The Objective

Linear Algebra is all the rage

A language to simplify Matrices
Team Roles

Language Guru - Abhishek Walia
Project Manager - Anuraag Advani
System Architect - Shardendu Gautam
Matrix Hello World

function main() {
    a = 1;                        /*We also had /*nested comments*/ working */
    b = 2;
    print (a+b);
    return 0;
}
Architecture

Source Program (source.matcv)

Scanner.ml

Tokens Generated

parser.ml

AST Generated

semant.ml

Annotated AST with inferred types

MatCV Library

codegen.ml

LLVM IR

Code generated in the target language
Key Features

Just start writing code, no main function required!!

```plaintext
a = 2;
b = 2;
c = 1 + b;
d = [5][6];
```

...
Types Supported

- Integer
- Boolean
- Matrices (N - Dimensional!)
- Void

But don’t worry about them, as they are inferred!!
Type Inference

Code:

```plaintext
b = foo();

function foo(){
    c={1,2,3;4,5,6;7,8,9};
    return c;
}
```

After semantic analysis:

```plaintext
Mat(2) b = foo();

Mat(2) Func_foo(){
    Mat(2) c = {1,2,3;4,5,6;7,8,9};
    return c;
}
```
Type Inference

- Construct an annotated parse tree.
- Collect constraints.
- Solve these constraints to infer types.

After semantic analysis:

```plaintext
Mat(2) b = foo();
Mat(2) Func_foo(){
    Mat(2) c = {1,2,3;4,5,6;7,8,9};
    return c;
}
```
Function

- We pass matrices by reference in functions, a design choice to make our language convenient for users.
- However, integers and booleans are passed by value.
- To declare a function the following syntax is used:

  ```
  function foo()
  ```

- “main()” is not needed and is reserved and cannot be used as a function.
Key Features

How many dimensions do you want in a matrix?

\[ a = \begin{bmatrix} 5 & 3 & 2 & 3 \\ \vdots \end{bmatrix} \]

We support n dimensions along with key features like add, subtract, etc.

Want a different way to allocate matrices?

We support it:  
\[ a = \{ 1 , 2 , c + d , 4 ; 5 , 6 , \text{func()} , a[0] ; 9 , \text{foo + bar} , 11 , 12 \} \]
It’s all an illusion

- We use only 1’D matrices but the user uses it as a normal n’D matrix, you ask how? Well, some pointer magic.

**Want the index of an element?**

| Index 0: Number Of Dimensions | Index 1: Size along Dimension 1 | Index 2: Size along Dimension 2 | …… | Matrix content In Row major -> |
Scoping done right

Declare variables anywhere:

A local and global map are used to manage the scope for blocks.

Separate memory map to keep track of allocations
Key Features

Control Flow operations

If..else

for(;;)

while()

continue
For loop done right

Added an additional block to support continue.
Memory Management

- We malloc memory for the variable sized matrices on the heap and the integers and booleans are stored on the stack.
- A memory map is maintained which can be used to free the unused memory.
A powerful language with a powerful library

Supports matrix functions eg.

- Add
- Subtract

Result of the operation stored in the first operand
<table>
<thead>
<tr>
<th>Local</th>
<th>Global</th>
<th>Put in memory map?</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Put, alloca, add to local map</td>
</tr>
<tr>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Put, alloca, add to local map</td>
</tr>
<tr>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Alloc, Add to local map</td>
</tr>
</tbody>
</table>
Testing

Separate Testing for different modules

Pass and fail tests

- Parser/Scanner tests
- Semant tests
- Codegen Tests
- AST Printing to ease debugging

```
Mat(3) r = [3][2][4];
Mat(3) r[2][2][1]=3;
Int q = 3;
Int b = 4;
```

```
./matcv < testParserFail
Fatal error: exception Parsing.Parse_error
make: *** [runtest] Error 2
```
Demo Time!