Stock Trading Language
STL

Rekha Duthulur
Matt (Yu-Ming) Chang
Rui Hu
Nikhil Jhawar
Thomas Lippincott

{rd2241, yc2345, rh2333, njj2104, tml2115} @columbia.edu

INDEX

1	Introduction	pg 03
2	Language Tutorial	pg 04
3	Language Manual	pg 06
4	Project Plan	pg 12
5	Architectural Design	pg 17
6	Test Plan	pg 20
7	Lessons Learned	pg 27
8	Appendix	pg 29

1. Introduction

STL is an interpreted computer language designed to facilitate the rapid prototyping of stock trading strategies. It is aimed at individuals in the financial industry, with little or no formal background in computer science. It therefore strives to use the industry's terminology whenever possible, and implement only the most useful and intuitive control structures. With the limited scope of financial simulation, defining "the right thing to do" in such situations is less problematic than for a general-purpose programming language.

The **goal** of the Stock Trading Language is to enable a stock trader to write code to trade in the stock market to automate his buying/selling requirements.

Ease-of-use is the key aspect of the language. The language is really user friendly in the sense that traders can write simple buy/sell codes using simple words which they use in their financial dialect.

The language is also very **portable** as the user written code is converted into java code thus enabling it to be compatible with any machine running the Java Virtual Machine.

The language brings in a lot of **automation** for a trader who would have to otherwise sit in front of the computer for the whole time of the trading session. Now, the trader can simply write a program which will buy/sell certain quantity of stock(s) when the price of the stock reaches a particular level

2. Language Tutorial

STL has three datatypes, each with a corresponding character: numbers ($), strings (%), and lists of strings (@). These characters are important because they begin any variable or function of that type/return type:

```
function $double($x){ return(2*$x); }
$y = 2;
//$y and $double(NUMBER) are interchangable in terms of type,
//and this is reflected visually
```

There is no special syntax for variable declaration: each possible identifier has either been assigned a value or not. Consequently, the only local variables are those within functions.

The only mechanism for accessing the contents of a list is the foreach keyword:

```
foreach %item(@list){
        <CODE>
}
```

STL accesses the values and deltas of stocks in real time. Furthermore, each time the interpreter is run, it creates two internal variables to track a dollar amount and a list of stock quantities (initially set to $0 and empty, respectively). These are altered by buy() and sell(), and may be queried directly with money() and stocks().

```
@my_stocks = stocks();
$my_money = money();
```

Furthermore, the money quantity may be directly set by providing an argument to money():

```
print(money(1000));
```

From these basic operations, the user can implement convenience functions that wrap this behavior:

```
function $pretty_print(@stocks){
        print("STOCK VALUE DELTA");
        foreach %i(@stocks){
                print(%i);
                print(" ");
                print(value(%i));
                print(" ");
                print(delta(%i));
                print("\n");
        }
}
//demonstrates the function declaration syntax and the foreach syntax
```

An important feature that makes the GUI an important development tool is that can maintain the money and portfolio values between interpreter runs. This will let you experiment with running different one-line commands, without having to set up these fields each time. By consulting the language documentation, you can quickly capture simple behavior:

```
function $sell_high(@stocks,$count){
        $high = 0;
        foreach %i(@stocks){
                $temp = value(%i);
                if($temp>$high){
                        $high = $temp;
                        %target = %i;
                }
        }
        buy($count,%high);
}
```

By constructing your own libraries, it will quickly become possible to simulate comprehensive trading strategies in real-time. Remember, the wait() function is useful to spread out queries over longer periods of time:

```
function $watch(%stock,$target){
        $cur = 0;
        while($cur<$target){
                wait(10);
                $cur = value(%stock);
        }
        buy(money()/$cur,%stock);
        //buy as many as possible!
}
```

3. Language Reference Manual

Notation

The manual indicates syntactic categories with *italic* script and keywords with **bold** typeface. When there are several options for an item, they are separated by pipes (e.g. A|B|C).

1. Lexical Conventions

1.1 Comments

Like the C convention, comments begin with "/*" and end with "*/" and may continue over multiple lines. "//" is single line comment

1.2 Identifiers

Only three variable types are permitted in STL, each with a distinctive first character that clearly identifies the type at a glance:

1.21 Numerical quantities may be stored in variables starting with a dollar ($) sign,

1.22 Strings (typically stock names, in this context) may be stored in variables starting with a percentage (%) sign,

1.23 Lists may be stored in variables starting with an "at" (@) sign.

1.3 Keywords

The following words are reserved (see section 8 for definitions):

buy	**sell**
wma	**foreach**
while	**if**
else	**elsif**
wait	**function**
return	**print**
delta	

1.4 Constants

1.41 A number is specified as one or more digits, with an optional decimal portion. A decimal portion must be preceded with a number (so "0.012", not ".012").

1.42 A string is specified as any combination of ASCII characters enclosed within double quotes ("). To include a quotation mark in the string, it must be escaped with a preceding backslash (\). The string literal terminates with the first non-escaped matching quotation mark.

1.43 Lists are specified as comma-separated elements between straight brackets ([ ]). The elements must be strings and string variables.

1.5 Whitespace

Whitespace is ignored by the STL compiler. The purpose of whitespace is to separate out different tokens and allow users to follow the code in an easier fashion. Whitespace includes indentations, tabs, spaces and line terminators (including support for DOS, UNIX and MAC standards).

1.6 Separators

The following characters are used in STL as separators:
{} – code block enclosure
() – parameter list enclosure
[] - list enclosure
; - statement delimiter

2. Expressions

The following section discusses expressions used in STL. Expression operators are listed in order of highest to lowest level of precedence. Operators in the same section are considered to have equal precedence and evaluate from left to right. Using any operator on a type without an implicit cast defined (i.e. multiplication on two strings) will result in a compile time error.

2.1 Primary Expressions

A primary expression is the simplest form of an expression and is typically used to represent a single value. Primary expressions include identifiers, constants and function calls. Furthermore, parenthesized expressions are considered to be primary expressions. Primary expressions are of highest precedence in STL

2.2 Unary Expressions

The STL compiler allows two unary operators which are the NOT operator and the unary MINUS operator. The MINUS operator sets the value of a number to negative. The result of a unary NOT expression is the logical negation of the expression. Thus, the negation returns a one when the expression returns a zero and returns a zero when the expression is non-zero.

2.3 Multiplicative Operators

The two multiplicative operators: * for multiplication, / for division can be used to perform multiplicative operations. They are grouped left to right and can be applied to numerical constants.

2.4 Additive Operators

The two additive operators, + for addition and – for subtraction, can be used to perform additive operations. They are grouped left to right and can be applied to numerical data types.

2.5 Assignment Operators

The assignment operator, "=", stores the value returned by the expression on its right side into the identifier on its left side. It has the lowest precedence, and is associative from right to left. This allows multiple assignments to be made on the same line, in the following manner:

identifier1 = identifier2 = identifier3 = expression;

In this statement, the low precedence and right to left associatively of the assignment operator ensures that the entire expression is evaluated first. Next, the value of the expression is assigned to identifier3, then identifier2, then identifier1.

2.6 Relational and Equality Expressions

The following table summarizes the 6 relational operators available.

Operator	Description
>	Greater Than
<	Less Than
>=	Greater Than or Equal To
<=	Less Than or Equal To
==	Equal To
<>	Not Equal To

Relational and equality expressions are constructed as follows:

expression operator expression

Relational expressions return either zero or one, indicating whether the expression was false or true, respectively.

2.7 Logical Expressions

There are three logical operators available in STL. The logical AND operator, takes higher precedence than the logical inclusive OR operator. These operators are generally applied to relational and equality expressions, and other logical expressions.

The values are evaluated, and the logical expression returns true or false (1 or 0, respectively). Logical expressions may be grouped with parenthesis like any other binary operation. The exclusive OR operator is not defined. Logical expressions treat zero as false, and non-zero values as true.

3. Operator Precedence

The following table shows the precedence of operators in STL from highest to lowest:

Operators	Associativity
(*expression*)	Left to Right
NOT	Right to Left
*, /	Left to Right
+, -	Left to Right
<, <=, >=, >, ==, <>	Left to Right
AND	Left to Right
OR	Left to Right
=	Right to Left

4. Function Declarations

User defined function has the following format:

function [$|%|@]function_id (parameters)
{
statement-list
}

The character starting the function name indicates the return type. A function may only have return statements of this type: upon terminating without a return value, a function returns its type's default value (0, "" and [] respectively). The distinction between function and variable names is that the former always terminate with parentheses enclosing the function arguments, while the latter are strictly alphanumeric.

5. Statements

Except as indicated, statements are executed in sequence

5.1 Expression statement

Most statements are expression statements, which have the form

expression ;

5.2 Block

{*statement-list*}

5.3 Conditional statement

if (*expression*) { *statement* }
if (*expression*) { *statement* } **else** { *statement* }
if (*expression*) { *statement* } **elsif** { *statement* } **else** { *statement* }

In both cases the expression is evaluated and if it is non-zero, the first sub statement is executed. In the second case the second sub statement is executed if the expression is 0. As usual the **else** ambiguity is resolved by connecting an **else** with the last encountered **else**less **if**

5.4 Continually execute *statement-list* until *expression* evaluates to false (0).

while(*expression*) { *statement* }

5.5 Iterate *statement-list* over the *list*, placing the current iteration's element in *identifier*.

foreach *identifier* (*list*) { *statement* }

5.6 Pause execution until *expression* evaluates to true (1).

wait(*expression*);

5.7 In a function scope, exit the function and returns the value that *expression* evaluates to. Outside of a function scope, terminate the current program.

return(*expression*);

6. Scope rules

Variables defined in a function have the function body as its scope. The scope for variable defined outside of a function extends from their definition through the end of the file in which they appear.

7. Other Statements

7.1 purchase *number* of stock *string*, or *number* of each stock in *list*

buy(*number,string|list***)**

7.2 sell *number* of stock *string*, or *number* of each stock in *list*

sell(*number,string|list***)**

7.3 print out arguments appropriately (string, digits, or list thereof)

print(*string|list|number***)**

8. Built-in Operations

8.1 return the **weighted market average** of stock *string*, or of all the stocks in *list*

wma(*string*|*list***)**

8.2 return the **price change** today of stock *string*, or of all stocks in *list*

delta(*string*|*list***)**

8.3 return the **current value** of a particular stock

value(*string***)**

8.4 set the amount of **money** you have

money(*int***)**

8.5 return the number of **stocks** you have for a particular stock

stocks(*string***)**

4. Project Plan

4.1 Process used for planning, specification, development and testing

It had been our constant endeavor to meet at least once a week, if not more to discuss upon how each member has been progressing on his/her task. Another reason to meet was to have a common ground for any issues and major bugs that had to be looked upon. We used to meet every Saturdays for a period of 2-3 hours approximately. In addition to this weekly meeting, we also used to discuss notes after class on Mondays and Wednesdays.

4.2 Programming style used for coding

While, building a compiler is a difficult task with due to the complexities involved we had to make sure that the code we wrote was legible and matched with programming standards.

The following is how we tried to implement a programming style into our compiler, though it was not enforced, we more or less tried to be in the purview of it.

4.2.1 Antlr coding style

- If an Antlr rule is short and contains only one choice without any action, then it will be written in one line.
- The colon ":" always starts at the ninth column unless the string in front of ":" is too long. In the one line mode, ";" is at the end of this line.
- Lexem (token) names are in upper cases and syntactic items (non-terminals) are in lower cases, which may contain the underscore "_".

4.2.2 Java coding style

Indentation and spacing:

- Indentation of each level is one tab.
- The left brace "{" occupies a full line and is at the same column as the first character of the previous text.
- The right brace "}" occupies a full line and at the same column of its corresponding "{".
- One space between arguments and their parentheses "(" ")", but no space between function name and "(".
- Add spaces between operators and operands for outer expressions.

- Use tabs not spaces for indentation.

- The then-part and else-part of an “if” statement must not be at the same line of \if" or “else”. Similar for “for” statement.

Names:

- Class names start with “Stl” followed by English words whose first character is capital.

- Variables are in lower cases, and words are separated by underscore “_”.

- Do not use long names for local variables. More concise, the better.

- Follow Java ‘javadoc’ standard for comments.

4.3 Project Timeline

Task	Date Completed
Brainstorming & Idea Generation	February 1,2007
White Paper	February 7,2007
Architectural Design	February 14,2007
Features Addition	February 21,2007
Grammar Design, Lexer	February 25,2007
Language Reference Manual	March 5,2007
Parser	March 21,2007
Walker, Interpreter	April 2,2007
Code Generation	April 18,2007
Demo With TA	April 19,2007
Complete Testing, Making Appropriate Changes & Documentation	May 05,2007

4.4 Identify roles and responsibilities of each team member

Person	Roles
Rui	Lexer & Parser
Matt, Rekha	AST, Interpreter, Code Generation
Thomas	Lead Tester, GUI, Bug Reporting
Nikhil	Program Management, Documentation, Testing

4.5 Software development environment used

Most of the programs are written in Java. The lexer and parser are written in Antlr, and will be translated to java code.

4.5.1 Java 1.5

We used the Java 1.5 environment for development purposes as it is currently the most widely used platform and all the team members had it installed on their machines.

4.5.2 Operating System

Team members used a mix of Linux and Windows as their operating system of choice.

4.5.3 Antlr

The language parser is written in Antlr, "a language tool that provides a framework for constructing recognizers, compilers, and translators from grammatical descriptions containing C++ or Java actions."

4.5.3 CVS

CVS is a well-known concurrent version system for developers. We kept our CVS repository on a Linux machine, and use SSH to check in/out our programs with secured network connections.

4.5.4 Bugzilla

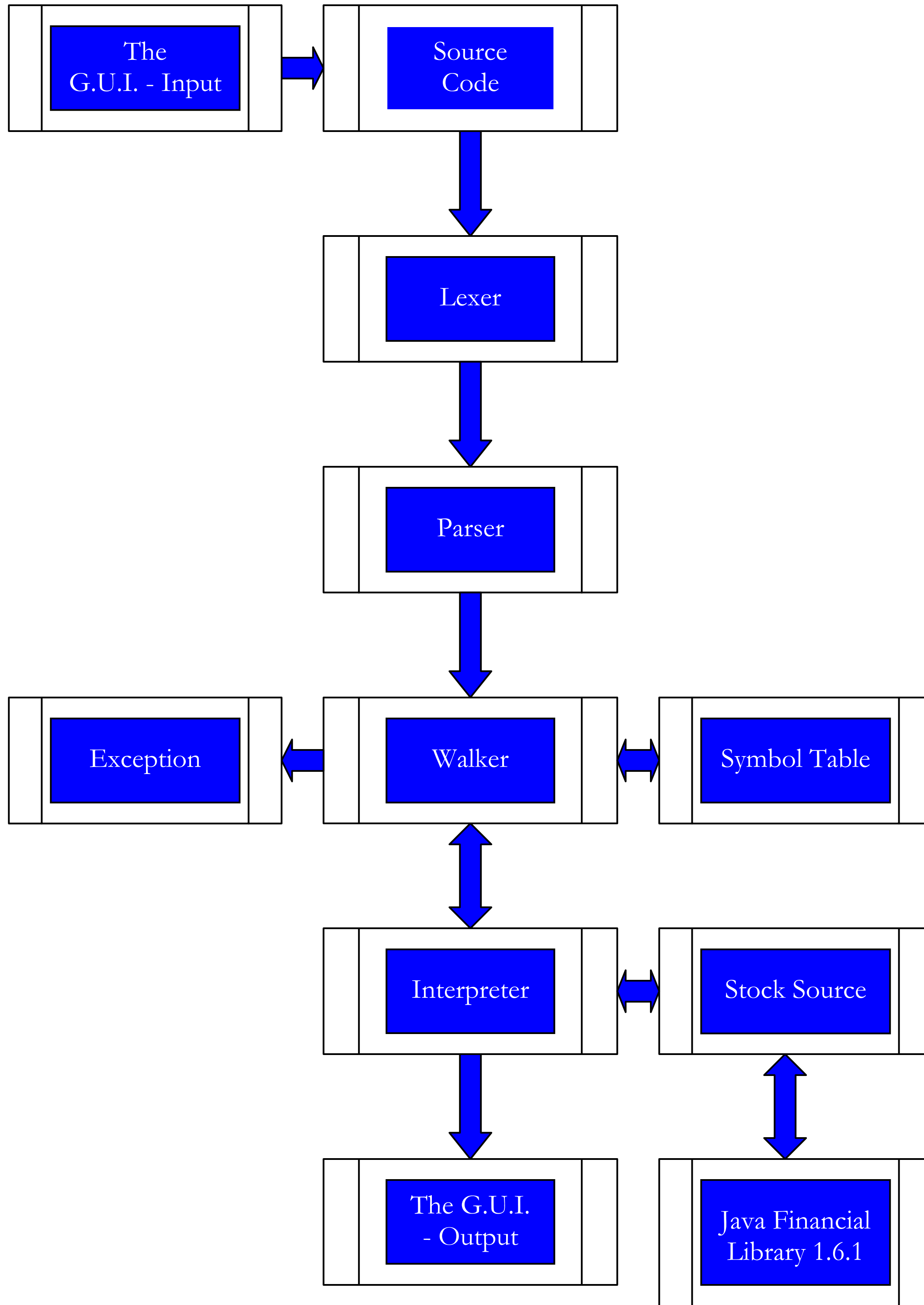
We have used Bugzilla for error reporting purposes which assigns error handling tasks to the members who caused them. Bugzilla is a "Defect Tracking System" or "Bug-Tracking System". Defect Tracking Systems allow individual or groups of developers to keep track of outstanding bugs in their product effectively.

4.6 Project Log

Date Completed	Task
31 January,2007	First Team Project Meeting
02 February,2007	Stock Trading Language Evolves
07 February,2007	White Paper completed
11 February,2007	Code conventions formed
14 February,2007	Lexer started
18 February,2007	Parser started
21 February,2007	Walker started
01 March,2007	Interpreter over Compiler chosen
05 March,2007	Language Reference Manual completed
20 March,2007	Graphical User Interface started
25 March,2007	Testing of Java and STL code started
April 5,2007	Bugzilla(for error reporting) implemented
April 10,2007	Lexer/Parser completed
April 19,2007	Simple demo with TA
April 26,2007	Walker finalized
April 29,2007	Graphical User Interface completed
May 5,2007	Testing of STL functionality completed
May 6,2007	Final project completed
May 7,2007	Project presented

5. Architectural Design

5.1 Block diagram showing the working of the Stock Trading Language



5.2 Interfaces between components

The Stock Trading Language consists of eight inter-related components: The GUI, Lexer, Parser, Walker, Interpreter, Exception, Symbol Table and Stock Source.

1. The source code is fed into the interpreter using the **GUI**. The **Lexer** then reads the code and translates a stream of characters into a stream of **tokens** and then outputs tokens to the **Parser**.
2. The Parser analyzes the syntactic structure of the program and translates the stream of tokens into an **Abstract Syntax Tree (A.S.T.).**
3. **Walker** reads the AST and checks whether the context constraints are obeyed, such as:
   - checking if operands are defined
   - checking that variables are defined
   - making sure operands are consistent with the operator
   - making sure booleans are the only thing being tested in "if" statements
   - other syntax errors the parser can not avoid
   - making sure function call arguments match arguments of function definitions
   - making sure a variable has not been redefined
4. The **Interpreter** reads the AST and executes the program. When the interpreter (at the backend) executes the program, there are three things to do:
   - Looks up the **symbol table** to find the corresponding functions and variables.
   - Executes the various financial functions such as buy(), sell(), delta() by interfacing with the **stockSource.java**.
   - The file stockSource.java interfaces with the Java Financial Library to fetch real-team stock quotes for use in our program.
   - Error handling and **exception** catching.

5.2.1 A note on Java Financial Library v 1.6.1.

The Java Financial Library is an API and application for converting between monetary currencies and fetching stock quotes from the Internet for use in customized Java programs as required. It requires a Java runtime environment for it to be able to work.

The JFL is a library and quick application for financial software. This software package was previously released as the Java Currency Converter, but now it covers stocks - hence the name change.

5.3 Who did what?

Person	Roles
Rui	Lexer & Parser
Matt, Rekha	AST, Interpreter, Code Generation
Thomas	Testing, GUI, Bug Reporting
Nikhil	Program Management, Documentation

6.1 Test Plan

Each person who coded a feature was responsible for testing that feature. After a feature was deemed workable on its own, we would run the formal tests for the new feature and all the old features. This regression testing method was used to make sure that no new features broke older, working features of the language.

We had dedicated two team people for the job of testing the whole language, Tom and Nikhil. The tests were laid out beforehand any substantial progress was made and the results that were produced were used as benchmark for the whole developmental lifecycle of the project.

The test suite was an automated script which ran through all the files ending with .stl in the src/test/ directory.

6.1.1 Show source code language converted to the target language

Since, we have developed an interpreter language; the source code is converted to java which is taken as input by the JFL library to fetch stock details from the internet.

As a result, we cannot show the target code for display.

6.1.2 Test Suite

arithmetic.stl: Reason – To test all the mathematical operators.

```
$i=100;

$j=50;

$mul=$i*$j;

$add=$i+$j;

$div=$i/$j;

$sub=$i-$j;

print($mul);

print("\n");

print($add);

print("\n");
```

```
print($div);

print("\n");

print($sub);

print("\n");
```

loops.stl: Reason – To check for the loops and test control flow

```
$counter = 2;

while($counter<10){

     print($counter);

     $counter = $counter + 1;
}

if($counter<1){

     print("less than 1\n");

}

else{

     print("greater\n");

}
```

buy.stl: Reason – Simple buy() functionality test

```
$i=money(10000);

$j=value("GOOG");

if($j<400){

     buy(10,"GOOG");

}

else{

     print("Stock too costly...");
```

```
}
```

sell.stl: Reason – To test a simple sell() functionality

```
$i=money(10000);

$j=value("MSFT");

if($j<400){

     buy(50,"MSFT");

     $k=stocks("MSFT");

     print("You currently own ");

     print($k);

     print(" stocks of Microsoft");

     print("\n");

}

else{

     print("Stock too costly...");

}

$l=stocks("MSFT");

sell(20,"MSFT");

$j=stocks("MSFT");

print("After selling You currently own ");

print($j);

print(" stocks of Microsoft");
```

comments.stl: Reason – To test the working of single and multi line comments

```
print("should see this - 1\n");
```

```
/*

print("shouldn't see this - 1");

*/

print("should see this - 2\n");

/*print("shouldn't see this - 2");*/
```

delta.stl: Reason- To test the delta() functionality. delta() returns the stock variance in a day

```
$i=money(1000);

@portfolio=["IBM","GOOG","YHOO","CECO","MSFT"];


print("Delta Values\n");

print("-------------------------\n");

foreach %iterate(@portfolio){


print(%iterate);

print(": ");

print(delta(%iterate));

print("\n");

}

$k = 0.5;

print("Stocks bought...\n");

print("-----------------\n");

foreach %read(@portfolio){

if(delta(%read)<0.5){
```

```
     buy(10,%read);

     print(%read);

     print(" bought\n");

}

else{


print(%read);

print(" too volatile to buy\n");

}
}
```

functions.stl: Reason – To test if calling a function works

```
function @join(%stockA,%stockB,%stockC){

@ret = [%stockA,%stockB,%stockC];

return @ret;

}

print(@join("IBM","GOOG","SUN"));
```

gcd.stl: Reason – To test for recursive functions

```
function $gcd($x,$y){
if($y<=0){
     return($x);
}
else{
     while($x  >= $y){
          $x = $x - $y;
     }
     return($gcd($y,$x));
}
}

print($gcd(33,9));
print("\n");
```

keywords.stl: Reason - To test for keywords

```
%stock = "IBM";

print("test print\n");

print(value(%stock));

$delta = delta(%stock);

$avg = wma(%stock);
```

listtest.stl: Reason – To test the list function

```
@a = [  "GOOG","MSFT"  ,"INFY","CECO","YHOO"];


$money=money(10000);

foreach %i(@a){


    $portfolio_item=value(%i);

    print("Currently traded price for ");

    print(%i);

    print(":");

    print($portfolio_item);

    print("\n");

    }

print("------------------------------------------------------------------------------------------\n");

print("Buying stocks only if their current traded price is less than 80$\n");

print("------------------------------------------------------------------------------------------\n");

print("Shares bought...\n");
```

```
print("--------------------------------------------------------
----------------------------------------\n");

foreach %j(@a){


    $portfolio_item=value(%j);

    $max=80;


    if($portfolio_item<$max){

         buy(10,%j);

         print(%j);

         print(":");

         print($portfolio_item);

         print("\n");

    }

}
```

7. Lessons Learned

Nikhil Jhawar

- One thing that I realized very early on in the project is that good communication and proper respect for deadlines would hold us in good stead through our project development.
- I also realized that having proper interfaces defined the various components on which different team members would work, would make our project much more modular and independent of each other.
- Also, it is necessary to have some sort of fun in team meetings to create a bond in the team; it releases pressure and motivates everyone to work in a more creative environment.
- A team meeting at least once a week helps in ironing out issues and working together in team meetings help develop pace in developmental cycle.
- Setting realistic goals is of utmost importance, as envisioning something really grandiose and then not being able to convert it into something real is as bad as not as doing anything.
- Also, deciding between an interpreter and a compiler for your language early at the start helps a lot.
- Having proper roles defined as per each member's strengths is a good way to divide work rather than randomly doing it.

Rui Hu

- Design phase is critical; many problems can be avoided if we really thought through them at design stage
- Experience does matter, we did not foresee some problems because we did not know they could possibly happen, when we realize, it is already too late to change them. But at least we know now and will avoid them next time

Rekha Duthulur

- Start early.
- Have fixed goals in mind, and fix the LRM early in the project.
- Perform tests on every module as it is developed.
- Scoping is difficult to implement, so allocate enough time to get it working.

Matt (Yu-Ming) Chang

- Working on a team project in eclipse using CVS was a great learning experience and taught me to work in teams.

- Splitting resources and then be able to co-operate is another good aspect that I gained from the project.
- The joy of developing my own language.
- Learned to how to develop a grammar and understand how the lexer and the parser work.
- The interpreter gave me whole new understanding of what goes inside a compiler.
- Adding a level of indirection is one of the best ways to solve a problem in programming.

Thomas Lippincott

- In the future, my first task on a project such as this will be to set up the infrastructure and have everyone use the simplest tools available. I feel that Eclipse is simply too bloated for a project like this: we had ongoing problems with configuration, CVS use, etc. This made it very difficult to tell whether people were making progress on their tasks.
- In terms of team management, it is very important to discover people's strengths and weaknesses early on. In a small group composed initially of strangers, working on an unfamiliar task, a little extra time determining this can avert catastrophes later on.
- As for the project itself, we certainly ended with less functionality in our language than originally intended: its most striking features, e.g. live stock quotes, were in fact the easiest parts to implement. I would have preferred to have used a more "academic" language for this project, SMLNJ or OCAML for instance, because I find Java has none of the conceptual challenges that force one to learn. I think a functional language like the examples I mentioned would force a deeper understanding of the theory.

7.1 Advice for future teams

- Get a start real early, helps in a lot of ways other than being on time.
- Have a good camaraderie between team-mates, helps resolve any discomfort.
- Set realistic goals; do not aim too high or too low.
- Divide responsibilities based on strengths of each member.
- At any give time, everyone should have a decent higher level feel of what is happening in terms of progress and where the project is heading. Helps in keeping a track of things.
- Set regular meetings with TA, their experiences would be able to help you give a different perspective of your project.
- Most importantly, deliver on your responsibilities!

8. Appendix

FILE NAME: src/stl_grammar.g

```
/*
  Lexer and Parser for STL
  STL Team
*/

class STLLexer extends Lexer;

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

protected
ALPHA   : 'a'..'z' | 'A'..'Z' | '_' ;

protected
DIGIT   :   '0'..'9';

WS   : (' ' | '\t')+ { $setType(Token.SKIP); }
        ;


NL      : ('\n' | ('\r' '\n') => '\r' '\n' | '\r')
           { $setType(Token.SKIP); newline(); }
          ;


COMMENT : ( "/*" (
                  options {greedy=false;} :
                  (NL)
                  | ~( '\n' | '\r' )
                )* "*/"
            | "//" (~( '\n' | '\r' ))* (NL)
          )                  { $setType(Token.SKIP); }
        ;

LPAREN  : '(';
RPAREN  : ')';
MULT    : '*';
PLUS    : '+';
MINUS   : '-';
```

```
RDV     : '/';
SEMI    : ';';
LBRACE  : '{';
RBRACE  : '}';
LBRK    : '[';
RBRK    : ']';
ASGN    : '=';
COMMA   : ',';
PLUSEQ  : "+=";
MINUSEQ : "-=";
MULTEQ  : "*=";
RDVEQ   : "/=";
GE      : ">=";
LE      : "<=";
GT      : '>';
LT      : '<';
EQ      : "==";
NEQ     : "<>";
COLON   : ':';


ID  options { testLiterals = true; }
        : ALPHA (ALPHA|DIGIT)*
        | ('$' | '%' | '@') ALPHA (ALPHA|DIGIT)*
        ;

NUMBER  : (DIGIT)+
          (
          '.'(DIGIT)*
          | /* empty */
          )
          ;


STRING  : '"'!
              (  ~('"' | '\n')
               | ('"'!'"')
              )*
          '"'!
        ;




LIST
   : '[' ( (' ' | '\t' | NL )*  (ID | STRING) (' ' | '\t' | NL )* )  ( COMMA   ( (' ' | '\t' | NL )*
 (ID | STRING) (' ' | '\t' | NL )* )  )*  ']'
```

```
    ;

class STLParser extends Parser;

options{
    k = 2;
    buildAST = true;
    exportVocab = STLAntlr;
    defaultErrorHandler = false;
}

//this is very important Antlr feature
tokens {
   STATEMENT;
   VAR_LIST;
   EXPR_LIST;
   FUNC_CALL;
   FOREACH_CON;
   LOOP;
   UPLUS;
   UMINUS;
}

program
        : ( statement | func_def )* EOF!
            {#program = #([STATEMENT,"PROG"], program); }
        ;

//add empty statement later
statement
        : foreach_stmt
         | if_stmt
         | return_stmt
         | assignment
         | while_stmt
         | buy_stmt
         | sell_stmt
         | print_stmt
         | wait_stmt
         | func_call_stmt
         | LBRACE! (statement)* RBRACE!
            {#statement = #([STATEMENT,"STATEMENT"], statement); }
        ;
```

```
foreach_stmt
	: "foreach"^ ID LPAREN! (LIST | ID )RPAREN! statement
	;



if_stmt
	: "if"^ LPAREN! expression RPAREN! statement
	   (options {greedy = true;}: "else"! statement )?
	;

return_stmt
	: "return"^ (expression)? SEMI!
	;

assignment
	: l_value ( ASGN^ | PLUSEQ^ | MINUSEQ^ | MULTEQ^
		) expression SEMI!
	;

while_stmt : "while"^ LPAREN! expression_while RPAREN! statement ;

buy_stmt : "buy"^ LPAREN! expr_list RPAREN! SEMI! ;

sell_stmt : "sell"^ LPAREN! expr_list RPAREN! SEMI! ;

print_stmt : "print"^ LPAREN! expression RPAREN! SEMI! ;

wait_stmt : "wait"^ LPAREN! expr_list RPAREN! SEMI! ;


func_call_stmt
	: func_call SEMI!
	;

func_call
	: ID LPAREN! expr_list RPAREN!
	   {#func_call = #([FUNC_CALL,"FUNC_CALL"], func_call); }
	;

expr_list
	: expression ( COMMA! expression )*
	   {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
	|
	   {#expr_list = #([EXPR_LIST,"EXPR_LIST"], expr_list); }
	;

func_def
```

```
        : "function"^ ID LPAREN! var_list RPAREN! func_body
        ;

var_list
        : ID ( COMMA! ID )*
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        |
            {#var_list = #([VAR_LIST,"VAR_LIST"], var_list); }
        ;

func_body
        : LBRACE! (statement)* RBRACE!
            {#func_body = #([STATEMENT,"FUNC_BODY"], func_body); }
        ;

expression_while
    : logic_factor ( ("and"^ | "or"^)  logic_factor )*
    ;

expression
        : logic_term ( "or"^ logic_term )*
        ;

logic_term
        : logic_factor ( "and"^ logic_factor )*
        ;

logic_factor
        : ("not"^)? relat_expr
        ;

relat_expr
        : arith_expr ( (GE^ | LE^ | GT^ | LT^ | EQ^ | NEQ^) arith_expr )?
        ;

arith_expr
        : arith_term ( (PLUS^ | MINUS^) arith_term )*
        ;

arith_term
        : arith_factor ( (MULT^| RDV^) arith_factor )*
        ;

arith_factor
        : PLUS! r_value
            {#arith_factor = #([UPLUS,"UPLUS"], arith_factor); }
        | MINUS! r_value
            {#arith_factor = #([UMINUS,"UMINUS"], arith_factor); }
```

```
        | r_value
              ;

r_value
        : l_value
        | func_call
        | built_in_operators
        | NUMBER
        | STRING
        | LIST
        | "true"
        | "false"
        | LPAREN! expression RPAREN!
        ;

l_value
        : ID
        ;

built_in_operators : wma_operator | value_operator | delta_operator | money_operator |
stocks_operator ;

wma_operator : "wma"^LPAREN! expr_list RPAREN!;

value_operator : "value"^LPAREN! expr_list RPAREN!;

delta_operator : "delta"^LPAREN! expr_list RPAREN!;

money_operator : "money"^LPAREN! expr_list RPAREN!;

stocks_operator : "stocks"^LPAREN! expr_list RPAREN!;


cl_statement
        : ( statement | func_def )
        | EOF!
            { System.exit(0); }
        ;



{
import java.io.*;
import java.util.*;
}

class STLWalker extends TreeParser;
{
```

```
    static STLDataType null_data = new STLDataType( "<NULL>" );
    STLInterpreter intpret = new STLInterpreter();
}

expr returns [ STLDataType r ]
{
    STLDataType a, b;
    STLDataType[] x;
    Vector v;
    String s = null;
    String[] sx;
    r = null_data;
}
        : #("or" a=expr right_or:.)
            {
                if ( a instanceof STLBool )
                    r = ( ((STLBool)a).var ? a : expr(#right_or) );
                else
                    r = a.OR( expr(#right_or) );
            }
        | #("and" a=expr right_and:.)
            {
                if ( a instanceof STLBool )
                    r = ( ((STLBool)a).var ? expr(#right_and) : a );
                else
                    r = a.AND( expr(#right_and) );
            }
        | #("not" a=expr)              { r = a.NOT(); }
        | #(GE a=expr b=expr)          { r = a.GE( b ); }
        | #(LE a=expr b=expr)          { r = a.LE( b ); }
        | #(GT a=expr b=expr)          { r = a.GT( b ); }
        | #(LT a=expr b=expr)          { r = a.LT( b ); }
        | #(EQ a=expr b=expr)          { r = a.EQ( b ); }
        | #(NEQ a=expr b=expr)         { r = a.NE( b ); }
        | #(PLUS a=expr b=expr)        { r = a.ADD( b ); }
        | #(MINUS a=expr b=expr)       { r = a.SUBTRACT( b ); }
        | #(MULT a=expr b=expr)        { r = a.MULT( b ); }
        | #(RDV a=expr b=expr)         { r = a.DIV( b ); }
        | #(ASGN a=expr b=expr)        { r = intpret.assign(a,b); }
        | num:NUMBER                   { r = intpret.getNumber( num.getText() ); }
        | str:STRING                   { r = new STLString( str.getText() ); }
        | list:LIST                    { r = intpret.getList( list.getText() ); }
        | "true"                       { r = new STLBool( true ); }
        | "false"                      { r = new STLBool( false ); }
        |#("print" a=expr)             { r = intpret.print(a);}
        |#("buy" x=mexpr)              { r = intpret.buy(x);}
        |#("sell" x=mexpr)             { r = intpret.sell(x);}
        |#("value" x=mexpr)            { r = intpret.value(x);}
```

```
|#("wma"  x=mexpr)            { r = intpret.wma(x);}
|#("delta" x=mexpr)          { r = intpret.delta(x);}
|#("money" x=mexpr)            { r = intpret.money(x);}
|#("stocks" x=mexpr)          { r = intpret.stocks(x);}
|#("wait" x=mexpr)          { r = intpret.wait(x);}
|#(UMINUS a=expr)            { r = a.uminus(); }
| #(id:ID                   { r = intpret.getVariable( id.getText() ); }

  )
| #(FUNC_CALL a=expr x=mexpr)
   { r = intpret.funcInvoke( this, a, x ); }

| #("while" x=wexpr whilebody:.)
   {
        if ( !(x[0] instanceof STLBool) )
              return x[0].syntaxError("while: expression should be bool");
        else
        {
              if(((STLNumber)x[3]).getNum() < 8)
              {
                    intpret.whileInit(x);

                    while ( intpret.whileCanProceed(x) )
                    {
                          r = expr(#whilebody);
                    }
              }
              else //(((STLNumber)x[3]).getNum() == 8)
              {
                    while (intpret.whileOrAndCanProceed(x))
                    {
                          r = expr(#whilebody);
                    }
                    intpret.whileOrAndOperator(0); //clear vOrAndRecord
              }
        }
   }

| #("foreach" a=expr b=expr foreachbody:.)
  {

             for(int i = 0; i < ((STLList)b).getVlist().size() ; i++)
             {
                   intpret.foreachProceed(a, b, i);
                   r = expr(#foreachbody);
             }
  }
```

```
    | #("if" a=expr thenp:. (elsep:.)?)
      {
        if ( !( a instanceof STLBool ) )
          return a.syntaxError( "if: expression should be bool" );
        if ( ((STLBool)a).var )
        {
          r = expr( #thenp );

        }
        else if ( null != elsep )
          r = expr( #elsep );
      }
    | #(STATEMENT (stmt:. { if ( intpret.canProceed() ) r = expr(#stmt); } )*)
    |  #("break" (breakid:ID      { s = breakid.getText(); }
            )?
      )                      { intpret.setBreak( s ); }
    | #("return" ( a=expr        { r = intpret.rvalue( a ); }
            )?
      )                      { intpret.setReturn( null ); }
    | #("function" fname:ID sx=vlist fbody:.)
      { intpret.funcRegister( fname.getText(), sx, #fbody ); }
    ;

mexpr returns [ STLDataType[] dt ]
{
  STLDataType a;
  dt = null;
  Vector v;
}
    : #(EXPR_LIST        { v = new Vector(); }
          ( a=expr      { v.add( a ); }
          )*
      )                 { dt = intpret.convertExprList( v ); }
    ;


vlist returns [ String[] sv ]
{
  Vector v;
  sv = null;
}
    : #(VAR_LIST     { v = new Vector(); }
         (s:ID      { v.add( s.getText() ); }
         )*
      )             { sv = intpret.convertVarList( v ); }
    ;

wexpr returns [ STLDataType[] x]
```

```
{
   STLDataType a, b;
   x = new STLDataType[4];
   Vector v;
   String s = null;
   String[] sx;
}

          : #("or" a=exprOrAndWhile b=exprOrAndWhile) {x[0]= a.OR(b); x[1]= new
STLBool(true); x[2]=new STLBool(true); x[3] = new STLNumber(8);
                                        intpret.whileOrAndOperator(1);}
       | #("and" a=exprOrAndWhile b=exprOrAndWhile) {x[0]=a.AND(b); x[1]=new
STLBool(true); x[2]=new STLBool(true); x[3] = new STLNumber(9);
                                       intpret.whileOrAndOperator(2);}
       | #(GE a=expr b=expr)        { x[0] = a.GE( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(0);}
       | #(LE a=expr b=expr)        { x[0] = a.LE( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(1);}
       | #(GT a=expr b=expr)        { x[0] = a.GT( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(2);}
       | #(LT a=expr b=expr)        { x[0] = a.LT( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(3);}
       | #(EQ a=expr b=expr)        { x[0] = a.EQ( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(4);}
       | #(NEQ a=expr b=expr)       { x[0] = a.NE( b ); x[1] = a; x[2] = b; x[3] = new
STLNumber(5);}
       | "true"                { x[0] = new STLBool( true ); x[1] = new STLBool(true); x[2] =
new STLBool(true);x[3] = new STLNumber(6);}
       | "false"               { x[0] = new STLBool( false );x[1] = new STLBool(false); x[2] =
new STLBool(false); x[3] = new STLNumber(7);}
       ;

exprOrAndWhile returns [STLDataType r]
{
       STLDataType a, b;
       r = null;
}
      : #(GE a=expr b=expr)        {r = a.GE( b ); intpret.whileOrAndInit(a, b);
intpret.whileOrAndRecord(a, b, 0);}
      | #(LE a=expr b=expr)        {r = a.LE( b ); intpret.whileOrAndInit(a, b);
intpret.whileOrAndRecord(a, b, 1);}
      | #(GT a=expr b=expr)        {r = a.GT( b ); intpret.whileOrAndInit(a, b);
intpret.whileOrAndRecord(a, b, 2);}
      | #(LT a=expr b=expr)        {r = a.LT( b ); intpret.whileOrAndInit(a, b);
intpret.whileOrAndRecord(a, b, 3);}
      | #(EQ a=expr b=expr)        {r = a.EQ( b ); intpret.whileOrAndInit(a, b);
intpret.whileOrAndRecord(a, b, 4);}
      | #(NEQ a=expr b=expr)       {r = a.NE( b ); intpret.whileOrAndInit(a, b);
```

```
intpret.whileOrAndRecord(a, b, 5);}
        | "true"                    {r = new STLBool( true ); intpret.whileOrAndRecord(new
STLBool(true), new STLBool(true), 6);}
        | "false"                   {r = new STLBool( false ); intpret.whileOrAndRecord(new
STLBool(false), new STLBool(false), 7);}
        | #("or" a=exprOrAndWhile b=exprOrAndWhile)  {intpret.whileOrAndOperator(1);r
= a.OR(b);}
        | #("and" a=exprOrAndWhile b=exprOrAndWhile) {intpret.whileOrAndOperator(2);r
= a.AND(b);}
        ;
```

FILE NAME: src/Main.java

```
/*
 * Simple front-end for an ANTLR lexer/parser that dumps the AST
 * textually and displays it graphically. Also calls the
 * treewalker. Good for debugging AST
 * construction rules.
 *
 */

import java.io.*;
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;

public class Main {
  public static void main(String args[]) {
    try {
      DataInputStream input = new DataInputStream(new FileInputStream(args[0]));

      // Create the lexer and parser and feed them the input
      PltLexer lexer = new PltLexer(input);
      PltParser parser = new PltParser(lexer);
      parser.program(); // "file" is the main rule in the parser

      // Get the AST from the parser
      CommonAST parseTree = (CommonAST)parser.getAST();

      // Print the AST in a human-readable format
      System.out.println(parseTree.toStringList());

      STLWalker walker = new STLWalker();
      STLDataType dt = walker.expr(parseTree);

      // Open a window in which the AST is displayed graphically
      ASTFrame frame = new ASTFrame("AST from the STL parser", parseTree);
      frame.setVisible(true);

    } catch(Exception e) { System.err.println("Exception: "+e); }
  }
}
```

FILE NAME: src/STLBool.java

```
import java.io.PrintWriter;

/**
 * @author Team STL
 * This class inherits from STLDataType, it
 * it responsible for all boolean operands and
 * operations.
 */
public class STLBool extends STLDataType{

    boolean var;

    /**
     * Contructor for the STLBool Class.
     * @param   truth: value to initialize var boolean variable
     * @return  none
     * @see STLDataType
     */
    public STLBool(boolean truth){
        var = truth;
    }


     /**
     * Returns a STLDataType object.The function returns the value
     * not(var).
     * @return STLDataType
     * @see STLDataType
     */
    public STLDataType NOT() {//Modified by Matt
        return new STLBool( !var );
    }


     /**
     * Returns a STLDataType object.The function returns the value
     * (dt AND var).
     * @return STLDataType, incase of error returns STLException
     * @see STLDataType
     */
    public STLDataType AND(STLDataType dt) {//Modified by Matt
    if ( dt instanceof STLBool )
      return new STLBool( var && ((STLBool)dt).var );
    return error( dt, "and" );
    }
```

```
 /**
  * Returns a STLDataType object.The function returns the value
  * (dt OR var).
  * @return STLDataType, incase of error returns STLException
  * @see STLDataType
  */
public STLDataType OR(STLDataType dt) {//Modified by Matt
if ( dt instanceof STLBool )
   return new STLBool( var || ((STLBool)dt).var );
return error( dt, "or" );
}

 /**
  * Returns an error because there no GE for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
public STLDataType GE(STLDataType dt) {
      return error( dt, ">=" );
}

 /**
  * Returns an error because there no GT for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
public STLDataType GT(STLDataType dt) {
      return error( dt, ">" );
}

/**
  * Returns an error because there no LE for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
public STLDataType LE(STLDataType dt) {
      return error( dt, "<=" );
}

/**
  * Returns an error because there no LT for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
```

```
public STLDataType LT(STLDataType dt) {
        return error( dt, "<" );
}

/**
  * Returns the value of checking if two boolean operands are equal.
  * operands.
  * @return STLException
  * @see STLDataType
  */
public STLDataType EQ(STLDataType dt) {//Modified by Matt
if ( dt instanceof STLBool )
    return new STLBool( ( var && ((STLBool)dt).var )
                    || ( !var && !((STLBool)dt).var ) );
return error( dt, "==" );
}

/**
  * Returns a STLDataType object. Returns true if the operands are
  * equal otherwise returns false. Incase of error in syntax returns
  * STLException
  * @return STLException in case of error
  *        boolean value
  * @see STLDataType
  */
public STLDataType NE(STLDataType dt) {//Modified by Matt
if ( dt instanceof STLBool )
    return new STLBool( ( var && !((STLBool)dt).var )
                    || ( !var && ((STLBool)dt).var ) );
return error( dt, "<>" );
}

/**
  * Returns an error because there no MULT for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
public STLDataType MULT(STLDataType dt) {
        return error( dt, "*" );
}

/**
  * Returns an error because there no DIV for boolean
  * operands.
  * @return STLException
  * @see STLDataType
  */
```

```
    public STLDataType DIV(STLDataType dt) {
          return error( dt, "/" );
    }

    /**
     * Returns an error because there no ADD for boolean
     * operands.
     * @return STLException
     * @see STLDataType
     */
    public STLDataType ADD(STLDataType dt) {
          return error( dt, "+" );
    }

    /**
     * Returns an error because there no SUBTRACT for boolean
     * operands.
     * @return STLException
     * @see STLDataType
     */
    public STLDataType SUBTRACT(STLDataType dt) {
          return error( dt, "-" );
    }

    /**
     * Returns String. Returns type of STLBoolean.
     * operands.
     * @return String "bool"
     * @see STLDataType
     */
    public String typeName(){
       return "bool";
}

    public STLDataType syntaxError(String str){
       //TODO
       return null;
}

     /**
     * The method prints out the information about the STLBool.
     * @return  none
     * @see STLDataType
     */
    public void print( PrintWriter w ) {
    if ( getName() != null )
       w.print( getName() + " = " );
    w.println( var ? "true" : "false" );
```

```
    }
}
```

FILE NAME: src/STLDataType.java

```
import java.io.PrintWriter;

/***
 * @author STL
 * The class is the base class for all the DataTypes.
 *
 */
public class  STLDataType {

	private String name; //The name of the variable

	//Constructor for STLDataType
	public STLDataType(){}

	//Constructor for STLDataType which takes string name arg
	public STLDataType(String name){
		this.name = name;
	}

	/**
	 * Returns an String. The function returns the name
	 * of a the STLDataType.
	 * @return  the name of the STLDataType
	 */
	public String getName() {
		return name;
	}

	/**
	 * Returns an String. The function sets the name
	 * of the STLDataType
	 * @return  none
	 */
	public void setName(String name) {
		this.name = name;
	}

	/**
	 * Returns a STLDataType object. Incase there is an error an
	 * STLException is thrown.
	 * @param: The message shown in the Exception.
	 * @return  STLDataType object
	 */
    public STLDataType error( String msg ) {
	throw new STLException( "illegal operation: " + msg
			+ "( <" + typeName() + "> "
```

```
                    + ( name != null ? name : "<?>" )
                    + " )" );
}

    /**
     * Returns a STLDataType object. Incase there is an error an
     * STLException is thrown.
     * Following Mx style error handling.
     * @param: b: STLDataType object
     * @param: msg: The msg shown in the Exception text.
     * @return  STLDataType object
     */
public STLDataType error( STLDataType b, String msg ) {
    if ( null == b )
        return error( msg );
    throw new STLException(
        "illegal operation: " + msg
        + "( <" + typeName() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + typeName() + "> "
        + ( name != null ? name : "<?>" )
        + " )" );
}

    /**
     * Returns a STLDataType object. Incase there is an error an
     * STLException is thrown.
     * Following Mx style error handling.
     * @param: b: STLDataType object
     * @param a: STLDataType obj
     * @param: msg: The msg shown in the Exception text.
     * @return  STLDataType object
     */
 public STLDataType error( STLDataType a,STLDataType b, String msg ) {
    if ( null == b  || null == a)
        return error( msg );
    throw new STLException(
        "illegal operation: " + msg
        + "( <" + a.typeName() + "> "
        + ( name != null ? name : "<?>" )
        + " and "
        + "<" + b.typeName() + "> "
        + ( b.name != null ? b.name : "<?>" )
        + " )" );
}

     /**
```

```
     * The function prints out information about the STLDataType object
     * @param: w: PrintWriter specifies where the output is sent.
     * @return  none
     */
public void print( PrintWriter w ) {
   if ( name != null )
      w.print( name + " = " );
   w.println( "<undefined>" );
}

    /**
     * The function prints out information about the STLDataType object
     * @return  none
     */
public void print() {
   print( new PrintWriter( System.out, true ) );
}

    /**
     * Returns a STLDataType object.The function returns an error incase a user
     * tries to perform operation NOT on a STLDataType Object.
     * @return  STLException
     * @see STLException, STLDataType
     */
   public STLDataType NOT(){
   return error( "not" );
   }

    /**
     * Returns a STLDataType object.The function returns an error incase a user
     * tries to perform operation AND on a STLDataType Object.
     * @param   dt: STLDataType object
     * @return  STLException
     * @see STLException, STLDataType
     */
   public STLDataType AND(STLDataType dt){
   return error( "and" );
   }

    /**
     * Returns a STLDataType object.The function returns an error incase a user
     * tries to perform operation OR on a STLDataType Object.
     * @param   dt: STLDataType object
     * @return  STLException
     * @see STLException, STLDataType
     */
   public STLDataType OR(STLDataType dt){
   return error( "or" );
```

```
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation GE on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public STLDataType GE(STLDataType dt){
return error( "ge" );
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation GT on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public STLDataType GT(STLDataType dt){
return error( "gt" );
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation LE on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public  STLDataType LE(STLDataType dt){
return error( "le" );
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation LT on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public  STLDataType LT(STLDataType dt){
return error( "lt" );
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
```

```
  * tries to perform operation EQ on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
 public  STLDataType EQ(STLDataType dt){
 return error( "eq" );
 }

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation NE on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public  STLDataType NE(STLDataType dt){
 return error( "ne" );
 }

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation ADD on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
 public  STLDataType ADD(STLDataType dt){
 return error( "add" );
 }

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation SUBTRACT on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
 public  STLDataType SUBTRACT(STLDataType dt){
 return error( "subtract" );
 }

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation MULT on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
```

```
 */
public  STLDataType MULT(STLDataType dt){
return error( "mult" );
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation DIV on a STLDataType Object.
  * @param   dt: STLDataType object
  * @return  STLException
  * @see STLException, STLDataType
  */
public  STLDataType DIV(STLDataType dt){
   return error( "div" );
   }


 /**
  * Returns a String.It returns "unknown" if the user
  * tries to get the typeName of a STLDataType which is not
  * one of the child class types.
  * @return  "unknown"
  * @see STLDataType
  */
public String typeName(){
      return "unknown";
}

 /**
  * Returns a STLDataType object.The function returns an error incase of
  * a sytax error for conditional statements.
  * @param   str: error message
  * @return  STLException
  * @see STLException, STLDataType
  */
public STLDataType syntaxError(String str){
      return error(str);
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation uminus on a STLDataType Object.
  * @return  STLException
  * @see STLException, STLDataType
  */
   public STLDataType uminus() {
         return error( "uminus" );
   }
```

```
}
```

FILE NAME: src/STLException.java

```
/**
 * @author STL
 * The class is responsible for handling errors
 * when the program is running.
 *
 */
public class STLException extends RuntimeException{

	//Incase of errors, this function is called and the error is displayed and
	//program stops execution.
	public STLException(String msg){
		System.err.println(msg);
	}


}
```

FILE NAME: src/STLFunction.java

```
import java.io.PrintWriter;

import antlr.collections.AST;

/**
 * @author Team STL
 * The class is responsible for handling functions.
 */
public class STLFunction extends STLDataType{

    String[] params; //The function arguments
    AST body;        //The body of the function
    STLSymbolTable fstbl;  //The symbol table for the function.


        /**
         * Contructor for the STLFunction Class.
         * @param   name: function name
         * @param   params: function arguments
         * @param   body: function body.
         * @param   stbl: Symboltable
         * @return  none
         * @see STLDataType
         */
    public STLFunction( String name, String[] params,
                    AST body, STLSymbolTable stbl) {
        super(name);
        this.params = params;
        this.body = body;
        this.fstbl = stbl;
    }

        /**
         * Contructor for the STLFunction Class.
         * @param   name: function name
         * @return  none
         * @see STLDataType
         */
    public STLFunction( String name) {
        super( name );
        this.params = null;
        fstbl = null;
        body = null;
    }

        /**
```

```
     * Returns a String.It returns "function" if the user
     * tries to get the typeName of a STLFunction.
     * @return  "function"
     * @see STLDataType
     */
public String typename() {
   return "function";
}


    /**
     * The method prints out the information about the function.
     * @return  none
     * @see STLDataType
     */
public void print( PrintWriter w ) {
  if (getName() != null)
          w.print(getName() + " = ");

  w.print("<function>(");
          for (int i = 0; i < params.length; i++) {
                w.print(params[i]);
                w.print(",");
          }
          w.println(")");
}


    /**
    * Returns an String[].
    * @return  String[], args of function
    */
public String[] getArgs() {
   return params;
}

    /**
    * Returns an STLSymbolTable of parent.
    * @return  symboltable of function.
    */
public STLSymbolTable getParentSymbolTable() {
   return fstbl;
}

    /**
    * Returns an body of function.
    * @return  AST, body of function
    */
```

```
    public AST getBody() {
        return body;
    }

}
```

FILE NAME: src/STLGui.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.text.*;
import java.io.*;
import java.util.*;
import java.text.*;
import antlr.CommonAST;
import antlr.debug.misc.ASTFrame;

/**
 *
 * @author Team STL
 * The STLGui class was initially written for testing purposes, but is now a convenient
 * way to present the language.  It should not be considered a substitute for running the
 * interpreter by itself, as its multithreaded nature and additional features can easily
 * introduce confusing errors into otherwise valid STL programs.
 *
 */
public final class STLGui extends JFrame implements KeyListener, ActionListener,
AdjustmentListener{

    String testdir = "./test/";
    double money;
    Hashtable portfolio;
    PrintStream other_stream;

    //stl components
    STLLexer lexer;
    STLParser parser;
    CommonAST parseTree;
    STLWalker walker;
    STLDataType dt;
    ASTFrame frame;

    //swing components
    JPanel top;
    JMenuBar menu;
    JTextPane input, output;
    //JFormattedTextField cash;
    JLabel cash;
    JList stock_list;

    JButton quit, save, load, run, clear, parse, test;
    JToggleButton toggle,onin;
    DefaultStyledDocument input_doc, output_doc;
```

```
JScrollPane inScroll, outScroll;
JSplitPane io, listdiv;
FilteredStream err_redirect,out_redirect;

/**
 * Contructor for the STLString Class. Initialize str.
 * @param   tmp
 * @return  none
 * @see STLDataType
 */
public STLGui(){
   try{

        stock_list = new JList();
        money = 0;

        out_redirect = new FilteredStream(new Color(255,255,255));
        err_redirect = new FilteredStream(new Color(255,0,0));
        other_stream = System.out;
        System.setOut(new PrintStream(out_redirect,true));

        System.setErr(new PrintStream(err_redirect,true));
        top = new JPanel();
        top.setLayout(new BoxLayout(top,BoxLayout.PAGE_AXIS));

        input_doc = new STLCode();
        output_doc = new DefaultStyledDocument();

        input = new JTextPane(input_doc);
        output = new JTextPane(output_doc);

        menu = new JMenuBar();
        quit = new JButton("Quit");
        save = new JButton("Save");
        load = new JButton("Open");
        run = new JButton("Run");
        clear = new JButton("Clear");
        parse = new JButton("Parse");
        test = new JButton("Tests");
        toggle = new JToggleButton("IO");
        onin = new JToggleButton("AutoScroll");

        //cash = new JFormattedTextField("$0");
        cash = new JLabel("$0");

        save.addActionListener(this);
        load.addActionListener(this);
        quit.addActionListener(this);
```

```
        run.addActionListener(this);
        clear.addActionListener(this);
        parse.addActionListener(this);
        test.addActionListener(this);
        toggle.addActionListener(this);
        onin.addActionListener(this);

        menu.add(test);
        menu.add(clear);
        menu.add(run);
        menu.add(parse);
        menu.add(load);
        menu.add(save);
        menu.add(quit);
        menu.add(toggle);
        menu.add(onin);
        menu.add(cash);

        setJMenuBar(menu);

        inScroll = new JScrollPane(input);
        inScroll.setColumnHeaderView(new JLabel("Input"));
        outScroll = new JScrollPane(output);
        outScroll.setColumnHeaderView(new JLabel("Output"));
        outScroll.getVerticalScrollBar().addAdjustmentListener(this);

        input.setBackground(Color.white);
        output.setBackground(Color.white);
        output.setEditable(false);
        io = new JSplitPane(JSplitPane.VERTICAL_SPLIT);
        listdiv = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT);
        io.setTopComponent(outScroll);
        io.setBottomComponent(inScroll);
        output.setToolTipText("Output from the interpreter");
        input.setToolTipText("Enter code (execute with F1, open file with F12, save with
F11)");
        input.addKeyListener(this);

        listdiv.setLeftComponent(io);
        listdiv.setRightComponent(stock_list);
        top.add(listdiv);
        add(top);
        setDefaultCloseOperation (JFrame. DISPOSE_ON_CLOSE);
        setMinimumSize(new Dimension(640,480));
        pack();
        io.setDividerLocation(.5);
        listdiv.setDividerLocation(.75);
```

```
            String r[] = {"<no stocks>"};
            stock_list.setListData(r);

            setVisible(true);

        }catch(Exception e){}
    }

    public void adjustmentValueChanged(AdjustmentEvent ev){
        //System.out.println(ev.getAdjustmentType());
        if(onin.isSelected()){
outScroll.getVerticalScrollBar().setValue(outScroll.getVerticalScrollBar().getMaximum());
        }
    }


    //implement the ActionListener interface
    public void actionPerformed(ActionEvent ev){
        if(ev.getActionCommand()=="Save"){
            save();
        }
        else if(ev.getActionCommand()=="Open"){
            load();
        }
        else if(ev.getActionCommand()=="Quit"){
            quit();
        }
        else if(ev.getActionCommand()=="Run"){
            run();
        }
        else if(ev.getActionCommand()=="Clear"){
            try{
                input_doc.remove(0,input_doc.getLength());
            }catch(Exception e){}
        }
        else if(ev.getActionCommand()=="Parse"){
            parse();
        }
        else if(ev.getActionCommand()=="Tests"){
            tests();
        }
        else if(ev.getActionCommand()=="IO"){
            PrintStream temp = System.out;
            System.setOut(other_stream);
            System.setErr(other_stream);
            other_stream = temp;
        }
```

```
      else if(ev.getActionCommand()=="AutoScroll"){
      }
   }

   void tests(){
      String files[] = new File(testdir).list();
      for(int i=0;i<files.length;i++){
         if(files[i].endsWith(".stl")){
            System.out.print(files[i] + "\n");
            File f = new File(testdir + files[i]);
            loadin(f);
            run();
            System.out.print("\n*************************\n");
            try{
               input_doc.remove(0,input_doc.getLength());
            }catch(Exception e){}
         }
      }
   }

   void load(){
      try{
         JFileChooser fc = new JFileChooser();
         fc.showOpenDialog(this);
         File file = fc.getSelectedFile();
         if(file.isFile()){
            System.out.println(file.toString());
            loadin(file);
         }
      }catch(Exception e){ System.err.println("Exception: "+e); }
   }

   void loadin(File file){
      try{
         BufferedReader reader = new BufferedReader(new FileReader(file));
         while(reader.ready()){

input_doc.insertString(input_doc.getEndPosition().getOffset(),reader.readLine()+"\n",null);
         }
      }catch(Exception e){}
   }

   void save(){
      try{
         JFileChooser fc = new JFileChooser();
         fc.showSaveDialog(this);
         File file = fc.getSelectedFile();
         System.out.println(file.toString());
```

```
        BufferedWriter writer = new BufferedWriter(new FileWriter(file));
        String text = input_doc.getText(0,input_doc.getLength());
        writer.write(text,0,text.length());
        writer.flush();
    }catch(Exception e){ System.err.println("Exception: "+e); }
}

void parse(){
    try{
        lexer = new STLLexer(new StringReader(input.getText()));
        parser = new STLParser(lexer);
        parser.program();
        parseTree = (CommonAST)parser.getAST();
        frame = new ASTFrame("AST from the STL parser", parseTree);
        frame.setVisible(true);
    }catch(Exception e){System.err.println("Exception: "+e);}
}

void run(){
    try{

        lexer = new STLLexer(new StringReader(input.getText()));
        parser = new STLParser(lexer);
        parser.program();
        parseTree = (CommonAST)parser.getAST();
        walker = new STLWalker();
        if(portfolio!=null){
            walker.intpret.stockSource.portfolio = portfolio;
        }
        walker.intpret.stockSource.money = money;
        dt = walker.expr(parseTree);
        cash.setText(" $" + walker.intpret.stockSource.money +" ");
        Object l[] = walker.intpret.stockSource.stocks();
        portfolio = walker.intpret.stockSource.portfolio;
        money = walker.intpret.stockSource.money;
        if(l!=null){
            for(int c=0;c<l.length;c++){
                l[c] = l[c] + " " + walker.intpret.stockSource.stocks((String)(l[c]));
            }
            stock_list.setListData(l);
        }
        else{
            String r[] = {"<no stocks>"};
            stock_list.setListData(r);
        }

    }catch(Exception e){ System.err.println("Exception: "+e);}
}
```

```
void quit(){
   System.exit(0);
}

//implement the KeyListener interface
public void keyReleased(KeyEvent ev){}
public void keyTyped(KeyEvent ev){}
public void keyPressed(KeyEvent ev){
      if(ev.getKeyCode()==112){
         run();
      }
      else if(ev.getKeyCode()==123){
         load();
      }
      else if(ev.getKeyCode()==122){
         save();
      }
}

//this class is for redirecting stdout (by extending a class to write where we want)
class FilteredStream extends FilterOutputStream {
   MutableAttributeSet style;
   Color def;

   public FilteredStream(Color c) {
      super(new ByteArrayOutputStream());
      style = new SimpleAttributeSet();
      StyleConstants.setBackground(style,c);
   }

   public void write(byte b[]) throws IOException {
      try{
         String aString = new String(b);
         output_doc.insertString(output_doc.getEndPosition().getOffset(),aString,style);
         flush();
      }catch(Exception e){}
   }

   public void write(byte b[], int off, int len) throws IOException {
      try{
         String aString = new String(b , off , len);
         output_doc.insertString(output_doc.getEndPosition().getOffset(),aString,style);
         flush();
      }catch(Exception e){}
   }
}
```

```
    class STLCode extends DefaultStyledDocument{
        public void insertString(int offset,String str,AttributeSet a) throws
BadLocationException{
            try{
                super.insertString(offset,str,a);
            }catch(Exception e){}
        }
    }

    //just create an instance of the interpreter
    public static void main(String[] args){
        STLGui mw = new STLGui();
    }
}
```

FILE NAME: src/STLInterpreter.java

```
import java.util.StringTokenizer;
import java.util.Vector;

import net.neurotech.quotes.QuoteException;
import antlr.RecognitionException;
import antlr.collections.AST;

public class STLInterpreter {

        STLSymbolTable symtbl; //Symbol Table

    final static int no_flow = 0; //Needed to indicate normal flow
    final static int return_flow = 3; //needed to indicate return statement reached.

    private int control = no_flow; //Control variable to check flow.
    private String label;

    private Vector vWhileOrAnd = new Vector();
    //private Vector vOrAnd = new Vector();//record result expr result in while
    private Vector vOrAndRecord = new Vector();//record sequence of operator "and" "or"

    //The class to interact with to get stock information and maintain user portfolio
information.

    public STLStockSource stockSource = new STLStockSource();
        //Constructor to initialize STLInterpreter.
    public STLInterpreter(){
                symtbl = new STLSymbolTable(null);
        }

        /**
         * Returns an STLDataType object. The function does assigns
         * the value of the right hand side variable/constant to the
         * left side variable, if they are of the same type. Otherwise
         * it throws an STLException.
         *
         * @param  a  the left side variable in the assignment
         * @param  b the right side variable/constant in the assignment
         * @return  the value of the assignment of type STLDataType
         */
        public STLDataType assign(STLDataType a, STLDataType b){
                STLDataType y = null;

                if (a.getName() != null)// && (a.typeName().equals(b.typeName())))
        {
                        if (a.getName().substring(0,1).equals("$")){
```

```
                y = new STLNumber(a.getName());
            }else if (a.getName().substring(0,1).equals("%")){
                y = new STLString(a.getName(), "");
            }else if (a.getName().substring(0,1).equals("@")){
                y = new STLList(a.getName());
            }

    STLDataType x = b;

    if(b.typeName().equals(y.typeName())){
     x.setName( y.getName() );

     symtbl.setValue( x.getName() , x, true, 0 );  // scope?
     return x;
    }else{
        return a.error(y, b, "=" );
    }
  }
  return a.error( b, "=" );
  }


/*      public STLDataType invokeFunc(STLWalker walker, STLDataType funct,
STLDataType[] params){
        return null;
  }
  */

  /**
   * Returns an STLDataType object. The function converts a
   * String into a STLNumber STLDataType. This required for various
   * assignments and operations, since they only operate if the
   * the operands are of type STLDataType.
   *
   * @param  s  String s to be converted to type STLNumber
   * @return  the STLNumber value of String s.
   * @see STLNumber, STLDataType
   */
  public STLDataType getNumber(String s){
        if(s.indexOf(".") >= 0)
              return new STLNumber(Double.parseDouble(s));
        else
              return new STLNumber(Integer.parseInt(s));
  }


  /**
   * Returns an STLDataType object. The function converts a
```

```
 * String into a STLList STLDataType. This required for various
 * assignments and operations, since they only operate if the
 * the operands are of type STLDataType.
 *
 * @param  s  String s to be converted to type STLList
 * @return  the STLList value of String s.
 * @see STLList, STLDataType
 */
public STLDataType getList(String s){
	String temps = s.substring(1, s.length()- 1);
	Vector v = new Vector();
	StringTokenizer token = new StringTokenizer(temps,", ");
	while(token.hasMoreTokens()){
				String tmp = token.nextToken().replace(" ", "");

		if (tmp.substring(0, 1).equals("%")) {
			if(symtbl.containsKey(tmp)){
				v.add(((STLString)symtbl.getValue(tmp,true,0)).getStr());//TODO
check if level is always 0
			}else
				return (new STLList(tmp)).error("Undefined String variable in
list");
		} else {
			v.add(tmp);
		}
	}

	return new STLList(v);

}

/**
 * Returns an STLDataType object. The function takes in a
 * String s which represents the name of a variable. It then
 * returns the value of type STLVariable (STLDataType object) which contains the
 * value of variable as obtained from the symboltable.
 *
 * @param  s  String s to be converted to type STLVariable
 * @return  the STLVariable value of String s.
 * @see STLVariable, STLDataType
 */
public STLDataType getVariable(String s){
	STLDataType x = symtbl.getValue( s, true, 0 );
	if ( x == null )
   return new STLVariable( s );
return x;
}
```

```
    /**
     * Returns an STLDataType object. The function takes in a
     * STLDataType object, and returns its value.
     *
     * @param  STLDataType dt
     * @return  the STLDataType value of dt.
     * @see STLDataType
     */
public STLDataType rvalue(STLDataType dt){
        if ( null == dt.getName())
        return dt;
    return dt;
}

    /**
     * Returns an STLDataType object. The function prints out
     * the value of STLDataType a.
     *
     * @param  a  STLDataType a which is to be printed out.
     * @return  null.
     * @see STLDataType
     */
    public STLDataType print(STLDataType a) {
            a.print();
            return (new STLDataType());
    }




    /**
     * Returns an array of STLDataType objects. The function takes in a
     * Vector v and converts it into a array of STLDataType objects.
     *
     * @param  v  Vector
     * @return  Array of STLDataType objects
     * @see STLDataType
     */
    public STLDataType[] convertExprList(Vector v) {
            STLDataType[] x = new STLDataType[v.size()];
    for ( int i=0; i<x.length; i++ )
        x[i] = (STLDataType) v.elementAt( i );
    return x;
    }

    /**
     * Initialize while expr with symbol table
```

```
 * @param expr
 */
public void whileInit( STLDataType[] expr ) {

      symtbl = new STLSymbolTable(new STLSymbolTable(symtbl), symtbl);
      if (expr[1] instanceof STLNumber && expr[1].getName()!= null)
      {
            STLDataType x = symtbl.getValue(expr[1].getName(), true, 0);
            symtbl.setValue(expr[1].getName(), expr[1], true, 0);
      }

      if (expr[2] instanceof STLNumber && expr[2].getName()!= null)
      {
            STLDataType x = symtbl.getValue(expr[2].getName(), true, 0);
            symtbl.setValue(expr[2].getName(), expr[2], true, 0);
      }
}
/**
 * Initialize while loop with Or And exprs with symbol table
 * @param dtl
 * @param dtr
 */
public void whileOrAndInit( STLDataType dtl, STLDataType dtr) {
      symtbl = new STLSymbolTable(new STLSymbolTable(symtbl), symtbl);

      if (dtl instanceof STLNumber && dtl.getName()!= null)
      {
            STLDataType x = symtbl.getValue(dtl.getName(), true, 0);
            symtbl.setValue(dtl.getName(), dtl, true, 0);
      }

      if (dtr instanceof STLNumber && dtr.getName()!= null)
      {
            STLDataType x = symtbl.getValue(dtr.getName(), true, 0);
            symtbl.setValue(dtr.getName(), dtr, true, 0);
      }
}

public void whileOrAndRecord( STLDataType dtl, STLDataType dtr, int i) {
      vWhileOrAnd.addElement(dtl);
      vWhileOrAnd.addElement(dtr);
      vWhileOrAnd.addElement(new Integer(i));
}

public void whileOrAndOperator(int i) {
      switch(i)
      {
      case 0:
```

```
			vOrAndRecord.clear();
			break;
		case 1:
			vOrAndRecord.addElement(new Integer(1));
			break;
		case 2:
			vOrAndRecord.addElement(new Integer(2));
			break;
		}
	}

	/**
	 * while loop with mix and or exprs
	 * @param expr
	 * @return
	 */
	public boolean whileOrAndCanProceed(STLDataType[] expr) {

		STLDataType a, b;
		a = null;
		b = null;
		double lvalue = 0.0;
		double rvalue = 0.0;
		Vector vExprResult = new Vector();


		for (int i = 0; i < vWhileOrAnd.size(); i++)
		{
			if ( i % 3 == 0)
			{
				a = (STLDataType)vWhileOrAnd.elementAt(i);
				b = (STLDataType)vWhileOrAnd.elementAt(i+1);
			}
			else if (i % 3 == 2)
			{
				int y = (Integer)vWhileOrAnd.elementAt(i);
				switch (y)
				{
				case 0:
						if ( a.getName() != null && symtbl.containsVar(a.getName()))
							a = symtbl.getValue(a.getName(), true, 0);
						if (b.getName() != null && symtbl.containsVar(b.getName()))
							b = symtbl.getValue(b.getName(), true, 0);
						lvalue = ((STLNumber)a).getNum();
						rvalue = ((STLNumber)b).getNum();
						if (lvalue >= rvalue)
						{
							vExprResult.add(new Boolean(true));
```

```
            }
            else
            {
                vExprResult.add(new Boolean(false));
            }
    break;

    case 1:
            if ( a.getName() != null && symtbl.containsVar(a.getName()))
                a = symtbl.getValue(a.getName(), true, 0);
            if (b.getName() != null && symtbl.containsVar(b.getName()))
                b = symtbl.getValue(b.getName(), true, 0);
            lvalue = ((STLNumber)a).getNum();
            rvalue = ((STLNumber)b).getNum();
            if (lvalue <= rvalue)
            {
                vExprResult.add(new Boolean(true));
            }
            else
            {
                vExprResult.add(new Boolean(false));
            }

    break;

    case 2:
            if ( a.getName() != null && symtbl.containsVar(a.getName()))
                a = symtbl.getValue(a.getName(), true, 0);
            if (b.getName() != null && symtbl.containsVar(b.getName()))
                b = symtbl.getValue(b.getName(), true, 0);
            lvalue = ((STLNumber)a).getNum();
            rvalue = ((STLNumber)b).getNum();
            if (lvalue > rvalue)
            {
                vExprResult.add(new Boolean(true));
            }
            else
            {
                vExprResult.add(new Boolean(false));
            }

    break;

    case 3:
            if ( a.getName() != null && symtbl.containsVar(a.getName()))
                a = symtbl.getValue(a.getName(), true, 0);
            if (b.getName() != null && symtbl.containsVar(b.getName()))
                b = symtbl.getValue(b.getName(), true, 0);
```

```
			lvalue = ((STLNumber)a).getNum();
			rvalue = ((STLNumber)b).getNum();
			if (lvalue < rvalue)
			{
				vExprResult.add(new Boolean(true));
			}
			else
			{
				vExprResult.add(new Boolean(false));
			}

		break;

		case 4:
			if ( a.getName() != null && symtbl.containsVar(a.getName()))
				a = symtbl.getValue(a.getName(), true, 0);
			if (b.getName() != null && symtbl.containsVar(b.getName()))
				b = symtbl.getValue(b.getName(), true, 0);
			lvalue = ((STLNumber)a).getNum();
			rvalue = ((STLNumber)b).getNum();
			if (lvalue == rvalue)
			{
				vExprResult.add(new Boolean(true));
			}
			else
			{
				vExprResult.add(new Boolean(false));
			}

		break;

		case 5:
			if ( a.getName() != null && symtbl.containsVar(a.getName()))
				a = symtbl.getValue(a.getName(), true, 0);
			if (b.getName() != null && symtbl.containsVar(b.getName()))
				b = symtbl.getValue(b.getName(), true, 0);
			lvalue = ((STLNumber)a).getNum();
			rvalue = ((STLNumber)b).getNum();
			if (lvalue != rvalue)
			{
				vExprResult.add(new Boolean(true));
			}
			else
			{
				vExprResult.add(new Boolean(false));
			}

		break;
```

```
			case 6:
				vExprResult.add(new Boolean(true));
			break;

			case 7:
				vExprResult.add(new Boolean(false));
			break;

			default:
			break;
			}
		}
	}

	boolean flag = (Boolean)vExprResult.elementAt(0);

	Vector vExprOp = new Vector();
	Vector vAndIndex = new Vector();

	int andCount = 0;
	int andIndex = 0;
	boolean tempFlag = (Boolean)vExprResult.elementAt(0);

	for (int p = 0; p < vExprResult.size(); p++)
	{
		vExprOp.add(vExprResult.elementAt(p));
		if (p != (vExprResult.size() - 1))
			vExprOp.add(vOrAndRecord.elementAt(p));
	}


	for (int q = 0; q < vExprOp.size(); q++)//build vAndIndex
	{
		if (q % 2 == 1)
		{
			if (((Integer)vExprOp.elementAt(q)).intValue() == 2)
			{
				andCount++;
				vAndIndex.add(new Integer(q));
			}
		}
	}


	if (andCount == 0)
	{
		for (int i = 0; i < vExprResult.size(); i++)
```

```
				{
					flag = flag || (Boolean)vExprResult.elementAt(i);
				}

				return flag;
			}
			else if (andCount == ((vExprOp.size() - 1) / 2))
			{
				for (int y = 0; y < vExprResult.size(); y++)
				{
					flag = flag && (Boolean)vExprResult.elementAt(y);
				}

				return flag;
			}
			else
			{

				for (int m = 0; m < andCount; m++)//scanning vExprOp for m times
				{
					//TODO logic
					boolean conAndExist = false;
					int conAndIndex = 0; //the index of first consecutive "and"
					int conAndNumber = 1; // conAndNumber = 1 means "and"
conAndNumber = 2 means "and" "and"

					for (int n = 0; n < vAndIndex.size(); n++)//find consecutive "and" in
vAndIndex
					{
						if (n <= vAndIndex.size() - 2 &&
(Integer)vAndIndex.elementAt(n+1) - (Integer)vAndIndex.elementAt(n) == 2) //handle the
last element in vAndIndex
						{
							if (conAndExist == false)
							{
								conAndIndex = n;//assign index of the first
consecutive "and"
								conAndExist = true;
								conAndNumber++;// the number of consecutive
"and" is 2 now
								int p = conAndIndex + 1;//start to find the length of
consecutive right now is "and" "and", from this point find the max length of consecutive
"and"
								while ( p != vAndIndex.size() - 1 &&
(Integer)vAndIndex.elementAt(p+1) - (Integer)vAndIndex.elementAt(p) == 2)
								{
									conAndNumber++;
									p++;
```

```
                        }
                    }

                }

            }

            if (conAndExist)
            {
                int tIndexVEXPROP =
((Integer)vAndIndex.elementAt(conAndIndex)).intValue() - 1;//the index starting combine
                boolean tFlagVEXPROP =
((Boolean)vExprOp.elementAt(tIndexVEXPROP)).booleanValue();
                int combineIndex = tIndexVEXPROP;

                for (int k = 0; k < conAndNumber; k++)//execute consecutive
"and"
                {
                    tFlagVEXPROP = tFlagVEXPROP &&
((Boolean)vExprOp.elementAt(combineIndex + 2)).booleanValue();
                    combineIndex = combineIndex + 2;
                }

                vExprOp.set(tIndexVEXPROP, new
Boolean(tFlagVEXPROP));// update vExprOp boolean value

                for (int p = 0; p < conAndNumber * 2; p++)//update vExprOp
size
                {
                    vExprOp.remove(tIndexVEXPROP+1);
                }

                for (int e = 0; e < conAndNumber; e++)//update vAndIndex size
                {
                    vAndIndex.remove(0);
                }

                if (!vAndIndex.isEmpty())//update vAndIndex index value
                {
                    for (int v = 0; v < vAndIndex.size(); v++)
                    {
                        int oldIndex =
((Integer)vAndIndex.elementAt(v)).intValue();
                        vAndIndex.set(v, new Integer(oldIndex -
conAndNumber * 2));
                    }
                }
```

```
                        andCount = andCount - conAndNumber;

                    }
                    else//there is no conAnd just find "and" then combine; times are
according to andCount
                    {
                        for (int d = 0; d < andCount; d++)
                        {
                            int combineStart =
((Integer)vAndIndex.elementAt(0)).intValue() - 1;
                            boolean tFlagVEXPROP2 =
((Boolean)vExprOp.elementAt(combineStart)).booleanValue();
                            tFlagVEXPROP2 = tFlagVEXPROP2 &&
((Boolean)vExprOp.elementAt(combineStart + 2)).booleanValue();
                            vExprOp.set(combineStart, new
Boolean(tFlagVEXPROP2));//update vExprOp boolean
                            vExprOp.remove(combineStart + 1);//update vExprOp size
                            vExprOp.remove(combineStart + 1);
                            vAndIndex.remove(0);// update vAndIndex size
                            for (int w = 0; w < vAndIndex.size(); w++)// update
vAndIndex index value
                            {
                                int oldIndex2 =
((Integer)vAndIndex.elementAt(w)).intValue();
                                vAndIndex.set(w, new Integer(oldIndex2 - 2));
                            }
                        }

                        andCount = 0;
                    }
                }//end of for (int m = 0; m < andCount; m++)//scanning vExprOp for m
times

                flag = ((Boolean)vExprOp.elementAt(0)).booleanValue();

                for (int v = 2; v < vExprOp.size(); v=v+2)
                {
                    flag = flag || (Boolean)vExprOp.elementAt(v);
                }

                return flag;
            }

        }

        public boolean whileCanProceed( STLDataType[] expr) {

            if(symtbl.containsVar(expr[1].getName()))
```

```
{
    //System.out.println("get in whilecanproceed 1");
    expr[1] = symtbl.getValue(expr[1].getName(), true, 0);
    if (expr[1] instanceof STLNumber)
    {
        double index = ((STLNumber)expr[3]).getNum();
        double rvalue = ((STLNumber)expr[2]).getNum();
        double lvalue = ((STLNumber)expr[1]).getNum();

        if ( index == 0.0) //GE
        {
            if (lvalue >= rvalue)
                return true;
            else return false;
        }

        if ( index == 1.0) //LE
        {
            if (lvalue <= rvalue)
                return true;
            else return false;
        }

        if ( index == 2.0) //GT
        {
            if (lvalue > rvalue)
                return true;
            else return false;
        }

        if ( index == 3.0) //LT
        {
            if (lvalue < rvalue)
                return true;
            else return false;
        }

        if ( index == 4.0) //EQ
        {
            if (lvalue == rvalue)
                return true;
            else return false;
        }

        if ( index == 5.0) //NEQ
        {
            if (lvalue != rvalue)
                return true;
```

```
                        else return false;
                }

        }

}

if(symtbl.containsVar(expr[2].getName()))//update the variable in wexpr
{

        expr[2] = symtbl.getValue(expr[2].getName(), true, 0);

        if (expr[2] instanceof STLNumber)// Type check and then update boolean
of wexpr
        {

                double index = ((STLNumber)expr[3]).getNum();
                double rvalue = ((STLNumber)expr[2]).getNum();
                double lvalue = ((STLNumber)expr[1]).getNum();

                if ( index == 0.0) //GE
                {
                        if (lvalue >= rvalue)
                                return true;
                        else return false;
                }

                if ( index == 1.0) //LE
                {
                        if (lvalue <= rvalue)
                                return true;
                        else return false;
                }

                if ( index == 2.0) //GT
                {
                        if (lvalue > rvalue)
                                return true;
                        else return false;
                }

                if ( index == 3.0) //LT
                {
                        if (lvalue < rvalue)
                                return true;
                        else return false;
                }
```

```
                if ( index == 4.0) //EQ
                {
                        if (lvalue == rvalue)
                                return true;
                        else return false;
                }

                if ( index == 5.0) //NEQ
                {
                        if (lvalue != rvalue)
                                return true;
                        else return false;
                }

        }

}

if (((STLNumber)expr[3]).getNum() == 6.0)
        return true;
else if (((STLNumber)expr[3]).getNum() == 7.0)
        return false;
else
{
        double index = ((STLNumber)expr[3]).getNum();
        double rvalue = ((STLNumber)expr[2]).getNum();
        double lvalue = ((STLNumber)expr[1]).getNum();

        if ( index == 0.0) //GE
        {
                if (lvalue >= rvalue)
                        return true;
                else return false;
        }

        if ( index == 1.0) //LE
        {
                if (lvalue <= rvalue)
                        return true;
                else return false;
        }

        if ( index == 2.0) //GT
        {
                if (lvalue > rvalue)
                        return true;
                else return false;
        }
```

```
            if ( index == 3.0) //LT
            {
                if (lvalue < rvalue)
                    return true;
                else return false;
            }

            if ( index == 4.0) //EQ
            {
                if (lvalue == rvalue)
                    return true;
                else return false;
            }

            if ( index == 5.0) //NEQ
            {
                if (lvalue != rvalue)
                    return true;
                else return false;
            }

        }


        System.out.println("Something lost in Number comparison in wexpr!!!");
        return false;
    }
    /**
     * The function is called to check the boolean condition of foreach
     * @param a
     * @param b
     * @param i
     */
    public void foreachProceed(STLDataType a, STLDataType b, int i) {
        STLString tempS = new STLString((String)(((STLList)b).getVlist()).elementAt(i));
        symtbl.setValue( a.getName(), tempS, true, 0 );
    }


    /**
     * The function is called to see if the function
     * should proceed with the next statement or check
     * if a return statement has been called.
     * It returns true if it can proceed.
     *
     * @return  boolean :can proceed or not.
     */
```

```
    public boolean canProceed(){
        return control == no_flow;
    }

    public void setBreak(String s){
        ;
    }

  public void setContinue(STLDataType dt){
        ;
    }


    /**
     * The function is called when a return statement is reached. It
     * sets the value of the control variable to indicate that the function
     * should return to calling function
     *
     * @param  value String
     * @return  none
     */
  public void setReturn(String value){
        control = return_flow;
    }

    /**
     * The function is called when a return statement is reached. It
     * changes the value of control variable, to indicate that the function
     * should not continue after the return, but instead return to the
     * calling function.
     *
     * @param  flow_label String
     * @return  none
     */
  public void tryResetFlowControl( String flow_label ) {
    if ( null == label || label.equals( flow_label ) )
      control = no_flow;
  }

  /**
     * Returns an STLDataType object. The function is called when
     * a function is called. It processes the function by
     * executing its contents.
     *
     * @param  walker STLWalker: as the tree is walked ,the program is executed
     * @param  fname (STLDataType): the STLFunction which was defined earlier on in
the program.
     * @param  x (STLDataType[]): The parameters of the function.
```

```
     * @return  the object of type STLDataType which is returned by the function.
STLException
     *         returned incase of errors.
     * @see STLFunction, STLDataType
     */
    public STLDataType funcInvoke(STLWalker walker, STLDataType fname,
STLDataType[] x) {

    // func must have been defined earlier
    if ( !( fname instanceof STLFunction ) )
       return fname.error( "not a function" );

    // check defined function syntax is same as called function syntax
    String[] args = ((STLFunction)fname).getArgs();

    if ( args != null && args.length != x.length )
       return fname.error( "unmatched length of parameters" );

    // A symbol table must be created for each new function
    symtbl = new STLSymbolTable( ((STLFunction)fname).getParentSymbolTable(),
                     symtbl);


    // set up arguments of function
    if (args != null) {
            for (int i = 0; i < args.length; i++) {
                STLDataType d = rvalue(x[i]);
                d.setName(args[i]);
                symtbl.setValue(args[i], d, false, 0);
            }
        }

    // execute body of function
    STLDataType r = null;
        try {
            r = walker.expr( ((STLFunction)fname).getBody() );
        } catch (RecognitionException e) {
            return fname.error("Error in body of function");
        }

    // if a return was called
    if ( control == return_flow ){
       tryResetFlowControl( ((STLFunction)fname).getName());
    }

    // remove symbol table and return
    symtbl = symtbl.getParent();
```

```
	//check if return type matches function return type.
	if(checkTypeMismatch(fname,r)){
		return fname.error("Type mismatch in return statement in function "+
fname.getName());
	}
	return r;

	}

	/**
	 * Returns a boolean value. The function a STLDataType object
	 * function and the return type of the function being executed.
	 * If the return type of the function is the same as the value
	 * being returned false is returned (no type mismatch), otherwise
	 * false is returned.
	 *
	 * @param  fname: STLFunction STLDataType
	 * @param  r: The return value of the function
	 * @return  true incase of type mismatch
	 *		false otherwise.
	 * @see STLFunction, STLDataType
	 */
	private boolean checkTypeMismatch(STLDataType fname, STLDataType r){
		String funcName = fname.getName();
		try{
		if (funcName.substring(0,1).equals("$")){  //If function expects STLNumber
returned
			STLNumber stlnum = (STLNumber)r;
		}else if (funcName.substring(0,1).equals("%")){ //If function expects STLString
return
			STLString stlstr = (STLString)r;
		}else if (funcName.substring(0,1).equals("@")){ //If function expects STLList
return
			STLList stlist = (STLList)r;
		}
		}catch(ClassCastException e){
			return true;		//type mismatch so return true
		}
		return false;
	}

	/**
	 * Returns an STLDataType object. The function takes in an
	 * array of DataTypes which contain a number of stocks and the
	 * name of the stock which the user wishes to buy.
	 * Ex: buy(5,"IBM");
	 *
	 * @param  x  Array of STLDataType objects
```

```
         * @return  null incase of correct operation, otherwise returns STLException.
         * @see StockSource, STLDataType
         */
        public STLDataType buy(STLDataType[] x) {
                int numStocks = 0;
                String stockName = "";
                if(x.length != 2)
                        return x[0].error("buy # arguments incorrect");
                try {
                        numStocks = (int) ((STLNumber) x[0]).getNum();
                        stockName = ((STLString) x[1]).getStr();

                        stockSource.buy(numStocks, stockName);
                }catch (ClassCastException e) {
                        return x[0].error("Incorrect arguments in buy statement expect buy(number,
string)");
                }
                catch (QuoteException e) {
                        return x[1].error("Incorrect stock name arguments in buy statement ");
                }
                return null;
        }

        /**
         * Returns an STLDataType object. The function takes in an
         * array of STLDataType objects which contain a number of stocks and the
         * name of the stock which the user wishes to sell.
         * Ex: sell(5,"IBM");
         *
         * @param  x  Array of STLDataType objects
         * @return  null incase of correct operation, otherwise returns STLException.
         * @see StockSource, STLDataType
         */
        public STLDataType sell(STLDataType[] x) {
                int numStocks = 0;
                String stockName = "";
                if(x.length != 2)
                        return x[0].error("sell # arguments incorrect");

                try {
                        numStocks = (int) ((STLNumber) x[0]).getNum();
                        stockName = ((STLString) x[1]).getStr();
                        stockSource.sell(numStocks, stockName);
                } catch (ClassCastException e) {
                        return x[0]
                                        .error("Incorrect arguments in sell statement expect sell(number,
string)");
                } catch (QuoteException e) {
```

```
			return x[1]
					.error("Incorrect stockname arguments in sell statement");
		}
		return null;
	}

	/**
	 * Returns an STLDataType object. The function takes in an
	 * array of STLDataType objects which contains a single string.
	 * Ex: value("IBM");
	 *
	 * @param  x  Array of STLDataType objects
	 * @return  current value of the stock incase of correct operation,
	 *          otherwise returns STLException.
	 * @see StockSource, STLDataType
	 */
	public STLDataType value(STLDataType[] x) {
		String valname = "";
		double value = 0;

		try {
		if(x.length != 1){
			return x[0].error("wrong syntax for value");
		}else{
			valname = ((STLString)x[0]).getStr();
		}
			value = stockSource.value(valname,0);
		}catch (ClassCastException e) {	//incase the semantics of the call are
incorrect
			return x[0].error("incorrect arguments to value function expect
value(stockname)");
		} catch (QuoteException e) {   //incase the stockname specified does not exist
			return x[0].error("illegal operation stock symbol not found "+valname);
		} catch (Exception ex){
			return x[0].error("Incorrect argument sent for value expecting string got: ");
		}
		return new STLNumber(value);
	}


	/**
	 * Returns an STLDataType object. The function takes in an
	 * array of STLDataType objects which contain a string which
	 * specifies a particular stock.
	 * Ex: wma("IBM");
	 *
	 * @param  x  Array of STLDataType objects
	 * @return  the wma value incase of correct operation,
```

```
 *         otherwise returns STLException.
 * @see StockSource, STLDataType
 */
public STLDataType wma(STLDataType[] x) {
        String wmavalue = "";
        double wmaresult = 0;
        try {
        if(x.length != 1){
                return x[0].error("wrong syntax for wma");
        }else{
                wmavalue = ((STLString)x[0]).getStr(); //call stockSource wma method.
        }
                wmaresult = stockSource.wma(wmavalue,null);
        } catch (ClassCastException ex){  //incase incorrect arguments to the call.
                return x[0].error("Incorrect argument sent for wma expecting string");
        } catch (QuoteException e) {   //incase the stockname specified does not exist
                return x[0].error("illegal operation stock symbol not found "+wmavalue);
        }
        return new STLNumber(wmaresult);
}


/**
 * Returns an STLDataType object. The function takes in an
 * array of STLDataType objects which contain a number of stocks and the
 * name of the stock which the user wishes to find the delta value for.
 * Ex: delta(5,"IBM");
 *
 * @param  x  Array of STLDataType objects
 * @return  delta value incase of correct operation,
 *         otherwise returns STLException.
 * @see StockSource, STLDataType
 */
public STLDataType delta(STLDataType[] x) {
        String stockName = "";
        double deltaval = 0;
        try {
        if(x.length != 1)
                return x[0].error("delta arguments incorrect");
        stockName  = ((STLString)x[0]).getStr();

                deltaval = stockSource.delta(stockName,0);

        } catch (QuoteException e) {
                return x[0].error("delta function error)");
        } catch (ClassCastException ex){
                return x[0].error("error in delta function expecting syntax delta(numofstocks,
stockname)");
```

```
        }

        return new STLNumber(deltaval);
}


/**
 * The method is called when a function definition is found. It
 * takes in the name of the function, its arguments and its body.It
 * then puts the function into the symbol table so that it
 * can be accessed when the function is called.
 *
 * @param  name: Function name
 * @param  args[]: Arguments to the function
 * @param  fbody: Body of the function
 * @return  none
 * @see STLSymbolTable, STLDataType
 */
public void funcRegister(String name, String[] args, AST fbody) {
symtbl.put( name, new STLFunction( name, args, fbody, symtbl ) );
}

/**
 * Returns an array of Strings. The function takes in a
 * Vector v and converts into a String array.
 *
 * @param  v: Vector
 * @return  String[]: Array of strings.
 */
public String[] convertVarList(Vector v) {
        String[] arr = new String[v.size()];
for ( int i=0; i < arr.length; i++ )
   arr[i] = (String) v.elementAt( i );
return arr;
}

/**
 * Returns an STLDataType object. The function takes in an
 * array of STLDataType objects which contain a number seconds
 * to suspend the program
 * Ex: wait(5);
 *
 * @param  x  Array of STLDataType objects
 * @return  null incase of correct operation,
 *         otherwise returns STLException.
 * @see StockSource, STLDataType
 */
public STLDataType wait(STLDataType[] x) {
```

```
		double wait_time = 0;
		if(x.length == 1){
			try {
				wait_time = ((STLNumber) x[0]).getNum(); //get the numerical value
of wait time.
				Thread.sleep((long) (wait_time * 1000)); //suspend thread for required
time.
			}catch (ClassCastException e) {  //incorrect syntax.
				return x[0].error("Error in the wait statement,expect numerical
argument");
			}catch (InterruptedException e) { //the wait statement was interrupted
				return x[0].error("Error in the wait statement, interrupted");
			}
			return null;
		}
		return (new STLDataType()).error("Error in syntax of wait");
	}

	/**
	 * Returns an STLDataType object.
	 * The function can take in either an empty STLDataType object
	 * Ex: money(). This syntax would return the amount of money
	 * the user has available.
	 * Otherwise the user could set the amount of money they have
	 * by doing Ex: money(100);
	 *
	 * @param  x  Array of STLDataType objects
	 * @return  The amount of money the user has available.
	 *         In case of error: STLException
	 * @see StockSource, STLDataType
	 */
	public STLDataType money(STLDataType[] x) {
		double money_amt = 0;
		try{
		if(x.length == 0){
			money_amt = stockSource.money();  //to get the amount of money
available money().
			return (new STLNumber(money_amt));
		}else if (x.length == 1){
			money_amt = ((STLNumber)x[0]).getNum();
			stockSource.money(money_amt);     //to set the money available
money(amt)
			return (new STLNumber(money_amt));
		}else
		   return x[0].error("incorrect number of arguments to money function expect
money() or money($amt)");
		}catch(ClassCastException e){
			return (new STLDataType()).error("incorrect argument to delta function");
```

```
        }
    }

    /**
     * Returns an STLDataType object.
     * The function can take in either an empty STLDataType object
     * Ex: stocks(). This syntax would return the an STLList of
     * all the stocks the user has available.
     * Otherwise the user could set get the number of stocks of
     * a particular stockname that the user has. Ex: stocks("IBM");
     *
     * @param  x  Array of STLDataType objects
     * @return  The STLList of stocks the user has
     *         or the number of stocks of a particular type.
     *         or In case of error: STLException
     * @see StockSource, STLDataType
     */
    public STLDataType stocks(STLDataType[] x) {
        String stockName;
        double number;
        if (x.length == 0){ //for stocks()
            Object[] stocklist = stockSource.stocks();    //get the list of stocks the user
has

            //Convert into STLList object
            Vector v = new Vector();
            if(stocklist != null){
            for (int i = 0; i < stocklist.length; i++) {
                    System.out.println("each element of stocklist "
                            + stocklist[i]);
                    v.addElement(stocklist[i]);
                }
            }
            return new STLList(v);
        }
        else if(x.length == 1){        // for stocks(type)
            try {
                stockName = ((STLString) x[0]).getStr(); //get the name of the stock
                number = stockSource.stocks(stockName); //get number available of
specified type
                return new STLNumber(number);
            } catch (ClassCastException e) {
                return x[0].error("Incorrect arg type to stocks(ID)"); //incorrect syntax.
            }
        }
        return x[0].error("Incorrect stock funct syntax");
    }
```

}

FILE NAME: src/STLList.java

```
import java.io.PrintWriter;
import java.util.Vector;

/**
 * @author Team STL
 * This class inherits from the STLDataType class. It
 * deals with the specific class of lists.
 */
public class STLList extends STLDataType{

	Vector vlist;

	/**
	  * Contructor for the STLList Class.
	  * @param   v: Vector v of contents of list
	  * @return  none
	  * @see STLDataType
	  */
	public STLList(Vector v){
		vlist = v;
	}

	/**
	  * Contructor for the STLList Class.
	  * @param   name: list name
	  * @return  none
	  * @see STLDataType
	  */
	public STLList(String name){
		super(name);
	}

	 /**
	  * Returns a String.It returns "list" if the user
	  * tries to get the typeName of a STLList.
	  * @return  "list"
	  * @see STLDataType
	  */
	public String typeName(){
	    return "list";
    }

	public STLDataType syntaxError(String str){
	    //TODO
	    return null;
    }
```

```
     /**
      * The method prints out the information about the STLList.
      * @return  none
      * @see STLDataType
      */
     public void print( PrintWriter w ) {
     for(int i = 0; i < vlist.size(); i++){
        w.print( vlist.get(i) );
        w.flush();
     }
  }


     /**
      * Returns vector v of contents of list
      * @return  vector v.
      */
     public Vector getVlist() {
            return vlist;
     }




}
```

FILE NAME: src/STLNumber.java

```
import java.io.PrintWriter;

/**
 *
 * @author Team STL
 * The class inherits from STLDataType. It deals with
 * numbers (int or double).
 *
 */
public class STLNumber extends STLDataType{

	private double num;	//value of number

	/**
	 * Constructor for STLNumber. Initializes num.
	 * @param dnum
	 * @return: none
	 */
	public STLNumber(double dnum){
		num = dnum;
	}

	/**
	 * Constructor for STLNumber. Initializes name of STLDataType.
	 * @param name
	 * @return: none
	 */
	public STLNumber(String name){
		super(name);
	}


	/**
	 * Returns an number
	 * @return  double.
	 */
	public double getNum() {
		return num;
	}


	/**
	 * Returns numerical double value if the datatype dt is of
	 * type STLNumber. Otherwise returns STLException.
	 * @param STLDataType dt
	 * @return  double or
```

```
 *         STLException incase of error
 * @see STLDataType
 */
public static double isSTLNumber(STLDataType dt){// Modified by Matt
       if ( dt instanceof STLNumber )
              return ((STLNumber)dt).num;
       dt.error(dt, "This is not a STLNumber");
       return 0;
}

/**
 * The function sets the value of the num private variable
 * @param num
 * @returns none
 */
public void setNum(double num) {
       this.num = num;
}


 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation GE on a STLDataType Object which is not STLnumber.
  * Otherwise it returns the value of the GE
  * @param   dt: STLDataType object
  * @return  STLException incase of error or
  *         The value of the GE evaluation in STLDataType
  * @see STLException, STLDataType
  */
public STLDataType GE(STLDataType dt) {//Modified by Matt
       if (dt instanceof STLNumber){
       return new STLBool( num >= isSTLNumber(dt) );}
       else
              return error(dt,"wrong datatype for GE");
}


 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation GT on a STLDataType Object which is not STLnumber.
  * Otherwise it returns the value of the GT
  * @param   dt: STLDataType object
  * @return  STLException incase of error or
  *         The value of the GT evaluation in STLDataType
  * @see STLException, STLDataType
  */
public STLDataType GT(STLDataType dt) {//Modified by Matt
       if (dt instanceof STLNumber){
```

```
            return new STLBool( num > isSTLNumber(dt) );}
            else
                  return error(dt,"wrong datatype for GT");
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation LE on a STLDataType Object which is not STLnumber.
  * Otherwise it returns the value of the LE
  * @param   dt: STLDataType object
  * @return  STLException incase of error or
  *         The value of the LE evaluation in STLDataType
  * @see STLException, STLDataType
  */
public STLDataType LE(STLDataType dt) {//Modified by Matt
      if (dt instanceof STLNumber){
            return new STLBool( num <= isSTLNumber(dt) );}
            else
                  return error(dt,"wrong datatype for LE");
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation LT on a STLDataType Object which is not STLnumber.
  * Otherwise it returns the value of the LT
  * @param   dt: STLDataType object
  * @return  STLException incase of error or
  *         The value of the LT evaluation in STLDataType
  * @see STLException, STLDataType
  */
public STLDataType LT(STLDataType dt) {//Modified by Matt
      if (dt instanceof STLNumber){
            return new STLBool( num < isSTLNumber(dt) );}
            else
                  return error(dt,"wrong datatype for LT");
}

 /**
  * Returns a STLDataType object.The function returns an error incase a user
  * tries to perform operation EQ on a STLDataType Object which is not STLnumber.
  * Otherwise it returns the value of the EQ
  * @param   dt: STLDataType object
  * @return  STLException incase of error or
  *         The value of the EQ evaluation in STLDataType
  * @see STLException, STLDataType
  */
public STLDataType EQ(STLDataType dt) {//Modified by Matt
      if (dt instanceof STLNumber){
```

```
                return new STLBool(num == isSTLNumber(dt));
            }
            else
                return error(dt, "wrong datatype for EQ");
        }

        /**
         * Returns a STLDataType object.The function returns an error incase a user
         * tries to perform operation NE on a STLDataType Object which is not STLnumber.
         * Otherwise it returns the value of the NE
         * @param   dt: STLDataType object
         * @return  STLException incase of error or
         *          The value of the NE evaluation in STLDataType
         * @see STLException, STLDataType
         */
        public STLDataType NE(STLDataType dt) {
            if (dt instanceof STLNumber){
                return new STLBool(num != isSTLNumber(dt));
            }
            else
                return error(dt, "wrong datatype for NE");
        }

        /**
         * Returns a STLDataType object.The function returns an error incase a user
         * tries to perform operation MULT on a STLDataType Object which is not
STLnumber.
         * Otherwise it returns the value of the MULT of operands
         * @param   dt: STLDataType object
         * @return  STLException incase of error or
         *          The value of the MULT evaluation in STLDataType
         * @see STLException, STLDataType
         */
        public STLDataType MULT(STLDataType dt) {
            if(dt instanceof STLNumber){
                return new STLNumber(num * ((STLNumber)dt).num) ;
            }else
                return error(dt,"wrong datatype for multiplication");
        }

        /**
         * Returns a STLDataType object.The function returns an error incase a user
         * tries to perform operation DIV on a STLDataType Object which is not
STLnumber.
         * Otherwise it returns the value of the DIV of operands
         * @param   dt: STLDataType object
         * @return  STLException incase of error or
         *          The value of the DIV evaluation in STLDataType
```

```
     * @see STLException, STLDataType
     */
    public STLDataType DIV(STLDataType dt) {
        if(dt instanceof STLNumber && ((STLNumber) dt).num !=0){
            return new STLNumber(num / ((STLNumber)dt).num) ;
        }else
            return error(dt,"wrong datatype for division");
    }

    /**
     * Returns a STLDataType object.The function returns an error incase a user
     * tries to perform operation ADD on a STLDataType Object which is not
STLnumber.
     * Otherwise it returns the value of the ADD of operands
     * @param   dt: STLDataType object
     * @return  STLException incase of error or
     *          The value of the ADD evaluation in STLDataType
     * @see STLException, STLDataType
     */
    public STLDataType ADD(STLDataType dt) {
        if(dt instanceof STLNumber){
          return new STLNumber(num + ((STLNumber)dt).num) ;
        }else
            return error(dt,"wrong datatype for addition");
    }

    /**
     * Returns a STLDataType object.The function returns an error incase a user
     * tries to perform operation SUBTRACT on a STLDataType Object which is not
STLnumber.
     * Otherwise it returns the value of the SUBTRACT of operands
     * @param   dt: STLDataType object
     * @return  STLException incase of error or
     *          The value of the SUBTRACT evaluation in STLDataType
     * @see STLException, STLDataType
     */
    public STLDataType SUBTRACT(STLDataType dt) {
        if(dt instanceof STLNumber){
            return new STLNumber(num - ((STLNumber)dt).num) ;
        }else
            return error(dt,"wrong datatype for subtraction");
    }

    /**
     * Returns a STLDataType object.The function returns the negated
     * version of num.
     * @return  The value of the -(num) evaluation in STLDataType
     * @see STLDataType
```

```
     */
    public STLDataType uminus() {
          return new STLNumber(-num);
    }

     /**
      * Returns a String.The function returns typename of STLNumber.
      * @return  "number"
      * @see STLDataType
      */
    public String typeName(){
       return "number";
  }

    public STLDataType syntaxError(String str){
       //TODO
       return null;
  }

     /**
      * The method prints out the information about the STLNumber.
      * @return  none
      * @see STLDataType
      */
      public void print( PrintWriter w ) {
           w.print( Double.toString( num ) );
           w.flush();
       }

}
```

FILE NAME: src/STLStockSource.java

```
import net.neurotech.quotes.*;
import java.util.Hashtable;
import java.lang.Math;

/**
 *
 * @author Team STL
 * The class STLStockSource utilizes screen-scraping classes
 * from the Java Financial Library.
 *
 */
public class STLStockSource{

    QuoteFactory q;
    double money;
    Hashtable portfolio;

    /**
     * Contructor for the STLStockSource Class.
     * @param   none
     * @return  none
     */
    public STLStockSource(){
        money = 0;
        portfolio = new Hashtable();
        q = new QuoteFactory();
    }


    /**
     * Returns the value of the specified stock.
     * @param   id: stock name
     * @return  double: current price
     */
    double value(String id,double time) throws QuoteException{
        return q.getQuote(id).getValue();
    }

    /**
     * Returns the percent-change of the stock today.
     * @param   id: stock name
     * @return  double: today's percent-change
     */
    double delta(String id,double time) throws QuoteException{
        return q.getQuote(id).getPctChange();
    }
```

```
/**
 * Returns the weighted market average of the stock.
 * @param   id: stock name
 * @return  double: weighted market average
 */
double wma(String id,String[] portfolio) throws QuoteException{
    return value(id,0)/1000;
}

/**
 * Buys count shares of stock id.
 * @param   id: stock name count: number
 */
void buy(int count,String id) throws QuoteException{
    double cost = value(id,0);
    if(cost*count>money){
        System.out.println("not enough money!");
    }
    else{
        money -= cost*count;
        if(portfolio.containsKey(id)){
            double have = (Double)(portfolio.get(id));
            portfolio.put(id,new Double(count+have));
        }
        else{
            portfolio.put(id,new Double(count));
        }
    }
    money = Math.rint(money*100)/100;
}

/**
 * Sells count shares of stock id.
 * @param   id: stock name count: number
 */
void sell(int count,String id) throws QuoteException{
    double cost = value(id,0);
    if(portfolio.containsKey(id)){
        double have = (Double)(portfolio.get(id));
        if(have<count){
            System.out.println("you don't have that many!");
        }
        else{
            portfolio.put(id,have-count);
            money += count*cost;
        }
    }
```

```
    else{
        System.out.println("you have none of that stock!");
    }
    money = Math.rint(money*100)/100;
}

/**
 * Get current money.
 * @return double: amount of money
 */
double money(){
    return money;
}

/**
 * Sets the current money.
 * @param   dollars: new value
 */
void money(double dollars){
    money = dollars;
    money = Math.rint(money*100)/100;
}

/**
 * Gets a list of stocks in portfolio.
 * @return String[]: list of stocks
 */
Object[] stocks(){
    if(portfolio.size()>0){
        return (portfolio.keySet().toArray());
    }
    else{
        return null;
    }
}

/**
 * Returns the number of shares owned of the given stock.
 * @param   id: stock name
 * @return  double: share count
 */
double stocks(String id){
    if(portfolio.containsKey(id)){
        return (Double)(portfolio.get(id));
    }
    else{
        return 0;
    }
```

```
    }
}
```

FILE NAME: src/STLString.java

```
import java.io.PrintWriter;

/**
 *
 * @author Team STL
 * The class STLString inherits from STLDataType, and
 * performs the manipulations required for strings.
 *
 */
public class STLString extends STLDataType {

	private String str;

	 /**
	  * Contructor for the STLString Class. Initialize str.
	  * @param   tmp
	  * @return  none
	  * @see STLDataType
	  */
	public STLString(String temp){
		str = temp;

	}

	 /**
	  * Contructor for the STLString Class.
	  * @param   name: string name
	  * @param   tmp: string to initialize str
	  * @return  none
	  * @see STLDataType
	  */
	public STLString(String name, String temp){
		super(name);
		str = temp;
	}

	/**
	 * Returns an String value str
	 * @return  String.
	 */
	public String getStr() {
		return str;
	}

	 /**
	  * Returns a STLDataType object.The function returns an error incase a user
	  * tries to perform operation EQ on a STLDataType Object which is not STLString.
```

```
     * Otherwise it returns the value of the EQ operation.
     * @param   dt: STLDataType object
     * @return  STLException incase of error or
     *          The value of the EQ evaluation in STLDataType
     * @see STLException, STLDataType
     */
    public STLDataType EQ(STLDataType dt) {
        if (dt instanceof STLString){
            return new STLBool(str.equals(((STLString)dt).str));
        }
        else
            return error(dt, "wrong datatype for EQ");
    }

     /**
     * Returns a String.The function returns typename of STLString.
     * @return  "String"
     * @see STLDataType
     */
    public String typeName(){
      return "String";
  }

    public STLDataType syntaxError(String str){          //TODO
      return null;
  }

     /**
     * The method prints out the information about the STLString.
     * @return  none
     * @see STLDataType
     */
    public void print( PrintWriter w ) {
      String temp = str.replaceAll("\\\\n","\n");
      w.print( temp );
      w.flush();
  }

/*      public STLDataType copy() {//Modified by Matt
    return new STLString( str );
  }

  public STLDataType plus( STLDataType dt ) {//Modified by Matt
    if ( dt instanceof STLString )
      return new STLString( str + ((STLString)dt).str );

    return error( dt, "+" );
  }
```

```
    public STLDataType add( STLDataType dt ) {//Modified by Matt
        if ( dt instanceof STLString )
        {
            str = str + ((STLString)dt).str;
            return this;
        }

        return error( dt, "+=" );
    }*/

}
```

FILE NAME: src/STLSymbolTable.java

```
import java.util.*;

/**
 *
 * @author Team STL
 * This class inherits HashMap is responsible for maintaining
 * scope information for the program, functions etc.
 */
public class STLSymbolTable extends HashMap {

        protected STLSymbolTable parent; //symbol table of parent.

        protected STLSymbolTable child; //symbol table of child.

        boolean readOnly;

        /**
         * Contructor for the STLSymbolTable Class. Initializes parent symboltable.
         * @param   st: Symboltable
         * @return  none
         * @see STLDataType
         */
        public STLSymbolTable(STLSymbolTable st) {
                parent = st;
                readOnly = false;
        }

        /**
         * Contructor for the STLSymbolTable Class. Initializes child and
         * parent symboltables.
         * @param   child: symboltable
         * @param   dparent: Symboltable
         * @return  none
         * @see STLDataType
         */
        public STLSymbolTable(STLSymbolTable child, STLSymbolTable dparent) {
                this.child = child;
                this.parent = dparent;
                readOnly = false;
        }

        /**
         * Returns an STLSymbolTable of parent.
         * @return  STLSymbolTable.
         */
        public STLSymbolTable getParent() {
                return parent;
```

```
}

/**
 * The function returns true or false depending on whether a variable exists in
 * the symbol table or not.
 * @param name
 * @return boolean: true indicates presence in symboltable
 */
public final boolean containsVar(String name) {
	return containsKey(name);
}

/**
 * The method goes to the required level to get the
 * correct symbol table to check for values.
 * Followed Mx Code.
 * @param level
 * @return
 */
private final STLSymbolTable gotoNestingLevel(int level) {
	STLSymbolTable st = this;

	if (level < 0) {
		// global variable
		while (null != st.getParent())
			st = st.getParent();
	} else {
		// local variable
		for (int i = level; i > 0; i--) {
			while (st.readOnly) {
				st = st.getParent();
				assert st != null;
			}

			if (null != st.getParent())
				st = st.getParent();
			else
				break;
		}
	}

	return st;
}

/**
 * The function gets the value of a particular variable from the symbol table.
 * @param name: variable name to find value.
 * @param is_static: true.
```

```
     * @param level: level of symbol table to check
     * @return value of variable of type dataType.
     */
    public final STLDataType getValue(String name, boolean is_static, int level) {
        STLSymbolTable st = gotoNestingLevel(level);
        Object x = st.get(name);

        while (x == null && null != st.getParent()) {
            st = st.getParent();
            x = st.get(name);
        }

        return (STLDataType) x;
    }

    /**
     * The function sets the value of a variable in the symboltable
     * @param name: name of variable
     * @param data: the value of the variable
     * @param is_static: true
     * @param level: level of symboltable to save into.
     * @return none
     */
    public final void setValue(String name, STLDataType data, boolean is_static,
            int level) {

        STLSymbolTable st = gotoNestingLevel(level);
        while (st.readOnly) {
            st = st.getParent();
            assert st != null;
        }

        st.put(name, data);
    }

}
```

FILE NAME: src/STLVariable.java

```
import java.io.PrintWriter;

/**
 * @author Team STL
 * This class inherits from STLDataType
 * It is responsible for variables.
 */

public class STLVariable extends STLDataType {

	/**
	 * Contructor for the STLVariable Class.
	 * @param   name: initalize Dataype name
	 * @return  none
	 * @see STLDataType
	 */
	public STLVariable(String name) {
		super(name);
	}

	/**
	 * Returns a String.The function returns typename of STLVariable.
	 * @return  "undefined-variable"
	 * @see STLDataType
	 */
	public String typename() {
		return "undefined-variable";
	}

	/**
	 * The method prints out the information about the STLVariable.
	 * @return  none
	 * @see STLDataType
	 */
	public void print(PrintWriter w) {
		w.println(getName() + " = <undefined>");
	}

	/*		public STLDataType copy() {
	 throw new STLException( "Variable " + getName() + " has not been defined" );
	 }*/


}
```