blur

Melissa Kaufman-Gomez | Team Leader
Timothy Goodwin | System Architect
Dexter Callender | Language Guru
Daniel Hong | Test Wizard

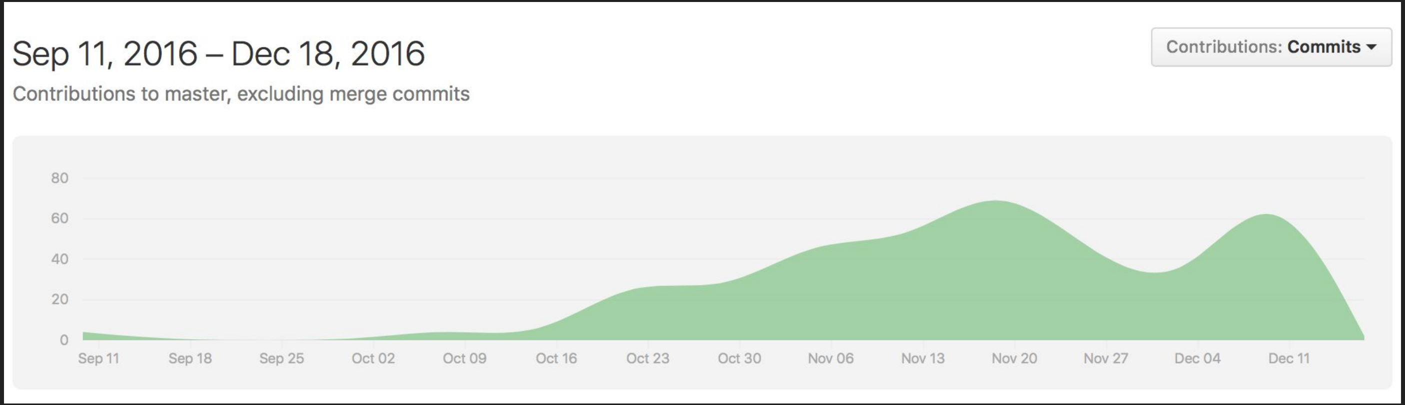
blur : -*noun*. a programming language for creating and modifying ASCII art.

blur provides the programmer with:

- Familiar syntactic conventions
- an intuitive set of built-in functions and operators catered to ASCII art creation
- a standard library of prebuilt image manipulation functions

Project Management Timeline

- 3 in-person group meetings per week
- Continuous communication via Slack
- Conflict resolution with TA during office hours



Basic Syntax

Primitive Types:

```
bool, char, int, double, string, void
```

Operators:

```
+ - * / % < > == != <= >= = and or !
||
```

Loops:

```
int i;
int j;
for( i=0; i<10; i=i+1 ){
    for( j=0; j<5; j=j+1 ){
        print(“hey”);
    }
}
```

Conditionals:

```
bool blur = true;
if( blur ){
    println(“ bluriffic “);
} else {
    println(“ sad day. “);
}
```

Syntax Basics: Arrays

- Stack or heap based, depending on declaration
- Sized Array Type
  - Declared as space on stack
  - Short-term data storage / handling
  - Can only read/assign its elements
- Unsized Array Types
  - are immediately assigned
  - Stored on heap, handled by reference
  - can pass them and return them.

<u>Sized Array Type</u>

```
int[50][50] grid;

int[50] col = grid[0];
```

<u>Unsized Array Type</u>

```
char[][] cv = canvas(“fire.jpg”);

double[] gpa = [4.0, 3.87, 3.64, 3.52];
```

<u>Built-in length function:</u>

```
int[][] img = myPixLoader(“earth.jpg”);

int width = len(img); /* 400 */

int height = len(img[0]); /* 640 */
```

Supports more complex functions

Recursion

```
int recurse( int a ){
     if( a == 0 ){
          Return 0;
     }
     println( a );
     recurse( a - 1 );
}
```

Sorting

```
int[] insertionSort(int[] arr) {
    int i;
    int key;
    int k;
    for(i = 1; i < len(arr); i = i + 1) {
        key = arr[i];
        k = i - 1;
        while(k >= 0 and arr[k] > key) {
            arr[k + 1] = arr[k];
            k = k - 1;
        }
        arr[k + 1] = key;
    }
     return arr;
}
```

Built-in Functions

Implemented in blur

- `char intensityToChar(int intensity);`

  receives an intensity value as an integer from 0-255 and returns a character that is approximately mapped to that intensity value. Handles out of bounds w/ modular arith, abs value.

- `int charToIntensity(char c);`

  Receives a character and returns an integer intensity value (0 - 255).

Implemented in c

- `char[][] readGrayscaleImage(string filename);`

  Reads an image file using OPENGL and returns a struct containing a pointer to the heap allocated image and its dimensions

- `Char[][] canvas(string filename);`

  Reads an image file using OPENGL and returns a struct containing a pointer to the heap allocated image and its dimensions

The Magnitude Operator

- Provides a clean interface to the `intensityToChar()` and `charToIntensity()` functions.
- Convert between pixel intensities and ASCII characters
- Consults the Blur ASCII grayscale ramp in the builtin library.

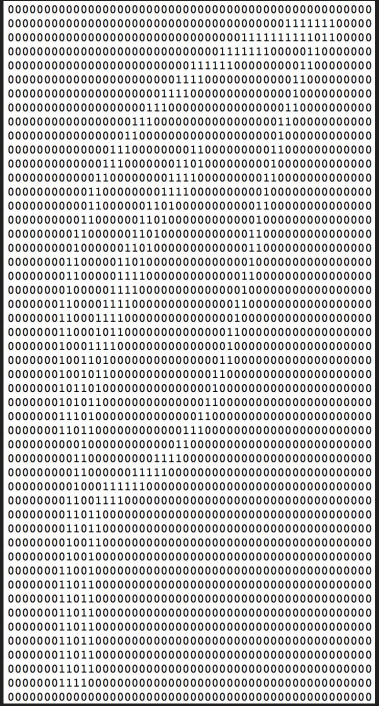
```
int[][] a = readGrayscaleImage(“sun.jpg”);
for(i = 0; i < width; i = i + 1) {
     for(j = 0; j < height; j = j + 1) {
          px = | a[i][j] |;
          canvas[i][j] = px;
     }
}
return canvas;
```

Usage:

```
int pxd = |’$’|; /* 255 */

print(|’@’| > |’.’|); /* true */

char pxl = |2|; /* ‘,’ */

char c = ‘%’;

char darker = ||c| + 1|; /* ‘%’ -> ‘$’ */
```

The Blur Standard Library 1.0

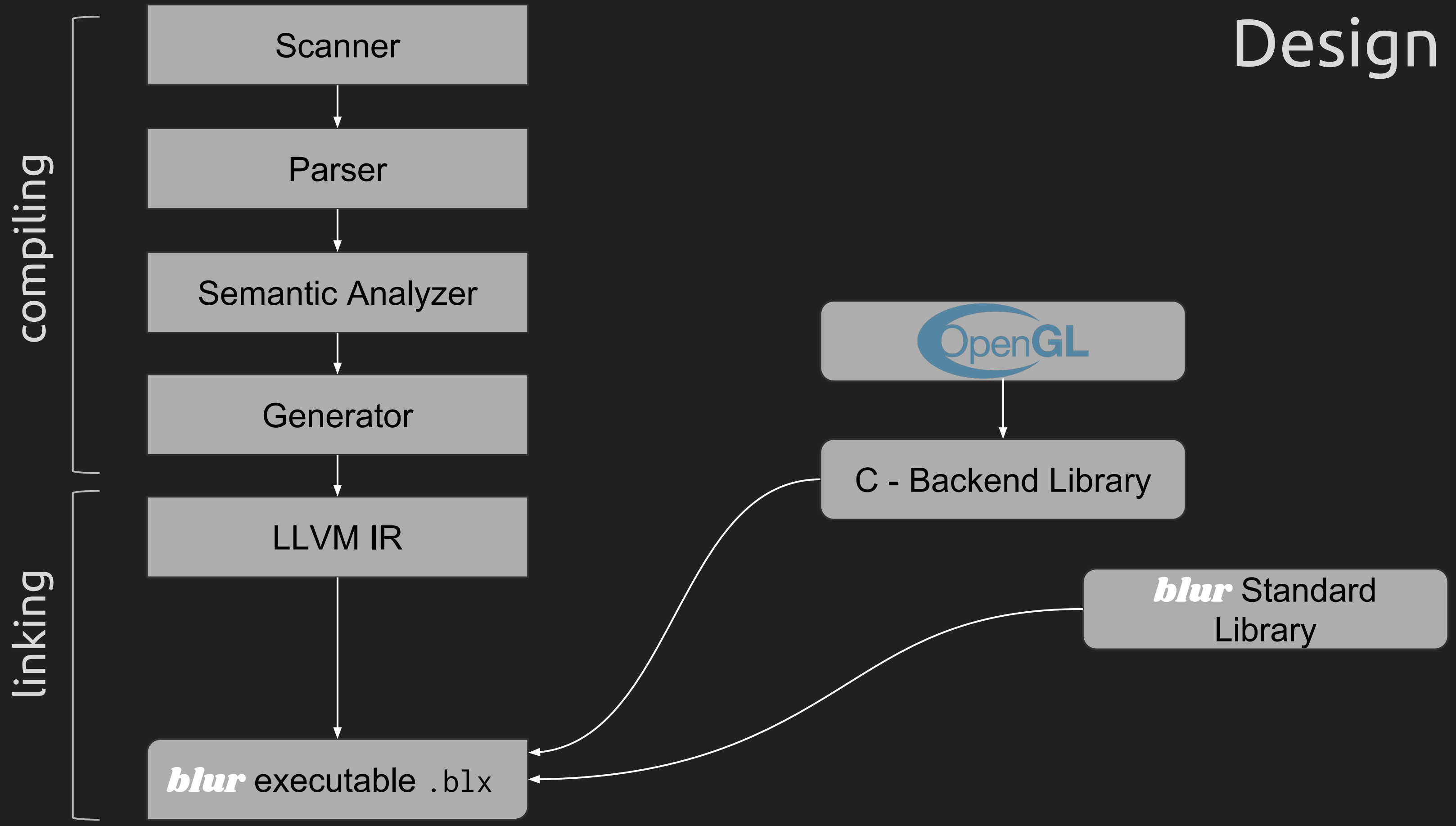
- `char[][] dither(string fileName);`
- `int[][] edgeDetect(string fname, int threshold);`
  - Identifies edge pixels within an image based on an intensity threshold
- `char[][] impose(char[][] c1, int[][] mask, char marker);`
  - Draws a 2D bitmask onto an ASCII image.
  - e.g. for illustrating edge detection results.



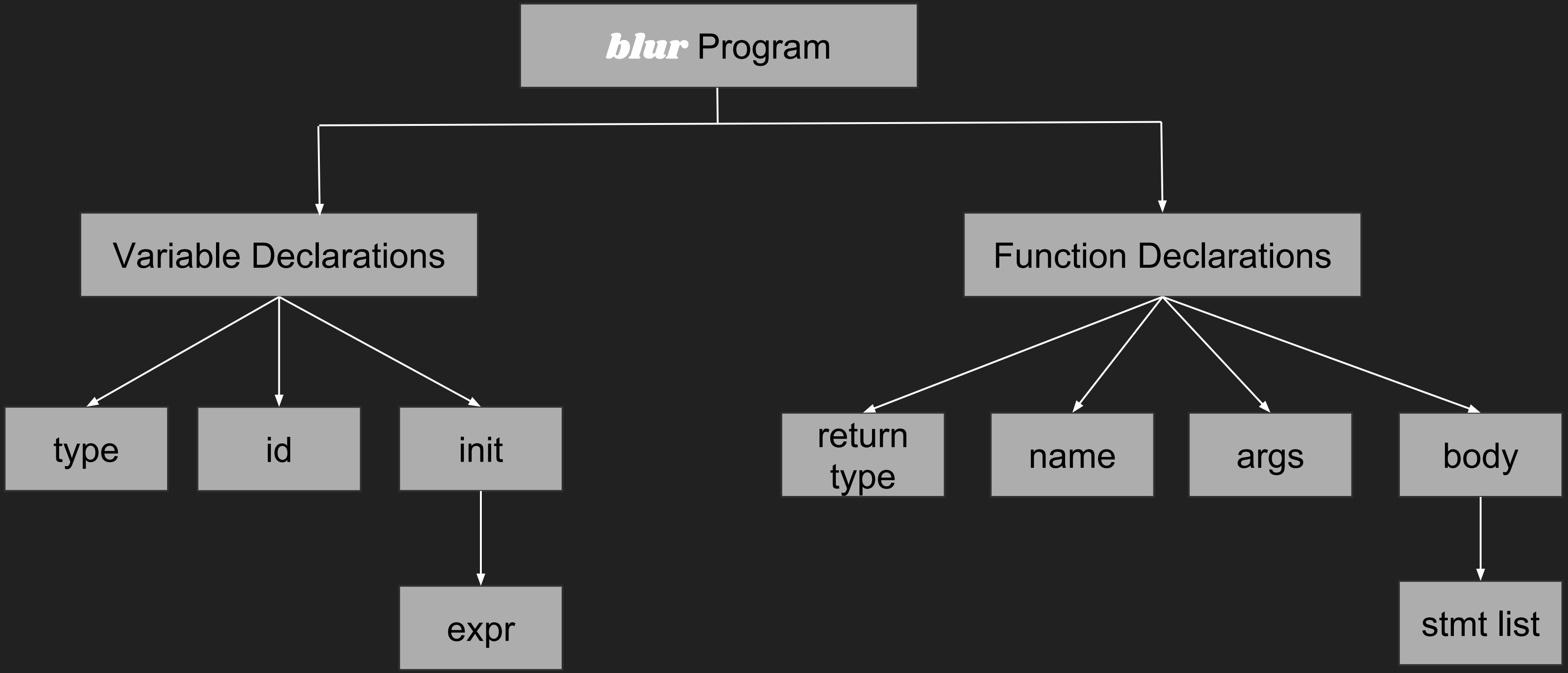
```
00000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000111111100000
00000000000000000000000000000000111111111011000000
00000000000000000000000000000111111100000110000000
00000000000000000000000001111110000000011000000000
00000000000000000000000111100000000000011000000000
00000000000000000000011110000000000000010000000000
00000000000000000001110000000000000000110000000000
00000000000000000111000000000000000001100000000000
00000000000000001100000000000000000001000000000000
00000000000000111000000001100000000011000000000000
00000000000001110000000110100000000010000000000000
00000000000011000000001111000000000110000000000000
00000000000110000000011110000000000100000000000000
00000000000110000001101000000000001100000000000000
00000000001100000011010000000000001000000000000000
00000000011000000110100000000000011000000000000000
00000000010000001101000000000000011000000000000000
00000000110000011010000000000000010000000000000000
00000000110000111100000000000000110000000000000000
00000000100000111100000000000000100000000000000000
00000001100001111000000000000001100000000000000000
00000001100011110000000000000001000000000000000000
00000001100010110000000000000011000000000000000000
00000001000111000000000000000010000000000000000000
00000001001101000000000000000110000000000000000000
00000001001011000000000000001100000000000000000000
00000001011010000000000000001000000000000000000000
00000001010110000000000000011000000000000000000000
00000001110100000000000000110000000000000000000000
00000001101100000000000011100000000000000000000000
00000000001000000000000110000000000000000000000000
00000000011000000000111100000000000000000000000000
00000000011000000111110000000000000000000000000000
00000000010001111110000000000000000000000000000000
00000000110011100000000000000000000000000000000000
00000000110110000000000000000000000000000000000000
00000000110110000000000000000000000000000000000000
00000000100110000000000000000000000000000000000000
00000000100100000000000000000000000000000000000000
00000001100100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001101100000000000000000000000000000000000000
00000001111000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000
```

Under-the-hood Implementation

Design


compiling
linking
Scanner
Parser
Semantic Analyzer
Generator
LLVM IR
blur executable .blx
OpenGL
C - Backend Library
blur Standard Library

AST



blur Program
Variable Declarations
Function Declarations
type
id
init
expr
return type
name
args
body
stmt list

Technical Considerations

A Backend C Library, Communicating with Blur Programs

C-backend returns structs containing image dimensions and a pointer to heap-allocated image data.

C-backend represents image data as 1-D array (the OPENGL representation). Blur presents this data as 2-D array to the user.

This allows for more intuitive dimension retrieval and iteration.

```
// in C

struct ImageStruct{
    int width;
    int height;
    int depth;
    int *imageData;
};

struct ImageStruct readGrayscaleImage( char *
filename );

/* in Blur */

int[][] image = readGrayscaleImage(“x.jpg”);
int width = len(image);
int height = len(image[0]);
```

Supporting Dynamically Sized Arrays

Blur supports arrays with size determined at runtime (e.g. dimensions of an image).

Challenge: cannot store any array dimension info at compile time.

Solution: use our C `ImageStruct` approach for the implementation of all Unsized Arrays in the LLVM IR.

```
; in Blur LLVM Module

declare {i32; i32; i32; i8* }
@canvas(i8*, … )
```

Structuring Semantic Analysis

Originally, we implemented semantic analysis using lists and maps as microc does.

We then realized that for blur, using an environment, which we update as we read through the program, would allow greater flexibility.

For example, variables can be declared throughout the program, not only at the top of a function.

```
type symbol_table = {
   parent: symbol_table option;
   args: argdecl list;
   variables: vardecl list
}

type func_entry = {
   name: string;
   arg_types: datatype list;
   return_type: datatype
}

type env = {
   symtab: symbol_table;
   funcs: func_entry list;
   return_type: datatype option
}
```

Standard Library versus Built-in Functions

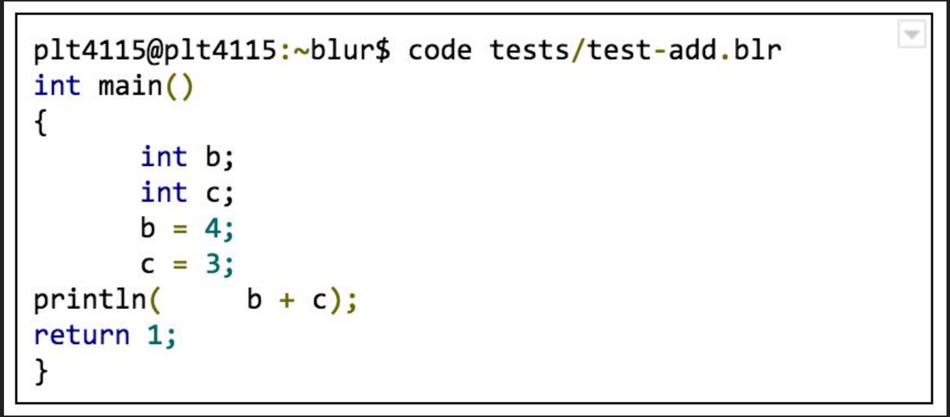
```
/* Using Standard Library Functions */


/* Dither */
char[][] dither(string filename);

/* Edge Detect */
int[][] edgeDetect(string filename, int
threshold);

/* Impose */
char[][] impose(char[][] asciiArt,
int[][] edges );
```

```
/* Using Builtin Functions */

int[][] x =
readGrayscaleImage("file.jpg");

char[][] customDither(int[][] a){
    int width = len(a);
    int height = len(a[0]);
    int i;
    int j;
    for( i=0; i < width; i++){
        for( j=0; j < height; j++){
          … IntensityToChar('a');
     }
    }
    return a;
}
```

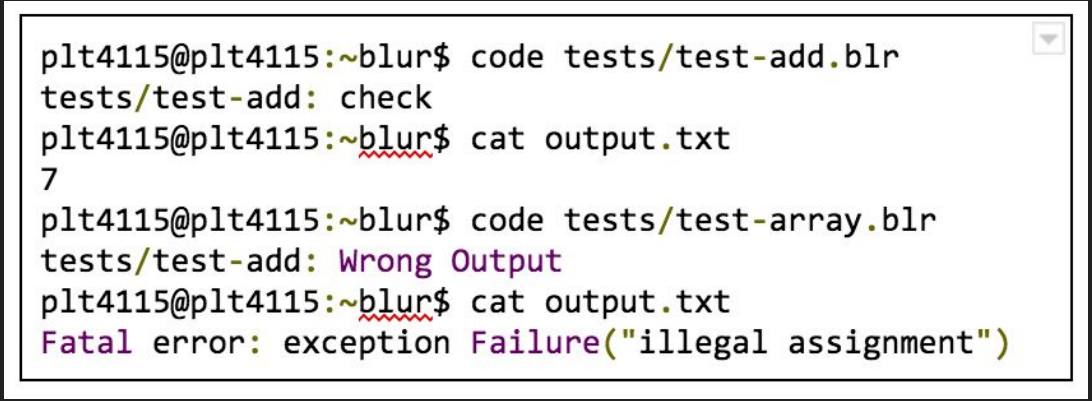
Test Suite - PP & Semantic & Generator & Output

Compile & Run

- This intermediate version of compile + run test script does not include libraries



```
plt4115@plt4115:~blur$ code tests/test-add.blr
int main()
{
        int b;
        int c;
        b = 4;
        c = 3;
println(     b + c);
return 1;
}
```



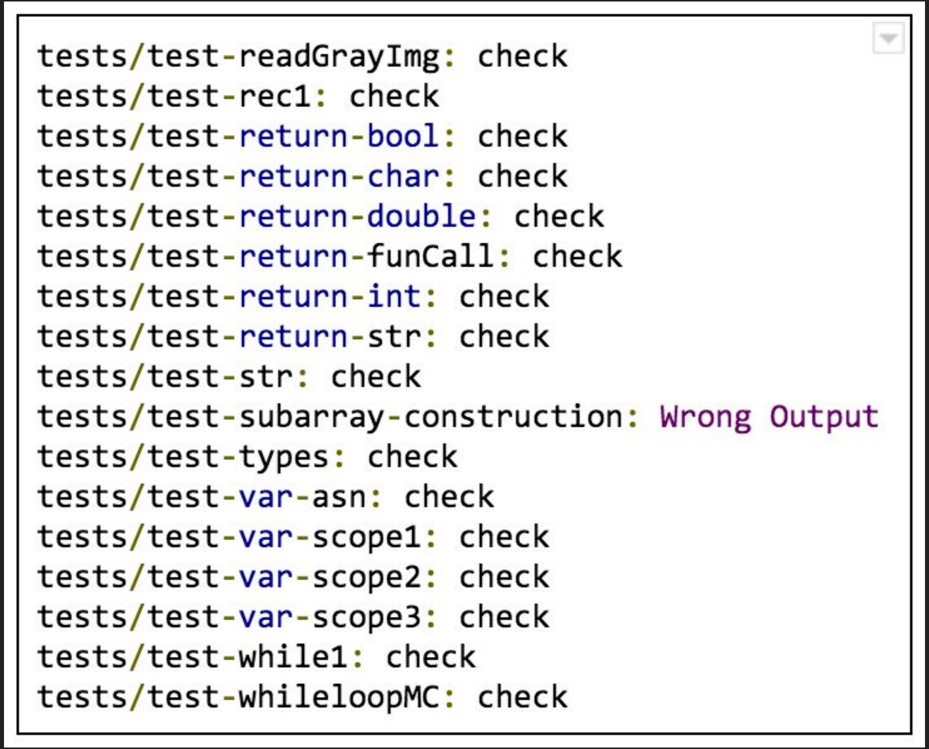
```
plt4115@plt4115:~blur$ code tests/test-add.blr
tests/test-add: check
plt4115@plt4115:~blur$ cat output.txt
7
plt4115@plt4115:~blur$ code tests/test-array.blr
tests/test-add: Wrong Output
plt4115@plt4115:~blur$ cat output.txt
Fatal error: exception Failure("illegal assignment")
```

Test Suite - Suite Automation

Automated Testing

- This final version of test script includes built-in and standard libraries.

  Automated run of the entire test suite is supported.



```
tests/test-readGrayImg: check
tests/test-rec1: check
tests/test-return-bool: check
tests/test-return-char: check
tests/test-return-double: check
tests/test-return-funCall: check
tests/test-return-int: check
tests/test-return-str: check
tests/test-str: check
tests/test-subarray-construction: Wrong Output
tests/test-types: check
tests/test-var-asn: check
tests/test-var-scope1: check
tests/test-var-scope2: check
tests/test-var-scope3: check
tests/test-while1: check
tests/test-whileloopMC: check
```

DEMO