PANAMX

Final Project Report

Project Manager: James Wong (tw2686)
Language Guru: Brian (Byeongho) Hwang (bh2656)
System Architect: Xingchen Li (xl2801)
Tester: Zach Boughner (ztb2003)

1 Introduction

Matrix manipulation has very important applications in many scientific disciplines, yet many standard general-purpose programming languages (e.g., Java, C, Python) lack the built-in features to facilitate easy use of matrices when writing programs. Implementing the complex operations needed for many advanced matrix applications is prohibitive to practitioners who lack significant programming experience. Though libraries that support matrix manipulation do exist, they still serve as a barrier to entry to most. PANAMX is an imperative, highly intuitive and user friendly programming language specialized for matrix manipulation.

In order to make it intuitive and efficient, we implemented the matrix data type that is widely used in the areas of scientific computations that involves linear algebra. We also implemented a rich library of built-in matrix operators and functions for the convenience of users. The syntax of PANAMX is similar to that of C, which makes it intuitive to learn and use.

2 Tutorial

This tutorial assumes a certain familiarity with programming languages, command-line arguments, and the use of git. Use `git clone` to download a local copy of our repository.

```
git clone https://github.com/zboughner/coms-4115_PANAMX.git
```

2.1 Environment Setup

PANAMX is developed using OCaml and LLVM libraries. Please install the following packages or ensure that the dependencies are already installed on your machine. Can use package manager to install dependencies such as `homebrew` on mac or `apt-get` on Ubuntu on Windows.

```
ocaml
opam
llvm
```

2.2 Build the Compiler

Navigate to the source folder of the repository using cd PANAMX. Then, run make. This will build the compiler and link the relevant *.o files from C. Lastly, run ./testall.sh to test the compiler against our test cases.

2.3 Basic Syntax in PANAMX

PANAMX has a similar syntax to C and program structure. Executables run from their main functions and all functions must have a return statement. In addition to the basic types supported by C, PANAMX also supports strings, matrices and struct data types.

2.4 Programming Examples in PANAMX

Below are two example programs in PANAMX, which shows both matrix manipulation and struct declaration cases.

Matrix:

```
int main() {
    matrix m;
    matrix n;
    m = [1.6, 3.9;
         2.4, 4.3;
         5.7, 7.2];
    n = [12.3, 15.6, 14.5, 18.8;
         23.4, 21.8, 9.5,  10.7];
    m = m * n;
    printm(m);
    return 0;
}
/* output */
[
      110.940   109.980   60.250   71.810
      130.140   131.180   75.650   91.130
```

Struct:

```
struct Node {
    int x;
    int y;
};

int main() {
    struct Node p;
```

```
    p = <struct Node>;
    p.x = 121;
    p.y = 999;
    print(p.x);
    print(p.y);
    return 0;
}
/* output */
121
999
```

3 Language Reference Manual

3.1 Data Types

3.1.1 Primitive Data Types

There are seven kinds of basic data types in our language: int, double, boolean, void, string, matrix, struct.

3.1.1.1 Integer

The integer type represents a 32-bit signed integer.

```
int a;
a = 1;
```

The language does not automatically convert a double to an integer if a double is assigned to a variable of type integer.

3.1.1.2 Double

The double type represents a 64-bit float-point number.

```
double b;
b = 2.0;
```

The language automatically promotes an integer to a double if an integer is assigned to a variable of type double.

3.1.1.3 Boolean

The boolean type is an 8-bit data that represents true or false.

```
bool e;
e = true;
```

3.1.1.3 Void

This returns no type.

3.1.1.4 String

This is the type an array of characters

```
string s;
s = “example of string”;
```

3.1.2 Derived Data Types

3.1.2.1 Matrix

Matrix is a special data structure that comprises of a two-dimensional array, and two integers that represent the number of rows and columns of this matrix. The matrix data type allows the user to build *n* x *m* matrix of double types, where n is the number of columns and m is the number of rows of the matrix. Matrix values can input integers and doubles. When the input contains integers, the matrix type will simply convert them into doubles. Similarly, printm(), which is our function to print matrix will always print matrices contained in square brackets, while also printing each value in doubles form with three decimal places for significance. The matrix type is defined in C library, and linked to the LLVM compiled native code. The following is how it is defined in C code.

```
typedef struct Matrix
{
    int row;
    int col;
    double **mat;
} *matrix;
```

3.1.2.2 Struct

Our language also supports struct that allows the user to define customized data structures. We can define members of a struct with integer, boolean, double, and matrix type. We can declare, initialize, and access the members of the struct. The struct type can be defined such as the following

```
struct STRUCT_NAME
{
      TYPE_NAME NAME_1;
      TYPE_NAME NAME_2;
};
```

3.2 Lexical Conventions

3.2.1 Tokens

There are five classes of tokens: identifiers, keywords, constants, operators, and separators.

3.2.2 Identifiers

Identifiers are sequences of characters used for naming variables and functions. Characters consist of letters, digits, and the underscore character '_'. Identifiers are also case sensitive such that '`A`' and '`a`' can be different identifiers. An identifier cannot be the same as any existing keywords. An identifier cannot have the same name even if they are different types.

3.2.3 Keywords

Keywords are special identifiers reserved for use by the language.

```
int, double, boolean, matrix, if, else, for, while, return, void,
true, false, struct
```

3.2.4 Literals

3.2.4.1 Integer

An integer literal is a sequence of digits, with an optional prefix. Sequences of digits preceded by '`0x`' will be taken to be hexadecimal, and '`0`' will be taken to be octal. Otherwise, no prefix will be taken as decimal.

```
1;
```

3.2.4.2 Double

A double literal consists of an integer part, a decimal point, and a fraction part. Both the integer and fraction part consist of a sequence of digits, where either parts can be empty, but not both. For instance '`1.`' and '`.1`' are allowed.

```
1.0;
0.1;
```

3.2.4.3 String Literal

String literals or string constants are sequences of zero or more characters, digits, and escape sequences enclosed within double quotation marks.

```
"this is a string example.";
```

3.3 Operators

Operators are special tokens reserved for use by the language for operations and may not be used otherwise. Refer to the expressions section for functionality and use cases.

3.3.1 Basic Operators

Symbol	Description
=	Assignment
+	Addition
-	Subtraction
*	Multiplication

/	Division
++	Increment by one
−−	Decrement by one
!	Logical NOT
&&	Logical AND
||	Logical OR

3.3.2 Relational Operators

Symbol	Description
>	Greater than
<	Less than
<=	Greater than or equal to
>=	Less than or equal to
==	Equivalence test (by value)
!=	Non-equivalence test (by value)

3.3.3 Matrix Operators

Basic operators for matrix also works as expected, such as the following ones

1. Matrix Arithmetic

The addition and subtraction operators work similarly, allowing users to add or subtract two matrices with the same dimensions and type

```
matrix X;
matrix Y;
X = [1, 2 ; 3, 4];
Y = [-1, -2; 0, 1];
```

```
X = X + Y;
printm(X);
[
    0.000  0.000
    3.000  5.000
]
```

2. Matrix Multiplication

The multiplication operator allows the user to multiply two matrices of varying dimensions. However, keep in mind rules from linear algebra: the width of the first matrix has to equal the height of the second matrix.

```
matrix X;
matrix Y;
X = [1, 2 ; 0, 1; -1, 0];
Y = [1, 5; 0, 1];
X = X * Y;
printm(X);
[
   1.000   7.000
   0.000   1.000
  -1.000  -5.000
]
```

3. Element-Wise Multiplication, Division

An element-wise multiplication means the multiplication of two numbers in the same position in their matrix. The two matrices should have the same dimensions.

```
matrix X;
matrix Y;
matrix a;
matrix b;
X = [2, 6; 12, 20];
Y = [1, 2; 3, 4];
a = X .* Y;  // element-wise multiplication
printm(a);
[
```

```
   2.000 12.000
   36.000 80.000
]

b = x ./ y;  // element-wise division
printm(b);
[
   2.000 3.000
   4.000 5.000
]
```

4. Indexing

Matrices can be index using matrix_name[i][j], where i is the row index, and j is the column index. Indexes start count from 0.

```
// Accesses element in the 4rd row and 5th column
m[3][4];
```

5. Slicing

Matrices can also be sliced to smaller dimensions, by using matrix_name[i:j][k:l], where i is the beginning index of the row, j is the non inclusive ending index of the row, k is the beginning index of the column, while l is the non inclusive ending index of column. For instance, if you want to know the number of rows returned by slicing, subtract i from j.

```
// Accesses elements in the 3nd row and 2nd to 5th columns
m[2:3][1:5];
```

3.3.4 Operator Precedence and Associativity

Operator (Ranked from high to low precedence)	**Associativity**
() [] .	L-R
! -	R-L
* / %	L-R
+ -	L-R

< <= > >=	L-R
== !=	L-R
&	L-R
^	L-R
|	L-R
&&	L-R
||	L-R
= += -= *= /=	L-R

3.3.5 Separators

Separators are specific tokens reserved for use by the language and may not be used otherwise. Separators are used to denote separation between tokens. Whitespace is considered a separator.

```
( ) [ ] { } ; , . :
```

3.3.6 Whitespace

Whitespace consists of the space character, the tab character, and the newline character. Whitespace is ignored and optional, when its not used to separate tokens. Whitespace is not required between operators, nor is it required between other separators. For example:

```
matrix foo ( matrix A ) {
     matrix B;
     B = [1, 1; 1, 1];
     return A + B;
}
```

The above function can be written as the following:

```
matrix foo(matrix A){matrix B; B=<2,2>; B=[1,1;1,1];return A+B;}
```

As mentioned, whitespace is used as separators of tokens such as the following:

```
string s;
```

```
s = "this is a string example";
```

The above is not equivalent to below:

```
string s;
s = "thisisastringexample";
```

3.3.7 Comments

Single line comments are introduced with two consecutive forward-slash characters. Multiline comments are introduced by a forward-slash immediately followed by an asterisk and terminated by an asterisk followed by a forward-slash.

```
// Single line comment example

/* Multiline
comment example */
```

3.4 Syntax

3.4.1 Basic Expression

These basic expressions are the building blocks of more complicated statements in this language: an *identifier*, given that it has been declared in the way described further below, any *constant*, any *string literal*, or any *function invocation* that returns a value (e.g., does not return `none`).

Expression	Example
Identifier	`a = 1; // val = 1`
Constant	`2 // val = 2`
String Literal	`“Example” // val = “Example”`
Function Invocation	`gcd(8, 12) // val = 4`

3.4.2 Complex Expressions

More complicated expressions may be formed by manipulating these basic expressions as the operands of some operator:

```
EXPRESSION OP EXPRESSION;
```

For example:

```
1 + gcd(8, 12); // val = 5

gcd(10 - 2, 12) + 1; // val = 5
```

Furthermore, we use parentheses to group subexpressions and to determine order of evaluation, which gives precedence to the innermost subexpressions:

```
(EXPRESSION OP EXPRESSION) OP EXPRESSION;
```

For example, in the following code snippet, the innermost subexpressions are evaluated first:

```
(poweri(10, 2)) - (gcd(2, 4) + (-2 + 1)) // val = 99
```

3.4.3 Declaration and Initialization

3.4.3.1 Primitive Types

Variables are declared on one line, and initialized in a separate line. Any variables that need to be declared must be done on the top of the program, and any initialization must be done after all declarations. Initialization can be done using the assignment operator, provided that the programmer provides matching right hand side *value* of that same type of the id provided on the left hand side. The values will be type-checked based on the type and value provided. For example:

```
int x;
double y;
boolean b;
string s;
x = 1;
y = 1.0;
b = true;
string s  = 'this is a string';
```

This covers declaration and initialization for our language's primitive types. For the language's derived types (e.g., `struct`, `matrix`), declaration and initialization can be a bit more nuanced. See below.

3.4.3.2 Derived Types

1. Matrix

As mentioned previously, the `matrix` type is defined using the C library. It's a special case because the underlying two-dimensional array requires that some dimensional information be provided by the user at declaration. A programmer can provide the needed dimensional information at initialization, but can also initialize without it. Dimensional information may be provided explicitly by providing integer values for the number of rows and columns, in that order, between a '<' character and a '>' character and separated by a comma. If no data is provided between the following brackets, then a `matrix` is constructed according to the provided dimensions with all values initialized to zero:

```
matrix m;
m = <2, 3>; // initializes a 2 x 3 matrix of zeroes
```

The user can also initialize the values of the matrix by using square brackets, with each value in the same row separated by comma and each row separated by semicolon. This can also be split in multiple separate lines. The values can be either integer or doubles. If the values inputted are integers, the matrix data type will automatically convert the integers to doubles when the matrix is initialized.

```
m = [1, 2; 3, 4]; // or
m = [1, 2;
     3, 4];
```

If the user initializes the matrix with square brackets values with different dimensions after declaring the dimensions. For instance

```
m = <4,3>;
m = [1, 2; 3, 4];
/* output */
[
   1.000  2.000
   3.000  4.000
]
```

The matrix will simply take the values and dimensions of the latest one.

2. Struct

Earlier, we mentioned how to define a user-defined data type, called a `struct`, in our language. Again, structs can be defined with member data types or bool, double, or matrix type. For example

```
struct Node {
   int aa;
   bool bb;
   double cc;
   matrix dd;
};
```

Now that we have node struct defined, we show declaration and initialization for user-defined data types:

```
struct Node n;
n = <struct Node>;
```

Finally, we can access the members of the struct by appending a point after the declared struct id, and followed by the member name. For example.

```
n.cc = 1.23;
n.dd = [1, 2; 3, 4];
printf(n.cc);
printm(n.dd);
/* output */
1.23
[
   1.000  2.000
   3.000  4.000
]
```

Note that users are not allowed to provide a partial list of attribute values at initialization. This will throw an error.

3.5 Statements and Control Flow

Unless otherwise indicated by control flow, statements are executed in sequential order.

3.5.1 Basic Statement

Any statement in our language is terminated by a semicolon, and a simple statement could be one of the following: a *basic expression*, a *complex expression*, *variable assignment*, *variable declaration*, a *function call*, or a *return* statement.

3.5.2 Compound Statement

A compound statement is a sequence of statements, usually enclosed in braces and managed by some control flow. What follows is a description of some paradigms for compound statements supported by our language.

3.5.3. For Loop

Executes body until *termination condition* is false; initial index value defined by *initialization statement* and updated by *increment rule.*

```
for (INITIALIZATION_STATEMENT; TERMINATION_CONDITION; INCREMENT_RULE)
{
      // do something
}
```

3.5.4 While Loop

Executes body until the *condition* is false.

```
while (CONDITION)
{
      // do something
}
```

3.5.5 Conditional Statement

Executes body only if the *condition* is true.

```
if (CONDITION)
```

```
{
      // do something
}
```

Executes the first statement if the *condition* is true, or executes the second statement if it is false. The `else` must immediately follow the body of `if` (*condition*)

```
if (CONDITION)
{
      // do something
}
else
{
      // do something
}
```

3.5.6 Return Statement

Terminates the current function and returns to its caller the contents of the following expression. If expression it not specified, none is returned.

3.6 Scope

Scope refers to the visibility of entities within a block, function, or file. Declarations made at the top of the file, and not within the function, are visible to the entire file. These declarations unless specified otherwise can also be visible within said functions. On the other hand, declarations made inside functions are only visible within that function. Preceding declarations cannot see declarations made after it.

For example, the following will work because ‘`a`’ is declared before/outside of function `foo`

```
int a;
a = 0;
int foo() {
      return a + 1;
}
```

The following will not work, since ‘`a`’ is declared within the function, yet is being called outside

```
int b;
```

```
int foo() {
     int a;
     a = 0;
     return a;
}
b = a + 1;
```

Furthermore, the following will work, since ‘a’ is initialized before ‘b’

```
int a;
int b;
a = 1;
b = a + 1;
```

The following will not work, since ‘b’ is declared before ‘a’

```
int a;
int b;
a = b + 1;
b = 1;
```

3.7 Functions

Most functions defined in PANAMX standard library resemble much with those in C libraries and Java API. They support manipulations of matrix that are defined in C library. This section covers how to define a function, as well as definitions of math, matrix that are built in PANAMX.

3.7.1 Function Definition

Functions must be defined in any context before they are used, following this structure:

```
RETURN_TYPE FUNCTION_NAME (ARGUMENT_1, ARGUMENT_2, ...)
{
     FUNCTION_BODY;
}
```

Unless the return type of a function `void`, a function must return a valid value with corresponding type as declared in the function header. See the following simple example that

takes two matrices and concatenates them from top to bottom, and returns a merged matrix of both input matrices.

```
matrix combineTopDown(matrix a, matrix b) {
    int i;
    int j;
    int arow;
    int acol;
    int brow;
    int bcol;
    matrix new;

    arow = matrixHeight(a);
    acol = matrixWidth(a);
    brow = matrixHeight(b);
    bcol = matrixWidth(b);

    if (acol != bcol) {
        prints("Cannot combine matrices with different widths");
        return a;
    }

    new = <arow+brow, acol>;
    for (i = 0; i < arow; i++) {
        for (j = 0; j < acol; j++) {
            new[i][j] = a[i][j];
        }
    }
    for (i = arow; i < arow+brow; i++) {
        for (j = 0; j < bcol; j++) {
            new[i][j] = b[i-arow][j];
        }
    }
    return new;
}
```

3.7.2 Function Calls

Functions are called by specifying the function name and its parameters, as follows:

```
FUNCTION_NAME(ARGUMENT_1, ARGUMENT_2, ...);
```

Function calls can also be used as part of an expression:

```
matrix x;
x = comebineTopDown(a, b);
```

3.8 Built-in Functions

3.8.1 Math Functions

The language will support more mathematical functions than those listed below, but here is a list of functions will be frequently used:

Function	**Description**
`sqrti(int n)|sqrtd(double d)`	Returns square root of the argument.
`nrooti(int n)|nrootd(double d)`	Returns nth root of the argument.
`absi(int n)|absd(double d)`	Returns absolute value of the argument.
`poweri(int n)|powerd(double d)`	Returns nth power of the argument.

3.8.2 Matrix Functions

PANAMX is a programming language focusing on matrix manipulation, and thus it contains many useful functions that easily do complex linear algebra calculations:

Function	**Description**
`matrixHeight(matrix m)`	Returns number of rows of the matrix
`matrixWidth(matrix m)`	Returns number of columns of the matrix
`sum(matrix m)`	Returns the sum of all elements of the matrix
`mean(matrix m)`	Returns the mean of all elements in the matrix
`trans(matrix m)`	Returns the transposed version of the matrix

`det(matrix m)`	Computes the eigenvalues of the matrix
`rank(matrix m)`	Returns the rank of the matrix
`rref(matrix m)`	Returns the matrix in rref form

4 Project Plan

4.1 Team Process

Planning Process: Once we assembled our team, we met multiple times the first two weeks for long sessions to brainstorm ideas for our project. We also used that time to set up a WeChat group, and Google Docs for our project ideas. In addition to the TA's feedback, and our unfamiliarity with building compilers and designing languages, we decided that the four of us were most comfortable with working on a language enabling matrix manipulation. Then, we proceeded to create the project proposal, and following the language reference manual. We worked on both of these together, since we needed to decide the basic syntax and grammar of our language. We then scheduled regular weekly meetings and adjusted the times depending on everybody's changing schedules.

Specification Process: We originally had many build-in features we wanted to include such as file I-O for data manipulation, and other matrix functions. After meeting with the TA, we decided it would be best to focus on implementing the matrix, and user defined data structure struct, and made sure that they worked properly.

Development Process: After the LRM, we started working on creating the string data type to enable the Hello World program. This allowed everybody to become familiar with the structure of the compiler, the architecture. We ended up implementing a prints() function that specifically prints strings.

Afterwards, we immediately proceeded to implementing increment and decrement operators, as further practice. Then, we discussed possible ways to implement matrix, and tested many different implementations. We kept updating the language reference manual based on our implementations as we worked. We ended up defining our matrix using C library, and linked it using the codegen file. We then worked on modifying the matrix declaration and initialization, so that we can initialize without specifying dimensions. We also worked on the parser and ast to specify the format of our declaration to the same as what we planned in the LRM. With most

major changes, we made sure to keep each team member up to date, and had at least one other person approve before commiting to github.

After successfully implementing matrix, we moved onto developing the user defined data type, struct, and used a similar process of testing multiple versions, starting with the most simple. After that, we worked on implementing build-in matrix functions in the C library, and continued to fix any bugs that manifested through consistently rigorous testing.

Testing Process: For every new feature implemented, we added a new test case that demonstrated its functionality. For more information, see section TESTING.

Workflow: The team used its first week or two to outline team norms and our overall project workflow. The focal point of our workflow were our weekly meetings. In order to accommodate everyone's very different and fully-loaded schedules, we did not have a regular time for this meeting. Rather, we would determine the time and location of each at the end of the previous meeting.

At the beginning of the project, we established this meeting time as an opportunity for the team to get on the same page logistically as well as a time to get some tangible work done on the project together, in the same physical space. Therefore, we stipulated that everyone should attend the meeting for at least one hour and should try to longer, schedule-permitting. This time requirement meant that everyone would be there to discuss logistics and that it was strongly encouraged that everyone stay as long as possible to take advantage of the opportunity to work together. It also gave us all a reasonable expectation for meeting length when planning our other commitments. In general, these weekly meetings lasted 2-3 hours each on average.

Each meeting would begin with everyone sharing updates on what they've been working on and learned in the past week that might be relevant to the project. Then, we would proceed by having time for general questions and possible answers from other group members. We would then outline our goals and action items for the upcoming week and assign specific tasks before breaking to start the "programming" portion of the meeting. Whenever the first person had to leave, we would determine the following meeting time and location.

In addition to these weekly meetings we had, of course, bi-weekly meetings with our advisor, Ryan Bernstein. These would occur at 2:00 PM on Friday and would last for thirty minutes. We frequently took time from our weekly meeting to coordinate how we might best use this time with our TA. Usually, we would show him our progress, and then proceed to ask him some "spotlight" questions that have really been bugging us.

Finally, at the week before an assignment is due, we would have ad-hoc meetings for finishing the required work. Usually, this meant meeting as often as everyone could for long stretches of time, and working together to try and complete the objective. We frequently worked on the project remotely, especially during crunch times; however, we tried to minimize this as we felt like everyone was more productive, and effective, when working together in the same physical location.

Communication: Our group utilized WeChat to manage communications between members. We chose this particular messaging application because most of us were familiar with the technology, and it had useful features such as group chats, video chats, and file uploads. All of our group members were responsive to messages and replied immediately. This helped us work around each other's schedule, communicate ideas/bugs/issues, and assign tasks. After a few

4.2 Style Guide

Our group followed the style guide below when make any code changes to PANAMX. Our goal was to ensure readability, and ease of navigation.

1. Correct indentation of each block of OCaml code. One tab distance for anything inside another block.
2. Add new-line and comments if necessary in between each block of code.
3. Follow C-program style in the test files.
4. Test files end in .px.
5. Use descriptive names and acronyms to name functions.
6. Write functions for any piece of code which has been used more than a few times.
7. Git commit messages should be substantive. Ex: "Fixed index error bug in semant.ml" instead of "fixed bug".

4.3 Project Timeline

Milestone	**Week(s)**
Introductions, Project Workflow	1
Proposal	3
Language Reference Manual	6
"Hello World"	10
Implement Matrix	11-12
Implement Struct	13

Implement Functions	14
Presentation	15
Final Report	15-16

4.4 Project Roles

Role	**Student**	**UNI**
Manager	James Wong	tw2686
System Architect	Xingchen Li	xl2801
Language Guru	Byeongho Brian Hwang	bh2656
Tester	Zach Boughner	ztb2003

4.5 Project Tools

Programming Languages: OCaml, C, Bash
Version Control: Git
Repository Management: GitHub
Testing: Bash
Editors: Vim, VS Code, XCode, Atom, Vim, Sublime Text
Platforms: MacOS 10.14.3 Mojave, Ubuntu (via VMWare)
Documentation: Google Docs, Google Slides, Google Sheets
Communication: WeChat, Gmail

4.6 Project Log

The two charts depict our workflow from GitHub. The first shows the number of additions and deletions per week. The second shows the number of commits.

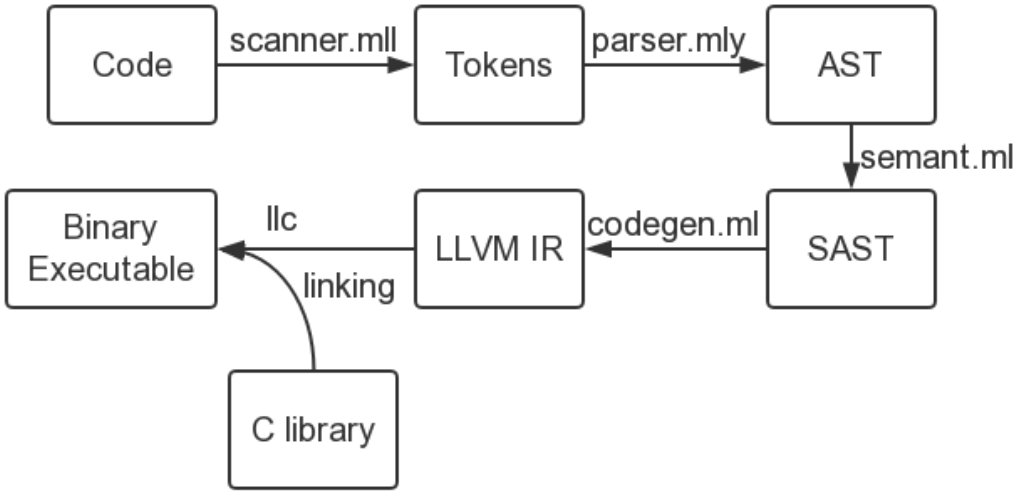
5 Language Evolution

Ultimately, the development of PANAMX focused on two core features that the team agreed were minimum requirements for our final deliverable. Those two core features were implementing a matrix type and a user-defined data structure. The team implemented other features after we felt comfortable with the outcomes of those goals. In our LRM, we pictured our language having a lot of C-like features and syntax. However, we focused on implementing the core features of our language as opposed to some “nice-to-have” things that we described in our LRM. Throughout the project, we realized some features were unfeasible with the time we had, and instead focused on the core features that we mentioned earlier.

For example, our LRM described a “swiss-army-knife” type print function that could print all types seamlessly. However, we ended up only implementing type-specific print functions, and we did not have the time to revisit this at the end of the project. Another feature we initially wanted was file input output functionalities for csv data parsing. After speaking with the TA, we realized implementing this feature would also be unfeasible and decided to remove it. Finally, while implementing core features such as the matrix data type, we went through many different versions. For instance, the first version of matrix we implemented was defined using LLVM libraries as a two dimensional array in codegen. Unfortunately, we found this implementation to be limited, and we had difficult to allow initializing matrix with optional dimensions input, and matrix assignment. After a few versions, we ended up defining the matrix data type in C, which allowed us to implement all the build-in functions such as matrix slicing, and matrix rref in C. In the end, we successfully met our goal, and implemented matrix data type with built-in operators and functions, and user defined data structure structs.

6 Architectural Design

6.1 Diagram



6.2 Compiler Components

Scanner

The scanner is the lexical analyzer and it can parse the source code from a sequence of characters into tokens. During this phase, the white spaces are taken out and tokens are generated for anything that has syntactic meaning our language. This includes any literals, integer, string, and double, variable names, matrix, struct types.

Parser
The parser takes the tokens generated from the scanner and merges them into abstract syntax tree (AST). Meanwhile it can detect some basic syntax errors. Our implementation resembles that of the MicroC compiler.

Semant
Semant represents the semantic checking phase, and it takes the AST and yields the semantic checked abstract syntax tree (SAST). It can also detect semantic errors like duplicate names and type mismatch. For matrices, the semantic checking checks for valid index types when indexing or slicing matrix, checks for whether matrix dimensions are zero, checks for whether values being passed are either integer or double type, and checks for whether matrix dimensions are integers. For structs, semantic checking checks for duplicate names and whether the members of the struct are assigned the matching value type.

Codegen
The code generator will take the SAST and translate it into LLVM IR code. We also declare the built-in functions for matrix manipulation and mathematics written in C here, and pass the corresponding arguments. The codegen is then responsible for linking the IR code to our C library.

C Library
We developed a C library that defines the matrix structure here and other built-in functions. This C library is compiled then linked to the LLVM.

7 Test Plan

7.1 Test Programs

The following are several programs written using PANAMX language, and the expected output of the programs, and the generated LLVM code. We included simple unit tests, failure tests, and demo tests.

7.1.2 Matrix

The following is a simple unit test program, which tests declaring, initializing, and printing matrices in PANAMX.

```
int main() {
    matrix m;
    m = [1, 2, 3;
         4, 5, 6;
         7, 8, 9];
    printm(m);
    return 0;
}
```

Below is the output of this program.

```
[
      1.000   4.000   7.000
      2.000   5.000   8.000
      3.000   6.000   9.000
]
```

Below is the generated LLVM code for this program.

```
; ModuleID = 'Panamx'
source_filename = "Panamx"

%Matrix = type opaque

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1

declare i32 @printf(i8*, ...)

declare %Matrix* @buildMatrixEmpty(i32, i32)

declare double @matrixAccess(%Matrix*, i32, i32)

declare double @matrixAssign(%Matrix*, i32, i32, double)

declare %Matrix* @matrixSlice(%Matrix*, i32, i32, i32, i32)
```

```
declare i32 @printMatrix(%Matrix*)

declare i32 @freeMatrix(%Matrix*)

declare i32 @getHeight(%Matrix*)

declare i32 @getWidth(%Matrix*)

declare %Matrix* @addMatrixDouble(%Matrix*, double, i32)

declare %Matrix* @addMatrixMatrix(%Matrix*, %Matrix*, i32)

declare %Matrix* @subDoubleMatrix(double, %Matrix*)

declare %Matrix* @mulMatrixDouble(%Matrix*, double, i32)

declare %Matrix* @mulMatrixMatrix(%Matrix*, %Matrix*)

declare %Matrix* @mulElementWiseMatrix(%Matrix*, %Matrix*, i32)

declare double @sum(%Matrix*)

declare double @mean(%Matrix*)

declare %Matrix* @trans(%Matrix*)

declare %Matrix* @rref(%Matrix*)

declare double @rank(%Matrix*)

declare double @det(%Matrix*)

declare %Matrix* @inv(%Matrix*)

declare %Matrix* @concatTB(%Matrix*, %Matrix*)

declare %Matrix* @concatLR(%Matrix*, %Matrix*)

declare i32 @sqrti(i32)

declare double @sqrtd(double)

declare i32 @nrooti(i32, i32)

declare i32 @nrootd(i32, double)

declare i32 @absi(i32)
```

```
declare double @absd(double)

declare i32 @poweri(i32, i32)

declare double @powerd(double, i32)

define i32 @main() {
entry:
  %m = alloca %Matrix*
  %matrix_array = alloca [9 x double]
  store [9 x double] [double 1.000000e+00, double 2.000000e+00, double
3.000000e+00, double 4.000000e+00, double 5.000000e+00, double 6.000000e+00, double
7.000000e+00, double 8.000000e+00, double 9.000000e+00], [9 x double]*
%matrix_array
  %get_array_ptr = getelementptr [9 x double], [9 x double]* %matrix_array, i32 0,
i32 0
  %init_matrix = call %Matrix* @buildMatrix(i32 3, i32 3, double* %get_array_ptr)
  store %Matrix* %init_matrix, %Matrix** %m
  %m1 = load %Matrix*, %Matrix** %m
  %print_matrix = call i32 @printMatrix(%Matrix* %m1)
  ret i32 0
}

declare %Matrix* @buildMatrix(i32, i32, double*)
```

The matrix data type only accepts integer or double, hence strings are not accepted.
Below is a program that throws an error after trying to initialize a matrix with invalid types.

```
int main(){
   matrix m;
// m consists of str, double
   m = ["s", 3.3;
        2.3, 3.4];
      return 0;
}
```

This will produce the following error message

```
Fatal error: exception Failure("matrix elements can only be
int/double type")
```

7.1.3 Structs

The following is a simple unit test program, which tests declaring, initializing, and accessing structs in PANAMX.

```
struct Node {
    matrix x;
    matrix y;
};

int main() {
    struct Node p;
    p = <struct Node>;
    p.x = [1,2;3,4];
    p.y = [5,6;7,8];
    printm(p.x);
    printm(p.y);
    return 0;
}
```

Below is the output of this program.

```
[
      1.000   2.000
      3.000   4.000
]
[
      5.000   6.000
      7.000   8.000
]
```

Below is the generated LLVM code for this program.

```
; ModuleID = 'Panamx'
source_filename = "Panamx"

%Matrix = type opaque
%Node = type { %Matrix*, %Matrix* }
```

```
@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00", align 1
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00", align 1
@fmt.2 = private unnamed_addr constant [4 x i8] c"%s\0A\00", align 1

declare i32 @printf(i8*, ...)

declare %Matrix* @buildMatrixEmpty(i32, i32)

declare double @matrixAccess(%Matrix*, i32, i32)

declare double @matrixAssign(%Matrix*, i32, i32, double)

declare %Matrix* @matrixSlice(%Matrix*, i32, i32, i32, i32)

declare i32 @printMatrix(%Matrix*)

declare i32 @freeMatrix(%Matrix*)

declare i32 @getHeight(%Matrix*)

declare i32 @getWidth(%Matrix*)

declare %Matrix* @addMatrixDouble(%Matrix*, double, i32)

declare %Matrix* @addMatrixMatrix(%Matrix*, %Matrix*, i32)

declare %Matrix* @subDoubleMatrix(double, %Matrix*)

declare %Matrix* @mulMatrixDouble(%Matrix*, double, i32)

declare %Matrix* @mulMatrixMatrix(%Matrix*, %Matrix*)

declare %Matrix* @mulElementWiseMatrix(%Matrix*, %Matrix*, i32)

declare double @sum(%Matrix*)

declare double @mean(%Matrix*)

declare %Matrix* @trans(%Matrix*)

declare %Matrix* @rref(%Matrix*)

declare double @rank(%Matrix*)

declare double @det(%Matrix*)

declare %Matrix* @inv(%Matrix*)
```

```
declare %Matrix* @concatTB(%Matrix*, %Matrix*)

declare %Matrix* @concatLR(%Matrix*, %Matrix*)

declare i32 @sqrti(i32)

declare double @sqrtd(double)

declare i32 @nrooti(i32, i32)

declare i32 @nrootd(i32, double)

declare i32 @absi(i32)

declare double @absd(double)

declare i32 @poweri(i32, i32)

declare double @powerd(double, i32)

define i32 @main() {
entry:
  %p = alloca %Node*
  %malloccall = tail call i8* @malloc(i32 trunc (i64 mul nuw (i64 ptrtoint (i1**
getelementptr (i1*, i1** null, i32 1) to i64), i64 2) to i32))
  %Node_body = bitcast i8* %malloccall to %Node*
  store %Node* %Node_body, %Node** %p
  %matrix_array = alloca [4 x double]
  store [4 x double] [double 1.000000e+00, double 2.000000e+00, double
3.000000e+00, double 4.000000e+00], [4 x double]* %matrix_array
  %get_array_ptr = getelementptr [4 x double], [4 x double]* %matrix_array, i32 0,
i32 0
  %init_matrix = call %Matrix* @buildMatrix(i32 2, i32 2, double* %get_array_ptr)
  %p1 = load %Node*, %Node** %p
  %Nodex = getelementptr inbounds %Node, %Node* %p1, i32 0, i32 0
  store %Matrix* %init_matrix, %Matrix** %Nodex
  %matrix_array2 = alloca [4 x double]
  store [4 x double] [double 5.000000e+00, double 6.000000e+00, double
7.000000e+00, double 8.000000e+00], [4 x double]* %matrix_array2
  %get_array_ptr3 = getelementptr [4 x double], [4 x double]* %matrix_array2, i32
0, i32 0
  %init_matrix4 = call %Matrix* @buildMatrix(i32 2, i32 2, double* %get_array_ptr3)
  %p5 = load %Node*, %Node** %p
  %Nodey = getelementptr inbounds %Node, %Node* %p5, i32 0, i32 1
  store %Matrix* %init_matrix4, %Matrix** %Nodey
  %p6 = load %Node*, %Node** %p
```

```
  %Nodex7 = getelementptr inbounds %Node, %Node* %p6, i32 0, i32 0
  %px = load %Matrix*, %Matrix** %Nodex7
  %print_matrix = call i32 @printMatrix(%Matrix* %px)
  %p8 = load %Node*, %Node** %p
  %Nodey9 = getelementptr inbounds %Node, %Node* %p8, i32 0, i32 1
  %py = load %Matrix*, %Matrix** %Nodey9
  %print_matrix10 = call i32 @printMatrix(%Matrix* %py)
  ret i32 0
}

declare noalias i8* @malloc(i32)

declare %Matrix* @buildMatrix(i32, i32, double*)
```

Below is a program that throws an error after assigning an invalid value to a member type.

```
struct Node {
    int x;
};

int main() {
    struct Node p;
    p = <struct Node>;
    p.x = 2.0;
    return 0;
}
```

This will produce the following error

```
Fatal error: exception Failure("illegal assignment int = float in p.x
= 2.0")
```

7.2 Test Suites

The test suite is located in the /tests directory. We have three different file extensions for test files, .px for PANAMX programs, .out for expected output, and .err for expected errors.

7.3 Script Automation

We modified the testall.sh file from MicroC to incorporate our new test directories, which showed us everything we needed to complete the testing process.

7.4 Process

To thoroughly test the features we implemented into our compiler, we made several test cases. Everytime we implemented a new feature, we would add a new test case. We also wrote many test cases to test our semantic checks.
We separted the test cases in to four directories. The directory tests/demos contains demos test cases used in our presentation. The directory tests/mc_tests contains relevant test cases from MicroC that still applied to our language. The directory tests/px_fails contains fail cases we wrote for our semantic checker. The directory tests/px_tests contains PANAMX speicifc test cases, such as anything to do with matrices and structs.
To test efficiently, we wrote have a test script, testall.sh that automatically goes through all the test directories in the tests folder and runs the test suite. Run ./testall.sh in the terminal to execute the testing script. You can also use the “make test” command to compile and run ./testall.sh with one command.

8 Lessons Learned

8.1 James

I think the most important I learned from this project is how to design a compiler, and the many layers that needs to be configured. For instance, to implement a feature, we have to go through each file from scanner to codegen, and make sure each component is modified properly. Working on various components of compiler has made me appreciate how much time and effort goes into developing new programming languages. Especially in the semantic checker, where each edge case has to be considered, throwing errors, and how functional programming can fill in the gaps. Furthermore, it was interesting to have to think about the trade-offs between functionality and usability of programming language from a user’s standpoint. Furthermore, since our team members didn’t know each other in the beginning, we really had to learn to work together and get along in a short amount of time. Many times as the manager, I found myself having to make fast decisions to meet project deadlines and make sure everybody gets their opinions across. These moments have taught me the importance of communication, and delegating short term tasks as to not overwhelm people. I think that this project experience and the class material has expanded my knowledge in computer science that will help me tackle complex problems.

8.2 Zach

Personally, I feel like this project taught me the most about communication. Effective group work requires fluid and transparent dialogue between the team, especially when the project is as long and complicated as implementing a programming language across multiple files in OCaml and C. First, I approach asking "stupid" questions differently than prior to this exercise. I learned that asking a "stupid" question early is highly beneficial to the team because it saves everyone time and moves the project forward more quickly. It might be that you don't want to waste anyone's time asking them the question or that you don't want to look unprepared or unintelligent, but taking five to ten minutes to ask and hear the answer to a "stupid" question could save you hours, and you'll want those hours back when it takes a really long time to implement complicated features later. I think I experienced the most when I was trying to install *LLVM* on my Windows machine and wasted hours trying to do so on my own when, ultimately, Brian had a solution to the issue the whole time. Furthermore, because of the size of the project, I think we all learned a lot about the importance of communicating what we were working on despite a good version control system. This helped us make sure that work was not repeated. Finally, I also learned that it's important to have more than one way to reach someone. For example, almost all of our offline communications we used a single messaging application. I ran into trouble once when I was really late to a group meeting because I had my notifications settings incorrectly set up and, therefore, wasn't getting notifications of group communications. The problem could have been mitigated if the team had another way to contact me when they realized that I wasn't responding.

Beyond communication improvements, I learned a lot about testing and good practices for programming in a group environment. For example, I learned very quickly about the need to test code in tiny bits and pieces as opposed to trying to debug it in huge pieces. In theory, I definitely knew this before, but it became overwhelmingly clear in a project that spanned so many files. Furthermore, I think the whole group learned that we tended to work better when we were all together in the same place rather than trying to work remotely. It definitely helped to facilitate question asking, so we were able to be more productive and save time by helping each other out rather than struggling on our own. The peer pressure also probably helped keep people focused, too. Finally, I realized that it's more effective to get started right away rather than spending inordinate amounts of time up front to try and make yourself feel totally comfortable. After getting started coding, even if I didn't know exactly what I was doing, more specific, appropriate questions would arise. Therefore, I learned not to waste a lot of time up front trying to "learn everything" to feel comfortable.

8.3 Xingchen

This project gives me a close look on how a modern compiler works, and it shows me the connection from the lecture content to real practice. As a summary, I learned 3 things in total. One is the knowledge of how compiler works and how they link to the real compiler component. Only by doing a project that can deeply impress myself the knowledge learned from class, from scanner to code generation, from the front-end to back-end. Another thing is the ability to design and implement a compiler, from parsing syntax to the LLVM IR code. The third thing is the experience to work with a group, and this experience will certainly benefit my future works.

8.4 Brian

This is the first group project I have ever done in computer science, and at this point, I think I learned how computer language is really designed and how it interacts with systems, but more importantly, I learned what a group project is and how to deal with difficult issues like time scheduling, communication, and computer environments. In the beginning of the semester, I feel like we did not do our best to do group assignments, though we did all of them in time. It might have been probably better, if we had tried to keep ahead of time, since the real implementation of components like semant and codegen were much harder than what we expected. Also, when it comes to computer environments, Zach used virtual machine on Windows system, both Jame and Xingchen used mac OS, and I installed Ubuntu 18.04 on external HDD. At first, it did not seem to be a thing to too much worry about, but there have been some serious issues that were very hard to handle. As Zack said in the meeting, I agree on that it might have been a bit better if we used some shared environment like docker io. Academically, it was a good practice that we implemented what we vaguely learned in 3000 level courses like Computer Science Theory and Fundamentals of Computer Systems.

9 Appendix

9.1 Source code

9.1.1 scanner.mll

```
(* Ocamllex scanner for PANAMX *)

{ open Parser }

let digit = ['0' - '9']
let digits = digit+

rule token = parse
  [' ' '\t' '\r' '\n'] { token lexbuf } (* Whitespace *)
| "/*"     { comment lexbuf }           (* Comments *)
| "//"     { line_comment lexbuf}       (* One-line comment *)
| '('      { LPAREN }
| ')'      { RPAREN }
| '{'      { LBRACE }
| '}'      { RBRACE }
| '['      { LBRACKET }
| ']'      { RBRACKET }
| ';'      { SEMI }
| ':'      { COLON }
| ','      { COMMA }
| '.'      { DOT }
| '+'      { PLUS }
| '-'      { MINUS }
| '*'      { TIMES }
| '/'      { DIVIDE }
| '%'      { MODULO }
| ".*"     { MMUL }
```

```
| "./"     { MDIV }
| "++"     { INCREMENT }
| "--"     { DECREMENT }
| '='      { ASSIGN }
| "=="     { EQ }
| "!="     { NEQ }
| '<'      { LT }
| "<="     { LEQ }
| '>'      { GT }
| ">="     { GEQ }
| "&&"     { AND }
| "||"     { OR }
| "!"      { NOT }
| "if"     { IF }
| "else"   { ELSE }
| "for"    { FOR }
| "while"  { WHILE }
| "return" { RETURN }
| "int"    { INT }
| "bool"   { BOOL }
| "string" { STRING }
| "double" { FLOAT }
| "void"   { VOID }
| "true"   { BLIT(true)  }
| "false"  { BLIT(false) }
| "matrix" { MATRIX }
| "struct" { STRUCT }
| ".height" { HEIGHT }
| ".width"  { WIDTH }
| digits as lxm { LITERAL(int_of_string lxm) }
| digits '.'  digit* ( ['e' 'E'] ['+' '-']? digits )? as lxm {
FLIT(lxm) }
| ['a'-'z' 'A'-'Z']['a'-'z' 'A'-'Z' '0'-'9' '_']*     as lxm {
ID(lxm) }
| '"' ([^ '"']* as lxm) '"'  { STRLIT(lxm) }
| eof { EOF }
| _ as char { raise (Failure("illegal character " ^ Char.escaped
char)) }
```

```
and comment = parse
  "*/" { token lexbuf }
| _  { comment lexbuf }

and line_comment = parse
  '\n' { token lexbuf }
| _  { line_comment lexbuf }
```

9.1.2 parser.mly

```
/* Ocamlyacc parser for PANAMX */

%{
open Ast
%}


%token SEMI COLON LPAREN RPAREN LBRACE RBRACE COMMA LBRACKET RBRACKET
%token PLUS MINUS TIMES DIVIDE MODULO ASSIGN INCREMENT DECREMENT
%token NOT EQ NEQ LT LEQ GT GEQ AND OR DOT STRUCT
%token HEIGHT WIDTH MMUL MDIV
%token RETURN IF ELSE FOR WHILE INT BOOL STRING FLOAT VOID MATRIX
%token <int> LITERAL
%token <bool> BLIT
%token <string> STRLIT ID FLIT
%token EOF

%start program
%type <Ast.program> program

%nonassoc NOELSE
%nonassoc ELSE
%right ASSIGN
%left OR
%left AND
```

```
%left EQ NEQ
%left LT GT LEQ GEQ
%left PLUS MINUS
%left TIMES DIVIDE MODULO MMUL MDIV
%left INCREMENT DECREMENT
%right NOT

%%

program:
  decls EOF { $1 }

decls:
   /* nothing */ { ([], [], [])                          }
 | decls vdecl { let (a, b, c) = $1 in (($2 :: a), b, c) }
 | decls fdecl { let (a, b, c) = $1 in (a, ($2 :: b), c) }
 | decls sdecl { let (a, b, c) = $1 in (a, b, ($2 :: c)) }

fdecl:
   typ ID LPAREN formals_opt RPAREN LBRACE vdecl_list stmt_list
RBRACE
     { { typ = $1;
     fname = $2;
     formals = List.rev $4;
     locals = List.rev $7;
     body = List.rev $8 } }

formals_opt:
     /* nothing */ { [] }
  | formal_list   { $1 }

formal_list:
     typ ID                   { [($1,$2)]     }
  | formal_list COMMA typ ID { ($3,$4) :: $1 }

typ:
     INT        { Int   }
  | BOOL   { Bool  }
```

```
  | FLOAT  { Float }
  | STRING { String }
  | VOID   { Void }
  | MATRIX { Matrix }
  | STRUCT ID { Struct($2) }

sdecl:
     STRUCT ID LBRACE sbody_list RBRACE SEMI { {
     sname = $2;
     svar = $4 } }

sbody_list:
     vdecl            { [$1] }
  | vdecl sbody_list { $1 :: $2 }

vdecl_list:
     /* nothing */    { [] }
  | vdecl_list vdecl { $2 :: $1 }

vdecl:
   typ ID SEMI { ($1, $2) }

stmt_list:
     /* nothing */  { [] }
  | stmt_list stmt { $2 :: $1 }

stmt:
     expr SEMI                               { Expr $1             }
  | RETURN expr_opt SEMI                     { Return $2           }
  | LBRACE stmt_list RBRACE                  { Block(List.rev $2)  }
  | IF LPAREN expr RPAREN stmt %prec NOELSE { 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)       }

expr_opt:
     /* nothing */ { Noexpr }
```

```
  | expr         { $1 }

expr:
     LITERAL          { Literal($1)         }
  | FLIT              { Fliteral($1)        }
  | BLIT              { BoolLit($1)         }
  | STRLIT            { StrLit($1)          }
  | ID                { Id($1)              }
  | expr PLUS   expr { Binop($1, Add,   $3)   }
  | expr MINUS  expr { Binop($1, Sub,   $3)   }
  | expr TIMES  expr { Binop($1, Mult,  $3)   }
  | expr DIVIDE expr { Binop($1, Div,   $3)   }
  | expr MODULO expr { Binop($1, Mod,   $3)   }
  | expr EQ     expr { Binop($1, Equal, $3)   }
  | expr NEQ    expr { Binop($1, Neq,   $3)   }
  | expr LT     expr { Binop($1, Less,  $3)   }
  | expr LEQ    expr { Binop($1, Leq,   $3)   }
  | expr GT     expr { Binop($1, Greater, $3) }
  | expr GEQ    expr { Binop($1, Geq,   $3)   }
  | expr AND    expr { Binop($1, And,   $3)   }
  | expr OR     expr { Binop($1, Or,  $3)   }
  | expr MMUL   expr { Binop($1, Mmul,  $3)   }
  | expr MDIV   expr { Binop($1, Mdiv,  $3)   }
  | MINUS expr %prec NOT { Unop(Neg, $2)    }
  | NOT expr          { Unop(Not, $2)       }
  | ID ASSIGN expr   { Assign($1, $3)       }
  | expr INCREMENT   { Unop(Inc, $1)        }
  | expr DECREMENT   { Unop(Dec, $1)        }
  | ID LPAREN args_opt RPAREN { Call($1, $3)  }
  | LPAREN expr RPAREN { $2                 }
  | LBRACKET matrixlit RBRACKET  { MatLit($2) }
  | LT expr COMMA expr GT { MatLitEmpty($2, $4) }
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET { MatIndex($1,
$3, $6) }
  | ID LBRACKET expr RBRACKET LBRACKET expr RBRACKET ASSIGN expr {
MatAssign($1, $3, $6, $9) }
  | ID LBRACKET expr COLON expr RBRACKET LBRACKET expr COLON expr
RBRACKET { MatSlice($1, $3, $5, $8, $10) }
```

```
  | ID HEIGHT          { Call("matrixHeight", [Id($1)]) }
  | ID WIDTH           { Call("matrixWidth",  [Id($1)]) }
  | LT STRUCT ID GT  { StructLit($3)        }
  | ID DOT ID          { Member($1, $3)      }
  | ID DOT ID ASSIGN expr { MemAssign($1, $3, $5) }

matrixlit:
      arraylit              { [$1] }
  | arraylit SEMI matrixlit { $1 :: $3 }

arraylit:
      expr            { [$1] }
  | expr COMMA arraylit { $1 :: $3 }

args_opt:
      /* nothing */ { [] }
  | args_list  { List.rev $1 }

args_list:
      expr                  { [$1] }
  | args_list COMMA expr { $3 :: $1 }
```

9.1.3 ast.ml

```
(* Abstract Syntax Tree and functions for printing it *)

type op = Add | Sub | Mult | Div | Mod | Equal | Neq | Less | Leq |
Greater | Geq |
           And | Or  | Mmul | Mdiv

type uop = Neg | Not | Inc | Dec

type typ = Int | Bool | String | Float | Void | Matrix | Struct of
string
```

```
type bind = typ * string

type expr =
      Literal of int
  | Fliteral of string
  | BoolLit of bool
  | StrLit of string
  | Id of string
  | Binop of expr * op * expr
  | Unop of uop * expr
  | Assign of string * expr
  | Call of string * expr list
  | Noexpr
  | MatLit of expr list list
  | MatLitEmpty of expr * expr
  | MatIndex of string * expr * expr
  | MatAssign of string * expr * expr * expr
  | MatSlice of string * expr * expr * expr * expr
  | StructLit of string
  | Member of string * string
  | MemAssign of string * string * expr

type stmt =
      Block of stmt list
  | Expr of expr
  | Return of expr
  | If of expr * stmt * stmt
  | For of expr * expr * expr * stmt
  | While of expr * stmt

type func_decl = {
      typ : typ;
      fname : string;
      formals : bind list;
      locals : bind list;
      body : stmt list;
  }
```

```
type struct_decl = {
  sname : string;
  svar  : bind list;
}

type program = bind list * func_decl list * struct_decl list

(* Pretty-printing functions *)

let string_of_op = function
    Add -> "+"
  | Sub -> "-"
  | Mult -> "*"
  | Div -> "/"
  | Mod -> "%"
  | Equal -> "=="
  | Neq -> "!="
  | Less -> "<"
  | Leq -> "<="
  | Greater -> ">"
  | Geq -> ">="
  | And -> "&&"
  | Or -> "||"
  | Mmul -> ".*"
  | Mdiv -> "./"

let string_of_uop = function
    Neg -> "-"
  | Not -> "!"
  | Inc -> "++"
  | Dec -> "--"

let rec string_of_expr = function
    Literal(l) -> string_of_int l
  | Fliteral(l) -> l
  | BoolLit(true) -> "true"
  | BoolLit(false) -> "false"
  | StrLit(s) -> s
```

```
  | Id(s) -> s
  | Binop(e1, o, e2) ->
      string_of_expr e1 ^ " " ^ string_of_op o ^ " " ^ string_of_expr
e2
  | Unop(o, e) -> string_of_uop o ^ string_of_expr e
  | Assign(v, e) -> v ^ " = " ^ string_of_expr e
  | Call(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_expr el) ^ ")"
  | Noexpr -> ""
  | MatLit _ -> "matrix"
  | MatLitEmpty _ -> "matrix"
  | MatIndex (s, i, j) -> s ^ "[" ^ (string_of_expr i) ^ "][" ^
(string_of_expr j) ^ "]"
  | MatAssign (s, i, j, e) -> s ^ "[" ^ (string_of_expr i) ^ "][" ^
(string_of_expr j) ^ "]= " ^ (string_of_expr e)
  | MatSlice (s, i, j, k, l) -> s ^ "[" ^ (string_of_expr i) ^ ":" ^
(string_of_expr j) ^ "][" ^
      (string_of_expr k) ^ ":" ^ (string_of_expr l) ^ "]"
  | StructLit e -> "new struct " ^ e ^ " ()"
  | Member (s, e) -> s ^ "." ^ e
  | MemAssign (s, m, e) -> s ^ "." ^ m ^ " = " ^ (string_of_expr e)

let rec string_of_stmt = function
      Block(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_stmt stmts) ^
"}\n"
  | Expr(expr) -> string_of_expr expr ^ ";\n";
  | Return(expr) -> "return " ^ string_of_expr expr ^ ";\n";
  | If(e, s, Block([])) -> "if (" ^ string_of_expr e ^ ")\n" ^
string_of_stmt s
  | If(e, s1, s2) ->  "if (" ^ string_of_expr e ^ ")\n" ^
      string_of_stmt s1 ^ "else\n" ^ string_of_stmt s2
  | For(e1, e2, e3, s) ->
      "for (" ^ string_of_expr e1  ^ " ; " ^ string_of_expr e2 ^ " ;
" ^
      string_of_expr e3  ^ ") " ^ string_of_stmt s
  | While(e, s) -> "while (" ^ string_of_expr e ^ ") " ^
string_of_stmt s
```

```
let string_of_typ = function
      Int -> "int"
  | Bool -> "bool"
  | String -> "string"
  | Float -> "float"
  | Void -> "void"
  | Matrix -> "matrix"
  | Struct e -> "struct " ^ e

let string_of_vdecl (t, id) = string_of_typ t ^ " " ^ id ^ ";\n"

let string_of_fdecl fdecl =
  string_of_typ fdecl.typ ^ " " ^
  fdecl.fname ^ "(" ^ String.concat ", " (List.map snd fdecl.formals)
^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.locals) ^
  String.concat "" (List.map string_of_stmt fdecl.body) ^
  "}\n"

let string_of_program (vars, funcs, _) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_fdecl funcs)
```

9.1.4 sast.ml

```
(* Semantically-checked Abstract Syntax Tree and functions for
printing it *)

open Ast

type sexpr = typ * sx
and sx =
      SLiteral of int
  | SFliteral of string
  | SBoolLit of bool
```

```
  | SStrLit of string
  | SId of string
  | SBinop of sexpr * op * sexpr
  | SUnop of uop * sexpr
  | SAssign of string * sexpr
  | SCall of string * sexpr list
  | SNoexpr
  | SMatLit of int * int * sexpr list
  | SMatLitEmpty of sexpr * sexpr
  | SMatIndex of string * sexpr * sexpr
  | SMatAssign of string * sexpr * sexpr * sexpr
  | SMatSlice of string * sexpr * sexpr * sexpr * sexpr
  | SStructLit of string
  | SMember of string * string
  | SMemAssign of string * string * sexpr

type sstmt =
      SBlock of sstmt list
  | SExpr of sexpr
  | SReturn of sexpr
  | SIf of sexpr * sstmt * sstmt
  | SFor of sexpr * sexpr * sexpr * sstmt
  | SWhile of sexpr * sstmt

type sfunc_decl = {
      styp : typ;
      sfname : string;
      sformals : bind list;
      slocals : bind list;
      sbody : sstmt list;
  }

type sstruct_decl = {
  ssname : string;
  ssvar : bind list;
}

type sprogram = bind list * sfunc_decl list * sstruct_decl list
```

```
(* Pretty-printing functions *)

let rec string_of_sexpr (t, e) =
  "(" ^ string_of_typ t ^ " : " ^ (match e with
      SLiteral(l) -> string_of_int l
    | SBoolLit(true) -> "true"
    | SBoolLit(false) -> "false"
    | SStrLit(s) -> s
    | SFliteral(l) -> l
    | SId(s) -> s
    | SBinop(e1, o, e2) ->
      string_of_sexpr e1 ^ " " ^ string_of_op o ^ " " ^
string_of_sexpr e2
    | SUnop(o, e) -> string_of_uop o ^ string_of_sexpr e
    | SAssign(v, e) -> v ^ " = " ^ string_of_sexpr e
    | SCall(f, el) ->
      f ^ "(" ^ String.concat ", " (List.map string_of_sexpr el) ^
")"
    | SNoexpr -> ""
    | SMatLit _ -> "matrix"
    | SMatLitEmpty _ -> "matrix"
    | SMatIndex (s, i, j) -> s ^ "[" ^ (string_of_sexpr i) ^ "][" ^
(string_of_sexpr j) ^ "]"
    | SMatAssign (s, i, j, e) -> s ^ "[" ^ (string_of_sexpr i) ^ "][" ^
(string_of_sexpr j) ^ "]= " ^ (string_of_sexpr e)
    | SMatSlice (s, i, j, k, l) -> s ^ "[" ^ (string_of_sexpr i) ^ ":"
^ (string_of_sexpr j) ^ "][" ^
      (string_of_sexpr k) ^ ":" ^ (string_of_sexpr l) ^ "]"
    | SStructLit e -> "new struct " ^ e ^ " ()"
    | SMember (s, e) -> s ^ "." ^ e
    | SMemAssign (s, m, e) -> s ^ "." ^ m ^ " = " ^ (string_of_sexpr e)
       ) ^ ")"

let rec string_of_sstmt = function
      SBlock(stmts) ->
      "{\n" ^ String.concat "" (List.map string_of_sstmt stmts) ^
"}\n"
```

```
  | SExpr(expr) -> string_of_sexpr expr ^ ";\n";
  | SReturn(expr) -> "return " ^ string_of_sexpr expr ^ ";\n";
  | SIf(e, s, SBlock([])) ->
      "if (" ^ string_of_sexpr e ^ ")\n" ^ string_of_sstmt s
  | SIf(e, s1, s2) ->  "if (" ^ string_of_sexpr e ^ ")\n" ^
      string_of_sstmt s1 ^ "else\n" ^ string_of_sstmt s2
  | SFor(e1, e2, e3, s) ->
      "for (" ^ string_of_sexpr e1  ^ " ; " ^ string_of_sexpr e2 ^ "
; " ^
      string_of_sexpr e3  ^ ") " ^ string_of_sstmt s
  | SWhile(e, s) -> "while (" ^ string_of_sexpr e ^ ") " ^
string_of_sstmt s

let string_of_sfdecl fdecl =
  string_of_typ fdecl.styp ^ " " ^
  fdecl.sfname ^ "(" ^ String.concat ", " (List.map snd
fdecl.sformals) ^
  ")\n{\n" ^
  String.concat "" (List.map string_of_vdecl fdecl.slocals) ^
  String.concat "" (List.map string_of_sstmt fdecl.sbody) ^
  "}\n"

let string_of_sprogram (vars, funcs, _) =
  String.concat "" (List.map string_of_vdecl vars) ^ "\n" ^
  String.concat "\n" (List.map string_of_sfdecl funcs)
```

9.1.5 semant.ml

```
(* Semantic checking for the PANAMX compiler *)

open Ast
open Sast

module StringMap = Map.Make(String)
```

```
(* Semantic checking of the AST. Returns an SAST if successful,
   throws an exception if something is wrong.

   Check each global variable, then check each function *)

let check (globals, functions, structs) =

  (* Verify a list of bindings has no void types or duplicate names
*)
  let check_binds (kind : string) (binds : bind list) =
      List.iter (function
            (Void, b) -> raise (Failure ("illegal void " ^ kind ^ " "
^ b))
      | _ -> ()) binds;
      let rec dups = function
      [] -> ()
      |    ((_,n1) :: (_,n2) :: _) when n1 = n2 ->
            raise (Failure ("duplicate " ^ kind ^ " " ^ n1))
      | _ :: t -> dups t
      in dups (List.sort (fun (_,a) (_,b) -> compare a b) binds)
  in

  (**** Check global variables ****)

  check_binds "global" globals;

  (**** Check functions ****)

  (* Collect function declarations for built-in functions: no bodies
*)
  let built_in_decls =
      let add_bind map (name, arg_list, return_ty) = StringMap.add
name {
      typ = return_ty;
      fname = name;
      formals = arg_list;
      locals = []; body = [] } map
      in List.fold_left add_bind StringMap.empty
```

```
      [ ("print",  [(Int, "x")], Void);
      ("printb", [(Bool, "x")], Void);
      ("prints", [(String, "x")], Void);
      ("printf", [(Float, "x")], Void);
      ("printm", [(Matrix, "x")], Void);
      ("free",   [(Matrix, "x")], Void);
      ("matrixHeight", [(Matrix, "x")], Int);
      ("matrixWidth",  [(Matrix, "x")], Int);
      ("sum",    [(Matrix, "x")], Float);
      ("mean",   [(Matrix, "x")], Float);
      ("trans",  [(Matrix, "x")], Matrix);
      ("rref",   [(Matrix, "x")], Matrix);
      ("rank",   [(Matrix, "x")], Float);
      ("det",    [(Matrix, "x")], Float);
      ("inv",    [(Matrix, "x")], Matrix);
      ("concatTB", [(Matrix, "x"); (Matrix, "y")], Matrix);
      ("concatLR", [(Matrix, "x"); (Matrix, "y")], Matrix);
      ("sqrti",  [(Int, "x")], Int);
      ("sqrtd",  [(Float, "x")], Float);
      ("nrooti",  [(Int, "x"); (Int, "y")], Int);
      ("nrootd",  [(Int, "x"); (Float, "y")], Float);
      ("absi",  [(Int, "x")], Int);
      ("absd",  [(Float, "x")], Float);
      ("poweri",  [(Int, "x"); (Int, "y")], Int);
      ("powerd",  [(Float, "x"); (Int, "y")], Float)]
  in

  (* Add function name to symbol table *)
  let add_func map fd =
     let built_in_err = "function " ^ fd.fname ^ " may not be
defined"
     and dup_err = "duplicate function " ^ fd.fname
     and make_err er = raise (Failure er)
     and n = fd.fname (* Name of the function *)
     in match fd with (* No duplicate functions or redefinitions of
built-ins *)
     _ when StringMap.mem n built_in_decls -> make_err built_in_err
     | _ when StringMap.mem n map -> make_err dup_err
```

```
      | _ ->  StringMap.add n fd map
  in

  (* Collect all function names into one symbol table *)
  let function_decls = List.fold_left add_func built_in_decls
functions
  in

  (* Return a function from our symbol table *)
  let find_func s =
     try StringMap.find s function_decls
     with Not_found -> raise (Failure ("unrecognized function " ^
s))
  in

  let _ = find_func "main" in (* Ensure "main" is defined *)

  let check_function func =
     (* Make sure no formals or locals are void or duplicates *)
     check_binds "formal" func.formals;
     check_binds "local" func.locals;

     (* Raise an exception if the given rvalue type cannot be
assigned to
     the given lvalue type *)
     let check_assign lvaluet rvaluet err =
     if lvaluet = rvaluet then lvaluet else raise (Failure err)
     in

     (* Build local symbol table of variables for this function *)
     let symbols = List.fold_left (fun m (ty, name) -> StringMap.add
name ty m)
                 StringMap.empty (globals @ func.formals @
func.locals )
     in

     (* Return a variable from our local symbol table *)
     let type_of_identifier s =
```

```
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^
s))
      in

      let get_struct_name (s : string) = match (type_of_identifier s)
with
                              Struct n -> n
                              | _ -> raise (Failure ("Invalid access(.)
operation for " ^ s))
      in

      let find_struct (id : string) = (
      let rec find_svar = function
            [] -> raise (Failure ("Cannot find struct " ^ id))
            | sdecl :: _ when sdecl.sname = id -> sdecl.svar
            | _ :: tl -> find_svar tl
      in find_svar structs)
      in

      let get_smember_type (s : string) (e : string) = (
      let rec get_member_type = function
            [] -> raise (Failure ("Struct " ^ s ^ " does not have
member " ^ e))
            | (ty, name) :: _ when name = e -> ty
            | _ :: tl -> get_member_type tl
      in get_member_type (find_struct s))
      in

      (* Return a semantically-checked expression, i.e., with a type
*)
      let rec expr = function
      Literal  l -> (Int, SLiteral l)
      | Fliteral l -> (Float, SFliteral l)
      | BoolLit l  -> (Bool, SBoolLit l)
      | StrLit l -> (String, SStrLit l)
      | Noexpr   -> (Void, SNoexpr)
      | Id s     -> (type_of_identifier s, SId s)
```

```
      | Assign(var, e) as ex ->
           let lt = type_of_identifier var
           and (rt, e') = expr e in
           let err = "illegal assignment " ^ string_of_typ lt ^ " = "
^
           string_of_typ rt ^ " in " ^ string_of_expr ex
           in (check_assign lt rt err, SAssign(var, (rt, e')))
      | Unop(op, e) as ex ->
           let (t, e') = expr e in
           let ty = match op with
           Neg when t = Int || t = Float || t = String -> t
           | Not when t = Bool -> Bool
           | Inc when t = Int || t = Float -> t
           | Dec when t = Int || t = Float -> t
           | _ -> raise (Failure ("illegal unary operator " ^
                                  string_of_uop op ^ string_of_typ t ^
                                  " in " ^ string_of_expr ex))
           in (ty, SUnop(op, (t, e')))
      | Binop(e1, op, e2) as e ->
           let (t1, e1') = expr e1
           and (t2, e2') = expr e2 in
           (* All binary operators require operands of the same type
*)
           let same = t1 = t2 in
           let is_float = t1 != Matrix && t2 != Matrix && (t1 = Float
|| t2 = Float) in
           (* Determine expression type based on operator and operand
types *)
           let ty = match op with
           Add | Sub | Mult | Div | Mod when same && t1 = Int -> Int
           | Add | Sub | Mult | Div | Mod when is_float -> Float
           | Add | Sub | Mult | Div when t1 = Matrix || t2 = Matrix
-> Matrix
           | Mmul | Mdiv when t1 = Matrix && t2 = Matrix -> Matrix
           | Equal | Neq when same -> Bool
           | Less | Leq | Greater | Geq when same && (t1 = Int || t1
= Float) -> Bool
           | And | Or when same && t1 = Bool -> Bool
```

```
          | _ -> raise (Failure ("illegal binary operator " ^
                      string_of_typ t1 ^ " " ^ string_of_op op ^ " "
^
                      string_of_typ t2 ^ " in " ^ string_of_expr e))
          in (ty, SBinop((t1, e1'), op, (t2, e2')))
     | Call(fname, args) as call ->
          let fd = find_func fname in
          let param_length = List.length fd.formals in
          if List.length args != param_length then
          raise (Failure ("expecting " ^ string_of_int param_length
^
                          " arguments in " ^ string_of_expr call))
          else let check_call (ft, _) e =
          let (et, e') = expr e in
          let err = "illegal argument found " ^ string_of_typ et ^
          " expected " ^ string_of_typ ft ^ " in " ^ string_of_expr
e
          in (check_assign ft et err, e')
          in
          let args' = List.map2 check_call fd.formals args
          in (fd.typ, SCall(fname, args'))

     | MatLit elts ->
     let rows = List.length elts in
     if rows = 0 then raise (Failure ("matrix height cannot be
zero")) else
     let cols = List.length (List.hd elts) in
     if cols = 0 then raise (Failure ("matrix width cannot be
zero")) else
     let selts = List.map expr (List.fold_left (fun x y -> x @ y) []
elts) in
     if List.fold_left (fun x y -> x && (fst y = Int || fst y =
Float)) true selts
     then (Matrix, SMatLit(rows, cols, selts))
     else raise (Failure ("matrix elements can only be int/double
type"))

     | MatLitEmpty (i, j) ->
```

```
      let (ti, ei) = expr i
      and (tj, ej) = expr j in
      if ti != Int then raise (Failure ("matrix height must be
integer")) else
      if tj != Int then raise (Failure ("matrix width must be
integer")) else
      (Matrix, SMatLitEmpty((ti, ei), (tj, ej)))

      | MatIndex(id, i, j) -> check_matrix_index id i j

      | MatAssign(id, i, j, e) -> let (_, smatindex) =
check_matrix_index id i j
      and (tr, er) = expr e in
      if tr != Int && tr != Float then raise (Failure ("matrix
element must be int/double"))
      else (match smatindex with
            SMatIndex(_, (ti, ei), (tj, ej)) -> (Float, SMatAssign(id,
(ti, ei), (tj, ej), (tr, er)))
            | _ -> raise (Failure ("should not happen - matrix")))

      | MatSlice(id, i, j, k, l) -> check_matrix_slice id i j k l

      | StructLit id -> ignore(find_struct id); (Struct(id),
SStructLit(id))

      | Member (s, m) ->
      let n = get_struct_name s in
      let ty = get_smember_type n m in (ty, SMember(s, m))

      | MemAssign (s, m, e) as ex ->
      let n = get_struct_name s in
      let lt = get_smember_type n m in
      let (rt, e') = expr e in
      let err = "illegal assignment " ^ string_of_typ lt ^ " = " ^
            string_of_typ rt ^ " in " ^ string_of_expr ex
      in (check_assign lt rt err, SMemAssign(s, m, (rt, e')))

      and check_matrix_index (id : string) (i : expr) (j : expr) =
```

```
      let (ti, ei) = expr i
      and (tj, ej) = expr j in
      if ti != Int || tj != Int then raise (Failure ("index must be
integer"))
      else (Float, SMatIndex(id, (ti, ei), (tj, ej)))

      and check_matrix_slice (id: string) (i: expr) (j: expr) (k:
expr) (l: expr) =
      let (ti, ei) = expr i
      and (tj, ej) = expr j
      and (tk, ek) = expr k
      and (tl, el) = expr l in
      if ti != Int || tj != Int || tk != Int || tl != Int then raise
(Failure ("index must be integer"))
      else (Matrix, SMatSlice(id, (ti, ei), (tj, ej), (tk, ek), (tl,
el)))

      in

      let check_bool_expr e =
      let (t', e') = expr e
      and err = "expected Boolean expression in " ^ string_of_expr e
      in if t' != Bool then raise (Failure err) else (t', e')
      in

      (* Return a semantically-checked statement i.e. containing
sexprs *)
      let rec check_stmt = function
      Expr e -> SExpr (expr e)
      | If(p, b1, b2) -> SIf(check_bool_expr p, check_stmt b1,
check_stmt b2)
      | For(e1, e2, e3, st) ->
            SFor(expr e1, check_bool_expr e2, expr e3, check_stmt st)
      | While(p, s) -> SWhile(check_bool_expr p, check_stmt s)
      | Return e -> let (t, e') = expr e in
      if t = func.typ then SReturn (t, e')
      else raise (Failure ("return gives " ^ string_of_typ t ^ "
expected " ^
```

```
          string_of_typ func.typ ^ " in " ^ string_of_expr e))

     (* A block is correct if each statement is correct and nothing
          follows any Return statement.  Nested blocks are
flattened. *)
     | Block sl ->
          let rec check_stmt_list = function
          [Return _ as s] -> [check_stmt s]
          | Return _ :: _   -> raise (Failure "nothing may follow a
return")
          | Block sl :: ss  -> check_stmt_list (sl @ ss) (* Flatten
blocks *)
          | s :: ss         -> check_stmt s :: check_stmt_list ss
          | []              -> []
          in SBlock(check_stmt_list sl)

     in (* body of check_function *)
     { styp = func.typ;
     sfname = func.fname;
     sformals = func.formals;
     slocals  = func.locals;
     sbody = match check_stmt (Block func.body) with
          SBlock(sl) -> sl
     | _ -> raise (Failure ("internal error: block didn't become a
block?"))
     }
  in

  (**** Check structs ****)
  let check_structs (st_list : struct_decl list) =
     (* check struct name *)
     let rec struct_dups = function
     [] -> ()
     | (s1 :: s2 :: _) when s1.sname = s2.sname ->
     raise (Failure ("duplicate struct name " ^ s1.sname))
     | _ :: tl -> struct_dups tl
     in struct_dups st_list;
```

```
      (* check struct body *)
      let struct_binds (st : struct_decl) =
      if List.length st.svar = 0 then raise (Failure ("Empty struct
body"))
      else check_binds "struct" st.svar
      in List.iter struct_binds st_list;

      let check_struct (st : struct_decl) = {
      ssname = st.sname;
      ssvar = st.svar;
      }
      in List.map check_struct st_list

  in (globals, List.map check_function functions, check_structs
structs)
```

9.1.6 codegen.ml

```
(* LLVM IR code generation for the PANAMX compiler *)

module L = Llvm
module A = Ast
open Sast

module StringMap = Map.Make(String)

(* translate : Sast.program -> Llvm.module *)
let translate (globals, functions, structs) =
  let context    = L.global_context () in

  (* Create the LLVM compilation module into which
     we will generate code *)
  let the_module = L.create_module context "Panamx" in

  (* Get types from the context *)
```

```
  let i32_t      = L.i32_type    context
  and i8_t       = L.i8_type     context
  and i1_t       = L.i1_type     context
  and float_t    = L.double_type context
  and void_t     = L.void_type   context
  and pointer_t  = L.pointer_type
  and array_t    = L.array_type
  and matrix_t   = L.pointer_type (L.named_struct_type context
"Matrix")
  in

  (* Return the LLVM type for a MicroC type *)
  let ltype_of_typ = function
      A.Int -> i32_t
    | A.Bool   -> i1_t
    | A.String -> pointer_t i8_t
    | A.Float  -> float_t
    | A.Void   -> void_t
    | A.Matrix -> matrix_t
    | A.Struct e -> pointer_t (L.named_struct_type context e)
  in

  (* Create a map of global variables after creating each *)
  let global_vars : L.llvalue StringMap.t =
      let global_var m (t, n) =
      let init = match t with
           A.Float -> L.const_float (ltype_of_typ t) 0.0
      | _ -> L.const_int (ltype_of_typ t) 0
      in StringMap.add n (L.define_global n init the_module) m in
      List.fold_left global_var StringMap.empty globals in

  let struct_decls : (L.lltype * sstruct_decl) StringMap.t =
      let define_struct m sdecl =
      let get_sbody_ty ((ty, _) : A.bind) = ltype_of_typ ty in
      let sname = sdecl.ssname
      and svar_type = Array.of_list (List.map get_sbody_ty
sdecl.ssvar) in
      let stype = L.named_struct_type context sname in
```

```
      L.struct_set_body stype svar_type false;
      StringMap.add sname (stype, sdecl) m in
      List.fold_left define_struct StringMap.empty structs
  in

  let printf_t : L.lltype =
      L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
      L.declare_function "printf" printf_t the_module in

  let build_matrix_empty_t : L.lltype =
      L.function_type matrix_t [| i32_t; i32_t |] in
  let build_matrix_empty_func : L.llvalue =
      L.declare_function "buildMatrixEmpty" build_matrix_empty_t
the_module in

  let matrix_access_t : L.lltype =
      L.function_type float_t [| matrix_t; i32_t; i32_t |] in
  let matrix_access_func : L.llvalue =
      L.declare_function "matrixAccess" matrix_access_t the_module in

  let matrix_assign_t : L.lltype =
      L.function_type float_t [| matrix_t; i32_t; i32_t; float_t |]
in
  let matrix_assign_funct : L.llvalue =
      L.declare_function "matrixAssign" matrix_assign_t the_module in

  let matrix_slice_t : L.lltype =
      L.function_type matrix_t [| matrix_t; i32_t; i32_t; i32_t;
i32_t |] in
  let matrix_slice_func : L.llvalue =
      L.declare_function "matrixSlice" matrix_slice_t the_module in

  let print_matrix_t : L.lltype =
      L.function_type i32_t [| matrix_t |] in
  let print_matrix_func : L.llvalue =
      L.declare_function "printMatrix" print_matrix_t the_module in
```

```
let free_matrix_t : L.lltype =
    L.function_type i32_t [| matrix_t |] in
let free_matrix_func : L.llvalue =
    L.declare_function "freeMatrix" free_matrix_t the_module in

let matrix_height_t : L.lltype =
    L.function_type i32_t [| matrix_t |] in
let matrix_height_func : L.llvalue =
    L.declare_function "getHeight" matrix_height_t the_module in

let matrix_width_t : L.lltype =
    L.function_type i32_t [| matrix_t |] in
let matrix_width_func : L.llvalue =
    L.declare_function "getWidth" matrix_width_t the_module in

let add_md_t : L.lltype =
    L.function_type matrix_t [| matrix_t; float_t; i32_t |] in
let add_md_func : L.llvalue =
    L.declare_function "addMatrixDouble" add_md_t the_module in

let add_mm_t : L.lltype =
    L.function_type matrix_t [| matrix_t; matrix_t; i32_t |] in
let add_mm_func : L.llvalue =
    L.declare_function "addMatrixMatrix" add_mm_t the_module in

let sub_dm_t : L.lltype =
    L.function_type matrix_t [| float_t; matrix_t |] in
let sub_dm_func : L.llvalue =
    L.declare_function "subDoubleMatrix" sub_dm_t the_module in

let mul_md_t : L.lltype =
    L.function_type matrix_t [| matrix_t; float_t; i32_t |] in
let mul_md_func : L.llvalue =
    L.declare_function "mulMatrixDouble" mul_md_t the_module in

let mul_mm_t : L.lltype =
    L.function_type matrix_t [| matrix_t; matrix_t |] in
let mul_mm_func : L.llvalue =
```

```
    L.declare_function "mulMatrixMatrix" mul_mm_t the_module in

  let mul_ew_mm_t : L.lltype =
      L.function_type matrix_t [| matrix_t; matrix_t; i32_t |] in
  let mul_ew_mm_func : L.llvalue =
      L.declare_function "mulElementWiseMatrix" mul_ew_mm_t
the_module in

  let matrix_sum_t : L.lltype =
      L.function_type float_t [| matrix_t |] in
  let matrix_sum_func : L.llvalue =
      L.declare_function "sum" matrix_sum_t the_module in

  let matrix_mean_t : L.lltype =
      L.function_type float_t [| matrix_t |] in
  let matrix_mean_func : L.llvalue =
      L.declare_function "mean" matrix_mean_t the_module in

  let matrix_trans_t : L.lltype =
      L.function_type matrix_t [| matrix_t |] in
  let matrix_trans_func : L.llvalue =
      L.declare_function "trans" matrix_trans_t the_module in

  let matrix_rref_t : L.lltype =
      L.function_type matrix_t [| matrix_t |] in
  let matrix_rref_func : L.llvalue =
      L.declare_function "rref" matrix_rref_t the_module in

  let matrix_rank_t : L.lltype =
      L.function_type float_t [| matrix_t |] in
  let matrix_rank_func : L.llvalue =
      L.declare_function "rank" matrix_rank_t the_module in

  let matrix_det_t : L.lltype =
      L.function_type float_t [| matrix_t |] in
  let matrix_det_func : L.llvalue =
      L.declare_function "det" matrix_det_t the_module in
```

```
let matrix_inv_t : L.lltype =
    L.function_type matrix_t [| matrix_t |] in
let matrix_inv_func : L.llvalue =
    L.declare_function "inv" matrix_inv_t the_module in

let matrix_concatTB_t : L.lltype =
    L.function_type matrix_t [| matrix_t; matrix_t |] in
let matrix_concatTB_func : L.llvalue =
    L.declare_function "concatTB" matrix_concatTB_t the_module in

let matrix_concatLR_t : L.lltype =
    L.function_type matrix_t [| matrix_t; matrix_t |] in
let matrix_concatLR_func : L.llvalue =
    L.declare_function "concatLR" matrix_concatLR_t the_module in

let int_sqrti_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let int_sqrti_func : L.llvalue =
    L.declare_function "sqrti" int_sqrti_t the_module in

let float_sqrtd_t : L.lltype =
    L.function_type float_t [| float_t |] in
let float_sqrtd_func : L.llvalue =
    L.declare_function "sqrtd" float_sqrtd_t the_module in

let nrooti_t : L.lltype =
    L.function_type i32_t [| i32_t ; i32_t |] in
let nrooti_func : L.llvalue =
    L.declare_function "nrooti" nrooti_t the_module in

let nrootd_t : L.lltype =
    L.function_type i32_t [| i32_t ; float_t |] in
let nrootd_func : L.llvalue =
    L.declare_function "nrootd" nrootd_t the_module in

let absi_t : L.lltype =
    L.function_type i32_t [| i32_t |] in
let absi_func : L.llvalue =
```

```
    L.declare_function "absi" absi_t the_module in

  let absd_t : L.lltype =
      L.function_type float_t [| float_t |] in
  let absd_func : L.llvalue =
      L.declare_function "absd" absd_t the_module in

  let poweri_t : L.lltype =
      L.function_type i32_t [| i32_t ; i32_t |] in
  let poweri_func : L.llvalue =
      L.declare_function "poweri" poweri_t the_module in

  let powerd_t : L.lltype =
      L.function_type float_t [| float_t ; i32_t|] in
  let powerd_func : L.llvalue =
      L.declare_function "powerd" powerd_t the_module in


  (* Define each function (arguments and return type) so we can
      call it even before we've created its body *)
  let function_decls : (L.llvalue * sfunc_decl) StringMap.t =
      let function_decl m fdecl =
      let name = fdecl.sfname
      and formal_types = Array.of_list (List.map (fun (t,_) ->
ltype_of_typ t) fdecl.sformals)
      in let ftype = L.function_type (ltype_of_typ fdecl.styp)
formal_types in
      StringMap.add name (L.define_function name ftype the_module,
fdecl) m in
      List.fold_left function_decl StringMap.empty functions in

  (* Fill in the body of the given function *)
  let build_function_body fdecl =
      let (the_function, _) = StringMap.find fdecl.sfname
function_decls in
      let builder = L.builder_at_end context (L.entry_block
the_function) in
```

```
      let int_format_str = L.build_global_stringptr "%d\n" "fmt"
builder
      and float_format_str = L.build_global_stringptr "%g\n" "fmt"
builder
      and string_format_str = L.build_global_stringptr "%s\n" "fmt"
builder in

      (* Construct the function's "locals": formal arguments and
locally
      declared variables.  Allocate each on the stack, initialize
their
      value, if appropriate, and remember their values in the
"locals" map *)
      let local_vars =
      let add_formal m (t, n) p =
      L.set_value_name n p;
      let local = L.build_alloca (ltype_of_typ t) n builder in
      ignore (L.build_store p local builder);
            StringMap.add n local m

      (* Allocate space for any locally declared variables and add
the
      * resulting registers to our map *)
      and add_local m (t, n) =
      let local_var = match t with
            A.Struct e ->
            let (sty, _) = StringMap.find e struct_decls
            in L.build_alloca (pointer_t sty) n builder
      | _ -> L.build_alloca (ltype_of_typ t) n builder
      in StringMap.add n local_var m
      in

      let formals = List.fold_left2 add_formal StringMap.empty
fdecl.sformals
            (Array.to_list (L.params the_function)) in
      List.fold_left add_local formals fdecl.slocals
      in
```

```
      let symbols = List.fold_left (fun m (ty, name) -> StringMap.add
name ty m)
                  StringMap.empty (fdecl.sformals @ fdecl.slocals)
      in

      let type_of_identifier s =
      try StringMap.find s symbols
      with Not_found -> raise (Failure ("undeclared identifier " ^
s))
      in

      (* Return the value for a variable or formal argument.
      Check local names first, then global names *)
      let lookup n = try StringMap.find n local_vars
                  with Not_found -> StringMap.find n global_vars
      in

      (* Construct code for an expression; return its value *)
      let rec expr builder ((_, e) : sexpr) = match e with
            SLiteral i  -> L.const_int i32_t i
      | SBoolLit b  -> L.const_int i1_t (if b then 1 else 0)
      | SStrLit s   -> L.build_global_stringptr s "tmp" builder
      | SFliteral l -> L.const_float_of_string float_t l
      | SNoexpr  -> L.const_int i32_t 0
      | SId s    -> L.build_load (lookup s) s builder

      | SMatLit (rows, cols, e) ->
      let build_matrix_t : L.lltype =
            L.function_type matrix_t [| i32_t; i32_t; pointer_t
float_t |] in
      let build_matrix_func : L.llvalue =
            L.declare_function "buildMatrix" build_matrix_t the_module
in
      let llarray = L.const_array float_t (Array.of_list (List.map
(int_to_float builder) e)) in
      let array_ptr_ty = array_t float_t (rows * cols) in
      let llptr = L.build_alloca array_ptr_ty "matrix_array" builder
in
```

```
      ignore(L.build_store llarray llptr builder);
      let ptr = L.build_gep llptr [| L.const_int i32_t 0; L.const_int
i32_t 0 |] "get_array_ptr" builder in
      L.build_call build_matrix_func
           [| L.const_int i32_t rows; L.const_int i32_t cols; ptr |]
"init_matrix" builder

      | SMatLitEmpty (i, j) ->
      let rows = expr builder i
      and cols = expr builder j in
      L.build_call build_matrix_empty_func [| rows; cols |]
"init_matrix_empty" builder

      | SMatIndex (s, i, j) ->
      let i' = expr builder i
      and j' = expr builder j
      and s' = L.build_load (lookup s) s builder in
      L.build_call matrix_access_func [| s'; i'; j' |]
"matrix_access" builder

      | SMatAssign (s, i, j, e) ->
      let i' = expr builder i
      and j' = expr builder j
      and e' = int_to_float builder e
      and s' = L.build_load (lookup s) s builder in
      L.build_call matrix_assign_funct [| s'; i'; j'; e' |]
"matrix_assign" builder

      | SMatSlice (s, i, j, k, l) ->
      let i' = expr builder i
      and j' = expr builder j
      and k' = expr builder k
      and l' = expr builder l
      and s' = L.build_load (lookup s) s builder in
      L.build_call matrix_slice_func [| s'; i'; j'; k'; l' |]
"matrix_slice" builder

      | SStructLit id ->
```

```
      let (sty, _) = StringMap.find id struct_decls
      in L.build_malloc sty (id ^ "_body") builder

      | SMember (s, m) -> let mem_p = get_smem_ptr builder s m in
      L.build_load mem_p (s ^ m) builder

      | SMemAssign (s, m, e) ->
      let e' = expr builder e in
      let mem_p = get_smem_ptr builder s m in
      ignore(L.build_store e' mem_p builder); e'

      | SAssign (s, e) -> let e' = expr builder e in
                            ignore(L.build_store e' (lookup s)
builder); e'
      | SBinop (((ty1, _) as e1), op, ((ty2, _) as e2)) when ty1 =
A.Int && ty2 = A.Int ->
      let e1' = expr builder e1
      and e2' = expr builder e2 in
            (match op with
            A.Add      -> L.build_add
            | A.Sub    -> L.build_sub
            | A.Mult   -> L.build_mul
            | A.Div    -> L.build_sdiv
            | A.Mod    -> L.build_srem
            | A.And    -> L.build_and
            | A.Or     -> L.build_or
            | A.Equal   -> L.build_icmp L.Icmp.Eq
            | A.Neq    -> L.build_icmp L.Icmp.Ne
            | A.Less   -> L.build_icmp L.Icmp.Slt
            | A.Leq    -> L.build_icmp L.Icmp.Sle
            | A.Greater -> L.build_icmp L.Icmp.Sgt
            | A.Geq    -> L.build_icmp L.Icmp.Sge
            | _        -> raise (Failure "should not happen")
            ) e1' e2' "tmp" builder
      | SBinop (((ty1, _) as e1), op, ((ty2, _) as e2)) when ty1 =
A.Bool && ty2 = A.Bool ->
      let e1' = expr builder e1
      and e2' = expr builder e2 in
```

```
          (match op with
          A.And   -> L.build_and
          | A.Or     -> L.build_or
          | A.Equal -> L.build_icmp L.Icmp.Eq
          | A.Neq   -> L.build_icmp L.Icmp.Ne
          | _        -> raise (Failure "invalid operation on bool")
          ) e1' e2' "tmp" builder
     | SBinop (((ty1, _) as e1), op, ((ty2, _) as e2)) when ty1 =
A.Matrix || ty2 = A.Matrix ->
     let e1' = int_to_float builder e1
     and e2' = int_to_float builder e2 in
          (match op with
          A.Add  ->
          if ty1 = ty2 && ty1 = A.Matrix
          then L.build_call add_mm_func [| e1'; e2'; L.const_int
i32_t 0 |] "add_matrix" builder
          else if ty1 = A.Matrix
          then L.build_call add_md_func [| e1'; e2'; L.const_int
i32_t 0 |] "add_matrix" builder
          else L.build_call add_md_func [| e2'; e1'; L.const_int
i32_t 0 |] "add_matrix" builder
          | A.Sub  ->
          if ty1 = ty2 && ty1 = A.Matrix
          then L.build_call add_mm_func [| e1'; e2'; L.const_int
i32_t 1 |] "sub_matrix" builder
          else if ty1 = A.Matrix
          then L.build_call add_md_func [| e1'; e2'; L.const_int
i32_t 1 |] "sub_matrix" builder
          else L.build_call sub_dm_func [| e1'; e2' |] "sub_matrix"
builder
          | A.Mult ->
          if ty1 = ty2 && ty1 = A.Matrix
          then L.build_call mul_mm_func [| e1'; e2' |] "mul_matrix"
builder
          else if ty1 = A.Matrix
          then L.build_call mul_md_func [| e1'; e2'; L.const_int
i32_t 0 |] "mul_matrix" builder
          else L.build_call mul_md_func [| e2'; e1'; L.const_int
```

```
i32_t 0 |] "mul_matrix" builder
           | A.Div ->
           if ty1 = A.Matrix && ty2 != A.Matrix
           then L.build_call mul_md_func [| e1'; e2'; L.const_int
i32_t 1 |] "div_matrix" builder
           else raise (Failure "illegal operation / on matrix")
           | A.Mmul when ty1 = ty2 && ty1 = A.Matrix ->
           L.build_call mul_ew_mm_func [| e1'; e2'; L.const_int i32_t
0 |] "elewise_matrix" builder
           | A.Mdiv when ty1 = ty2 && ty1 = A.Matrix ->
           L.build_call mul_ew_mm_func [| e1'; e2'; L.const_int i32_t
1 |] "elewise_matrix" builder
           | _   -> raise (Failure
           ("illegal operation " ^ (A.string_of_op op) ^ " on
matrix"))
           )
     | SBinop (e1, op, e2) ->
     let e1' = int_to_float builder e1
     and e2' = int_to_float builder e2 in
          (match op with
          A.Add      -> L.build_fadd
          | A.Sub    -> L.build_fsub
          | A.Mult   -> L.build_fmul
          | A.Div    -> L.build_fdiv
          | A.Mod    -> L.build_frem
          | A.Equal   -> L.build_fcmp L.Fcmp.Oeq
          | A.Neq    -> L.build_fcmp L.Fcmp.One
          | A.Less   -> L.build_fcmp L.Fcmp.Olt
          | A.Leq    -> L.build_fcmp L.Fcmp.Ole
          | A.Greater -> L.build_fcmp L.Fcmp.Ogt
          | A.Geq    -> L.build_fcmp L.Fcmp.Oge
          | A.And | A.Or ->
          raise (Failure "internal error: semant should have
rejected and/or on float")
          | _        -> raise (Failure "should not happen")
          ) e1' e2' "tmp" builder
     | SUnop(op, ((t, ex) as e)) ->
          let e' = expr builder e in
```

```
            (match op with
            A.Neg when t = A.Float -> L.build_fneg e' "tmp" builder
            | A.Neg               -> L.build_neg e' "tmp" builder
            | A.Not               -> L.build_not e' "tmp" builder
            | A.Inc               ->
            L.build_store (L.build_add e' (L.const_int i32_t 1) "tmp"
builder) (lookup (match ex with
            SId id -> id
            | _ -> raise (Failure "increment error"))) builder
            | A.Dec               ->
            L.build_store (L.build_sub e' (L.const_int i32_t 1) "tmp"
builder) (lookup (match ex with
            SId id -> id
            | _ -> raise (Failure "decrement error"))) builder
            )
      | SCall ("printb", [e]) ->
            L.build_call printf_func [| int_format_str ; (expr builder
e) |]
            "printf" builder
      | SCall ("printf", [e]) ->
      L.build_call printf_func [| float_format_str ; (expr builder e)
|]
            "printf" builder
      | SCall ("prints", [e]) ->
      L.build_call printf_func [| string_format_str; (expr builder e)
|]
            "printf" builder
      | SCall ("printm", [e]) ->
      L.build_call print_matrix_func [| expr builder e |]
"print_matrix" builder
      | SCall ("free", [e]) ->
      L.build_call free_matrix_func [| expr builder e |]
"free_matrix" builder
      (* L.build_free (expr builder e) builder *)
      | SCall ("matrixHeight", [e]) ->
      L.build_call matrix_height_func [| expr builder e |]
"matrix_height" builder
      | SCall ("matrixWidth", [e]) ->
```

```
      L.build_call matrix_width_func [| expr builder e |]
"matrix_width" builder

      | SCall ("sum", [e]) ->
      L.build_call matrix_sum_func [| expr builder e |] "matrix_sum"
builder
      | SCall ("mean", [e]) ->
      L.build_call matrix_mean_func [| expr builder e |]
"matrix_mean" builder
      | SCall ("trans", [e]) ->
      L.build_call matrix_trans_func [| expr builder e |]
"matrix_trans" builder
      | SCall ("rref", [e]) ->
      L.build_call matrix_rref_func [| expr builder e |]
"matrix_rref" builder
      | SCall ("rank", [e]) ->
      L.build_call matrix_rank_func [| expr builder e |]
"matrix_rank" builder
      | SCall ("det", [e]) ->
      L.build_call matrix_det_func [| expr builder e |] "matrix_det"
builder
      | SCall ("inv", [e]) ->
      L.build_call matrix_inv_func [| expr builder e |] "matrix_inv"
builder
      | SCall ("concatTB", [e1; e2]) ->
      L.build_call matrix_concatTB_func [| expr builder e1; expr
builder e2 |] "matrix_concatTB" builder
      | SCall ("concatLR", [e1; e2]) ->
      L.build_call matrix_concatLR_func [| expr builder e1; expr
builder e2 |] "matrix_concatLR" builder

      | SCall ("sqrti", [e]) ->
      L.build_call int_sqrti_func [| expr builder e |] "int_sqrti"
builder
      | SCall ("sqrtd", [e]) ->
      L.build_call float_sqrtd_func [| expr builder e |]
"float_sqrtd" builder
      | SCall ("nrooti", [e1; e2]) ->
```

```
      L.build_call nrooti_func [| expr builder e1; expr builder e2 |]
"nrooti" builder
      | SCall ("nrootd", [e1; e2]) ->
      L.build_call nrootd_func [| expr builder e1; expr builder e2 |]
"nrooti" builder
      | SCall ("absi", [e]) ->
      L.build_call absi_func [| expr builder e |] "absi" builder
      | SCall ("absd", [e]) ->
      L.build_call absd_func [| expr builder e|] "absd" builder
      | SCall ("poweri", [e1; e2]) ->
      L.build_call poweri_func [| expr builder e1; expr builder e2 |]
"poweri" builder
      | SCall ("powerd", [e1; e2]) ->
      L.build_call powerd_func [| expr builder e1; expr builder e2 |]
"powerd" builder

      | SCall (f, args) ->
      let (fdef, fdecl) = StringMap.find f function_decls in
            let llargs = List.rev (List.map (expr builder) (List.rev
args)) in
            let result = (match fdecl.styp with
                        A.Void -> ""
                        | _ -> f ^ "_result") in
      L.build_call fdef (Array.of_list llargs) result builder

      and int_to_float builder e =
            let e' = expr builder e in
            match fst e with
            A.Int -> L.build_sitofp e' float_t "tmp" builder
            | _   -> e'

      (* get the member m's pointer in struct *)
      and get_smem_ptr builder s m =
            let sname = match (type_of_identifier s) with
                  A.Struct n -> n
            | _ -> raise (Failure ("Invalid access(.) operation for "
^ s)) in
            let (_, sdecl) = StringMap.find sname struct_decls in
```

```
            let idx =
            let rec find_idx = function
                  [] -> raise (Failure ("Struct " ^ sname ^ " does not
have member " ^ m))
            | (_, var) :: _ when var = m -> 0
            | _ :: tl -> 1 + find_idx tl
            in find_idx sdecl.ssvar in
            let struct_p = L.build_load (lookup s) s builder in
            L.build_struct_gep struct_p idx (sname ^ m) builder

      in

      (* LLVM insists each basic block end with exactly one
"terminator"
      instruction that transfers control.  This function runs "instr
builder"
      if the current block does not already have a terminator.  Used,
      e.g., to handle the "fall off the end of the function" case. *)
      let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
            Some _ -> ()
      | None -> ignore (instr builder) in

      (* Build the code for the given statement; return the builder
for
      the statement's successor (i.e., the next instruction will be
built
      after the one generated by this call) *)

      let rec stmt builder = function
            SBlock sl -> List.fold_left stmt builder sl
      | SExpr e -> ignore(expr builder e); builder
      | SReturn e -> ignore(match fdecl.styp with
                              (* Special "return nothing" instr *)
                              A.Void -> L.build_ret_void builder
                              (* Build return statement *)
                              | _ -> L.build_ret (expr builder e)
builder );
```

```
                        builder
      | SIf (predicate, then_stmt, else_stmt) ->
      let bool_val = expr builder predicate in
           let merge_bb = L.append_block context "merge" the_function
in
      let build_br_merge = L.build_br merge_bb in (* partial function
*)

           let then_bb = L.append_block context "then" the_function
in
           add_terminal (stmt (L.builder_at_end context then_bb)
then_stmt)
           build_br_merge;

           let else_bb = L.append_block context "else" the_function
in
           add_terminal (stmt (L.builder_at_end context else_bb)
else_stmt)
           build_br_merge;

      ignore(L.build_cond_br bool_val then_bb else_bb builder);
      L.builder_at_end context merge_bb

      | SWhile (predicate, body) ->
      let pred_bb = L.append_block context "while" the_function in
      ignore(L.build_br pred_bb builder);

      let body_bb = L.append_block context "while_body" the_function
in
      add_terminal (stmt (L.builder_at_end context body_bb) body)
      (L.build_br pred_bb);

      let pred_builder = L.builder_at_end context pred_bb in
      let bool_val = expr pred_builder predicate in

      let merge_bb = L.append_block context "merge" the_function in
      ignore(L.build_cond_br bool_val body_bb merge_bb pred_builder);
      L.builder_at_end context merge_bb
```

```
    (* Implement for loops as while loops *)
    | SFor (e1, e2, e3, body) -> stmt builder
    ( SBlock [SExpr e1 ; SWhile (e2, SBlock [body ; SExpr e3]) ] )
    in

    (* Build the code for each statement in the function *)
    let builder = stmt builder (SBlock fdecl.sbody) in

    (* Add a return if the last block falls off the end *)
    add_terminal builder (match fdecl.styp with
        A.Void -> L.build_ret_void
    | A.Float -> L.build_ret (L.const_float float_t 0.0)
    | t -> L.build_ret (L.const_int (ltype_of_typ t) 0))
  in

  List.iter build_function_body functions;
  the_module
```

9.1.7 matrix.c

This file ‘matrix.c’ is a library of all functions built in PANAMX, such as concatLR(m, n) which merges two matrices horizontally, rank(m) which returns rank of a matrix ‘m’, and much more.

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <float.h>

typedef struct Matrix {
  int row;
  int col;
  double **mat;
} *matrix;

matrix initMatrix(int row, int col) {
```

```
  matrix m = (matrix)malloc(sizeof(struct Matrix));
  m->row = row;
  m->col = col;
  m->mat = (double **)malloc(sizeof(double *) * row);
  for (int i = 0; i < row; i++) {
      m->mat[i] = (double *)malloc(sizeof(double) * col);
  }
  return m;
}

matrix buildMatrix(int row, int col, double *arr) {
  matrix m = initMatrix(row, col);
  for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
      m->mat[i][j] = *(arr + i * col + j);
      }
  }
  return m;
}

matrix buildMatrixEmpty(int row, int col) {
  matrix m = initMatrix(row, col);
  for (int i = 0; i < row; i++) {
      for (int j = 0; j < col; j++) {
      m->mat[i][j] = 0;
      }
  }
  return m;
}

double matrixAccess(matrix m, int row, int col) {
  if (m->row <= row || m->col <= col) {
      perror("matrix index out of bound");
      exit(1);
  }
  return m->mat[row][col];
}
```

```
matrix matrixSlice(matrix m, int s1, int e1, int s2, int e2) {
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  if (m->row < e1 || m->col < e2) {
      perror("matrix index out of bound");
      exit(1);
  }
  int row = e1 - s1;
  int col = e2 - s2;
  if (row <= 0 || col <= 0) {
      perror("Matrix dimensions should be greater than zero");
      exit(1);
  }
  matrix tmp = initMatrix(row, col);
  for (int i = s1, k = 0; i < e1; i++, k++) {
      for (int j = s2, l = 0; j < e2; j++, l++) {
      tmp->mat[k][l] = m->mat[i][j];
      }
  }
  return tmp;
}

double matrixAssign(matrix m, int i, int j, double val) {
  if (m->row <= i || m->col <= j) {
      perror("Matrix Index Out of Bounds");
      exit(1);
  }
  m->mat[i][j] = val;
  return val;
}

void printMatrix(matrix m) {
  if (m->row == 0 || m->col == 0) {
      printf("Empty Matrix\n");
  }
  else {
```

```
      printf("[\n");
      for (int i = 0; i < m->row; i++) {
      printf("\t");
      for (int j = 0; j < m->col; j++) {
      printf("%.3f   ", m->mat[i][j]);
      }
      printf("\n");
      }
      printf("]\n");
  }
}

void freeMatrix(matrix m) {
  for (int i = 0; i < m->row; i++)
  free(m->mat[i]);
  free(m->mat);
  free(m);
}

int getHeight(matrix m) {
  if (m == NULL || m->mat == NULL)
  return 0;
  else
  return m->row;
}

int getWidth(matrix m) {
  if (m == NULL || m->mat == NULL)
  return 0;
  else
  return m->col;
}

matrix addMatrixDouble(matrix m, double n, int minus) {
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
```

```
  if (minus) {
      n = -n;
  }
  int h = m->row, w = m->col;
  matrix result = initMatrix(h, w);
  for (int i = 0; i < h; i++) {
      for (int j = 0; j < w; j++) {
      result->mat[i][j] = m->mat[i][j] + n;
      }
  }
  return result;
}

matrix subDoubleMatrix(double n, matrix m) {
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  int h = m->row, w = m->col;
  matrix result = initMatrix(h, w);
  for (int i = 0; i < h; i++) {
      for (int j = 0; j < w; j++) {
      result->mat[i][j] = n - m->mat[i][j];
      }
  }
  return result;
}

matrix addMatrixMatrix(matrix m, matrix n, int minus) {
  if (m == NULL || m->mat == NULL || n == NULL || n->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  if (m->row != n->row || m->col != n->col) {
      perror("Error in matrix addition: dimension mismatched");
      exit(1);
  }
  int h = m->row, w = m->col;
```

```
  matrix result = initMatrix(h, w);
  if (minus) {
      for (int i = 0; i < h; i++)
      for (int j = 0; j < w; j++)
      result->mat[i][j] = m->mat[i][j] - n->mat[i][j];
  }
  else {
      for (int i = 0; i < h; i++)
      for (int j = 0; j < w; j++)
      result->mat[i][j] = m->mat[i][j] + n->mat[i][j];
  }
  return result;
}

matrix mulMatrixDouble(matrix m, double n, int div) {
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  int h = m->row, w = m->col;
  matrix result = initMatrix(h, w);
  if (div) {
      n = 1 / n;
  }
  for (int i = 0; i < h; i++) {
      for (int j = 0; j < w; j++) {
      result->mat[i][j] = m->mat[i][j] * n;
      }
  }
  return result;
}

matrix mulMatrixMatrix(matrix m, matrix n) {
  if (m == NULL || m->mat == NULL || n == NULL || n->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  if (m->col != n->row) {
```

```
        perror("Error in matrix multiplication: dimension mismatched");
        exit(1);
    }
    int h = m->row, w = n->col, p = m->col;
    matrix result = initMatrix(h, w);
    for (int i = 0; i < h; i++) {
        for (int j = 0; j < w; j++) {
        double tmp = 0;
        for (int k = 0; k < p; k++) {
        tmp += m->mat[i][k] * n->mat[k][j];
        }
        result->mat[i][j] = tmp;
        }
    }
    return result;
}

matrix mulElementWiseMatrix(matrix m, matrix n, int div) {
    if (m == NULL || m->mat == NULL || n == NULL || n->mat == NULL) {
        perror("Empty Matrix");
        exit(1);
    }
    if (m->row != n->row || m->col != n->col) {
        perror("Error in element wise operation: dimension
mismatched");
        exit(1);
    }
    int h = m->row, w = m->col;
    matrix result = initMatrix(h, w);
    if (div) {
        for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++)
        result->mat[i][j] = m->mat[i][j] / n->mat[i][j];
    }
    else {
        for (int i = 0; i < h; i++)
        for (int j = 0; j < w; j++)
        result->mat[i][j] = m->mat[i][j] * n->mat[i][j];
```

```
  }
  return result;
}

// helper function for rref() function, swaps rows of given matrix by
reference
static void swap(matrix m, int a, int b) {
  double temp;
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  else if (m->row <= a || m->col <= b) {
      perror("Matrix Index Out of Bounds");
      exit(1);
  }
  else {
      for (int i = 0; i < m->col; i++) {
      temp = m->mat[i][a];
      m->mat[i][a] = m->mat[i][b];
      m->mat[i][b] = temp;
      }
  }
}

// helper function for rref() function, divides specified row in
matrix by a non-zero scalar
static void divideRow(matrix m, int row, double scalar) {
  if (m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  else if (scalar == 0) {
      perror("Divide By Zero Error");
      exit(1);
  }
  else if (m->row <= row) {
      perror("Matrix Index Out of Bounds");
```

```
        exit(1);
    }
    else {
        for (int i = 0; i < m->col; i++) {
        m->mat[i][row] *= scalar;
        }
    }
}

// helper function for rref() function, subtracts a multiple of one
row from another
// (e.g., row a = row a - row b * (scalar)
static void subtractRow(matrix m, int a, int b, double scalar) {
    if (m == NULL || m->mat == NULL) {
        perror("Empty Matrix");
        exit(1);
    }
    else if (m->row <= a || m->row <= b) { // note: should this be <=
        perror("Matrix Index Out of Bounds");
        exit(1);
    }
    else {
        for (int i = 0; m->col; i++) {
        m->mat[i][a] -= m->mat[i][b]*scalar;
        }
    }
}

// swaps rows a and b from the given matrix m
matrix rowSwap(matrix m, int a, int b) {
    // should not work if matrix is empty
    if (m == NULL || m->mat == NULL) {
        perror("Empty Matrix");
        exit(1);
    }
    // should not work for rows out of bounds
    else if (m->row <= a || m->row <= b) {
        perror("Matrix Index Out of Bounds");
```

```
        exit(1);
    }
    // should work for all other cases
    else {
        // create new matrix with all entries initialized to zero
        matrix new = buildMatrixEmpty(m->row, m->col);
        // iterate through all entries to initialize correct values
        for (int i = 0; i < new->row; i++) {
        for (int j = 0; j < new->col; j++) {
        if (i == a) {
              new->mat[i][j] = m->mat[b][j];
        }
        else if (i == b) {
              new->mat[i][j] = m->mat[a][j];
        }
        else {
              new->mat[i][j] = m->mat[i][j];
        }
        }
        }
        return new;
    }
}

// swaps columns a and b from the given matrix m
matrix colSwap(matrix m, int a, int b) {
    // should not work if matrix is empty
    if (m == NULL || m->mat == NULL) {
        perror("Empty Matrix");
        exit(1);
    }
    // should not work for columns out of bounds
    else if (m->col <= a || m->col <= b) {
        perror("Matrix Index Out of Bounds");
        exit(1);
    }
    // should work for all other cases
    else {
```

```
      // create new matrix with all entries initialized to zero
      matrix new = buildMatrixEmpty(m->row, m->col);
      // iterate through all entries to initialize correct values
      for (int i = 0; i < new->row; i++) {
      for (int j = 0; j < new->col; j++) {
      if (j == a) {
            new->mat[i][j] = m->mat[i][b];
      }
      else if (i == b) {
            new->mat[i][j] = m->mat[i][b];
      }
      else {
            new->mat[i][j] = m->mat[i][j];
      }
      }
      }
      return new;
  }
}

// returns the sum of all values in the matrix m; for now, only works
for
// double type
double sum(matrix m) {
  // total starts at zero; handles error cases
  double s = 0;
  if(m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  else {
      // iterate through all entries
      for (int i = 0; i < m->row; i++) {
      for (int j = 0; j < m->col; j++) {
      // add each entry to running total
      s = s + m->mat[i][j];
      }
      }
```

```
  }
  return s;
}

// returns the average of all entries in a matrix m; for now, only
works for
// double type
double mean(matrix m) {
  double avg = 0;
  double tot = 0;
  if(m == NULL || m->mat == NULL) {
      perror("Empty Matrix");
      exit(1);
  }
  else {
      // iterate through all entries
      for (int i = 0; i < m->row; i++) {
      for (int j = 0; j < m->col; j++) {
      // avg keeps track of the sum; tot keeps track of the size
      tot++;
      avg = avg + m->mat[i][j];
      }
      }
  }
  // average = total / size
  avg = avg / tot;
  return avg;
}

matrix trans(matrix m) {
  // if the matrix m does not exist, return NULL
  if(m == NULL || m->mat == NULL) {
      printf("Empty Matrix\n");
      return NULL;
  }
  // all other cases, transpose by swapping elements accordingly
  else {
      // create new matrix with all entries initialized to zero
```

```
      matrix new = buildMatrixEmpty(m->col, m->row);
      // iterate through all entries to initialize correct values
      for (int i = 0; i < new->row; i++) {
      for (int j = 0; j < new->col; j++) {
      // transpose each value
      new->mat[i][j] = m->mat[j][i];
      }
      }
      return new;
  }
}

// returns nxn identity matrix given n
matrix iden(int n) {
  matrix empty = buildMatrixEmpty(n, n);
  for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
      if (i == j) {
      empty->mat[i][j] = 1;
      }
      }
  }
  return empty;
}

// returns an "array" (e.g., 1 x N matrix) of eigenvalues for given
matrix
matrix eig(matrix m) {
  return 0;
}

void getCofactor(matrix m, matrix tmp, int p, int q, int n) {
  int i = 0, j = 0;
  for (int row = 0; row < n; row++) {
      for (int col = 0; col < n; col++) {
      if (row != p && col != q) {
      tmp->mat[i][j++] = m->mat[row][col];
      if (j == n - 1) {
```

```
            j = 0;
            i++;
        }
        }
        }
    }
}

double detOfMatrix(matrix m, int n) {
    double D = 0;
    if (n == 1) {
        return m->mat[0][0];
    }
    int sign = 1;
    matrix tmp = initMatrix(m->row, m->col);
    for (int f = 0; f < n; f++) {
        getCofactor(m, tmp, 0, f, n);
        D += sign * m->mat[0][f] * detOfMatrix(tmp, n-1);
        sign = -sign;
    }
    return D;
}

// returns the determinant of given matrix
double det(matrix m) {
    if (m == NULL || m->mat == NULL) {
        perror("Empty Matrix");
        exit(1);
    }
    else if (m->row != m->col) {
        perror("Cannot find determinant of non-square matrices.");
        exit(1);
    }
    int n = m->col;
    double D = detOfMatrix(m, n);
    return D;
}
```

```
// Function to get adjoint
void adjoint(matrix a, matrix adj) {
  if (a->row == 1) {
      adj->mat[0][0] = 1;
      return;
  }
  int sign = 1;
  matrix tmp = initMatrix(a->row, a->col);
  int n = a->row;
  for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
      getCofactor(a, tmp, i, j, n);
      sign = ((i+j)%2==0)? 1: -1;
      adj->mat[j][i] = (sign)*(detOfMatrix(tmp, n-1));
      }
  }
}

// returns inverse matrix if invertible
matrix inv(matrix m) {
  double deter = det(m);
  if (deter == 0) {
      perror("Cannot find inverse of Singular Matrix.");
      exit(1);
  }
  matrix adj = initMatrix(m->row, m->col);
  adjoint(m, adj);
  matrix inv = initMatrix(m->row, m->col);
  for (int i = 0; i < inv->row; i++) {
      for (int j = 0; j < inv->col; j++) {
      inv->mat[i][j] = adj->mat[i][j]/deter;
      }
  }
  return inv;
}


void mulandaddRows(matrix m, int dest, int src, double mplr)
```

```
{
  double *drow, *srow;
  drow = m->mat[dest];
  srow = m->mat[src];
  for (int i = 0; i < m->col; i++)
  drow[i] += mplr * srow[i];
}

void swapRows(matrix m, int a, int b) {
  double *r1, *r2, temp;
  if (a == b) return;
  r1 = m->mat[a];
  r2 = m->mat[b];
  for (int i = 0; i < m->col; i++) {
      temp = r1[i];
      r1[i] = r2[i];
      r2[i] = temp;
  }
}

void normalizeRow(matrix m, int row, int lead)
{
  double *drow = m->mat[row];
  double lv = drow[lead];
  if (fabs(lv) <= 0.00001) {
      if (lv < 0)
      lv = -0.00001;
      else
      lv = 0.00001;
  }
  for (int i = 0; i < m->col; i++)
  drow[i] /= lv;
}

// returns rref of given matrix m
matrix rref(matrix m) {
  int i;
  double lv;
```

```
  int rowCount = m->row;
  int lead = 0;
  for (int r = 0; r < rowCount; r++) {
      if (lead >= m->col)
      break;
      i = r;
      while (fabs(m->mat[i][lead]) <= 0.00001) {
      i++;
      if (i == rowCount) {
      i = r;
      lead++;
      if (lead == m->col)
      break;
      }
      }
      swapRows(m, i, r);
      normalizeRow(m, r, lead );
      for (i = 0; i < rowCount; i++) {
      if (i != r) {
      lv = m->mat[i][lead];
      mulandaddRows(m, i, r, -lv) ;
      }
      }
      lead++;
  }
  return m;
}


// returns rank of a given matrix
double rank(matrix m) {
  // get the reduced row echelon form of given matrix
  matrix new = rref(m);
  double rnk = 0;
  // count the number of non-zero rows
  for (int i = 0; i < new->row; i++) {
      int zero = 0;
      for (int j = 0; j < new->col; j++) {
```

```
      if (new->mat[i][j] != 0) {
      zero = 1;
      break;
      }
      }
      rnk += zero;
  }
  return rnk;
}

// concats top bot matrices
matrix concatTB(matrix a, matrix b) {
  int arow = a->row;
  int acol = a->col;
  int brow = b->row;
  int bcol = b->col;
  if (a == NULL || a->mat == NULL) {
      return b;
  }
  else if (b == NULL || b->mat == NULL) {
      return a;
  }
  else if (acol != bcol) {
      perror("Cannot combine matrices with different widths");
      exit(1);
  }
  else {
      matrix new = initMatrix(arow+brow, acol);
      for (int i = 0; i < arow; i++) {
      for (int j = 0; j < acol; j++) {
      new->mat[i][j] = a->mat[i][j];
      }
      }
      for (int i = arow; i < arow+brow; i++) {
      for (int j = 0; j < bcol; j++) {
      new->mat[i][j] = b->mat[i-arow][j];
      }
      }
```

```
      return new;
  }
}

// concats left right matrices
matrix concatLR(matrix a, matrix b) {
  int arow = a->row;
  int acol = a->col;
  int brow = b->row;
  int bcol = b->col;
  if (a == NULL || a->mat == NULL) {
      return b;
  }
  if (b == NULL || b->mat == NULL) {
      return a;
  }
  if (arow != brow) {
      perror("Cannot combine matrices with different heights");
      exit(1);
  }
  matrix new = initMatrix(arow, acol+bcol);
  for (int i = 0; i < arow; i++) {
      for (int j = 0; j < acol; j++) {
      new->mat[i][j] = a->mat[i][j];
      }
  }
  for (int i = 0; i < brow; i++) {
      for (int j = acol; j < acol+bcol; j++) {
      new->mat[i][j] = b->mat[i][j-acol];
      }
  }
  return new;
}

// Calculate square root of double
double sqrtd(double num) {
  double guess, e, upperbound;
  guess = 1;
```

```
  e = 0.001;
  do {
      upperbound = num / guess;
      guess = (upperbound + guess) / 2;
  } while (!(guess * guess >= num - e && guess * guess <= num + e));
  return guess;
}

int sqrti(int x) {
  return (int)sqrtd((double)x);
}

int absi(int N){
  return ((N<0)?(-N):(N));
}

double absd(double N){
  return ((N<0)?(-N):(N));
}

double poweri(int e, int x) {
  int i;
  int r = 1;
  for (i = 0; i < e; i++) {
      r *= x;
  }
  return r;
}

double powerd(double e, int x) {
  int i;
  double r = 1;
  for (i = 0; i < e; i++) {
      r *= x;
  }
  return r;
}
```

```
double nrooti(int n, int x) {
  int d, r = 1;
  if (!x) {
      return 0;
  }
  if (n < 1 || (x < 0 && !(n&1))) {
      return NAN;
  }
  do {
      d = (x / poweri(r, n - 1) - r) / n;
      r += d;
  }
  while (d >= DBL_EPSILON * 10 || d <= -DBL_EPSILON * 10);
  return r;
}


double nrootd(int n, double x) {
  double d, r = 1;
  if (!x) {
      return 0;
  }
  if (n < 1 || (x < 0 && !(n&1))) {
      return NAN;
  }
  do {
      d = (x / powerd(r, n - 1) - r) / n;
      r += d;
  }
  while (d >= DBL_EPSILON * 10 || d <= -DBL_EPSILON * 10);
  return r;
}
```

9.1.8 Makefile

```
# "make all" builds the executable

.PHONY : all
all : panamx.native matrix.o
```

```
# "make panamx.native" compiles the compiler

panamx.native :
     opam config exec -- \
     ocamlbuild -use-ocamlfind panamx.native

# "make test" Compiles everything and runs the regression tests

.PHONY : test
test : all testall.sh
     ./testall.sh

# "make clean" removes all generated files

.PHONY : clean
clean :
     ocamlbuild -clean
     rm -rf testall.log ocamlllvm *.diff matrix.o *.ll *.out *.exe
*.s *.err

# Link the matrix.c to llvm

matrix : matrix.c
     cc -o matrix -DBUILD_TEST matrix.c
```

9.2 Demo Cases

These cases are in the directory tests/demos. It consists of demo cases we used in our presentation.

9.2.1 test-axb.px

This program showcases how to solve a matrix equation, expressed as “ax = b”. It also shows you that you can build your own matrices-merging function, which is combineTopDown(a,b) written in the code block.

```
matrix solve(matrix A, matrix b) {
```

```
	return inv(A) * b;
}
matrix combineTopDown(matrix a, matrix b) {
	int i;
	int j;
	int arow;
	int acol;
	int brow;
	int bcol;
	matrix new;

	arow = matrixHeight(a);
	acol = matrixWidth(a);
	brow = matrixHeight(b);
	bcol = matrixWidth(b);

	if (acol != bcol) {
	prints("Cannot combine matrices with different widths");
	return a;
	}

	new = <arow+brow, acol>;
	for (i = 0; i < arow; i++) {
	for (j = 0; j < acol; j++) {
		new[i][j] = a[i][j];
	}
	}
	for (i = arow; i < arow+brow; i++) {
	for (j = 0; j < bcol; j++) {
		new[i][j] = b[i-arow][j];
	}
	}
	return new;
}

int main() {
	matrix e1;
	matrix e2;
```

```
    matrix e3;
    matrix comb;
    matrix A;
    matrix b;
    matrix x;

    // Equations
    e1 = [1, 1, 1, 6];
    e2 = [0, 2, 5, -4];
    e3 = [2, 5, -1, 27];

    // Combine into 3 X 4 matrix
    comb = combineTopDown(e1, e2);
    comb = combineTopDown(comb, e3);

    // Slice matrix
    A = comb[0:3][0:3];
    b = comb[0:3][3:4];
    x = solve(A, b);

    prints("Matrix A:");
    printm(A);
    prints("Matrix B:");
    printm(b);
    prints("Solve for x:");
    printm(x);

    return 0;
}
```

test-axb.out

```
Matrix A:
[
    1.000   1.000   1.000
    0.000   2.000   5.000
    2.000   5.000   -1.000
]
Matrix B:
```

```
[
    6.000
    -4.000
    27.000
]
Solve for x:
[
    5.000
    3.000
    -2.000
]
```

9.2.2 test-eigen.px

```
matrix eigen(matrix A) {
    double a;
    double b;
    double c;
    matrix vec;
    double e1;
    double e2;
    vec = <1, 2>;

    a = 1.0;
    b = -A[0][0] - A[1][1];
    c = A[0][0] * A[1][1] - A[0][1] * A[1][0];

    e1 = (-b+sqrtd((b*b-4.0*a*c))) / (2.0 * a);
    e2 = (-b-sqrtd((b*b-4.0*a*c))) / (2.0 * a);
    vec[0][0] = e1;
    vec[0][1] = e2;
    return vec;
}

int main() {
    matrix A;
```

```
    double a;
    double b;
    double c;
    matrix eigenvec;

    A = [3, 2;
         1, 4];
    eigenvec = eigen(A);

    prints("Matrix A:");
    printm(A);
    prints("Eigenvalues:");
    printm(eigenvec);
    return 0;
}
```

test-eigen.out

```
Matrix A:
[
      3.000   2.000
      1.000   4.000
]
Eigenvalues:
[
      5.000   2.000
]
```

9.2.3 test-perceptron.px

This program shows implementation of basic perceptron learning algorithm that is widely used in machine learning research.

```
// Perceptron Learning Algorithm
int sign(double x) {
  if (x > 0.0) {
      return 1;
  }
  if (x < 0.0) {
      return -1;
```

```
  }
  return 0;
}

int f(matrix X, matrix w) {
  double sum;
  matrix tmp;
  sum = 0.0;
  tmp = X * trans(w);
  sum = tmp[0][0];
  return sign(sum);
}
matrix perceptron(matrix df) {
  matrix X;
  matrix Y;
  matrix ones;
  matrix w;
  matrix output;
  matrix prev_w;
  int n;
  int i;
  matrix diffvec;
  double diff;
  int iter;

  n = matrixHeight(df);
  X = df[0:n][0:2];
  Y = df[0:n][2:3];
  ones = <n, 1>;
  for (i = 0; i < n; i++) {
      ones[i][0] = 1;
  }
  X = concatLR(X, ones);
  w = [0, 0, 0];
  output = [0, 0, 0];
  iter = 1;
  while (diff != 0.0) {
      diff = 0.0;
```

```
      prev_w = w;
      for (i = 0; i < n; i++) {
      if (Y[i][0] * f(X[i:i+1][0:3], w) <= 0.0) {
      w = w + Y[i][0] * X[i:i+1][0:3];
      }
      }
      diffvec = w - prev_w;
      diff = 0.0;
      for (i = 0; i < 3; i++) {
      diff = diff + absd(diffvec[0][i]);
      }
      iter = iter + 1;
      output = concatTB(output, w);
  }
  n = matrixHeight(output);
  output = output[1:n][0:3];
  return output;
}

int main() {
      matrix df;
      matrix res;
      int last;
      df = [8, -11, 1;
      7, 7, -1;
      12, -20, 1;
      14, -3, -1;
      12, 8, -1;
      1, -12, 1;
      15, 5, -1;
      7, -10, 1;
      10, 4, -1;
      6, 2, 1;
      8, 12, -1;
      2, 20, -1;
      1, -12, 1;
      9, 8, -1;
      3, 3, 1;
```

```
      5, 6, 1;
      1, 11, 1];

      prints("Input Data:");
      printm(df);

      res = perceptron(df);

      prints("Output Results:");
      printm(res);

      prints("Decision Boundary:");
      last = matrixHeight(res);
      res = res[last-1:last][0:3];
      printm(res);

      prints("Slope:");
      printf(-res[0][0]/res[0][1]);

      prints("Offset:");
      printf(res[0][2]/res[0][1]);

      return 0;
}
```

test-perceptron.out

```
Input Data:
[
    8.000   -11.000   1.000
    7.000   7.000   -1.000
    12.000   -20.000   1.000
    14.000   -3.000   -1.000
    12.000   8.000   -1.000
    1.000   -12.000   1.000
    15.000   5.000   -1.000
    7.000   -10.000   1.000
    10.000   4.000   -1.000
    6.000   2.000   1.000
    8.000   12.000   -1.000
```

```
    2.000   20.000   -1.000
    1.000   -12.000   1.000
    9.000   8.000   -1.000
    3.000   3.000   1.000
    5.000   6.000   1.000
    1.000   11.000   1.000
]
Output Results:
[
    8.000   3.000   3.000
    3.000   9.000   6.000
    12.000   6.000   10.000
    6.000   1.000   12.000
    7.000   9.000   15.000
    5.000   2.000   16.000
    5.000   -1.000   18.000
    6.000   7.000   21.000
    3.000   8.000   23.000
    -1.000   -2.000   24.000
    2.000   7.000   26.000
    3.000   3.000   28.000
    3.000   6.000   30.000
    0.000   7.000   32.000
    1.000   3.000   34.000
    1.000   6.000   36.000
    2.000   2.000   38.000
    2.000   5.000   40.000
    -5.000   -2.000   39.000
    -5.000   -2.000   39.000
]
Decision Boundary:
[
    -5.000   -2.000   39.000
]
Slope:
-2.5
Offset:
-19.5
```

9.2.4 test-structMatrix.px

```
struct Node {
    matrix x;
    matrix y;
};

int main() {
    struct Node p;
    p = <struct Node>;
    p.x = [1,2;3,4];
    p.y = [5,6;7,8];
    printm(p.x);
    printm(p.y);
    return 0;
}
```

test-structMatrix.out

```
[
      1.000   2.000
      3.000   4.000
]
[
      5.000   6.000
      7.000   8.000
]
```

9.3 Test Cases

These cases are in the directory tests/px_tests. It consists of comprehensive test cases that tests the features we implemented in PANAMX.

9.3.1 test-incdec.px

```
int main() {
    int a;
```

```
    a = 7;
    a--;
    print(a);
    a++;
    print(a);
    return 0;
}
```

test-incdec.out

```
6
7
```

9.3.2 test-matrix1.px

```
int main() {
    int i;
    int j;
    matrix mat;
    mat = <2, 3>;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf(mat[i][j]);
        }
    }

    mat[0][0] = 1.1;
    mat[0][1] = 2.2;
    mat[0][2] = 3.3;
    mat[1][0] = 4.4;
    mat[1][1] = 5.5;
    mat[1][2] = 666;
    for (i = 0; i < 2; i++) {
        for (j = 0; j < 3; j++) {
            printf(mat[i][j]);
        }
    }
    return 0;
}
```

test-matrix1.out

```
0
0
0
0
0
0
1.1
2.2
3.3
4.4
5.5
666
```

9.3.3 test-matrix2.px

```
int main() {
    matrix mat;
    mat = [1122, 2233; 7788, 8899];
    printm(mat);
    mat = [11, 2.2, 3.3; 4.4, 5.5, 6.6];
    printm(mat);
    return 0;
}
```

test-matrix2.out

```
[
     1122.000   2233.000
     7788.000   8899.000
]
[
     11.000   2.200   3.300
     4.400   5.500   6.600
]
```

9.3.4 test-matirx3.px

```
int main() {
```

```
    matrix mat;
    mat = <2, 3>;

    mat[0][0] = 1.1;
    mat[0][1] = 2.2;
    mat[0][2] = 3.3;
    mat[1][0] = 4.4;
    mat[1][1] = 5.5;
    mat[1][2] = 666;
    prints("matrix height:");
    print(matrixHeight(mat));
    prints("matrix width:");
    print(matrixWidth(mat));
    return 0;
}
```

test-matrix3.out

```
matrix height:
2
matrix width:
3
```

9.3.5 test-matrix4.px

```
int main() {
    matrix m;
    matrix n;
    m = [1, 2; 3, 4];
    n = 8.5 + m;
    printm(n);
    m = n - 1.5;
    printm(m);
    n = m * 2;
    printm(n);
    m = n / 4;
    printm(m);
    return 0;
}
```

test-matrix4.out

```
[
      9.500   10.500
      11.500   12.500
]
[
      8.000   9.000
      10.000   11.000
]
[
      16.000   18.000
      20.000   22.000
]
[
      4.000   4.500
      5.000   5.500
]
```

9.3.6 test-matrix5.px

```
int main() {
    matrix m;
    matrix n;
    m = [1.6, 3.9;
         2.4, 4.3;
         5.7, 7.2];
    n = [12.3, 15.6, 14.5, 18.8;
         23.4, 21.8, 9.5,  10.7];
    m = m * n;
    printm(m);
    free(m);
    free(n);
    return 0;
}
```

test-matrix5.out

```
[
      110.940   109.980   60.250   71.810
      130.140   131.180   75.650   91.130
```

```
      238.590   245.880   151.050   184.200
]
```

9.3.7 test-matrix6.px

```
int main() {
    matrix m;
    matrix n;
    matrix q;
    m = [1, 9, 3;
         2, 5, 7;
         3, 6, 4];
    n = [17.1, 23.2, 10.9;
         15.6, 21.7, 10.5;
         9.8,  18.2, 21.5];
    q = m + n;
    printm(q);
    q = n - m;
    printm(q);
    q = m .* n;
    printm(q);
    q = n ./ m;
    printm(q);
    return 0;
}
```

test-matrix6.out

```
[
      18.100   32.200   13.900
      17.600   26.700   17.500
      12.800   24.200   25.500
]
[
      16.100   14.200   7.900
      13.600   16.700   3.500
      6.800   12.200   17.500
]
[
      17.100   208.800   32.700
```

```
      31.200   108.500   73.500
      29.400   109.200   86.000
]
[
      17.100   2.578   3.633
      7.800   4.340   1.500
      3.267   3.033   5.375
]
```

9.3.8 test-matrixFunc.px

```
int main() {
    matrix m;
    m = [1, 2, 3;
         4, 5, 6;
         7, 8, 9];
    printm(m);
    printf(sum(m));
    printf(mean(m));
    printm(trans(m));
    printf(det(m));
    printm(rref(m));
    printf(rank(m));
    return 0;
}
```

test-matrixFunc.out

```
[
      1.000   2.000   3.000
      4.000   5.000   6.000
      7.000   8.000   9.000
]
45
5
[
      1.000   4.000   7.000
      2.000   5.000   8.000
      3.000   6.000   9.000
]
```

```
0
[
      1.000   0.000   -1.000
      -0.000   1.000   2.000
      0.000   0.000   0.000
]
2
```

9.3.9 test-matrixInv.px

```
int main() {
    matrix m;
    m = [5, -2, 2, 7;
         1, 0, 0, 3;
         -3, 1, 5, 0;
         3, -1, -9, 4];
    printm(inv(m));
    return 0;
}
```

test-matrixInv.out

```
[
      -0.136   0.864   -0.682   -0.409
      -0.636   2.364   -0.932   -0.659
      0.045   0.045   -0.023   -0.114
       0.045   0.045   0.227   0.136
]
```

9.3.10 test-matrixSlice.px

```
int main() {
    matrix m;
    m = [1, 2, 3, 4;
         5, 6, 7, 8;
         9, 10, 11, 12;
         13, 14, 15, 16];
    printm(m[1:3][2:4]);
    return 0;
}
```

test-matrixSlice.out

```
[
      7.000   8.000
      11.000   12.000
]
```

9.3.11 test-op3.px

```
int main() {
    double d;
    d = -2.75 + 15.125;
    printf(d);
    d = -6.5 / 2;
    printf(d);
    return 0;
}
```

test-op3.out

```
12.375
-3.25
```

9.3.12 test-printstr.px

```
int main()
{
  string sdf;
  sdf = "Hello World";
  prints(sdf);
  return 0;
}
```

test-printstr.out

```
Hello World
```

9.3.13 test-printTest.px

```
int main()
{
  int i;
  bool b;
  double a;
```

```
  string s;
  matrix m;
  i = 10;
  b = false;
  a = 11.0;
  s = "Hello World";
  m = [1,2;
       3,4];
  print(i);
  printb(b);
  printf(a);
  prints(s);
  printm(m);
  return 0;
}
```

test-printTest.out

```
10
0
11
Hello World
[
      1.000   2.000
      3.000   4.000
]
```

9.3.14 test-structMem.px

```
struct Node {
    int x;
    int y;
};

int main() {
    struct Node p;
    p = <struct Node>;
    p.x = 121;
```

```
    p.y = 999;
    print(p.x);
    print(p.y);
    return 0;
}
```

test-structMem.out

```
121
999
```

9.4 Fail Cases

These cases are in the directory tests/px_fails. It consists of fail cases that our semantic checker checks for.

9.4.1 fail-matrixDim.px

```
int main()
{
  matrix m;
  m = <2.0, false>; /* should fail because dimensions have invalid
types */
  m = [1, 2; 3, 4];
}
```

fail-matrixDim.err:

```
Fatal error: exception Failure("matrix height must be integer")
```

9.4.2 fail-matrixIndex.px

```
int main()
{
  matrix m;
  m = [1, 2; 3, 4];
  printf(m[0.0]["one"]); /* this should fail because indexes can only
be integers */
```

```
}
```

fail-matrixIndex.err:

```
Fatal error: exception Failure("index must be integer")
```

9.4.31 fail-matrixInit.px

```
int main() {
      matrix m;
      m = [1.6, 3.9;
      2.4, 4.3;
      5.7];        // 'm' ill-formed matrix.
      printm(m);
      return 0;
}
```

fail-matrixInit.err:

```
Stored value type does not match pointer operand type!
  store [5 x double] [double 1.600000e+00, double 3.900000e+00,
double 2.400000e+00, double 4.300000e+00, double 5.700000e+00], [6 x
double]* %matrix_array
 [6 x double]LLVM ERROR: Broken module found, compilation aborted!
```

9.4.4 fail-matrixTypeAssign.px

```
int main(){
  matrix m;
  m = ["s", 3.3;
       2.3, 3.4]; /* this should fail because matrix elements can
only be int/double */
  return 0;
}
```

fail-matrixTypeAssign.err:

```
Fatal error: exception Failure("matrix elements can only be
int/double type")
```

9.4.5 fail-structAssign.px

```
struct Node {
    int x;
};

int main() {
    struct Node p;
    p = <struct Node>;
    p.x = 2.0; /* this should fail because member type does not match
with value */
    return 0;
}
```

fail-structAssign.err:

```
Fatal error: exception Failure("illegal assignment int = float in p.x
= 2.0")
```

9.4.6 fail-structDup.px

```
struct Node {
    matrix x;
};

struct Node {
    matrix y; /* this should fail because this is a duplicate struct
with same name */
};

int main() {
    struct Node p;
    p = <struct Node>;
    return 0;
}
```

fail-structDup.err:

```
Fatal error: exception Failure("duplicate struct name Node")
```