Angela_Z Final Report

2014/12/17
W4115 Programing Language and Translators

Fei Liu (fl2312)
Taikun Liu (tl2582)
Jiayi Yan (jy2677)
Mengdi Zhang (mz2472)
1. Introduction to Angela_Z

1.1 Background
1.2 Related Work
1.3 Goals of Angela_Z

2. Tutorial

2.1 Sample Code
2.2 File Organization
2.3 Compiling and Running


3.1 Language definition
3.2 Lexical Conventions
3.3 Program Structure
3.4 Type Conversions
3.5 Expressions and Operators
3.6 Declarations
3.7 Statements
3.8 System Functions

4. Project Plan

4.1 Project process
4.2 Team Responsibilities
4.3 Project Timeline

5. Architectural Design

5.1 Architecture
5.2 The Runtime Environment

6. Testing Plan

6.1 Goals
6.2 Unit test
6.3 Integration test

7. Lesson Learned

Appendix A
Appendix B
1. Introduction to Angela_Z

1.1 Background

The idea of matrix dating back to the ancient Chinese math book “Nine Chapters of the Mathematical arts” between 300 BC and AD 200. Today, we made this modern attempt to celebrate once more for the wisdom of our ancestor and help our gorgeous friends working in investing banks a little bit.

Angela is a nickname for Angel and Z stands for Zen. Zen is a kind of Buddhism belief, which calls on people to sit there doing nothing, like a sleeping angel/beauty. We hope people can take advantage of our language and take a break with their life with ease.

1.2 Related Work

Matlab has similar implementation. However, their approach of handling strings in a cell and regard everything as a matrix object is not intuitive to everyone especially our friends in financial industry. People, especially people working in financial district, prefer a way of doing the business in a flat and simple fashion. We take advantage of the stringification of any data structure related to our implementation and let the user pass in whatever data type they care to use as a String and they themselves convert back to the data type they want to use. In this case, complicated extension and inheritance is not necessary and everything falls into argument passing and processing. In this sense, not everyone is bored with the Matrix operation and they can actually play with any data type they care to use hopefully with ease and retain their instinct with primary data type from their intro programming classes, while enabling complex structures and String manipulation that they bother using.

Operators are intuitive as we keep tattooing them through adulthood or earlier. However, handy language like Java does not allow overloading of operators and the C++ way is complicated. Is there a way that differentiates the operation between Matrixes, primary types, or between Matrix and Primary type? Yes, Angela zens.

Quantitative finance is sophisticated and challenging because of our ignorance of applying certain knowledge. How to help new financial employees hook up with their job quicker and easier? We introduce an instance of utilizing our language to price options based on single one entry and based on a matrix input.
1.3 Goals of Angela_Z

As always, “Faster, Higher, Stronger”. No kidding, Faster as the complicated Matrix operations are tokened like in a linear algebra textbook; Stronger as the functionality and logic can be much stronger; and Taller as any user of this language stands on the shoulder of gigantic 4 cool coders who knows and solve problems for them before they even started using it. As always, “Go and beyond”. The extensibility of a language is very important. How to extend the usage of a language to a foreign field is important. We offer a way of customizing the language to a certain field by extending the build-in Structure type and then specify what is required for certain type of computation and further define user-defined functions that handles those pass-in’s gracefully behind the scene and the end user of those extended functionalities can then benefit from it and rest in ZEN!

2. Tutorial

2.1 Sample Code

2.1.1 “Live with Matrix”

```
Int i;
Boolean b;

Matrix main2(Int argc, String argv) {
    Matrix m3(2,2);
    Matrix m(2,2);
    m[0][0]=1;
    m[0][1]=2;
    m[1][0]=3;
    m[1][1]=4;
    m3 = (((m +. m') *. m~) *.. 4)+.. m^;
    return m3;
}

Void main(Int argc2, String m) {
```

This demo illustrate the usage of the assignment of matrix elements and usage of matrix operators and then how to nice-print the result of the computation.
First, access the elements of the matrix just like in a two dimensional array and we handle the internal structure for you.
Second, use the pre-defined operators to apply the build-in Matrix calculations.
The trick is that the plane “+”、“/” are for the primary type, the “.+”, “.-”, “.*”, and “./.” are matrix-wise operations, and the “+.,” “-.,” “.*.”, and “./.” are the mixture of the matrix and a primary type Float or Int. However, the order of the operands matters. Float or Int has to be after the Matrix in such operation. We can make it more free-stylish, but we want to coerce ordering, which is helpful in writing maintainable and easy-to-understand code, which is also what Ocaml did to us.

2.1.2 “Live with Struct”

```ocaml
Int i;
Boolean b;

Structure main2(Int argc, String argv) {
  Structure s={a="1", b= toString(argc)};
  i=toInt(s -> a);
  return s;
}

Void main(Int argc2, String m) {
  Structure result={};
  result=main2(0, "str");
  print(result);
}
```

This demo illustrate the usage of Struct type. This is a type to pass useful information only in String type. We can easily convert any type to String pass it and then convert it back. Instead of tracking the type, we give the user the opportunity to pass in anything and it will work just fine as long as a user knows how to retrieve it. This approach push the responsibility to the user and the user
can thereby choose to implement anything from their own application and
would not have nay compiler error. Also, compared with the Java extension and
inheritance approach. Our way has little overhead to be familiar with the OOP
design pattern and the rules of extension. They however assume the
responsibility to understand what they actually passed into the function and
convert it back using the right function.

2.1.3 “Application in Option”

```
Float i;
/*
   Here is the comment you don't care about!
   *

Option main2(Int argc, String argv) {
    Option s={strike="100.0", stock= "150.0", interestRate="0.1", period="1.0",
    sigma="2.0", optionType="call");
    i=toFloat(s - strike);
    return s;
}

Matrix main3(Int a) {
    Matrix strike(1,2);
    strike[0][0]=10;
    strike[0][1]=20;
    Matrix stock(1,2);
    stock[0][0]=15;
    stock[0][1]=25;
    Matrix interestRate(1,2);
    interestRate[0][0]=0.4;
    interestRate[0][1]=0.1;
    Matrix period(1,2);
    period[0][0]=3;
    period[0][1]=4;
    Matrix sigma(1,2);
    sigma[0][0]=0.1;
    sigma[0][1]=0.2;

    Matrix s(0,0);
```
s = priceM(strike, stock, interestRate, period, sigma);
return s;
}

Void main(Int argc2, String m) {
Option result={};
result = main2(0, "str");
Float d;
d = price(result);
print(d);
Matrix result1(0, 0);
result1 = main3(0);
print(result1);
}

This demo illustrates the usage of extending the Structure into customized approach and get adapted for a certain application. The financial world has many ready to use closed-form formula and the users actually don’t need to repeatedly do things over and over again. By extending just the Structure, we allow the application to be fully application based by allowing only certain parameters to pass in and customized methods to be called. This approach is enforced by allowing the operation to operate on top of Matrix and the computation can be easily loop free and whatever optimization and underground work can be possible. In this demo, we create Matrixes filled with parameters for a portfolio of options and for each of them we want to calculate a price. The parameters are passed in as a matrix and the priceM() method takes those matrix and compute the result and then the result is printed.

2.2 File Organization

Angela-z’s source code file has an extension “.az”, such as “code1.az”. The first type of compile is to do the type checking. Type checking does not yield any new files but throw exceptions when it finds errors. After type checking, you can use another command line to generate java codes files. The java codes files’ structure are:
User can then run the java file within the right folder.
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

2.3 Compiling and Running

2.3.1 Generate the code

# to make the depository clean
$ make clean
# making the compiler
$ make
# the output will be saved in java/Output.java
$./angelaZ <your_source_code>
# Copy everything in the java folder and put it into an eclipse Java-project environment to run.
Open the Output.java and run from it.

2.3.2 Test script

(PLEASE NOTE that repeating “make clean” is important at each step because the target for Makefile is different and the dependency is different.)

##### test a single file
# to make the depository clean
$ make clean
# making the compiler
$ make
# the output will be printed to the console
$./angelaZTest <your_source_code>
##### test a single file
$ make clean
# compiles
$ make
# the output will be printed to the console
$./testall.sh

2.3.3 Type checking

# to make the depository clean
$ make clean
# making the compiler
$ make
$.angelaZ -s < <your_source_code>

If semantic error happens, it will throw an exception indicating the error, such as “cannot use keyword xxx as variable name.” If no error happens, it will show that "Semantic analysis completed successfully.". This procedure does not output any file.


3.1 Language definition

3.1.1 Usage

Angela-Z is a language designed for matrix computing with build in module specializing on facilitating option pricings. It can also be used as a general-purpose programming language with similar modifications made to facilitate option pricing. (In other words, option specialization is just a demo of usage of matrix computation. The true application can be as wild as human imagination.)

3.1.2 What special feature do we have?

We support the built-in Matrix operators, Matrix I/O, and Option pricing features. Matrix-wise operations such as Plus, Minus, Multiply, Division, Transpose, Inversion, and Determinant are all supported. Facilitating functions for nice matrix printing and element accessing are also included. Option as an concrete application of using the toolbox above is provided for pricing with Black-Shores formula. The language should be able to shelter the user from intensive loops and manipulation of matrices or collection of option pricing data.

3.2 Lexical Conventions

3.2.1 Comments

Comments are delimited by {* and *}, like
3.2.2 Identifiers

An identifier is a combination of a series of characters, which includes letters, digits as well as underscore ‘_’. Identifiers can be used as name for variables and functions. The identifiers are case sensitive which means that ‘Foo’ is not the same as ‘foo’. And it can only start with letters.

There are several key words reserved from our system, detaily listed in section 2.3. Those keywords cannot be used as identifiers.

3.2.3 Keywords

The following are a list of reserved keywords in the language and can not be used as variable or function names.

if 
else 
for 
while 
return 
Boolean, true, false 
Matrix 
Structure 
Option 
Int 
Float 
String 
Void
3.2.4 Literals Types

Literal primitive types
The following table contains some examples of literal primitive types.

<table>
<thead>
<tr>
<th>Type</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Int</td>
<td>0, 1, -30, 99</td>
</tr>
<tr>
<td>Float</td>
<td>0.0, -0.002, 2e-3, 99.0</td>
</tr>
<tr>
<td>String</td>
<td>&quot;Hello World&quot;</td>
</tr>
<tr>
<td>Boolean</td>
<td>true, false</td>
</tr>
</tbody>
</table>

Int
An integer consists of a sequence of digits and no decimal point and can start with a “-” to indicate that it is negative. A positive integer has no “+” sign. Integers are represented only in decimal notation.

Float
A float can be represented in two forms. One consists of an integer part and a decimal part. The second form also contains an integer part, as well as an e or E, an integer exponent.

String
A string is enclosed in double quotation, for example "Hello World". This data type can store a string of potentially unlimited length and does not have an upper bound limit. The length is only limited by the amount of computing resources (e.g. memory) available.

Boolean
A boolean is either true or false. It has no other value.

Literal collection type
Matrix
This type only supports two dimensional matrix. A matrix data type consists of rows and columns of only floats and could not be less than 1. The elements in the matrix can be accessed by its index starting from 0, for example m[0][1] can access the element in the first row and second column of matrix m. This can store a matrix of potentially unlimited length along each dimension and does not have an upper bound limit. The size of this type is only limited by computing resources available.

### 3.2.5 Object types

**Structure**

This type is like the struct in C language. A structure can take an unlimited amount of fields or elements. But the fields can only be string type variables. The structure can not be declared without initialization. and all fields must be declared at that time. Otherwise, the structure has no fields at all. The operator -> is used to access its member fields. We do not support adding or deleting fields to an structure. However one can change the value of fields within a structure.

**Option**

This type is an extension of Structure Type. It has some built-in functions and preset (key, value) pairs. Like structures, the fields of options can only be string type variables. An option can not be declared without initialization. The operator <-> is used to access its member fields.

### 3.2.6 Operators

An operator is an evaluation performed on one or more operands. Each data type has its own set of operators.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>Performs addition</td>
</tr>
<tr>
<td>-</td>
<td>Performs minus</td>
</tr>
<tr>
<td>*</td>
<td>Performs multiplication</td>
</tr>
</tbody>
</table>
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Performs division</td>
</tr>
<tr>
<td>=</td>
<td>Performs assignment</td>
</tr>
</tbody>
</table>

### Operators for String

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Performs assignment</td>
</tr>
<tr>
<td>+</td>
<td>Performs concatenation</td>
</tr>
</tbody>
</table>

### Operators for Matrix

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+.</td>
<td>Performs elementwise addition between two matrixes</td>
</tr>
<tr>
<td>-.</td>
<td>Performs elementwise minus between two matrixes</td>
</tr>
<tr>
<td>*.</td>
<td>Performs elementwise multiplication between two matrixes</td>
</tr>
<tr>
<td>/.</td>
<td>Performs elementwise division between two matrixes, basically it is same with matrix <em>.</em>. inversion(matrix)</td>
</tr>
<tr>
<td>=</td>
<td>Performs assignment</td>
</tr>
<tr>
<td>(, )</td>
<td>Performs initialization</td>
</tr>
<tr>
<td>[ ][ ]</td>
<td>Performs index accessing</td>
</tr>
</tbody>
</table>

### Operators for Matrix and Int/Float

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>+..</td>
<td>Matrix+..Int or Float, add Int or Float to all Matrix elements</td>
</tr>
<tr>
<td>-.</td>
<td>Matrix-.Int or Float, subtract Int or Float from all Matrix elements</td>
</tr>
<tr>
<td>*.</td>
<td>Matrix*.Int or Float, multiple Matrix elements with Int or Float</td>
</tr>
<tr>
<td>/.</td>
<td>Matrix/..Int or Float, divide all Matrix elements by Int or Float</td>
</tr>
</tbody>
</table>

### Operators for Option, Structure
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>Performs assignment</td>
</tr>
<tr>
<td>{ }</td>
<td>Performs field initialization</td>
</tr>
<tr>
<td>-&gt;</td>
<td>Performs field accessing in the object</td>
</tr>
</tbody>
</table>

Operators for Boolean

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&amp;&amp;</td>
<td>Performs AND operation of two Boolean expressions</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Equality Operators

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>Test whether two expressions are equal</td>
</tr>
<tr>
<td>!=</td>
<td>Test whether two expressions are different</td>
</tr>
</tbody>
</table>

Relational Operators for only Int and Float

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=</td>
<td>Test whether left expression is smaller or equal to right expression</td>
</tr>
<tr>
<td>&lt;</td>
<td>Test whether left expression is smaller than right expression</td>
</tr>
<tr>
<td>&gt;=</td>
<td>Test whether left expression is greater or equal to right expression</td>
</tr>
<tr>
<td>&gt;</td>
<td>Test whether left expression is greater than right expression</td>
</tr>
</tbody>
</table>

### 3.2.7 Punctuators

A punctuator is a symbol that does not specify a specific operation to be performed. It has a syntactic meaning to compiler and is primarily used in formatting code. A punctuator is one of the symbols below:
3.3 Program Structure

3.3.1 Basic structure

An Angela_Z program consists of variable declaration statements and function declaration statements. A program has only one entry function which is the main function that takes an int and a string as its arguments. Variables declared outside main function are global variables. Global variables can not be assigned to any value outside the main function, i.e its initialization must take place inside main.

3.3.2 Scoping

A global variable is a variable declared outside function declarations. A global variable is accessible from all position in the file after declaration. A local variable is a variable declared inside a function declaration. A local variable's valid scoping range is only inside the blocks it's declared in.

3.4 Type Conversions

Type conversion from Int to Float is allowed. When an operation, such as + - * / < <= > >=, is conducted between a Float and an Int, the Int will be explicitly converted to Float first and then do the operation. Or an Int is given where a Float is expected, the Int will also be converted to Float. Reverse conversion from Float to Int is not allowed. Conversion for other types are not allowed. For example, Boolean can not converted to Int or vise versa.
3.5 Expressions and Operators

3.5.1 Precedence and Associativity Rule

The following table is a list of operator precedence and associativity for operators. Operators on the same row are of the same precedence and the table is in the order from highest to lowest precedence. Parentheses can be used to force precedence in the language. Initialization, i.e., Structure or Option initialization, e.g. `op { foo = “1” , bar =”2” }` and Matrix initialization, e.g. `m ( 1, 2 )`, only appears in expression which contains only itself, so there is no need to assign precedence order to initialization.

<table>
<thead>
<tr>
<th>Operator Symbol</th>
<th>Operator Description</th>
<th>Associativity</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt;</td>
<td>Structure or Option element accessing</td>
<td>Left</td>
</tr>
<tr>
<td>[] [] ()</td>
<td>Matrix element accessing</td>
<td></td>
</tr>
<tr>
<td>()</td>
<td>Function call</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>negative sign</td>
<td>Non-associative</td>
</tr>
<tr>
<td>* / .* /..</td>
<td>Times, Divide</td>
<td>Left</td>
</tr>
<tr>
<td>+ - +. -.</td>
<td>Plus, Minus</td>
<td>Left</td>
</tr>
<tr>
<td>+. -.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;= &gt;=</td>
<td>Less than, Less than or equal to, Larger than, Larger than or equal to</td>
<td>Left</td>
</tr>
<tr>
<td>== !=</td>
<td>Equality, Inequality</td>
<td>Left</td>
</tr>
<tr>
<td>&amp;&amp;</td>
<td>Logical AND</td>
<td>Left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>=</td>
<td>Assignment</td>
<td>Right</td>
</tr>
</tbody>
</table>

3.5.2 Primary Expressions

The following are all considered to be primary expressions:
Identifiers: An identifier refers to either a variable or a function. Examples include x_1 and hilbert, but not 2nd.

Constants: A constant’s type is defined by its form and value. See the table in Section 2.4.1 for examples of constants.

String literals: String literals are translated directly to Java strings by our compiler, and are treated accordingly.

Parenthesized expressions: A parenthesized expression’s type and value are equal to those of the un-parenthesized expression. The presence of parentheses is used to identify an expression’s precedence and its evaluation priority.

### 3.5.3 Function Calls

To be able to call a function, it must be declared and implemented before. Calling a function that hasn’t been defined before causes an error.

To call a function, the syntax is: `function_name(first argument, second argument,....)`. The function call returns the value of the data type defined as return type in the function declaration. Note that when calling the function, the arguments should be of the same order and the same data types as defined in function declaration. Otherwise, error occurs.

For example, if a function declaration is as follows:

```java
int diff(int a, int b)
{
    return a-b;
}
```

Valid function call is like:

```java
diff(0, 1);
```

Invalid function calls are as follows:

```java
diff(0, 1.2); /* second argument is of incorrect data type */
diff(0); /* number of arguments is different from what is defined in declaration */
```
### 3.5.4 Additive Operators

The additive operators are binary operators with left-to-right associativity:

- **Plus (+)**
- **Minus (-)**
- **Matrix Plus (+.)**
- **Matrix Minus (-.)**
- **Matrix Int/Float Plus (+..)**
- **Matrix Int/Float Minus (-..)**

Types used with the first two additive operators are Int and Float. Type conversion from Int to Float applies here if plus or minus is done between an Int and a float. The result of the plus (+) operator is the sum of the operands. The result of the minus (-) operator is the difference between the operands.

Examples are as follows:

```java
Int a;
Int b;
a = 3;
b = a + 2;
{ /* b = 5. */ }
```

Types used with the middle two additive operators are Matrix. Matrix Plus (+.) operator performs entrywise summation of elements of two matrixes. Matrix Minus (-.) operator performs entrywise subtraction of elements of two matrixes. Note that Matrix before and after operators should of same size. All elements in the matrix are treated as Float.

Examples are as follows:

```java
Matrix a(1,2);
a[0,0] = 1.0;
a[0,1] = 5.0;
Matrix b(1,2);
b[0,0] = 3.0;
b[0,1] = 3.0;
Matrix c(1,2);
```
\[ c = a - b; \]
\[
\{* c \text{ is a matrix of size 1*2: } 2.0, 2.0 *\}
\]

Types used with the last two additive operators are a Matrix and an Int/Float. Matrix Int/Float Plus (+..) operator performs entrywise summation of matrix elements and the number. Matrix Int/Float Minus (-..) operator performs entrywise subtraction of matrix elements and the number. Since matrix only contains data of Float type, Int will be explicitly converted to Float first.

Examples are as follows:

```plaintext
Matrix a(1,2);
a[0,0] = 1.0;
a[0,1] = 5;
Int b = 1;
Matrix c(1,2);
c = a +.. b;
\{* c \text{ is a matrix of size 1*2: } 2.0, 6.0 *\}
```

1.1.1. Multiplicative Operators

The multiplicative operators are binary operators with left-to-right associativity:

- Times (*)
- Divide (/)
- Matrix Times (*.)
- Matrix Divide (/.)
- Matrix Int/Float Times (*..)
- Matrix Int/Float Divide (/..)

Types used with the first two multiplicative operators are Int and Float. Type conversion from Int to Float applies here if multiplication or division is done between an Int and a float. The multiplication operator (*) yields the result of multiplying the first operand by the second. The division operator (/) yields the result of dividing the first operand by the second. Division by 0 is undefined and not allowed.

Examples are as follows:

```plaintext
Float a;
Float b;
a = 1.1;
```
\[
b = a * 0.2; \\
\{^* b = 2.2^*\}
\]

Types used with the middle two multiplicative operators are Matrix. Matrix multiplication (\(*.\)) operator performs row-by-column multiplication of two matrixes. Matrix division (\(./\)) operator performs row-by-column division of two matrixes. Note that number of columns in the Matrix before the operator equals the number of rows in the one after the operator. Dimension checking will be done and error will occur in case of dimension mismatch.

Examples are as follows:

\[
Matrix a(1,2); \\
a[0][0] = 1.0; \\
a[0][1] = 2.0; \\
Matrix b(2,1); \\
b[0][0] = 3.0; \\
b[1][0] = 2.0; \\
Matrix c(1,1); \\
c = a *. b; \\
\{^* c \text{ is a matrix of size 1}\times1: 7.0 \}^*
\]

Types used with the last two multiplicative operators are a Matrix and an Int/Float. Matrix Int/Float Multiplication (\(*..\)) operator performs entrywise multiplication of matrix elements and the number. Matrix Int/Float Division (\(/..\)) operator performs entrywise division of matrix elements and the number. Since matrix only contains data of Float type, Int will be explicitly converted to Float first.

Examples are as follows:

\[
Matrix a(1,2); \\
a[0][0] = 1.0; \\
a[0][1] = 5.0; \\
Int b = 2; \\
Matrix c(1,2); \\
c = a /.. b; \\
\{^* c \text{ is a matrix of size 1}\times2: 0.5, 2.5 \}^*
\]
3.5.5 Relational Operators

The relational operators are binary operators with left-to-right associativity:

- Less than (<)
- Greater than (>)
- Less than or equal to (<=)
- Greater than or equal to (>=)

Types used with the relational operators are Int and Float. They yield values of type Boolean. The value returned is false if the relationship in the expression is false; otherwise, the value returned is true. Relation comparison of an Int and a Float is supported and the Int will be converted to Float first.

Examples are as follows:

```java
Float a;
Int b;
a = 1.1;
b = 2;
if ( a >= 1.1 )
{
    ...
}  // a >= 1.1 returns true
if ( a < b )
{
    ...
}  // a < b returns true
```

3.5.6 Equality Operators

The equality operators are binary operators with left-to-right associativity:

- Equal to (==)
- Not equal to (!=)

Types used with the relational operators are Int and Float and String. The equal-to operator (==) returns true if both operands have the same value; otherwise, it returns false. The not-equal-to operator (!=) returns true if the
operands do not have the same value; otherwise, it returns false. Equality comparison of different types is not supported.

Examples are as follows:

```java
Float a;
String b;
a = 1.1;
b = "str";
if ( b == "st" )
{...}
if ( a == b )
{...}
/*
   b == "st" returns false
   a == b gives an error since a is Float and b is String
*/
```

### 3.5.7 Boolean Operators

The Boolean operators are binary operators with left-to-right associativity:

- **AND (&&)**
- **OR (||)**

Types used with the Boolean operators are Boolean expressions. The logical AND operator (&&) returns the Boolean value true if both operands are true and returns false otherwise. Does not support (1 && 1==1)

Examples are as follows:

```java
if ( true && false )
{...}
if ( 1 == 1 || false )
{...}
/*
   true && false returns false
   1 == 1 || false returns true since 1 == 1 is evaluated as true
*/
```
3.5.8 Assignment Operators

The assignment operators are binary operators with right-to-left associativity:

- Assign (=)

Types used with the assignment operators are Int, Float, Boolean, String, Structure, Matrix, Option, Assignment stores the value of the second operand in the first operand. Assignment between different types are not allowed. Assignment can not be done within variable declaration. A variable has to be declared first then assign.

Examples are as follows:

```java
Float a;
Float b;
a = 1.1;
b = a * 2.0;
{/*
   a is assigned value of 1.1
   b is assigned value of 2.2
 */}
```

3.5.9 Initialization Operators

The Initialization Operators have no associativity:

- Structure or Option initialization ({{ foo=", bar = " }})
- Matrix initialization (<nrows>, <ncols>)

Types used with the first initialization operator are Structure and Option. It should be of format Type Id{key=value, key=value….}. And an initializer list must exist in the declaration statement of the variable. Note that trying to access a Structure or Option that hasn’t been initialized causes error. Multiple fields are separated by a comma in the initializer list.

Examples are as follows:

```java
Option a{id="5", country= “China”};
```
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

Type used with the second initialization operator is Matrix. The first value in the parentheses is the number of rows and the second value is the number of columns. Both values have to be positive Int and use of other types or 0 or negative Int causes error. Elements in the matrix is initialized as random Float numbers. Note that trying to access a Structure or Option that hasn’t been initialized causes error.

Examples are as follows:

```java
Matrix a(1, 2);
/* a is initialized as Matrix Object with size 1*2 */
```

### 3.5.10 Element Access Operators

The Element Access Operators:

- Structure or Option element Access (->)
- Matrix element Access ([ ] [ ])

Types used with the first element access operator are Structure and Option. It returns the element with if the id indicated by the string after the operator. Before trying to access elements, existence of the element of id will be checked and error occurs if it doesn’t exist.

Examples are as follows:

```java
Option a{id="5", country= "China"};
if ( a->country == "China")
{
...
}
/* a->country returns “China” and so a->country == “China is true” */
```

Types used with the second element access operator are Matrix. It returns the object in the index position indicated by the pair [row index][column index]. User can either get the value in the position or assign new value to the element. Row and column starts from 0. An index range check is performed before trying to access elements and error occurs if index is out of range.
Examples are as follows:

```
Matrix a(1,1);
a[0][0] = 1.0

Float b;
b = a[0][0]
/* b = 1.0 */
```

3.6 Declarations

3.6.1 Function Declarations

Functions consist of a function header and a function body. The function header contains return type, function name, and parameter list. The function name must be a valid identifier. The function body is enclosed in braces. For example:

```
Int foo( Int a, Int b) { ... }
```

3.6.2 Variable Declarations

Variables are declared in the following way:

```
dataType  varName;
```

dataType could be int, float, string, boolean, void, matrix, structure, option. varName must be a valid identifier that is a combination of characters, and can only begin with letters.

Matrix, structure and option types have their own declaration convention. Matrix are declared with their dimensions specified in the parentheses and its columns and rows could not be less than 1. For structures and options, they are declared in the same way. Their member fields are enclosed in the curly braces and are separated by a comma. They must be initialized during declaration. See examples below.

The variables and constants could be initialized with a literal value, an arithmetic expression or be set equal to another variable or constant that has already been declared in this scope. The type of the value and the type of variable
it has been assigned to must match with each other. The following are some examples of variable declaration and initialization.

```plaintext
Int a;
Int a = 3;
Int a = 2 + 3;
Int b = a;
Matrix m(1,2);
Structure s { a = “Hello”, b = “World”};
Option o { a = “foo”, b = “bar”};
```

### 3.7 Statements

#### 3.7.1 Block

A block encloses a series of statements by braces.

```plaintext
{
  stmt1;
  stmt2;
  stmt3;
}
```

#### 3.7.2 Conditional Statement

There are two forms of conditional statement that consist of a “if” and an optional “else”.

```plaintext
if ( boolean expression) { stmt1 }

if ( boolean expression)  stmt1
else stmt2
```

The boolean expression will be evaluated first, and if it is true then stmt1 will be executed, otherwise, in the second case, stmt2 will be executed.
3.7.3 For Loops

For loops are in the following format:

\[ \text{for } (\text{expr1, b_expr, expr2}) \text{ stmt} \]

expr1 is evaluated only once during first iteration, the b_expr is a boolean expression and it is checked before each iteration, if it is true, then the stmt is executed. expr2 is executed in the end of each iteration. Both expr1 and expr2 have to be assignment expressions.

3.7.4 While Loops

While loops are in the following format:

\[ \text{while } (\text{b_expr}) \text{ stmt} \]

The b_expr is a boolean expression and it is evaluated before every iteration. If it is true then the statement stmt will be executed. The loop will stop when b_expr is evaluated false.

3.7.5 Return Statement

Return statement have two formats:

\[ \text{return stmt;} \]
\[ \text{return;} \]

Functions could contain a return statement, it could have another statement or just an empty return statement.
3.8 System Functions

3.8.1 print() Function

The print function is used to output a variable’s value or a string literal. For example:

```c
Int a = 3;
print (a);
{ /* 3 */

print ("Hello World!");
{ /* Hello World! */
```

3.8.2 printM() Function

Print the Matrix. For example:

```c
Matrix m(1,1);
m[0][0]=4.5;
printM(m);
```

3.8.3 toInt() Function

Convert a String to a Int if the format is correct. For example:

```c
String a="1";
print(toInt(1)+1);
```

3.8.4 toFloat() Function

Convert a String to a Float if the format is correct. For example:

```c
String a="1.5";
print(toFloat(1)+1.0);
```
3.8.5 toBoolean() Function

Convert a String to a Boolean if the format is correct. For example:

```java
String a="false";
print(toBoolean(a));
```

3.8.6 toString() Function

Convert any object with type other than Void into String. For example:

```java
Matrix a(1,1);
print(toString(a));
Option b={myname="1"};
print(toString(b));
```

3.8.7 price() Function

Price for a well-defined option. For example:

```java
Option main2(Int argc, String argv) {
  Option s={strike="100.0", stock="150.0", interestRate="0.1", period="1.0", sigma="2.0",
    optionType="call"};
i=toFloat(s -> strike);
return s;
}
```

4. Project Plan

Our project would not be finished without a detailed plan. The following plan was made at the beginning of our project to outline the main procedures of the project and specify where we were along the road.
4.1 Project process

4.1.1 Planning and Designing

The first phase of the project is coming up with the objective and basic design of the language, and making developing plans. The first draft of our LRM is the blue print of our language, we kept it up-to-date when we made modification to the language grammar. We meet on a weekly base to discuss the current status and next step of the project.

4.1.2 Development

We divided our developing phase into four major steps each along with their own testing process. The first step is to design AST, parser and scanner. Then the second step is to do type and scope checking by defining a SAST. The third step is the implementation of our code generator which generates the target code from our own language. The final step is using our language to develop some small domain specific applications that can demonstrate the advantages of our language.

4.1.3 Testing

Testing is done within every step along the progress to get feedback from every developing phase. Testing each phase individually and then test the whole integrated project ensured our project’s validity while it is growing. Testing plans are discussed in detail in chapter 6.

4.2 Team Responsibilities

Fei Liu: Language guru  
Taikun Liu: System architect  
Jiayi Yan: Manager  
Mengdi Zhang: Verification and validation

4.3 Project Timeline

The following time table recorded our main milestones. We regard these as deadlines for development.
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

Software Development Environment
Our team developed out project on Mac OS X using OCaml 4.0 and OCamlyacc. And we used github for version control. We also wrote a shell script for running the test cases.

Project Logs and Personal Contributions
The following table lists actual progress in our development.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sep-10-2014</td>
<td>Team formed</td>
</tr>
<tr>
<td>Sep-13-2014</td>
<td>Discussion of Language Objective</td>
</tr>
<tr>
<td>Sep-20-2014</td>
<td>Proposal Completed</td>
</tr>
<tr>
<td>Oct-04-2014</td>
<td>Parser and Lexer, first working version</td>
</tr>
<tr>
<td>Oct-11-2014</td>
<td>LRM first version</td>
</tr>
<tr>
<td>Oct-25-2014</td>
<td>LRM completed</td>
</tr>
<tr>
<td>Nov-01-2014</td>
<td>Parser and lexer updated</td>
</tr>
<tr>
<td>Nov-15-2014</td>
<td>SAST, TypeChecking, first working version</td>
</tr>
<tr>
<td>Nov-22-2014</td>
<td>Code generation, first version</td>
</tr>
<tr>
<td>Dec-8-2014</td>
<td>Final report started</td>
</tr>
<tr>
<td>Dec-15-2014</td>
<td>Testing completed</td>
</tr>
<tr>
<td>Dec-17-2014</td>
<td>Final report completed</td>
</tr>
</tbody>
</table>

The following table lists personal contributions to this project

<table>
<thead>
<tr>
<th>Module</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scanner, Parser, AST</td>
<td>Jiayi Yan (major); Fei Liu; Taikun Liu</td>
</tr>
<tr>
<td>SAST, Typecheck</td>
<td>Taikun Liu</td>
</tr>
<tr>
<td>javagen, java codes</td>
<td>Fei Liu</td>
</tr>
</tbody>
</table>
5. Architectural Design

5.1 Architecture

As can be seen, the steps can be divided into the following sections:
1. Scanning
2. Parsing/AST
3. Typechecking/SAST
4. Java code generation

Scanning
The Scanner takes a file with extension of "az" as input file and tokenizes the characters into meaningful symbols. Tokens include keywords, constant literals and operators used by angelaZ language. The process consist of discarding whitespace and comments and combining remaining character streams into tokens. Illegal character combination errors are caught here. The scanner is built with OCaml lexer.

**Parsing/AST**

The parser takes tokens provided by the scanner and generates an abstract syntax tree (AST). The process consists of defining operator precedence, pattern matching and pattern reduction. Syntax errors are caught here. The parser is built with OCaml yacc program.

An AST is an intermediate representation of the input file "sample.az" after parsing before semantic checking.

**Type checking/SAST**

Type checking takes AST provided by parsing and generates a type-safe semantic abstract syntax tree (SAST). The process consists of walking through the AST tree, checking and generating information such as type matching and scoping table. Semantic errors, such as type mismatching and using undeclared variables or functions are caught here.

For our project, since we have to check whether a field within a structure or option has been declared ahead. We use a different way to walk through the AST.

Our program consists of global statements and functions. So we first check global statements and function headers (without checking the statements inside functions at first). Secondly, our checking starts with main function. Checking all statements inside main function. If one statement calls another function, we then check whether that function exists, arguments match formals, arguments are well defined before go into that function to check that function's body statement. Consequently, functions that are not called will not be checked for their body statements.

An SAST is an intermediate type-safe representation of the input file "sample.az" after semantic checking.

**Java code generation**

Java code generation walks through SAST provided by type checking and generates a runnable java file “output.java”. Errors found during evaluation of
expressions, such as invalid index value -1, are caught here. Although this module generates java code, it does not compile it.

The output file “output.java” is a ready-to-run error-free Java file and can return the final result after being compiled by Java compiler.

5.2 The Runtime Environment

The runtime environment provides the infrastructure of operations in the generated java file. A set of runtime Java classes consist of implementation classes: “Structure.java”, “Option.java”, “MatrixMathematics.java” and “OptionMatrix.java”; exception handling classes: “NoSquareException.java” and “IllegalDimensionException.java”; print classes: “ToString.java”. The driver class is the Output.java. As dependencies, Matrix, Structure, and Option cover the types of the language and operations they have for only one such instance but not the interactions between them. For instance, Matrix multiplication is not covered because it is between Matrixes. The MatrixMathematics.java is the file that invoke the java methods to do the actual computation. OptionMatrix.java is the file that contains the code for the Matrix-wise option pricing. The Exception suffix files are there for the Exceptions we may encounter. MatrixMathematics.java defines the operator methods that are between Matrixes.

The class declaration without the content of the classes is listed as follows:

```java
public class Structure {
}

public class Option extends Structure {
}
public class MatrixMathematics {
}

public class OptionMatrix {
}

public class NoSquareException extends Exception {
}

public class IllegalDimensionException extends Exception {
}

public class ToString {
}
```
6. Testing Plan

6.1 Goals

Test cases are carefully selected to test all the development phases thoroughly without which the quality of the project is not ensured. The whole development cycle is decomposed into four majors phases and we aimed at designing unit test on each modules and then run integration and system test on the whole integrated project. By running tests frequently, we could catch any backwards-incompatible changes during development. This system also allows test-driven development, as the test suite can be run with tests that use unimplemented language features, simply failing until those features are implemented.

6.2 Unit test

The aspects that we designed tests on during unit test are:
Variable declaration for each data type:
Int, Float, String, Boolean, Void, Matrix, Structure, Option
Legitimacy of identifiers

Variable assignment for each data type:
variable assigned by literals
variable assigned by variable
arithmetic assignment

Function declaration and calling:
main function
system function
parameter list

Operators:
arithmetic operators
boolean operators
structure access
option access
matrix access
Program structure:
  - global variable declaration
  - entrance function and user defined function

Statements:
  - if/else statements
  - for loops
  - while loops

Type checking:
  - for declarations
  - for operators
  - for statements
  - matrix dimension matching

Scoping checking:
  - uninitialized variable usage
  - global and local variable usage

Others:
  - comments
  - type conversion
  - precedence

### 6.3 Integration test

We designed test cases for integrated testing Parser/Scanner with type checking. Also we tested the correctness of semantic checking and code generation models together with the parser. We generated top level executable for semantic checking and code generation. The output of semantic checking is the SAST which is the SAT with types of all the variables resolved. The output of code generation is the java code.

### 7. Lesson Learned

When designing languages, syntax matters and one thing about syntax that shouldn't be considered is minimizing the number of keywords. The metric may cause much difficulty in distinguishing between language rules in parsing. As a programmer, you are either writing code or fixing it. However, fixing code in OCaml can be painful, since debugging info is not informative when it comes to
shift/reduce error. Therefore, it’s a wise choice to first design a language structure as comprehensive as possible, then begin with a small part of the functionality and add new features after making sure the codes you’ve written are correct.

Even the best programmer cannot remember everything, so write it down. When discussing tokens, operators, logic and everything else about the language, write it down in good format. Order all rules neatly in Language Reference Manual before starting to code so that whenever you have doubt about anything rule, you can always check the LRM for reference.

Frequent and effective communication is a key to success of the project. Meet each week with your teammates to discuss problems you’re having, the plan for the next week and so on. Also, meeting with TA each week to discuss the project is also helpful.

Some tools are recommended for the project, Github for version control and automated testing framework for debugging.

Last but not least, after finally coming up with a prototype next comes the hardest part, selling the language. A language no one uses is not a language, so prepare your presentation, documentation and tutorials well to show it off.
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.
Appendix A

Test Cases

**test-1.az**

```c
Int main (Int c, String s) {
}
```

**test-2.az**

```c
Int i;
Boolean b;
Matrix m2(1,1);
Matrix m3(1,1);

Int main(Int argc, String argv) {
    Structure s = { a = "1", b = "2", c = "3" }; // Removed the semicolon
    Matrix m(1,1);
    m3 = (m + m2) * 4;
    Option o = {a = "1"};
    o -> a;
    argv = toInt(s -> a);
    if (i == 2 || i > 0) {
        return argv;
    } else {
        return i + 1;
    }
    while (i >= 2 || (i < 100 && i != 3))
        argv = "argv";
    for (i = 0; i < 5; i = i + 1) {
        i = i * (5 + i);
    }
}
```

**test-3.az**

```c
Int main(Int argc, String argv){
    Int a;
}
```

**test-4.az**

```c
Boolean b1;
Boolean b2;
Boolean b3;
```
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

Boolean b4;
Int main(Int argc, String argv){
  b1 = true;
  b2 = false;
  b3 = b1 && b2;
  b4 = b1 || b2;
}

test-5.az
{*
  hello
*}
Int a;
{**}
Int main(Int argc, String argv){

  hello
  world
*}
}
test-6.az
Float f1;
Float f2;
Float f3;
Int main(Int argc, String argv){
  f1 = 1;
  f1 = 0.2;
  f1 = 1e10;
  f1 = 2E15;
  f1 = 2e-9;
  f1 = 2e+12;
  f1 = 2E-0;
  f1 = 2E+12;
  
  f2 = f1 + 1;
  f2 = f1 + 1.5;
  f2 = 1.5 + f1;
  f2 = f1 + 19e-10;
  f2 = 1.5 + 19e10;
  
  f2 = 9 - f1;
  f2 = 10 - 9.3;
  f2 = 9e10 - 8E+2;

  f3 = f1 * f2;
  f3 = f1 * 2;
f3 = f1 * 10e12;
f3 = 10e12 * f1;
f1 = 3e-19 * 1.5;

f1 = f2 / 2;
f1 = f2 / 1.5;
f1 = 1.5 / f2;
f1 = f2 / 9e10;
f1 = 9e10 / 5E2;
f1 = 9e10 / 1.6;
}

**test-7.az**
Int main(Int argc, String argv){
  Int a;
  Boolean b1;
  b1 = true;
  for (a = 0; a < 10; a = a + 1){
    Int b;
    b = 1;
  }
  for (a = 0; a < 10; a = a + 1)
    Int c;

  for (a = 0; b1; a = a + 1){
    Int d;
    d = 1;
  }
}

**test-8.az**
Int main(Int argc, String argv){
  Int asdf;
  Int ASDF;
  Int A0a9;
  Int asdf__;
  Int asdf_ASDF;
}

**test-9.az**
Int a;
Int b;
Int c;
Int main(Int argc, String argv){
  b = 2;
a = 10;
if (a > b){
    c = 1;
}
else
    c = 0;

if (a > b)
    c = 1;
else
    c = 0;

if (a > b)
    c = 1;

if( a > b)
    c = 1;
else {
    c = 0;
}
if (a >= b )
    c = 1;
if (a < b)
    c = 1;
if (a <= b)
    c = 1;
if (a == b)
    c = 1;
if (a != b)
    c = 1;

Boolean b1;
Boolean b2;
b1 = true;
b2 = false;
if (b1)
    c = 1;
if (b1 && b2)
    c = 1;
if (b1 || b2)
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

c = 1;
}

**test-10.az**

```c
int main(int argc, string argv) {
    int a;
    a = 3;

    int b;
    b = a + 5;
    b = 5 + a;

    int c;
    a = b - 1;
    a = 1 - b;
    c = a - b;

    c = b;
    c = b * 3;
    c = a * b;

    b = a / 2;
    c = a / b;
    c = 1 / a;
}
```

**test-11.az**

```c
int main(int argc, string argv) {
    int a;
    int b;
    int c;

    matrix m1(2, 2);
    matrix m2(2, 2);
    matrix m3(2, 2);

    m1[1][1] = 2;
    float a;
    a = m1[1][1];
    m2[1][1] = m1[1][1];

    m3 = m1 + m2;
    m2 = m2 - m3;
    m1 = m2 * m3;
    m3 = m1 / m2;
}
```

**test-12.az**

```c
int main(int argc, string argv) {
    int a;
    int b;
    int c;
```
Boolean b1;
Boolean b2;
Boolean b3;

a = 2;
b = 1;

c = (a + b) * 3;
c = (a + b + 3) * 2;
c = 2 * (a + b);
c = a + (1 + a);
c = (1 + a) * (1 + b);
c = (a + b);

b1 = true;
b2 = false;
b3 = b1 && (b1 || b2);
b3 = (b1 && a && b1);
}

**test-13.az**
Int main(Int argc, String argv){
String s;
s = " ",
s = " ",
s = "asdf",
s = "!@asf#$%^as325df34&*()1sfd",
s = "as\nasd",
s = "asd\r",
}

**test-14.az**
Int main(Int argc, String argv){
Int a;
Float b;
String c;
Matrix m(1,2);
Option o = {};
Structure s = {};
Boolean boo;
Void v;
}

**test-15.az**
Int main(Int argc, String argv){
Int a;
a = 5;
Boolean b1;
Boolean b2;
b1 = true;
b2 = false;

while (a > 0){
a = a - 1;
}

while (b1 && b2)
a = 10;
}

test-16.az
{" structure declaration *
Int main(Int argc, String argv){
Structure s2 = { a="" };  
Structure s3 = { a="a" };  
String str;
str = "str";
Structure s4 = { a=str };  
Structure s5 = { a=str, b = "b"};
}
test-17.az
{" structure access *
Int main(Int argc, String argv){
String str;
str = "str";
Structure s = { a=str, b="b"};  
s->a = s->b;
}
test-18.az
{" option declaration *
Int main(Int argc, String argv){
Option o2 = { a="" };  
Option o3 = { a="a" };  
String str;
str = "str";
Option o4 = { a=str };  
Option o5 = { a=str, b = "b"};
}
test-19.az
{" option access *

Int main(Int argc, String argv){
  String str;
  str = "str";
  Option o = { a=str, b="b"};
  o->a = o->b;
}

fail-1.az
Int main(Int argc, String argv){
  Structure s = { a = "sdf"};
  foo(s);
}

Void foo (Structure s){
  print(s->b);
}

{*Fatal error: exception Failure("Field b does not exist within struct")*}

fail-2.az
Int main(Int argc, String argv){
  Int a;
  a = "asf";
}

{*Fatal error: exception Failure("variable a need to be assigned with same type int string")*}

fail-3.az
Int main(Int argc, String argv){
  Int a;
  a = 1.5;
}

{*Fatal error: exception Failure("variable a need to be assigned with same type int float")*}

fail-4.az
Int main(Int argc, String argv){
  Int a;
  a = asd;
}

{*Fatal error: exception Failure("Cannot find variable named asd")*}

fail-5.az
Int main(Int argc, String argv){
  a = 2;
}
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

{Fatal error: exception Failure("Cannot find variable named a")}  
fail-6.az  
Int main(Int argc, String argv){  
a = 1.2;  
}

{Fatal error: exception Failure("Cannot find variable named a")}  
fail-7.az  
Int main(Int argc, String argv){  
a = true;  
}

{Fatal error: exception Failure("Cannot find variable named a")}  
fail-8.az  
Int main(Int argc, String argv){  
Float a;  
a = true;  
}

{Fatal error: exception Failure("variable a need to be float type")}  
fail-9.az  
Int main(Int argc, String argv){  
Float a;  
a = asd;  
}

{Fatal error: exception Failure("Cannot find variable named asd")}  
fail-10.az  
Int main(Int argc, String argv){  
Float a;  
a = "sasdf";  
}

{Fatal error: exception Failure("variable a need to be float type")}  
fail-11.az  
Int main(Int argc, String argv){  
Boolean b;  
b = 0;  
}

fail-12.az  
Int main(Int argc, String argv){  
Boolean b;  
b = asd;  
}
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

```java
fail-13.az
Int main(Int argc, String argv){
String s;
s = 'asd';
}
fail-14.az
Int main(Int argc, String argv){
String s;
s = 12;
}
fail-15.az
String s;
s = True;
fail-16.az
Int a;
Float a;
fail-17.az
Int main(Int argc, String argv){
Int a;
Int b;
b = a+1.0;
}
fail-18.az
Int main(Int argc, String argv){
Float a;
a = 2;
Int f;
f = a;
}
fail-19.az
Int main(Int argc, String argv){
Int a;
a = 0;
Boolean b;
b = a;
}
fail-20.az
Matrix m;
fail-21.az
Int main(Int argc, String argv){
Matrix m(0,0);
```
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.

```java
Int main(Int argc, String argv){
    Matrix m (-1,2);
}

fail-22.az
Int main(Int argc, String argv){
    Option o;
    o = {a = "asdf"};
}

fail-23.az
Int main(Int argc, String argv){
    Option o = { a = "asdf"; b = a;};
}

fail-24.az
Int main(Int argc, String argv){
    Option o = {a = "sdf"};
    o->a = 1;
}

fail-25.az
Int main(Int argc, String argv){
    Option o {a="asdf"};
    o->b = "ad";
}

fail-26.az
Int main(Int argc, String argv){
    Option o {a=1};
}

fail-27.az
Int main(Int argc, String argv){
    Option o { a="asdf "};
    o->b = "ad";
}

fail-28.az
Int main(Int argc, String argv){
    Option o {a};
    o->a = "asdf";
}

fail-29.az
Int main(Int argc, String argv){
    String str = "asdf";
    Option o [str];
}

fail-30.az
Int main(Int argc, String argv){
    Option o = {a = "ad"};
    Option o1;
    o1 = o;
}
```
fail-31.az
Int main(Int argc, String argv) {
    String s;
    s = "sdf";
    String s1;
    s1 = s;
    String s2 = s1 + s;
}

fail-32.az
Int main(Int argc, String argv) {
    String s;
    s = "sdf";
    String s1;
    s1 = s - "asdf";
}

fail-33.az
Int main(Int argc, String argv) {
    String s;
    s = "sdf";
    String s1;
    s1 = s / 1;
}

fail-34.az
Int main(Int argc, String argv) {
    String s;
    s = "sdf";
    String s1;
    s1 = s * 2;
}
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.
AngelaZ: Make it a world of Zen. Zen stands for peace and rest.
Appendix B

Code:
1{ 
2  open Lexing
3  open Parser
4}
5
let digit = ['0'..'9']
6let frac = "." digit*
7let exp = ['e' | 'E' | 'e' | 'E']? digit+
8let float = ['\'\'']? digit* frac? exp?

10rule token = parse
11 ["\"\[\""] [ token lexbuf ] (* Whitespace *)
12 "[\"] { comment lexbuf } (* Comments *)
13 | 'C' [ LPAREN ] (* precedence, matrix initialization, ifelseforwhile *)
14 | 'R' [ RPAREN ]
15 | '{' [ LRBRACE ] (* block of statements *)
16 | '}' [ RRBRACE ]
17 | '"' [ LSQUARE ] (* matrix index access *)
18 | '[' [ RSQUARE ]
19 | '"' [ SEMI ]
20 | '"' [ COMMA ]
21 | '"' [ PLUS ]
22 | '"' [ MPLUS ] (* plus matrix and matrix *)
23 | '"' [ MIPLUS ] (* plus matrix and int/float *)
24 | '"' [ MINUS ]
25 | '"' [ MMINUS ]
26 | '"' [ MMINUS ]
27 | '"' [ TIMES ]
28 | '"' [ MTIMES ]
29 | '"' [ MITIMES ]
30 | '"' [ DIVIDE ]
31 | '"' [ MIDIVIDE ]
32 | '"' [ MITIMES ]
33 | '"' [ ASSIGN ]
34 | '"' [ AND ]
35 | '"' [ OR ]
36 | '"' [ EQ ]
37 | '"' [ LT ]
38 | '"' [ LEQ ]
39 | '"' [ GT ]
40 | '"' [ GEQ ]
41 | '"' [ MPLUS ]
42 | '"' [ MMINUS ]
43 | '"' [ MINUS ]
44 | '"' [ PLUS ]
45 | '"' [ MMINUS ]
46 | '"' [ TIMES ]
47 | '"' [ MMINUS ]
48 | '"' [ MMINUS ]
49 | '"' [ MMINUS ]
50 | '"' [ MMINUS ]
51 | '"' [ MMINUS ]
52 | '"' [ MMINUS ]
53 | '"' [ MMINUS ]
54 | '"' [ MMINUS ]
55 | '"' [ MMINUS ]
56 | '"' [ MMINUS ]
57 | '"' [ MMINUS ]
58 | '"' [ MMINUS ]
59 | '"' [ MMINUS ]
60 | '"' [ MMINUS ]
61 | '"' [ MMINUS ]
62 | '"' [ MMINUS ]
63 | '"' [ MMINUS ]
64 | '"' [ MMINUS ]
65 | '"' [ MMINUS ]
66 | eof [ EOF ]
67 | _ { raise (Failure("illegal character ")) }
68
69and comment = parse
70 '"' [ token lexbuf ]
71 _ [ comment lexbuf ]
1% open Ast %
2
3/* token */
4%token SEMI COLON LPAREN RPAREN L SQUARE RSQUARE LBRACE RBRACE COMMA
5%token PLUS MINUS MPLUS MMINUS MIPLUS MMINUS TIMES DIVIDE MTIMES MDIVIDE MITIMES MITDIVIDE ASSIGN ARROW
6%token EQ NEQ LT LEQ GT GEQ AND OR
7%token RETURN IF ELSE FOR WHILE
8%token BOOLEAN TRUE FALSE MATRIX STRUCTURE OPTION INT FLOAT STRING VOID
9%token TRANSPOSE INVERSION DETERMINANT
10%token <int> INT_LIT
11%token <float> FLOAT_LIT
12%token <string> STRING_LIT
13%token <string> ID
14%token EOF
15
16/* associativity, precedence */
17%nonassoc NOELSE
18%nonassoc ELSE COMMA
19%right ASSIGN
20%left OR
21%left AND
22%left EQ NEQ
23%left LT GT LEQ GEQ
24%left PLUS MINUS MPLUS MMINUS MIPLUS MMINUS
25%left TIMES DIVIDE MTIMES MDIVIDE MITIMES MDIVIDE
26%left ARROW
27%nonassoc TRANSPOSE INVERSION DETERMINANT
28
29%start program
30%type <Ast.program> program
31
32%%
33
34/* program consists of statements and function declarations */
35program:
36  { [], [] }
37  | program stmt { [$2 :: fst $1], snd $1 }
38  | program fdecl { fst $1, ($2 :: snd $1) }
39
40 /* function declaration */
41fdecl:
42  retval formal_list RPAREN LBRACE stmt_list RBRACE
43  { { func_name = $1.vname;
44     formal = list.rev $2;
45     body = list.rev $5;
46     ret = $1.vtype
47     } }
48
49 /* return structure */
50retval:
51  INT LPAREN { { vtype=Int; vname=$2 } }
52  FLOAT LPAREN { { vtype=Float; vname=$2 } }
53  VOID LPAREN { { vtype=Void; vname=$2 } }
54  MATRIX LPAREN { { vtype=Matrix; vname=$2 } }
55  OPTION LPAREN { { vtype=Option; vname=$2 } }
56  STRUCTURE LPAREN { { vtype=Structure; vname=$2 } }
57  BOOLEAN LPAREN { { vtype=Boolean; vname=$2 } }
58  STRING LPAREN { { vtype=String; vname=$2 } }
59
60formal_list:
61  /* nothing */ { [] }
62  | formal { [$1] }
63  | formal_list COMMA formal { $3 :: $1 }
64
65 /* function arguments */
66formal:
67  tdecl { $1 }
68  | MATRIX { { vname=$2; vtype=Matrix } }
69  | OPTION { { vname = $2; vtype = Option } }
70  | STRUCTURE { { vname = $2; vtype = Structure } }
71
72 /* primitive variable type declaration */
73tdecl:
74  INT { { vname = $2; vtype = Int } }
75  FLOAT { { vname = $2; vtype = Float } }
76  VOID { { vname = $2; vtype = Void } }
77  BOOLEAN { { vname = $2; vtype = Boolean } }
78  STRING { { vname = $2; vtype = String } }
79
80 /* matrix declaration */
81mdecl:
82  MATRIX LPAREN INT_LIT COMMA INT_LIT RPAREN { { vname = $2; vtype = Matrix; mrow = $4; ncol = $6 } }
83
84stmt_list:
85  /* nothing */ { [] }
86  | stmt_list stmt { $2 :: $1 }
87
88 /* structure/option field declaration */
89struct_arg:
struct arg_list:
  /* nothing */
  struct arg [($1)]
  struct arg_list COMMA struct arg [($3::$1)]

stmt:
  expr SEMI [Expr($1)]
  RETURN expr SEMI [Return($2)]
  RETURN SEMI [ReturnNoexpr($1)]
  LBRAKE stmt_list RBRAKE [Block(list.rev $2)]
  IF LPAREN expr RPAREN stmt [If$3, $5, Block([])]
  IF LPAREN expr RPAREN stmt ELSE stmt [If$3, $5, $7]
  FOR LPAREN expr_opt SEMI expr SEMI expr_opt RPAREN stmt
  FOR($3, $5, $7, $9)
  WHILE LPAREN expr RPAREN stmt [While($3, $5)]
  ddec SEMI [Verdec($1)]
  ddec SEMI [Matdec($1)]
  STRUCTURE ID ASSIGN LBRAKE struct arg_list RBRAKE SEMI [Structdec($2, $5)]
  OPTION ID ASSIGN LBRAKE struct arg_list RBRAKE SEMI [Optiondec($2, $5)]
type dataType =
  | Int
  | Float
  | String
  | Matrix
  | Option
  | Structure
  | Boolean
  | Void

type var_dec = {
  | name : string;
  | vtype : dataType;
}

type mat_dec = {
  | name : string;
  | mtype : dataType;
  | mrow : int;
  | mccl : int;
}

type bin_op = Add | Sub | Times | Divide | And | Or | Eq | Neq | Lt | Gt | Leq | Geq

type mat_op = MTime | MDivide | MAdd | MSub | MInversion

type mat_uop = MTriangular | MInversion | MDeterminant

type expr =
  | Binary_op of expr * bin_op * expr
  | MatBinary_op of expr * mat_op * expr
  | Id
  | Int_lit of int
  | String_lit of string
  | Call of string * expr list
  | VVarAssign of string * expr
  | Matrix_element_assign of string * expr * expr * expr
  | Struct_element_assign of string * string * expr
  | Precedence of expr
  | Struct_element of string * expr
  | Bool_lit of int
  | Int_lit
  | Float_lit
  | MatUnary_op of expr
  | MatDec of var_dec
  | VDecl of var_dec
  | MDecl of mat_dec
  | VType of vtype
  | MType of mtype
  | VName of vname

string_of_dataType vdecl

string_of_vdecl vdecl

string_of_mat mdecl

string_of_mdecl mdecl

string_of_mat mdec

string_of_vdec vdec

string_of_vdecl vdecl

string_of_mdecl mdecl

string_of_mat mdec

string_of_vdec vdec

string_of_vdecl vdecl

string_of_mat mdec

string_of_vdec vdec

string_of_mdecl mdecl

let string_of_dataType = function
  | Int -> "int"
  | Float -> "float"
  | String -> "string"
  | Matrix -> "matrix"
  | Option -> "option"
  | Structure -> "structure"
  | Boolean -> "bool"
  | Void -> "void"

let string_of_vdecl vdecl = vdecl.vtype ^ " " ^ vdecl.name

let string_of_mdecl mdecl =
let rec string_of_expr = function
  | IntLit(i) -> "" ^ i ^ "\n"
  | StringLit(s) -> "\" ^ s ^ "\n"
  | Id(s) -> s
  | BinaryOp(e1, o, e2) ->
    string_of_expr e1 ^ o ^ string_of_expr e2
  | Match o with
    | Add -> "+" | Sub -> "-" | Times -> "*" | Divide -> "/" | Eq -> "=" | Neq -> "\!=" | Lt -> "<" | Leq -> "\leq" | Gt -> ">" | Geq -> "\geq"
  | And -> "\&\&" | Or -> "||"
  | A "" ^ string_of_expr e1
  | MatBinaryOp(e1, o, e2) ->
    string_of_expr e1 ^ o ^ string_of_expr e2
  | (match o with
    | NAdd -> "-+" | NSub -> "--+" | NTime -> "-\times" | NDiv -> "-\div"
    | | |)
    | Stringconcat,
    | ) (List.map string_of_expr el) (^ ^ "")
  | Call(f, el) -> f (^ ^ "(" ^ Stringconcat,
              (List.map string_of_expr el) (^ ^ "")
  | Noexpr -> "void"
  | PrecedenceExpr(e) -> (^ ^ "string_of_expr e ^ "")
  | StructElementAssign(id, i, e, e2) ->
    id (^ ^ "string_of_expr r ^ "^ ^ "string_of_expr c ^ "") ^ "string_of_expr e
  | MatrixElementAssign(id, r, c, e) ->
    id (^ ^ "string_of_expr r ^ "^ ^ "string_of_expr c ^ "") ^ "string_of_expr e
  | For (e1, e2, s) ->
    For (^ ^ "string_of_expr e1 ^ "^ ^ "string_of_expr e2 ^ "") ^ "string_of_expr s
  | While (e, s) ->
    While (^ ^ "string_of_expr e ^ "") ^ "string_of_expr s
  | If (el, e1, e2, s) ->
    If (^ ^ "string_of_expr e ^ "") ^ "string_of_expr s
  | Match (e, el) ->
    Match (^ ^ "string_of_expr e ^ "") ^ "string_of_expr s
  | List (mname, mrow) ->
    List (^ ^ "\[" ^ element_id (^ ^ "\] = " ^ string_of_expr mname (^ ^ "\] = " ^ string_of_expr mrow)
  | Map (string_of_expr mname, string_of_expr mrow) ->
    Map (^ ^ "\{" ^ element_id (^ ^ "\} = " ^ string_of_expr mname (^ ^ "\} = " ^ string_of_expr mrow)
  | BlockStmts (stmts) ->
    Block (^ ^ "\{\n" ^ Stringconcat,
            (List.map string_of_stmt stmts) (^ ^ "\}\n")
  | Expr (expr) ->
    Expr (^ ^ "string_of_expr e ^ "")
  | IfExpr (e, s1, s2) ->
    If (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt s1 ^ "else ^ "string_of_stmt s2
  | ForExpr (e1, e2, e3, s) ->
    For (^ ^ "string_of_expr e1 ^ "") ^ "string_of_expr e2 ^ "") ^ "string_of_stmt s
  | WhileExpr (e, s) ->
    While (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt s
  | ListExpr (mdecl) ->
    List (^ ^ "\{\n" ^ string_of_mdecl mdecl (^ ^ "\}"
  | MapExpr (mname, mrow) ->
    Map (^ ^ "\{" ^ mtype (^ ^ "\} = " ^ string_of_int mname (^ ^ "\} = " ^ string_of_int mrow)
  | BlockStmt (stmts) ->
    Block (^ ^ "\{\n" ^ Stringconcat,
            (List.map string_of_stmt stmts) (^ ^ "\}\n")
  | IfStmt (e, stmts) ->
    If (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | ForStmt (e, stmts) ->
    For (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | WhileStmt (e, stmts) ->
    While (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | MatchStmt (e, el) ->
    Match (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt el
  | ListStmt (mdecl) ->
    List (^ ^ "\{\n" ^ string_of_mdecl mdecl (^ ^ "\}"
  | MapStmt (mname, mrow) ->
    Map (^ ^ "\{" ^ mtype (^ ^ "\} = " ^ string_of_int mname (^ ^ "\} = " ^ string_of_int mrow)
  | Block (stmts) ->
    Block (^ ^ "\{\n" ^ Stringconcat,
            (List.map string_of_stmt stmts) (^ ^ "\}\n")
  | If (e, stmts) ->
    If (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | For (e, stmts) ->
    For (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | While (e, stmts) ->
    While (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt stmts
  | Match (e, el) ->
    Match (^ ^ "string_of_expr e ^ "") ^ "string_of_stmt el
open Ast
open Sast

(****************************************************************************** Collection symbol table*)
4
5 type struc_table = {
6  struc_name : string; (*name of a structure*)
7  mutable struc_fields : string list; (*keys within a structure*)
8 }
9 type option_table = {
10  option_name : string; (*name of an option*)
11  mutable option_fields : string list; (*keys within an option*)
12 }
13 type size_of_matrix = {
14  rows : int; (*number of rows*)
15  cols : int; (*number of cols*)
16 }
17 type matrix_table = {
18  matrix_name : string; (*name of a matrix*)
19  nsize : size_of_matrix; (*size of a matrix*)
20 }
21 (****************************************************************************** Symbol Tables *)
22 type symbol_table = {
23  (*general symbol table for variables*)
24  parent : symbol_table option;
25  mutable variables : (string * Ast.dataType) list;
26  mutable structs : struc_table list;
27  mutable options : option_table list;
28  mutable matrices : matrix_table list;
29 }
30
31 type environment = {
32  mutable func_return_type : Ast.dataType; (* Function return type *)
33  scope : symbol_table; (* symbol table for variables *)
34  mutable functions : (string * Ast.dataType list * Ast.dataType list) list; (* symbol table for global functions, nested function declaration not supported*)
35 }
36 (****************************************************************************** Debug Functions*)
37 let print_var_elem = print_endline (fst elem ^ "\t" ^ string_of_dataType (snd elem)^" is in scope")
38
39 let rec print_vars_list = function (*String[dataType list to print]*)
40  [] -> print_string "empty vars"
41  e::l -> print_var e ; print_vars_list l
42
43 let rec print_mvars = function
44  [] -> print_endline "empty matrices"
45  e::l -> print_endline e.matrix_name ; print_mvars l
46 (****************************************************************************** Initial Functions*)
47
48 let new_env : environment =
49  let core = ["toString", [String], String];["toString", [Int], String];
50 ["toString", [Float], String];["toString", [Matrix], String];
51 ["toString", [Structure], String];["toString", [Option], String];
52 ["toString", [Boolean], String];["toString", [Void], Void];
53 ["print", [String], Void];["print", [Int], String];
54 ["print", [Float], Void];["print", [Matrix], Void];
55 ["print", [Structure], Void];["print", [Option], Void];
56 ["print", [Boolean], Void];["toString", [String], Int];
57 ["toString", [String], Float];["toString", [Matrix], Matrix];
58 ["toString", [Matrix], Matrix];
59 ["toString", [Matrix], Matrix];
60 let s = (variables = []; structs = []; options = []; matrices = []; parent = None)
61 in
62 {scope = s; func_return_type = Void; functions = core;}
63
64 let inner_scope (env : environment) : environment =
65  let s = (variables = []; structs = []; options = []; matrices = []; parent = Some(env.scope))
66 in
67 {env with scope = s;}
68 (****************************************************************************** Utils*)
69
70 let rec check_dup l = match l with
71  [] -> false
72  l h::t ->
73  let x = ([List.filter (fun x -> x = h) t) in
74  if (x :: []) then
75  check_dup t
76  else
77  true
78
79 let rec sameLists l1 l2 = match l1, l2 with
80  [], [] -> true
81  l1, _ | _ :: l2 -> false
82  l1::xs, y::ys -> x = y && sameLists xs ys
83
84 let is_keyword (var_name:string) =
85  let keyword_set = ["main";"Void";"if";"else";"for";"while";"return";"true";"false";
86  "Int";"Float";"Matrix";"Structure";"Boolean"] in
87  try
88  List.find (fun x -> x=var_name) keyword_set;
let find_vars_exist (env_scope : symbol_table) (var_name : string) = try
  List.find (fun (s, _) -> s = var_name) env_scope.variables;
true with Not_found ->
try
  List.find (fun x -> x.struc_name = var_name) env_scope.structs;
true with Not_found ->
try
  List.find (fun x -> x.option_name = var_name) env_scope.options;
true with Not_found ->
try
  List.find (fun x -> x.matrix_name = var_name) env_scope.matrixes;
true with Not_found ->
(match env_scope.parent with
| Some(p) -> find_vars_exist p var_name
| _ -> false)

let find_vars (env_scope : symbol_table) (var_name : string) : Ast.dataType = try
  let (_, typ) = List.find (fun (s, _) -> s = var_name) env_scope.variables in
  typ with Not_found ->
try
  List.find (fun x -> x.struc_name = var_name) env_scope.structs;
Structure with Not_found ->
try
  List.find (fun x -> x.option_name = var_name) env_scope.options;
Option with Not_found ->
try
  List.find (fun x -> x.matrix_name = var_name) env_scope.matrixes;
Matrix with Not_found ->
(match env_scope.parent with
| Some(p) -> find_vars p var_name
| _ -> raise(Failure("Cannot find variable named " ^ var_name )

("The below three functions return corresponding data_type table instead of data_type or Boolean")

let find_structs (env_scope : symbol_table) (var_name : string) =
try
  let struct_find = List.find (fun x -> x.struc_name = var_name) env_scope.structs in
  struct_find with Not_found ->
  match env_scope.parent with
  | Some(p) -> find_structs p var_name
  | _ -> raise(Failure("Cannot find structre named " ^ var_name )

let find_options (env_scope : symbol_table) (var_name : string) =
try
  let option_find = List.find (fun x -> x.option_name = var_name) env_scope.options in
  option_find with Not_found ->
  match env_scope.parent with
  | Some(p) -> find_options p var_name
  | _ -> raise(Failure("Cannot find option named " ^ var_name )

let find_matrix (env_scope : symbol_table) (var_name : string) =
try
  let matrix_find = List.find (fun x -> x.matrix_name = var_name) env_scope.matrixes in
  matrix_find with Not_found ->
  match env_scope.parent with
  | Some(p) -> find_matrix p var_name
  | _ -> raise(Failure("Cannot find matrix named " ^ var_name )

let find_funs_exist (env : environment) (var_name : string) =
try
  let (_, typ) = List.find (fun (s, forlist, _) -> s = var_name && samelists forlist arglist) env.functions in
  typ with Not_found -> raise(Failure("Cannot find function named " ^ var_name )

("get type of an expression")
let get_type (e : Ast.expr_t) : Ast.dataType =
match e with
let

(* test matrix sizes are equal *)

let get_dimension (env: environment) (exp: Ast.expr) : size_of_matrix =

match exp with

| Id(s) ->

| m.size

| MatBinary_op (e1, op, e2) -> get_dimension env e1

| VarAssign (s, e) -> get_dimension env exp

| Precedence_expr (e) -> get_dimension env exp

| MatBinary_op (e, op) ->

let msize = get_dimension env e in

match op with

| MTranspose ->

| MInverse ->

| MDeterminant ->

| _ -> raise(Reason("Cannot find dimension which is not of matrix type"))

(* test matrix sizes are equal *)

let size_equal (size1: size_of_matrix) (size2: size_of_matrix) : bool =

if size1.rows = size2.rows && size1.cols = size2.cols

then true

else false

(* test matrix sizes are transpose equal *)

let size_mult_equal (size1: size_of_matrix) (size2: size_of_matrix) : bool =

if size1.rows = size2.cols && size1.cols = size2.rows

then true

else false

let get_formal_type (vardec: Ast.var_dec) : Ast.data_type =

vardec.vtype

let rec annotate_expr (env: environment) (e: Ast.expr) (func_ts : Ast.func_dec_t list) : Ast.expr_t =

match e with

| Binary_op (e1, op, e2) ->

| _ ->

match e1 with

| Float ->

| _ -> raise(Reason("Binary operation has un-consistent types"))

| Int ->

| _ -> raise(Reason("Binary operation has un-consistent types"))

| String ->

| _ -> raise(Reason("Binary operation has un-support types"))

| Sub | Times | Divide ->

| _ ->

match e2 with

| Float ->

| _ -> raise(Reason("Binary operation has un-consistent types"))

| Int ->

| _ -> raise(Reason("Binary operation has un-support types"))

| And | Or ->

| _ -> raise(Reason("Binary should be the types around boolean operations"))

else Binary_op (e1, op, e2, e1.t)
let e1_a = annotate_expr env e1 func_ts in
let e2_a = annotate_expr env e2 func_ts in
let e1_t = get_type e1_a in
let e2_t = get_type e2_a in
(match op with
| MAdd | MIDivide | MAdd | MSub |->
  if (e1_t <- Matrix) || (e2_t <- Matrix)
  then raise(Failure("Matrix operation has to be Matrix type on both sides"))
else
  let size1 = get_dimension env e1 in
  let size2 = get_dimension env e2 in
  (match op with
   | MAdd | MIDivide |->
     if size1_equal size2
     then Matrix_op_t(e1_a, op, e2_a, Matrix)
     else raise(Failure("Matrix addition or Matrix subtraction has incompitable sizes"))
   | MTime | MDivide | MAdd | MSub |->
     if size1_equal size2
     then Matrix_op_t(e1_a, op, e2_a, Matrix)
     else raise(Failure("Matrix addition or Matrix subtraction has incompitable sizes"))
)

Id(s) ->
let typ = find_vars env.scope s in Id_t(s,typ)
let FunLit(f) = float_lit_t(f,Float)
let IntLit(i) -> IntLit_t(i,Int)
let StringLit(s) -> StringLit_t(s,String)

Call(name, exl) ->
let exl_a = List.map (fun x -> annotate_expr env x func_ts) exl in
let arglist = List.map (fun x -> get_type x) exl_a in
let ret_type = find_funcs env name arglist in
(*Check whether this func exist*)
try
  let func_dec_t = List.find (fun x -> x.func_name_t = name) func_ts in
  (*Check for arguments and added to environment*)
  let formals = func_dec_t.formals_t in
  let arg_size = List.length exl in
  let inner_env = inner_scope env in
  inner_env.func_return_type <- ret_type;
  for i = 0 to arg_size-1 do
    let formal = List.nth formals i in
    let org = List.nth exl i in
    (match formal.typ with
     | Int | Float | Boolean | String ->
       inner_env.scope.variables <- [(formal.fname, formal.vtype)] @ inner_env.scope.variables
     | Matrix ->
       (match org with
        | Void(s) ->
          let mat_to_func = find_matrix env scope sn in
          inner_env.scope.matrices <- [mat_to_func with matrix_name = sn] :: inner_env.scope.matrices
          l => raise(Failure("Cannot pass this variable")))
      | Structure ->
        (match org with
         | Void(s) ->
           let struct_to_func = find_structs env.scope sn in
           inner_env.scope.structs <- [struct_to_func with struct_name = sn] :: inner_env.scope.structs
           l => raise(Failure("Cannot pass this variable")))
         | Option ->
           (match org with
            | Void(s) ->
              let option_to_func = find_options env.scope sn in
              inner_env.scope.options <- [option_to_func with option_name = sn] :: inner_env.scope.options
              l => raise(Failure("Cannot pass this variable")))
        )
    done;
    let stmtlist = func_dec_t.ori_body in
    func_dec_t.body_t = annotate_stmts inner_env stmtlist func_ts;
    Call_t(name,exl_a,ret_type)
with Not_found -> Call_t(name,exl_a,ret_type) (*Must be built-in function*)

VarAssign(s, e2) -> let exp = Id(s) in annotate_assign env exp e2 func_ts
Matrix_element_assign(s, e1, e2) ->
    let exp = Matrix_element(e1, e2) in annotate_assign env exp e3 func_ts
Struct_element_assign(s1, s2) in
    let exp = Struct_element(s1, s2) in annotate_assign env exp e func_ts
Precendece_expr(e) -> annotate_expr env e func_ts
Struct_element(s1,s2) -> check_struct env s1 s2
Matrix_element(s_me1,s2) -> check_matrix_elem env s_me1 me2 func_ts
Bool_lit(e) -> Bool_lit_t(e,Boolean)
Matrix_op(e,op) ->
let e_o = annotate_expr env e_func_ts
let e_t = get_type e_o in
if e_t <> Matrix
then raise(Failure("Matrix uni-operation has to be applied to Matrix type"))
else
(match op with
| MTTransposes MInsertion -> MatrixUnaryOp(e,op,Matrix)
| MTranspose   -> MatrixUnaryOp(e,op,Float)
| MDeterminant -> MatrixUnaryOp(e,op,Float)
| _            -> Nonexpr_t(VOID)

and check_matrix_elem (env : environment) (name:string)(e1:Ast.expr)(e2:Ast.expr)(func_ts : Sast.func_dec_t list) : Sast.expr_t =
let var_type = find_vars env.scope name in
match var_type with
| Matrix ->
let e1_o = annotate_expr env e1_func_ts in
let e2_o = annotate_expr env e2_func_ts in
let e1_typ = get_type e1_o in
let e2_typ = get_type e2_o in
if (e1_typ<>Int)||(e2_typ<>Int)
then raise(Failure("Indexes of Matrix must be of Int type"))
else Matrix_element_t(name,e1_o,e2_o,Float)
| _        -> raise(Failure("Should be of Structure type for accessing elements"))

and check_struc_elem (env:environment) (name:string)(key:string) : Sast.expr_t =
let var_type = find_vars env.scope name in
match var_type with
| Structure ->
(let struc = find_structs.env.scope.name in
try
List.find (fun s -> s = key) struc.struc_fields;
Struct_element_t(name,key,String)
with Not_found -> raise(Failure("Field "key" does not exist within struct"))
| Option ->
(let opt = find_options.env.scope.name in
try
List.find (fun s -> s = key) opt.option_fields;
Struct_element_t(name,key,String)
with Not_found -> raise(Failure("Field "key" does not exist within struct"))
| _        -> raise(Failure("Should be of Structure type for accessing fields"))

and annotate_assign (env : environment) (e1 : Ast.expr) (e2 : Ast.expr)(func_ts : Sast.func_dec_t list) : Sast.expr_t =
let e1_o = annotate_expr env e1_func_ts in
match e1 with
| Id(x) ->
let e2_type = get_type e2_o in
let e1_type = find_vars.env.scope.x in
match e1_type with
| Float ->
if (e2_type = Float) || (e2_type = Int)
then VarAssign_t(Id_t(x, e1_type), e2_o, e1_type)
else raise(Failure("Variable "x" need to be float type"))
| _        ->
if e2_type <> e1_type
then raise(Failure("Variable "x" need to be assigned with same type " ^ string_of_data_type e1_type ^ " " ^ string_of_data_type e2_type ))
else VarAssign_t(Id_t(x, e1_type), e2_o, e2_type)

| Matrix_element(s,me1,me2) ->
let e1_type = check_matrix_elem.env s me1 me2 func_ts in
let e2_type = get_type e2_o in
(match e2_type with
| Float|Int ->
let me1_o = annotate_expr env me1_func_ts in
let me2_o = annotate_expr env me2_func_ts in
if (get_type me1_o = Int) && (get_type me2_o = Int)
then Matrix_element_assign_t(s, me1_o, me2_o, e2_o, Float)
else raise(Failure("Matrix indexing has to be of type int"))
| _        -> raise(Failure("Only allow Float or Int assigned to Matrix element"))

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )

| )
445 | If(be, body1, body2) ->
446 | let be_a = annotate_expr env be func_ts in
447 | if (get_type be_a) <> Boolean
448 | then raise(Failure("there should be boolean expression within IF"))
449 | else
450 | let body1_a = annotate_stmt env body1 func_ts in
451 | let body2_a = annotate_stmt env body2 func_ts in
452 | if t(be_a, body1_a, body2_a)
453 | For(a1, be, ae2, body) ->
454 | let ae1_a =
455 | (match ae1 with
456 | | VarAssign(s, e2) -> let exp = Id(s) in annotate_assign env exp e2 func_ts
457 | | Matrix_element_assign(s, e1, e2, e3) ->
458 | | let exp = Matrix_element_assign(s, e1, e2, e3) in annotate_assign env exp e3 func_ts
459 | | Struct_element_assign(s1, s2, e) ->
460 | | let exp = Struct_element_assign(s1, s2, e) in annotate_assign env exp e func_ts
461 | | Noexpr -> Noexpr(t(Void)
462 | | _ -> raise(Failure("Need assignment in for loop header"))) in
463 | let be_a =
464 | (match be with
465 | | Binary_op(_, _, _) -> annotate_expr env be func_ts
466 | | Id(s) ->
467 | | let ret_type = find_vars env scope s in
468 | | if ret_type <> Boolean
469 | | then raise(Failure("condition expression within For loop is not a correct type"))
470 | | else
471 | | annotate_expr env be func_ts
472 | | Bool_lit(_.) -> annotate_expr env be func_ts
473 | | Noexpr -> Noexpr(t(Void)
474 | | _ -> raise(Failure("condition expression within For loop is not a correct type"))) in
475 | let ae2_a =
476 | (match ae2 with
477 | | VarAssign(s, e2) -> let exp = Id(s) in annotate_assign env exp e2 func_ts
478 | | Matrix_element_assign(s, e1, e2, e3) ->
479 | | let exp = Matrix_element_assign(s, e1, e2, e3) in annotate_assign env exp e3 func_ts
480 | | Struct_element_assign(s1, s2, e) ->
481 | | let exp = Struct_element_assign(s1, s2, e) in annotate_assign env exp e func_ts
482 | | Noexpr -> Noexpr(t(Void)
483 | | _ -> raise(Failure("there should be assign expression within For loop"))) in
484 | let body_a = annotate_stmt env body func_ts
485 | in
486 | if (get_type be_a <> Boolean) && (get_type be_a <> Void)
487 | then raise(Failure("there should be boolean expression within For"))
488 | else
489 | For(t(ae1_a, be_a, ae2_a, body_a)
490 | While(be, body) ->
491 | let be_a = annotate_expr env be func_ts in
492 | let body_a = annotate_stmt env body func_ts
493 | in
494 | if (get_type be_a <> Boolean) && (get_type be_a <> Void)
495 | then raise(Failure("there should be boolean expression within For"))
496 | else
497 | While_t(be_a, body_a)
498 | Return(e) ->
499 | let e_a = annotate_expr env e func_ts
500 | in
501 | let return_type = get_type e_a
502 | in
503 | if return_type <> env.func_return_type
504 | then raise(Failure("actual return type is not same with function return type: " ^ string_of_dataType env.func_return_type)
505 | else
506 | Return_t(e_a)
507 | Vardec(vd) ->
508 | let var_type = vd.vtype in
509 | let var_name = vd.vname in
510 | if is_keyword var_name
511 | then raise(Failure("Cannot use keyword " ^ var_name ^ " as variable name"))
512 | else
513 | let exist_v = find_vars_exist env.scope var_name
514 | in
515 | if exist_v
516 | then raise(Failure("Variable name " ^ var_name ^ " already been used."))
517 | else
518 |
519 | Vardec(vd, var_type)
520 | Matdec(md) ->
521 | let var_type = md.mtype in
522 | let var_name = md.mname in
523 | if is_keyword var_name
524 | then raise(Failure("Cannot use keyword " ^ var_name ^ " as variable name"))
525 | else
526 | if var_type <> Matrix
527 | then raise(Failure("The declaration of " ^ var_name ^ " only apply to Matrix"))
528 | else
529 | let exist_v = find_vars_exist env.scope var_name
530 | in
622

let annotate_stmt env x func_ts =
| else if get_type x = String
634
| then raise Failure("Only String can be assigned to filed of Structure")
635
| else fid | starglist |
636
| in
637
| let dup = check_dup fields |
638
| in
639
| if dup
640
| then raise Failure("There are duplicate fields inside structure " ^ var_name)
641
| else
642
| ( env.scope.structures <- env.scope.structures @ [struc_name = var_name; struc_fields = fields];
643
| Structdec (var_name, starglist, Structure)) |
644
| if is_keyword var_name
645
| then raise Failure("Cannot use keyword " ^ var_name " as variable name")
646
| else
647
| let exist_v = find_vars_exist env var_name
648
| in
649
| if exist_v
650
| then raise Failure("Variable name " ^ var_name " already been used.")
651
| else
652
| (check whether filed names have duplicates and type are of string*)
653
| let fields = list.map (fun x ->
654
| let fid = x.id in
655
| let fval = annotate_expr env x.value func_ts in
656
| if get_type fval = String
657
| then raise Failure("Only String can be assigned to filed of Structure")
658
| else fid | starglist |
659
| in
660
| let dup = check_dup fields |
661
| in
662
| if dup
663
| then raise Failure("There are duplicate fields inside Option " ^ var_name)
664
| else
665
| ( env.scope.options <- env.scope.options @ [option_name = var_name; option_fields = fields];
666
| Optiondec (var_name, starglist, Option)) |
667
| and annotate_stmts (env : environment) (stmts : Ast.stmt list) (func_ts : Sast.func_dec_t list) : Sast.stmt_t list =
668
| list.map (fun x -> (print_string (string_of_stmt x); annotate_stmt env x func_ts)) stmts |
669
| if is_keyword name && name = "main"
670
| then raise Failure("Cannot use keyword " ^ name " as function name")
671
| else
672
| let exist_f = find_funcs_exist env name |
673
| in
674
| if exist_f
675
| then raise Failure("Function name " ^ name " already been used.")
676
| else
677
| let formulas_list = func.formulas in
678
| let data_type_list = list.map (fun x -> get_formal_type x) formulas_list |
679
| env.functions <- env.functions @ [(name, data_type_list, ret_type)] |
680
| if is_keyword name
681
| then raise Failure("Cannot use keyword " ^ name " as variable name")
682
| else
683
| let exist_v = find_vars_exist env var_name |
684
| in
685
| if exist_v
686
| then raise Failure("Variable name " ^ var_name " already been used.")
687
| else
688
| let formals_list = func.formals in
689
| let data_type_list = list.map (fun x -> get_formal_type x) formulas_list |
690
| env.functions <- env.functions @ [(name, data_type_list, ret_type)] |
691
| if is_keyword name
692
| then raise Failure("Cannot use keyword " ^ name " as variable name")
693
| else
694
| let exist_v = find_vars_exist env var_name |
695
| in
696
| if exist_v
697
| then raise Failure("Variable name " ^ var_name " already been used.")
698
| else
699
| let formals_list = func.formals in
700
| let data_type_list = list.map (fun x -> get_formal_type x) formulas_list |
701
| env.functions <- env.functions @ [(name, data_type_list, ret_type)] |
702
| if is_keyword name
703
| then raise Failure("Cannot use keyword " ^ name " as variable name")
704
| else
705
| let exist_v = find_vars_exist env var_name |
706
| in
707
| if exist_v
708
| then raise Failure("Variable name " ^ var_name " already been used.")
709
| else
let check_from_main (env : environment) (func_ts : Sast.func_dec_t list) =
try
    let mainfunc = List.find (fun x -> x.func_name_t = "main") func_ts in
    let formls = mainfunc.formals_t in
    let size = List.length formls in
    for i = 0 to size-1 do
        let formal = List.nth formls i in
        print_endline (string_of_dataType formal.vtype);
        env.scope.variables <- [(formal.vname, formal.vtype)] @ env.scope.variables;
        env.func_return_type <- Int
    done;
    let mainbody = mainfunc.ori_body in
    let main_exist = find_funcs_exist env "main" in
    if main_exist then
        check_from_main env func_dec_t_list
    else
        raise (Failure ("Must declare main function"))
    mainbody_t <- annotate_stmts env mainbody func_ts
    print_endline "Semantic analysis completed successfully."

let check_program (stmts, funcs) =
    let globals = List.rev stmts in
    let functions = List.rev funcs in
    let env = new_env in
    annotate_global_stmts env globals;
    let func_dec_t_list = annotate_funcs env functions in
    let main_exist = find_funcs_exist env "main" in
    if main_exist then
        check_from_main env func_dec_t_list
    else
        raise(Failure("Must declare main function"))
```
1 open Act
2 (**SAST************)
3
4 (**Types Annotated***)
5
6 type expr_t =
7  Binary_op_t of expr_t * bin_op * expr_t * dataType
8  MatBinary_op_t of expr_t * mat_op * expr_t * dataType
9  Id_t of string * dataType
10  Float_lit_t of float * dataType
11  Int_lit_t of int * dataType
12  String_lit_t of string * dataType
13  Call_t of string * expr_t list * dataType
14  VarAssign_t of expr_t * expr_t * dataType
15  Matrix_element_assign_t of string * expr_t * expr_t * expr_t * dataType
16  Struct_element_assign_t of string * string * expr_t * dataType
17  Matrix_element_t of string * string * expr_t * expr_t * dataType
18  Precedence_expr_t of expr_t * dataType
19  Struct_element_t of string * string * dataType
20  Bool_lit_t of int * dataType
21  MatUnary_op_t of expr * mat_uop * dataType
22  Noexpr_t of dataType
23
24 type stmt_t =
25  Block_t of stmt_t list
26  Expr_t of expr_t
27  If_t of expr_t * stmt_t * stmt_t
28  For_t of expr_t * expr_t * expr_t * stmt_t
29  While_t of expr_t * stmt_t
30  Return_t of expr_t
31  Vardec_t of var_dec * dataType
32  Matdec_t of mat_dec * dataType
33  Structdec_t of string * struct_arg list * dataType
34  Optiondec_t of string * struct_arg list * dataType
35
36 type func_dec_t = {
37    ret_t : dataType;
38    func_name_t : string;
39    formals_t : var_dec list;
40    mutable body_t : stmt_t list;
41    mutable ori_body : stmt list;
42  }
43
44 type program_t = stmt_t list * func_dec_t list
45```
open Printf
open Ast

4(* data type string gen*)
5let string_of_dataType : function
6  Int     -> "int"
7  Float   -> "double"
8  String  -> "String"
9  Matrix  -> "Matrix"
10  Option  -> "Option"
11  Structure -> "Structure"
12  Boolean -> "boolean"
13  Void    -> "void"
14
15(* variable declaration string gen*)
16let string_of_vdecl vdecl = string_of_dataType vdecl.type ^ " " ^ vdecl.name ^ match vdecl.type with
17  Int     -> "-i"
18  Float   -> "-d"
19  String  -> "" ^ vdecl.name
20  Matrix  -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_mrow" ^ " " ^ string_of_int "dcl_mcol" ^ ""
21  Option  -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
22  Structure -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
23  Void    -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
24
25(* variable declaration string gen notice the static *)
26let string_of_global_vdecl vdecl = "static " ^ string_of_dataType vdecl.type ^ " " ^ vdecl.name ^ match vdecl.type with
27  Int     -> "-i"
28  Float   -> "-d"
29  String  -> "" ^ vdecl.name
30  Matrix  -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_mrow" ^ " " ^ string_of_int "dcl_mcol" ^ ""
31  Option  -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
32  Structure -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
33  Void    -> "" ^ string_of_int "dcl_name" ^ " " ^ string_of_int "dcl_value" ^ ""
34
35
36(*matrix binary operations*)
37let string_of_vdecl_argument vdecl = string_of_dataType vdecl.type ^ " " ^ vdecl.name
38let string_of_vdecl_mdecl = string_of_dataType mdecl.type ^ " " ^ mdecl.name
39  "new Matrix(" ^ string_of_int mdecl.mrow ^ "," ^ string_of_int mdecl.mcol ^ ")"
40(*put struct id into the declaration list inside struct declaration*)
41let rec tuple_id (id,nums) = match nums with
42  []      -> []
43  i :: tl   -> (id,head):: tuple_id (id,tail)
44(*string for expression*)
45let rec string_of экспрjavagen = function
46  FloatLit (l) -> string_of_expjavagen e1 ^ " " ^
47  (match o with
48    (* And | Or | Eq | Neq | Lt | Gt | Leq | Geq*)
49    Add -> "+" | Sub -> "-" | Times -> "*" | Divide -> "/" | Eq -> "="
50    Neq -> "\"" | Lt -> "<" | Leq -> "<=" | Gt -> ">" | Geq -> "\""
51    And -> "&&" | Or -> "||"
52    ) ^ " " ^
53  string_of_expjavagen e2
54  (*matrix binary operations*)
55let MatBinary_op (e1, e2) ->
56    string_of_expjavagen e1 ^ " " ^
57    (match o with
58      (*MatrixMathematics.add("string_of_expjavagen e1","string_of_expjavagen e2")*)
59      | MatrixMathematics.subtract("string_of_expjavagen e1","string_of_expjavagen e2")*)
60      | MatrixMathematics.multiply("string_of_expjavagen e1","string_of_expjavagen e2")*)
61      | MatrixMathematics.divide("string_of_expjavagen e1","string_of_expjavagen e2")*)
62    | MatrixMathematics.multiplyByConstant("string_of_expjavagen e1",Constant 1.000/("string_of_expjavagen e2")*)
63    | MatrixMathematics.addByConstant(1("string_of_expjavagen e1")*,"string_of_expjavagen e2")*)
64    | MatrixMathematics.addByConstant(1("string_of_expjavagen e1")*,"string_of_expjavagen e2")*)
65    | MatrixMathematics.multiplyByConstant("string_of_expjavagen e1",Constant 1.000/("string_of_expjavagen e2")*)
66    )
67    (*utility operations*)
68    | MatIdentity_op (e, o) ->
69      (match o with
70        (*MatrixMathematics.add("string_of_expjavagen e","string_of_expjavagen e")*)
71        | MatrixMathematics.subtract("string_of_expjavagen e","string_of_expjavagen e")*)
72        | MatrixMathematics.multiply("string_of_expjavagen e","string_of_expjavagen e")*)
73        | MatrixMathematics.divide("string_of_expjavagen e","string_of_expjavagen e")*)
74        | MatrixMathematics.multiplyByConstant("string_of_expjavagen e",Constant 1.000/("string_of_expjavagen e")*)
75        | MatrixMathematics.addByConstant(1("string_of_expjavagen e")*,"string_of_expjavagen e")*)
76        | MatrixMathematics.multiplyByConstant("string_of_expjavagen e",Constant 1.000/("string_of_expjavagen e")*)
77      )
78      (*functions that are not built in are listed here*)
79      | coll(f, e1) -> (match f with
80        (*print*) -> ("" ^ (List.map string_of_expjavagen e1) ^ ".print()")
81        | toInt()      -> (Integer.parseInt("string_of_expjavagen e1")*)
82        | toFloat()    -> (Double.parseDouble("string_of_expjavagen e1")*)
83        | toBoolean()  -> (Boolean.parseBoolean("string_of_expjavagen e1")*)
84        | toString()   -> (String.valueOf("string_of_expjavagen e1")*)
85        | toSystem()   -> (System.out.println("string_of_expjavagen e1")*)
86      )
javagen.ml

```java
public class %s
{
%s
%
}
```
1 2(* angelaZ compiler 
3 1. give stdin to scanner, get tokens 
4 2. give tokens to parser, get AST 
5 3. give AST to analyzer, get semantic tree 
6 4. give semantic tree to java converter, get java tree 
7 5. give java tree to java code generator, get java code 
8 6. give java code to java compiler, get executable 
9 7. run java executable 
10*)
11
12type action = Sast | Java 
13
14let _ = if Array.length Sys.argv > 1 then 
15 List.assoc Sys.argv.(1) [ ("-s", Sast); ("-j", Java)] 
16else Java in 
17let lexbuf = Lexing.from_channel stdin in 
18let program = Parser.program Scanner.token lexbuf in 
19match action with 
20Sast -> 
21Typecheck.check_program program 
22Java -> 
23let _ = Javagen.gen_program "Output" program in 
24print_string "Success! Compiled to java/output.java\n" 
25
open Printf
open Ast

let string_of_datatype = function
  | Int -> "int"
  | Float -> "double"
  | String -> "String"
  | Matrix -> "Matrix"
  | Option -> "Option"
  | Structure -> "Structure"
  | Boolean -> "boolean"
  | Void -> "void"

(* global variable string gen *)
let string_of_vdecl_vdecl = string_of_datatype vdecl.vtype ^ " " ^ vdecl.vname ^ match vdecl.vtype with
  | Int -> "int"
  | Float -> "double"
  | String -> "null"
  | Matrix -> "Matrix"
  | Option -> ""
  | Structure -> ""
  | Boolean -> "false"
  | Void -> ""

(* variable declaration string gen notice the static *)
let string_of_global_vdecl_vdecl = "static " ^ string_of_datatype vdecl.vtype ^ " " ^ vdecl.vname ^ match vdecl.vtype with
  | Int -> "int"
  | Float -> "double"
  | String -> "null"
  | Matrix -> "Matrix"
  | Option -> ""
  | Structure -> ""
  | Boolean -> "false"
  | Void -> ""

let string_of_vdecl_argument_vdecl = string_of_datatype vdecl.vtype ^ " " ^ vdecl.vname

let string_of_mdecl_vdecl = string_of_datatype mdecl.mtype ^ " " ^ mdecl.vname

(* put struct id into the declaration list inside struct declaration*)
let rec tuple_id (id,nums) = match nums with
  | [] -> []
  | head :: tail -> (head, tail)

(*string for expression*)
let rec string_of_expr_javagen = function
  | FloatLit(l) -> string_of_float l
  | IntLit(l) -> string_of_int l
  | StringLit(l) -> ^ "\"" ^ l ^ \""
  | Id(s) -> ^ s
  | Binary_op(e1, e2) -> ^ string_of_expr_javagen e1 ^ " " ^ (match o with
    | Add -> "+
    | Sub -> "-
    | Times -> "*
    | Divide -> "/
    | Lt -> "<
    | Leq -> "\≤"
    | Gt -> ">
    | Geq -> "\≥"
    | AndAlso -> "&&"
    | OrElse -> "||"
    | Not -> "!
  ) ^ " " ^ string_of_expr_javagen e2

(*matrix binary operations*)
let Matrix_binary_op(e1,e2) -> ^ (match o with
  | MAdd -> "MatrixMathematics.add(" ^ string_of_expr_javagen e1 ^ ", " ^ string_of_expr_javagen e2 ^ "")"
  | MSub -> "MatrixMathematics.subtract(" ^ string_of_expr_javagen e1 ^ ", " ^ string_of_expr_javagen e2 ^ "")"
  | MTime -> "MatrixMathematics.multiply(" ^ string_of_expr_javagen e1 ^ ", " ^ string_of_expr_javagen e2 ^ "")"
  | MDivide -> "MatrixMathematics.divide(" ^ string_of_expr_javagen e1 ^ ", " ^ string_of_expr_javagen e2 ^ "")"
  | MInverse -> "MatrixMathematics.inverse(" ^ string_of_expr_javagen e1 ^ "")"
  | MDeterminant -> "MatrixMathematics.determinant(" ^ string_of_expr_javagen e1 ^ ")"
    | MIdentity -> "MatrixMathematics.transpose(" ^ string_of_expr_javagen e1 ^ ")"
)

(*utility operation*)
let MatIdentity(e, o) -> ^ (match o with
  | MTranspose -> "MatrixMathematics.transpose" ^ " MatrixMathematics.inverse" ^ " MatrixMathematics.determinant"
    | Map -> "MatrixMathematics.add(" ^ string_of_expr_javagen e ^ ", " ^ string_of_expr_javagen e ^ ")"
    | Assign -> "MatrixMathematics.multiply(" ^ string_of_expr_javagen e ^ ", " ^ string_of_expr_javagen e ^ ")"
  ) ^ "(" ^ string_of_expr_javagen e ^ "", " ^ string_of_expr_javagen e ^ ")"

(* functions that are built in are listed here*)
let call(f, e1) -> (match f with
  | "printM" -> (("String.concatenate", "(List.map string_of_expr_javagen e1)"))")
  | "Int" -> "(Int.parseInt(" ^ String.concatenate ^ ", " ^ List.map string_of_expr_javagen e1 ^ "))"
  | "Float" -> "Double.parseDouble(" ^ String.concatenate ^ ", " ^ List.map string_of_expr_javagen e1 ^ "))"
  | "String" -> "(String.toString ^ " String.concatenate", "(List.map string_of_expr_javagen e1)" ^ ")"
  | "Option" -> "(Option.getPrice ^ " String.concatenate", "(List.map string_of_expr_javagen e1)" ^ ")"
  
  | "print" -> "System.out.println((" ^ String.concatenate ^ ", " ^ List.map string_of_expr_javagen e1 ^ "))"
  | "price" -> "(" ^ String.concatenate ^ ", " ^ List.map string_of_expr_javagen e1 ^ ")")

println "Hello, world!"
let rec string_of_global_stmt = function
    | Worddecl(vdecl) -> string_of_global_vdecl vdecl ^ ;

let rec string_of_vdecl argument = function
    | Void -> "void" ^ ;
    | Boolean -> "boolean" ^ ;
    | Integer -> "integer" ^ ;
    | Float -> "double" ^ ;
    | String -> "String" ^ ;
    | Char -> "char" ^ ;
    | Enum -> "enum" ^ ;
    | Class -> "class" ^ ;
    | Array -> "[]" ^ ;
    | Typeid(id) -> "id" ^ ;
    | List(id) -> "List<id>" ^ ;
    | Tuple(id) -> "Tuple<id>" ^ ;
    | Option(id) -> "Option<id>" ^ ;
    | Struct(id) -> "Structure<id>" ^ ;
    | Func(id) -> "Function<id>" ^ ;
    | Call(id, args) -> "Call<id, args>" ^ ;
    | Index(id, indices) -> "Index<id, indices>" ^ ;
    | Member(id, member) -> "Member<id, member>" ^ ;
    | Method(id, args) -> "Method<id, args>" ^ ;
    | TypeParameter(id) -> "TypeParameter<id>" ^ ;
    | ArrayType(id) -> "ArrayType<id>" ^ ;
    | ListType(id) -> "ListType<id>" ^ ;
    | TupleType(id) -> "TupleType<id>" ^ ;
    | OptionType(id) -> "OptionType<id>" ^ ;
    | StructType(id) -> "StructType<id>" ^ ;
    | FuncType(id) -> "FuncType<id>" ^ ;
    | JoinType(id, types) -> "JoinType<id, types>" ^ ;
    | Binary(id, types) -> "Binary<id, types>" ^ ;
    | Unary(id, type) -> "Unary<id, type>" ^ ;
    | ArrayTypeType(id) -> "ArrayTypeType<id>" ^ ;
    | ListTypeType(id) -> "ListTypeType<id>" ^ ;
    | TupleTypeType(id) -> "TupleTypeType<id>" ^ ;
    | OptionTypeType(id) -> "OptionTypeType<id>" ^ ;
    | StructTypeType(id) -> "StructTypeType<id>" ^ ;
    | FuncTypeType(id) -> "FuncTypeType<id>" ^ ;
    | JoinTypeType(id, types) -> "JoinTypeType<id, types>" ^ ;
    | BinaryType(id, types) -> "BinaryType<id, types>" ^ ;
    | UnaryType(id, type) -> "UnaryType<id, type>" ^ ;
    | ArrayTypeTypeType(id) -> "ArrayTypeTypeType<id>" ^ ;
    | ListTypeTypeType(id) -> "ListTypeTypeType<id>" ^ ;
    | TupleTypeTypeType(id) -> "TupleTypeTypeType<id>" ^ ;
    | OptionTypeTypeType(id) -> "OptionTypeTypeType<id>" ^ ;
    | StructTypeTypeType(id) -> "StructTypeTypeType<id>" ^ ;
    | FuncTypeTypeType(id) -> "FuncTypeTypeType<id>" ^ ;
    | JoinTypeTypeType(id, types) -> "JoinTypeTypeType<id, types>" ^ ;
    | BinaryTypeType(id, types) -> "BinaryTypeType<id, types>" ^ ;
    | UnaryTypeType(id, type) -> "UnaryTypeType<id, type>" ^ ;
    | ArrayTypeTypeTypeType(id) -> "ArrayTypeTypeTypeType<id>" ^ ;
    | ListTypeTypeTypeType(id) -> "ListTypeTypeTypeType<id>" ^ ;
    | TupleTypeTypeTypeType(id) -> "TupleTypeTypeTypeType<id>" ^ ;
    | OptionTypeTypeTypeType(id) -> "OptionTypeTypeTypeType<id>" ^ ;
    | StructTypeTypeTypeType(id) -> "StructTypeTypeTypeType<id>" ^ ;
    | FuncTypeTypeTypeType(id) -> "FuncTypeTypeTypeType<id>" ^ ;
    | JoinTypeTypeTypeType(id, types) -> "JoinTypeTypeTypeType<id, types>" ^ ;
    | BinaryTypeTypeType(id, types) -> "BinaryTypeTypeType<id, types>" ^ ;
    | UnaryTypeTypeType(id, type) -> "UnaryTypeTypeType<id, type>" ^ ;
    | ArrayTypeTypeTypeTypeType(id) -> "ArrayTypeTypeTypeTypeType<id>" ^ ;
    | ListTypeTypeTypeTypeType(id) -> "ListTypeTypeTypeTypeType<id>" ^ ;
    | TupleTypeTypeTypeTypeType(id) -> "TupleTypeTypeTypeTypeType<id>" ^ ;
    | OptionTypeTypeTypeTypeType(id) -> "OptionTypeTypeTypeTypeType<id>" ^ ;
    | StructTypeTypeTypeTypeType(id) -> "StructTypeTypeTypeTypeType<id>" ^ ;
    | FuncTypeTypeTypeTypeType(id) -> "FuncTypeTypeTypeTypeType<id>" ^ ;
    | JoinTypeTypeTypeTypeType(id, types) -> "JoinTypeTypeTypeTypeType<id, types>" ^ ;
    | BinaryTypeTypeTypeType(id, types) -> "BinaryTypeTypeTypeType<id, types>" ^ ;
    | UnaryTypeTypeTypeType(id, type) -> "UnaryTypeTypeTypeType<id, type>" ^ ;
    | ArrayTypeTypeTypeTypeTypeType(id) -> "ArrayTypeTypeTypeTypeTypeType<id>" ^ ;
    | ListTypeTypeTypeTypeTypeType(id) -> "ListTypeTypeTypeTypeTypeType<id>" ^ ;
    | TupleTypeTypeTypeTypeTypeType(id) -> "TupleTypeTypeTypeTypeTypeType<id>" ^ ;
    | OptionTypeTypeTypeTypeTypeType(id) -> "OptionTypeTypeTypeTypeTypeType<id>" ^ ;
    | StructTypeTypeTypeTypeTypeType(id) -> "StructTypeTypeTypeTypeTypeType<id>" ^ ;
    | FuncTypeTypeTypeTypeTypeType(id) -> "FuncTypeTypeTypeTypeTypeType<id>" ^ ;
    | JoinTypeTypeTypeTypeTypeType(id, types) -> "JoinTypeTypeTypeTypeTypeType<id, types>" ^ ;
    | BinaryTypeTypeTypeTypeType(id, types) -> "BinaryTypeTypeTypeTypeType<id, types>" ^ ;
    | UnaryTypeTypeTypeTypeType(id, type) -> "UnaryTypeTypeTypeTypeType<id, type>" ^ ;

(* angelaZ compiler

1. give stdin to scanner, get tokens
2. give tokens to parser, get AST
3. give AST to analyzer, get semantic tree
4. give semantic tree to java converter, get java tree
5. give java tree to java code generator, get java code
6. give java code to java compiler, get executable
7. run java executable

*)

type action = Sast | Java

let _ = if Array.length Sys.argv > 1 then
    List.assoc Sys.argv.(1) [ ("-s", Sast); ("-j", Java)]
else Java in

let lexbuf = Lexing.from_channel stdin in
let program = Parser.program Scanner.token lexbuf in
match action with
| Sast ->
    let result = Typecheck.check_program program in
    ()
| Java ->
    let result = JavagenTest.gen_program "Output" program in
    print_string result
Matrix main2(Int argc, String argv) {
    Matrix m3(2,2);
    Matrix m(2,2);
    m[0][0]=1;
    m[0][1]=2;
    m[1][0]=3;
    m[1][1]=4;
    m3 = (((m + m') .* m) * 4) + m^;
    return m3;
}

Void main(Int argc2, String m) {
    Matrix result(2,2);
    result=main2(0, "str");
    printM(result);
}

21
Structure main2(Int argc, String argv) {
    Structure s={a="1", b=toString(argc)};
    i=toInt(s -> a);
    return s;
}

Void main(Int argc2, String m) {
    Structure result={};
    result=main2(0, "str");
    print(result);
}
Option main2(Int argc, String argv) {
    Option s={strike="100.0", stock="150.0", interestRate="0.1", period="1.0", sigma="2.0", optionType="call"};
    i=toFloat(s -> strike);
    return s;
}

Matrix main3(Int a) {
    Matrix strike(1,2);
    strike[0][0]=10;
    strike[0][1]=20;
    Matrix stock(1,2);
    stock[0][0]=15;
    stock[0][1]=25;
    Matrix interestRate(1,2);
    interestRate[0][0]=0.4;
    interestRate[0][1]=0.3;
    Matrix period(1,2);
    period[0][0]=3;
    period[0][1]=4;
    Matrix sigma(1,2);
    sigma[0][0]=0.1;
    sigma[0][1]=0.2;
    Matrix s(1,1);
    s= priceM(strike,stock,interestRate,period,sigma);
    return s;
}

Void main(Int argc2, String m) {
    Option result={};
    result=main2(0, "str0");
    Float d;
    d=price(result);
    print(d);
    Matrix result1(1,1);
    result1=main3(0);
    print(result1);
}
OBJS = ast.cmo sast.cmo parser.cmo scanner.cmo typecheck.cmo javagen.cmo angelaZ.cmo javagenTest.cmo angelaZTest.cmo

OBJS_TEST = ast.cmo sast.cmo parser.cmo scanner.cmo typecheck.cmo javagenTest.cmo angelaZTest.cmo

angelaZ: $(OBJS)
    ocamlc -g -o angelaZ $(OBJS)

calc: $(OBJS)
    ocamlc -o calc $(OBJS)

angelaZTest: $(OBJS_TEST)
    ocamlc -g -o angelaZTest $(OBJS_TEST)

testall: testall.sh
    ./testall.sh

calculator.ml: calculator.mll
    ocamllex calculator.mll

parser.ml parser.mli: parser.mly
    ocamlyacc parser.mly

%.cmo: %.ml
    ocamlc -w -A -c $<

%.cmi: %.mli
    ocamlc -w -A -c $<

#calc.tar.gz: $(TARFILES)
    cd .. & & tar czf calc.tar.gz $(TARFILES: % = calc/%)

clean:
    rm -f calc parser.ml parser.mli scanner.ml testall.log \
    *.cmo *.cmi *.out *.diff

%: Generated by ocamldep *.ml *.mli

typecheck.cmo: ast.cmo ast.cmx

typecheck.cmx: ast.cmx ast.cmx

ast.cmo:
    ast.cmx:
        @ #calc.cmo: scanner.cmo parser.cmi ast.cmo
        @ #calc.cmx: scanner.cmx parser.cmx ast.cmx
        @ parser.cmo: ast.cmo parser.cmi
        @ parser.cmx: ast.cmx parser.cmx
        @ scanner.cmo: ast.cmo
        @ scanner.cmx: parser.cmx
        @ parser.cmi: ast.cmo
        @ javagen.cmo: ast.cmo
        @ javagen.cmx: ast.cmx
        @ javagenTest.cmo: ast.cmo
        @ javagenTest.cmx: ast.cmx
        @ pc.cmo: scanner.cmo parser.cmi javagen.cmo ast.cmo javagenTest.cmo
        @ pc.cmx: scanner.cmx parser.cmx javagen.cmx ast.cmx javagenTest.cmx
#!/bin/sh

MICROC=./angelaZTest

# Set time limit for all operations
ulimit -t 30

#globallog=testall.log
rm -f $globallog
error=0
globalerror=0
keep=0

Usage() {
  echo "Usage: testall.sh [options] [.mc files]"
  echo "-k    Keep intermediate files"
  echo "-h    Print this help"
  exit 1
}

SignalError() {
  if [ $error -eq 0 ]; then
    echo "FAILED"
    error=1
  fi
  echo "$1"
}

# Compare <outfile> <reffile> <difffile>
# Compares the outfile with reffile. Differences, if any, written to difffile
Compare() {
  generatedfiles="$generatedfiles $3"
  echo diff -b $1 $2 >"$3" 1>&2
  diff -b "$1" "$2" >"$3" 2>&1 || {
    SignalError "$1 $2 differs"
    echo "FAILED $1 differs from $2" 1>&2
  }
}

# Run <args>
# Report the command, run it, and report any errors
Run() {
  echo $* 1>&2
  eval $* || {
    SignalError "$1 failed on $*"
    return 1
  }
}

Check() {
  error=0
  basename=`echo $1 | sed 's/.*\///'
  s/.mc//`
  reffile=`echo $1 | sed 's/.mc$//'`
  basedir="`echo $1 | sed 's/\/\[^\//]*/\.&/.'`
  echo -n "$basename..."
  echo 1>&2
  echo "###### Testing $basename" 1>&2
  generatedfiles=""
  generatedfiles="$generatedfiles ${basename}.i.out"
  Run "$MICROC"  <"$1" >"${basename}.i.out"
  Compare "${basename}.i.out" ${reffile}.out "${basename}.i.diff"
  generatedfiles=""
  generatedfiles="$generatedfiles ${basename}.c.out"
  Run "$MICROC"  <"$1" >"${basename}.c.out"
  Compare "${basename}.c.out" ${reffile}.out "${basename}.c.diff"
  # Report the status and clean up the generated files
  if [ $error -eq 0 ]; then
    if [ $keep -eq 0 ]; then
      rm -f $generatedfiles
      echo "OK"
      echo "###### SUCCESS" 1>&2
    else
      echo "###### FAILED" 1>&2
      globalerror=$error
    fi
  fi
}

while getopts kdpsh c; do
  case $c in
    k) keep=1
      # Keep intermediate files
      break
    esac
  esac

echo "$1"

```
90   keep=1
91   ;
92 h) # Help
93    Usage
94   ;
95   esac
96 done
97
98 shift 'expr $OPTIND - 1'
99
100 if [ $# -ge 1 ]
101 then
102   files=$@
103 else
104   files="tests/fail-*.az tests/test-*.az"
105 fi
106
107 for file in $files
108 do
109   case $file in
110     *test-*)
111       Check $file 2>> $globallog
112         ;
113     *fail-*)
114       CheckFail $file 2>> $globallog
115         ;
116     *)
117       echo "unknown file type $file"
118       globalerror=1
119         ;
120     esac
121   done
122
123 exit $globalerror
124
```