212;break;}
            case 9:{if(year%4==0)date=day+244;else date=day+243;break;}
            case 10:{if(year%4==0)date=day+274;else date=day+273;break;}
            case 11:{if(year%4==0)date=day+305;else date=day+304;break;}
            case 12:{if(year%4==0)date=day+335;else date=day+334;break;}
            default:{System.out.println("Month invalid");System.exit(0);}
     }

     date=date+(year-1900)*365+(year/4-1900/4);

   }

   public double getunigadate(){
     return date;
   }

   public double getnormaldate(){
     return (year*10000+month*100+day);
   }
}
```

Scope.java

```
/*
 * Author: Yang Sha, Jiahua Ni
 * File name: Scope.java
 */
import java.util.Hashtable;
import java.util.Vector;

public class Scope {

     HashStack scope=null;
     Scope parent_env = null; //for activation records

     public Scope(){
            scope = null;
           parent_env= null;

     }

     public void enter_scope(){
           HashStack temp=scope;
           scope=new HashStack();
           scope.outer=temp;
     }

     public void leave_scope(){
           if(scope.get_outer()!=null)
                 scope.set_p_scope(scope.get_outer().get_p_scope());
     }

     public void add_or_modify(String s, double i){
           Double t;
           HashStack temp=scope;
           t=(Double)temp.get_variable(s);
           while(t==null && temp.outer != null)
           {
                 temp=temp.get_outer();
                 t=(Double)temp.get_variable(s);
           }
           if(t==null && temp.outer == null)
           {
                 t=new Double(i);

                 scope.add_or_modify_variable(s, t);
           }
           t=new Double(i);

           temp.add_or_modify_variable(s, t);
     }

     public void add_in_present_scope(String s, double i){
           Double t=new Double(i);

           scope.add_or_modify_variable(s, t);
     }

     public boolean check_in_present_scope(String s){
           if(scope.get_variable(s)!=null)
                 return true;
           else
                 return false;
     }

     public Double get_variable(String s){
           Double t;
           t=(Double)scope.get_variable(s);
           HashStack temp=scope.get_outer();
           while(t==null && temp != null)
           {
                 t=temp.get_variable(s);
                 temp=temp.get_outer();
           }
           return t;
     }
}

class HashStack
{
     Hashtable p_scope=null;
     HashStack outer=null;
     public HashStack(){
           p_scope=new Hashtable();
           outer=null;
     }

     public void add_or_modify_variable(String s, Double T){
           p_scope.put(s, T);
     }

     public Double get_variable(String s){
           Double t;
           t=(Double)p_scope.get(s);
           return t;
     }

     public Hashtable get_p_scope(){
           return p_scope;
     }

     public HashStack get_outer(){
           return outer;
     }

     public void set_p_scope(Hashtable h){
           p_scope=h;
     }

     public void set_outer(HashStack h){
           outer=h;
     }
}
```

FuncScope.java

```
/*
 * Author: Yang Sha, Jiahua Ni
 * File name: FuncScope.java
 */
import antlr.collections.AST;
import java.util.*;
import antlr.ASTFactory;

public class FuncScope{
      String args[];
      AST body;
      double returnVal;
      String name;
      static Hashtable func = new Hashtable();
      // variableStack is the stack storage for function arguments
      static Vector variableStack = new Vector();

      public FuncScope(String name, AST body, double return_value){
            this.name = name;
            this.body=body;
            this.returnVal=return_value;
      }

      public static void registerArgument(String argument){

            variableStack.addElement(argument);

      }

      public static void addReturnValue(Double return_value){
            variableStack.addElement(return_value);
      }

      public void registerArgumentList(){

            args = new String[variableStack.size()];
            for(int i=0;i<variableStack.size();i++)
```

```
                        args[i] = variableStack.elementAt(i).toString();
                variableStack.removeAllElements();

        }

        public static void setArgument(double v){
                variableStack.addElement(new Double(v));
        }

        public void checkArguments(Scope scope){
                if(variableStack.size()!=args.length){
                        // wrong arguments
                        System.out.println("Incorrect number of arguments");
                        System.exit(0);
                }

                for(int i=0;i<args.length;i++){
                        //double check the arguments.
                        //some_function(double a, ..., double a) is incorrect
                        if(scope.check_in_present_scope(args[i])==true){
                                System.out.println("Double use of argument "+args[i]);
                                System.exit(0);
                        }
                        double value =
                            ((Double)variableStack.elementAt(i)).doubleValue();
                        scope.add_in_present_scope(args[i], value);
                        //now we need to clear the vector
                        variableStack.removeAllElements();
                }
        }

       public static void functionDefine(String name, AST body, double return_value)
throws
        antlr.RecognitionException{
                if(func.get(name)!=null){
                        System.out.println("function "+name+" already defined");
                        System.exit(1);
                }

                FuncScope s = new FuncScope(name, body, return_value);
                s.registerArgumentList();
                func.put(name, s);
        }

        public static double funcCall(SimpWalker walker, String name, Scope scope){
                double ret_value=0.0;
                try{
                        FuncScope fs = (FuncScope)func.get(name);
                        if(fs == null){
                        System.out.println("Function "+name+" not defined");
                        System.exit(1);
                        }

                        fs.checkArguments(scope);
                        walker.subprogram(fs.getBody());

                        Double temp=(Double)variableStack.elementAt(0);

                        //get the return value
                        //check if the return value match
                        if(fs.returnVal==0&&temp!=null || fs.returnVal==1&&temp==null)
                        {
                            System.out.println("Return type of the function "
                                                                +name +" does not match");
                             System.exit(1);
                        }

                        if(temp!=null)
                             ret_value=temp.doubleValue();
                        else
                             ret_value=0.0;
```

```
                    variableStack.removeAllElements();

             }
             catch(Exception e){
                    System.out.println("This is an exception");
             }
             return ret_value;
      }

      public AST getBody(){
             return body;
      }

}
```

GetRealData.java

```
/*
 * Author: Yang Sha
 * File name: GetRealData.java
 */
import java.io.IOException;
import java.io.*;
import java.util.*;
import java.io.InputStream;
import java.net.HttpURLConnection;
import java.net.URL;
import java.net.URLEncoder;
import java.lang.*;
import java.util.Vector;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.w3c.dom.*;

public class GetRealData {
    private static String symbol;
    private static String query="http://finance.yahoo.com/q?s=";
    private StringBuffer priceData;
    private char signal[]={'L','a','s','t',' ','T','r','a','d','e'};
    private char bold[] ={'<','b','>'};
    private int mark=0;
    double price;
    private static Document document_in, document_out;
    private boolean validating;
    private Vector<String> high = new Vector<String>();
    private Vector<String> low = new Vector<String>();
    private Vector<String> open = new Vector<String>();
    private Vector<String> close = new Vector<String>();
    private Vector<String> volume = new Vector<String>();
    private Vector<String> date = new Vector<String>();
    private String filename = "data/market/";

    public void setStockData() throws ParserConfigurationException{
     DocumentBuilderFactory factory=DocumentBuilderFactory.newInstance();
     DocumentBuilder builder=factory.newDocumentBuilder();
     document_out=builder.newDocument();

     Element root=document_out.createElement("Stock");
     document_out.appendChild(root);

     for(int i=0;i<date.size();i++){
         Element auction = document_out.createElement("Record");
         root.appendChild(auction);

         Element auctionType=document_out.createElement("Date");
```

```
        auctionType.
            appendChild(document_out.createTextNode((String)date.elementAt(i)));
        auction.appendChild(auctionType);

        Element itemPrice=document_out.createElement("Open");
       itemPrice.
             appendChild(document_out.createTextNode((String)open.elementAt(i)));
        auction.appendChild(itemPrice);

        Element itemID=document_out.createElement("High");
        itemID.
            appendChild(document_out.createTextNode((String)high.elementAt(i)));
        auction.appendChild(itemID);

        Element itemID4=document_out.createElement("Low");
        itemID4.
             appendChild(document_out.createTextNode((String)low.elementAt(i)));
        auction.appendChild(itemID4);

        Element itemID2=document_out.createElement("Close");
        itemID2.
            appendChild(document_out.createTextNode((String)close.elementAt(i)));
        auction.appendChild(itemID2);

        Element itemID3=document_out.createElement("Volume");
        itemID3.
            appendChild(document_out.createTextNode((String)volume.elementAt(i)));
        auction.appendChild(itemID3);

    }
    // save the xml file
    try{
        TransformerFactory tf=TransformerFactory.newInstance();
        Transformer transformer=tf.newTransformer();
        DOMSource source=new DOMSource(document_out);
        transformer.setOutputProperty(OutputKeys.ENCODING,"UTF-8");
        transformer.setOutputProperty(OutputKeys.INDENT,"yes");
        PrintWriter pw=new PrintWriter(new FileOutputStream(filename));
        StreamResult result=new StreamResult(pw);
        transformer.transform(source,result);
        pw.close();
    }
    catch(TransformerException mye){
        mye.printStackTrace();
    }
    catch(IOException exp){
        exp.printStackTrace();
    }

   }

   public boolean readStockData(){
    filename=filename+symbol+".xml";
    String myStr = new String();
    try{
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setValidating(validating);
        DocumentBuilder builder = factory.newDocumentBuilder();
        document_in = builder.parse(new File(filename));
        Element node = document_in.getDocumentElement();
        NodeList allList = node.getElementsByTagName("Record");
        // start to get the data
        for (int i = 0; i < allList.getLength(); i++) {
            Element list = (Element)allList.item(i);
            NodeList tlist = list.getElementsByTagName("Date");
            if(tlist.getLength()==1){
                Element e = (Element)tlist.item(0);
                myStr = e.getFirstChild().getNodeValue();

                date.add((String)myStr);
            }
            NodeList plist = list.getElementsByTagName("Open");
            if(plist.getLength()==1){
                Element e = (Element)plist.item(0);
                myStr = e.getFirstChild().getNodeValue();
                //System.out.println(myStr);
                open.add(myStr);
            }

            NodeList clist = list.getElementsByTagName("High");
            if(clist.getLength()==1){
                Element e = (Element)clist.item(0);
                myStr = e.getFirstChild().getNodeValue();
                //System.out.println(myStr);
                high.add(myStr);
            }

            NodeList list1 = list.getElementsByTagName("Low");
                if(list1.getLength()==1){
                    Element e = (Element)list1.item(0);
                    myStr = e.getFirstChild().getNodeValue();
                    //System.out.println(myStr);
                    low.add(myStr);
                }

            NodeList list2 = list.getElementsByTagName("Close");
                if(list2.getLength()==1){
                    Element e = (Element)list2.item(0);
                    myStr = e.getFirstChild().getNodeValue();
                    //System.out.println(myStr);
                    close.add(myStr);
                }

            NodeList list3 = list.getElementsByTagName("Volume");
                if(list3.getLength()==1){
                    Element e = (Element)list3.item(0);
                    myStr = e.getFirstChild().getNodeValue();
                    //System.out.println(myStr);
                    volume.add(myStr);
                }


        }


    }catch(Exception exp){
        exp.printStackTrace();
    return false; }
    return true;
   }

   public double displayItems(String s) throws IOException {
    symbol=s;
    URL url = new URL(query+symbol);
    priceData = new StringBuffer();

    HttpURLConnection httpConnection = (HttpURLConnection)url.openConnection();
    InputStream inputStream = httpConnection.getInputStream();
    int ch;
    boolean finished=false;
    boolean flag=false;

    while (!finished && ((ch=inputStream.read()) > 0)) {

        if(ch==signal[mark])
            mark++;
        else
            mark=0;
        if(mark==10){
            mark=0;
            //System.out.print((char)ch);
```

```
          while (!finished&&((ch=inputStream.read()) > 0)){
              if(flag){
                 // now start to fetch the stock data

                 if((char)ch=='<' | (char)ch==' '){
                     // stop to get the data
                     finished=true;
                     flag=false;
                  }
                 else{
                     if(ch=='.')
                        priceData.append('.');
                     else
                        priceData.append(ch-'0');
                  }
              }
              if(ch==bold[mark])
                 mark++;
              else
                 mark=0;
              if(mark==3){
                 mark=0;

                 flag=true;

              }
          }
       }
   }

   price=Double.valueOf(priceData.toString().trim());
   // output the price of retrieved from the document
   try{
       readStockData();

       // change the price
       // if price for current date is missing, create a new record

       int year = Calendar.getInstance().get(Calendar.YEAR);
       int month = Calendar.getInstance().get(Calendar.MONTH)+1;
       int day = Calendar.getInstance().get(Calendar.DAY_OF_MONTH);
       String currentDate = Integer.toString(month)
       +"/"+Integer.toString(day)+"/"+Integer.toString(year);
       if(date.size()>0 && currentDate.equals(date.lastElement())){
          // update the record

          high.setElementAt(new String(Double.toString(price+0.5)),high.size()-
1);
          low.setElementAt(Double.toString(price-0.5), low.size()-1);
          open.setElementAt(Double.toString(price-1), open.size()-1);
          close.setElementAt(Double.toString(price), close.size()-1);
          volume.setElementAt(Integer.toString(19820415), volume.size()-1);
       }
       else{
          // create a new record
          date.add(currentDate);
          high.add(new String(Double.toString(price+0.5)));
          low.add(Double.toString(price-0.5));
          open.add(Double.toString(price-1));
          close.add(Double.toString(price));
          volume.add(Integer.toString(19820415));
       }
       setStockData();
   }catch(Exception ex){}
   return price;
  }

  public static void main(String[] args) throws IOException {
   if(null==args | args.length<1)
       System.out.println("give me a symbol");
   GetRealData stock=new GetRealData();
```

```
     stock.displayItems(args[0].toString().trim());
    }
}
```

ActivationRecord.java

```
/*
 * Author: Yang Sha, Jiahua Ni
 * File name: ActivationRecord.java
 */
public class ActivationRecord
{
    public ActivationRecord(){
    }

    public static Scope create (Scope caller){
     Scope new_scope = new Scope();
     new_scope.parent_env = caller;
     return new_scope;
    }

    public static Scope remove (Scope present){
     Scope temp = present.parent_env;
     present = null;
     return temp;
    }
}
```

CommonASTWithLine.java

```
/*
 * Author: Jiahua Ni
 * File name: CommonASTWithLines.java
 */
import antlr.CommonAST;
import antlr.Token;

public class CommonASTWithLines extends CommonAST {

     private int line = 0;
     private int column = 0;
      private static final long serialVersionUID=1;

     public void initialize(Token tok) {
         super.initialize(tok);
         line=tok.getLine();
         column=tok.getColumn();
     }

     public int getLine() { return line; }

     public int getColumn() { return column; }
}
```

ErrorException.java

```
/*
 * Author: Leon Wu
 * File name: ErrorException.java
 * Handle error exception
 */
import java.text.*;
import java.util.*;
import java.util.Date;
import java.io.*;

public class ErrorException {


   public ErrorException() {
   }

   // print error message
   public String error(){

     String errormsg = "Error happened.\n";
     return errormsg;

   }

}
```

The programs followed are test cases. There are totally 36 test cases listed, 30 of which are for unit test, and the remaining 6 are regression test cases.

and.uniga

```
/*
*Author: Yu Song
*unit test case for the operator &
*/
main()
{
   double a=1, b=0;
   if a&b then{
     return 0;
   }
   else{
     return 1;
   }
}
```

assign.uniga

```
/*
```

```
*Author: Yu Song
*unit test case for operator =
*/
main()
{
    double a=1;
    if a==1 then{
      return 1;
    } else { return 0; }

}
```

average.uniga

```
/*
*Author: Yang Sha
*unit test case for built in function average()
*/
main(){
    double r;
    r=average("MSFT");
    println "the average of stock prices in history";
    println r;
}
```

builtinfunc.uniga

```
/*
*Author: Jian Pan
*unit test case for built in function open, close, high, low, and volume
*/
main(){
    double price;
    if high "MSFT" {2} > 3 then {
      print "high the msft price is larger than 3";
      print "The Open, Close, High, Low, Volume 2 days ago are:";
      print open "MSFT" {2};
      print close "MSFT" {2};
      print high "MSFT" {2};
      print low "MSFT" {2};
      print volume "MSFT" {2};
    }

    for(price=open "MSFT" {2};price>9;price=price-1){
      print "Bought MSFT with 1000 shares at";
      print price;
    }
}
```

buy.uniga

```
/*
*Author: Jian Pan
*unit test case for built in function buy
*/
main()
```

```
{
   buy "MSFT" 1000 0 28.5;
}
```

date.uniga

```
/*
*Author: Jian Pan
*unit test case for date transformation
*/
main(){
   double d1=date[20070404];
   double d2=date[20070330];
   print "The number of days between are:";
   println d1-d2;
}
```

division.uniga

```
/*
*Author: Yu Song
*unit test case for operator /
*/
main()
{
   double a=6, b=3;
   if a/b==2 then{
     return 1;
   } else { return 0; }

}
```

double.uniga

```
/*
*Author: Yu Song
*unit test case for data type double
*/

main()
{
   double a=2.435;
   if a==2.435 then{
     return 1;
   } else {
     return 0;
   }
}
```

dudefine.uniga

```
/*
*Author: Yu Song
*unit test case for variable scope checking
```

```
*/
main(){
   double a;
   double a=date[20070405];
}
```

equal.uniga

```
/*
*Author: Yu Song
*unit test case for operator ==
*/
main()
{
   double a=2, b=2;
   if a==b then{
     return 1;
   } else { return 0; }

}
```

for.uniga

```
/*
*Author: Jian Pan
*unit test case for keyword “for”
*/

main()
{
   double a=0;
   double i;
   for (i=0;i<5;i=i+1) {
     a=a+i;
     println a;
     a=a;
   }
   if a==10 then{
     return 1;
   } else {
     return 0;
   }
}
```

while.uniga

```
/*
*Author: Yu Song
*unit test case for keyword “while”
*/

main()
{
   double a,i=0;
   while (i<5) {
     double t=0;
```

```
      a=a+i;
      i=i+1;
      println i;
    }
    println t;
    if a==10 then{
      return 1;
    } else {
      return 0;
    }
}
```

functions.uniga

```
/*
*Author: Yang Sha
*unit test case for user defined function
*/

double getMSFTPrice(){
  double r = market “MSFT”;
  return r;
}
void printPrice(double r){
  println "in print price. The price is ";
  println r;
  return;
}
main()
{
  double r;
  r= getMSFTPrice();
  printPrice(r);
}
```

greater.uniga

```
/*
*Author: Yu Song
*unit test case for operator >
*/

main()
{
    double a=2, b=1;
    if a>b then{
      return 1;
    } else { return 0; }

}
```

holding.uniga

```
/*
*Author: Jian Pan
```

```
*unit test case for built in function “holdings”
*/

main(){
   double amt=holdingStockType();
   println amt;
   holdings;

}
```

if.uniga

```
/*
*Author: Yu Song
*unit test case for keyword “if”
*/
main()
{
   if 1>0 then {
     return 1;
   } else {
     return 0;
   }
}
```

less.uniga

```
/*
*Author: Yu Song
*unit test case for operator “<”
*/

main()
{
   double a=0, b=1;
   if a<b then{
     return 1;
   } else { return 0; }

}
```

market.uniga

```
/*
*Author: Jian Pan
*unit test case for built in function “market”
*/
main(){
  print "the market price for Microsoft is $";
  double r=market "MSFT";
  println r;
}
```

minus.uniga

```
/*
*Author: Yu Song
*unit test case for operator "-"
*/

main()
{
   double a=3, b=1;
   if a-b==2 then{
     return 1;
   } else { return 0; }

}
```

multiply.uniga

```
/*
*Author: Yu Song
*unit test case for operator "*"
*/

main()
{
   double a=2, b=3;
   if a*b==6 then{
     return 1;
   } else { return 0; }

}
```

or.uniga

```
/*
*Author: Yu Song
*unit test case for operator "|"
*/

main()
{
   double a=1, b=0;
   if a|b then{
     return 1;
   } else { return 0; }

}
```

pl.uniga

```
/*
*Author: Jian Pan
*unit test case for built in function "pl"
*/
main(){
   double r;
   r=pl();
   println r;
}
```

plus.uniga

```
/*
*Author: Jian Pan
*unit test case for operator "+"
*/

main()
{
   double a=1, b=0;
   if a+b==1 then{
     print "1";
   } else { print "0"; }

}
```

portfolioSum.uniga

```
/*
*Author: Jian Pan
*unit test case for built in function sum()
*/

main(){
   double r;
   r=sum();
   println "the sum of portfolios";
   println r;
}
```

recursion.uniga

```
/*
*Author: Jian Pan
*unit test case for function recursion
*/

double recurse(double r){
  if r<0 then return 0;
  print r;
  return recurse(r-1);
}

main(){
  recurse(10);
}
```

reserved.uniga

```
/*
*Author: Yang Sha
*unit test case for scopes. The reserved keywords cannot be redefined by users.
*/

void pl(){
     println "in pl";
     return;
}
void average(){
     println "in average";
     return;
}
void sum(){
     println "in sum";
     return;
}

main(){
     println("When user redefines the reserved keywords, an error should be
            popped up");
     average();
     pl();
     sum();
}
```

return.uniga

```
/*
*Author: Yu Song
*unit test case return statement.
*/

double testReturn(double i){
  return i+1;
}
main()
{
   double i=0;
   if testReturn(i)-i>0 then print "correct return";
   else print "incorrect return";
}
```

scope_1.uniga

```
/*
*Author: Jian Pan
*unit test case for scopes. Variables cannot be defined twice.
*/

main(){
     double a;
     double a=0;
}
```

scope_2.uniga

```
/*
*Author: Jian Pan
*unit test case for scopes. Variables should be defined before assignment.
*/


main(){
     double a=0;
     b=high "MSFT" {3};

}
```

sell.uniga

```
/*
*Author: Yu Song
*unit test case for built in function “sell”
*/

main()
{
   println "sell 1000 shares Microsoft without setting stop and limit prices";
    sell "MSFT" 1000 0 0;
   println "sell 500 shares Microsoft with stop price 28.5 and limit price 28.8";
    sell "MSFT" 500 28.5 28.8;
}
```

forinwhile.uniga

```
/*
*Author: jian pan
*regression test case for keyword “while” and built in function “buy”
*/

main()
{
    double j,i=0;

    while (i<5){
      for(j=0;j<5;j=j+1)
      {if j*2==0 then
             {println "intialize: j=0";}
       else
             {println j;}
      }
      i=i+1;
    }

}
```

forandbuy.uniga

```
/*
```

```
*Author: Yang Sha
*regression test case for keyword "for" and built in function "buy"
*/
main()
{  double stopprice=0, limitprice=40;
   double i;
   for(i=0;i<16;i=i+5){
     stopprice=stopprice+i;
     buy "MSFT" 1000 stopprice limitprice;
   }


}
```

whileandopen.uniga

```
/*
*Author: Yu Song
*regression test case for keyword "while" and built in function "open"
*/

main(){
double price=open "MSFT" {2};
while(price >9){
   print "Bought MSFT with 1000 shares at ";
   print price;
        price = price-1;
   }
}
```

whileandsell.uniga

```
/*
*Author: Yu Song
*regression test case for keyword "while" and built in function "sell"
*/

main()
{  double stopprice=30,limitprice=50;
    while(limitprice>stopprice){
     sell "INTC" 500 stopprice limitprice;
     limitprice=limitprice-5;
   }
}
```

portfolio.uniga

```
/*
*Author: Yang Sha
*regression test case for buy, sell, sum(), and pl().
*/

main()
{
   double s;
   s = sum();
   print "Sum of portfolio: ";
   print s;
   println " ";
   println "----------------------";

   buy "MSFT" 500 10.00 0;
   buy "MSFT" 500 28.50 0;
   sell "INTC" 500 0 0;
   buy "HPQ" 500 0 0;

   s = sum();
   print "Sum of portfolio: ";
   print s;
   println " ";
   println "----------------------";

   double r;
   r = pl();
   print "Profit and loss: ";
   print r;
   println " ";
   println "----------------------";

}
```

Strategy_1.uniga

```
/*
*Author: Jiahua Ni
*Regression test case.  Implement an equity trading strategy.
*/

main()
{  double s;
   s = sum();
   print "Sum of portfolio: ";
   print s;
   println "----------------------";

   double five_high;
   five_high=high "MSFT" {5} + high "MSFT" {4} +
                   high "MSFT" {3} + high "MSFT" {2} + high "MSFT" {1};
   five_high=five_high/5;

   double five_low;
   five_low=low "MSFT" {5} + low "MSFT" {4} +
                   low "MSFT" {3} + low "MSFT" {2} + low "MSFT" {1};
   five_low=five_low/5;

   double five_average;
   five_average=five_high+five_low;
   five_average=five_average/2;

   double tenfive_high=high "MSFT" {10} + high "MSFT" {9} +
                   high "MSFT" {8} + high "MSFT" {7} + high "MSFT" {6};
   tenfive_high=tenfive_high/5;
   double tenfive_low=low "MSFT" {10} + low "MSFT" {9} +
                   low "MSFT" {8} + low "MSFT" {7} + low "MSFT" {6};
   tenfive_low=tenfive_low/5;
   double ten_average=tenfive_high+tenfive_low;
```

```
ten_average=ten_average/10;
ten_average=ten_average+five_average;
ten_average=ten_average/2;

double current_price=market "MSFT";
double limit,stop;
double buyshare=0,sellshare=0;

while(current_price > five_average & buyshare < 1900){
  if open "MSFT" {0}>five_average then {
         limit= open "MSFT" {0};
         buy "MSFT" 500 five_low limit;

  }
  else {
         stop=open "MSFT" {0};
         buy "MSFT" 500 stop five_high;
  }
  buyshare=buyshare+500;

  current_price=market "MSFT";
}


if current_price < five_average then {
   if current_price < ten_average then {
        sell "MSFT" 1000 current_price five_high;
   }
   else {
        sell "MSFT" 500 ten_average five_high;
   }
}

double delta_average,delta;
if open "MSFT" {0} - close "MSFT" {1} > 0 then{
   delta=open "MSFT" {0} - close "MSFT" {1};
}
else {
   delta=close "MSFT" {1} - open "MSFT" {0};
}

if ten_average > five_average then {
   delta_average=ten_average-five_average;
   stop=open "MSFT" {0} - delta_average;
   limit= market "MSFT" + delta;
   sell "MSFT" 1000 stop limit;
}
else {
   delta_average=five_average-ten_average;
   stop=open "MSFT" {0} - delta;
   limit= market "MSFT" + delta_average;
   buy "MSFT" 1000 stop limit;
}


s = sum();
print "Sum of portfolio: ";
print s;
println " ";
println "----------------------";

double r;
r = pl();
```

```
   print "Profit and loss: ";
   print r;
   println " ";
   println "----------------------";


}
```

The programs followed are XML files (storing user transaction records, current portfolio holdings, and historical stock prices). They are inside the folder /data

ORDERS.xml

```
/*
*Author: Leon Wu
*Stores the historical transaction records
*/

<?xml version="1.0" encoding="UTF-8"?>
<Orders>
	<Record>
		<Date>4/16/2007</Date>
		<Type>buy</Type>
		<ID>MSFT</ID>
		<Amount>500</Amount>
		<Stop>0</Stop>
		<Limit>0</Limit>
		<FilledStatus>1</FilledStatus>
		<FilledPrice>28.48</FilledPrice>
	</Record>
	<Record>
		<Date>4/16/2007</Date>
		<Type>sell</Type>
		<ID>INTC</ID>
		<Amount>500</Amount>
		<Stop>0</Stop>
		<Limit>0</Limit>
		<FilledStatus>1</FilledStatus>
		<FilledPrice>20.56</FilledPrice>
	</Record>
	<Record>
		<Date>4/17/2007</Date>
		<Type>buy</Type>
		<ID>MSFT</ID>
		<Amount>500.0</Amount>
		<Stop>10.0</Stop>
		<Limit>0.0</Limit>
		<FilledStatus>0</FilledStatus>
		<FilledPrice>0.0</FilledPrice>
	</Record>

…
</Orders>
```

PORTFOLIO.xml

```
/*
*Author: Leon Wu
*Stores portfolio holdings
*/

<?xml version="1.0" encoding="UTF-8"?>
<Portfolio>
	<Record>
		<Date>5/7/2007</Date>
		<ID>CASH</ID>
		<Amount>-198425.0</Amount>
	</Record>
	<Record>
		<Date>5/7/2007</Date>
		<ID>MSFT</ID>
		<Amount>22500.0</Amount>
	</Record>
	<Record>
		<Date>5/7/2007</Date>
		<ID>INTC</ID>
		<Amount>-17500.0</Amount>
	</Record>
	<Record>
		<Date>5/7/2007</Date>
		<ID>HPQ</ID>
		<Amount>19500.0</Amount>
	</Record>
	<Record>
		<Date>5/7/2007</Date>
		<ID>ORCL</ID>
		<Amount>1000.0</Amount>
	</Record>
</Portfolio>
```

MSFT.xml

```
/*
*Author: Yang Sha
*Stores historical stock data for Microsoft (symbol ID "MSFT")
*/

<?xml version="1.0" encoding="UTF-8"?>
<Stock>
	<Record>
		<Date>4/19/2007</Date>
		<Open>28.60</Open>
		<High>28.75</High>
		<Low>28.21</Low>
		<Close>28.73</Close>
		<Volume>30740100</Volume>
	</Record>
	<Record>
		<Date>4/20/2007</Date>
		<Open>29.60</Open>
		<High>27.75</High>
		<Low>26.21</Low>
		<Close>28.73</Close>
		<Volume>30740100</Volume>
	</Record>
	<Record>
		<Date>4/21/2007</Date>
		<Open>26.60</Open>
		<High>27.75</High>
		<Low>26.21</Low>
```

```
        <Close>28.73</Close>
        <Volume>30740100</Volume>
</Record>
<Record>
        <Date>4/22/2007</Date>
        <Open>28.60</Open>
        <High>28.75</High>
        <Low>28.21</Low>
        <Close>28.73</Close>
        <Volume>30740100</Volume>
</Record>
<Record>
        <Date>4/23/2007</Date>
        <Open>28.60</Open>
        <High>28.75</High>
        <Low>28.21</Low>
        <Close>28.73</Close>
        <Volume>3014320</Volume>
</Record>
<Record>
        <Date>4/24/2007</Date>
        <Open>28.50</Open>
        <High>29.75</High>
        <Low>27.21</Low>
        <Close>28.63</Close>
        <Volume>30553430</Volume>
</Record>
<Record>
        <Date>4/25/2007</Date>
        <Open>27.60</Open>
        <High>28.75</High>
        <Low>28.21</Low>
        <Close>28.73</Close>
        <Volume>3655543</Volume>
</Record>
<Record>
        <Date>4/26/2007</Date>
        <Open>28.12</Open>
        <High>29.62</High>
        <Low>28.62</Low>
        <Close>29.12</Close>
        <Volume>19811221</Volume>
</Record>
<Record>
        <Date>5/2/2007</Date>
        <Open>29.61</Open>
        <High>31.11</High>
        <Low>30.11</Low>
        <Close>30.61</Close>
        <Volume>55041445</Volume>
</Record>
<Record>
        <Date>5/3/2007</Date>
        <Open>29.61</Open>
        <High>31.11</High>
```

```
                <Low>30.11</Low>
                <Close>30.61</Close>
                <Volume>48234235</Volume>
        </Record>
        <Record>
                <Date>5/4/2007</Date>
                <Open>29.56</Open>
                <High>31.06</High>
                <Low>30.06</Low>
                <Close>30.56</Close>
                <Volume>347703415</Volume>
        </Record>
        <Record>
                <Date>5/6/2007</Date>
                <Open>29.56</Open>
                <High>31.06</High>
                <Low>30.06</Low>
                <Close>30.56</Close>
                <Volume>25223415</Volume>
        </Record>
        <Record>
                <Date>5/7/2007</Date>
                <Open>29.61</Open>
                <High>31.11</High>
                <Low>30.11</Low>
                <Close>30.61</Close>
                <Volume>3631215</Volume>
        </Record>
</Stock>
```