TWISTER

a language designed for
image manipulation

TEAM MEMBERS

Manager: Anand Sundaram (as5209)

Language Guru: Arushi Gupta (ag3309)

System Architect: Annalise Mariottini (aim2120)

Tester: Chuan Tian (ct2698)

THE GOAL

Twister is an image manipulation language designed with users in mind who may not be familiar with complex syntax. In this presentation we will demonstrate how users can use Twister to write a convolution function.

PROJECT LOG

Feb 22 Scanner compiles! Seems mostly complete.

Mar 1 Parser, ast, and scanner all build w/o errors!

Mar 20 Added a codegen file

Mar 25 Hello world works now

Mar 26 Tests running on Travis

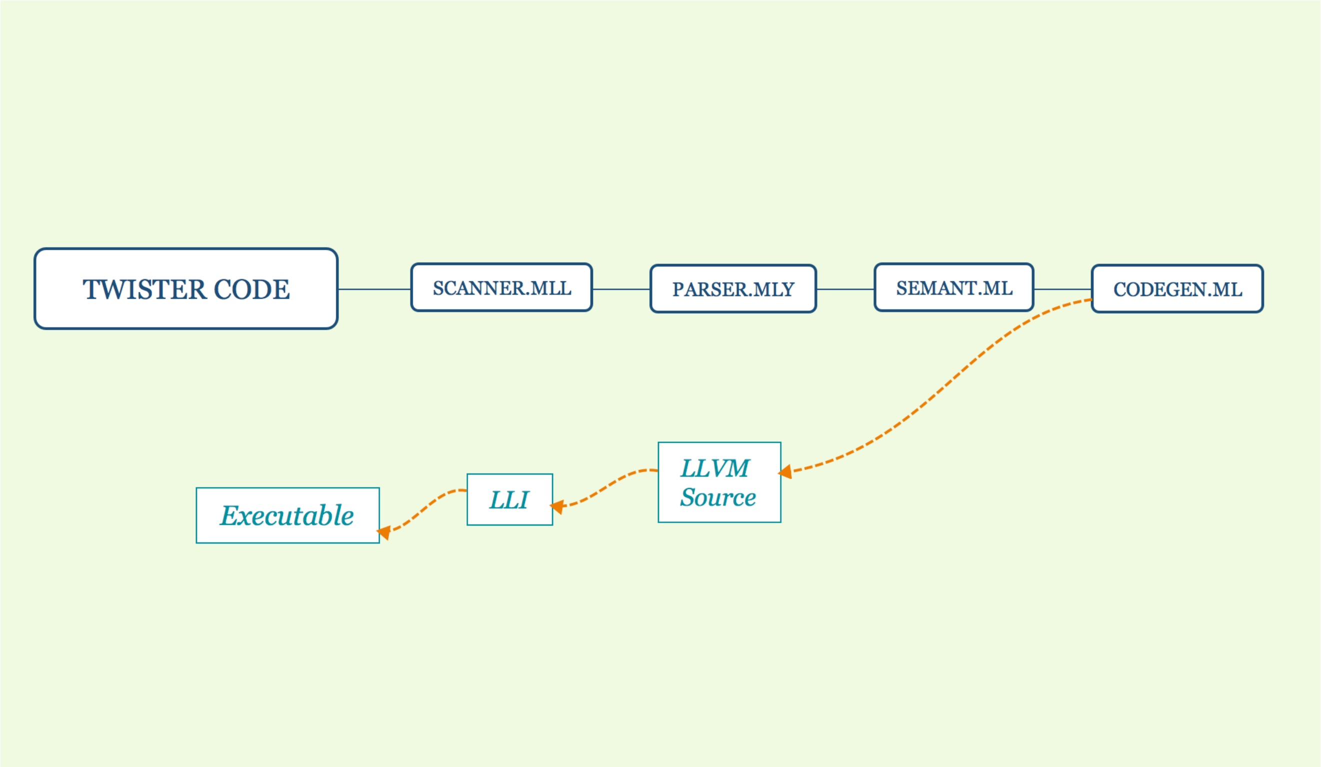
Apr 24 Semant is ready for testing

May 10 Final presentation

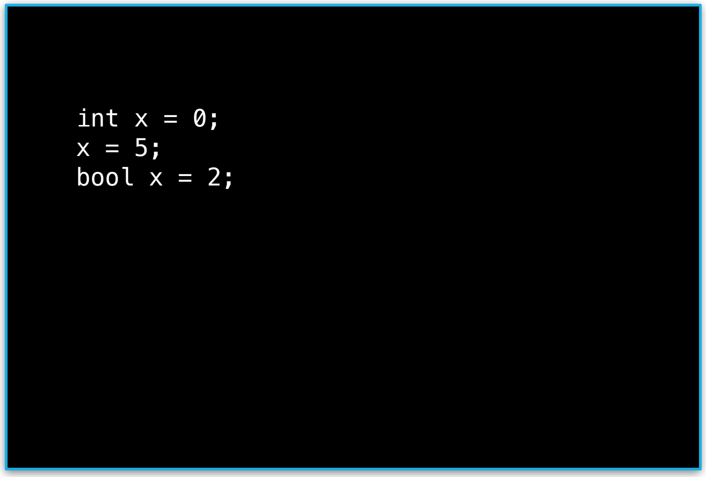
LANGUAGE FEATURES

- Nested functions
- Easy Matrix Manipulation
- Functions do not have to be declared at the beginning of a file
- Simplified for loops with for(elem in range()) syntax

COMPILER ARCHITECTURE



SEMANT: UNIQUE VARIABLE NAMES

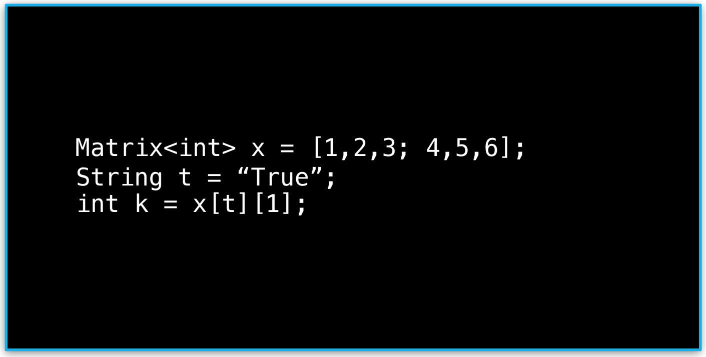


```
int x = 0;
x = 5;
bool x = 2;
```

Output:

```
Error: "
Name x is already declared with type int
" @ statement: bool x = 2;
```

SEMANT: TYPE CHECK INDICES



```
Matrix<int> x = [1,2,3; 4,5,6];
String t = “True”;
int k = x[t][1];
```

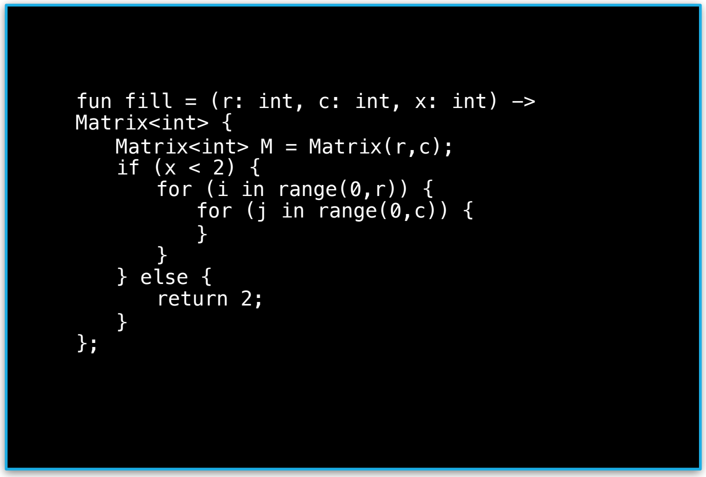
Output:

Error: “

Expression t has type List<Char> and cannot be used as a matrix index.

“ @ statement: int k = x[t][1];

SEMANT: TYPE CHECK RETURN

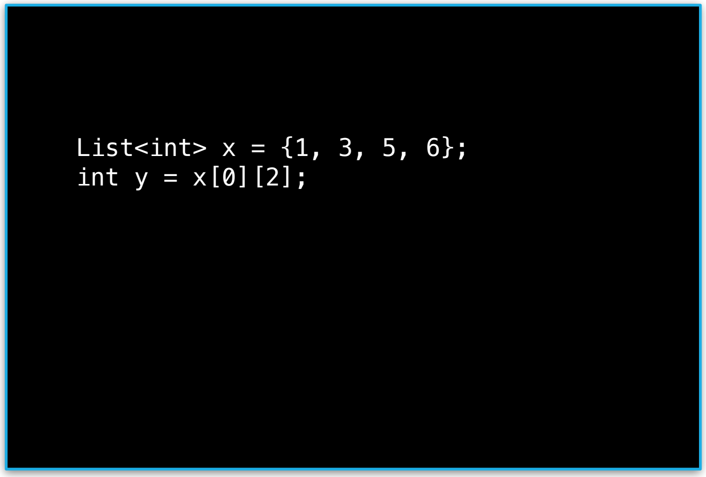


```
fun fill = (r: int, c: int, x: int) ->
Matrix<int> {
   Matrix<int> M = Matrix(r,c);
   if (x < 2) {
      for (i in range(0,r)) {
         for (j in range(0,c)) {
         }
      }
   } else {
      return 2;
   }
};
```

Output:

```
Error: "
Error: "
Error: "
Statement "
return 2;
"
 would return invalid type int where return values should be of type Matrix<int>
" @ statement: return 2;
" @ statement: if (x < 2) {
for i in range(0,r): {
 for j in range(0,r): {
 M[i][j] = x;
}
}
return M;
} else {
return 2;
}
" @ statement: fun fill = (r : int,c : int,x : int) -> Matrix<int> {Matrix<int> M = Matrix(r,c);
if (x < 2) {
for i in range(0,r): {
 for j in range(0,r): {
 M[i][j] = x;
}
}
return M;
} else {
return 2;
}};
```

SEMANT: TYPE CHECK RETURN



```
List<int> x = {1, 3, 5, 6};
int y = x[0][2];
```

Output:

```
Error: "
Variable x is of type List<int>, not of type
Matrix<element_type>, and cannot be indexed into.
" @ statement: int y = x[0][2];
```

NESTED FUNCTIONS

LLVM IR (trimmed)

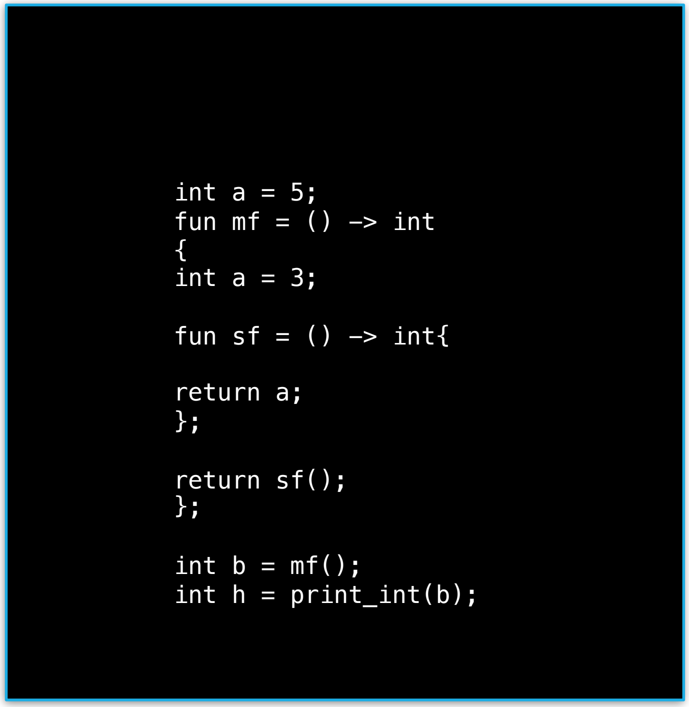
```
define i32 @sf(i32 %x, i32 %a) {
entry:
  %x1 = alloca i32
  store i32 %x, i32* %x1
  %a2 = alloca i32
  store i32 %a, i32* %a2
  %"%tmp" = load i32, i32* %x1
  %"%tmp3" = load i32, i32* %a2
  %"%tmp4" = add i32 %"%tmp", %"%tmp3"
  ret i32 %"%tmp4"
}

define i32 @mf() {
entry:
  %a = alloca i32
  store i32 3, i32* %a
  %a1 = load i32, i32* %a
  %sf_result = call i32 @sf(i32 9, i32 %a1)
  ret i32 %sf_result
}

define i32 @main() {
entry:
  %l = call i32 @mf()
  %l1 = alloca i32
  store i32 %l, i32* %l1
  %"%tmp" = load i32, i32* %l1
  %print = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([3 x i8], [3 x i8]* @fmt, i32 0, i32
0), i32 %"%tmp")
  %g = alloca i1
  store i1 true, i1* %g
  ret i32 0
}
```

```
fun mf = () -> int{

int a = 3;
fun sf = (x: int) -> int{



return x + a;
};


return sf(9);
};


int l = mf();
```

SCOPE

Twister is statically scoped

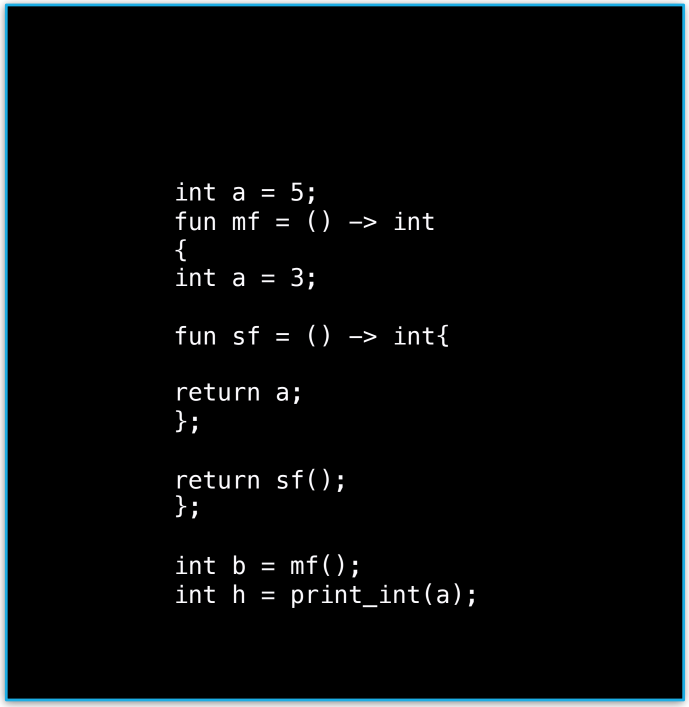
The result of printing this function is 3, not 5



```
int a = 5;
fun mf = () -> int
{
int a = 3;

fun sf = () -> int{

return a;
};

return sf();
};

int b = mf();
int h = print_int(b);
```

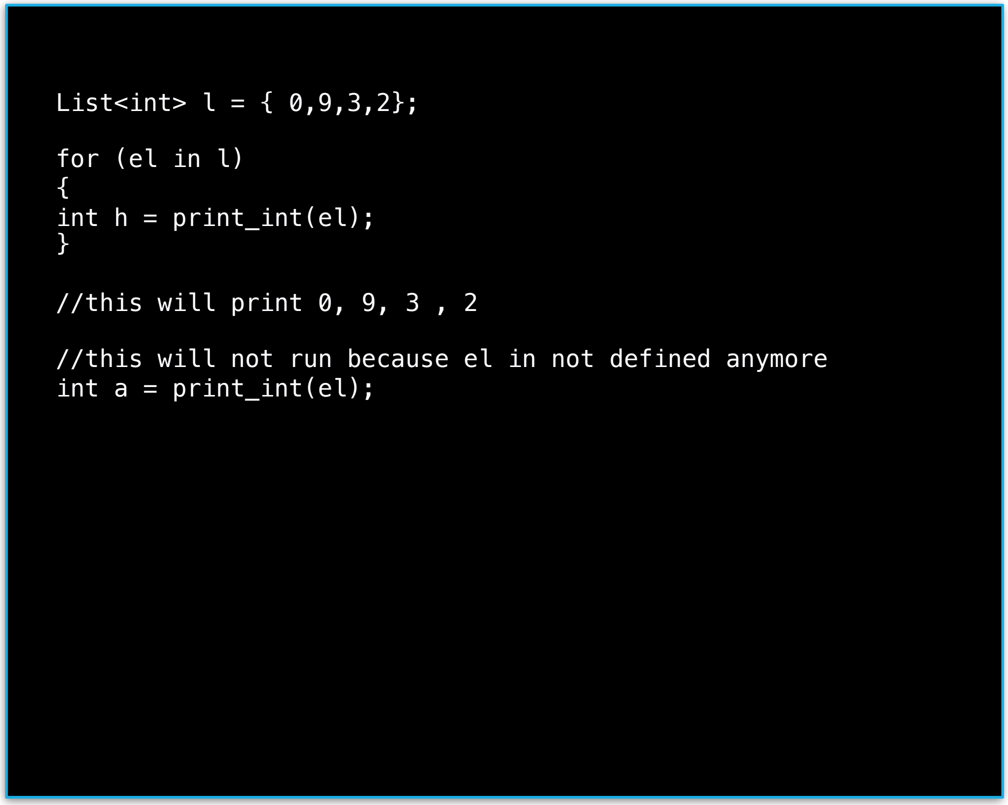
SCOPE

On the other hand, this will print 5



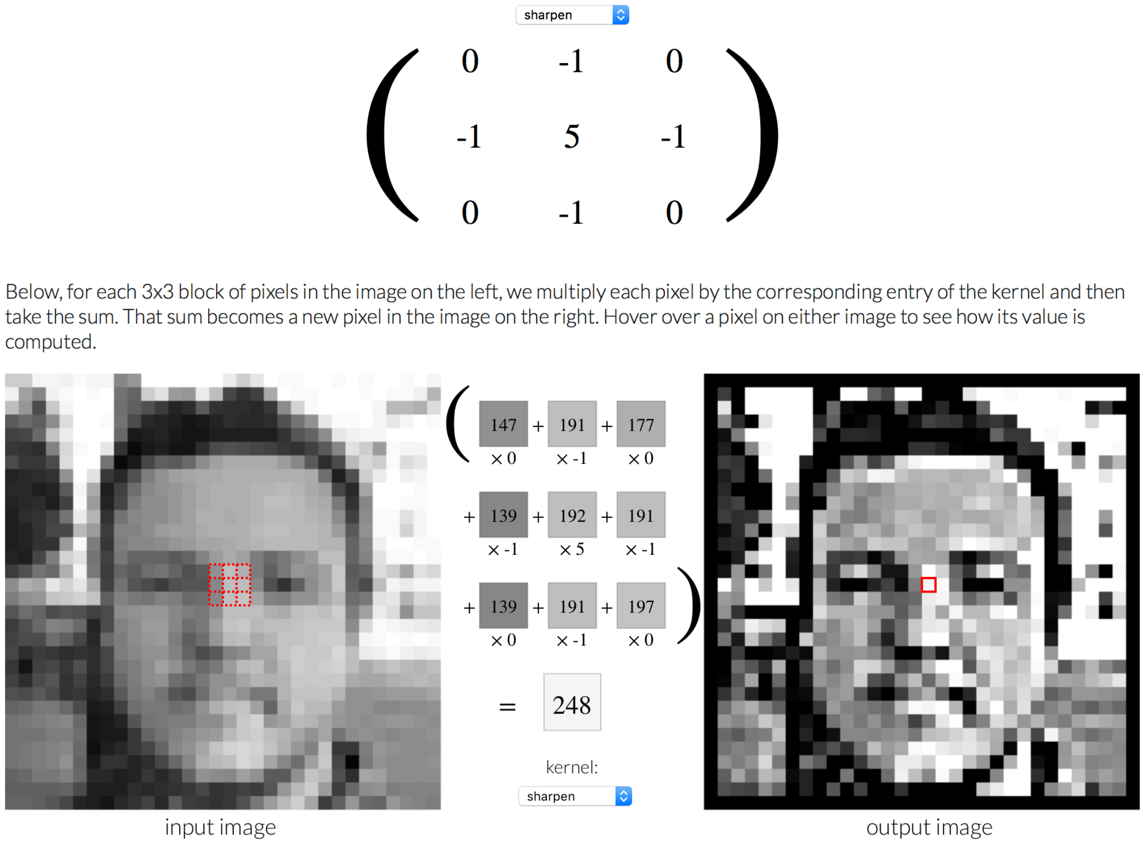
```
int a = 5;
fun mf = () -> int
{
int a = 3;

fun sf = () -> int{

return a;
};

return sf();
};

int b = mf();
int h = print_int(a);
```

FOR LIST SCOPE



```
List<int> l = { 0,9,3,2};

for (el in l)
{
int h = print_int(el);
}

//this will print 0, 9, 3 , 2

//this will not run because el in not defined anymore
int a = print_int(el);
```

CONVOLUTION



sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

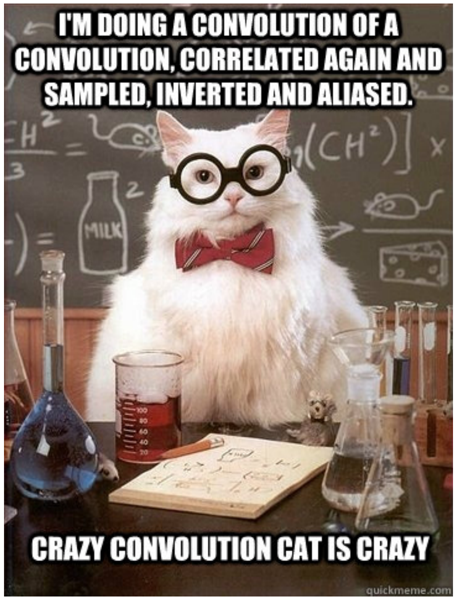
Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.

$$\begin{pmatrix} 147 \times 0 + 191 \times -1 + 177 \times 0 \\ + 139 \times -1 + 192 \times 5 + 191 \times -1 \\ + 139 \times 0 + 191 \times -1 + 197 \times 0 \end{pmatrix} = 248$$

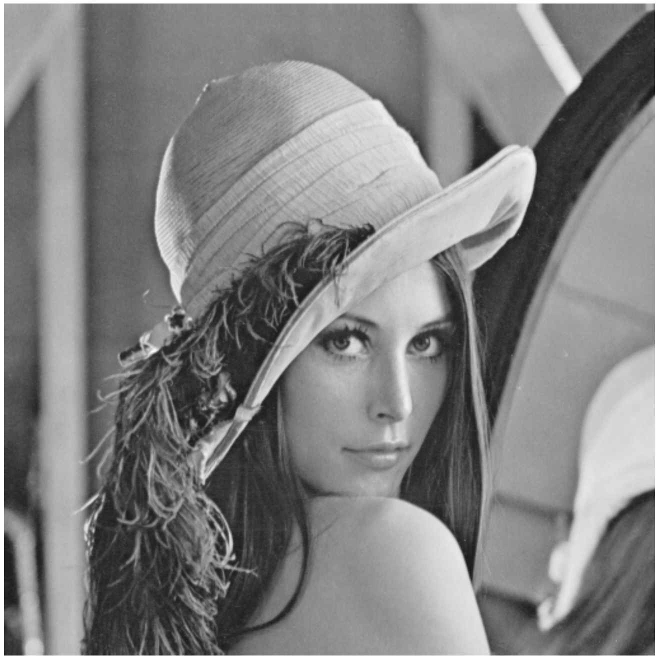
kernel: sharpen

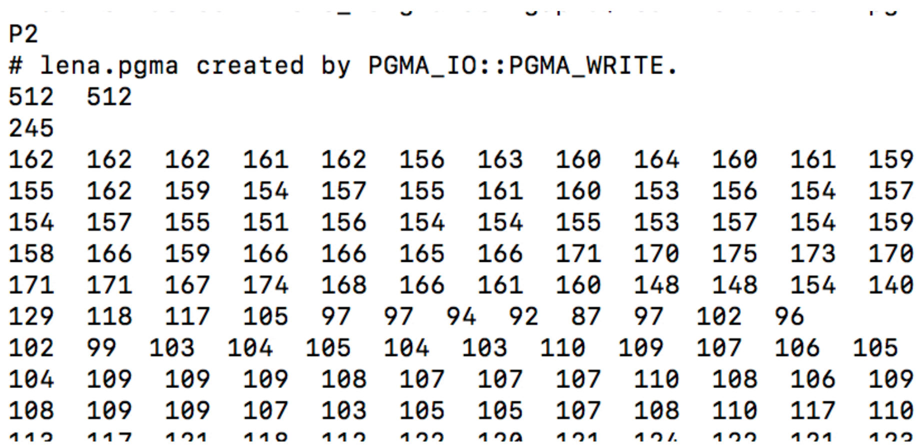
input image

output image


I'M DOING A CONVOLUTION OF A CONVOLUTION, CORRELATED AGAIN AND SAMPLED, INVERTED AND ALIASED.
CRAZY CONVOLUTION CAT IS CRAZY
quickmeme.com

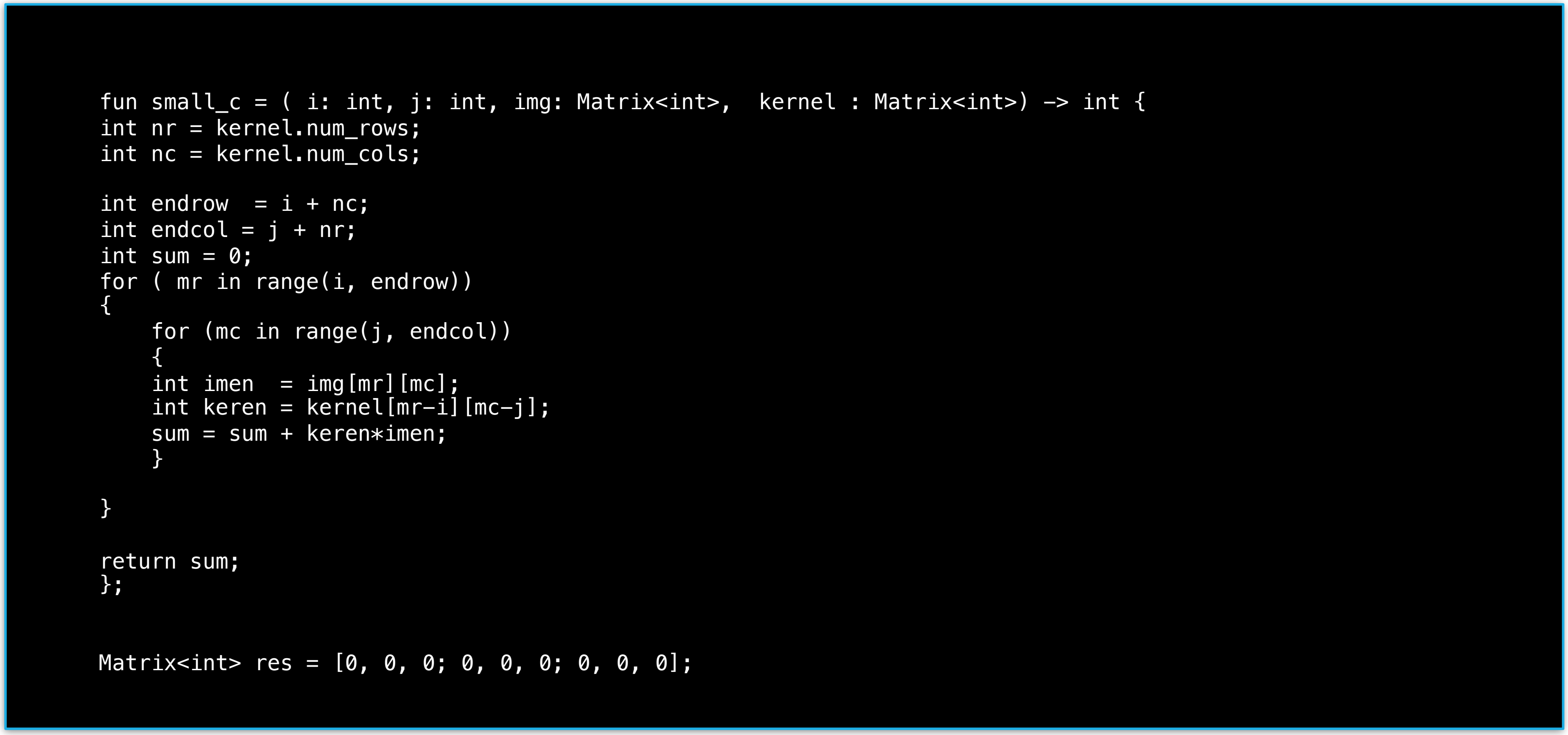
THE PGM FILE FORMAT





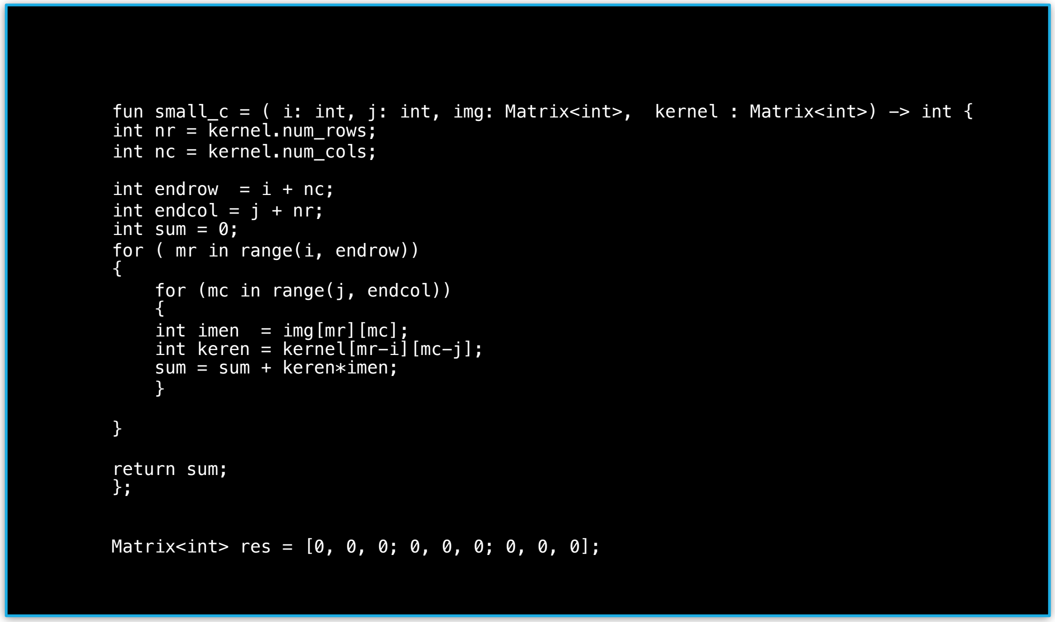
```
P2
# lena.pgma created by PGMA_IO::PGMA_WRITE.
512  512
245
162  162  162  161  162  156  163  160  164  160  161  159
155  162  159  154  157  155  161  160  153  156  154  157
154  157  155  151  156  154  154  155  153  157  154  159
158  166  159  166  166  165  166  171  170  175  173  170
171  171  167  174  168  166  161  160  148  148  154  140
129  118  117  105  97   97   94   92   87   97   102  96
102  99   103  104  105  104  103  110  109  107  106  105
104  109  109  109  108  107  107  107  110  108  106  109
108  109  109  107  103  105  105  107  108  110  117  110
```

EXAMPLE PROGRAM: CONVOLUTION



```
fun small_c = ( i: int, j: int, img: Matrix<int>,  kernel : Matrix<int>) -> int {
int nr = kernel.num_rows;
int nc = kernel.num_cols;

int endrow  = i + nc;
int endcol = j + nr;
int sum = 0;
for ( mr in range(i, endrow))
{
    for (mc in range(j, endcol))
    {
    int imen  = img[mr][mc];
    int keren = kernel[mr-i][mc-j];
    sum = sum + keren*imen;
    }

}

return sum;
};


Matrix<int> res = [0, 0, 0; 0, 0, 0; 0, 0, 0];
```

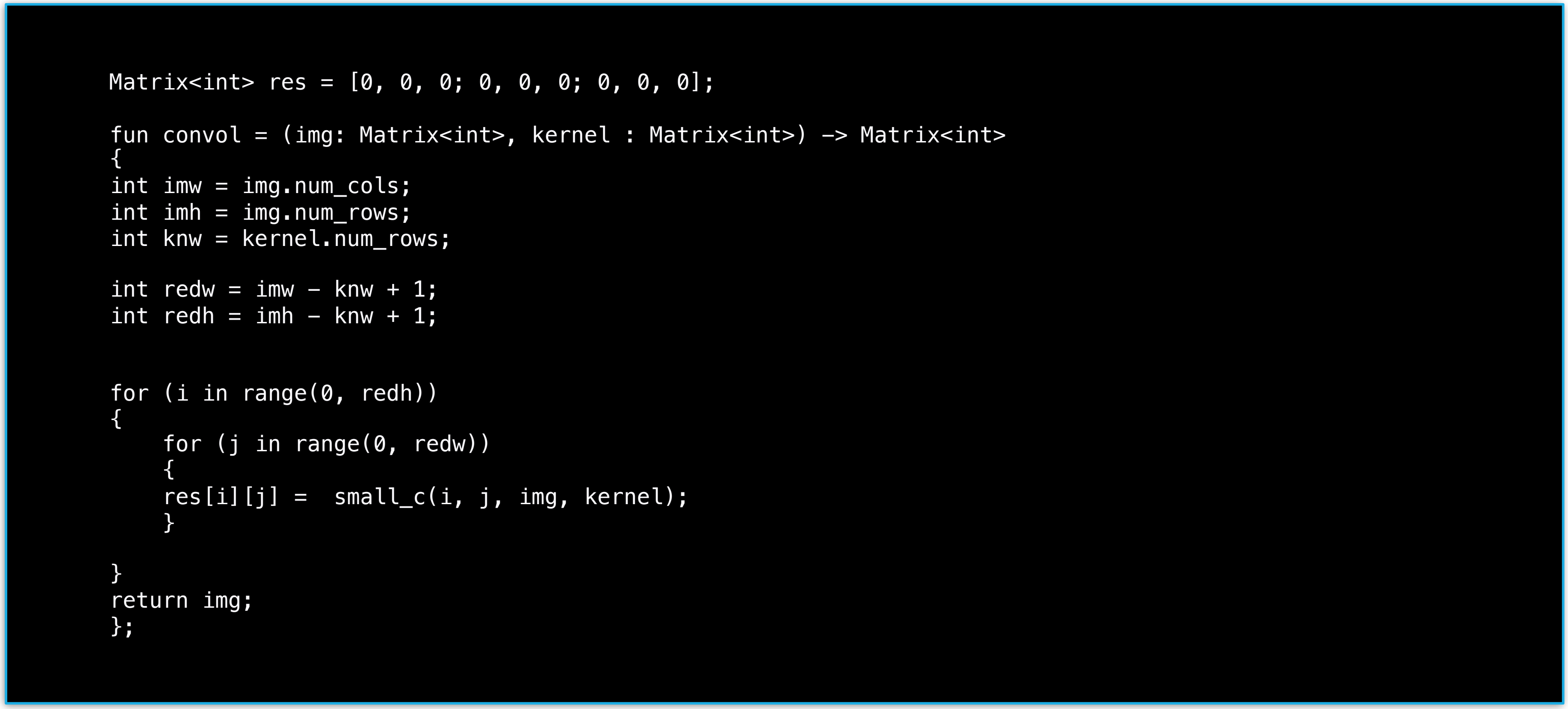
EXAMPLE PROGRAM: CONVOLUTION



```
fun small_c = ( i: int, j: int, img: Matrix<int>,  kernel : Matrix<int>) -> int {
int nr = kernel.num_rows;
int nc = kernel.num_cols;

int endrow  = i + nc;
int endcol = j + nr;
int sum = 0;
for ( mr in range(i, endrow))
{
    for (mc in range(j, endcol))
    {
    int imen  = img[mr][mc];
    int keren = kernel[mr-i][mc-j];
    sum = sum + keren*imen;
    }

}

return sum;
};


Matrix<int> res = [0, 0, 0; 0, 0, 0; 0, 0, 0];
```

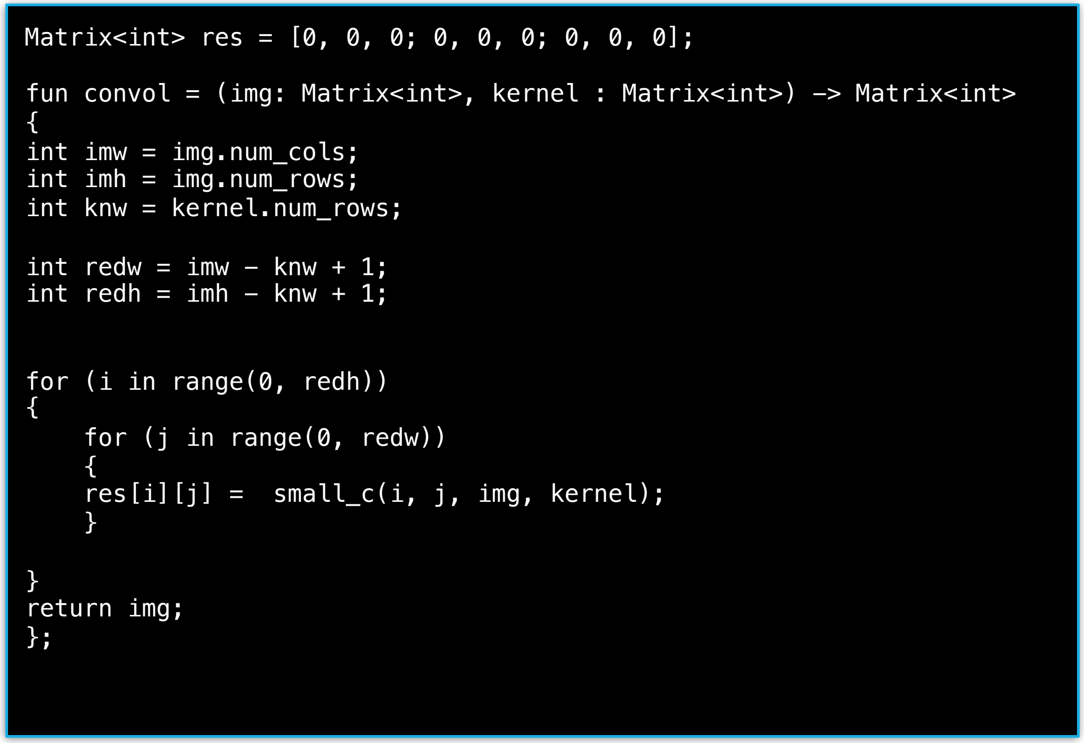
- computes a single pixel of our output image
- range(i, j) syntax
- variables may be defined before or after functions
- easily interchange between being inside functions & outside of them

EXAMPLE PROGRAM: MATRIX ROW SUM OUTPUT



```
Matrix<int> res = [0, 0, 0; 0, 0, 0; 0, 0, 0];

fun convol = (img: Matrix<int>, kernel : Matrix<int>) -> Matrix<int>
{
int imw = img.num_cols;
int imh = img.num_rows;
int knw = kernel.num_rows;

int redw = imw - knw + 1;
int redh = imh - knw + 1;


for (i in range(0, redh))
{
    for (j in range(0, redw))
    {
    res[i][j] =  small_c(i, j, img, kernel);
    }

}
return img;
};
```

CONVOLUTION CTD



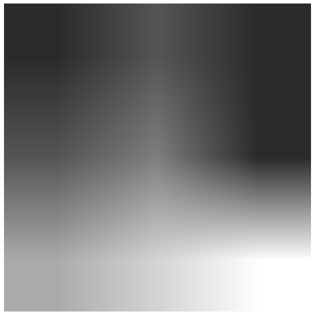
```
Matrix<int> res = [0, 0, 0; 0, 0, 0; 0, 0, 0];

fun convol = (img: Matrix<int>, kernel : Matrix<int>) -> Matrix<int>
{
int imw = img.num_cols;
int imh = img.num_rows;
int knw = kernel.num_rows;

int redw = imw – knw + 1;
int redh = imh – knw + 1;


for (i in range(0, redh))
{
    for (j in range(0, redw))
    {
    res[i][j] =  small_c(i, j, img, kernel);
    }

}
return img;
};
```

- functions see variables above their scope
- functions do not see variables below their scope
- Can return original image as matrix from function

EXAMPLE

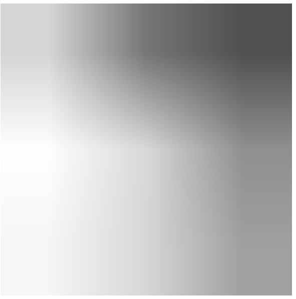
Original Image

```
18 2 0 1 5
3 1 2 1 4
7 9 3 1 2
9 5 6 7 1
0 8 6 2 3
```



Kernel

```
1 2 1
2 3 1
4 5 6
```



After Convolution

```
124 69 47
148 126 84
143 125 93
```

RGB IMAGES AND EXTRACTING COLORS

Matrices store tuples for RGB images

(type represents type of contents of tuple)

Small example:

```
Matrix<int> a= [(1,2,4);(2,4,1)];

Tup<int> ftup = a[0][0];
int h = println_int(ftup[1]);

will print 2.
```

TEST AND DEBUG

Test Procedures for scanner/ parser/ semant/ codegen:

clean.sh

build.sh

test.sh

Automating Tests for every build:

.travis.yml .travis-ci.sh

THANK YOU!